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. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

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.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

Université d'Ottawa · University of Ottawa

An Architectural Model For Mobile Agent-Based Applications

by

Bruce Ford, B.A.Sc.

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 Engineering
Department of Electrical Engineering
Faculty of Engineering
University of Ottawa
Ottawa, Ontario, Canada

1 December 1997

Abstract

Recently, several important trends have emerged in the context of the rapidly expanding Internet, including increased consumption of network bandwidth as multimedia applications become more heavily used, strong consumer pressure for more powerful tools for efficiently navigating through the immense volumes of information available on the Internet and user demand for services that are customized to the user's individual needs.

Mobile agents possess many of the attributes required to address these trends. Mobile agents can be generally defined as software entities that migrate through a network and perform processing functions on a client's behalf at remote sites. Mobile agents contain executable software scripts that are interpreted by each network site that is visited. By migrating the processing functions that are traditionally performed at the client site to a remote server, mobile agents represent a new paradigm for network communications.

This thesis describes a mobile agent architectural model that is capable of efficiently supporting multimedia applications. The architectural model describes an internal system architecture based on a set of re-useable architectural building blocks. Another key facet of the architectural model is the specification of an application-independent structure for mobile agents and the integration of this structure into an application layer protocol. The application protocol, referred to as Mobile Agent Protocol (MAP), is specifically designed to enable mobile agents to travel through the network in a secure and efficient manner; the protocol is based on the TCP/IP protocol stack and is specified using Abstract Syntax Notation One (ASN.1).

Acknowledgements

I would like to express my appreciation to my thesis supervisor, Dr. Ahmed Karmouch for his guidance and support throughout the program. I would also like to thank my wife, Nina, for all of the encouragement received throughout the M.Sc. program, especially during these past few months. And finally, I would like to acknowledge the support received from the Public Carrier Networks (PCN) System Engineering (SE) management team at Nortel, namely Patrick Hampson and Roch Charbonneau.

Table of Contents

List of Figures

List of Abbreviations

AEE	Agent Execution Environment
AI	Artificial Intelligence
ASN.1	Abstract Syntax Notation One
BER	Basic Encoding Rules
GUI	Graphical User Interface
FTP	File Transfer Protocol
HTTP	HyperText Transfer Protocol
ICMP	Internet Control Message Protocol
IP	Internet Protocol
JVM	Java Virtual Machine
KQML	Knowledge Query and Manipulation Language
MAP	Mobile Agent Protocol
MIRL	Multimedia Information Research Laboratory
NMH	Network Message Handler
RPC	Remote Procedure Call
SNMP	Simple Network Management Protocol
Tcl/Tk	Tool Command Language / ToolKit
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WWW	World Wide Web

Chapter 1

1 Introduction

1.1 Motivation

Although the concept of mobile agents was pioneered before the recent explosion in popularity of the Internet, much of the current interest in mobile agents is directly related to their potential to dramatically evolve the way the Internet is used. Furthermore, a number of significant trends on the Internet can be directly addressed by mobile agent technology. These trends translate into requirements for improved effectiveness of network resource utilization, increased user efficiency and customization of services.

Currently, text and image are the predominant forms of information exchanged over the Internet. However, there are clear signs that the situation is rapidly changing with the recent emergence of multimedia applications, resulting in increased consumption of network bandwidth [1]. This trend has obvious implications on the backbone network and will also affect the technology solutions used to access the Internet. Therefore, mobile agent technology must support mechanisms for efficiently handling multimedia data.

As the amount of information on the Internet continues to grow at a rapid pace, it is becoming more of a challenge for users to make efficient use of the Internet. The world wide web (www) represents a very large and dynamic database, with no centralized administration and no guarantee that information available on one day will be available on the next day. Web browsers currently provide users with a mechanism for navigating this dynamic environment; however, as browsers require direct user interaction, it is often a very time consuming task for

users to retrieve information from the web. Mobile agents offer a better alternative to web browsing, as they can retrieve information from the web without requiring direct user interaction.

Yet another important trend relates to customized services. Consumers are looking for service providers who can provide solutions that are customized to their individual needs. Furthermore, as the global economy becomes increasingly competitive, service providers understand the importance of personalized service for retaining a loyal customer base. As mobile agents typically contain user preference information, mobile agent technology is well-suited for enabling the delivery of services that are customized to the end user's needs.

1.2 Objectives

The main objective of this thesis is to develop an architectural model for mobile agents that is designed for use in a multimedia environment. Specifically, emphasis will be placed on developing a mobile agent system architecture that includes the specification of an application-independent structure for representing mobile agents and a Mobile Agent Protocol (MAP) for communicating over a TCP/IP network.

Upon completion of the system design, a prototype system will be developed within the University of Ottawa's Multimedia Information Research Laboratory (MIRL) to demonstrate the validity of the mobile agent architectural model.

1.3 Thesis Outline

The rest of the thesis is structured according to the following outline. Chapter 2 introduces the topic of mobile agents, by providing definitions and by contrasting the mobile agent communications model with traditional communications models. Chapter 2 also provides insight into recent research and development activities related to the field of agents. Chapter 3

describes the network level architecture of the mobile agent framework and specifies the internal architecture of the two principal network elements, the User Site and the Network Site. In addition, a layered network communications model is described, upon which the design of the Mobile Agent Protocol (MAP) described in Chapter 5 is based.

Chapter 4 defines the internal structure of a mobile agent that uses a set of re-useable components to provide the flexibility to handle a range of applications. Chapter 5 specifies a TCP/IP-based protocol, referred to as Mobile Agent Protocol (MAP), that supports a number of mechanisms for providing a secure and efficient environment for the movement of mobile agents through the network. This protocol is formally described in Appendix A using the Abstract Syntax Notation One (ASN.1) specification language; examples of MAP protocol encodings based on the use of the Basic Encoding Rules (BER) transfer syntax are provided in Appendix B.

Chapter 6 describes the implementation of a Multimedia Document Retrieval application that was designed based on the mobile agent architecture developed in this thesis. Chapter 7 summarizes the conclusions of this thesis and provides suggestions for future research activities.

1.4 Main Contribution

The main contribution of this research is to design and implement an architectural model for mobile agents that is capable of efficiently supporting multimedia applications. The first aspect of this research is the development of the system architecture based on a set of fundamental requirements specified at the outset of this thesis. Special emphasis is placed on developing a set of re-useable architectural components, with each component representing a logical grouping of system functions. Furthermore, to provide an architectural framework for the development of the Mobile Agent Protocol (MAP), a layered network communications model is developed.

Another key facet of this thesis is the specification of a multimedia mobile agent structure suitable for use with a range of multimedia applications; this is accomplished by seeking to represent mobile agents as a collection of reusable components. An equally important focus of this research is the coupling of the mobile agent structure with an application layer protocol designed specifically for mobile agents.

The Mobile Agent Protocol (MAP) is developed on the TCP/IP protocol stack to provide compatibility with the Internet environment. The primary objective of the protocol is to enable the transfer of mobile agents through the network in a secure and efficient manner. The Mobile Agent Protocol (MAP) is specified using Abstract Syntax Notation One (ASN.1), a formal language developed by the International Telecommunications Union (ITU) for the specification of application layer protocols. Furthermore, the use of Basic Encoding Rules (BER) for encoding the MAP into a transfer syntax is explored.

The validity of the architectural design will be demonstrated through the development of a prototype system at the University of Ottawa's Multimedia Information Research Laboratory (MIRL). While the prototype will not be implemented directly on the Internet, most aspects of the design will be directly applicable to an Internet environment.

Chapter 2

2 The Current State of Agent Technology

2.1 Introduction

This chapter examines the current state of agent technology, which encompasses a broad range of research and development activities related to the field of agents. As the topic of this thesis is mobile agents, it is convenient to categorize agents according to whether they support mobility or not. Most of the emphasis in this chapter will be placed on mobile agents. However, as discussed in subsequent sections, there are many interesting and relevant research and development activities on the general topic of agents (i.e. including agents that do not support mobility), especially in the context of the Internet. Some of these developments will be briefly described.

The term 'agent' is a challenge to clearly define, owing to the number of definitions that are currently in use today. For example, the term 'agent' is often used to lend marketing appeal to traditional software products that would otherwise not be labeled as agents, to the general concern of members of the agent research community [2]. At one end of the scale are relatively simple, client-based software applications that can assist users in performing mundane tasks such as sorting e-mail or downloading web pages from the world wide web (www). This class of agent is often referred to as 'personal assistant' agents. At the other end of the scale is the concept of sophisticated software entities possessing Artificial Intelligence (AI) that autonomously travel through a network environment and make complex decisions on a user's behalf.

The mobile agent architectural model described in this thesis is based upon a definition of mobile agents that lies somewhere between these two extremes. Specifically, the mobile agents described in this thesis fully support the concept of mobility yet are somewhat limited in their ability to make complex decisions. This classification of mobile agents, although not strictly adhering to the definition of 'intelligent agents' originally proposed by the Artificial Intelligence (AI) community, is generally acknowledged to be a useful categorization [3] and has started to gain widespread acceptance.

2.2 Definition of Mobile Agents

Mobile agents can be generally defined as software entities that migrate through a network and perform processing functions on a client's behalf at remote sites. Other terms used to describe this category of agent include itinerant agents, transportable agents and autonomous agents. Mobile agents contain executable software scripts that are interpreted by each site that is visited. By migrating the processing functions that are traditionally performed at the client site to the remote server, mobile agents represent a new paradigm for network communications.

Mobile agents have been simply defined as "named program that can migrate from machine to machine in a heterogeneous environment" [4]. Furthermore, a mobile agent framework has been defined as an architecture that "permits the secure interaction of agents written in multiple scripting languages, communicating in various agent communication languages and travelling via multiple transport services in a heterogeneous environment" [5]. And finally, mobile agents have been defined as a technology that "enables the creation of a new breed of network that supports the development of communicating applications by making the network a platform for developers" [6].

The defining property of mobile agents is their ability to initiate travel from one site to another. Specifically, mobile agents must be able to suspend the execution of their script, capture their state variables, initiate a transfer to a new site and resume execution exactly at the point where the program execution was suspended. The ability to preserve program state is an essential requirement.

Another defining property that is common to all categories of agents is autonomy; namely the ability to perform a task on a client's behalf without requiring direct intervention by the user. In this way, agents are able to relieve a client of time intensive tasks such as making arrangements for meetings involving many participants.

Other important properties of a mobile agent system include platform independence and security. Platform independence refers to the ability to operate in a heterogeneous network (i.e. a network containing a variety of computing platforms) and is typically achieved through the use of interpreted programming languages such as Tcl or Java. Security is also an essential property of mobile agent system, to guarantee the privacy and integrity of transactions between clients and network servers.

2.3 Evolution of the Mobile Agent Concept

2.3.1 Fundamentals of Network Communication

In its simplest form, network communications involves the exchange of information between two computing devices connected to a network. Consider the case in which one device supports a client-side application (i.e. representing a user seeking information) and the other device supports a network server with a large database of information. There are many possible modes of communication between these two computing devices, with varying degrees of complexity.

The simplest form of communication is to transfer segments of raw data stored in the database to the client site, where the client can subsequently analyze the information. This approach is generally considered to be wasteful of network resources, as much of the raw data transferred to the user is typically not useful.

A more efficient approach is to implement a set of procedures at the server site that can be remotely 'called' by the client. In this way, the client requests a specific subset of information from the remote server by specifying a procedure name along with a list of data parameters. The information request is processed at the server by a software program that implements the called procedure. This model offers improved network efficiency over the previous model and is referred to as Remote Procedure Call (RPC). It is also referred to as the client-server model.

The most advanced model of communications involves transferring a software program contained the required procedures to the remote site. It is this model that provides the underlying communication foundation for mobile agents and is referred to as Remote Programming (RP) [6].

The concept of remote execution of programs and platform independent scripts was first introduced by the PostScript language, which enables script-based procedures to be sent from a user's computing device to a remote printer device. Java and Safe-Tcl provide more recent examples of this model in which procedures are transported without state information to a remote site for execution from start to finish. Safe-Tcl was developed as an extension to J. Ousterhout's Tcl scripting language [7] to permit Tcl scripts to be transported via e-mail to a remote site for execution. And finally, the most sophisticated variation of this model is General Magic's Telescript language, in which a partially executed procedure (i.e. with state information) is

transported to remote server for further execution. Telescript will be described in more detail later in this chapter.

2.3.2 Benefits of the Mobile Agent Communication Model

There are a number of benefits of the mobile agent communication model over the traditional communications models. By transferring the executable program (i.e. mobile agent) to the source of the data (i.e. network site), network efficiencies can be obtained; all information processing is performed at the remote site, thereby minimizing the amount of extraneous raw data returned to the client. In particular, this mode of communication derives its efficiency from the assumption that the size of the mobile agent is significantly smaller that the volume of information that would otherwise need to be transferred across the network from the network site to the user's site. However, in cases where this assumption does not hold true (i.e. in cases where the transfer of program code is not warranted), the traditional method of remote message passing is expected to provide a more effective mode of communication. For this reason, remote message passing still has an important role to play within mobile agent systems.

Secondly, by permitting users to transfer programs to remote sites for execution, new applications can be more easily introduced; for example, customized applications can be implemented without requiring any changes to the application software at the network site. In this way, mobile agents provide the flexibility to overcome constraints of the traditional Remote Procedure Call (RPC) model that is dependent upon the use of a pre-defined set of procedures.

Finally, mobile agents do not require the establishment of a connection between the user site and the network server, thereby allowing mobile agents to perform task on a user's behalf even when the user is disconnected from the network.

2.4 Mobile Agent Applications

Mobile software agents have been proposed for a variety of non-real time applications, including information retrieval, event monitoring, network management, portable computing, collaboration, electronic commerce and workflow.

For information retrieval applications, mobile agents can relieve users of the burden of manually located information of interest through the use of sophisticated searching and filtering techniques. An excellent example of this type of application is the multimedia newspaper application based on the use of personalized mobile agents developed at the University of Ottawa's Multimedia Information Research Laboratory (MIRL) [8]. In the context of the Internet, there is recognized need for a new class of agent-based information retrieval applications to overcome the inadequacies of existing web search engines [2].

Event monitoring refers to the ability of mobile agents to notify users when a particular event occurs. For example, mobile agents could notify users when a stock reaches a certain price, when a consumer item goes on sale or when tickets for a concert become available.

Network management applications can exploit mobile agent technology to monitor and configure a set of distributed network devices. For example, in a corporate computing environment, mobile agents can be dispatched to create an inventory of the hardware and software supported by each computing device in the network. Another example is the management of network devices such as routers; mobile agents can be used to collect statistics and to initiate actions in response to changing network conditions. Existing network management techniques based on protocols such as SNMP are fairly limited as remote devices can only report on network conditions without initiating action; there is a recognized need for

more sophisticated network management solutions with the intelligence to analyze and initiate corrective actions, as required [9].

Mobile computing is another very promising application, which takes advantage of the ability of mobile agents to perform functions on a user's behalf even when the user is disconnected from the network. For example, a business user could initiate a task from the office, disconnect from network and fly to another city; many hours later, the user could reconnect to the network and retrieve the results from the previously initiated activity. For applications in which the user wants to minimize connection time to the network (e.g. to reduce costly air time on a wireless network), the use of mobile agents can also be very advantageous.

Electronic commerce is another important application. The concept for a mobile agent electronic marketplace, in which mobile agents acting on their users behalf participate in commercial transactions, was proposed by J. White [11]. In one example, the user's agent consults an electronic yellow pages directory to obtain a list of stores within the region that are likely to sell the desired item. The mobile agent visits each store to obtain pricing information and returns to the store with the lowest price and purchases the item by providing the user's credit card number and shipping address. In the context of the Internet, ShopBot [12] is an example of a prototype system for purchasing CD's and computer software on the world wide web.

Workflow applications are characterized by the step-by-step execution of functions at different sites. For example, mobile agents could be used to serially route an authorization form to a hierarchical list of persons on an approval list. Another potential workflow application is the use of mobile agents during a procurement process; mobile agent technology is used to facilitate

the purchasing of items by soliciting bids from multiple vendors, identifying the best proposal and submitting an order for the desired items [4].

2.5 Research and Development Activities

2.5.1 Introduction

This section provides a brief overview into research and development on the general topic of agents. Special attention is given to activities related to mobile agents; however, there are also a number of interesting developments related to non-mobile agents that are relevant to this thesis.

There have recently been a number of organizations established for facilitating the exchange of information on the topic of agents, including the Foundation for Intelligent Physical Agents (FIPA) [13] and The Agent Society [14]. The Agent Society is comprised of a number of companies and academic institutions with the common goal of assisting the development of intelligent agent technologies.

There are currently more than 25 universities engaged in agent research, including the University of Ottawa's Multimedia Information Research Laboratory (MIRL) [15], the MIT Media Lab [16] and Dartmouth College [17]. Within industry, companies such as IBM [18], Oracle [19] and General Magic [20] are actively involved in agent research, with a growing number of Internet application developers incorporating agent concepts into their products.

2.5.2 General Magic's Telescript

As General Magic's Telescript product is generally considered to be the first commercial implementation of a mobile agent framework, it will be described in considerable detail. Telescript provides a programming environment for mobile agent application developers and was initially proposed as a means of realizing the electronic marketplace. However, with the

explosive growth of the Internet and the widespread popularity of the Java programming language amongst application developers, General Magic are now exploring the use of Java-based mobile agents for use on the Internet.

Telescript is based on the concept of agents and places. An 'agent' is a software entity, consisting of software procedures and state information, with the ability to travel and perform transactions on behalf of a user. A 'place' is a permanent site on the network (i.e. with a fixed address) and is typically populated by a single permanent agent acting on behalf of the site owner.

Telescript also introduces the concept of tickets, meetings, authorities and permits. Tickets are issued to the software agent prior to travel, and provide information on the destination, duration of trip, etc. To initiate travel, the 'go' command is executed which causes an agent to be transported from one physical computing device to another; the 'go' commands supports a variety of communications protocols, including TCP/IP, X.25 and e-mail.

Meetings provide the means by which two co-located agents interact (e.g. shopper agent interacting with store agent); meetings are initiated using the 'meet' command. Either party has the right to refuse to participate in the meeting. Authorities are intended to provide a secure mechanism for determining the 'real world' owner of the software agent (e.g. the name of the buyer to whom the financial transaction will be charged). Anonymity is not permitted within the Telescript environment in order to, according to General Magic, provide a strong deterrent against the proliferation of software viruses [6]. Permits are issued whenever an agent enters a new place, to limit the scope of the agent's activities in terms of executable instruction set and resource consumption (e.g. limits can be imposed to control the usage of CPU time, disk space, memory, etc).

The Telescript language is a high level object oriented language that is customized for use in a mobile agent environment. For example, it provides support for suspending agent execution at an arbitrary point, capturing the agent's state variables (including program counter) and subsequently resuming execution at another site. Telescript is not a scripting language; once compiled, it assumes the form of a low-level interpreted language, to be executed by a Telescript engine.

Telescript's principal weakness relates to its use of a proprietary language and the fact that it is a commercial product (i.e. unlike Tcl and Java, which are freely distributed). Other limitations include lack of explicit support for multimedia information and the requirement that agents must be co-located in order to communicate, which can lead to inefficient utilization of network resources.

2.5.3 Recent Developments

There are currently a number of agent-based products available, most of which are designed for use on the Internet. Although the majority of these agent-based products do not support the concept of mobility, this can be largely attributed to the immaturity of mobile agent technology (e.g. lack of support for a standardized application layer protocol). However, non-mobile forms of some of the mobile agent applications described are starting to appear, albeit with less sophisticated capabilities.

For example, a number of products have been released to enable users to make more efficient use of the Internet. Off-line web browsers, such as WebWhacker [21] developed by The ForeFront Group, is an example of an information retrieval application that automatically downloads web pages from sites on the world wide web (www) to a user's hard drive. While this application requires the user's computing device to be connected to the network, the user

does not need to be present when the web pages are downloaded. Furthermore, by storing the pages locally, response time for viewing web pages is significantly improved.

Filtering agents have been developed by Firefly Inc. [22] to allow a user's agent to consult with other agents to leverage their experiences. Firefly's products are based on the collaborative filtering technique developed at MIT's Media Lab [16] in which users are first asked to rate their preferences on selected topics as a means of teaching the agent about the user's likes and dislikes. The user's agent then consults with other agents to identify a community of agents with similar interests. By sharing individual preference information, the user's agent is able to recommend new items that are likely to be of interest to the user. For example, an on-line video store could present the user with a single shelf of videos that are customized to the user's tastes based upon the previously described "word of mouth" collaborative filtering technique.

Information monitoring agents are now available that notify users when web content at a particular site has been updated. Shopping agents have also been developed to electronically purchase specified items at the lowest available price.

Although not designed for use on the Internet, FTP Software Inc. have developed a network management application using Java-based mobile agents [9] that offer improvements over the traditional client-server approach used by the Simple Network Management Protocol (SNMP).

2.6 Future of Mobile Agents

2.6.1 Responsible Use of Mobile Agents

Recently, there have been serious concerns raised within the Internet community regarding the use of software robots that automatically retrieve information from sites on the

world wide web (www) on behalf of users and service providers [23]. This class of software entity is heavily relied on by commercial web search engines (e.g. Lycos, Yahoo) to continually monitor the web for new and changed web site content. In some instances, often as a result of multiple "rapid fire" requests to a single network server, server and network response times have been significantly degraded. In response to these concerns, mechanisms for allowing web site administrators to selectively control software robot access have been devised, as described in [23]. In addition, a code of ethics has been published to provide a set of guidelines for Internet software robot developers. It is expected that similar guidelines will be required to govern the use of mobile agents on the Internet.

Furthermore, a distinction can be made between the software robots used by search engines and those software agents that act on a single user's behalf. Although software robots employed by search engines consume considerable amounts of network bandwidth, they ultimately act in the general interests of the web community by providing a valuable information indexing service. Software agents sponsored by individual users, such as off-line web browsers, can also have a detrimental effect upon network performance but do not offer any benefit in return to the Internet community. For this reason, it is anticipated that mobile agents will often need to be registered with a particular site, before access to that site is granted.

2.6.2 Analysis of Current Trends

This section briefly examines a number of trends that are likely to impact the future evolution of mobile agents.

Arguably the most significant global trend during the past few years has been the explosive growth of the Internet. The Internet has become the world's largest public data network, with millions of users across all five continents. Furthermore, Internet technology is

now being transplanted into private corporate networks referred to as 'Intranets', as corporations are seeking to exploit the efficiencies of Internet technology. It is evident that mobile agent technology, if it is to flourish, must be fully compatible with the Internet environment.

Other noteworthy Internet trends include the increased popularity of multimedia applications (fuelled by technology improvements in personal computing devices), slow response times (due to network congestion) and a strong demand for more efficient navigation tools. Mobile agents are well positioned to address each of these trends, providing they can make efficient use of network resource and include provisions for handling multimedia content.

Another important trend in the past year is the emergence of Java as an architecturally neutral execution environment for the development of software applications. This trend is very promising for mobile agents, as it provides a ubiquitous and secure environment in which mobile agents can operate.

Chapter 3

3 System Architecture

3.1 Introduction

The first objective of this thesis is the development of an architectural model for mobile agents that is sufficiently flexible to accommodate a range of applications, such as multimedia information retrieval, electronic commerce, network management and work flow applications.

The approach taken in this chapter is to first identify a set of high level requirements for the mobile agent system. These requirements are further refined through the exploration of a hypothetical application that serves to demonstrate many of fundamental properties of a mobile agent system.

The next step is to decompose the high level requirements into a set of system functions, which can then be logically mapped to architectural building blocks. These building blocks are used to define the internal architecture of the principal network elements; namely, the user site and the network site. This internal architecture is subsequently used to form the basis of the prototype implementation described in Chapter 6. And finally, owing to the importance of network communication to the mobile agent system, a layered network communication model is developed.

3.2 System Requirements

3.2.1 Introduction

This section describes the high level functions that must be supported by the mobile agent system. In subsequent sections of this thesis, these high level functions will be used to guide the design of the system architecture, the mobile agent structure and the Mobile Agent Protocol

(MAP). For example, as described later in this section, a requirement for a mobile agent tracking capability has been identified. This capability will influence the development of the network site architecture, by justifying the need for a Front Desk architectural entity for registering mobile agents when they visit a network site. It will also influence the design of the Mobile Agent Protocol (MAP), by creating a need for a protocol message for determining the status of a mobile agent at a particular network site.

By carefully specifying the requirements at the outset of the project, a robust system architecture is developed that provides application developers with a powerful framework for the development of a range of mobile agent-based applications.

3.2.2 Information Discovery

Much of the motivation for mobile agents is the promise of providing end users with a more efficient model for navigating the Internet. For this reason, a key requirement for mobile agents is the ability to locate information content without direct involvement from the end user; that is, the end user simply needs to specify what information is required without specifying the location of the information.

This requirement can be translated into a need for network-based directory servers; for example, mobile agents can visit one or more directory servers at the start of their journey to build up a list of candidate sites to visit. The use of directory servers is a basic requirement for most mobile agent systems, and is previously described in [5] and [6]. Furthermore, for situations in which the user wants to limit the mobile agent's travels to a particular network site (e.g. a favourite newspaper site) or a particular set of network sites (e.g. a network of trusted sites affiliated with a trustworthy organization for electronic commerce), convenient mechanisms are required to allow users to specify the itinerary. For instance, the use of a string-based address

based on the Universal Resource Identifier (URI) scheme is required to relieve the user from manually inputting network IP addresses.

3.2.3 Information Retrieval

As information retrieval represents one of the lead applications for mobile agent technology, it is evident that mobile agents must be capable of transporting information as they travel. What is perhaps less evident is the need for mobile agents to retrieve information in a manner that makes efficient and responsible use of network resources. For example, given the dramatic increase in the use of multimedia information and considering that multimedia files are inherently large, mobile agents containing multimedia information must possess the attributes to enable them to be move through the network in a manner that is mindful of network resources. Furthermore, as previously discussed in Chapter 2, mobile agent systems must possess safeguards to address the concerns from the Internet community that mobile agents pose a threat to overall network performance.

The requirement for network efficiency can be translated into a requirement for a protocol mechanism for remotely requesting entry to a site before physically migrating to that site; this will prevent situations from arising in which a mobile agent travels to a network site only to be refused entry upon arrival. Furthermore, the system architecture must include provisions for allowing mobile agents to collect pointers to information content, such that large files can be transferred only upon arrival at the user site at the end of the mission (i.e. in lieu of hopping through the network with a large payload). A mechanism to limit the payload size of a mobile agent is also required.

3.2.4 Security

There are several dimensions of security that must be addressed by the mobile agent architecture. Network privacy is required, to prevent intermediate network devices from eavesdropping and tampering with mobile agents as they travel from site to site. From the user's perspective, users needs to be confident that the mobile agents will not be altered to perform unauthorized actions and will not divulge confidential information. Security is also an important consideration at the network site to ensure that all actions performed by a visiting mobile agent are legitimate.

These requirements can be translated into requirements for the use of interpreted languages and for the support of public key cryptography, as described in detail in Chapter 5.

3.2.5 Agent Tracking / Interception

Agent tracking and interception are two closely related functions that are required by the mobile agent system. Tracking is the ability to locate a mobile agent as it travels through the network. Tracking can be used to retrace the itinerary of a mobile agent in the event of an unreported failure condition (i.e. to locate a 'lost' agent); it is also an invaluable debugging tool for application developers. Interception refers to the ability to initiate communications with a travelling mobile agent for the purposes of updating the mobile agent's mission. For example, interception enables a user to remotely instruct a mobile agent to update its program content (e.g. in a network management application, the user may want to invoke an instruction requesting an interim report on current network conditions) or to simply abort the mission altogether.

These two functions are considered to be complementary in the sense that to intercept a mobile agent, it is first necessary to pinpoint the network location through the use of a tracking utility.

3.2.6 User Notification / Consultation

As per [6], mobile agents are well suited to event notification services, such as informing a user when a stock reaches a particular price. This requirement is specifically intended to enable mobile agents to remotely notify users of significant events (i.e. without returning to the user site); in this way, the mobile agent can continue to execute its mission while providing the user with intermediate updates.

A closely related requirement is the ability for a mobile agent to remotely consult with the user to clarify the agent's instructions, especially in the event of unexpected or ambiguous situations. For example, in an information retrieval application in which a database search on a particular topic results in an unexpectedly high number of matches, the mobile agent may want to request the user to provide additional search criteria. Given that a real-time consultation capability requires the user to be available in a quasi on-line mode (i.e. available to provide responses within a reasonable amount of time), this places a requirement on the protocol to support a message for determining the availability of the user. In cases where the user is not available (e.g. disconnected from network), the mobile agent's script is expected to provide a default course of action.

The need for a consultation service can be further justified by considering that mobile agents are likely to possess only limited Artificial Intelligence (AI) capabilities, if any, and thus are not likely to support a complex decision making process. Additionally, mobile agents' knowledge of user preferences is typically limited in scope and thus can not be solely relied upon for critical decisions. And finally, although agents are expected to be well acquainted with the remote applications with which they interact, without prior knowledge of the raw information

'content' (where 'content' is distinct from 'format'), mobile agents can not be expected to always be capable of interpreting this information.

3.2.7 Information Presentation

The final stage upon completion of the mobile agent's mission is the presentation of the results to the end user. This includes the ability to interact with other applications to assist with the presentation and final processing of the information. For example, in an information retrieval application, it may be necessary to launch an application to view a particular file (e.g. launch Netscape to view a MPEG document). In addition, the results from the mission may be exchanged with other applications residing at the user's site, including e-mail applications, word processing packages and telephony devices. It is interesting to note that Internet off-line web browser products such as WebWhacker [21] have announced plans to support user notification via e-mail, fax or pager (e.g. to notify a user when a stock price reaches a specific value).

Although beyond the main focus of this thesis, there are cases in which it may be beneficial for the initial task defined by the user to be handled by multiple cooperating mobile agents. This will require a mechanism for analyzing and consolidating the results gathered by multiple agents prior to presentation to the user. Although a single agent may be sufficient in most cases, it is envisioned that some tasks can be optimized by launching multiple agents in parallel. Research into the use of multiple cooperative agents for handling complex tasks is currently being conducted at Carnegie Mellon University [26].

3.3 Illustrative Example of an Information Retrieval Application

3.3.1 Introduction

An example of a sophisticated information retrieval application has been devised that provides an opportunity to validate all of the key system requirements described in the previous

section. The example is based on a travel reservation scenario that seamlessly incorporates elements of electronic commerce, information retrieval and workflow applications. A travel reservation application was chosen because it is arguably one of the most compelling scenarios for illustrating the power of mobile agent technology; other conceptual variations of travel reservation scenarios based on the use of mobile agent technology are described in [5] and [6]. Furthermore, this scenario provides a convenient conceptual testbed for exploring and resolving many design issues that are likely to be encountered during the development of the mobile agent based system architecture.

The remainder of this section describes a hypothetical implementation of a travel reservation application developed using a mobile agent framework. For the purposes of this example, the mobile agent application is described according to a four stage lifecycle: Initial User-Agent Interaction Stage, Planning Stage, Information Gathering Stage and Results Synthesis & Presentation Stage.

3.3.2 Initial User-Agent Interaction Stage

Prior to using the mobile agent-based travel application for the first time, it must be initialized with the user's profile (e.g. name, address, telephone number, credit card numbers, etc.) and preference information. As discussed in [27] and [28], there are several learning techniques that can be employed by the agent: the simplest mechanism requires the user to directly input all data. A more complex solution requires the application to compile user profile and user preference data based on observation of the user's behavior, preferably over an extended period of time.

User profile information is used to denote unambiguous facts (e.g. user's name, address, frequent flyer memberships, etc.), whereas user preference data indicates assumed behaviors (e.g.

user prefers hotel with indoor pool close to airport). The application must be capable of extrapolating facts to derive preference information (e.g. a non-smoker will prefer a non-smoking room and car). Furthermore, similar to the mechanisms referenced earlier, an internal weighting scheme should be used to capture the relative strength and importance of each of the user's preferences. For example, a user could indicate a strong preference for a window seat on a plane but rank the relative importance of this preference as low. In the event that window seats are not available, user is likely to accept a non-window seat rather than to wait for another plane.

Subsequent to the inputting of user profile and preference information, the application prompts the user for a description of the high level task to be performed. In this example, the user indicates the nature of the trip (e.g. business vs. personal, where 'business' may imply rigid travel dates and 'personal' may imply travel date flexibility and a preference for lower cost travel), the departure and destination cities and the desired travel dates. Other supplementary information such as vehicle profile (e.g. mid-size car with air conditioning), flight profile (e.g. vegetarian meal), accommodations profile (e.g. hotel with indoor swimming pool, room with king size bed, room rate not to exceed $100, etc.) can be derived by the application and presented to the user for confirmation or modification.

In general, the application should support a range of options for determining which network sites to visit. For example, the application could be pre-programmed to visit only those sites that are affiliated with a particular trusted organization. Another possible approach is to allow the mobile agent to autonomously discover the location of network sites by visiting one or more directory servers.

To facilitate the interactions between the user and the application, a Graphical User Interface (GUI) is recommended; to provide a more sophisticated user interface, voice recognition technology could be used.

3.3.3 Planning Stage

During the planning stage, the application builds a script that captures all of the tasks to be performed by the mobile agent. The script, which contains a customized set of executable instructions for the mobile agent, is typically derived from an existing library of script templates. It is expected that a given application will support a number of scripts, and will select the script that best matches the task at hand.

In some instances, it may be advantageous to employ multiple mobile agents, in which case the application must partition the high level task across several scripts, identify the relationships between the scripts and develop an overall communications strategy for mobile agent collaboration. Although in most cases a single mobile agent will suffice, this example adopts a 'worst case' approach and assumes the use of multiple cooperating agents, with each agent containing a single script. For this example, a script is created for each of the three items for which reservations are required; namely, accommodations, plane and car. As each of the three reservation items must be carefully coordinated (e.g. car reservation pickup time must coincide with expected flight arrival time at destination city), each mobile agent must be synchronized with its peers.

In the final stage of planning, the travel application identifies the first network site to be visited by each mobile agent. In many cases, the first network site corresponds to a directory server.

3.3.4 Information Gathering Stage

During this stage, the individual mobile agents travel throughout the network visiting the sites of interest. To exchange synchronization information, the mobile agents can track the progress of their peers through the use of a tracking utility. For example, if there are no flights available on the user's preferred departure date, each mobile agent can be located using the tracking utility and updated with a new departure date. Update information can be shared by converging at a particular network site or by remote message passing.

In some cases, it may be necessary for a mobile agent to establish contact with the user to clarify the original task. For example, consider the case where the mobile agent responsible for accommodations visits a network site and locates more than a hundred hotels that satisfy the user's selection criteria. At this point, the mobile agent could consider returning to the user with multimedia files describing each of the available hotels. However, given the inefficiency of this approach (e.g. wasteful of network bandwidth, high probability that user will discard most of the retrieved information), a better alternative is to consult with the user to further refine the selection criteria (e.g. prompt the user to select a particular district within the city).

3.3.5 Results Synthesis and Presentation Stage

During this stage, each of the mobile agents returns to the user site and shares its results with the application. The application consolidates all of the retrieved information and discards any information considered to be superfluous. The application can then present the user with a small set of travel options that are customized to the user's tastes. For each of the options, supplementary information may be presented to assist the user with the final selection process (e.g. actual room cost, image of the interior of the room, video of the street scene that can be viewed from the room, etc.).

Upon selection of a preferred travel option, the application will often need to initiate further actions. For example, to pay for a plane ticket or to cancel a previously made reservation, the application may launch a new mobile agent or simply use remote message passing to communicate with a particular network site. Local actions may also be initiated at the user site, such as updating the user's personal scheduler or completing a travel pre-authorization form for corporate travel. Furthermore, given the suitability of mobile agents for workflow applications, the task of 'walking' the pre-authorization form through the various levels of authority could be automatically handled by a mobile agent.

3.4 System Architecture

3.4.1 Introduction

This section describes an architectural model for a mobile agent system that meets the system requirements identified earlier in this chapter. A network view of the mobile agent framework is described, followed by a detailed description of each of the key architectural components. The architecture described in this section forms the basis for the mobile agent-based Multimedia Document Retrieval prototype described in Chapter 6.

A network view of the mobile agent framework is depicted in Figure 1 and consists of three principal network elements: the user site, the mobile agent and the network site.

The user site is responsible for a number of functions, including interacting with the user, initializing and launching the mobile agent and displaying the contents of the mobile agent upon completion of the mission.

Mobile agents are self-contained structures that are capable of migrating through the network and executing at remote network sites. Mobile agents possess a finite degree of

autonomy, in that they can make decisions on a user's behalf without requiring direct intervention from the user. Mobile agents contain executable program code that is interpreted at network sites. The mobile agent internal structure is described in Chapter 4.

The network site is responsible for maintaining a secure environment for hosting mobile agents and for providing services to mobile agents. Two essential network elements for most mobile agent systems are directory servers and security servers. The directory server shown in Figure 1 represents a specialized instance of a network site; it is responsible for assisting mobile agents to discover the location of network sites of interest. The security server maintains a register of private and public encryption keys to facilitate secure transactions and is typically administered by a trusted third party security agency.

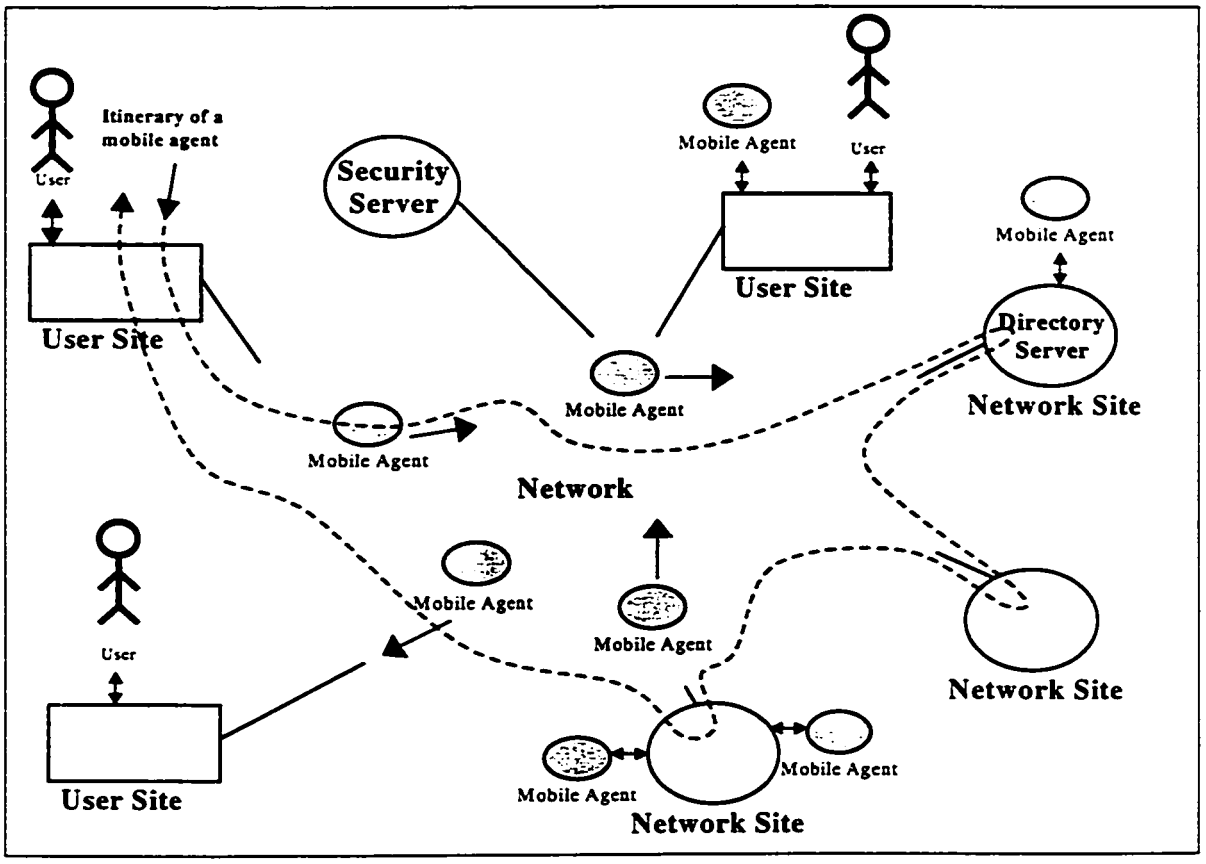


Figure 1 - Network View of Mobile Agent Framework

3.4.2 User Site Architecture

3.4.2.1 Overview

As shown in Figure 2, the user site contains three architectural components that provide explicit support for mobile agents: the Network Message Handler (NMH), the Agent Execution Environment (AEE) and the user application. The NMH and the AEE are also used as part of the internal architecture of the network site.

The NMH is responsible for managing all aspects of mobile agent communications over the network. The AEE provides an environment for interpreting mobile agent scripts. The user application is responsible for interacting with the user (e.g. via a Graphical User Interface), for creating the mobile agent (i.e. building a customized script for each mobile agent), for displaying the contents of the mobile agent upon completion of the mission and for interacting with other applications residing at the user site. Each of these components is described in detail in the following section.

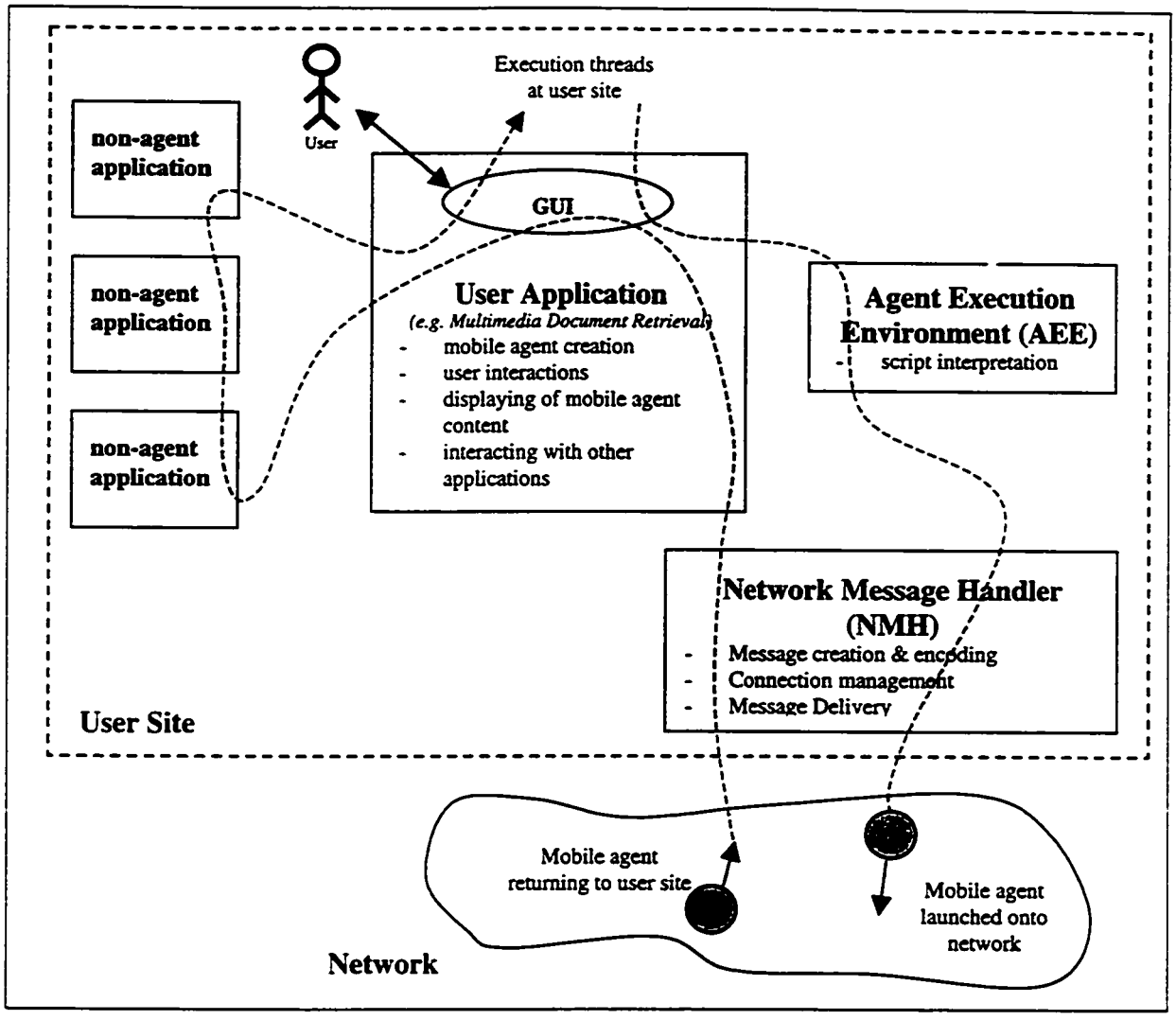


Figure 2 - User Site Architecture

3.4.2.2 Network Message Handler (NMH)

The Network Message Handler (NMH) is common to both the user site and the network site. It is responsible for handling network communications, including the implementation of the Mobile Agent Protocol (MAP). Key functions include message creation, message encoding, connection management and message delivery. For example, to launch a mobile agent onto the network, the NMH encapsulates the individual components of the mobile agent into a Protocol Data Unit (PDU), encrypts and authenticates the message (as required), encodes the PDU into an

appropriate transmission format, opens a network connection to the destination host and writes the data into the appropriate output buffer.

3.4.2.3 Agent Execution Environment (AEE)

The Agent Execution Environment (AEE) architectural component is common to both the user site and the network site. It is responsible for providing an environment in which mobile agent scripts can be safely executed; for example, the set of instructions that can be executed by a mobile agent is carefully controlled to ensure that the mobile agent's activities do not adversely affect the host site. In the context of the user site, the AEE typically performs a fairly limited role; namely, it serves as a convenient mechanism for launching mobile agents onto the network.

When the user is ready to launch a mobile agent, the user application sends the mobile agent's script to the AEE. The AEE initializes the script and starts the interpretation process: typically, one of the first instructions in the script is the JUMP command, triggering the AEE to suspend program execution, capture the state variables and instruct the NMH to initiate the transfer of the mobile agent to the specified network site.

3.4.2.4 User Application

The user application is simply a mobile agent-based software application. Key functions include 'learning' the user profile and user preference data, capturing the user's high level task description, building a customized executable script that represents the user's task and sending the script to the AEE to be launched onto the network. For incoming mobile agents, the user application is responsible for presenting the contents of the mobile agent to the user and for interacting with other applications at the user site.

3.4.3 Network Site

3.4.3.1 Overview

As shown in Figure 3, the network site contains a number of architectural components, including the Network Message Handler (NMH), the Agent Execution Environment (AEE), the Front Desk entity, the System Administrator entity and one or more service applications. As previously described, the NMH and the AEE are also used within the internal architecture of the user site.

The NMH is responsible for handling network communications. The AEE provides an environment for interpreting mobile agent scripts. Analogous to the role of a front desk at a hotel, the Front Desk entity is responsible for controlling access to the network site, for maintaining a registry of all mobile agents that visit the network site and for responding to inquiries regarding system status and available services. The System Administrator entity provides a set of Operations, Administration and Maintenance (OA&M) tools for enabling the network site owner to efficiently operate the site. And finally, the service applications are software applications with which the mobile agent interacts to obtain information. Each of these components is described in detail in the following section.

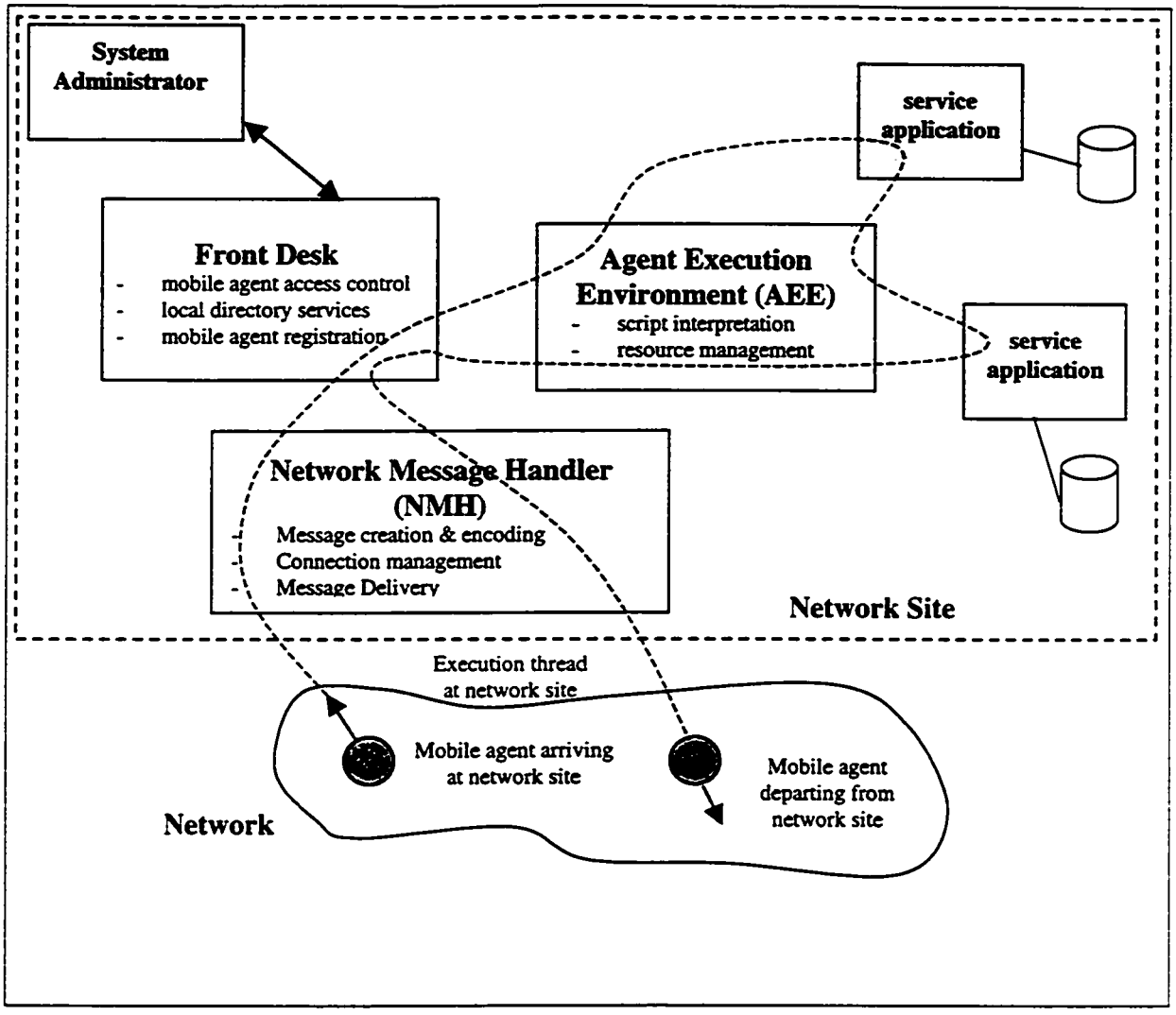


Figure 3 - Network Site Architecture

3.4.3.2 Network Message Handler (NMH)

As previously described, the Network Message Handler (NMH) is responsible for handling all aspects of network communications, including connection management, message decoding, message validation and message delivery. For example, for mobile agents arriving at a network site, the NMH monitors the network for TCP/IP connection requests from other host sites. Upon receipt of a request, the NMH sets up a connection and reads the incoming data from the appropriate input buffer. The NMH decodes the message, performs decryption and

authentication (as required), validates the message (e.g. to check for protocol violations) and, upon recognizing the message as a mobile agent, disassembles the mobile agent in accordance with the defined mobile agent structure and notifies the Front Desk entity of the agent's arrival.

3.4.3.3 Front Desk

An important consideration for most mobile agent systems is the ability to inspect an incoming mobile agent's credentials. In Telescript, credentials are checked whenever a mobile agent enters a new region, where a region is defined as "a set of Telescript places all operated by the same individual or organization" [11]. In IBM's itinerant agent framework [5], the use of a 'Concierge' is proposed, a site resident gatekeeper that inspects the 'passport' of incoming mobile agents and assists outgoing mobile agents in reaching their next destination.

In the architectural model proposed in this thesis, the Front Desk provides two basic functions; namely, it controls access to the network site and provides local directory services. Upon arrival of a mobile agent, the Front Desk inspects the mobile agent's credentials contained within the mobile agent's structure and determines whether to accept or reject the mobile agent. Furthermore, depending on the sensitivity of the services offered at the network site (e.g. electronic commerce), the Front Desk may require the user's credentials to be authenticated (through the use of public keys). If the mobile agent is permitted entry, the Front Desk will send the mobile agent's script to the AEE for execution. Otherwise, the Front Desk will instruct the NMH to reject the mobile agent in compliance with the application layer protocol. As an option, the Front Desk supports a prioritization service to instruct the AEE to allocate additional resources (e.g. improved response time) to preferred mobile agent customers.

The Front Desk maintains a registry of mobile agents that are currently visiting the site, as well as an archive of mobile agents that have recently visited the site. The Front Desk also maintains a listing of available services and overall system status.

3.4.3.4 Agent Execution Environment (AEE)

An essential architectural component of any mobile agent system is the agent execution environment. In the system architecture described in this thesis, the Agent Execution Environment (AEE) provides two primary functions; namely, script interpretation and resource management, both of which are closely related.

Script interpretation refers to the ability to execute the program contained within the mobile agent, which is typically encoded using a scripting language such as Java or Tcl. Scripting languages are generally supported across a range of computing platforms and thus are suitable for use in a heterogeneous network environment. Scripting languages also provide mechanisms for controlling the actions of the executing program (e.g. to prevent unauthorized access to the local file system).

Resource management refers to the ability to control the system resources that are made available to individual mobile agents. Such controls are necessary to prevent individual mobile agents from monopolizing local resources. Upon arrival of a mobile agent, the resource manager allocates execution resources to the mobile agent; upon completion of execution, the resource manager suspends the program execution, captures the state variables and de-allocates the execution resources.

3.4.3.5 System Administrator

The System Administrator entity is responsible for all aspects of network site administration, including the collection of usage statistics and system configuration data. The System Administrator entity is responsible for the resource allocation policy and the assignment of privileges to customers; it also provides utilities to force the termination of executing mobile agents.

3.4.3.6 Service Applications

Service applications provide mobile agents, acting on behalf of users, with access to information services. As the majority of information services require access to database resources, a key role of service applications is to translate mobile agent requests into database queries and, based on the database response, to deliver the requested information to the agent. The service application is also responsible for billing the user for the services used and for providing this information to the Front Desk for inclusion in the body of the mobile agent.

An essential consideration of any mobile agent-based system is the representation of information in an application independent way, such that a mobile agent and a service application interacting at a network site can 'speak the same language'. For example, consider the problem of trying to design a mobile agent to communicate with a world wide web (www) server. Web sites are designed to present information in a way that is appealing to the human user, whereas mobile agents require a much more structured and standardized method of representing information [12]. For this reason, the development of a standardized query language and knowledge representation is essential for the widespread availability of mobile agents on the Internet. The Simple HTML Ontology Extensions (SHOE) project at the University of Maryland

[29] is an example of a research activity aimed at making web pages readable to mobile agents, by specifying machine-readable HTML extensions for annotating web pages.

The development of a language for facilitating dialogue amongst different software entities is beyond the scope of this thesis. However, it is noted that a declarative language such as Agent Communication Language (ACL) is well suited for facilitating communications between independently developed applications [30]. ACL consists of three parts: a vocabulary (i.e. a dictionary of words, often customized for a particular application), an 'inner' language for encoding knowledge and an 'outer' language known as Knowledge Query and Manipulation Language (KQML), for building messages between the sender and the receiver. In the context of the mobile agent framework described in this thesis, ACL could be used as a common language to facilitate the communications between a mobile agent executing at a particular site and the local service applications.

There are several research activities that explore the use of KQML with agents. For example, IBM have described an abstract framework for mobile agents based on the principles of KQML [5]; research at Stanford University has described the use of KQML-like agents and their applicability to the world wide web [3].

3.5 Network Communications Architecture

3.5.1 Introduction

An important aspect of this thesis is the design of an application layer protocol for mobile agents. Although the detailed specification of the Mobile Agent Protocol (MAP) is described in Chapter 5, this section develops a layered architectural model for supporting this protocol.

Based on the system requirements identified at the outset of this chapter, it is evident that two modes of network communication are required. The first mode of communication involves the transfer of mobile agents across the network; given the 'intelligence' of mobile agents, this type of messaging can be referred to as 'intelligent messaging'. The second mode of communication is remote messaging passing, which represents the traditional technique for sending messages between two hosts. Both forms of communication are complementary; for example, prior to transferring a mobile agent with a payload of multimedia documents across the network, lightweight protocol messages can be exchanged between host sites to ensure that the mobile agent transfer is warranted. In this way, network efficiency can be optimized.

3.5.2 Layered Architectural Model

3.5.2.1 Overview

As depicted in Figure 4, the main functions of the application layer protocol stack can be logically partitioned into four sub-layers. From a system architecture perspective, the bottom three layers belong to the Network Message Handler (NMH) and the upper layer is associated with either the Front Desk entity (network site) or user application (user site).

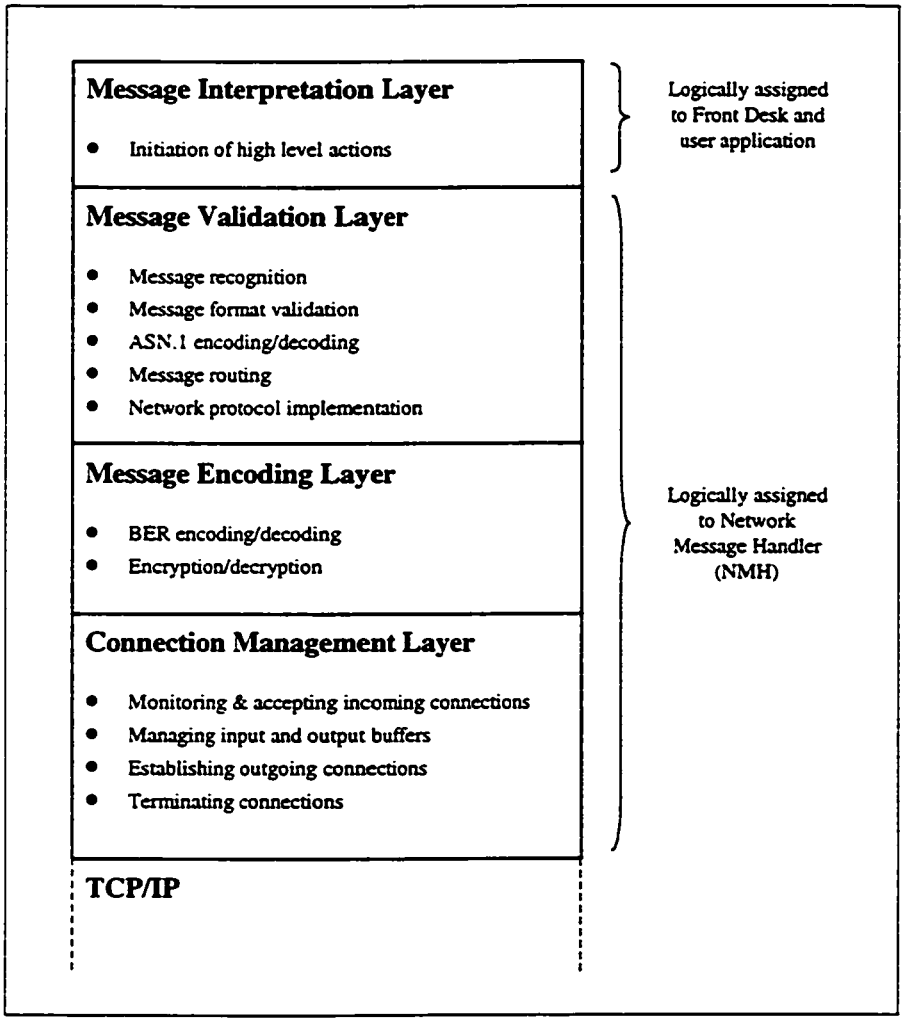


Figure 4 - Application Layer Architectural Model

3.5.2.2 *Connection Management Layer*

The connection management layer functions include monitoring and accepting incoming connection requests and notifying the next higher layer when data is received in the input buffer. Conversely, when data from the upper layer is received, the connection management layer is responsible for establishing outgoing connections. Upon completion of messaging dialogue with other hosts, the connection management layer closes the connection. At this layer, all information is viewed as a stream of bytes.

3.5.2.3 *Message Encoding Layer*

The message encoding layer is used to encode outgoing messages and to decode incoming messages. For incoming messages, this layer processes the raw stream of data bytes received from the adjacent lower layer and organizes the byte stream into messages. For example, header information pertaining to total message length is processed; if the expected number of bytes is not received, the message encoding layer generates an error message. Using a transfer syntax such as Basic Encoding Rules (BER) specified by ITU [31], the incoming message is organized according to a Type–Length–Value structure and passed to the next higher layer. If the message has been encrypted, it will be decrypted prior to notifying the adjacent layer.

3.5.2.4 *Message Validation Layer*

The message validation layer is responsible for validating incoming messages to ensure they adhere to the protocol structure and for delivering the message to the appropriate application entity. Messages are represented at this layer using the Abstract Syntax Notation One (ASN.1) [32] protocol specification language; it is at this layer that different message types are distinguishable (e.g. 'intelligent' messages can be distinguished from traditional messages). Upon receipt of a mobile agent, this layer assigns the internal components of the mobile agent to

a set of local variables and assigns a port identifier that is used as an internal index for the duration of the mobile agent's visit. This layer is also responsible for maintaining timers and state machines for the protocol messages. If the message is valid, the contents of the message are passed to the uppermost layer for final interpretation; otherwise, an error message is generated.

3.5.2.5 Message Interpretation Layer

The uppermost layer of the protocol architecture is the message interpretation layer. Based on contents of the message, this layer will initiate higher level actions such as notify the end user of an event or transfer the script of a mobile agent to the AEE.

Chapter 4

4 Mobile Agent Structure

4.1 Introduction

One of the areas of focus of this thesis is the design of a mobile agent structure that provides the flexibility to support a range of applications. To this end, a generic mobile agent structure was specified based on a set of application-independent components. To facilitate agent portability between different applications, this structure was described using the Abstract Syntax Notation One (ASN.1) formal specification language [32]. Furthermore, the agent structure described in this chapter is imbedded into the application layer Mobile Agent Protocol (MAP) defined in Chapter 5.

To view the ASN.1 specification of the mobile agent structure, refer to the ASN.1 specification of the PUSH – TRANSFER_AGENT message of the Mobile Agent Protocol (MAP) in Appendix A.

4.2 Design Approach

Although it is possible to construct a mobile agent as a single executable script with state information, a design decision was made to define a set of reusable components that are specified externally from the script. This approach, which is proposed in IBM's itinerant mobile agent framework [5], provides a convenient and logical separation of the agent data from the executable script and greatly simplifies the inspection of incoming agents upon arrival at a network site. For example, the user component can be easily extracted from the mobile agent structure and sent to the Front Desk for validation of credentials (e.g. to determine whether the

user who is represented by the agent is authorized to use the services provided by the site). In this way, the network site can determine whether to accept or deny access to the agent without having to actually execute the agent. Furthermore, by using separate components for storing the executable script and command library, the process of loading a mobile agent's executable program content is greatly simplified.

Another example is the use of a separate logical component for capturing site journal information (e.g. site name, list of services used, error logs, etc.). To prevent a network site from accessing information recorded at previously visited sites (e.g. a site administrator observing the services that were provided at a competitor's site), each site journal can be encrypted using public key technology to ensure that the contents of the journal file can only be read by the user.

The mobile agent structure developed as part of this thesis is comprised of six reusable components as shown in Figure 5. A detailed description of each of these components is contained in the following sections.

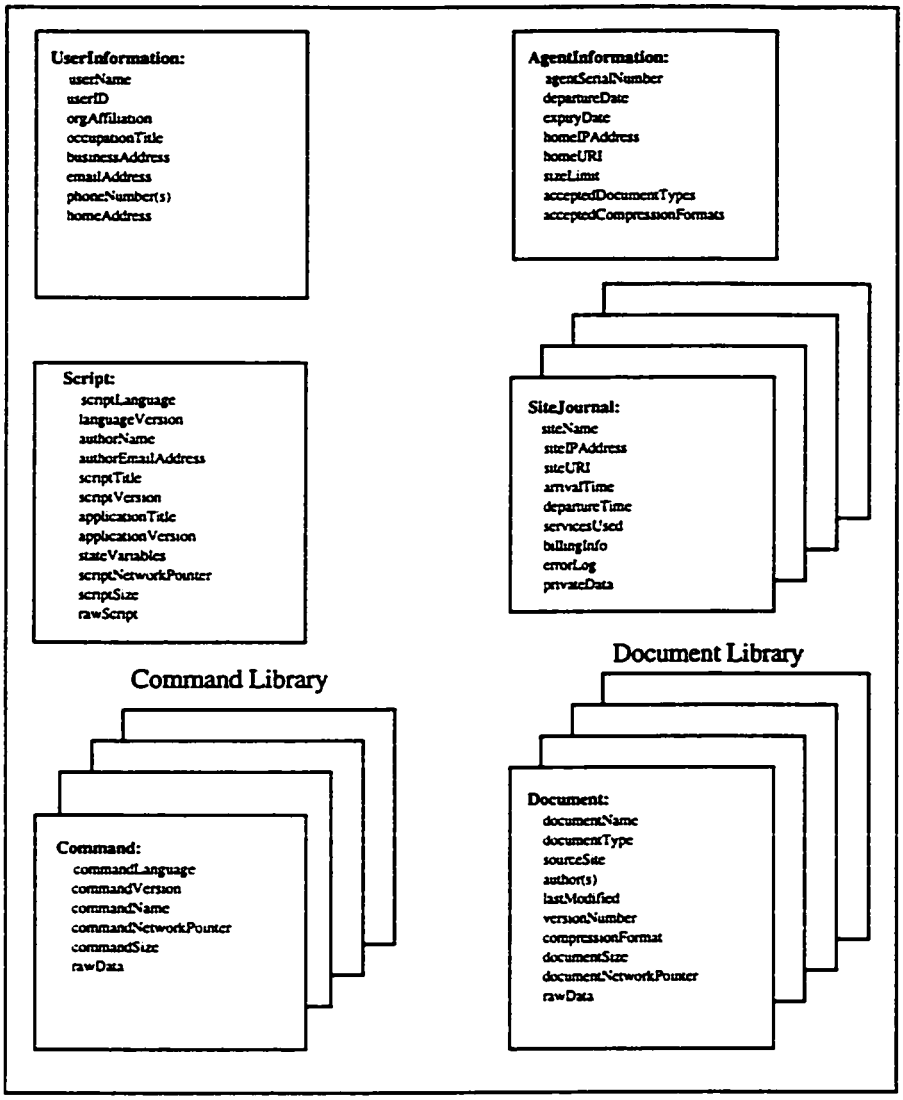


Figure 5 - Mobile Agent Structure

4.3 Description of Structural Components

4.3.1 User Information

The User Information component contains information pertaining to the user, including user name, organizational affiliation and e-mail address. Depending on the application, there may be little value or interest in using all of the fields supported by the User Information component. For example, due to privacy considerations, a user may choose limit the amount of personal

information that is made available to the network site. For this reason, many of the fields are defined as optional, to be included or excluded from the agent structure at the discretion of the application programmer and/or the user.

The inclusion of a user ID represents an important concept for mobile agents. It is envisioned that for most applications, the user will pre-register with the administrator of the application to receive a unique user ID. For example, within a closed system such as a corporate Intranet or a university campus, the user ID may be defined as the employee's ID or student number, respectively. User IDs are used to ensure that only authorized users are able to access network services and can be used by network site administrators to track and bill service usage on a per user basis. Furthermore, for sensitive applications in which strong security precautions are required (e.g. electronic purchasing), the user ID can be used in concert with public key cryptography. For example, a network site wishing to validate the authority of an agent can use the user ID to query a third party security administrator to obtain the user's public key. By using the user's public key to decrypt the user's digital signature, the network site can verify the authenticity and integrity of the incoming mobile agent.

As mentioned above, digital signatures can be used to provide security in mobile agent applications. Given that mobile agents typically traverse multiple network sites and given the dynamic nature of the agent's content (e.g. accumulation of multimedia documents, modification of state variables, etc.), it is recommended that only the static parts of the mobile agent be secured with a digital signature. This is described in greater detail in Chapter 5; specific provision are built into the Mobile Agent Protocol (MAP) for handing digital signatures.

4.3.2 Agent Information

The Agent Information component contains information that relates specifically to the software agent. Given that a single user application may simultaneously employ multiple mobile agents, an agent serial number is required to uniquely identify each of the user's agents. In this way, individual agents can be uniquely tracked and communicated with as they migrate through the network.

Another important field within the Agent Information component is the expiry date, similar in concept with the Time To Live (TTL) parameter specified in the IP protocol [34]. One of the concerns regarding the widespread usage of mobile agents in a public environment, such as the Internet, is the uncontrolled proliferation of agents. By specifying an expiry date for each agent, responsible users can ensure that the life span of their mobile agents is controlled. In this way, any network sites encountering a mobile agent that has reached its expiry date can be programmed to automatically remove the expired agent from the network.

Other fields of interest include initial departure time from the user's site, the agent's home IP address on the network and size limit. For information gathering applications that collect multimedia documentation, a size limit is used to prevent individual agents from becoming too large. To optimize the use of network resources, it is recommended that upon reaching the pre-specified size limit, any additional documents gathered by the mobile agent be collected as references (e.g. URLs) using the document pointer field. In this way, upon arrival at the user's site, referenced documents can be directly fetched by the user application.

4.3.3 Script

The Script component contains the executable script that expresses the agent's tasks. In addition to containing the script itself, a number of other fields are supported to provide the

network site with additional information. For example, information is provided to describe the language and version number of the script to allow the network site to directly determine whether the appropriate agent execution environment is supported. Author name and e-mail address provides the network site administrator with a contact within an application development agency in case of any program execution irregularities. Application name and version number fields are also supported, to further identify the source and context of the script.

In some cases, it may be desirable to provide a reference to the executable script (e.g. the network address of a software distribution server) rather than including the script directly in the body of the agent. For this reason, the script pointer field is supported.

The state variables are specified externally from the script. Depending on the sophistication of the application and the scripting language, the state variables may contain a register of variable names and corresponding data values and may even support the capturing of the entire program execution stack. When a mobile agent departs from a particular site, the state information is captured by the Agent Execution Environment. Upon arrival at the next network site, this preserved state information is restored.

It should be noted that a mobile agent contains only one script component. Although the use of multiple scripts within a mobile agent was considered, it was determined that such an approach would introduce unnecessary complexity with very little benefit. For example, upon arrival at a new network site, it would be necessary to determine which scripts to execute from the set of available scripts. Furthermore, if one script initiated a request to migrate to a new network site, this request would have to be arbitrated with the needs of other scripts that may not have completed their execution cycle. Even the task of sending a message to a remotely located agent becomes more complex when there are multiple scripts involved.

4.3.4 Site Journal

The Site Journal component is used to capture information pertaining to the mobile agent's activities at each site. Upon departure from each network site, a record of the agent's visit is produced and captured within the body of the mobile agent. This information can be subsequently examined by the user application to determine exactly what functions were performed by the mobile agent at each site; in the event of program execution difficulties, the error log field within the site journal can provide an invaluable source of debugging information.

Key data fields include the name and address of the network site, arrival and departure times and a list of services used. For services in which the user is billed, a billing information field is included to record all user charges. There is also a private data field for notifying the user application of site-related news such as changes in service charges and availability of new services.

An important benefit of the Site Journal component pertains to security. In some applications, it may be necessary to hide information gathered at one site from the next site that is visited (e.g. a shopping application in which the mobile agent searches for the best offer; if a subsequently visited network site has access to the information provided by its previously visited competitor, it gains an unfair competitive advantage). In these cases, it is recommended that any sensitive data from the site be captured in the private data field of the Site Journal component and that each network site encrypt its site journal using the user's public key. In this way, the contents of the site journal can only be viewed by the user and its contents remain hidden to all intermediate sites.

4.3.5 Command Library

The Command Library component can be viewed as a supplement to the executable script. In the context of a Tcl/Tk script, it is possible that the network site does not support all of the commands contained in the mobile agent's script. In this case, the command library can provide a pointer to the network address (e.g. URL) of a software distribution server where the referenced command can be found, as otherwise the mobile agent's script will not be successfully executed. This assumes that the referenced command is expressed in a safe, interpreted language such as Tcl/Tk, as the introduction of command coded using a traditional programming language (e.g. C++) could leave the network site vulnerable to security violations. It is also possible to directly imbed the code for each command in the body of the mobile agent, in the event that the referenced command is not widely supported in the network.

In general, the Command Library contains a collection of individual commands and/or classes that are referenced by the mobile agent's script and may not be available at all network sites.

4.3.6 Document Library

The Document Library provides a convenient structure for representing documents that are accumulated as the mobile agent travels through the network. For information retrieval applications, the Document Library represents the agent's payload.

For each multimedia document stored within the library, supporting data fields such as document name, application type and date of last modification are included. Other fields are included to describe document type (e.g. MPEG) and compression format (e.g. gunzip). A document pointer field is also supported, which provides a network pointer to documents that are not directly stored within the mobile agent structure. For example, if a mobile agent is gathering

large multimedia files from several network sites and has reached its size limit, the mobile agent can simply store a network pointer to the document. Upon arrival at the user's site, the mobile agent system can initiate a direct file transfer.

Chapter 5

5 Network Communications Protocol

5.1 Introduction

5.1.1 Motivation

This chapter describes an application layer network communications protocol specifically designed for mobile agents, referred to as the Mobile Agent Protocol (MAP). This protocol is designed to operate on the TCP/IP protocol stack, and therefore is compatible with the communications infrastructure provided by the Internet.

There are several motivating factors for developing an application level protocol for mobile agents. As identified in earlier sections of this thesis, an essential requirement of the mobile agent system is to provide a secure and efficient mechanism for facilitating the movement of mobile agents through the network. To meet this requirement, an application level protocol that supports built-in security functions and lightweight 'look-ahead/discovery' messages is required. Secondly, to provide an environment for sophisticated mobile agent functions such as communicating with a remotely located agent and interacting with the end user during the mission, a customized set of messages is required. Another consideration is the need to incorporate utilities that are directly supported in the protocol message set for requesting and transferring multimedia documents from network sites. And finally, a generic model for encapsulating mobile agents within a protocol message is required, with the flexibility to support a range of mobile agent-based applications.

5.1.2 Overview of the OSI Model

Although the focus of this chapter is on designing an application layer protocol for mobile agents, the application protocol is dependent on the lower layers to provide a number of essential services. For this reason, it is important to establish the appropriate context for the protocol design by briefly discussing the OSI model.

The Open Systems Interconnect (OSI) reference model was developed by the International Standards Organization (ISO) to facilitate the development of network communications standards. It introduces the concept of a layered model, in which seven protocol layers are defined, each supporting a logical set of functions. Each layer relies on the next lower layer to perform more primitive functions; in this way, lower level details can be easily concealed from higher levels. Furthermore, as can be expected, each layer is responsible for providing services to the next higher layer.

The OSI protocol stack supports both peer-to-peer and layer-to-layer messages. Peer-to-peer messages consist of 'horizontal' messages sent between peer entities in different host systems, whereas layer-to-layer messages are messages that are internally used by a given layer to communicate with adjacent layers. Although peer-to-peer messages are sent between peer entities at different host sites, the process of physically transmitting these messages requires that the messages be passed down through a given protocol stack to the lowest layer (i.e. the Physical Layer). At the physical layer, raw bits are transmitted across the network to the destination site and then passed back up the protocol stack to the appropriate layer (i.e. the peer entity). A diagram of the OSI reference model is shown in Figure 6.

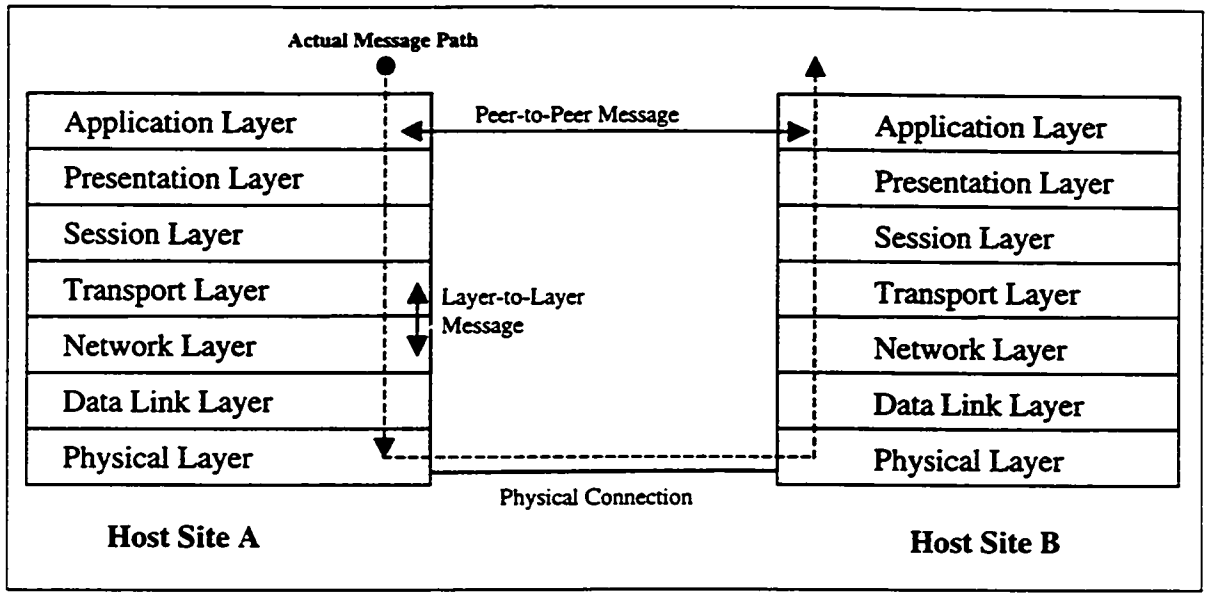


Figure 6 - OSI Reference Model

Given the focus of this research on the TCP/IP protocol stack, it is instructive to note that TCP/IP was proposed several years prior to the OSI model. TCP/IP employs a four layer model and was developed with a specific set of protocols in mind (i.e. IP, TCP, UDP). Fortunately, the TCP/IP protocol stack can be easily mapped onto the OSI model, as shown in Figure 7. A detailed discussion on TCP/IP and its relation to the OSI reference model can be found in [33].

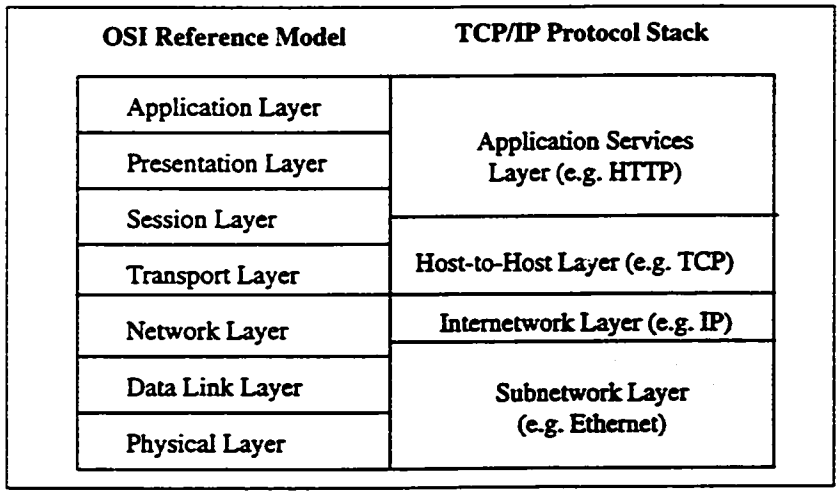


Figure 7 - Comparison Between OSI Reference Model and TCP/IP Protocol Stack

5.1.3 TCP/IP Protocol Stack

5.1.3.1 Network Layer

The network layer of the TCP/IP protocol stack, referred to as the Internetwork layer in TCP/IP terminology, is provided by the Internet Protocol (IP). IP provides the foundation upon which the Internet is based, and is supported in both Local Area Network (LAN) and Wide Area Network (WAN) environments.

IP provides a connectionless packet delivery service that masks all details of the datalink and physical layer protocols from the upper layer protocols. For example, whether IP is supported over Ethernet, Frame Relay or ATM, it has no effect upon the application layer. Another important characteristic of IP is that it supports a universal addressing scheme that is hierarchically structured; this provides a universal structure for the configuration of network routers.

IP uses a best effort approach and thus does not provide a guaranteed packet delivery service; if packets are lost in transit (e.g. discarded by a router due to overflow conditions), there is no indication sent to the source host at the IP layer. This greatly simplifies the protocol (e.g. no support for sequencing, retransmission of lost packets, etc.) and implies that for applications requiring a guaranteed delivery service, a 'reliable' transport layer protocol such as TCP must also be used.

Other functions of IP include fragmentation and payload indication. Fragmentation enables the original IP packet to transit networks that support a smaller packet size; upon arrival at the destination host, the original IP packet is reassembled from its segmented parts. The payload indication is implemented by the protocol indicator field, which identifies the message

type encapsulated within the IP packet (e.g. TCP, ICMP, UDP). An overview of the key IP packet fields is shown in Figure 8 as per [34].

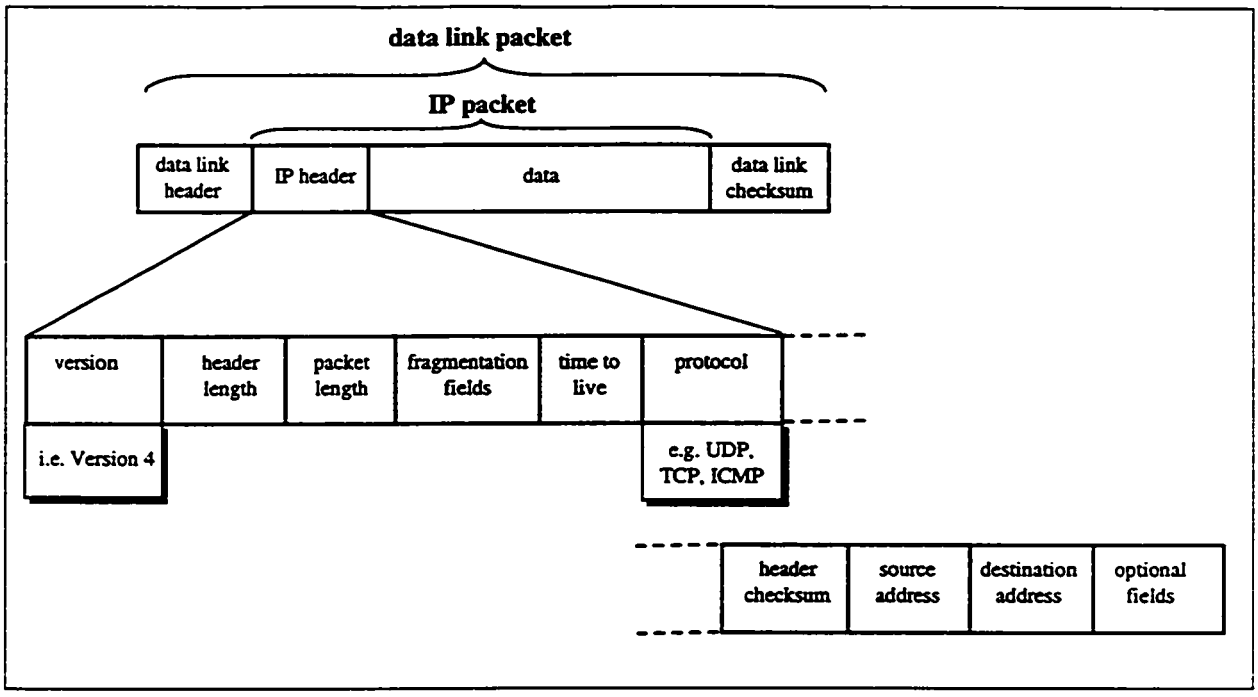


Figure 8 - Overview of IPv4 Packet Format

5.1.3.2 Transport Layer

Within the TCP/IP protocol stack, there are two transport layer protocols defined: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). TCP provides a stream oriented, reliable packet delivery service, whereas UDP uses a best effort approach that does not guarantee packet delivery. TCP is a connection-oriented protocol, with explicit support for opening and closing connections between two host machines. TCP is suitable for non real-time applications in which there is sufficient messaging dialog exchange between host sites to justify the overhead of connection establishment (e.g. Telnet, FTP, HTTP). UDP supports a broadcasting capability and, due to its lightweight nature (e.g. lack of support for retransmission), is well suited for real-time applications. UDP is also used for applications that do not require a significant amount of messaging dialog (e.g. SNMP).

Given the non real-time nature of mobile agent applications and the need for connections (justified by potentially large message sizes and frequent messaging dialogue between host sites), TCP was selected for use with the Mobile Agent Protocol (MAP).

Key features of TCP include error detection, sequencing, control flow and retransmission. Error detection is handled through the use of a checksum indicator. The sequence number and acknowledgement number is used to account for each byte of data transmitted in each direction. Control flow is implemented using a variable size window technique. And finally, in the event of checksum failures or detection of missing data bytes, TCP automatically requests retransmission.

TCP uses an address scheme based on port numbers. As the lower layer IP protocol is responsible for transmitting packets between host sites (i.e. based on IP address), TCP only needs to provide a mechanism for specifying the port number within a given host site. When a TCP connection is established, the source host often assigns an arbitrary port number (e.g. 15679); however, to ensure the packet is received by the correct process at the destination host, RFC 1700 [35] defines a number of "well known ports". For example, Domain Name Servers use port 53, HTTP uses port 80 and Mobile IP Agents are designated to use port 434.

An overview of the key TCP packet fields is shown in Figure 9 as per [36].

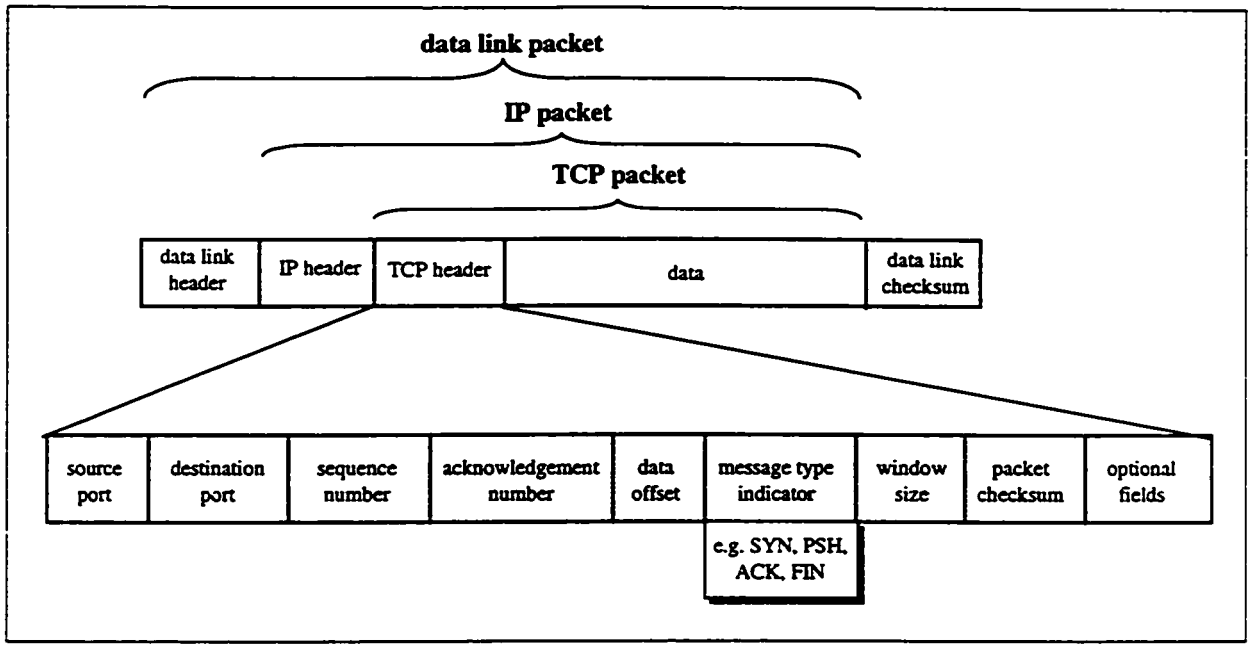


Figure 9 - Overview of TCP Packet Format

5.2 Application Layer Protocol Design Considerations

5.2.1 Evaluation of Existing Protocols

5.2.1.1 Introduction

In this section, three different TCP/IP-based application layer protocols are briefly described, to provide a better understanding of real-world protocol designs and to provide a general context for the development of the Mobile Agent Protocol (MAP).

5.2.1.2 Simple Network Management Protocol (SNMP)

SNMP is a UDP-based protocol developed to provide a standardized protocol (i.e. vendor independent) for centrally managing routers and other network devices. As the name suggests, SNMP supports a relatively simple set of messages. In total, SNMP defines five basic message types: GetRequest, GetNextRequest, GetResponse, SetRequest and Trap [37]. This limited message set is sufficient to enable a centralized network management system to query the status

of remotely located network devices, and in some cases, to update system variables contained within the remote devices. There is also a trap mechanism used by network devices to notify the network management system of exceptional events.

An important characteristic of SNMP that is pertinent to the design of the Mobile Agent Protocol (MAP) is the use of ASN.1 protocol representation and Basic Encoding Rules (BER) protocol encoding. For historical reasons, use of ASN.1 and BER is rare for TCP/IP protocols. Both of these OSI-based techniques are incorporated into the design of the Mobile Agent Protocol (MAP), as described in subsequent sections of this chapter.

5.2.1.3 Hyper Text Transfer Protocol (HTTP)

HTTP is an application layer protocol used on the world wide web (www) between clients and servers. It is a stateless protocol in which the underlying TCP connection is only established for the duration of each request / response round trip transaction (i.e. upon retrieval of a single web page, the connection is closed). HTTP supports a limited set of operations, including the GET command which allows clients to retrieve specified documents (e.g. web pages) from a web server.

HTTP provides several functions that are relevant to the design of the Mobile Agent Protocol, including a primitive negotiation mechanism in which the client sends a list of accepted format types (e.g. type of media, preferred language, compression technique) to the server; the documents returned by the server must adhere to these constraints. HTTP also supports a simple challenge mechanism for access control in which the server may prompt the user for additional authorization information prior to downloading documents.

Similar to other Internet-based application protocols, HTTP uses the concept of the Domain Name System (DNS) string-based representation of IP addresses. This technique is also partly relevant to mobile agent systems (e.g. user can instruct a mobile agent to visit a particular network site by specifying a 'easy to remember' character string that can be subsequently mapped to the IP address of the destination host).

5.2.1.4 Agent Transfer Protocol (ATP)

Recently, IBM's Tokyo Research Laboratory has proposed an Agent Transfer Protocol (ATP) [38] that is principally aimed at facilitating the transportation of aglets over the Internet. Aglets [39], a term that is formed by concatenating 'agent' with 'applet', are self-contained Java scripts that are capable of accumulating state information (i.e. in contrast with applets). However, aglets can not be truly classified as mobile agents, as all movement within the network must be externally initiated (e.g. manually pulled by user).

Several important ideas are captured within the ATP specification, including the specification of 'ATP' as the first element (i.e. the protocol scheme) of the Universal Resource Locator (URL). For example, to send an applet to the IBM laboratory in Tokyo using the Agent Transfer Protocol, a URL of 'atp://www.trl.ibm.co.jp' could be specified. Another important observation is the use of TCP port 434 as the default assignment for mobile agents.

Although beyond the scope of this discussion, IBM's ATP protocol is fairly limited in scope and does not provide any specific mechanisms for multimedia agents; furthermore, many aspects of the protocol are tightly coupled with the Java programming language.

5.2.2 Protocol Representation

At the outset of this thesis, a decision was made to investigate the use of Abstract Syntax Notation One (ASN.1) for specifying the agent structure and application layer protocol. ASN.1 provides a formal language for specifying protocols and is commonly used in industry for specifying protocols that are intended to be implemented by multiple development agencies. The ASN.1 standards were re-issued in 1994 and now comprise four separate specifications [32][40][41][42]. As per [32], ASN.1 is "particularly, but not exclusively, applicable to application layer protocols"; furthermore, it is stated that "application layer standards need to define quite complex data types to carry their messages, without concern for binary representation".

A key advantage of ASN.1 is that it provides the flexibility to handle all data types; that is, in addition to using a predefined set of universal data types (e.g. INTEGER), developers can define their own data types to uniquely identify protocol objects. As noted above, another advantage of ASN.1 is that the protocol can be specified independently of the format used to encode the messages for transmission.

Detailed ASN.1 specifications of the Mobile Agent Protocol (MAP) message set are included in Appendix A. A pictorial representation of each of the Mobile Agent Protocol (MAP) messages is included in subsequent sections of this chapter.

5.2.3 Protocol Encoding

As noted in the previous section, ASN.1 specifies a protocol independently from the data byte representation that is transmitted across the network. With reference to the OSI model, protocol encoding is a function of the presentation layer (i.e. layer 6). Presentation layer

functions, which correspond to the Message Encoding Sublayer of the architectural model described in Chapter 3, include encoding / decoding, message segmentation and encryption.

The message encoding of the Mobile Agent Protocol (MAP) is based on the use of Basic Encoding Rules (BER) transfer syntax. Individual MAP messages are specified using ASN.1 and encoded using the BER transfer syntax; the resulting sequence of hexidecimal values are inserted into TCP packets and subsequently delivered to the appropriate host sites.

The BER encoding scheme is based on use of Type–Length–Value triplets. Each application layer protocol field is encoded using a type indicator (e.g. hex '03' corresponds to integer type), a length indicator and a data value. The length indicator permits values of up to $2^{1008} - 1$ bytes in length (and is thus suitable for very large multimedia mobile agents). A key benefit of the Type–Length–Value approach is that the length of each message does not need to be explicitly indicated in the application layer; furthermore, it provides the flexibility to easily accommodate future changes to the length of any particular data field.

In 1994, a new set of encoding rules was released by the International Telecommunication Union [31][43]. In addition to the Basic Encoding Rules (BER), several other types of encoding are defined, including the Distinguished Encoding Rules (DER), Canonical Encoding Rules (CER) and Packed Encoding Rules (PER). Although BER provides designers with a considerable amount of flexibility and extendability, there are tradeoffs in terms of compactness [44]. For this reason, it is suggested that alternate forms of encoding be used for applications with special requirements such as bandwidth restrictions or the need to perform encoding in real time. Several examples of BER encodings of the Mobile Agent Protocol (MAP) messages are included in Appendix B.

5.3 Specification of the Mobile Agent Protocol (MAP)

5.3.1 Supported Functions

5.3.1.1 Introduction

Many of the functions to be incorporated into the Mobile Agent Protocol (MAP) have been described earlier in this thesis. The intent of this section is to summarize those functions that have been previously mentioned, and in some cases, further expand those areas requiring a more in-depth treatment. By implementing a robust set of basic functions directly in the MAP message set, application developers will be able to use these messages to support higher level functions. For example, by providing a message to check whether a mobile agent has visited a particular network site, a higher level utility can be developed for tracking mobile agents as they travel throughout the network.

5.3.1.2 Agent Transfer

The ability to transfer a mobile agent from one host to another is a fundamental requirement. In addition to providing a reliable mechanism for transferring mobile agents, special considerations are required for making efficient usage of network resources and for handling potentially large multimedia agents.

5.3.1.3 Addressing

Although TCP/IP provides the basic addressing scheme required for delivering messages to specific ports at specific host machines, a mechanism for uniquely identifying mobile agents is required. Therefore, each mobile agent acting on behalf of a particular user is assigned (by the user's application) an agent serial number. To uniquely identify a mobile agent at a particular site, the user's name is coupled with the agent serial number.

To uniquely identify the user within the network, a user identifier is specified. As described in the subsequent section on security, the user identifier will typically be administered by a trusted security agency with responsibility for registering a pair of private and public keys for each user.

5.3.1.4 Site Interrogation

The MAP message set is required to provide support for remotely interrogating a network site. For example, a protocol mechanism is required to determine which services are available at a particular site and whether a particular mobile agent has checked in to the site.

5.3.1.5 Agent Intervention

The MAP message set is required to support a mechanism for remotely communicating with a mobile agent. For example, the agent's user may wish to instruct the agent to return home immediately; alternatively, the user may wish to update the agent with new mission data.

5.3.1.6 User Intervention

The MAP message set is also required to support a mechanism for establishing communications with a remotely located user. This mechanism can be used by the mobile agent to notify the user of a significant event or to request additional instructions from the user.

5.3.1.7 File Transfer Utility

The MAP message set is required to support a file transfer utility for transferring files to a remote site and for retrieving files from a remote site. This utility can be used in lieu of encapsulating documents within the body of the mobile agent. For example, in the case of a large multimedia document discovered at a remote site, the mobile agent can return to the user

with a pointer specifying the network address of the document; in this way, the user can download the file only if it is deemed to be worthwhile.

5.3.1.8 Security

Security mechanisms are directly incorporated into the Mobile Agent Protocol (MAP), as security is a key consideration when developing a mobile agent framework that is suitable for use with a range of applications (e.g. electronic commerce over the Internet). However, the task of providing security for a mobile agent application involving any number of intermediate sites is quite complex. Although not a main focus of this thesis, an attempt is made to outline the basic security requirements of the mobile agent architecture.

Most of the applications proposed for mobile agents require some degree of security; for this reason, most mobile agent systems specify the use of public key cryptography techniques [5] [6].

Given that agents by definition perform tasks on behalf of users, network sites need to be confident that the agent is legitimately operating under the declared user's authority. Furthermore, for sensitive transactions, users and network sites need to be assured that the mobile agent has not been tampered with at any intermediate point in the network. For these reasons, authentication and integrity checking are essential considerations. Equally important is the need for privacy, as mobile agents must be capable of securely carrying sensitive information (e.g. credit card numbers for electronic commerce applications).

To provide the required security, a flexible approach using standard public key cryptography techniques is proposed. The level of security employed would typically be determined by the application developer according to the security requirements of the particular

mobile agent application (e.g. in some cases, no additional security required). From a network architecture perspective, this requires the introduction of a network security server typically administered by a trusted third party network agency. The security agency would be responsibility for the administration of private and public key pairs.

To provide authenticity and integrity checking, a digital signature (referred to as the message digest) is optionally included in each MAP message. The digital signature is calculated based on the user's private key and is applied only to the static components of the mobile agent structure (i.e. the components such as User Information that will not be updated at intermediate sites). Furthermore, for dynamic components of the mobile agent, options are supported to include digital signatures that apply only to that component. For example, a network site could authenticate a particular journal file (for subsequent validation by the user) or the script component (to be validated by the next network site, based on the premise that the previous network site is trustworthy). Upon arrival at the destination site, the authenticated components of the mobile agent can be analyzed using the public key of the authenticating entity and compared with the decrypted digital signature; if identical, the recipient can be confident that the source information has not been altered in any way.

To guarantee privacy, the source host site encrypts the selected components of the mobile agent using the recipient's public key. Upon arrival at the destination site, the private key of the recipient is used to decrypt the agent, thereby ensuring that the contents of the component remained private to all intermediate sites. This privacy mechanism could be used on a hop by hop basis between host sites to encrypt the entire message; this particular usage emphasizes the need to seek permission from a site before transferring a mobile agent. Another usage of the privacy mechanism is for a network site to encrypt a locally generated component (such as a

journal file or document) using the user's public key, such that it can only be deciphered by the user upon completion of the mission. For example, in an electronic commerce application, a network site would typically not want to make any pricing information available to a subsequent network site belonging to a competitor.

To further illustrate the use of security techniques, refer to Figure 10 for an illustrative example of the security procedures used during a mobile agent's round trip.

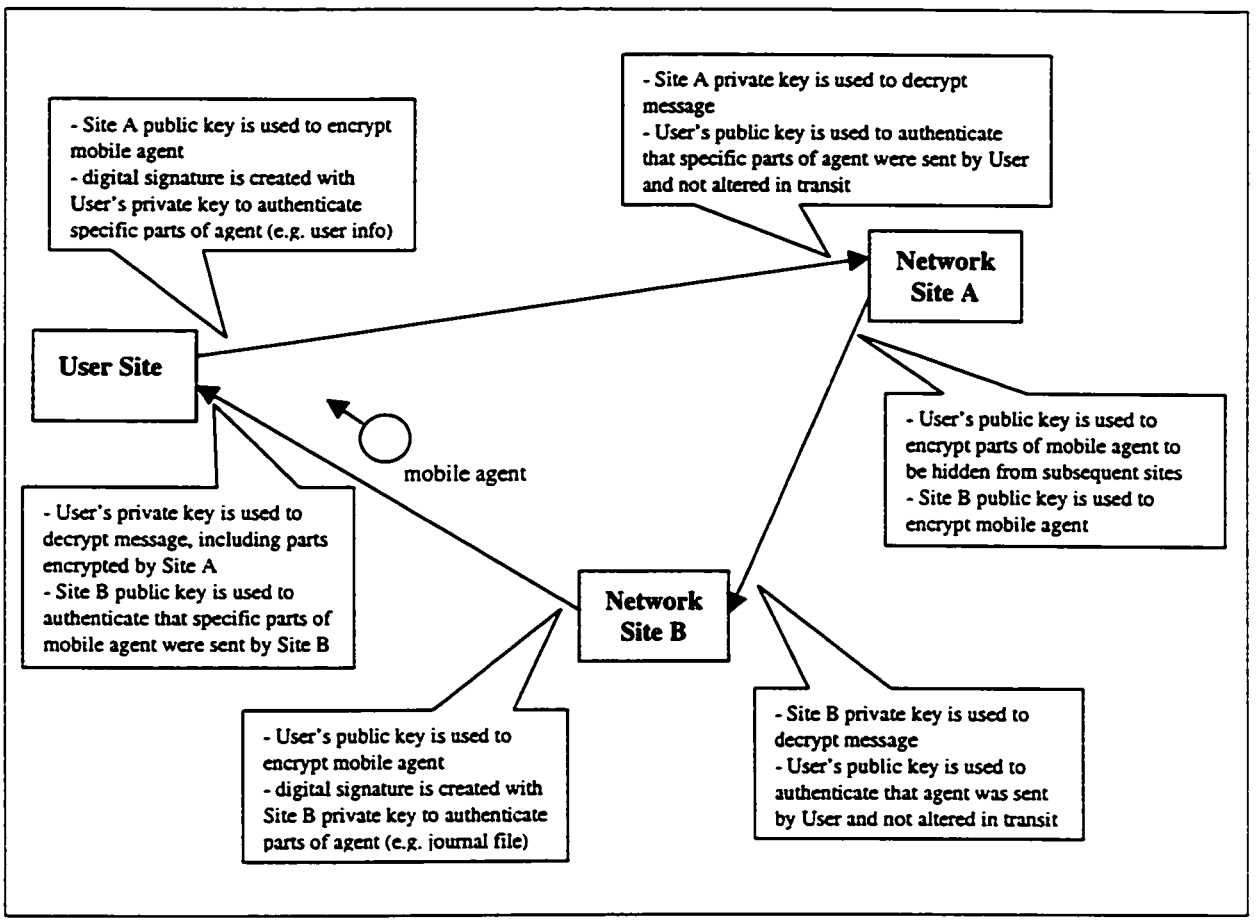


Figure 10 - Example of Security Techniques Used on Round Trip Journey

5.3.2 Defined Message Types

5.3.2.1 Overview

There are five basic message types defined for the Mobile Agent Protocol; namely, the PUSH, PULL, REQUEST, RESPONSE and ERROR messages.

The PUSH message is used to initiate a one way transfer of information to a destination site. The expression 'push technology' is frequently used when describing Internet applications such as Marimba [45] and PointCast [46] that 'push' information such as stock quotes and general news information to users who have subscribed to the service. The earliest (and most primitive) example of 'push technology' is e-mail.

The PULL message is used to obtain information from a destination host. The type of information that can be 'pulled' from a site include data registers (belonging to a predefined set of data objects) and document files.

The REQUEST message is used for high level service requests (e.g. requests for permission for a mobile agent to visit a site).

The RESPONSE message is typically sent in response to PULL and RESPONSE messages. However, if an acknowledgement message is specifically requested, a RESPONSE message will be returned following the receipt of a PUSH message.

The ERROR message is used to provide unsolicited notification of error conditions.

The general message format used by the Mobile Agent Protocol (MAP) is shown in Figure 11.

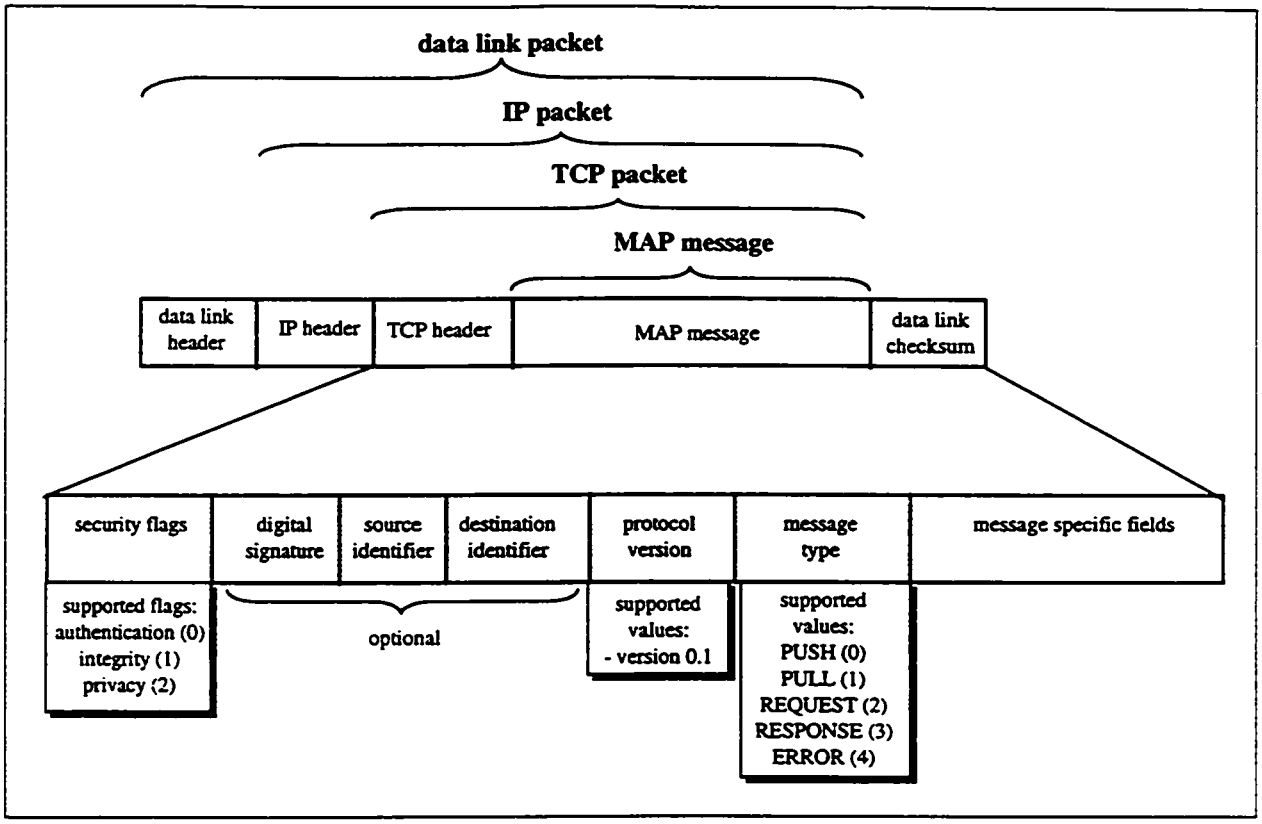


Figure 11 - General Format of MAP Message

Each message includes security flags to indicate the security options selected. For messages requiring with the privacy option and/or the authentication / integrity option selected, three additional fields are included; namely, digital signature, source identifier and destination identifier. The source and destination identifiers are assigned by the security agency to uniquely identify the sender and receiver entities; these identifiers correspond to a pair of public and private keys administered by the security agency.

A protocol version field and a message type field are also used. The protocol version field refers to the version of the Mobile Agent Protocol (i.e. version 0.1). The message type field is used to distinguish between each of the five supported message types.

5.3.2.2 PUSH Message

There are four message subtypes currently defined for the PUSH message; namely, TRANSFER_AGENT, UPDATE_AGENT, UPDATE_USER and PUT_FILE.

The TRANSFER_AGENT subtype is used to transfer a mobile agent from one site to another. Within the body of the message, there is a segment corresponding to each of the mobile agent components defined in Chapter 4. Static components (i.e. components that are set by the user and are not permitted to be changed) do not require special security fields, as their security is governed by the security fields at the start of the message. Dynamic components (i.e. components such as journal files, documents and script) are provided with additional security fields to enable network sites to individually secure sensitive components.

A diagram of the PUSH – TRANSFER_AGENT message is shown in Figure 12.

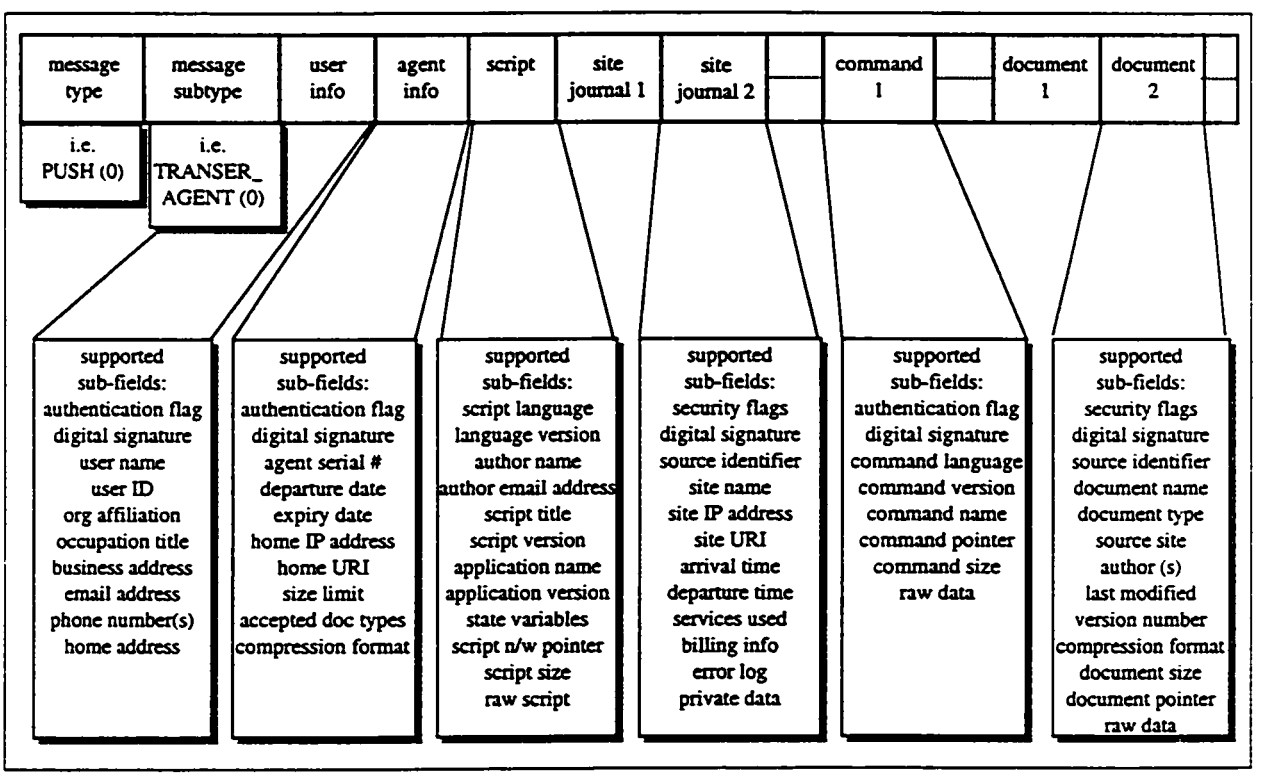


Figure 12 - PUSH: TRANSFER_AGENT Message Format

The UPDATE_AGENT subtype is used to remotely update the mission of a mobile agent. There are currently five remote operations defined; supported operations include instructing the mobile agent to return home immediately, to return home upon completion of current set of tasks (i.e. in lieu of jumping to the next network site, return to user site), to abort the mission and self-terminate, to accept new data and to reload a new script. Other supported fields include user name and agent serial number which are used to uniquely identify the mobile agent at the network site. An acknowledgement flag is also included, to specify whether the source site requires an acknowledgement that the operation was successful.

A diagram of the PUSH – UPDATE_AGENT message is shown in Figure 13.

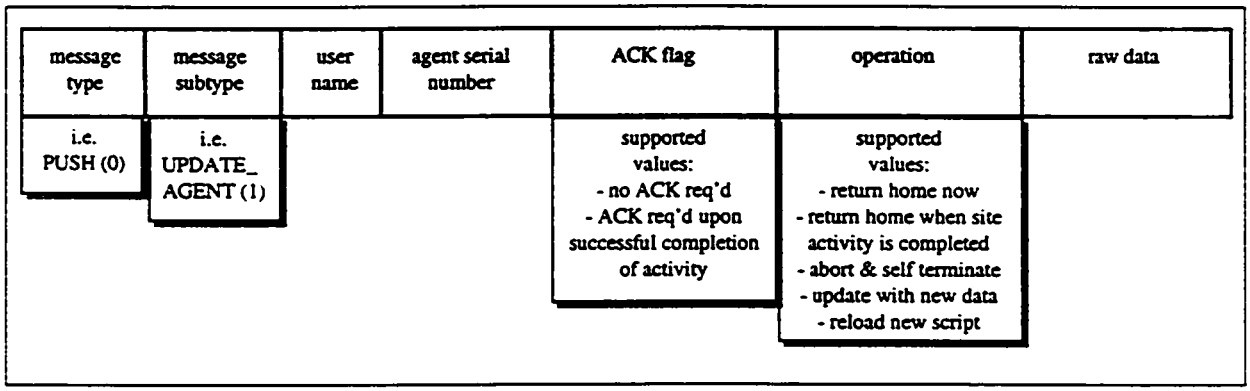


Figure 13 - PUSH: UPDATE_AGENT Message Format

The UPDATE_USER subtype is typically used by the mobile agent to update the user with intermediate results or to send an inquiry to the user to seek clarification of the mission. The user name field combined with the agent serial number is required by the user site to match the update with a particular mobile agent.

A diagram of the PUSH – UPDATE_USER message is shown in Figure 14.

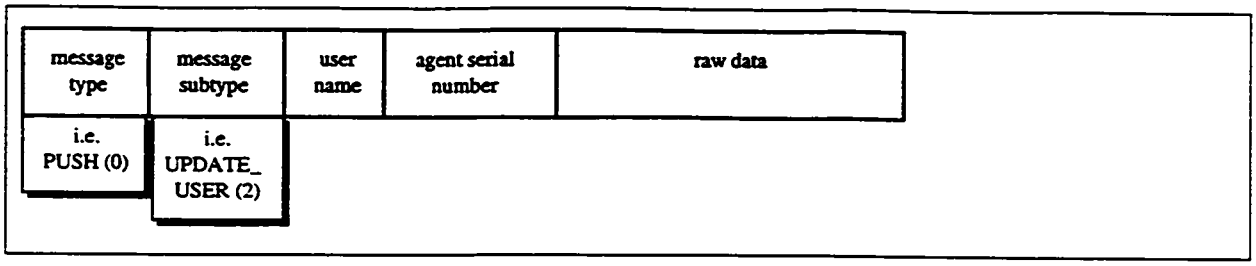


Figure 14 - PUSH: UPDATE_USER Message Format

The PUT_FILE subtype is used to transfer a file to a remote site. This message complements the PULL - GET_FILE message. In addition to specifying the file name and file path of the document to be transferred, a number of fields are supported to describe the contents of the file (e.g. size, author, compression format), consistent with the document component of the mobile agent structure defined in Chapter 4.

A diagram of the PUSH – PUT_FILE message is shown in Figure 15.

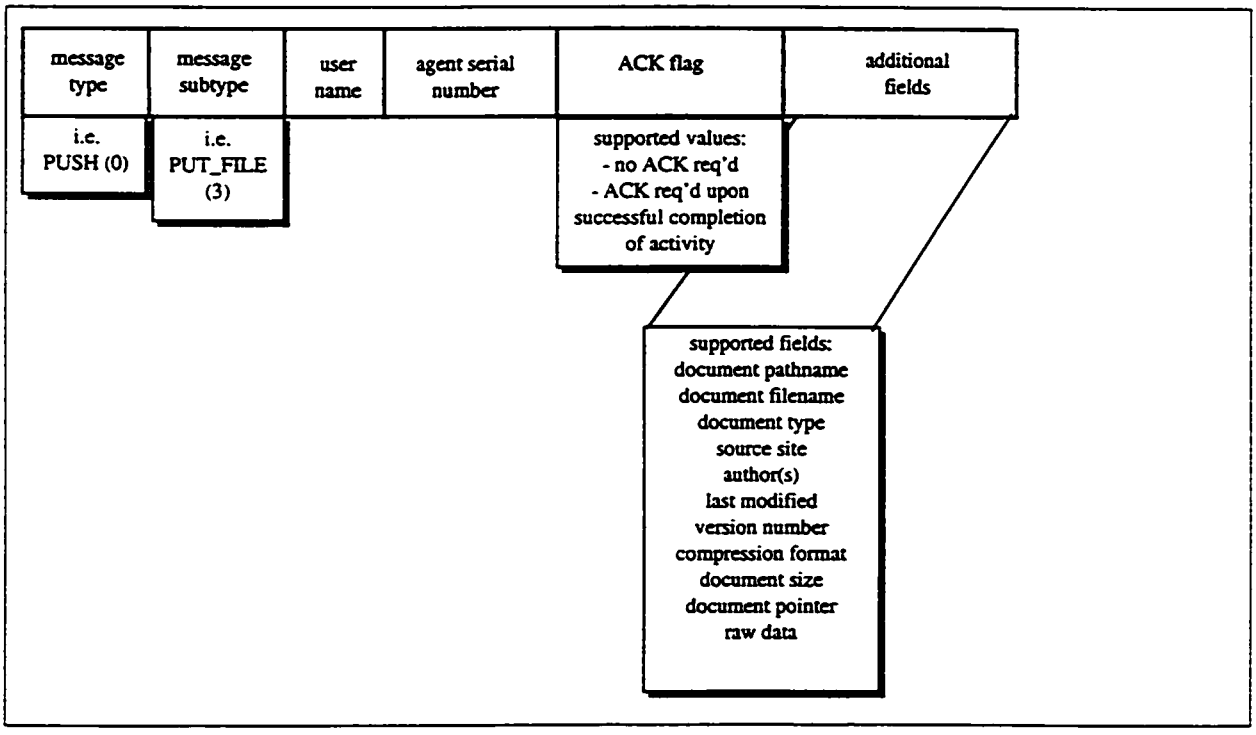


Figure 15 - PUSH: PUT_FILE Message Format

ASN.1 descriptions of each of the PUSH message subtypes are included in Appendix A.

5.3.2.3 PULL Message

There are two message subtypes currently defined for the PULL message; namely, the READ_OBJECT and the GET_FILE subtypes.

The READ_OBJECT subtype is used to read the current value of a predefined set of objects from a remote site. Currently, there are three classes of objects that are supported: host, user and mobile agent. For each object class, there is a predefined list of object identifiers supported. For example, the host class currently supports three objects describing the site name, the system status and supported services. Similar to the SNMP approach described earlier, a hierarchical object identification scheme is used; in this way, new object identifiers can be created with no impact on the messaging protocol.

In the event that a particular object is not available at the remote site, a RESPONSE message indicating an error condition is returned. For example, consider the case in which an object corresponding to a mobile agent is requested from a remote site. At the remote site, the Front Desk maintains a registry of all mobile agents that have recently visited the site. To provide a degree of security, for any request for an object belonging to the mobile agent class, the source site must specify the user name and agent serial number to be matched. In the event that there is no record of the targeted mobile agent in the registry, a response is sent from the remote site indicating that the requested object is not available.

A diagram of the PULL – READ_OBJECT message is shown in Figure 16.

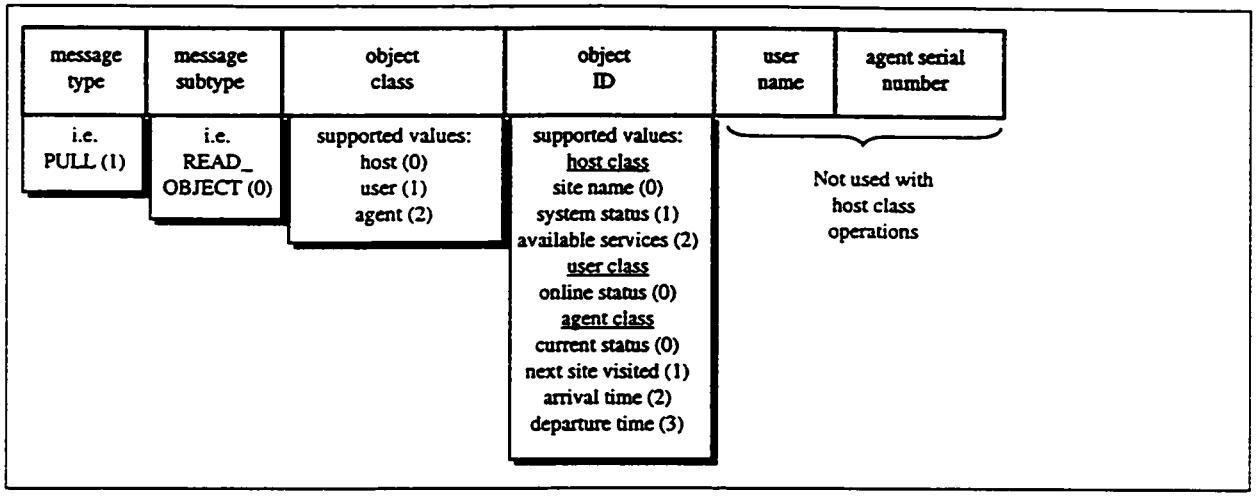


Figure 16 - PULL: READ_OBJECT Message Format

The GET_FILE subtype is used to retrieve a multimedia file from a remote site. This message complements the PUSH - PUT_FILE message described earlier. In addition to specifying the file name and file path of the document to be retrieved, two fields are included to describe the acceptable document formats and compression formats, respectively.

A diagram of the PULL – GET_FILE message is shown in Figure 17.

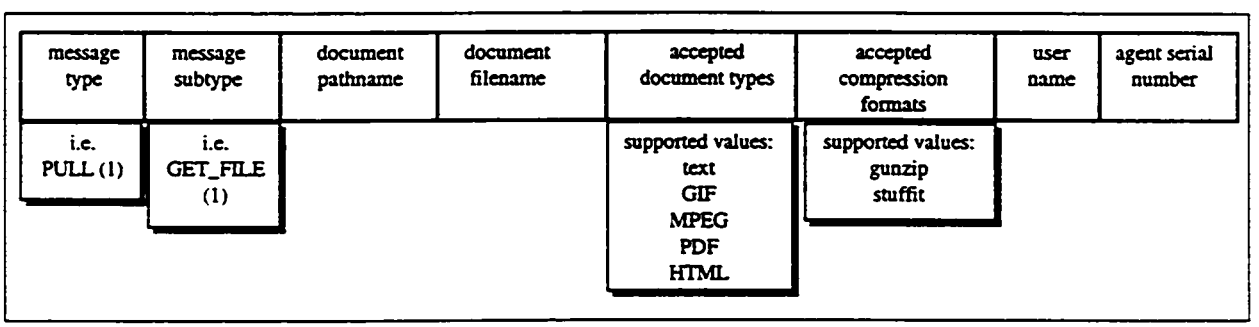


Figure 17 - PULL: GET_FILE Message Format

ASN.1 descriptions of each of the PULL message subtypes are included in Appendix A.

5.3.2.4 REQUEST Message

The REQUEST message is used to send specialized requests to the remote host site. Currently, three types of requests are supported: site entry, agent consultation and user consultation.

The site entry request is typically sent by a host site seeking to transfer a mobile agent to a remote site. In addition to specifying the user's name and identifier, the site entry request indicates the service required (e.g. Multimedia Document Retrieval) and the security requirements (e.g. encryption required). The response from the remote site indicates whether the mobile agent will be permitted entry.

The agent consultation request is typically sent to a remotely located mobile agent prior to sending a PUSH – UPDATE_AGENT message. If granting the request, the mobile agent will remain at the remote site awaiting an update.

The user consultation request may be sent to a remotely located user prior to sending a PUSH – UPDATE_USER message. For example, if a mobile agent is seeking further instruction from the user to clarify the mission, the user consultation request can be used to ensure the user is available. In cases where mobile agent needs only to inform the user of a particular event, simply sending a PUSH – UPDATE_USER message will suffice. In addition to specifying the user name, the serial number of the mobile agent is included to enable the user to determine which mobile agent is initiating the request.

A diagram of each of the supported variants of the REQUEST message is shown in Figure 18.

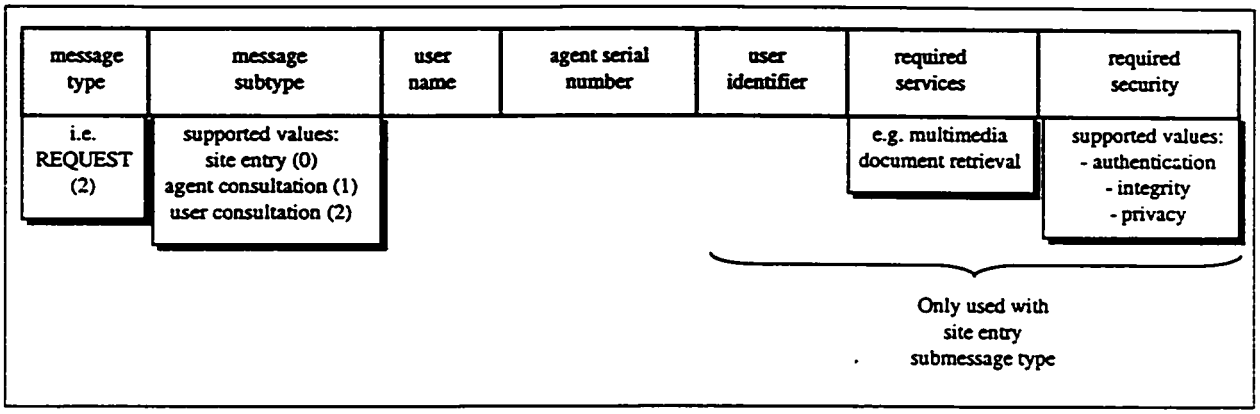


Figure 18 - REQUEST Message Format

5.3.2.5 RESPONSE Message

The RESPONSE message is returned following the receipt of a PULL or REQUEST message; it may also be used if an acknowledgement is specifically requested by a PUSH message.

In addition to specifying the standard set of header fields (e.g. security marker, protocol version), the RESPONSE message specifies the message type and subtype that prompted the response message. As shown in Figure 19, there are several format variants defined for the RESPONSE message.

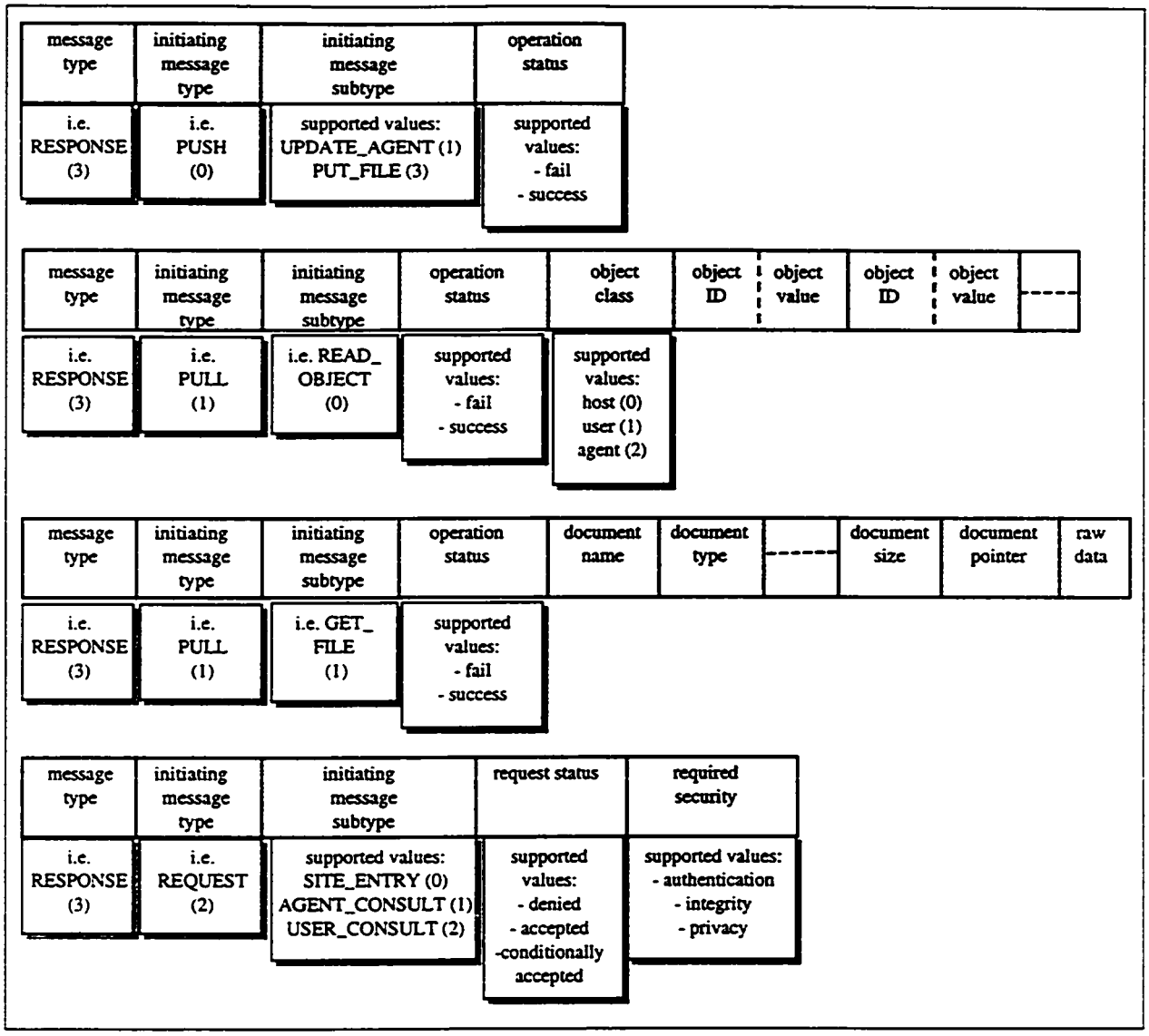


Figure 19 - RESPONSE Message Formats

5.3.2.6 ERROR Message

The ERROR message provides an unsolicited notification of a general failure condition. For example, if the decryption of a mobile agent fails, an ERROR message will be sent to the source host. A diagram of the ERROR message is shown in Figure 20.

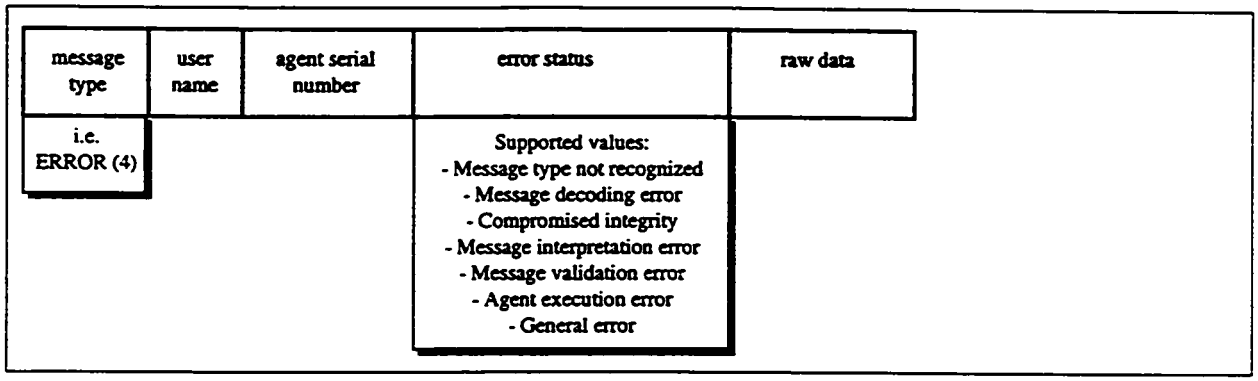


Figure 20 - ERROR Message Format

Chapter 6

6 System Implementation

6.1 Introduction

To validate the system design developed in this thesis, a prototype system was implemented in the University of Ottawa's Multimedia Information Research Laboratory (MIRL). The prototype successfully demonstrated a number of key concepts, including the validation of the system architecture, the mobile agent structure and the Mobile Agent Protocol (MAP).

The User Site and the Network Site system architectures, as described in Chapter 3, were successfully implemented. For example, the Front Desk entity was used to permit site entry only to pre-registered users. The layered protocol model developed in Chapter 3 was also implemented; within the software code, a set of procedures was explicitly associated with each of the four layers (i.e. Connection Management Layer, Message Encoding Layer, Message Validation Layer and Message Interpretation Layer). This layered model provided a convenient mechanism for separating functions into logical layers; in this way, software changes made in one layer of the application layer protocol stack are transparent to functions in the other layers.

The mobile agent structure defined in Chapter 4 was demonstrated to provide a convenient mechanism for encapsulating multimedia information. Mobile agents were used to collect a range of document types from network sites within the MIRL, including text documents, GIF files and large MPEG files.

The Mobile Agent Protocol (MAP) described in Chapter 5 was successfully implemented on a TCP/IP protocol stack. However, as noted in the subsequent section on suggested enhancements, implementation of all aspects of the protocol (e.g. optional digital signature

fields) was left as a future activity beyond the scope of this thesis. Furthermore, not all of the defined message subtypes were implemented owing to time considerations.

6.2 Prototype System

6.2.1 Overview

This section describes the prototype of the mobile agent-based Multimedia Document Retrieval application that was developed at MIRL, as shown in Figure 21. In addition to presenting a high level architectural diagram of the prototype, details are provided of the software implementation in the form of execution threads.

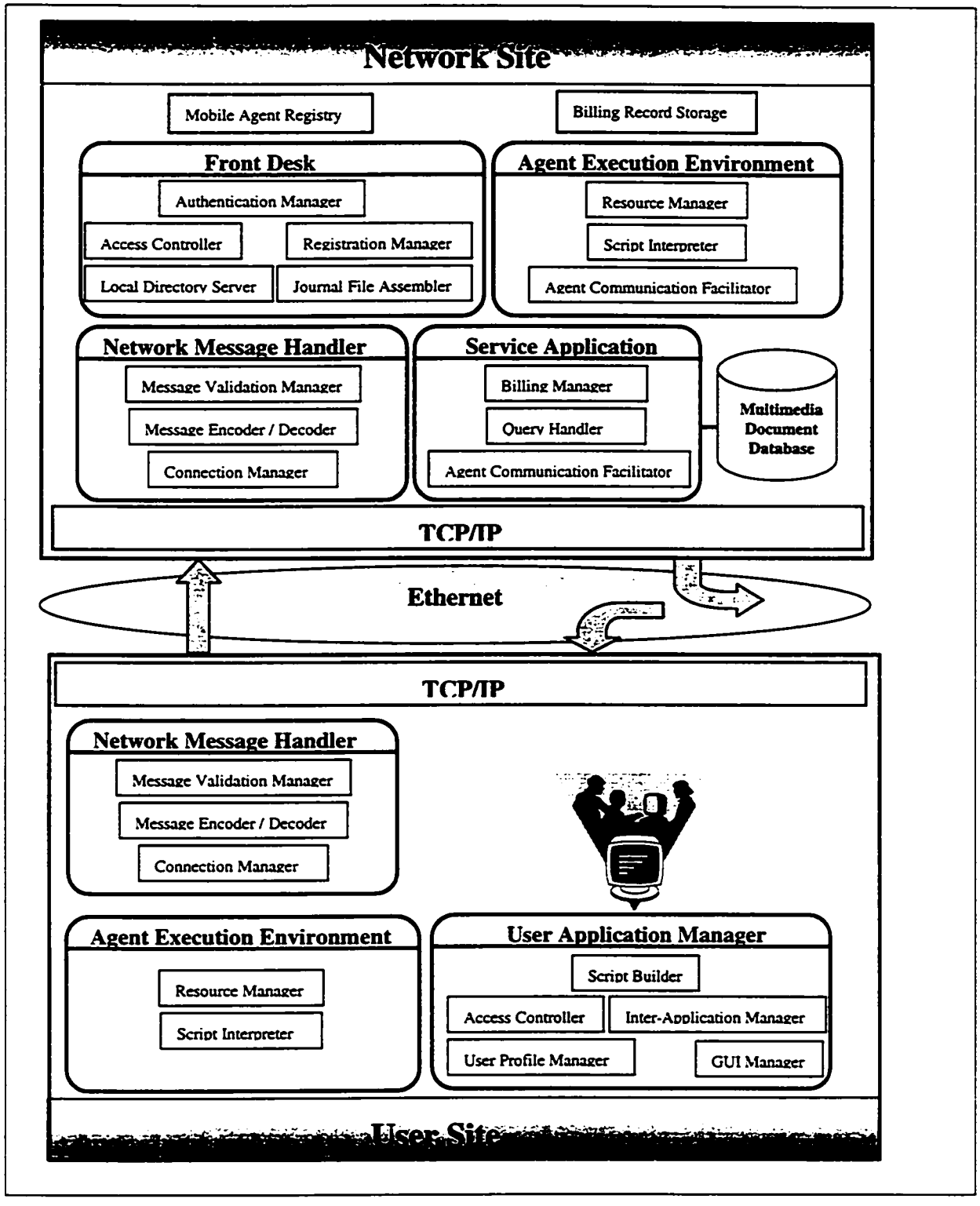


Figure 21 – Prototype System Architecture

6.2.2 Software Flow Diagrams

6.2.2.1 Introduction

Based on the architectural model described in Chapter 3, this section provides a high level pictorial representation of the three principal execution threads. The notation used to describe the program execution flow is based on the concept of timethreads [47], which provides a convenient form for recording software flow scenarios. For ease of understanding, only the success path threads are shown in Figure 22; each of these threads will be described in more detail in subsequent sections.

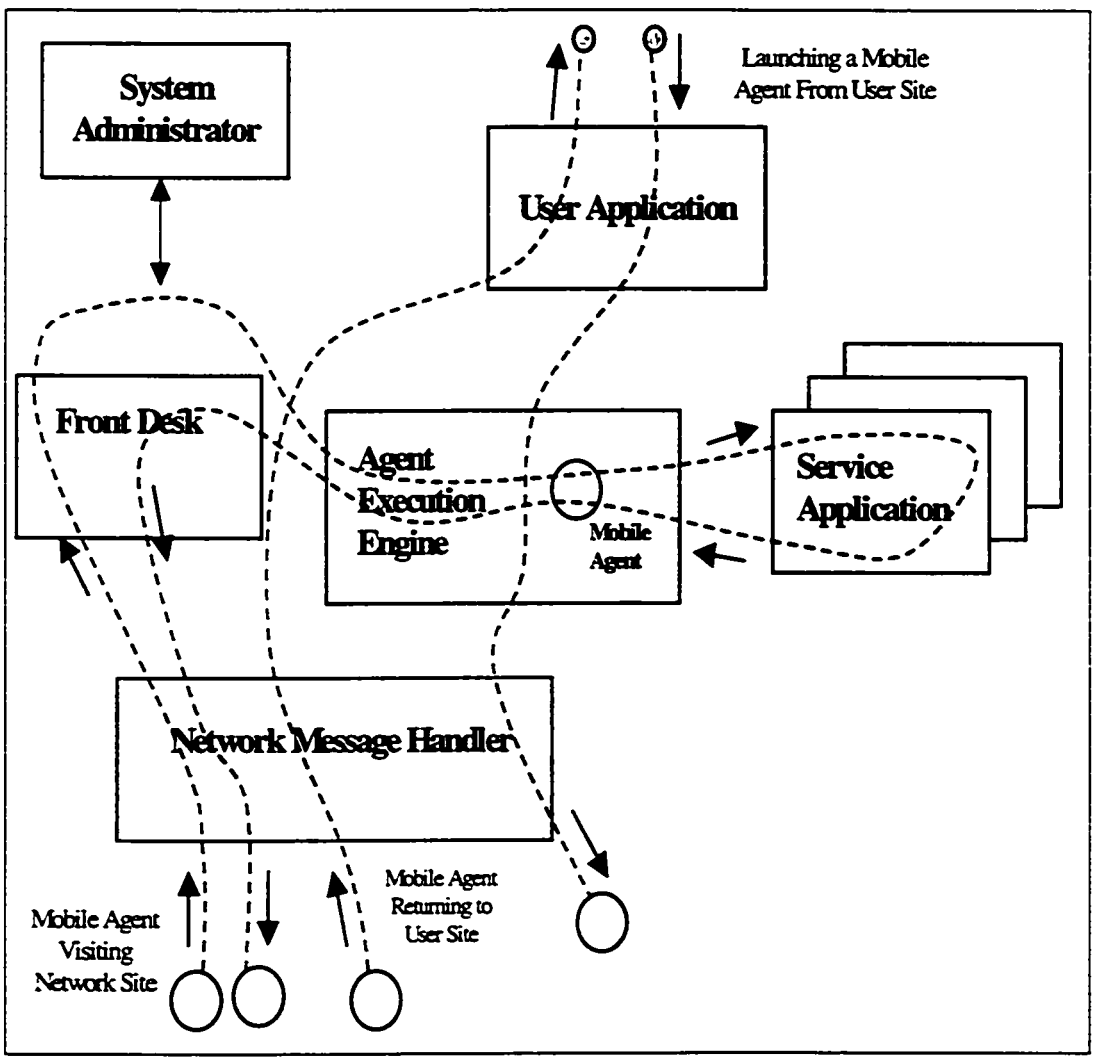


Figure 22 - Principal Execution Threads

6.2.2.2 *Interpretation of the Flow Diagrams*

This section provides a pictorial representation of the Tcl/Tk procedures that are invoked as part of the success path for the three principal execution threads described earlier. In addition to specifying name and function of each procedure, the architectural entity with which each procedure is associated (e.g. Front Desk, Network Message Handler, etc.) is indicated. In cases in which the procedure call is triggered by an external event, a brief description of the event is included.

Only the main Tcl/Tk procedures are shown, to improve the readability of the flow diagrams. For example, the procedure initiate_transfer is shown as a single procedure; in reality, initiate_transfer calls several other procedures that are not shown, such as encode_data and send_data. Furthermore, to overcome limitations of Tcl/Tk in handling binary data, C++ program bin2text.exe is invoked by the initiate_transfer procedure for performing file conversion on binary file types.

6.2.2.3 *Launching a Mobile Agent from User Site*

Figure 23 depicts the success path software flow for the scenario in which a mobile agent is launched from the user site.

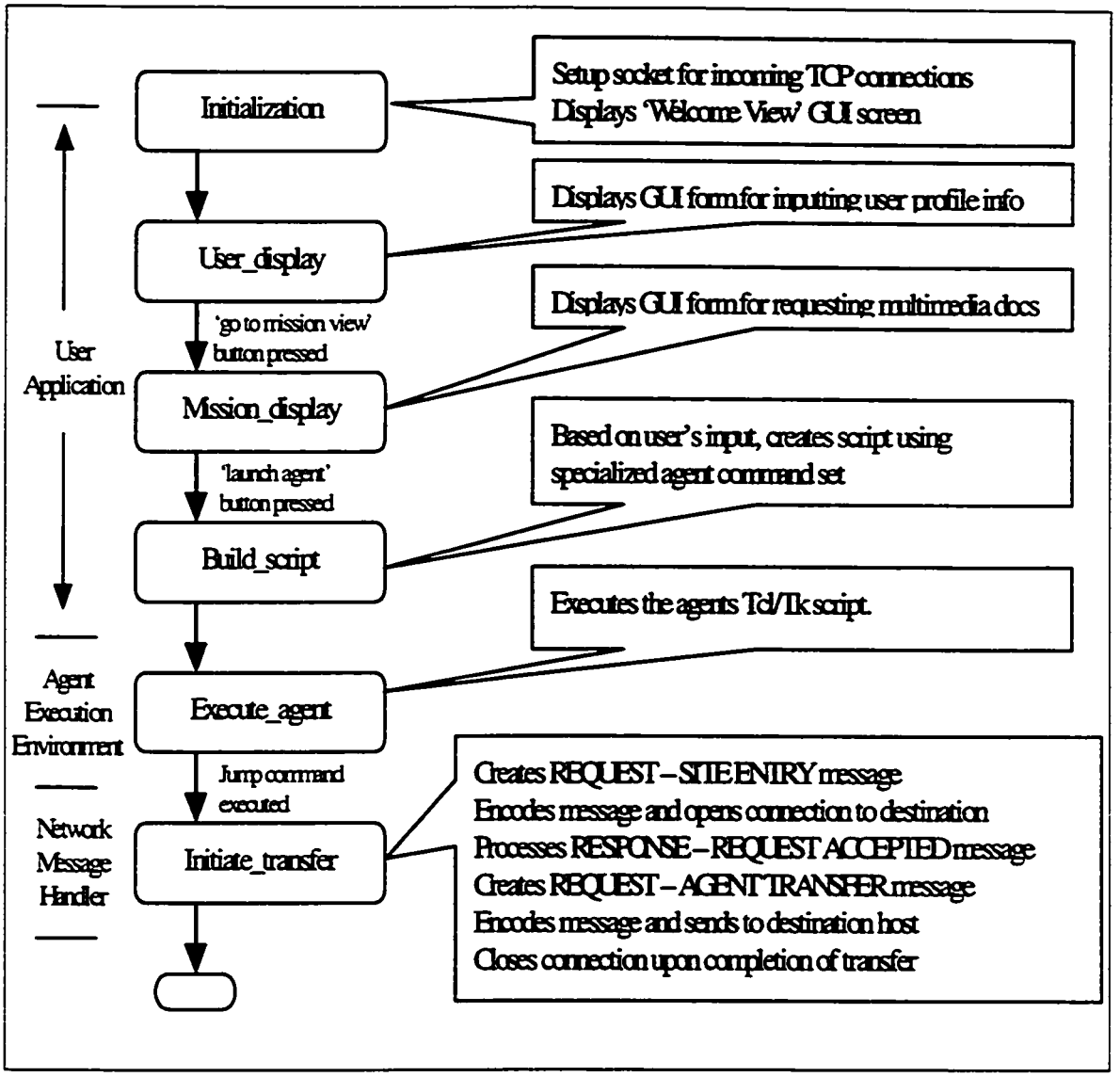


Figure 23 - Software Flow Diagram for Mobile Agent Launched From User Site

6.2.2.4 Mobile Agent Visiting a Network Site

Figure 24 depicts the success path software flow for the scenario in which a mobile agent visits a network site. As noted below, through the use of Tcl file events, a process is created each time a new incoming socket is opened. In this way, multiple simultaneous processes can be established to enable multiple mobile agents to concurrently reside at the network site.

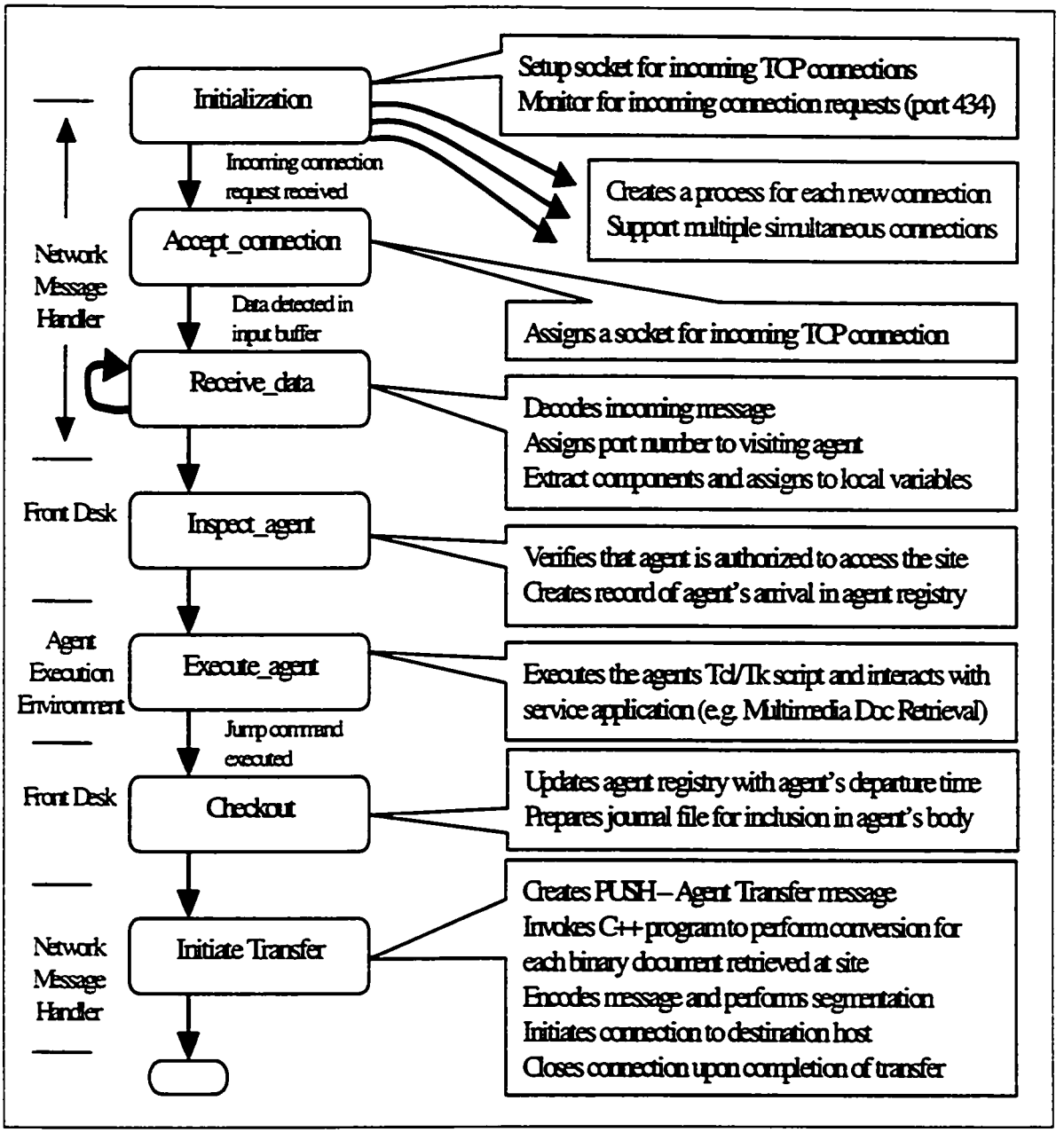


Figure 24 - Software Flow Diagram for Mobile Agent Visiting a Network Site

6.2.2.5 Mobile Agent Returning to User Site

Figure 25 depicts the success path software flow for the scenario in which a mobile agent returns to the user's site with multimedia documentation.

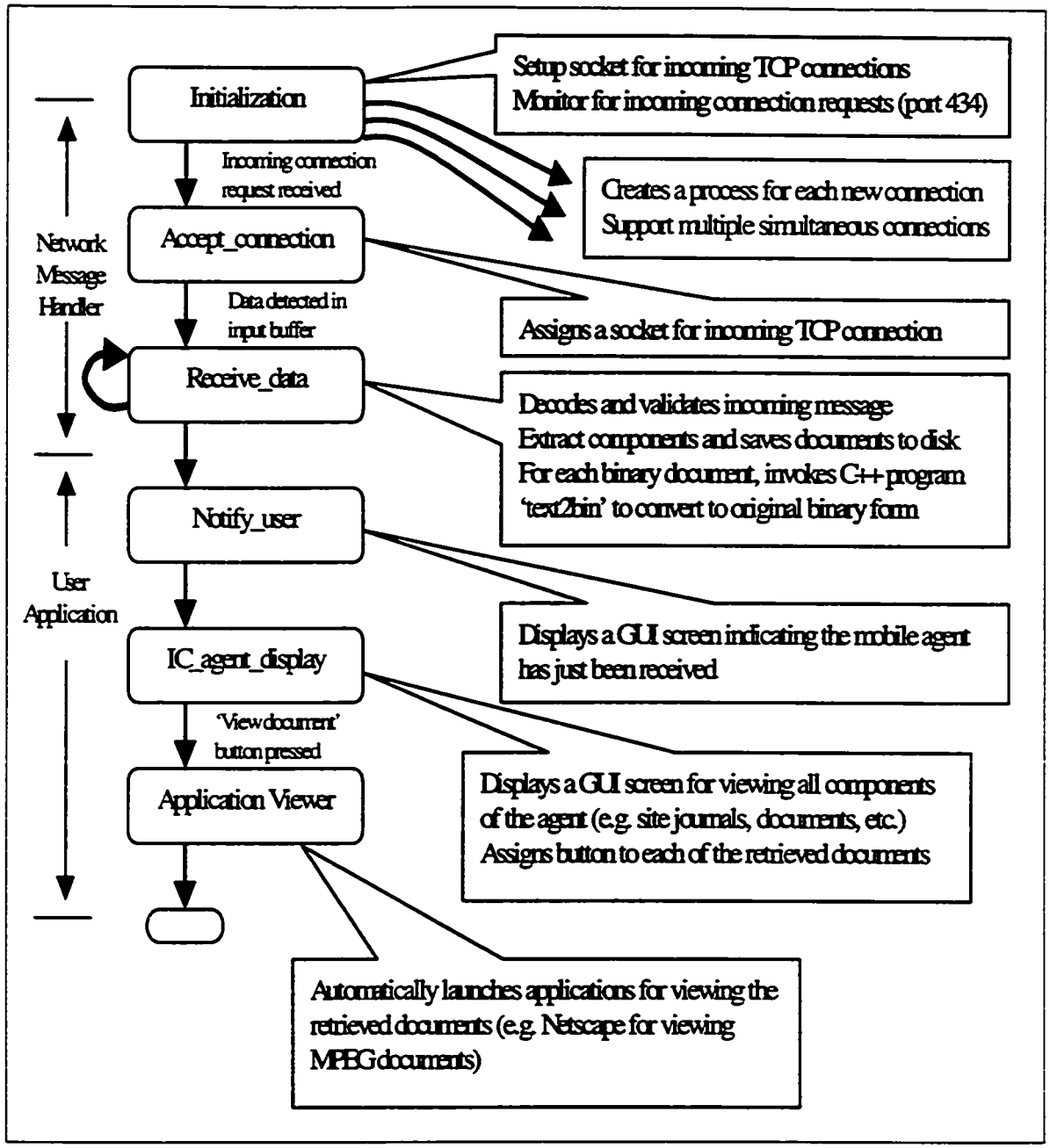


Figure 25 - Software Flow Diagram for Mobile Agent Returning to User Site

6.2.3 TCP/IP Communications

The Mobile Agent Protocol (MAP) was implemented on a TCP/IP protocol stack based on the use of sockets, as Tcl/Tk uses sockets to provide the programming interface to the TCP/IP stack. TCP was selected as the transport layer protocol for the mobile agent system as it provides

a reliable delivery service between two hosts. TCP connections were always established immediately prior to transferring MAP messages between hosts; upon completion of the messaging dialogue, the TCP connections were closed. All MAP messages were addressed to TCP port address 434 at the destination host, where port 434 is defined in RFC 1700 [35] as the "well known address" for mobile IP agents. In this way, destination host machines and intermediate routers can easily determine whether the TCP connection is being used for mobile agent-based applications.

All sockets were configured as non-blocking, such that upon arrival of data in the input buffer of an incoming socket, a Tcl event was automatically triggered (i.e. when the incoming socket became readable) and a specialized procedure was invoked to process the data. For large messages (e.g. a PUSH – AGENT TRANSFER message containing a mobile agent with a payload of one or more MPEG files), multiple events were triggered. The use of non-blocking sockets provided a convenient utility for handling multiple simultaneous connections.

6.2.4 Internal Addressing

To uniquely identify mobile agents within a particular network site, an internal addressing scheme was used. This internal address, referred to as the agent port number, is generated upon arrival of an incoming agent using a random number generator; to statistically guarantee uniqueness, a 14 digit address is used.

6.2.5 Protocol Encoding

Although the framework design specifies the use of Basic Encoding Rules (BER) for encoding the ASN.1 representation of the Mobile Agent Protocol (MAP), this approach was not deemed necessary for the Tcl implementation. BER uses a Type-Length-Value mechanism for encoding protocol messages; the advantage of this approach is that the software program does not

need to hardcode the size of the protocol fields which provides flexibility in handling future protocol extensions. However, through built-in support of commands for manipulating lists, Tcl can easily distinguish between fields based on Tcl string delimiters. This enables each value to have an arbitrary length, similar to BER, but eliminates the need to keep track of the length of each data field. However, consistent with BER, it is still necessary to keep track of the total length of each message; for this reason, the first field of each message is a message length indicator.

6.3 Programming Language

6.3.1 Programming Language Selection

6.3.1.1 Tcl/Tk

The Multimedia Document Retrieval application was programmed using release 8.0a2 of Tool Command Language / Tool Kit (Tcl/Tk) [7] [48] and release 4.51 of Borland C++. The application was executed in a networked PC environment running Windows 95. The majority of the program was written using Tcl/Tk; however, in cases where Tcl/Tk was not suitable (e.g. for handling binary data), Borland C++ was used.

Tcl/Tk is a freely distributed interpreted language that provides programmers with a large set of reusable commands; Tcl/Tk is supported on a variety of hardware platforms, including UNIX, Windows and Macintosh. Tcl/Tk programs, referred to as scripts, must be executed using a Tcl interpreter. As Tcl is a string-based command language, many commands are available for manipulating strings. Furthermore, Tcl allows application-specific extensions to be directly incorporated into the interpreter. The Tk command set provides explicit support for the development of GUIs; as such, Tk was used to create all of the GUI screens developed for the Multimedia Document Retrieval application.

Tcl uses a very straight forward syntax and thus is relatively easy to learn. For example, variables in Tcl do not need to be explicitly declared as integers or strings; Tcl determines the type of variable based on the context in which it is first used. Tcl scripts tend to be fairly compact; there have been reports of a tenfold reduction in program size compared with C. And finally, due to its interpreted nature, Tcl does not need to be compiled and thus greatly improves the speed of debugging.

The Multimedia Document Retrieval application was implemented using Tcl for many of the reasons outlined above (e.g. simplicity of use, widespread support across a range of hardware platforms). Tcl/Tk scripts are well-suited for encapsulation within a mobile agent; upon extraction of the script at the destination network site, the script can be readily executed using a local Tcl interpreter. The Tcl interpreter also includes a security mechanism that can be used to limit the commands that can be executed by a particular agent, which is an important consideration for any mobile agent-based system. Tcl/Tk has been previously used in the development of prototype mobile agent-based systems, as described in [4] and [28].

6.3.1.2 Java

Although Tcl/Tk was selected for the implementation of the Multimedia Document Retrieval application, Java's rising popularity amongst application developers and the promise of ubiquitous support across a wide range of network devices makes it a logical candidate for any mobile agent prototypes developed in the future. The intent of this section is to provide a brief description of the key properties of the Java programming language.

To qualify as a suitable programming language for mobile agents, the properties of portability and security need to be supported. Portability refers to the ability to execute a given software program on a variety of different computing devices. Security refers to the ability to

safely execute software code in an isolated fashion so as to prevent the code from directly accessing the underlying system resources, which generally implies the use of an interpreted language. Java supports both of these properties.

Java source code is compiled into machine-independent low level binary code referred to as 'byte-code'. Byte-code is similar to the native machine code of a microprocessor and can be readily transferred across a network and executed on any device that supports the Java Virtual Machine (JVM). The Java Virtual Machine, which is essentially a software implementation of a microprocessor, is used to safely interpret the Java byte-codes. The use of a low level binary code results in optimized performance compared with traditional scripting language, as it is often the case that script-based languages are characterized by inefficient performance (e.g. Tcl performance was previously estimated to be slower than C by a factor of 10,000 as referenced in [4], although recent versions of Tcl have made some progress in addressing this issue).

Java is widely used in web browsers such as Netscape Navigator to safely execute applets downloaded from web sites. Through the use of security levels, the execution of applets can be carefully controlled by restricting the applets' ability to access to the underlying file system and to initiate network connections.

Another important feature of Java is the overall efficiency of software development through the use of Object Oriented (OO) program concepts. As per [49], Java can be considered as a 99% pure Object Oriented (OO) language. All program code and data is contained within objects and classes, with the exception of simple types such as integers and characters which are implemented outside of the object system for reasons of efficiency.

6.3.2 Observations

Although Tcl/Tk does not provide support for Object Oriented (OO) concepts, it was generally found to be an effective language for implementing the mobile agent framework. It was relatively easy to learn, and complex procedures were written with very few lines of code. The built-in support for GUI commands was also very beneficial. However, as described in the following section, there are several programming language deficiencies with release 8.0 of Tcl/Tk that required the use of cumbersome workaround solutions. By far, the most significant deficiency is the inability of Tcl/Tk to support binary data; binary data is used extensively by media forms such as images, audio and video. Recently, Sun Microsystems have announced plans to enhance Tcl/Tk "so that binary data will be a first-class citizen in the Tcl world" [50].

A key challenge during the programming phase of this thesis was to devise a method for handling binary data, given Tcl's shortcomings in this area. A significant amount of effort was expended experimenting with existing Tcl commands (e.g. the 'binary' option available in 'fconfigure') searching for a possible solution; the available literature on Tcl/Tk makes no reference to this shortcoming. After concluding that Tcl could not handle any form of binary data, Borland C++ was used to write two data conversion programs. Prior to encapsulating a binary object (e.g. a MPEG file) within the body of a mobile agent, the Tcl program calls a C procedure to convert the raw binary data into a text string. This has the effect of doubling the size of the encapsulated file (e.g. the single byte hexidecimal 'f2' representation of binary data is converted to the two byte ASCII string representation 'f' '2'). Upon arrival at the user's site, the multimedia information is automatically reconverted from text to the original binary form by a second data conversion program.

Another obstacle encountered during the implementation of the application related to the transfer of large files. It was observed that a mobile agent containing a large MPEG file (e.g. > 2 M bytes) could not be written to the output buffer of the TCP/IP socket; any attempts to do this resulted in transmission failure. An approach was devised to segment the message containing the outgoing mobile agent (i.e. the PUSH – TRANSFER AGENT message) into fixed size segments (e.g. 10,000 bytes). Special provisions were required at the destination host to recognize (and discard) the End Of File (EOF) characters that were automatically appended by the source host at the end of each segment. Furthermore, it was observed that the fixed size segment sent by the source site did not reliably arrive at the destination site's input buffer as a fixed length segment (e.g. two consecutive 10,000 byte segments sent by source host might be received as three segments with lengths of 8914 bytes, 8207 bytes and 2879 bytes, respectively). This added to the complexity of the task of locating and deleting the extraneous EOF characters.

6.4 Multimedia Document Retrieval Application

6.4.1 Introduction

As previously stated, the purpose of the implementation was to demonstrate the validity of the mobile agent framework described in this thesis. Given the objective of accommodating a range of different applications, it was necessary to select an application that would exercise all of the key aspects of the framework design. A set of criteria was developed to aid in the selection of a suitable application; the list of criteria included the ability to support multimedia data, the ability to operate in a multi-site environment and the ability to utilize all of the Mobile Agent Protocol (MAP) message types.

Based on these criteria, the Multimedia Document Retrieval application was selected. In this application, the user identifies a list of multimedia documents to be retrieved by the mobile

agent. The mobile agent visits a number of sites within the network and returns to the user with the requested documents. As part of the user notification process, the agent provides the user with the option of immediately viewing the retrieved multimedia documents by launching the appropriate applications.

6.4.2 Initialization

The first time the Multimedia Document Retrieval application is used, the user inputs user profile information (e.g. name of user, organization affiliation, etc.). After entering this information, the application prompts the user to save the profile to disk. Each subsequent time the application is used, the user can quickly retrieve the stored information from disk. A screen capture of the User Profile GUI is shown in Figure 26.

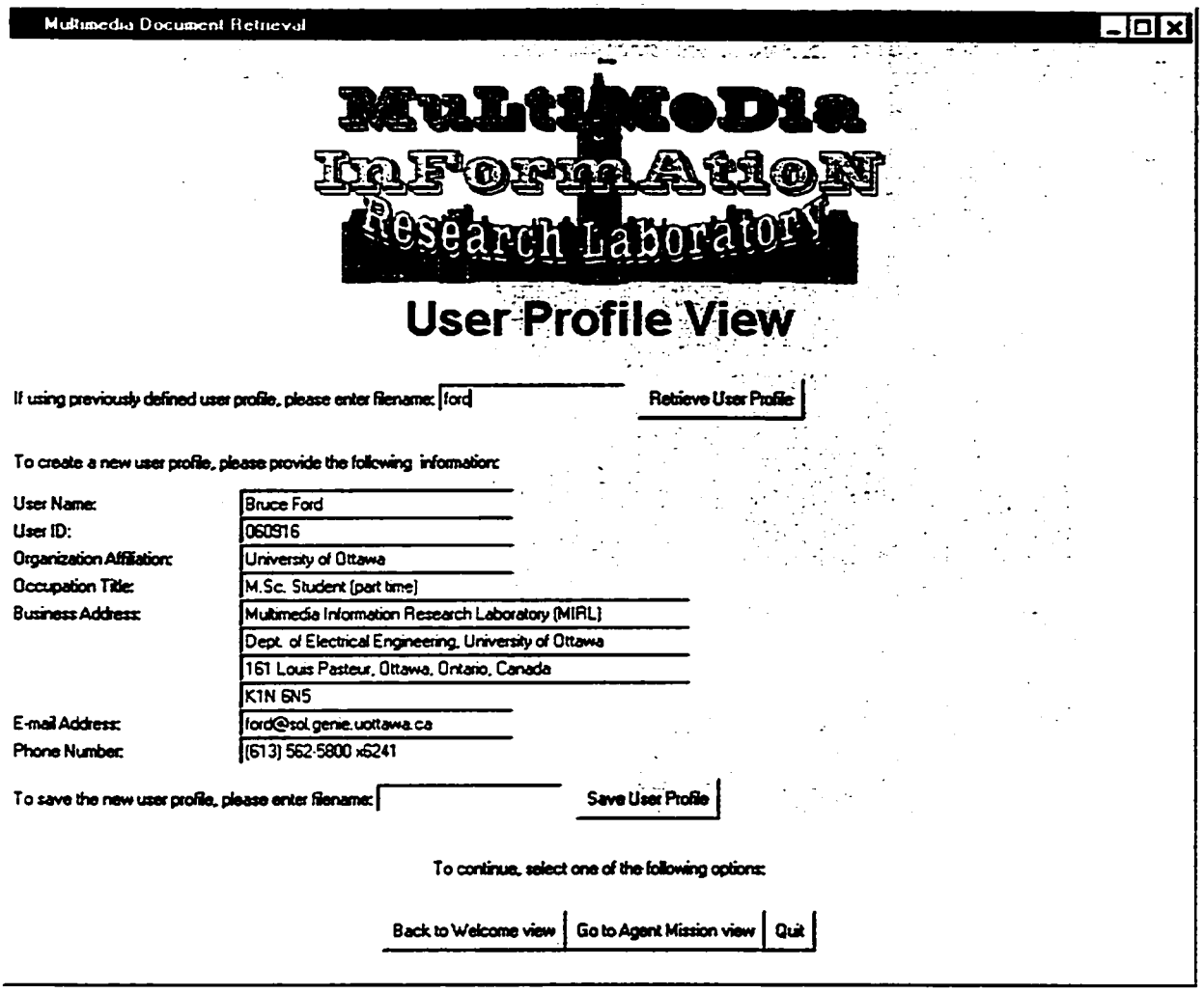


Figure 26 - GUI Screen of User Profile

6.4.3 Launching a Mobile Agent

Prior to launching a mobile agent, the user inputs a list of requested documents. This information is captured within the mobile agent's script; the mobile agent subsequently travels through the network searching for these documents. When the user has completed entering this form, the "Launch" button is selected to launch the mobile agent onto the network. A screen capture of the Document Request GUI is shown in Figure 27.

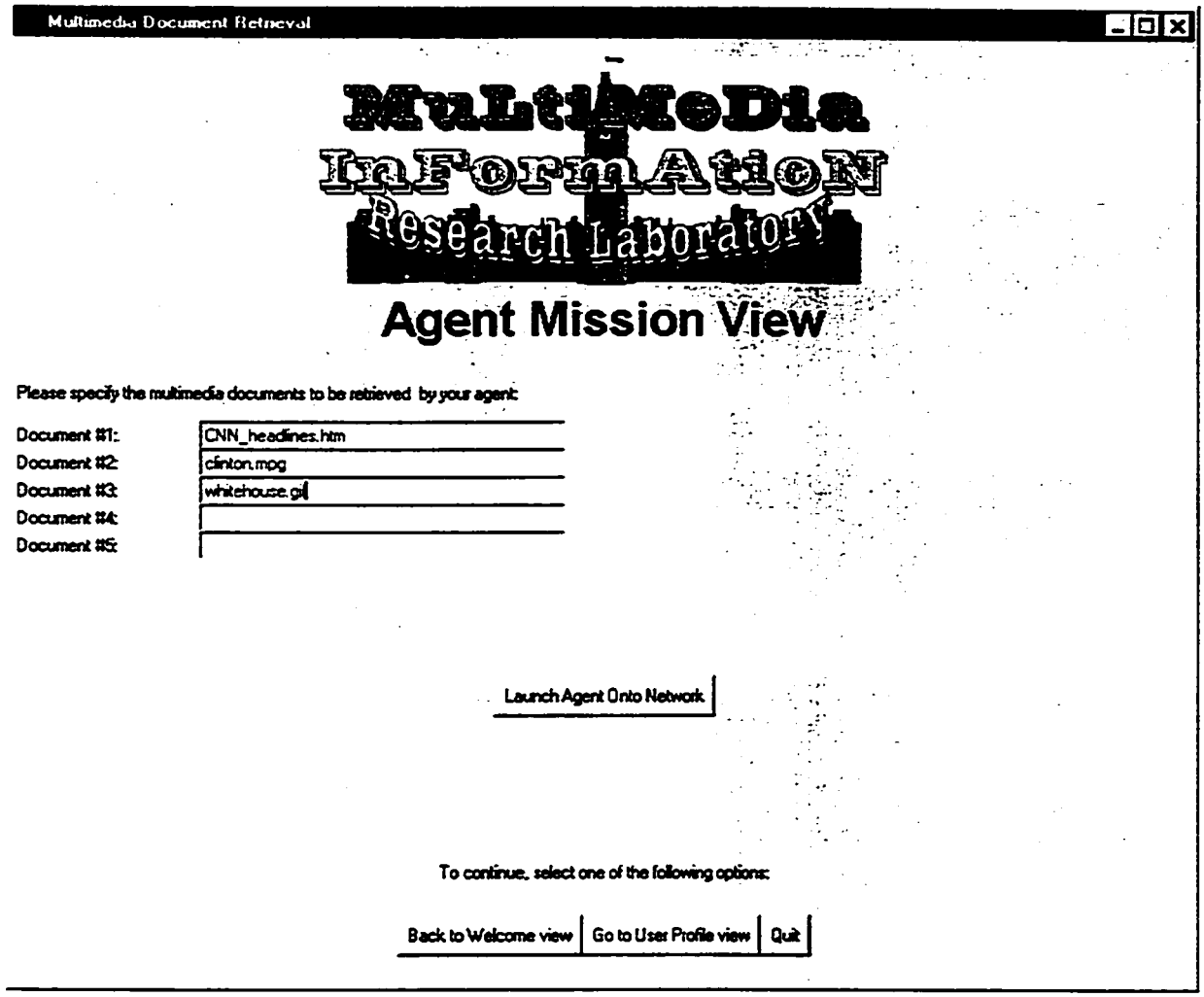


Figure 27 - GUI Screen for Document Request

6.4.4 Travelling Through the Network

Once launched, the agent travels throughout the network searching for the requested documents; at present, the agent's travels are restricted to five computing devices running Windows 95 within the University of Ottawa's Multimedia Information Research Laboratory. At each site, the credentials of the incoming mobile agent are validated by the Front Desk entity prior to execution of the agent's script. When a requested document is found, it is stored within the Document Library component of the mobile agent structure. Upon departure from each network site, a site journal is created and stored within the mobile agent's structure.

The agent travels from host to host within the network searching for the requested documents. Upon locating all of the documents or upon visiting each site (whichever occurs first), the agent returns home to the user's site.

6.4.5 Returning to the User Site

Upon arrival at the user's site, a GUI screen is used to inform the user that the agent has returned, as shown in Figure 28. To view a particular document, the user selects the appropriate button and the document is automatically opened by the corresponding viewer application (e.g. Microsoft Internet Explorer). To date, the Multimedia Document Retrieval application has been validated using text, HTML, GIF and MPEG file formats.

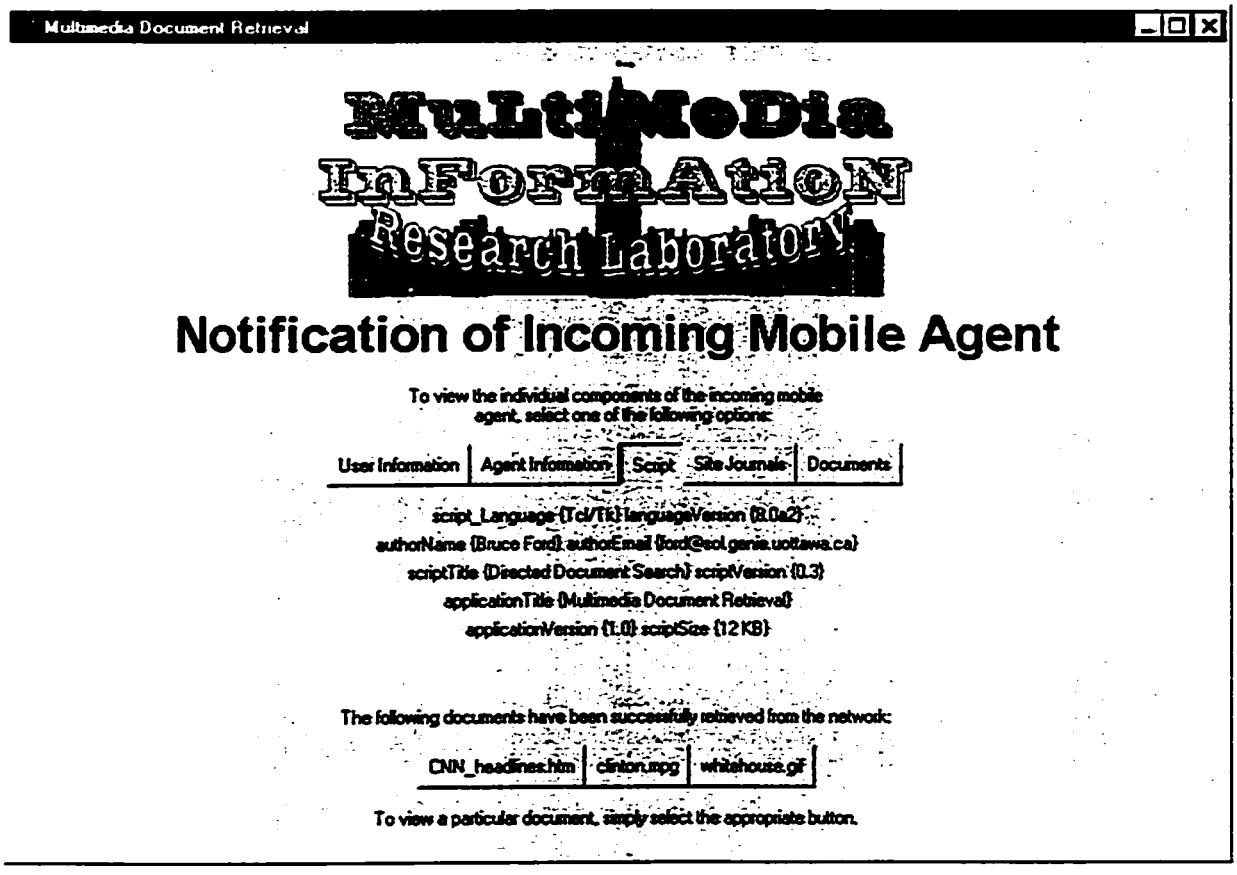


Figure 28 - GUI Screen for Incoming Mobile Agent

Chapter 7

7 Conclusions

7.1 Summary

In summary, the architectural model developed in this thesis provides a robust and secure environment for the development of mobile agent-based multimedia applications. This was successfully demonstrated through the implementation of a prototype system in the University of Ottawa's Multimedia Information Research Laboratory (MIRL).

The architecture of the user site and network site was based on a set of architectural building blocks. By developing a core set of architectural components that were common to both the user site and the network (e.g. Network Message Handler, Agent Execution Environment), it was demonstrated that a considerable amount of software could be re-used. With respect to the overall network design, it was found that the network site architecture was sufficiently flexible to allow specialized network servers, such as the directory server for mobile agents, to be implemented using a common architectural model. Furthermore, the use of a security server provided the cornerstone for a secure, mobile agent environment.

The specification of an application-independent structure for mobile agents was found to be beneficial during the development of the Multimedia Document Retrieval application. Specifically, it greatly simplified the task of developing the prototype application, as basic functions such as script encapsulation, user identification and multimedia document support are directly supported by the framework. Furthermore, the use of a site journal component provided a convenient mechanism for enabling the network site to securely capture information pertaining to the mobile agent's activities. Based on this experience, it is expected that any new

applications developed using this framework will also benefit from the re-use of these generic components.

The Mobile Agent Protocol (MAP) provided a robust and secure protocol that efficiently utilized the network communications infrastructure, through the use of specialized protocol messages such as REQUEST: SITE_ENTRY. Furthermore, by providing direct support for capabilities such as encryption, network pointers and remote consultation, the protocol is suitable for a range of mobile agent-based applications. In addition, the use of ASN.1 notation for specifying the application layer protocol was found to be beneficial.

To extend the prototype to support other multimedia applications, architectural components such as the Front Desk entity would not require any modification, as all new functionality could be isolated to a subset of architectural components; namely, the user application and the service application components. Furthermore, through the development of a universal knowledge language for agents (e.g. similar to KQML), network services and mobile agent scripts can be de-coupled, thereby allowing mobile agents to fully exploit their promise to facilitate the introduction of new applications into a public network environment.

Regarding the use of Tcl as a software programming language for mobile agents, it is concluded that Tcl offers advantages in terms of ease of programming and support for a powerful Graphical User Interface (GUI) toolkit, but is considerably hampered by its lack of support for binary data. Although future Tcl enhancements are expected to address this deficiency, it is expected that Java will provide a more suitable programming language for mobile agents, due to the widespread availability of the Java Virtual Machine (JVM) on network devices and the execution efficiency of Java byte code.

In terms of applicability to the Internet, the existing framework design can be readily adapted for use on the Internet (e.g. the MAP protocol is designed to operate on a TCP.IP protocol stack).

7.2 Future Work

Due to the broad nature of the mobile agent technology, there are a number of aspects of the mobile agent architecture that are recommended for further research. An investigation into the various transfer syntax encoding techniques would be beneficial to determine the optimal encoding for use with mobile agents containing multimedia data. Further investigation of compression and encryption techniques is also recommended, due to their relevance to mobile agents in permitting efficient and secure communications in an Internet environment.

With respect to the existing implementation, the use of the prototype to support new applications (e.g. Travel Reservation) is also suggested, to further substantiate and evolve the current framework design. Furthermore, the existing MAP design could be extended to provide additional message subtypes in support of sophisticated new functions.

Another major area of research, although beyond the scope of this thesis, is the development of more sophisticated scripts, especially with respect to supporting knowledge languages such as KQML that can provide an open communication language between mobile agents and network sites. In this way, mobile agents could begin to realize the promise of transforming the current Internet environment into an open platform for application developers.

And finally, further to previous comments, it has become apparent over the last year that Java is the programming language of choice for applications requiring portability across a range

of network devices. For this reason, it is left as an area for further study to implement the mobile agent framework described in this thesis using Java.

8 References

[1] R. Comerford, "The Internet's Growing Pains", *IEEE Spectrum*, Volume 33, Number 9, September 1996, pp. 46-55.

[2] B. Hermans, "Intelligent Software Agents on the Internet", thesis, Tilburg University, Tilburg, The Netherlands, 9 September 1996, http://www.hermans.org/agents/.

[3] C. Petrie, "Agent-Based Engineering, the Web, and Intelligence", *IEEE Expert*, Volume 11, Number 6, December 1996, pp. 24-29.

[4] R. Gray, "Transportable Agents", Ph.D. thesis proposal, Technical Report TR95-261, Dartmouth College, Hanover, New Hampshire, USA, 19 May 1995, http://www.cs.dartmouth.edu/reports/abstracts/TR95-261/.

[5] D. Chess, B. Grosof, C. Harrison et al., "Itinerant Agents for Mobile Computing", *IEEE Personal Communications*, October 1995, pp. 34-49.

[6] J. White, "Telescript Technology: Mobile Agents", White Paper, General Magic Inc., Sunnyvale, California, USA, 1996.

[7] J. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley Publishing Company, Reading, Massachusetts, USA, 1994.

[8] B. Kwan and A. Karmouch, "Multimedia Agents in a Distributed Broadband Environment", *IEEE Proceedings of ICC'96*, Dallas, Texas, USA, June 1996.

[9] "FTP Software Agent Technology", White Paper, FTP Software Inc., 20 February 1997, http://www.ftp.com/products/whitepapers/4agent.htm.

[10] "Oracle Mobile Agents", Oracle Corporation, 1997, http://www.oracle.com/products/networking/mobile_agents/ html/datasheet.html.

[11] J. White, "Telescript Technology: The Foundation for the Electronic Marketplace", White Paper, General Magic Inc., Sunnyvale, California, USA, 1994.

[12] R. Doorenbos, O. Etzioni and D. Weld, "A Scaleable Comparison Shopping Agent for the World Wide Web", *Agent '97 Conference Proceedings*, ACM, 1997.

[13] "Foundation for Intelligent Physical Agents (FIPA)", http://drogo.cselt.stet.it/fipa/.

[14] "The Agent Society", http://www.agent.org/.

[15] "University of Ottawa Multimedia Information Research Laboratory (MIRL)", http://deneb.genie.uottawa.ca/webdata/index3.html.

[16] "MIT Media Lab, Software Agents Group", http://agents.www.media.mit.edu/groups/agents/.

[17] "Mobile Agents at Dartmouth College", http://www.cs.dartmouth.edu/~agent/.

[18] "IBM Intelligent Agents", http://www.networking.ibm.com/iag/iaghome.html.

[19] "Oracle Mobile Agents", http://www.oracle.com/products/networking/mobile_agents/html/.

[20] "General Magic", http://www.genmagic.com/.

[21] "ForeFront's WebWhacker", http://www.ffg.com/whacker/.

[22] "Firefly", http://www.ffly.com/Home.html.

[23] M. Koster, "Robots in the Web: Threat or Treat?", *ConneXions*, Volume 9, No. 4, April 1995, http://info.webcrawler.com/mak/projects/robots/threat-or-treat.html.

[24] "Lycos", http://www.lycos.com/.

[25] "Yahoo!", http://www.yahoo.com/.

[26] "Carnegie Mellon University Intelligent Software Agents", http://www.cs.cmu.edu/~softagents/vision.html.

[27] B. Sheth, "A Learning Approach to Personalized Information Filtering", M. Sc. Thesis, Massachusetts Institute of Technology (MIT), Boston, Massachusetts, USA, February 1994.

[28] B. Kwan, "Personalized Multimedia News Agents in a Broadband Environment", M.Sc. Thesis, University of Ottawa, Ottawa, Ontario, Canada, September 1996.

[29] "University of Maryland Simple HTML Ontology Extensions (SHOE)", http://www.cs.umd.edu/projects/plus/SHOE/.

[30] M. Genesereth and S. Ketchpel, "Software Agents", *Communications of the ACM*, Volume 37, No. 7, pp. 48-53, July 1994.

[31] ITU-T Recommendation X.690: "Information Technology - ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", July 1994.

[32] ITU-T Recommendation X.680: "Information Technology - Abstract Syntax Notation One (ASN.1): Specification of Basic Notation", July 1994.

[33] W. Stallings, "Data and Computer Communications", Fourth Edition, pp. 450-463, Maxwell MacMillan Canada, Toronto, Ontario, Canada, 1994.

[34] J. Postel, "Internet Protocol", RFC0791, Internet Request For Comments, 1 September 1981.

[35] J. Reynolds and J. Postel, "Assigned Numbers", RFC1700, Internet Request For Comments, 20 October 1994.

[36] J. Postel, "Transmission Control Protocol", RFC0793, Internet Request For Comments, 1 September 1981.

[37] M. Schoffstall, M. Fedor, J. Davin et al., "A Simple Network Management Protocol (SNMP)", Internet Request For Comments, 10 May 1990.

[38] D. Lange, "Agent Transfer Protocol (ATP)", Draft version 0.1, IBM Tokyo Research Laboratory, http://www.trl.ibm.co.jp/aglets/atp/atp.htm, 29 July 1996.

[39] "Aglets", http://www.trl.ibm.co.jp/aglets.

[40] ITU-T Recommendation X.681: "Information Technology - Abstract Syntax Notation One (ASN.1): Information Object Specification", July 1994.

[41] ITU-T Recommendation X.682: "Information Technology - Abstract Syntax Notation One (ASN.1): Constraint Specification", July 1994.

[42] ITU-T Recommendation X.683: "Information Technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 Specifications", July 1994.

[43] ITU-T Recommendation X.691: "Information Technology - ASN.1 Encoding Rules: Specification of Packed Encoding Rules (PER)", July 1994.

[44] N. Mitra, "Efficient Encoding Rules For ASN.1-Based Protocols", AT&T Technical Journal, May/June 1994.

[45] "Marimba", http://www.pointcast.com/products/.

[46] "PointCast", http://www.pointcast.com/whatis_fromhome.html.

[47] R. Buhr and R. Casselman, "Designing with Timethreads", SCE-93-05, Carleton University, Ottawa, Ontario, Canada, 17 September 1993.

[48] B. Welch, "Practical Programming in Tcl and Tk", Prentice Hall, Upper Saddle River, New Jersey, USA, 1995.

[49] E. Anuff, "The Java Sourcebook", Wiley Computer Publishing, Toronto, Ontario, Canada, 1996.

[50] "Tcl/Tk Release 8.0", http://sunscript.sun.com/TclTkCore/8.0.html.

[51] T. Kientzle, "Internet File Formats", The Coriolis Group, Scottsdale, Arizona, USA, 1995.

Appendix A

9 ASN.1 Specification of Mobile Agent Protocol (MAP)

This appendix contains the ASN.1 representation of the Mobile Agent Protocol (MAP) defined in Chapter 5. With the introduction of automatic tags into the ASN.1 notation, it is no longer necessary to manually define tags [32]. However, to strengthen the linkages with the pictorial representations in Chapter 5 and the MAP protocol encodings in Appendix B, tags have been used in a few specific instances.

```
-- Mobile Agent Protocol (MAP)

MAP DEFINITIONS ::= BEGIN

-- general message format

MAP_Message ::= SEQUENCE {
        security_flag           BIT STRING {
                                        authentication (0),
                                        integrity (1),
                                        privacy (2)
                                }
        digital_signature       OCTET STRING OPTIONAL,
        source_identifier       VisibleString OPTIONAL,
        destination_identifer   VisibleString OPTIONAL,
        protocol_version        INTEGER {version 0.1 (0)},
        message_type            MessageType
}

-- defined message types

MessageType ::= CHOICE {
        push                    Push-PDU,
        pull                    Pull-PDU,
        request                 Request-PDU,
        response                Response-PDU,
        error                   Error-PDU
}


-- PUSH Protocol Data Unit (PDU)

Push-PDU ::= [APPLICATION 0] IMPLICIT CHOICE {
        transfer_agent          PushTransferAgent,
        update_agent            PushUpdateAgent,
        update_user             PushUpdateUser,
        put_file                PushPutFile
}
```

```
-- PUSH: TRANFER_AGENT message format

PushTransferAgent ::= [0] IMPLICIT SEQUENCE {
        user_information        UserInformation,
        agent_information       AgentInformation,
        script                  Script,
        site_journals           SEQUENCE OF SiteJournal,
        command_library SEQUENCE OF Command,
        document_library SEQUENCE OF Document
}

-- user information component

UserInformation ::= [0] IMPLICIT SET {
        authentication_flag     BIT STRING {
                authentication (0),
                integrity (1)
        },
        digital_signature       OCTET STRING OPTIONAL,
        user_name               Name,
        user_ID                 VisibleString,
        org_affiliation         VisibleString,
        occupation_title        VisibleString,
        business_address        Address,
        email_address           VisibleString,
        phone_number            VisibleString,
        home_address            Address
}

-- agent information component

AgentInformation ::= [1] IMPLICIT SET {
        Authentication_flag     BIT STRING {
                authentication (0),
                integrity (1)
        },
        digital_signature       OCTET STRING OPTIONAL,
        agent_serial_number     VisibleString,
        departure_date          Date,
        expiry_date             Date,
        home_IP_address         VisibleString,
        home_URI                VisibleString,
        size_limit              INTEGER,        -- size in bytes
        accepted_doc_types      DocumentType,
        compression_format      CompressionFormat
}

-- script component

Script ::= [2] IMPLICIT SET {
        script_language         SupportedLanguage,
        language_version        VisibleString,
        author_name             VisibleString,
        author_email_address    VisibleString,
        script_title            VisibleString,
        script_version          VisibleString,
```

```
        application_name        SupportedApplication,
        application_version     VisibleString,
        state_variables         OCTET STRING,
        script_pointer          VisibleString,
        raw_script              OCTET STRING
}

-- site journal component

SiteJournal ::= [3] IMPLICIT SET {
        security_flag           BIT STRING {
                authentication (0),
                integrity (1),
                privacy (2)
        },
        digital_signature       OCTET STRING OPTIONAL,
        source_identifer        VisibleString,
        site_name               VisibleString,
        site_IP_address         VisibleString,
        site_URI                VisibleString,
        arrival_time            UTCTime,
        departure_time          UTCTime,
        services_used           VisibleString
        billing_information     OCTET STRING,
        error_log               OCTET STRING,
        private_data            OCTET STRING
}

-- command component

Command ::= [4] IMPLICIT SET {
        Authentication_flag     BIT STRING {
                authentication (0),
                integrity (1)
        },
        digital_signature       OCTET STRING OPTIONAL,
        command_language        SupportedLanguage,
        command_version         VisibleString,
        command_name            VisibleString,
        command_pointer         VisibleString,
        command_size            INTEGER,        -- size in bytes
        raw_data                OCTET STRING
}

-- document component

Document ::= [5] IMPLICIT SET {
        security_flag           BIT STRING {
                authentication (0),
                integrity (1),
                privacy (2)
        },
        digital_signature       OCTET STRING OPTIONAL,
        source_identifer        VisibleString,
        document_name           VisibleString,
        document_type           DocumentType,
```

```
        source_site             VisibleString,
        author                  SEQUENCE OF VisibleString,
        last_modified           UTCTime,
        version_number          VisibleString,
        compression_format      CompressionFormat,
        document_size           INTEGER,        --size in bytes
        document_pointer        VisibleString,
        raw_data                OCTET STRING
}

-- commonly used data types (e.g. DocumentType) will be defined at the end of the module


-- PUSH: UPDATE_AGENT message format

PushUpdateAgent ::= [1] IMPLICIT SEQUENCE {
        user_name               Name,
        agent_serial_number     VisibleString,
        ack_flag                ENUMERATED {
                noAckRequired (0),
                ackRequiredUponSuccessfulCompletionOfActivity (1)
        }
        operation               ENUMERATED {
                returnHomeNow (0),
                returnHomeWhenSiteActivityIsCompleted (1),
                abortAndSelfTerminate (2),
                updateWithNewData (3),
                reloadNewScript (4)
        }
        raw_data                OCTET STRING
}


-- PUSH: UPDATE_USER message format

PushUpdateUser ::= [2] IMPLICIT SEQUENCE {
        user_name               Name,
        agent_serial_number     VisibleString,
        raw_data                OCTET STRING
}


-- PUSH: PUT_FILE message format

PushPutFile ::= [3] IMPLICIT SEQUENCE {
        user_name               Name,
        agent_serial_number     VisibleString,
        ack_flag                ENUMERATED {
                noAckRequired (0),
                ackRequiredUponSuccessfulCompletionOfActivity (1)
        }
        document_pathname       VisibleString,
        document_filename       VisibleString,
        document_type           DocumentType,
        source_site             VisibleString,
        author                  SEQUENCE OF VisibleString,
```

```
        last_modified           UTCTime,
        version_number          VisibleString,
        compression_format      CompressionFormat,
        document_size           INTEGER,        --size in bytes
        document_pointer        VisibleString,
        raw_data                OCTET STRING
}



-- PULL Protocol Data Unit (PDU)

Pull-PDU ::= [APPLICATION 1] IMPLICIT CHOICE {
        read_object             PullReadObject,
        get_file                PullGetFile
}


-- PULL: READ_OBJECT message format

PullReadObject ::= [0] IMPLICIT SEQUENCE {
        object                  CHOICE {
                host_class      HostClass,
                user_class      UserClass,
                agent_class     AgentClass
        }
        user_name               Name OPTIONAL,              -- not used with host class
        agent_serial_number     VisibleString OPTIONAL,     -- not used with host class
}

        HostClass               ::= [0] IMPLICIT SEQUENCE OF {
        -- contains list of defined objects for host class
                objectID                ENUMERATED {
                        site_name (0),
                        system_status (1),
                        available_services (2)
                }
        }

        UserClass               ::= [1] IMPLICIT SEQUENCE OF {
        -- contains list of defined objects for user class
                objectID                ENUMERATED {
                        online_status (0)
                }
        }

        AgentClass              ::= [2] IMPLICIT SEQUENCE OF {
        -- contains list of defined objects for agent class
                objectID                ENUMERATED {
                        current_status (0),
                        next_site_visited (1),
                        arrival_time (2),
                        departure_time (3)
                }
        }
```

```
-- PULL: GET_FILE message format

PullGetFile ::= [1] IMPLICIT SEQUENCE {
        document_pathname       VisibleString,
        document_filename       VisibleString,
        accepted_doc_type       DocumentType
        accepted_compression    CompressionFormat
        user_name               Name,
        agent_serial_number     VisibleString
}



-- REQUEST Protocol Data Unit (PDU)

Request-PDU ::= [APPLICATION 2] IMPLICIT CHOICE {
        site_entry              RequestSiteEntry,
        agent_consultation      RequestAgentConsultation,
        user_consultation       RequestUserConsultation
}


-- REQUEST: SITE_ENTRY message format

RequestSiteEntry ::= [0] IMPLICIT SEQUENCE {
        user_name               Name,
        agent_serial_number     VisibleString,
        user_identifier         VisibleString,
        required_services       SEQUENCE OF SupportedApplication,
        required_security       BIT STRING {
                authentication (0),
                integrity (1),
                privacy (2)
        }
}


-- REQUEST: AGENT_CONSULTATION message format

RequestAgentConsultation ::= [1] IMPLICIT SEQUENCE {
        user_name               Name,
        agent_serial_number     VisibleString
}


-- REQUEST: USER_CONSULTATION message format

RequestUserConsultation ::= [2] IMPLICIT SEQUENCE {
        user_name               Name,
        agent_serial_number     VisibleString
}
```

```
-- RESPONSE Protocol Data Unit (PDU)

Response-PDU ::= [APPLICATION 3] IMPLICIT CHOICE {
-- contains a list of initiating message types
        push                    PushResponse,
        pull                    PullResponse,
        request                 RequestResponse
}

        PushResponse ::= [0] IMPLICIT SEQUENCE {
                message_subtype                 INTEGER {
                        update_agent (1),
                        put_file (3)
                } (update_agent | put_file),
                operation_status                ENUMERATED {
                        fail (0),
                        success (1)
                }
        }

        PullResponse ::= [1] IMPLICIT CHOICE {
        -- contains a list of message format depending on the subtype of the initiating message
                pull_read_object                PullReadObject,
                pull_get_file                   PullGetFile
        }

        PullReadObject ::= [0] IMPLICIT SEQUENCE {
                operation_status                ENUMERATED {
                        fail (0),
                        success (1)
                }
                object_class                    ENUMERATED {
                        host (0),
                        user (1),
                        agent (2)
                }
                object                          SEQUENCE OF Object
        }

        Object ::= SEQUENCE {
                object_ID                       INTEGER,
                object_value                    VisibleString
        }

        PullGetFile ::= [1] IMPLICIT SEQUENCE {
                operation_status                ENUMERATED {
                        fail (0),
                        success (1)
                }
                document_name           VisibleString,
                document_type           DocumentType,
                source_site             VisibleString,
                author                  SEQUENCE OF VisibleString,
                last_modified           UTCTime,
                version_number          VisibleString,
                compression_format      CompressionFormat,
```

```
            document_size           INTEGER,        --size in bytes
            document_pointer        VisibleString,
            raw_data                OCTET STRING
        }


        RequestResponse ::= [2] IMPLICIT SEQUENCE {
            message_subtype                 ENUMERATED {
                    site_entry (0),
                    agent_consultation (1),
                    user_consultation (2)
            },
            request_status                  ENUMERATED {
                    denied (0),
                    accepted (1),
                    conditionally_accepted (2)
            },
            required_security               BIT STRING {
                    authentication (0),
                    integrity (1),
                    privacy (2)
            }
        }



-- ERROR Protocol Data Unit (PDU)

Error-PDU ::= [APPLICATION 4] IMPLICIT SEQUENCE {
        user_name               Name,
        agent_serial_number     VisibleString,
        error_status            ENUMERATED {
                message_type_not_recognized (0),
                message_decoding_error (1),
                compromised_integrity (2),
                message_interpretation_error (3),
                message_validation_error (4),
                agent_execution_error (5),
                general_error (6)
        }
        raw_data                OCTET STRING OPTIONAL
}


-- list of data types referenced throughout the module

Name ::= SEQUENCE {
        first_name              VisibleString,
        initial                 VisibleString OPTIONAL,
        last_name               VisibleString
}

Address ::= SEQUENCE {
        street                  VisibleString,
        city                    VisibleString,
        country                 VisibleString OPTIONAL,
```

```
        postal_code             VisibleString
}

Date ::= SEQUENCE {
        year                    INTEGER (SIZE(4)),
        month                   INTEGER (0..12),
        day                     INTEGER (0..31)
}

DocumentType ::= ENUMERATED {
        text (0),
        gif (1),
        mpeg (2),
        html (3),
        pdf (4),
        postscript (5),
        jpeg (6),
        rtf (7),
        mig (8)                 -- many more document types can be defined as per [51]
}

CompressionFormat ::= ENUMERATED {
        gzip (0),
        stuffit (1),
        zip (2),
        arc (3),
        shar (4),
        zoo (5)                 -- many more compression formats can be defined as per [51]
}

SupportedApplication ::= ENUMERATED {
        multimedia_document_retrieval (0)
}

SupportedLanguage ::= ENUMERATED {
        tcl (0),
        java (1),
        aglets (2),
        telescript (3),
        agent_tcl (4),
        obliq (5),
        tacoma (6),
        cyber_agent (7)         -- many more agent languages can be defined
}


END     -- end of MAP module
```

Appendix B

10 BER Encoding of Mobile Agent Protocol (MAP)

This appendix contains two examples of the Basic Encoding Rules (BER) encoding of Mobile Agent Protocol (MAP) messages specified in Appendix A using ASN.1 notation.

PULL: READ_OBJECT Message

This example depicts the BER encoding of a PULL: READ_OBJECT message sent from a user site to a network site. Specifically, this message is used to request the status of a mobile agent with serial number 'BCF7623178' belonging to user 'Bruce Ford'. All values are expressed in hexidecimal notation.

Type	Length	Value	Description
30	1d	-	SEQUENCE: MAP_Message
03	01	00	BIT STRING: security_flags (null)
61	18	-	Application tag: Pull-PDU
a0	16	-	Context-specific tag: PullReadObject
a2	03	-	Context-specific tag: AgentClass
02	01	00	INTEGER: current_status
30	0d	-	SEQUENCE: Name
1a	05	42 73 76 63 65	VisibleString: first_name: "Bruce"
1a	04	46 6f 63 65	VisibleString: last_name: "Ford"
1a	0a	42 43 46 37 36 32 33 31 37 38	VisibleString: agent_serial_number: "BCF7623178"

The raw data stream transmitted for this message is:

30 1d 03 01 00 61 18 a0 16 a2 03 02 01 00 30 0d 1a 05 42 73 76 63 65 1a 04 46 6f 63 65 1a 0a 42 43 46 37 36 32 33 31 37 38.

REQUEST: SITE_ENTRY Message

This example depicts the BER encoding of a REQUEST: SITE_ENTRY message sent from a user site to a network site. Specifically, this message is used to request site entry for a mobile agent with serial number 'BCF7623179' belonging to user 'Bruce Ford' with user ID '173242' to access the Multimedia Document Retrieval application. Furthermore, the user has requested that encryption techniques be used to ensure the privacy of the subsequent transactions. All values are expressed in hexidecimal notation.

Type	Length	Value	Description
30	30	-	SEQUENCE: MAP_Message
03	01	00	BIT STRING: security_flags (null)
62	2b	-	Application tag: Request-PDU
a0	29	-	Context-specific tag: RequestSiteEntry
30	0d	-	SEQUENCE: Name
1a	05	42 73 76 63 65	VisibleString: first_name: "Bruce"
1a	04	46 6f 63 65	VisibleString: last_name: "Ford"

Type	Length	Value	Description
1a	0a	42 43 46 37 36 32 33 31 37 39	VisibleString: agent_serial_number: "BCF7623179"
1a	06	31 37 33 32 34 32	VisibleString: user_identifier: "173242"
03	01	00	BIT STRING: required_services (Multimedia Doc Retrieval)
03	01	20	BIT STRING: required_security (Privacy)

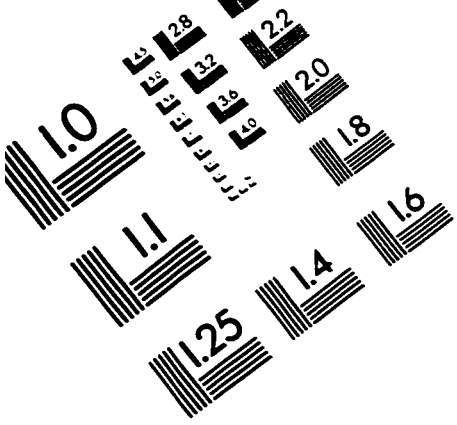
The raw data stream transmitted for this message is:

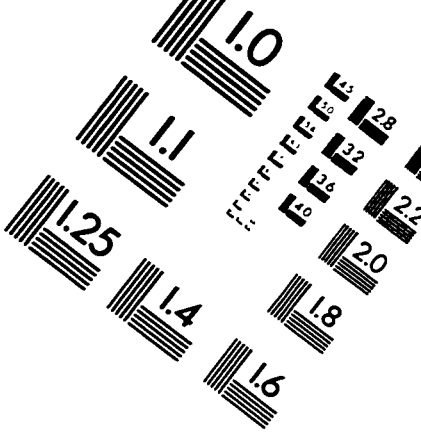
30 30 03 01 00 62 2b a0 29 30 0d 1a 05 42 73 76 63 65 1a 04 46 6f 63 65 1a 0a 42 43 46 37 36 32 33 31 37 39 1a 06 31 37 33 32 34 32 03 01 00 03 01 20.

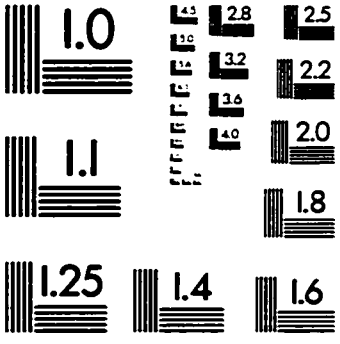
Appendix C

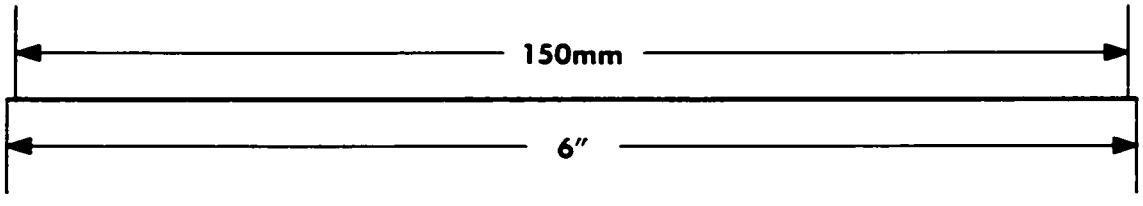
11 Publications

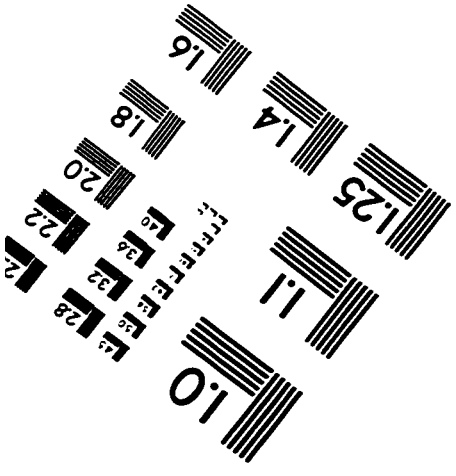
B. Ford and A. Karmouch, "An Architectural Model For Mobile Agent-based Multimedia Applications", *Proceedings from the Canadian Conference on Broadband Research (CCBR) '97*, Ottawa, Ontario, Canada, April 1997.

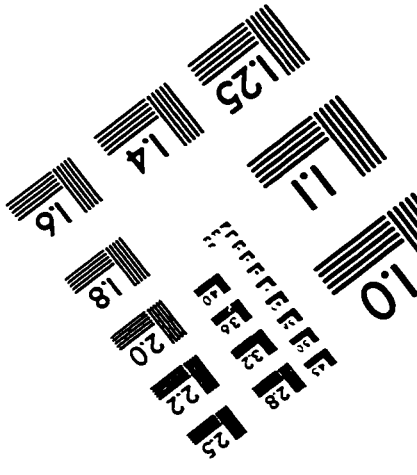












APPLIED IMAGE . Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989