AZIMAS: ALMOST ZERO INFRASTRUCTURE MOBILE AGENT SYSTEM

By

AMAR NALLA

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2001

ACKNOWLEDGMENTS

I would like to express my thanks to Dr. Abdelsalam (Sumi) Helal for all his support and motivation. I would also like to thank Dr. Michael Frank and Dr. Joachim Hammer for agreeing to serve in my committee and for reviewing my work.

I extend special thanks to my parents for being patient and for understanding my efforts that kept me away from home for a long time. I would also like to thank Choonhwa Lee for providing help with my Unix account in the Harris Lab. I also thank Sonali for understanding that this work was important and for letting me stay in the lab for long hours.

I also thank members of the Apache mailing list, the Agents mailing list and the Java-res mailing list for answering all of my questions and for helping me solve some tough problems. I also thank all the users of Java usenet groups for providing valuable suggestions. This work would not have been possible without all of this help.

TABLE OF CONTENTS

page

LIST OF FIGURES

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

AZIMAS: ALMOST ZERO INFRASTRUCTURE MOBILE AGENT SYSTEM

By

Amar Nalla

August, 2001

Chairman: Abdelsalam (Sumi) Helal
Major Department: Computer and Information Science and Engineering

Mobile agents promise to bring in a new era in the field of World Wide Web and Internet Computing. Many mobile agent systems have been developed but none of them are widely deployed on the Internet. Although these systems have been around for sometime, their full potential has not been realized because of the lack of a suitable infrastructure that would allow for the seamless integration of mobile agents on the Internet. Most of the developed systems advocate the use of proprietary architectures and protocols and it is difficult to integrate the new systems with the existing Internet architecture and well-established protocols like Hyper Text Transfer Protocol (HTTP). Thus, there is a need for a simple but effective mobile agent system that can be easily deployed on the Internet without requiring extensive additions or modifications to the present infrastructure.

This thesis presents the design and implementation of the, Almost Zero Infrastructure Mobile Agent System (AZIMAS), a mobile agent system that is easy to

deploy on the Internet. The platform that has been developed enables mobile agents to move at will on the Internet by extending the role of existing Apache web servers to act as hosts for visiting mobile agents. Hyper Text Transfer Protocol, which is a widely used transfer protocol on the Internet is used to enable the mobility of the agents. This infrastructure enables developers to build effective Internet agents that will open a whole new world of web applications based on mobile agents. The performance implications of using the developed infrastructure on a web server are also presented. Some of the problems faced while developing the system and the future developments necessary to extend the system are outlined.

CHAPTER 1
INTRODUCTION

There has been much interest in mobile agents for a long time. Their predicted emergence as the key future technologies for the Internet has not materialized for a number of valid reasons. Although many different mobile agent systems have been developed they have not been deployed on the World Wide Web (WWW) on a large scale because of the lack of a suitable infrastructure supporting their operation. Most of the current systems necessitate the existence of a specialized “agent server” at each host supporting a mobile agent. This presents an administrative overhead, and more seriously, an interoperability nightmare given the lack of standards for agent systems. This has seriously hindered the deployment of mobile agents on the Internet. Another problem in deploying mobile agents are the various security issues that need to be considered when building an agent system. Webmasters are reluctant to support agent systems because of security concerns, which adds to the problem. In addition to these problems, there is a dearth of agent-based applications that are popular among users and hence there is no demand on the network administrators to install support for agent-based systems. There is a need to develop a system that can be easily blended into the existing infrastructure and yet be powerful to present new ideas and application scenarios. This thesis presents AZIMAS (Almost Zero Infrastructure Mobile Agent System), an HTTP [BER99] based infrastructure that supports mobile agents on the Internet. This is achieved by making suitable extensions to existing web servers to support mobile agents with almost zero additional infrastructure.

1.1Vision of AZIMAS

There is a need to bring order to the vastly expanding cyberspace. People have often suggested that deploying agents on the Internet could be an ideal way to tap the vast amount of information and also to use the Internet to make our daily activities more useful and productive. We envision mobile agents freely roaming around the cyberspace doing simple tasks that make life easier for people all around the world. The AZIMAS system allows people to use personalized and highly efficient search engines to retrieve the exact and the latest information that is of interest. This can be done by building “true” web crawlers that move from host to host in search of information that is needed by the user. The system also strives to enable the use of agents for developing collaboration applications that tie together people and information sources. The idea is to develop mobile agents representing users that are capable of interacting with other users and with other mobile agents.

1.2 Goal of Thesis

The goal of this thesis is to develop the basic infrastructure that is necessary to deploy the AZIMAS agents. The development of a basic infrastructure that defines the services to be provided to an agent is the first step in creating any agent-based applications. Once the infrastructure is in place the next step would be to develop agents that can be deployed easily on the developed infrastructure. This would enable the users of the system to develop agent-based applications that are currently not prevalent on the Internet. The agents are developed using the Java programming language since Java provides many features that facilitate the development of mobile agents. The server-side infrastructure is based on the Apache web server. It is used as a launch pad for the agents

visiting a server. Deployment of mobile agents on a machine raises a number of security concerns that are addressed by the AZIMAS system. The thesis also studies the performance implications of deploying the system on existing web servers.

Thus, this thesis lays down the foundation of the platform that will be used to extensively deploy mobile agents on the Internet. This platform can be used as a base for future development of a full-fledged mobile agent system that is tightly integrated with the Internet. Currently, issues like agent cloning and inter-agent communication are not addressed in the AZIMAS system.

A suite of agent-based applications that use the AZIMAS platform is being developed separately using an agent-programming model proposed in [REN01]. This programming model will enable developers to quickly develop agent-based applications that use the power of the AZIMAS platform in conjunction with the Internet. The importance of this work is due to the fact that there have been no known previous attempts to incorporate mobile agents into a well known web server like the Apache web server.

This thesis also explores the powerful paradigm of mobile code, which is a relatively new field in the area of distributed and mobile computing. The next chapter presents a survey of related work in the area of mobile agents. We also look at some of the common web servers and other systems that attempt to develop agents that can be deployed on the Internet. Specifically we look at work [LIN95 , STRE98] in which attempts have been made to incorporate mobile agents into special web servers. Chapter 3 lays down the infrastructure requirements and security issues that come into consideration when developing a platform for mobile agents. Chapters 4 and 5 present the

design and implementation of the system. Chapter 6 presents the results of the work and shows the performance implications of using the system. Chapter 7 presents a sample “Thesis Defense Scheduling” application built using the AZIMAS system. Chapter 8 presents our experiences of developing the AZIMAS system and presents some pointers for future work.

CHAPTER 2
SURVEY OF RELATED WORK

2.1 Mobile Agents Research Issues

Research in mobile agents is a confluence of research in mobile computing and artificial intelligence. Artificial Intelligence researchers are interested in building systems that can effectively capture the needs of humans. They view agents as human surrogates that can represent people and have the ability to fulfill user-defined tasks. The emphasis is more on machine learning and knowledge retention and user interaction. Mobile computing researchers have concentrated on issues like creation of platforms for execution of agents, enabling mobility of agents etc. The focus of mobile computing researchers is the concept of mobile code as an alternate distributed computing technology.

Many mobile agent systems have been built to validate and resolve some of the common issues and challenges faced in this area. Most of these systems have been developed in Universities and other research centers. There has been very little adoption of this technology in the software industry. There have been some attempts to develop agent-based systems in the industry, but most of these systems are not full fledged and don't address a number of open issues. This could be a reason for mobile agents not being very popular in today's scenario. The area is still nascent and there are plenty of avenues for research, considering some of the open questions that are not yet solved.

2.1.1Taxonomy of Mobile Agents

Mobile agents are generally classified using the following guidelines.

- Language. The choice of language to develop mobile agents is often a big question. The language that is chosen needs to provide the abstractions and methods needed to develop mobile agents. Programming languages that compile the source code to native machine code are generally not suitable for development of agent-based systems, as the agents need to run in a heterogeneous environment. Usually, agents are written in languages that are interpreted rather than compiled. This arises from the need for the agent to migrate to different machines and execute in the host machine. Java is an ideal language of choice for development of mobile agents due to its "write once run anywhere" philosophy. Java is not the sole language of choice for developing mobile agents. Some systems have been built using other interpreted languages.

- Granularity of Mobility. The main advantage of mobile agents over other distributed computing technologies is their ability to migrate from one machine to another and perform computations at different locations and at the same time preserve their state of execution between two hosts. Some of the mobile agent systems provide thread level migration. Such mobile agents can retain their state and start from exactly the same point from where they left off in the previous host. Most of the mobile agents do not provide thread-level mobility, as it is often difficult to achieve using the existing primitives. Agents that are written in Java do not provide thread-level mobility because of the underlying JVM limitations, though there have been some attempts to modify the JVM and to provide thread-level mobility. The consensus among researchers is to build agents that do not need thread-level mobility, as most of the applications that have been envisioned for mobile agents do not need thread-level mobility.

- Agent Applications. Mobile Agent systems have been developed for a number of different applications. The applications can be divided into two distinct categories. The first category comprises of applications that use agents for Internet related functions. Agents have been developed that can be used in search applications and are generally known as search bots. Many e-commerce models have been developed that use agents as one of the components. It has also been proposed to use agent as human representatives in web auctions and other areas where there could be interaction between agents and other users. The other categories of agent applications include network configuration and maintenance, and monitoring of system usage.

2.1.2 Java Based Agents

Java has emerged as the standard language for developing mobile agents [WON99]. Some of the popular mobile agent systems that have been developed in Java include the Aglet System [LAN96] developed at IBM Research Labs in Tokyo, Voyager developed by ObjectSpace, Concordia [CON01] developed by Mitsubishi Electric and Ajanta [TRI99] developed at University of Minnesota.

One of the main reason to choose Java, as a language to develop agents is the many language-level features provided by Java that facilitates development of agents. Java is a popular language because of its multi-platform support and portability. There are also several features not found in any other languages that directly support implementation of mobile agents. For example, there is a need to transfer the agent state and code from one host to another. This can be easily achieved by using Java's serialization capabilities that enables conversion of the agent state to a format that can be transmitted over the network. This is further facilitated by the built-in support for networking and sockets in Java.

Additionally, Java facilitates migration of code and state via its class-loading mechanism. Java's class loaders can dynamically load classes included in an application either locally from the system CLASSPATH or through the network. Most of the mobile agent systems have a class loader for each agent that arrives to the system. These class loaders create a separate namespace for each of the agents and protect agents from tampering with each other.

The other main feature of Java that is very useful is its built-in security model that is both robust and secure. Security Managers creates a sandbox-based security model and highly restricts the agents from attacking the host system. The security policy that is imposed at a system can have multiple granularity and can be highly configured. Java does not provide solutions to all of the questions that are raised about security of mobile agents but it facilitates the building of systems that are secure and tamper-proof.

2.1.3 Non Java-Based Agents

The first mobile agents were developed by General Magic in an agent-specific language, Telescript [WHI96]. Telescript was not commercially successful, primarily

because it required programmers to learn a completely new language. General Magic shelved the Telescript project and later they built a Java based system called Odyssey that uses the same design framework. Tacoma [TAC01] was developed in joint project by Norway's University of Tromso and Cornell University. In Tacoma the agents are written in Tcl, although they can technically carry scripts written in other languages. Agent Tcl [GRA97] developed at Dartmouth has agents written in the Tcl scripting language.

2.2 Agent Servers

Each machine that intends to host mobile agents needs to provide an execution environment that executes agent code. Additionally, the host needs to provide primitive operations that allow the agent programmers to develop agents that communicate or access the host's resources [KAR98]. These machines are often referred to as "*Agent Servers*". The Agent Server is responsible for providing resources and run-time services for agents. It also provides a security manager to protect the host from the agent also to create a secure framework for the execution of the agents. Next we present the architecture of the agent server for two Java based mobile agents, Ajanta and Aglets.

2.2.1 Ajanta Server Architecture

Figure 2.1 shown below, shows the architecture of the Ajanta server with a visiting mobile agent. The interface between the visiting agents and the server is an *agent environment* object that is present in each server. The server's *domain database* keeps track of agents currently executing on the server. The *agent transfer* component is involved with the migration of the agent from current host to the next host. In the Ajanta system the server resources are visible to the agent in the form of application-specific resources. The *resource registry* creates a safe binding between the resources and agents.

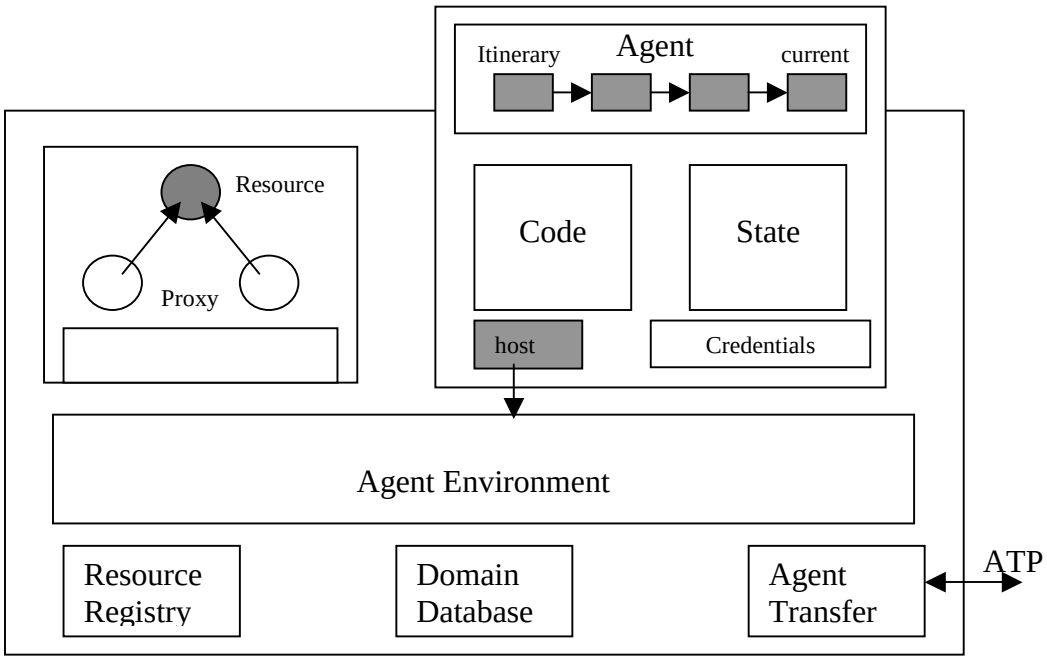

Figure 2-1 Ajanta server architecture [TRI99]

The Ajanta system uses it own customized Agent Transfer Protocol (ATP) for achieving mobility of agents from host to host. The protocol that is used is not a generic agent transfer protocol and is customized for the Ajanta servers.

2.2.2 Aglets Runtime Layer

Figure 2.2 shows the structure of the Aglets Runtime Layer. It consists of a Core Framework and a Communication Layer [OSH98]. The Aglets Runtime Layer is responsible for the execution of aglets on an agent server and it consists of a core framework and subcomponents The subcomponents are designed to be extensible and customizable The runtime layer does not provide communication mechanism for enabling mobility of agents. Instead, it uses the communication API that abstracts the communication between agent servers. The Aglets system uses the Agent Transfer Protocol (ATP) as the default implementation of the communication layer.

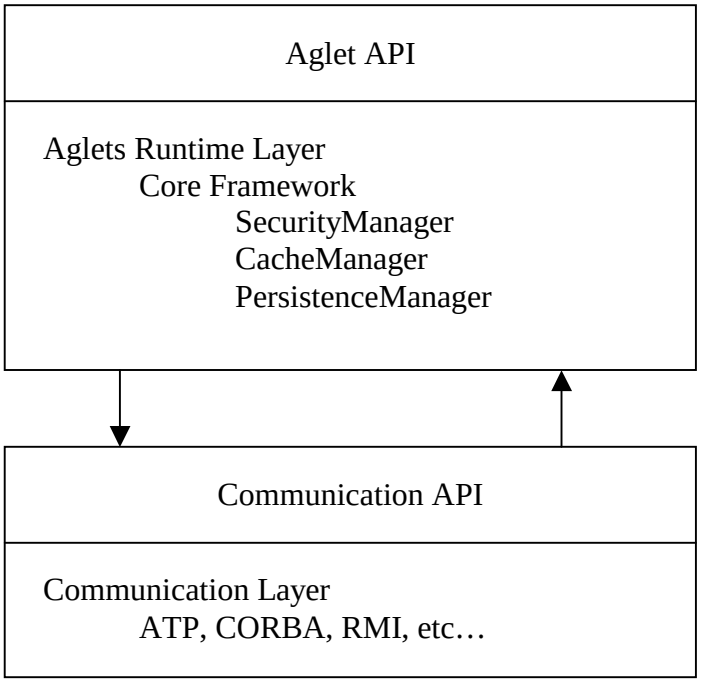

Figure 2-2 Aglets runtime layer and communication API [LAN96]

The core framework present in the runtime layer provides the following functionality for aglet execution:

- Serialization and de-serialization of agents
- Class loading and migration
- Garbage Collection

Some of the other sub-components present in the aglets runtime layer are:

- Persistence Manager. The Persistence Manager is responsible storing the serialized agent to a persistent medium.
- Cache Manager. The Cache Manager is responsible for maintaining the bytecode used by the aglet.
- Security Manager. The Security Manager is responsible for providing the necessary security features present in the system and it protects the host as well as the agent from malicious entities.

The Aglets system relies on an Agent Transfer Protocol (ATP) [LAN97] for transfer of agents from host to host. ATP is a new protocol that was conceived along with the Aglets system at IBM research labs in Tokyo. ATP is an application-level, platform-

independent platform protocol for transferring mobile agents between hosts. ATP was designed to handle agent mobility in a general and uniform manner and caters to agents written in different languages and to a variety of vendor specific agent platforms. Each host that desires to support agent mobility through ATP needs to provide an ATP-based agent service on the host. The agent platform also needs to have a handler for the ATP messages.

Both, Ajanta and Aglets are well-conceived mobile agent systems with extensive features. It is possible to develop many mobile agent applications using either of these two systems. The Aglets system has achieved limited deployment in the industry but the reach has not been widespread due to a number of unanswered questions.

One of the big drawbacks of both the Aglets and Ajanta system is their reliance on a specialized agent server at each host. This has hindered the deployment of both the systems on the Internet.

2.3 Web Servers

Web Servers have emerged as an important component of the infrastructure of the Internet. Most of the web servers that support today's web pages have been developed by software companies and other organizations. The most popular web server is the Apache web server [APA01]. This is an open source web server and the source code is available for modification by individual users under a license. Apache is highly flexible and enables developers to incorporate extensions to depending on functionality requirements. Apache web server runs on both UNIX and NT platforms. One of the other popular web servers is the IIS web server from Microsoft. This web server is specially built for the NT platform and is gaining popularity for NT based web servers. One of the web servers that needs special mention is the Jigsaw [JIG01] web server developed by the World Wide

Consortium (W3C). This web server is written in Java and is among the few web servers written in Java. The Java Web Server formerly known as Jeeves is a Java-based web server from Sun Microsystems.

2.4 Mobile Agents on Web Servers

The first attempt to integrate mobile agents with web servers is mentioned in [LIN95]. The work outlines the use of HTTP as an agent transport protocol and web server as agent servers. The web server used was a custom web server and the infrastructure developed catered to various types of mobile agents written in different languages. For each running agent, the server provides a *runtime environment*, which acts as an interface between the agent and the host. Inter agent communication is provided in the infrastructure through a model that is based on an abstract *information space*. This *information space* can be thought of as an information repository from where the agents can access information or add new information. The mechanism used to for communication with the *information space* is again based on HTTP. The agents can use a GET method to read information from the *information space* and use the POST method to add new information to the repository.

Some of the drawbacks of the system include very little support for security and difficulty of integrating the system with existing Internet infrastructure. Further survey of literature does not show any further attempts to extend the initial proposal and develop a full-fledged system by the authors.

The most comprehensive attempt to integrate mobile agents and web servers is presented by Stefan Funfrocken [STE 98]. The infrastructure was developed as part of the WASP (Web Agent based Service Providing) project, which aimed at providing services on Web data and using mobile agents to implement these services. The system consisted

of an HTTP server that provides standard Web server functionality. Additionally the system had a special server extension module called ‘Server Extension Environment (SAE)’ which provides the mobile agent support from the server. Both the web server and the SAE are developed in Java. The system only caters to agents written in Java.

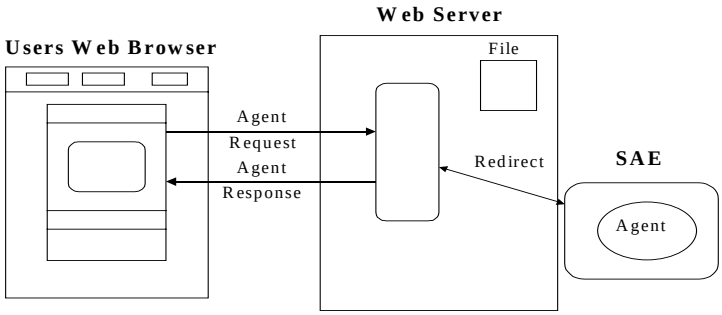

Figure 2-3 WASP architecture [STE98]

Figure 2.3 shows the architecture of the WASP infrastructure. The infrastructure provided by WASP has extensive support for mobile agents. The developers of the system have also implemented various security features that provide a basic protection to the host from the agents. In addition, the system provides agent to agent security and minimal host to agent security. The implementation is complete with a full-fledged working HTTP server and it serves the broader goal of providing web services through agents. All agents that can be invoked by a user are described in a normal html page. Users request the launch of one of the agents through a GET request. The agent is launched on the server and the agent’s GUI, which is in the form of an applet, is sent to the user’s browser. One of the key differences between the WASP infrastructure and other work including the AZIMAS system is in the fact that the agents initially reside on the web server and they have to be explicitly requested by the client as a service. In most

of the mobile agent systems the user creates agents that are then later dispatched to the different servers that have the capability to host the agents. Thus, in WASP the system does not have the provision to support any general-purpose agent developed by clients of the system.

CHAPTER 3
INFRASTRUCTURE REQUIREMENTS AND SECURITY ISSUES

It has been identified that it is impossible to take advantage of mobile agent systems due to the lack of a suitable infrastructure. There have been attempts to develop proprietary platforms that provide infrastructure for some mobile agent systems but these have not been widely accepted by the agent community and there still remains the need for a suitable infrastructure that can be easily deployed on the Internet. The infrastructure of a mobile agent system can be divided into two components, “agent client” and “agent servers”. The agent clients are responsible for creating the agents and launching them to the server. The agent servers host the agents and provide a runtime for their execution. The infrastructure also needs to lay special emphasis on security of both the host system and the agents. This chapter identifies the infrastructure requirements and security issues on the server side of a mobile agent system. It also looks at generic agent server architectures and some of the components that are needed to provide basic functionality in any mobile agent system. The next chapter will present the architecture of the AZIMAS system and the design criteria that are influenced by some of the requirements and issues presented here.

3.1 Infrastructure Requirements

The server infrastructure must provide a runtime environment for the execution of the mobile agents. It also needs to provide primitives allowing mobility of agents and inter-communication between agents. The infrastructure is also responsible for the

security of the underlying host and providing limited access to the resources on the server. In addition to this, the server also needs to create secure framework that would prevent the agents from being tampered by other agents. The infrastructure needs to provide other primitives and services that are necessary for the agents to perform its functions [ARI98]. Thus, some of the key issues that have been identified are:

1. Suitable Runtime Environment. The server needs provide a runtime environment in which the agents can be launched for execution. This environment is responsible for unpacking the agent, loading it and starting an execution thread containing the agent. In Java based mobile agent systems, the runtime environment is based on the underlying JVM. The runtime environment is also responsible for the serialization and de-serialization of the agent's state. This enables the agent to preserve its state across its itinerary.
2. Mobility Support. The server needs to implement the primitives that will enable the agent to move from the current host to the next host in its itinerary. Thus, the agent relies on the current server to send it correctly to the next host.
3. Security Issues. The platform is responsible for protecting the underlying host from any attack from agents. Additionally, the host has to authenticate the agents and also implement resource access control policies.
4. User Control of Agents. The agent system should provide features that enable the users to monitor and control the agents that have been sent by them.
5. Other Primitives. The agent needs additional primitives from the server that will enable it to perform its functions. For example, if the agent wants to send a mail or wants to browse through globally accessible, published resources on the server it needs to rely on the server to enable it to do its task.

3.2 Security Issues

Currently there are various unresolved security issues that limit the widespread deployment of agents on the Internet. The infrastructure needs to provide a robust security model that has to take care of various security issues that arise when deploying a mobile agent platform [TSC99]. If the mobile agent system is deployed in a trusted environment like, in a corporate network then some of the security constraints can be relaxed but for a full-fledged deployment on the Internet the security constraints increase by several magnitudes.

There are three issues that need to be taken into consideration when designing any secure mobile agent system [KAR98]. These are as follows:

- Agent and Server Authentication
- Access Control
- Protection of Agents

3.2.1 Agent and Server Authentication

Authentication in an agent system provides proof of identity of agents, hosts and other authorities. The host needs to know the identity of the mobile agent, which can be achieved by attaching the agent name with the agent. The agent name has to be signed by a trusted authority. Agents cannot carry private keys, and thus cannot sign and hence the agent name needs to be included with the agent.

With respect to authentication of hosts, the agent depends on the current host to send it correctly to the next host in the itinerary. Authentication code is generally not included in the agent as it can be easily spoofed by the current host. Thus, the agent will depend on the host for cryptographic processing, verification of signed material and authentication [SCH99]. Some of the attack scenarios that can be presented include redirection of the agent to a malicious host, capturing of the agent, tampering of the itinerary. Some of these cases can be dealt by providing strong host authentication schemes, but in some cases the agent may need to be monitored periodically for any tampering.

3.2.2 Access Control

The main question that needs to be addressed here pertains to protection of the host system from malicious agents. Allowing foreign mobile code to execute on a system is inherently unsafe and special care needs to be taken care with regard to this matter.

This is achieved by restricting the operations that are allowed to an agent. The runtime platform is responsible for providing methods to allow fine grained access of the host to the agents. The Java platform is ideal in this case as it provides a sandbox based model that restricts the operations of mobile code. Also the platform can be further configured by including security managers that implement the access control policy.

3.2.3 Protection of Agents

Agents need to be protected from hostile servers and from other agents executing concurrently in the same server. The agent information that needs protection includes the agent program code and other static information, the agent's state information, data that is carried by the agent. Static information is generally protected by code signing or by calculating a hash of the static portion. The data that is collected by the agent can be protected by encrypting it with the agent's public key. It is virtually impossible to protect the agent's state from the current host and currently there are no well-established schemes that guarantee complete protection of an executing agent. This is an active area of interest that is being pursued by researchers in the mobile agent community [SCH99].

An agent that is executing on the current host needs to be protected from other agents running on the same host. This is achieved by separating the different agent threads into separate namespaces.

Secure movement of the agent from the current server to the next server needs to be guaranteed. This prevents eavesdropping into the agent by unknown hosts or even tampering of agents by third parties. In addition, it is the responsibility of the agent platform to protect a currently executing agent from third parties that are not part of the system.

3.3 Resource Usage and Availability of Server

An agent that is currently executing at a server can either maliciously or unintentionally unduly consume excessive resources. These resources can be in the form of processor time, memory or network capacity. In such a situation the availability and performance of the system is affected. This situation could arise as part of a “denial of service” attack by an agent where a particular agent can block the resources and thus make the host unavailable to other agents.

The server can avoid this problem by having an effective resource allocation policy. The server is responsible for rejecting agents if its saturation point is reached. The policy is derived by calculating the amount of resources typically required by agents executing in the system and the amount of resources that are required for performing the other functions of the server.

3.4 Generic Agent Server

Generic Java-based mobile agent server consists of four major components: an agent manager; an inter-agent communications manager; a security manager and a resource manger. Figure 3-1 shows the design of a generic agent server.

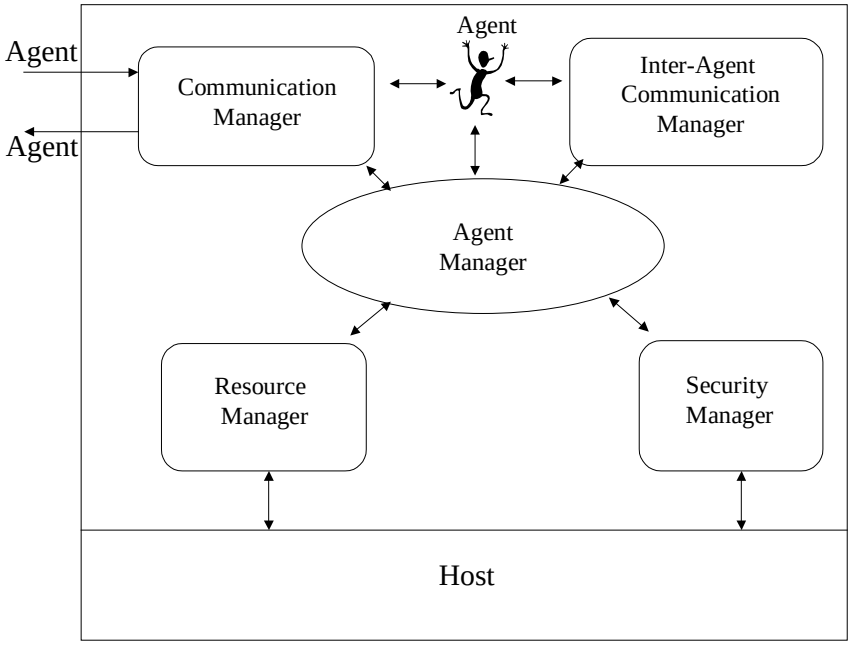

Figure 3-1 Generic mobile agent server

- Agent Manager. The Agent Manager is the main component and is responsible for receiving the agents and launching them in the runtime environment. The Agent Manager handles the serialization and de-serialization of the agents and starting the agent thread. It also acts interacts with the other components present in the server to provide a complete framework for the execution of an agent.
- Security Manager. The Security Manager is responsible for securing the host from the mobile agents. It takes care of implementing the access control policies and is accountable for authentication of visiting agents.
- Resource Manager. The Resource Manager is responsible for managing the resources available on the server. The agent may consume excessive resources and interfere with the normal performance of the server. The resource manager keeps track of the resources used by the agents and terminates agents that consume excessive resources. It also instructs the agent manager to not accept any new agents if the server is saturated.
- Inter-Agent Communication Manager. The Inter-Agent Communication Manager facilitates communication between mobile agents executing in the current server. Some systems need sophisticated forms of inter-agent communication to implement the application logic.
- Communication Manager. The Communication Manager is the interface to the mobility services provided by the server platform. It acts as a handler for the communication protocol that is responsible for enabling autonomous movement of agents.

CHAPTER 4
AZIMAS ARCHITECTURE

4.1 Introduction

The AZIMAS system has been designed considering a number of factors that would make mobile agents pervasive in the Internet. Some of the factors that were taken into consideration included ease of deployment of the platform over existing infrastructure and development of agents that can be easily built and deployed in the system. This is achieved by making minimal enhancements to the existing web servers on the Internet. This chapter will elaborate on the different components that constitute the overall system and the role played by each of them. We will try to relate some of the design choices to the infrastructure requirements and security issues in Chapter 2.

4.2 Overall Architecture

The key components in the AZIMAS system are the web servers that provide a run time infrastructure and the clients of the system that create the agents and dispatch them on the network. HTTP is the protocol used for transfer of agents from the client to the web server and also for movement of the agent from the current web server to the next web server. An agent is encapsulated as a multipart MIME message that is posted to a web server. The reason to choose HTTP, web servers and the other components has been elaborated earlier. Now we will address each of the components separately and the design considerations that went into choosing a particular approach.

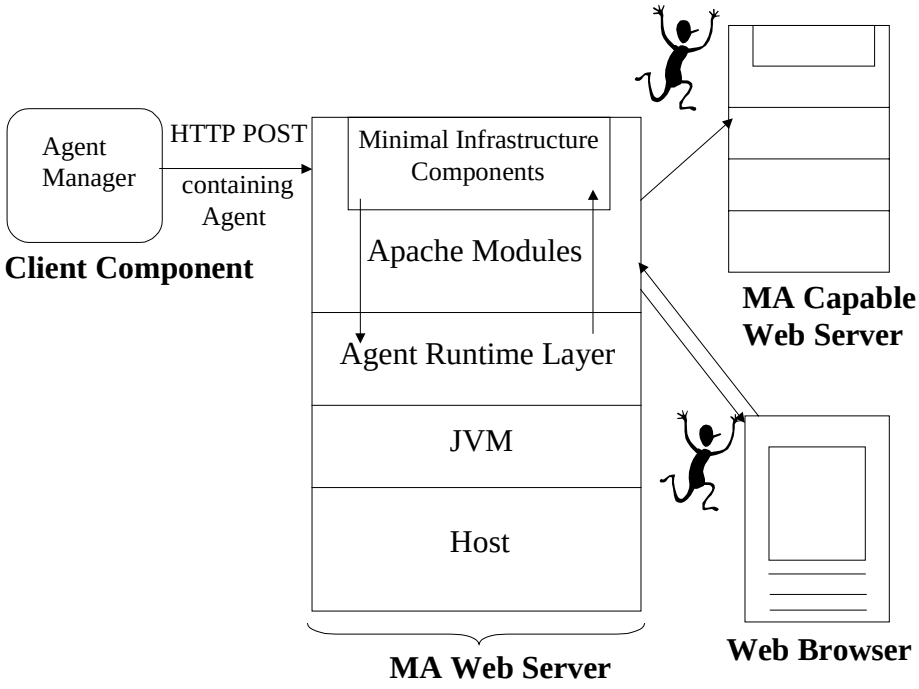

Figure 4-1 Overall AZIMAS architecture

Figure 4.1 shows the overall architecture of the AZIMAS system. AZIMAS agents are created by the users of the system by using the *Agent Manager* present in the client side. These agents are encapsulated in a HTTP request and sent to the first web server. While currently executing at the web server the agent could go a user's browser for interaction with a specified user as per the application logic of the agent. The agent uses the runtime services provided by the Apache server to move from its current location to the next server in its itinerary.

4.3 AZIMAS agents

In the AZIMAS system the basic functionality of the agent is encapsulated in the `AzimasAgent` class. Application specific agents can be created by extending from the `AzimasAgent` class and implementing the abstract *run()* method provided by this

class. The AZIMAS system is implemented in the Java language and caters for agents developed in Java. The architecture is designed to facilitate the developments of two types of agents, *Interactive Agents* and *Non-Interactive Agents*.

Interactive Agents: These agents have the capability to interact with human users through the use of e-mail and use applets for their GUI (Graphical User Interface). The agents present themselves to the users on their browsers in the form of an applet that is tightly integrated with the agent [REN01]. These agents are targeted for enabling collaboration applications and other type of applications that tie together people and information sources on the Internet. A good example is an agent for scheduling meetings between many users. The moderator can create an Interactive agent that visits each of the members invited for the meeting and gather input from each of the members and process it and arrive at a best meeting time convenient for all the interested parties.

Non-Interactive Agents: These are information-retrieving agents that need to move to multiple web servers to gather very specific information, process it and present the results to the user. These agents do not interact with other users and are primarily concerned with the data sources that are made available by the web server. Such agents can be used for creating web crawlers that move throughout the Internet to search for information. These crawlers can be called "*True Web Crawlers"* as they actually move from one site to another site to gather information unlike existing web crawlers that do not physically move site to site and gather information by sending requests to multiple sites.

4.4 Mobility of AZIMAS Agents

HTTP protocol and Java's object serialization facility are used to implement agent mobility in the AZIMAS system. This limits the granularity of mobility as object serialization captures the data values in an object's state, it does not capture the execution state of the agent's thread. Thus, on de-serialization a new thread is assigned that starts execution from the beginning.

In the AZIMAS system the agent travels between web servers by carrying its entire code and state. This approach is different from some of the other agent based system that advocate the use of a code base server that is contacted each time for the execution of the agent. In the Ajanta [TRI99] system no code is transported and all code migration takes place "on demand" during the agent's execution. The Aglets [LAN96] system has a flexible system and allows downloading of agent code on demand as well as the ability to transfer the classes along with the agent. There are both advantages and drawbacks to both the approaches. The advantage of not carrying code with agent is that the agent is extremely lightweight and easy to transport. The disadvantage of this scheme is that the classes may not always be available if they are not carried by the agents themselves, this could be either due to the unavailability of the code base server or due to firewall restrictions on the host machine. Carrying entire code makes the agent heavy and imposes network load but it has the advantage of avoiding the need for a code base server completely.

The AZIMAS architecture is designed for agents that carry the code with them as the system targets simple agents having small code size. These agents do not increase the payload of the HTTP message by a large value. Another reason to choose this approach

was the difficulty to include code base servers on the Internet, as they would demand the presence of dedicated machines to provide the agent's code.

The itinerary is either decided implicitly by the application logic of the agent or explicitly by using the host list present in the agent attributes. We expect a majority of applications to incorporate the itinerary as part of the application logic, still we have retained the feature of a pre-configured list as it is suitable for certain applications. The list is also used by the current server to verify that the agent has been routed to it through a host present in the attribute list.

In the AZIMAS system the agent is not allowed to move to an arbitrary host if pre-configured host list is already present.

4.5 Encapsulation of Agents as MIME contents

4.5.1 Mobile Agent Constituents

The structure of the mobile agent as it migrates from the current server to the next server is outlined below. The mobile agent contains the following:

1. Agent Attributes. The agent attributes contain the information describing the mobile agent. Some of the attributes include the agent name, its source host, its owner, public keys and other encryption related information. In addition the attributes may also contain the list of hosts to be visited by the agent. The attributes are read-only and should not be modified neither by the agent nor by the host web server. Table 4.1 shows a sample set of attributes for an agent in the AZIMAS system.

Table 4.1 Agent Attributes

Attribute-Name	*Attribute-Value*
Agent-Name	AZIMAsagent
Agent-Type	Interactive
Author	analla@cise.ufl.edu
Source	naples.harris.cise
Host-List	ramses.harris, cairo.harris,
Authentication Certificates	Akgfhrnokdhbg
Static Hash value	99

2. Agent Code. The agent code consists of the Java class files that are generated by the client components. These are static class files and should not be modified by the host web servers. In our architecture the agent carries its class files with it throughout its itinerary. This idea is elaborated further in the implementation section in the next chapter.
3. Agent State. The agent state is the serialized state of the agent that contains the agent's variables. Java's serialization feature is used to preserve the state of the agent across machine boundaries.

In addition to these components, an *Interactive* agent may also contain additional parts like applet class files and html files that are used by the agent. The additional files are transmitted as agent code but are not loaded by the agent runtime layer.

4.5.2 HTTP Message Structure

The mobile agent with all its constituent components is packaged as one single HTTP request and POSTed to the web servers. In HTTP, data (both request and response)

are transferred in a format based on MIME (Multipurpose Internet Mail Extensions). To achieve agent transport, we have defined an application specific MIME like content type, *application/agent*. The message is a multipart message and hence is divided into further subparts, each part containing one of the constituents of the mobile agent. A sample HTTP message is presented in the following figure.

```
POST /agents HTTP/1.1
From: analla@cise.ufl.edu
Content-Type: application/agent; boundary=abxyzf
--abxyzf
Content-Type: application/agent-attributes
Agent Name: aZIMAsagent
Source: host.cise.ufl.edu
Last Host: web.cise.ufl.edu
[Other Agent attributes]
--abxyzf
Content-Disposition: attachment; filename="Agent.class"
Content-Type: application/agent-code
[Class File follows]
--abxyzf
Content-Disposition: attachment; filename="agent_obj"
Content-Type: application/agent-state
[Agent State Follows]
--abxyzf—-
```

Figure 4-2 Agent encapsulated in a multipart MIME message.

4.6 Client Components

The client is responsible for creating and launching agents to the web server using WAPM (Web Agent Programming Model) [REN01], a programming model that is being currently developed at University of Florida. The model enables developers to build mobile agent based applications that use AZIMAS as the underlying mobile agent

system. It provides a development environment specifically geared towards building agents that need to interact with multiple users.

WAPM specifically provides:

- Agent Specification Language. To coordinate and relate multiple agents. The agent specification language is a high level language that treats agent invocations like function calls. It is a scripting language that provides a restricted but powerful set of language features to direct agent activities and relate multiple agents. The first step in developing an application is to write a script that decides the action of an agent; this script is processed by the Agent Manager.
- Pre-processor. For syntax and type checking of the script. The script written according to the Agent Specification Language is pre-processed to check for syntax and type safety before attempting to execute it.
- Agent Manager. Manages agents according to the script instructions. The Agent Manager is the implementation of the script; it creates agents and functions as the script interpreter.
- Agent Development Infrastructure. To develop collaborative application agents. WAPM provides a set of classes to build mobile agents of an interactive or non-interactive nature.

The components of the client side are illustrated in Figure 4.3 below.

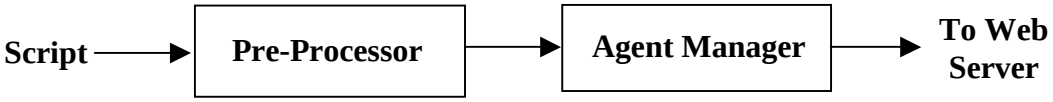

Figure 4-3 Client components in the AZIMAS system

4.7 Server Components

The web server plays an essential role in providing the infrastructure in the AZIMAS system, its job is to perform the functions of an “agent server” present in most of the other mobile agent systems. Using a web server to host agents in the Internet makes it possible to deploy agents without the inclusion of new specialized agent servers. Addition of specialized servers to the Internet is a daunting task and is difficult to achieve considering the vast scope of the Internet. It is also difficult to convince site

administrators to include new components into their existing infrastructure unless the utility of the whole concept is well established. Keeping these issues into consideration, our architecture uses existing web servers for all infrastructure requirements.

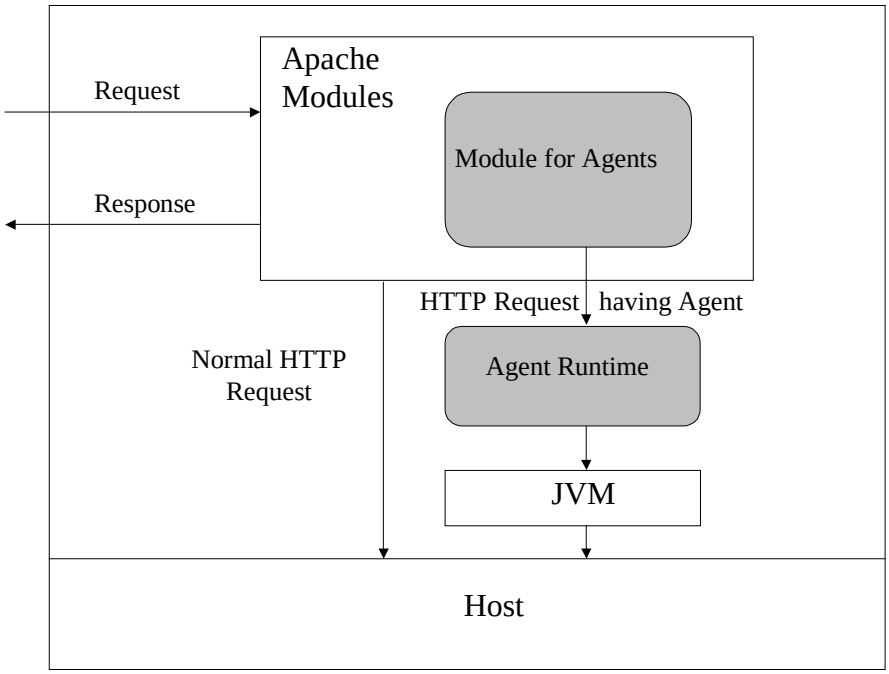

Figure 4-3 The server-side architecture of the AZIMAS system

Figure 4-3 shows the various components present in the extended Apache web server hosting AZIMAS agents. The figure shows the two components, the server extension module for agents and the runtime layer that have been added to the Apache web server. These extension modules are involved only with the processing of HTTP requests that contain mobile agents and the other non-agent HTTP requests are handled by the other existing modules in the Apache server. Using a web server to host agents raises a number of important questions that need to be addressed. The primary role of a web server is to serve web pages on the Internet and this should not be affected by assigning an additional role to it. Performance of the web server is of critical importance

and should be affected marginally if the new components are included. We carried out various simulation tests to determine the effect of the agents on the web servers. Security of the web server is also very important and it should be addressed completely. Arrival of agents should not compromise the security restrictions in the web server.

4.7.1 Apache: An Ideal Web Server

There were a number of reasons that went into the choice of Apache as the host web server for the mobile agents. The main reason for choosing Apache is its popularity and wide presence in the current scenario. Another factor that goes into the choice of Apache is that fact that it is an open source server and it is possible to add components with required functionality.

We could have decided to choose a Java based server, as it would have facilitated the development of the additional infrastructure for the mobile agents. A Java based web server was not chosen as there are inherent limitations of using one, and currently there are no commercial web servers that are completely based on Java. In addition, recent developments in the area of Apache server have seen the inclusion of a number of Java based components [JAV00]. Before describing the extensions proposed as part of AZIMAS we first present a brief introduction to the Apache working model.

4.7.2 How Apache Works

Apache is designed as a HTTP server that runs in the background as a daemon. It listens for incoming TCP/IP connections from web browsers, recognizes requests for URIs, and sends back information back to the browser. The Apache server provides various built in configuration directives that are used by the server during startup of the server and while processing requests [LAU99]. In addition to this it has a modular architecture and provides APIs that enable developers to extend the functionality of the

web server [STE99]. The web server itself is divided into a core part and additional modules, but the modules themselves provide much of the core functionality expected of a web server. Developers desiring to include new features have to write in a new module and plug it along with the other modules. The Apache web server has an API that is powerful and gives developers tremendous capabilities to tune the working of the web server. The Apache API is arranged to divide the handling of requests into a set of phases [THA96]. These phases are as follows:

- URI to filename translation
- Phases involved with access control
- Determination of MIME type of the requested entity
- Sending of data back to the client
- Logging of the request.

The server extension module can elect to handle any of these phases or can even abort processing of the request and invoke the server's error processing routine. Modules achieve this by declaring handlers that take the request as a parameter and return an integer status code that determines the success of the operation. The interface between the server code and the extension modules is through a special structure present in all the modules. This structure contains pointers to handles for various phases, or NULL, if the module elects not to handle that phase.

The server extension module can include special content handlers that are invoked to process the request at response time.

4.7.3 Apache Server Components

Apache Server, Version 1.3.3 for UNIX, is used as the development web server. The web server was installed on a Linux machine and was configured for the AZIMAS system. The web server used port 5000, rather than the default port 80 used by standard

web servers. The minimal infrastructure components added to the web server are optional and the standard functionality of the web server is not affected by including the additional components.

A special module written for the Apache Server is handed the charge for processing the agent. Control passes to the special module after the web server parses the content-type of the message and determines it is a mobile agent. The module is responsible for passing the class files and the object containing the agent state to the runtime layer. The module also goes through the agent attributes and make sure the agent meets the web server's security requirements.

4.7.4 RunTime Layer Components

The components in the runtime layer start processing the mobile agent after the web server module determines that a particular request consists of a mobile agent. The runtime layer implements the services that are provided by the host server to the visiting agent. The AZIMAS runtime layer provides the following mechanisms that are fundamental to the execution of the agents on the web servers.

- Mobility primitives for the migration of agents to web servers
- Execution of agents in a secure and controlled framework
- Secure access to server resources for visiting agents
- Logging of agent activities for monitoring purpose

The runtime layer consists of a number of components that interact with each other to provide these services. The components present in the runtime layer are shown in Figure 4.4.

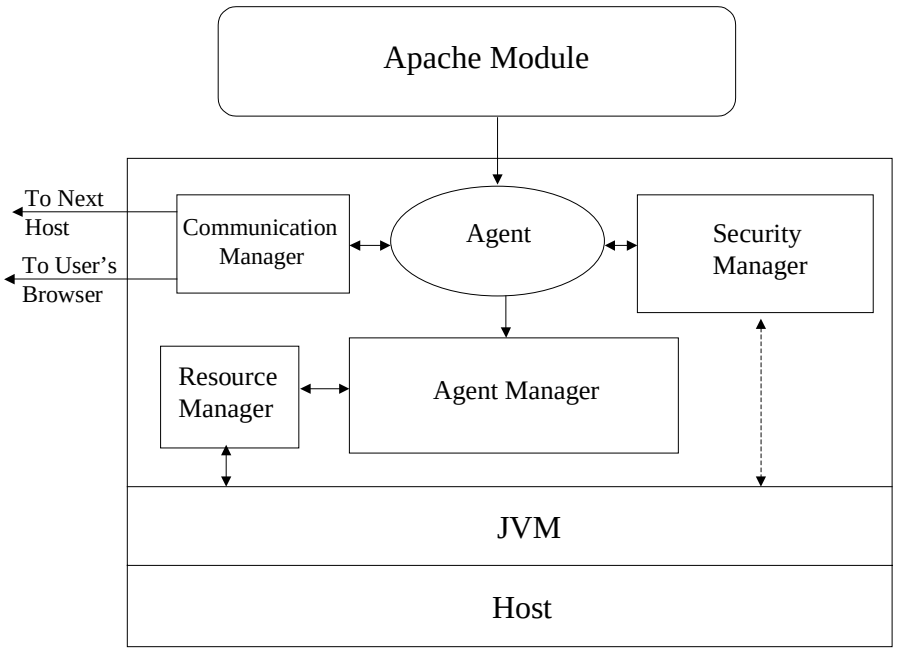

Figure 4-4 Agent Runtime Layer components

The components present in the runtime layer are *Agent Manager, Security Manager*, *Communication Manager* and a *Communication Manager*.

1. Agent Manager. This is the core components in the runtime layer and implements the services that are provided by the host server to the visiting agents. The Agent Manager starts the execution of the agent and controls the entire execution lifetime of the agents.
2. Security Manager. The Security Manager controls the resources that are accessed by the agents. It is responsible for the protection of the host from the visiting agents.
3. Communication Manager. The Communication Manager provides the mobility primitives that enable agents to move from the current web server to the next web server.
4. Resource Manager. The Resource Manager monitors the resources that are consumed by the agents. It is responsible for ensuring the availability and best performance of the system.

The runtime layer launches a thread to handle each agent, this improves the performance of the system and its scalability. The runtime layer is completely responsible

for the de-serialization of the agent before launching it and the serialization of the agent before sending it to the next server. It also logs the actions of the agents on the server. Each of the runtime layer components is described in detail in the next chapter.

CHAPTER 5
IMPLEMENTATION

The implementation of the AZIMAS system consists of two distinct components. The first component consists of the implementation of the Apache module that parses the request containing the agent and the second component consists of the runtime layer for execution of agents.

5.1 Apache Module

A new server extension module (*mod_agent.c*) was written to handle the mobile agents. This new module, written in C language can be plugged in with the other Apache modules to provide the additional functionality. To achieve its goal the module contains a content handler (*agent-exec-handler*) that is invoked by the server when processing the request containing the agent. The MIME type of the agent (*application/agent*) sets the content handler to *agent-exec-handler*. The module uses libapreq [GEN01], which is a generic library for processing client requests. This library facilitates the processing of multipart messages.

5.1.1 Handler Functionality

The handler first tries to connect to the runtime layer that can load the agent files. If the module determines that the runtime layer is not currently activated, it launches a child sub-process using the *ap_bspawn_child(pool p, int(*)(void *, child_info *), void *data, enum kill_conditions, BUFF **pipe_in, BUFF **pipe_out, BUFF **pipe_err)* API. This new process will invoke the Java Virtual Machine (JVM) to start the execution

of a Java program that listens on a specified port to listen for information from the Apache Server. This process is done only for the first agent that arrives at the server. For all future agents, the Apache server will directly contact the Java program and send the required information.

After contacting the JVM the server extension module parses the HTTP request containing the mobile and passes the agent attributes, the agent classes and the agent object to the runtime layer. The information that is passed between the two processes depends on the agent type, the details of which are given in the next section.

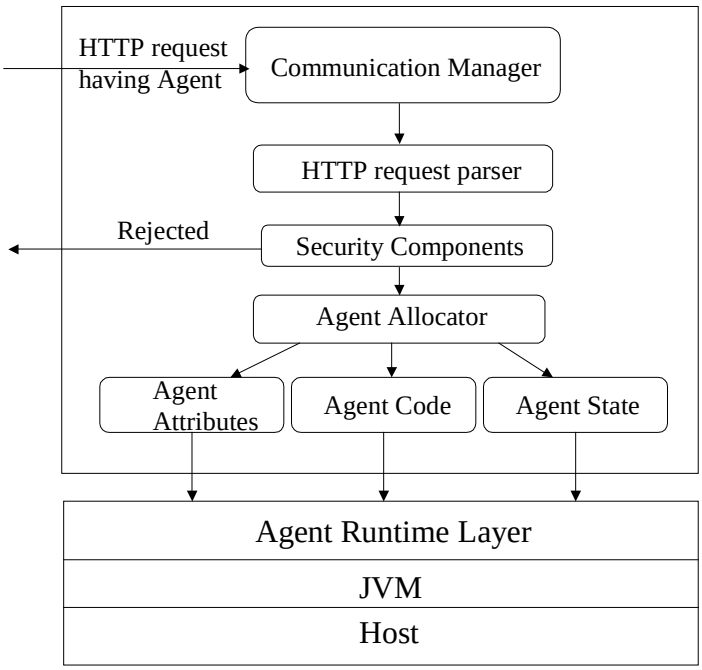

Figure 5-1 Components of the Apache server extension module and the interface with the Agent Runtime Layer

5.1.2 Apache Server – Agent Runtime Interface

The runtime layer is initially launched as a sub-process from Apache on the arrival of the agent. The launched process and Apache communicate with each other

through a TCP socket. This process is a multi-threaded process and listens on port 4444 for any information from the Apache Server. The information passed between the module and the runtime layer depends on the agent type. We had earlier classified agents as either interactive or non-interactive, and this classification distinguishes the agent loading and launching mechanisms.

Non-Interactive agents do not interact with users and are primarily concerned with processing of data that is published by the server for the agents. When the Apache web server receives a non-interactive mobile agent, it passes the attributes, class files and the object of the agent to the runtime layer directly without the need for any temporary storage. It is the responsibility of the runtime layer to process the agent class files and object, and dynamically load and launch the agent. In this case none of the agent files are stored in the web server's disk and this architecture is suitable for agents that need to move quickly to a number of servers. Such agents, when launched by the runtime layer try to search for information as per the application logic and then move on to the next server as soon as their task is completed at the current server.

Interactive agents interact with users through applets and e-mail. Such agents exist on the server for a longer time and need to move to the user's web browser for interaction with a user. When Apache receives an interactive agent it creates a temporary space for it and stores the files of the agent in it. In this case, the Apache server module parses the request and stores each of the class files and the agent object in a temporary directory. This temporary directory is created in a pre-configured location and the name of the directory is based on the name of the agent. The Apache server passes the attributes, the names of the class files and the name of the object to the run-time layer.

The runtime layer is responsible for loading only these class files and the agent object from the temporary directory. This is a security feature, as the runtime layer will never load any unwanted class files that did not come with the agent.

The actual loading and launching of agents by the runtime layers is explained in the following sections.

5.2 Runtime Layer Implementation

The runtime layer interacts with the Apache server extension module to receive the agents and then launches them in its framework. The runtime has a main thread which is its interface with the Apache module. The main thread launches a separate thread at each notification of the arrival of an agent from the Apache server. This thread interacts with the server extension module to read the contents of the agent. This thread plays the role of the Agent Manager in the framework.

The Agent Manager thread invokes the *createAgent()* function, which reads the agent attributes, agent code and agent state from the server extension module and creates a `RunningAgent` object. This object is an abstraction of the agent in the framework.

In the *createAgent()* method, the Agent Manager first reads the agent attributes and determines the future course of action. The agent name and its type are known through its attributes, the attributes also contain the encryption keys and other information that establishes the identity of the agent.

The data that is passed from the server extension module to the agent runtime layer depends on the agent type. In the case of non-interactive agents the runtime layer reads the agent attributes, the class names, the object name and the agent files directly from the server extension module. For interactive agents the class files of the agent are

retrieved from the temporary directory in which the server extension module stores the files.

In the AZIMAS system each agent is assigned a separate class loader that loads the agents classes that are received via the server extension module. This policy separates the privileges of the agent code (which is mobile) from the local system code. The system code is loaded using the default system class loader but all mobile code is loaded using the `AzimasClassLoader` object, which extends the system ClassLoader object.

Each `AzimasClassLoader` object creates a new `ClassSpace` object that forms the abstraction of a private class space for each agent. By having a separate class space for each agent the AZIMAS system avoids name clashes among classes belonging to different mobile threads. This also guarantees that a mobile agent uses only the classes that were shipped with it.

After the classes are loaded the Agent Manager uses the `AgentHandler` object to start a new thread of execution of the agent. The AgentHandler uses `AzimasObjectInputStream`, which extends the ObjectInputStream class to invoke the *readObject()* method to de-serialize the agent. The de-serialized agent is used by the Agent Manager to launch the agent thread. The Agent Manager logs the launching of each agent. It keeps track of the agents that are currently running on a particular server and adds a log entry when an agent finishes execution and decides to migrate to next server.

5.3 Agent Execution

The agents that are launched at the web server extend from the base `AzimasAgent` class and use its API to perform their tasks. The `AzimasAgent` class

uses the services provided by the web server to implement the functions as per the application logic of the agent.

Non-Interactive agents access the resources published by the web server during their execution time. These agents are primarily concerned with access to information sources available on the web server. The server launches a Non-Interactive agent and monitors the resource accesses by it. After the agent finishes its execution it requests migration to the next host by invoking its *go()* method.

Interactive agents execute on a host differently from the previous described Non-Interactive agents. During their execution, the agent decides to interact with users who can be contacted through the web server. The agent invokes its *sendApplet(String from, String to, String sub)* to contact the user whose identity is obtained through the *to* parameter of the method. The *sendApplet(String from, String to, String sub)* method calls the *sendEmail(String from String to[], String sub)* to contact the user. The e-mail contains the link to the GUI applet of the agent that is temporarily stored on the server's disk. When the user clicks the link, the agent's applet pops on the browser. The applet class loads the agent's classes and interacts with the user. After the user interaction is complete the agent goes back to the web server by invoking the *go(String dest, String[] aAttributes)* and resumes execution at the web server. The agent interacts sequentially with all the users who are accessible through the web server and processes the input from each user. After it finishes interaction with all the users at that particular web server the agent migrates to the next host to contact the remaining users in its contact list. After all the users are contacted and their input is processed the agent contacts the author of the agent via email to present all the data that has been processed by the agent.

5.4 AzimasAgent API

The AzimasAgent class provides an API that uses the primitives provided by the web server to implements its required functionality. Some of the main APIs currently provided by this class are outlined in the following sections.

5.4.1 The go() API

The class overloads the *go* method and provides two different versions of the method. The AZIMAS system uses the *go(String dest)* method to send the mobile agents from the current web server to next web server. The second version of the method is the *go(String dest, String[] attrbs)* method to send the agent back from the web browser to the web server. To move to the next web server the Interactive agents use an implementation of the *go(String dest)* method that retrieves the files from the server's disk and creates an HTTP POST request to encapsulate the agent and send it to the next web server. For Non-Interactive agents the implementation invokes a method provided by the server's infrastructure to access the contents of the agent and create an HTTP POST request.

The implementation of the *go(String dest, String attrbs[])* uses the destination web server which is obtained from the *dest* parameter to send back the agent from the web browser to the web server. The method first serializes the agent and creates an HTTP POST request that contains only the agent object. The agent uses its class files already present in the web server and the object retrieved from the POST request to resume its execution at the web server.

5.4.2 The launch() API

This method is used by the agent at the client end of the system to send itself to the first web server. The method creates an HTTP POST request and sends the agent attributes, agent code and the agent state to the first web server.

5.4.3 The mail() API

In the AZIMAS system, the Java Mail API [JAV01] is used by the agents to send e-mails to the users in the same network as the web server and to the author of the agent. The mail API uses SMTP (Simple Mail Transfer Protocol) to send e-mails and relies on the infrastructure to provide a SMTP host that can be used by it. The method first obtains a Session object using the host properties that contains the configured SMTP server. The method then uses the Session object to construct a MIME message using the various parameters like the recipient name, author name, subject and contents of the mail that are passed as parameters to the method. The Session object is used to create a Transport object, which is used to send the actual message to the recipients.

5.5 Implementation Of Security Policies

Currently the AZIMAS platform implements the following security mechanisms:

5.5.1 Agent and Server Authentication

The AZIMAS system uses the agent name as a proof of identity for an agent and currently does not have any other schemes for the authentication of agents. Simple Non-Interactive agents that visit web servers for retrieval of information and access data that is made available by the web server in the public domain do not need strict authentication and currently the system does not place any restrictions on the agents that wish to access this information. This is not sufficient for Non-Interactive agents that access resources

on the web server that are protected and are available based on identity of the agent. Similarly for certain Interactive agent applications, a name is not a sufficient proof of identity. For such applications the agent's identity needs to be signed by a trusted authority and the agent has to carry a certificate establishing its identity. This feature is currently not implemented and is among the future extensions to the AZIMAS system.

The agent is also dependent on the host to send it to the correct host in its itinerary. The host is trusted that it will not capture the agent and neither will it modify the itinerary by sending it to some other host rather than the one to which the agent has requested migration.

5.5.2 Access Control

The runtime layer depends on the security model provided by Java to implement the access control policies of the system. The system uses the default security manager [MAG01] that comes as part of the JVM and depends on the administrator of a particular server to configure the security policies to restrict the operations of an agent. The security policy configuration is done using "*policytool*", a tool that comes along with JDK (Java Development Kit). The tool can be used to specify different security policies for code coming from different sources. In the AZIMAS system the infrastructure code present in the system is given all permissions but all mobile code is only given read permission to the directory that contains the data that is made available by the web server for visiting agents. The agent is not given any other permissions, this mechanism is similar to the restrictions placed on applets by web browsers.

All mobile code is loaded by a separate class loader (`AzimasClassLoader`), and thus its privileges are different from the code that is present on the server and is loaded by the system class loader.

Concurrently executing agents cannot interfere in the execution of each other as each agent is loaded by a separate `AzimasClassLoader` that creates separate namespaces for each agent as mentioned earlier. Thus, an agent cannot reference objects belonging to other agents and also each agent uses only the class files that come as part of the agent.

5.5.3 Protection of Agents

The AZIMAS system currently does not implement any protection schemes for agents that are executing on the server. In Chapter 3 we had identified that it is difficult to protect an executing agent from its current host. Most of the agent systems do not provide any foolproof protection schemes. In Chapter 8 we outline some of the schemes that can be used to provide protection for the agents.

5.5.4 Resource Usage Policies

The system does not provide any method that would restrict the amount of resources that are used by agents currently executing on the server. Limiting the amount of resources consumed by agents and enabling resource accounting is difficult to achieve for Java based agents due to the inherent limitations of the JVM. Resource control in Java requires substantial support from native code libraries and reduces portability of the system. Due to these issues the current version of AZIMAS does not have thread level resource allocation policies and it is not possible to determine the amount of resources that are consumed by an agent. This is a challenging area of research and most of the current agent systems do not have elaborate resource allocation policies. In Chapter 8 we present approaches used by some other agent based systems [BEL00 , BIN01] and look at the possibility of adapting one of the provided solutions for the AZIMAS server. This would enable the server to implement negotiation policies that restrict the amount of

resources that are used by an agent. It would also enable the server to terminate agents that consume excessive amount of resources. This would guarantee an optimum performance of the server and prevent any “denial of service” attacks.

CHAPTER 6
PERFORMANCE EVALUATION

6.1 Introduction

The goal of the simulation experiments was to observe the effect on the performance of the web server after the inclusion of the AZIMAS extension module. The experiments were carried out by using a web server performance-measuring tool, httperf [MOS01] in conjunction with a set of tools that were developed for sending the AZIMAS agents to the test web server. The metrics used for testing the performance of a web server include *the response time, the response* rate and *the saturation level* of the web server. The *response time* gives a basic indication of the performance of the web server and measures the time taken by a server to send a response for a request. The time difference between sending of the first byte and receiving of the first byte by the client is measured. The *request rate* and the *response rate* test the saturation level of the server and are an indicator of the number of requests processed by the server and the number of replies sent by the server for a requested rate. In addition, the agents were benchmarked by sending agents of different sizes and resource requirements to the web server.

6.2 Experiment Setup

6.2.1 Web Server Performance Measuring Tools

Apache comes with a default *apachebench* utility to measure the performance of a web server. It can be used to obtain the performance of a web server but the statistics obtained are not as reliable as those obtained from httperf. httperf is a tool for measuring

the performance of a web server by generating various HTTP workloads and measuring various statistics. The performance tests were carried out by using httperf as a command line tool that generates HTTP requests at a varying rate. Figure 6.1 show a sample invocation of the httperf tool to measure the performance of the web server.

httperf –server 10.3.1.11 **–port** 5000 **–num-conns** 1000 **–rate** 10 **–timeout** 5

- **server** (the test web server)
- **port** (port for HTTP Daemon)
- **num-conns** (number of connections to be made)
- **rate** (request rate in requests/sec)
- **timeout** (client timeout in ms)

Figure 6-1 Httperf invocation

The server's IP address is 10.3.1.11 (tampa.harris.cise.ufl.edu) and the web server is listening on port 5000 for incoming requests. For this particular invocation, 1000 requests are sent at the rate of 10 requests per second. The timeout period for each request is set at 5 milliseconds.

There are six groups of results that are obtained: overall results, connection results, results relating to the issuing of HTTP requests, results relating to the replies received from the web server, miscellaneous results relating to the CPU time and network bandwidth used, and a summary or errors encountered[1]. For measuring the performance of the test web server, the response time and the reply rate of the web server were noted as part of the experiments. The response time is a direct indicator of the performance of the web server as it gives the time difference between the first byte that is sent as part of

[1] Information obtained from the "README" file present in the httperf tool.

the request and the first byte that is received as part of the response. The reply rate is used to measure the saturation point of the web server by comparing it with the request rate. The web server reaches saturation point if it is not able to send a reply to all the requests that are received by it and it drops some of the requests.

6.2.2 Agent Launcher

A separate set of utility tools have been developed that can be used to launch mobile agents to the web server at a varying rate. These tools are capable of sending both, Interactive and Non-Interactive agents at a varying rate to the web server. The tool takes as parameters the number of agents to be sent and the rate at which these agents are to be sent. Figure 6.2 shows the setup for the experiments. The experiments were carried out by simultaneously POSTing simple HTTP requests and HTTP requests containing AZIMAS agents.

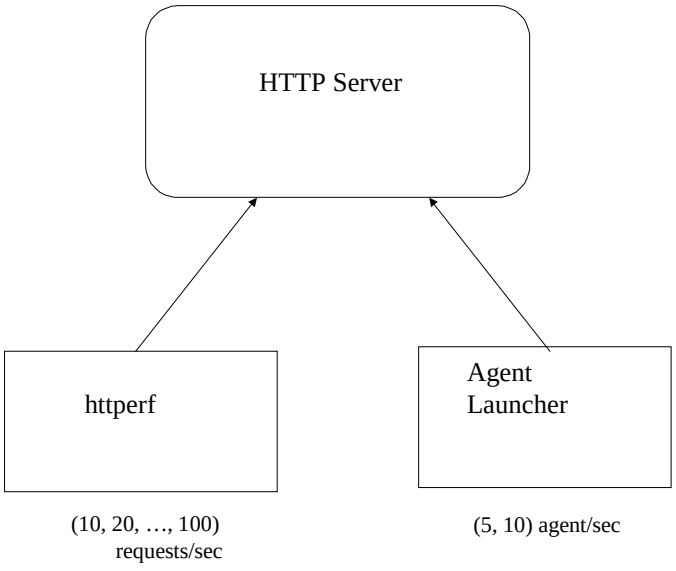

Figure 6-2 Experiment setup

6.3 Simulation Results

6.3.1 Response Time

Figure 6-3 plots a graph of the Response Time (of web server) vs. HTTP request rate and shows the affect on the response time of the web server when agents are sent to it at a rate of 5 agents/sec and 10 agents/sec. Non-Interactive agents have been used for this set of results. The experiments test the overhead of the server extension module and the runtime layer when the web server processes HTTP requests containing agents. The agents that are used for these experiments do not consume additional resources like memory and CPU on the server. In the base (no agents) case, HTTP requests were sent at a rate ranging from 10 to 90 request/sec. The second case is constructed by augmenting the base case by AZIMAS agents sent at a constant rate of 5 agents/sec. The third case is similar to the second case with the agents being sent at a rate of 10 agents/sec.

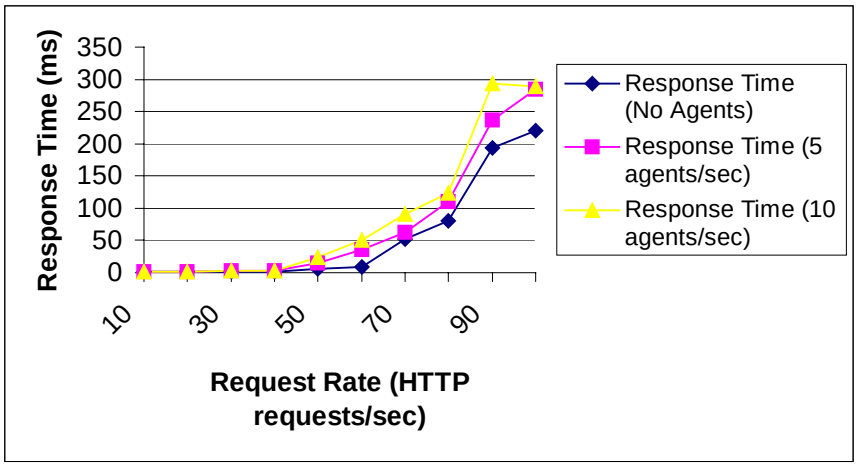

Figure 6-3 Response time of web servers under varying load of HTTP requests and agents

The graph shows that the response time of the web server increases when agents are sent to the server. This is expected, as the web server has to process each agent

request in addition to the normal HTTP request. The important observation is that the response time of the web server does not increase by a large value due to the presence of the agents (specially for request rates below 50 requests/sec). Thus the presence of the AZIMAS system does not affect the basic performance of the web server.

6.3.2 Saturation Level Of Web Server

Figure 6-4 Requests served by web server

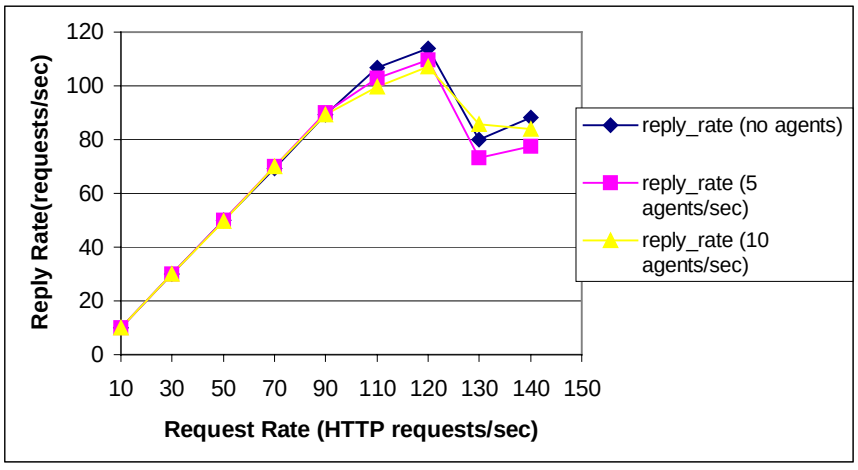

Figure 6-5 Replies sent by the web server

Figures 6-4 and 6-5 show the affect of the AZIMAS extensions on the saturation level of the web server. The saturation level of the test web server was reached for 130 requests/sec for normal HTTP requests with no agents being sent simultaneously. Simulation results with agents arriving at a varying rate show that the saturation level of the web server is not affected due to the presence of the agents on the system. The web server reaches its saturation level at around 110 to 120 requests/sec which is a little earlier than the normal saturation point. The difference is due to the presence of the agents but this can be attributed to additional load of the agents on the server.

6.3.3 Agent Benchmarking

The effect on the performance of the web server under the load imposed by agents with different memory and CPU resource consumption is presented in this section. In addition we also compare the performance effect of Interactive agents against Non-Interactive agents.

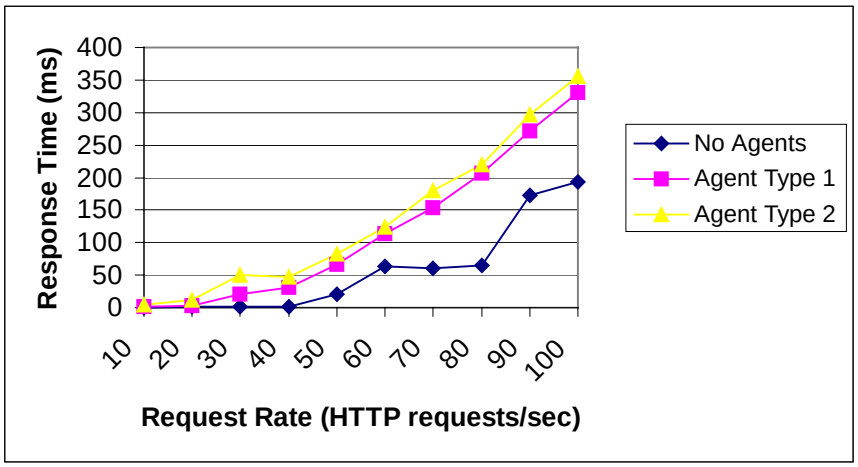

Figure 6-6 Response Time vs. Request Rate (load on web server by agents with different resource consumption)

Figure 6-6 shows the affect on the web server response time when agents with varying resource requirements are sent. Agent Type 1 refers to simple agents with very limited additional resource consumption on the server and these are primarily used to test the overhead of the extensions that have been proposed by us, comparatively Type 2 Agents are bulkier agents that consume both memory and CPU resources on the host server. These agents allocate memory on the server and also used the CPU processing time to perform computations like calculating random numbers and storing them in the allocated memory and also accessed resources like files that are accessible on the web server. The response time does not increase by a large value even when resource-consuming agents are sent because the runtime layer handling the agents does not interfere with the working of the other Apache modules handling non-agent HTTP requests. But there is an increase in the response time as overall system resources are used by the agents.

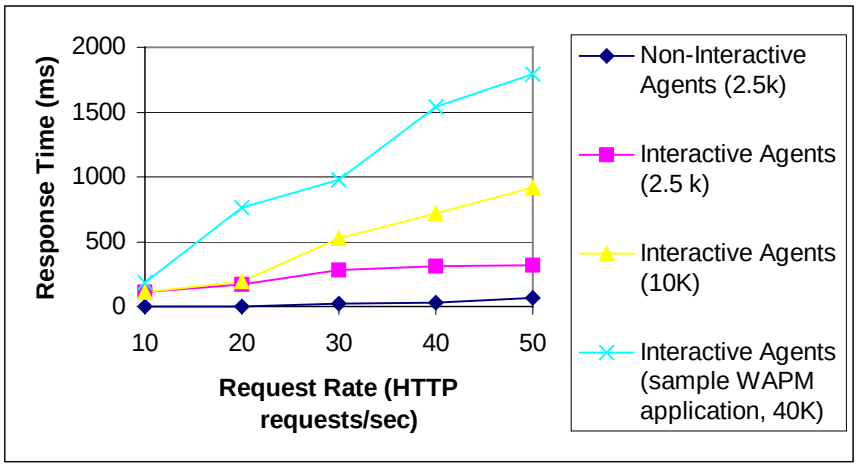

Figure 6-7 Comparison of response time (Interactive Agents and Non-Interactive Agents)

Figure 6-7 compares the effect on the response time of the web server when hit by Interactive agents of different sizes and Non-Interactive agents. The results show a large

difference in the effect on the response time of the web server between these two agent categories, which can be attributed to a number of factors, the main being the difference in size between the agents. We observe that the web server is sensitive to the size of an Interactive Agent. The graph shows a steep increase in the response time of the web server when hit by agents that are part of a complete application developed using the WAPM model. This sample application had 20 class files and additional files like html files used by the agent and the serialized agent. The total size of all the files that were shipped with agent was approximately 40K bytes. The results have prompted us to look into optimization of the WAPM model that would reduce the size of the classes that are shipped as part of the agent. We expect the final size of agent-based applications developed using WAPM to be smaller than the application used for testing purpose. We compared the overhead posed by Interactive agents of different sizes and noted better performance for Interactive agents of smaller size.

But the difference in performance for processing Non-Interactive agents and Interactive agents is considerable. This is because the Interactive agents are first stored in the local disk and later dynamically loaded from it during execution. Storing of files to the disk affects the processing time of the request and hence influences the overall performance of the web server.

6.4 Performance Analysis

The experimental results provide an initial indication of the performance implications of including the AZIMAS extensions in a web server. The overhead imposed by the agents and the server infrastructure used to launch and execute the agents is obtained through the above mentioned experiments.

We observe that the overhead of the extensions is minimal and the additional overhead of the currently executing agents does not affect the performance of the web server in a drastic manner.

The results also show that the overhead of executing Interactive agents is considerable due to the size of these agents and also the difference in the processing sequence of these agents at the web server. This requires us to improve the WAPM model to reduce the size of the Interactive agents and also come up with alternate and better schemes at the server end that would reduce the load of these agents on the server.

CHAPTER 7
APPLICATION SCENARIOS

To demonstrate the capabilities and features provided by the AZIMAS system we have developed a suite of applications. As the WAPM model further evolves we will have a rich collection of applications that use the features provided by the AZIMAS system. This chapter describes a sample agent based application that uses the functionality provided by the AZIMAS system. The application is a "Thesis Defense Scheduler" and it has been built using the WAPM programming model and is an example of an *Interactive* Agent Application.

7.1 Application Interface

The first step while using the tool is to create an agent using the user interface provided by the scheduling tool. A student who wishes to schedule his or her thesis defense date with the committee members invokes the application, which provides a user-interface to create an agent that contacts the committee members. Figure 7.1 below shows the user interface of the application.

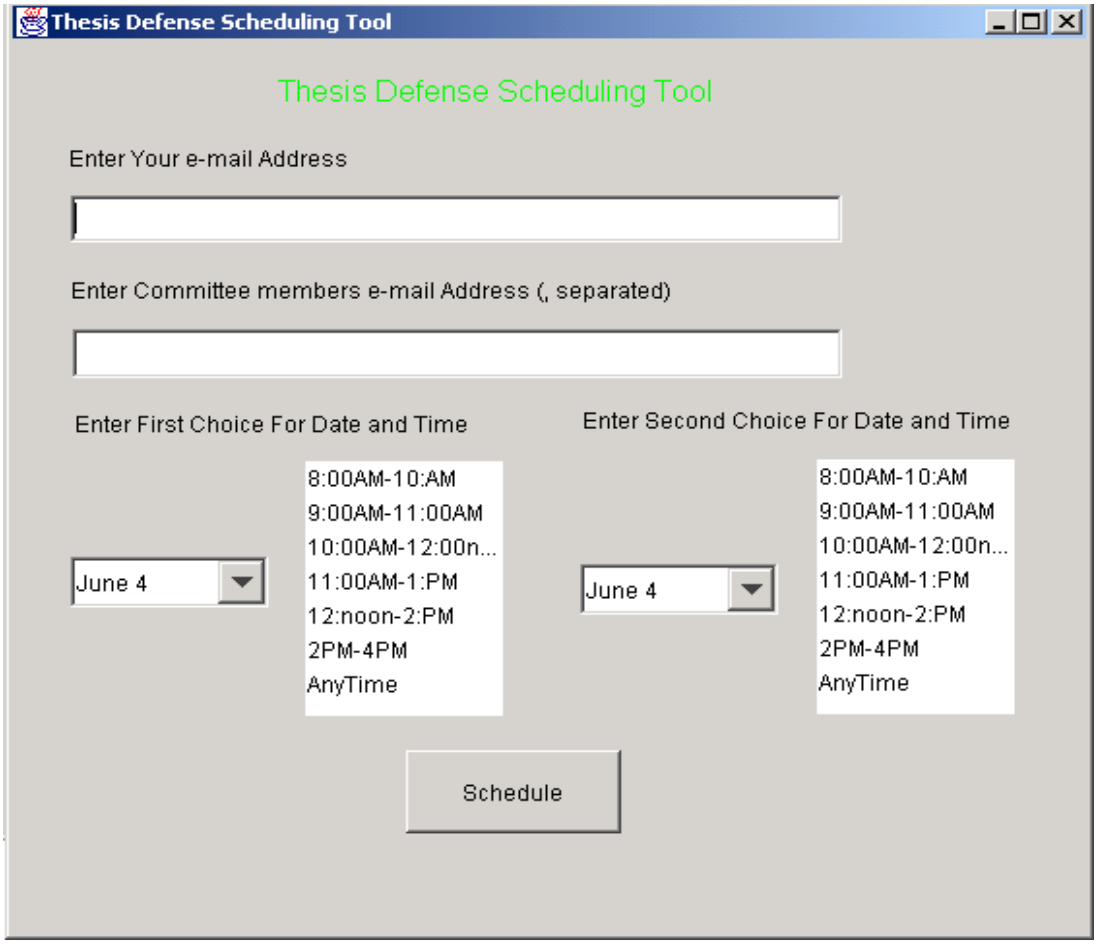

Figure 7-1 User interface for "Thesis Defense Scheduling Tool"

The student enters her e-mail address and the e-mail addresses of the committee members and selects two choices for the date and times that are convenient for her. The application creates a script from the various options configured by the user. The generated script is used by the WAPM Agent Manager to create an agent that is launched to the web server.

7.2 Agent Interface

The web server starts the execution of the agent that arrives on the web server. The agent sends e-mail, which contains a hyperlink to the agent's applet that is

temporarily residing in the web server to the list of users configured by the application. The e-mail contains a hyperlink to the agent's applet that is temporarily residing in the web server. Figure 7.2 below shows the agent's GUI applet.

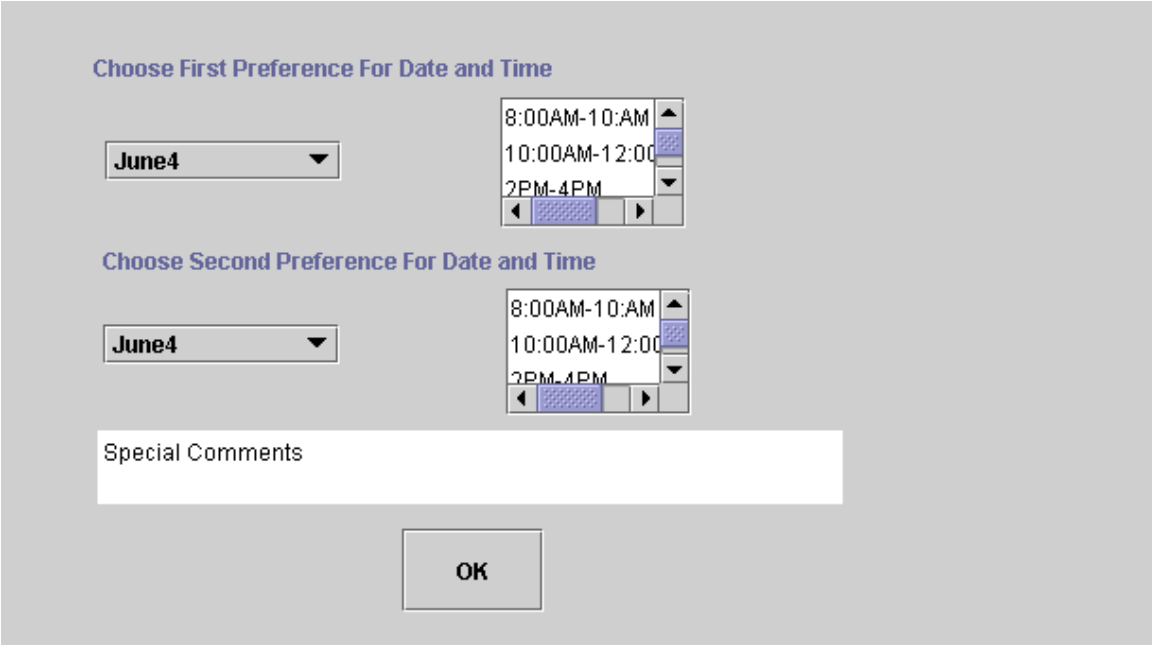

Figure 7-2 Scheduler Agent's applet interface

Clicking on the hyperlink displays the agent's applet that is shown in the above figure. This also brings in the agent to the web browser of the user. The user interacts with the agent and on clicking the OK button the agent is serialized and sent back to the web server. The agent again resumes its execution at the web server with the input that is provided by the user. After all the users are contacted the agent processes the input and sends the time-preferences of each of the committee members to the student who has deployed the agent.

The application that has been described above is a sample application developed using the powerful features of AZIMAS. It shows the entire cycle of creating agents

using the WAPM model and execution of the agents on the AZIMAS infrastructure. It also shows the ability to build Interactive-Agents that facilitate communication between different users through familiar interfaces like e-mail and the web browser. The system facilitates development of similar applications and other applications with a rich set of features that enable collaboration and sharing of information between multiple users.

CHAPTER 8
CONCLUSION AND FUTURE WORK

8.1 Current Status

We have described AZIMAS, a platform for enabling mobile agents on the Internet. The platform uses existing systems to create an infrastructure that enables deployment of mobile agents on the Internet. This has been achieved by making extensions to existing Apache web servers by making them capable of hosting mobile agents. As part of the infrastructure we have developed a server extension module for Apache that can be plugged in with the other existing Apache modules. A runtime environment for the execution of the agents at each web server has also been developed.

We have implemented security models that are based on the security requirements of a mobile agent system. Currently we provide schemes for protection of the host from the agents and also protection of the agents from each other while executing on a host machine. The system has provisions for elementary authentication of agents and restriction on the agents that visit a particular host.We also show that the presence of the mobile agents does not affect the performance of the web server in a drastic manner.

To demonstrate the power of the AZIMAS system we have built sample agent based collaboration applications and applications that use agents for information gathering and filtering on the Internet.

Parallel work is being currently done to develop a full agent-programming model that will enable users to create agents in a simple yet powerful scripting language and thus create agent based applications that are based on the AZIMAS system.

This thesis has contributed the following salient features and facilities in the AZIMAS system:

- Design and development of a server platform for the execution of agents on Apache web servers
- Performance study of the effect of deploying mobile on the developed system
- Preliminary security models for implementing security requirements
- Sample agent based applications that use the AZIMAS programming primitives and the AZIMAS infrastructure

8.2 Future Work

The development of a server platform for the execution of agents is the first but a crucial step in the development of a mobile agent system. The server provides the primitives and services for the mobile agents; currently AZIMAS provides primitives for agent mobility and agent interaction with users and services that enable agents to access resources on the web server. Some of the additions and improvements that have been identified are presented below.

8.2.1 Platform Extensions

The platform architecture has to be extended to support inter-agent communication. Provision for communication between agents is an invaluable feature for any mobile agent platform. This extension will enable the deployment of agents that interact with other agents present in the web server. The server also needs to provide further services that will enable mobile agents to be used as web-crawlers. Specifically, issues pertaining to processing of the data by an agent needs to be elaborated.

8.2.2 Security Model Extensions

The current security implementation is preliminary and insufficient for an agent system present on the Internet. The current security model needs to be extended to include the following features

- Protection of Agents. The AZIMAS system does not have any encryption policies for the protection of the agent. The agent needs to be encrypted with the public key of the next host before sending, as this will protect the agent from any unwanted interceptions and external attacks. This requires implementation of an encryption policy that is based on public key cryptography schemes. Additional schemes need to be designed that will protect an agent from the host where it is executing currently. This is difficult to achieve and there are no standard methods to guarantee the protection of an agent from the host and currently this is an active area of research. Future work would involve weighing the available options and solutions and choosing a standard scheme that fits well with the existing AZIMAS platform and also achieves the protection of agents.
- Protection of Host. The current security manager in the server needs to be extended to provide resource control policies with finer granularity of access to the system. Legitimate agents must be given access to protected resources through the use of authorization mechanisms that provide restricted access rights for agents.
- Authentication Policies. The AZIMAS system does not authenticate the source of the agents and neither does it verify the identity of the previous host that has sent the agent. Any full-fledged mobile agent system should enable the servers to authenticate the agents that are visiting the server. The AZIMAS server should be extended to implement authorization schemes that decide on agent capabilities depending on the identity of the agent.

8.2.3 Denial-Of-Service Attacks

The AZIMAS system does not prevent denial-of-service attacks by the agents. We are currently exploring some agent system that provide monitoring and resource control. SOMA (Secure and Open Mobile Agent) [BEL00] provides monitoring and control of agents through the use of JVMPI (JVM Profiler Interface) [SUN01] and JNI (Java Native Interface) libraries. J-SEAL2 [BIN01] is a mobile agent kernel that is resource-aware and prevents denial of service of attacks by agents. Another scheme to implement resource control in a Java based platform is through modification to the JVM. [BAC00] presents research systems that have been built to enable monitor the amount of memory and CPU resources consumed by a Java program. Some of the above mentioned systems could be incorporated into the AZIMAS system to enable resource accounting and control.

8.3 Applications For AZIMAS

The full power of the AZIMAS platform can only be realized when applications that use the services provided by the platform are developed. Research is being currently done in the Harris Lab at University of Florida to develop a programming model for developers to develop agent-based applications for the AZIMAS system [REN01].

Future extensions to the platform should enable effective means of communication between agents and users. Currently we are using e-mail and applets, the performance and the security implications of using these methods needs to be explored further.

The current platform is the starting point for a well-defined and effective platform for mobile agents on the Internet. The work in thesis forms the root of a system that will turn the dream of “Internet agents” a reality.

LIST OF REFERENCES

[APA01] Apache Web Server Project, http://httpd.apache.org, May 2001

[ARI98] Aridor, Y., Oshima, M., Infrastructure for Mobile Agents: Requirements and Design, In Proceeding of the 2nd Int. Workshop on Mobile Agents, LNCS 1477, volume 1477, 1998, pages 38—49

[BAC00] Back, G. Techniques for the Design of Java Operating Systems, Proceedings of the 2000 USENIX Annual Technical Conference, San Diego, USA, June 2000

[BEL00] Bellavista, P., Corradi, A., and Stefanelli, C., Monitor and Control of Mobile Agent Applications, ACM OOPSLA Workshop on Experiences with Autonomous Mobile Objects and Agent Based Systems, Minneapolis, USA, Oct. 2000.

[BER99] Berners-Lee, T., Fielding, R., Frystyk, H., Hypertext Transfer Protocol – HTTP/1.1, RFC 2616, June 1999.

[BIN01] Binder, W., Hulaas, G., J., and Villazon, A. Resource Control in J-SEAL2, Cahier du CUI No 124, Technical Report, March, 2001

[CON01] Concordia White Paper, http://www.meitca.com/HSL/Projects/Concordia/MobileAgentsWhitePaper.html, May 2001

[GEN01] Generic Apache Request Library, http://httpd.apache.org/dist/httpd/libapreq-0.31.readme, February 2001

[GRA97] Gray, R., Cybenko, G., Kotz, D., Rus, D., Agent Tcl, In William Cockayne and Michael Zyda, editors, Mobile Agents: Explanations and Examples, chapter 4, Manning Publishing, 1997. Imprints by Manning Publishing and Prentice Hall.

[JAV00] Java Apache Project, http://java.apache.org, December 2000

[JAV01] Java Mail API, http://java.sun.com/products/javamail/JavaMail-1.2.pdf, April 2001

[JIG01] Jigsaw – The W3C's server, http://www.w3.org/Jigsaw/, May 2001

[KAR98] Karnik, M, N., Tripathi, R, Anand., Design Issues in Mobile-Agent Programming Systems, IEEE Concurrency, July-Sep 1998, Page 52-61

[LAN96] Lange, D., Oshima, M., Aglets Software Development Kit, http://www.trl.ibm.com/aglets/, October 2000

[LAN97] Lange, D., Aridor, Y., Agent Transfer Protocol (ATP), Draft 4, March 19, 1997, http://www.trl.ibm.com/aglets/atp/atp.htm, May 2001.

[LAU99] Laurie, B., and Laurie, P., Apache – The Definitive Guide, Second Edition, O'Reilly

[LIN95] Lingnau, A., Drobnik, O., and Domel, P., An HTTP-based Infrastructure for Mobile Agents, WWW Journal – 4th International WWW Conference Proceedings, Boston, MA, Dec 11-14, 1995

[MAG01] Magelang Institute, Fundamentals of Java Security, http://developer.java.sun.com/developer/onlineTraining/Security/Fundamentals/index.html, November, 1998

[MOS01] Mosberger D, Jin T., httperf – A Tool for Measuring Web Server Performance, http://www.hpl.hp.com/personal/David_Mosberger/httperf/, April 2001

[OSH98] Oshima, M., Karjoth, G., and Ono, K., Aglets Specification 1.1 Draft, Draft 0.65, September, 8th, 1998

[REN01] Renganarayanan, V., Nalla, A., and Helal, A., Internet Agents for Effective Collaboration, Submitted to 5th IEEE International Conference on Mobile Agents, (MA 2001), Atlanta, USA, Dec 2-4, 2001. Available at http://www.harris.cise.ufl.edu/projects/publications/InternetAgents.pdf

[SCH99] Schelderup, K., Olnes, J., Mobile Agent Security – Issues and Directions, In H. Zuidweg, M. Campolargo, J. Delgado, A. Mullery (Eds.): Intelligence in Services and Networks. Paving the Way for an Open Service Market, Proceedings of the 6th International Conference on Intelligence in Services and Networks (IS&N'99), Springer-Verlag, LNCS 1597, pp.155-167, 1999

[STE98] Stefan, F., Integrating Java-based Mobile Agents into Web Servers under Security Concerns, Proceedings of 31st Hawaii International Conference on System Sciences (HICSS 31), Kona, Hawaii, January 6-9, 1998, pp 34-43

[STE99] Stein, L., and MacEachern, D., Writing Apache Modules with Perl and C, First Edition, O'Reilly

[SUN01] Sun Microsystems – Java Virtual Machine Profiler Interface (JVMPI), http://java.sun.com/products/jdk/1.2/docs/guide/jvmpi/jvmpi.html, March 2001

[TAC01] Tacoma - Operating System support for Mobile Agents, http://www.tacoma.cs.uit.no/index.html, May 2001

[THA96] Thau, R., Design considerations for the Apache Server API, Fifth International World Wide Conference May 6-10, 1996, Paris, France

[TRI99] Tripathi, R, A., Karnik, M, N., Vora, K, M., Ahmed, T., and Singh, D, R., Ajanta – A Mobile Agent Programming System, Revised version of Technical Report #TR98-016, Department of Computer Science, University of Minnesota, April 1999.

[TSC99] Tschudin, C., Mobile Agent Security. In: Matthias Klusch (Ed.): Intelligent information agents: agent based information discovery and management in the Internet, pp. 431 - 446, Springer-Verlag. 1999, http://mole.informatik.uni-stuttgart.de/security.html, May 2001

[WHI96] White, E., J., Telescript technology: Mobile agents. General Magic white paper. In J. Bradshaw, editor, Software Agents. AAAI/MIT Press, 1996.

[WON99] Wong, D., Noemi, P., and Dana, M., Java-based Mobile Agents, Communications of the ACM, Volume 42, Issue 3, 1999, Page 92

BIOGRAPHICAL SKETCH

Amar Nalla is native of Ranchi, India. Born on March 14, 1977 he obtained his bachelor's degree in Computer Science from the University of Roorkee (now IIT Roorkee), India in 1999. After finishing his undergraduate studies, Amar decided to pursue higher studies and was admitted in the MS program in Computer Science at the University of Florida. He was a teaching assistant for the class, "Computers and Modern Society" during his stay at University of Florida.

He is interested in programming and likes to spend to a lot of time in front of his computer. After completing his master's degree, Amar will explore the Pacific Northwest area and hopes to do something useful for Microsoft Corporation in Redmond, Washington.

Amar likes reading fiction and occasionally indulges in adventure sports like trekking and mountain climbing.