aZIMAs – almost Zero Infrastructure Mobile Agent System

Amar Nalla, Abdelsalam (Sumi) Helal and Vidya Renganarayanan

Department of Computer & Information Science & Engineering
University of Florida, Gainesville, FL 32611-6120, USA

helal@cise.ufl.edu

Abstract

Mobile agents promise to bring in a new era in the field of World Wide Web and Internet Computing. Although mobile agents have been around for sometime their full potential has not been realized due to the lack of a suitable infrastructure that would allow for their deployment and seamless integration on the Internet. In this paper we propose a system based on HTTP and existing web server that facilitates the deployment of Java based mobile agents on the Internet. By requiring no more than web servers as the mobile agent's environment, it will be possible to overcome the current hurdles and make mobile agents a pervasive model for internet computing. In this paper, we present aZIMAs, a simple mobile agent environment and a potentially ubiquitous server platform.

Keywords: *Mobile Agents, Web Agents*

1. Introduction

There has been a significant body of research and interest in the area of mobile agents for quite some years. However, their predicted emergence as one of the key future technologies for the Internet has not materialized due to a number of reasons. Although many different mobile agent systems have been developed, they have not been deployed on the World Wide Web on a large scale due to 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 that mobile agents may visit. This presented an administrative overhead, and more seriously, an interoperability nightmare given the lack of standards for agent systems. Hence, the agent server requirement seriously hindered the deployment of mobile agents on the Internet. In this paper we present aZIMAs (almost Zero Infrastructure Mobile Agents system), which is a mobile agent platform based on existing Apache[18] web servers and the HTTP protocol [3]. Our architecture does not require the presence of an "agent server" at each host and enables the deployment of mobile agents on the Internet with almost zero additional infrastructure.

We have extended the Apache web server to implement aZIMAs. The extended Apache web servers act as hosts for the mobile agents developed and launched by a user using the programming model provided at the client end of the system. The web server present at a host is responsible for accepting the agents, performing authentication and authorization of the visiting agents, and launching the agents for execution and provides mobility services for the agents.

The agents are written in Java and they execute in a runtime layer that is part of the web server. We have decided to choose Java because of its suitability and popularity as an agent programming language. Some of the other Java based Agent systems are the Aglets[1], General Magic's Odyssey [7] and University of Stuttgart's Mole [14]. A popular non-Java based agent system is Agent Tcl [9], developed at Dartmouth.

To achieve widespread deployment of mobile agents on the Internet it is essential that HTTP be chosen as the transport protocol for the agents. In addition to fulfilling its initial purpose of transferring web pages it is also being used for various kinds of data transfer like delivering of web services and uploading of documents. Some other agent systems, notably Aglets have proposed the idea of having a separate Agent Transport Protocol (ATP) [12], which is a more suitable protocol for agent mobility but the need for another protocol in addition to HTTP has discouraged the use of such systems on the Internet.

Although it could at a first glance be argued that, from a performance point of view, HTTP would not be the most suitable mechanism for transferring mobile agents, our experience was to the contrary of this argument. We found it feasible and easy to achieve agent migration from server to server and thus provide mobility for agents using HTTP. We have used Apache, which is an open source web server that is common on the Internet and is composed of an extensible architecture having various modules that interact with each other to provide the necessary functionality. We did not develop our own custom web server and decided to choose Apache as it a heavily deployed web server on the Internet and it would be easy to incorporate our infrastructure on the existing systems. Developing a separate agent server or web server would be counter-productive as it would be very difficult to deploy it in the real world. This is quite evident by the fact that none of the agent systems developed till now is prevalent on the Internet.

The architecture enables users to develop useful and effective Internet Agent based applications that can be easily integrated with the Internet. Some of the sample agent applications targeted by this architecture include web based collaboration agents and information filtering agents.

The focus of this paper is to present the architecture and implementation of the web server infrastructure for the agents. We have also developed an API for developing agents. The details of the programming model to develop agents for this infrastructure and sample applications can be found in [16]. The agent system that has been currently developed is relatively simple and issues like cloning of agents and inter-agent communication are not addressed by the system.

In the following section, we present an introduction to the aZIMAs agents and describe the services provided by our server platform for these agents. Section 3 presents the architecture of the aZIMAs system followed by the implementation details in Section 4. Section 5 presents a detailed analysis of performance and scalability characterization obtained by deploying the aZIMAs system on a test-bed of several web servers. Section 6 describes sample applications that have been built to exploit the power of the aZIMAs system. In section 7 we present some of the related work and section 8 concludes with a description of some of the planned future extensions to the aZIMAs system.

2. AZIMAS AGENTS

The aZIMAs platform provides services for two types of agents: *Non-Interactive* and *Interactive* agents.

- *Non-Interactive* agents are primarily concerned with information sources available at different hosts on the Internet. These agents are suitable for information search and filtering applications. These agents can be considered as mobile web crawlers.
- *Interactive* agents collaborate through e-mail with users housed in the same network of the web server. They have a highly programmable applet-based user interface built into them. These agents are used to enable collaboration-based applications that require interactions among multiple (and possibly a large number of) users and also access to information sources available on the Internet.

2.1 Infrastructure Provided by aZIMAs Server

The aZIMAs server platform provides a runtime layer that is built into each web server. It provides basic mobility services by enabling an agent to move at will from one web server to another, encapsulated in an HTTP POST request. The platform provides access to resources made available by the web server. It also enables the agents to communicate with users accessible through the current host. It also provides primitives that enable clients to keep track of the agents that they launch.

2.2 Security Requirements of aZIMAs Server

The infrastructure needs to provide a robust security model that addresses various security issues when mobile agents are deployed on the web. There are three issues regarding any secure mobile agent system [11]. These are:

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

2.2.1 Protection of Agents

The agent code needs to be protected from hostile servers and other agents. The aZIMAs system protects an agent running on a server from other agents as it creates a separate namespace for the execution of each agent. The system currently does not provide other protection schemes for agents from the host machine but has provisions to detect if an agent has been tampered.

2.2.2 Agent and Server Authentication

In the aZIMAs system, the agent depends on the current server to send it correctly to the next server. Thus a scenario is possible in which a server captures an agent and prevents it from moving to the next server in its itinerary or forwards it to a malicious host. This is achieved by keeping track of the progress of an aZIMAs agent in its itinerary.

2.2.3 Authorization and Access Control

The aZIMAs server relies on Java's security manager to restrict the operations of the currently executing mobile agent. The server also uses Apache's security features to restrict the access of the mobile agent on the server.

In addition to restricting access, the server needs to keep track of the amount of resources that are being used by the agents. The aZIMAs system currently does not keep account of resources consumed by each agent as this imposes a considerable load on the infrastructure and is not easy to achieve due to limitations in the JVM.

3. ARCHITECTURE OF THE AZIMAS SERVER

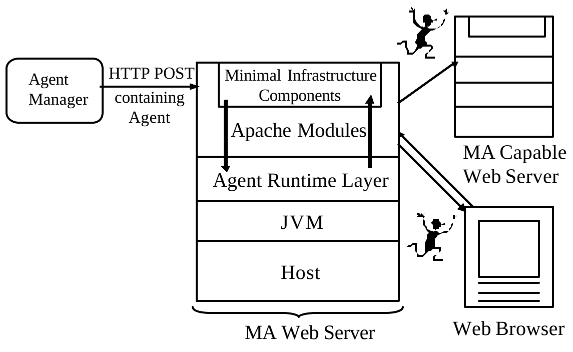

Figure 1. Architecture of aZIMAs server platform

Figure 1 shows the overall architecture consisting of both the client and server components of the aZIMAs system. The various components present in the web server are also illustrated. We will describe each of the components present in the server and correlate them to the infrastructure requirements of a machine that is capable of hosting mobile agents. First, we will briefly describe the client present in the system followed by a detailed description of the server side components.

3.1 Client Components

We have developed a programming model called WAPM (Web Agent Programming Model) [16] that enables users to rapidly develop agent-based applications that uses the aZIMAs system as the underlying platform. As part of the model, we have developed a simple scripting language used to direct and define the agent actions, a pre-processor to check the validity of the scripts, an agent manager that creates and manages agents according to the script. We have also defined a class hierarchy for developing application agents in WAPM.

3.2 Server Components

The server side infrastructure consists of two components, an Apache module and an agent runtime layer. The module written for Apache reads the multipart HTTP request containing the agent. The module first parses the HTTP request into its constituent subparts. The subparts consist of the agent attributes, the agent code and the object having the agent state. The module reads the attributes of the agent to decide on the future course of action. The Apache server can be configured to reject agents having certain names, or coming from certain hosts. If the agent falls into any of the above categories it is rejected otherwise the agent information is passed to the runtime environment. The runtime environment is actually responsible for loading the agent class files and launching the agent on the server. The runtime environment can first decide to go through the agent attributes to retrieve encryption keys or any other required information before loading the classes. After the agent classes are loaded, an agent execution thread is instantiated using the agent state. Thus, functionality needed by a machine to host mobile agents is provided by extending the web server by adding a server extension module and adding a runtime layer to execute agents. The web server functions like any other web server for normal HTTP requests but when an agent arrives, the extension module that is part of the infrastructure takes over and provides support for the agents. Figure 2 shows the design of the web server in the aZIMAs system. The gray colored components show the extensions that have been made to the web server.

This architecture is similar to the Servelet engine that is present in an Apache web server that is capable of hosting Java Servelets. Hence the presence of a separate runtime layer does not impose a heavy addition to the web server infrastructure. It can be easily incorporated into existing web servers by web administrators just as they include additional infrastructure to enable servelets.

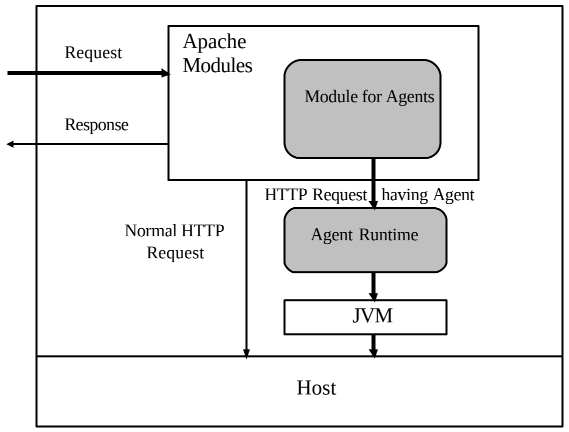

Figure 2. MA capable web server

3.3 Agent Request Structure

Each agent request consists of three parts, these are *agent attributes, agent code* and *agent state*. In the next section we will be elaborating on the agent attributes and their use by the extended web server to implement its security policies. The agent code consists of the Java bytecode that comprises the agent and is needed at each server for execution. This is the static portion of the agent, tampering of which is detected by hosts. The agent state is the serialized form representing the state of an executing agent. A sample HTTP request containing an aZIMAs agent is shown in Figure 3.

```
POST /agents /HTTP 1.1
From analla@cise.ufl.edu
Content-Type: application-agent; boundary = abxvf

--abxvf
Content-Type: application/agent-attributes

Agent Name: Filter Agent
Source:  h1@ufl.edu
Last Host: z@ufl.edu
[Other Agent Attributes]
--abxvf
Content-Disposition: attachment;
filename="Agent.class"
Content-Type: application/agent-code

[Class File follows]
--abxvf
Content-Disposition:attachment; filename="agent_obj"
Content-Type:application/agent-state
```

Figure 3. HTTP request encapsulating aZIMAs agent

The attributes are analogous to the concept of an agent passport presented in [17]. These attributes are used by the web server for various purposes including implementing security policies, detecting if agent has been tampered with, and other necessary functions such as choosing the next host from the pre-configured list of hosts. The key point here is that the agent attributes are read only as they cannot be modified by the host web server. One of the main fields is the agent name that is used to give unique identification to the agent. Some of the other fields are the source host and authentication certificates. The next figure shows some sample attributes of an agent.

Attribute Name	Attribute Value
`Agent-Name`	`AZIMAsagent`
`Source`	`harris.cise.ufl.edu`
`Host-List`	`ramses.harris.ufl.edu` `Cairo.harris.ufl.edu`
`Authentication Certificates`	*`Akgfhrnokdhbghjjhhaaaa`*
`Static Hash value`	`99`

Table 1: Attribute Structure of Agents

4. IMPLEMENTATION

We now describe the implementation of the two components that form the framework of the server platform for the aZIMAs system.

4.1 Apache Module

The Apache web server has a structure in which various modules interact with the core server to process an HTTP request. The processing of an HTTP request is divided into a fixed number of phases. The server extension module can elect to handle any of these phases or can even abort processing the request and invoke the server's error processing routine. The server extension module can include special content handlers that are invoked to process the request at response time. The structure of the server extension module is presented in the following figure.

Figure 4 shows a detailed view of the various components of the minimal infrastructure layer that resides as the server extension module of Apache and also its interface with the runtime layer. The communication manager is responsible for handling all agent mobility requirements. This is achieved by receiving agents sent by the user and by sending agents to another web server. The security component present in the Apache server module is responsible for implementing security level configurations set at the web server level. The interface to the Agent Runtime Layer plays a crucial role as it passes the agent classes, the agent object and also the agent attributes to the runtime layer.

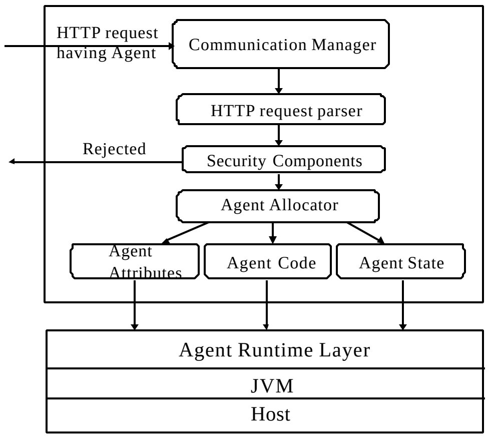

Figure 4. Components present in the Apache server extension module and the interface with the Agent Runtime Layer

As part of our implementation, we have developed a server extension module that handles requests containing mobile agents. We used Apache Server - Version 1.3.3 running on a Linux machine as our development and test platform. The module uses the Generic Apache Request Library (libapreq) [8] to retrieve the different subparts in the request. The module contains a special content handler (*agent-exec-handler*) that is invoked by the server when processing the request containing the agent. The MIME [5] type of the agent (*application/agent*) sets the content handler to *agent-exec-handler*. The handler first tries to connect to the runtime layer to pass on the agent for loading and launching. If the module determines that the runtime layer is not currently activated, it launches a child sub-process using an internal Apache 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 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 establish a communication link.

The next step after contacting the JVM is to parse the HTTP request containing the mobile agent and pass 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 whether the agent is *Non-Interactive* or *Interactive*.

***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. 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 user interaction. When the server receives an interactive agent it creates a temporary space for it and stores its class files. 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 its name is derived from 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. Interactive agents contain a `DialogApplet` class, which is a generic applet provided by the client side infrastructure to wrap the user interface provided by the application agent. The agent uses Java Mail API [10] to construct an e-mail message that contains a hyperlink to the agent's applet residing in the web server. The mail API uses an SMTP host that is configured by the infrastructure and is made available to the agent. When the user receives the e-mail message, he clicks on the hyperlink to render the applet on his browser. After the user finishes interaction with the applet, the agent is again serialized at the web browser and it is sent back to the web server.

4.2 Runtime Layer Implementation

The runtime layer is implemented in Java and its primary task is to launch the agents received from the Apache server. The components present in the runtime layer are shown in Figure 5.

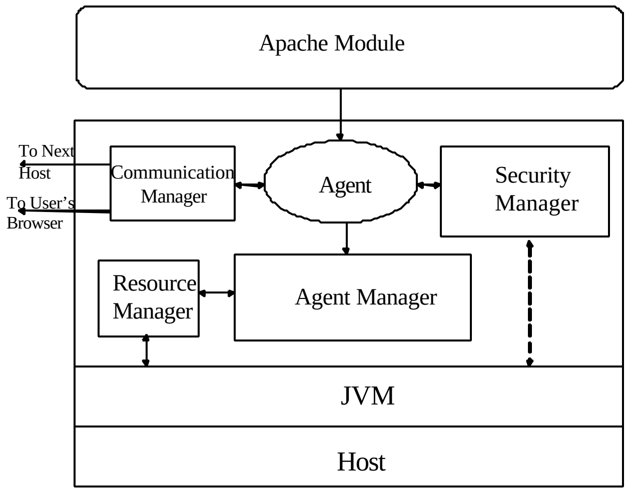

Figure 5. Agent Runtime Layer components

A separate thread is launched exclusively for each received agent. The newly launched thread is the AgentManager whose role is to start the agent thread for execution.

The AgentManager first tries to load the classes belonging to the agent by creating a separate class loader (`AzimasClassLoader`) for each agent. The class loader creates a new class space for each agent, this prevents name clashes among classes belonging to different threads and it also guarantees that only trusted classes loaded by the system as part of the agent are used during execution. After the classes belonging to the agent are loaded, the AgentManager uses the agent's object to de-serialize it. The de-serialization process is carried out by an `AgentHandler` object. The AgentHandler starts a new thread of execution for the agent. This thread is returned to the AgentManager which in turn instantiates a new agent thread. An agent that is executing in the server invokes its *go(String dest)* method to request migration to the next host. On request for migration, the AgentManager again serializes the agent and sends it to the next server.

5. PERFORMANCE EVALUATION

In this section, we present some of the performance results obtained after deploying our minimal infrastructure on our extended Apache web server. The performance of a web server is critical and should not be affected by the additional components. The metrics used to test the performance of a web server include, *the response time, the request rate,* and *the response rate*. 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.

5.1 Methodology

The tool used to measure the performance of the HTTP server is httperf [15], which is a command line tool that sends HTTP requests to the test web server. The tool takes as parameters, the number of connections to be a made, the request rate and some other optional parameters and provides various statistics that can be used to judge the performance of a web server. For our simulation experiments, we developed a separate tool that sends mobile agents to a server at a varying rate. This tool takes the total number of mobile agents to be sent and the rate at which the agents are sent as parameters. The simulation was carried out by sending HTTP requests containing: (1) aZIMAs agents and (2) simple queries, at a gradually increasing rate using httperf. We also benchmarked the load generated by agents with different resource requirements and tested the performance of the web server under different loads.

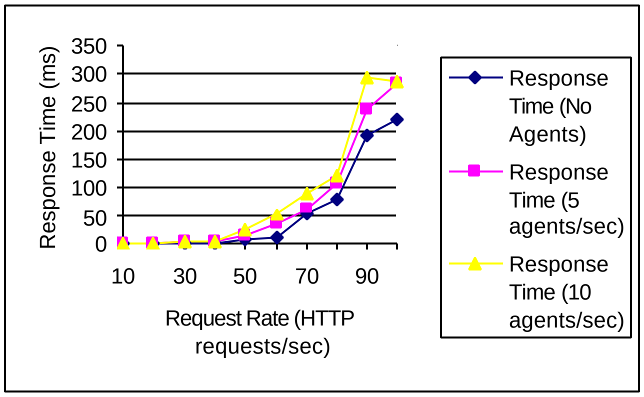

Figure 6. Response time of web server with no agents and with agents sent at varying rates

Figure 6 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. 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. The response time is compared in the three cases as shown in Figure 6. The agents that are used for this simulation are Non-Interactive test agents that use very limited resources on the server. From the graph we can observe that 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. One of the key things to note is that the response time of the web server does not increase greatly due to the presence of the agents. This is evident at request rates smaller than 50 requests/sec.

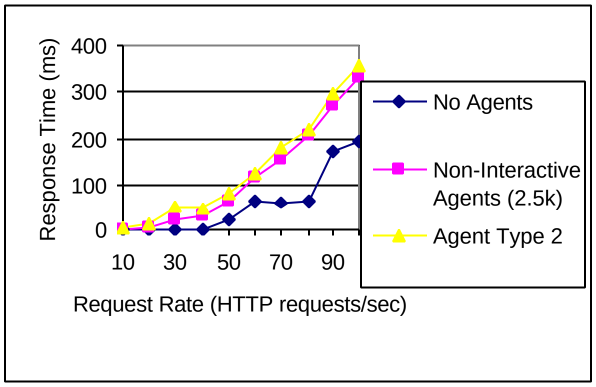

Figure 7. Response time of the web server with agents having different loads

Figure 7 shows the affect on the web server response time when agents with varying resource requirements are sent. Type 1 Agents have a predetermined lifetime and consume no resources on the web server, these agents are used to test the performance of the web server while the server extension module and the runtime layer is being used continuously but other resources are not being overly used. Comparatively Type 2 agents were bulkier agents that consumed both memory and CPU time on the host server. These agents consumed memory on the server and also used the CPU processing time to perform computations like calculating random numbers and also accessed server resources like files. The response time is not affected much 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 normal HTTP requests. But there is a slight increase in the response time as the overall system resources are used by the agents.

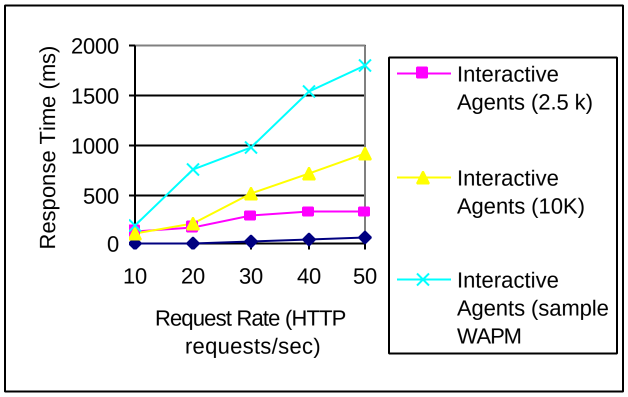

Figure 8. Comparison of Response Time for Interactive vs Non-Interactive Agents

Figure 8 compares the effect of agent size (code size) on the web server response time. It compares Non-interactive agents with Interactive agents of different sizes. While it is not unexpected to observe higher response time for larger agents, the results, unexpectedly, show *significant* difference between these two agent categories that can be attributed to a number of factors. The main observation is therefore 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 were 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 this agent was approximately 40K bytes. The results have prompted us to consider optimization of the WAPM model as a high priority future work. The planned optimization is simply to reduce the size of the classes that are shipped as part of the agent, by different techniques including caching.

But even with an optimized WAPM, we must make the observation that the difference in performance for processing Non-Interactive agents and Interactive agents is considerable. This is because Non-interactive agents are very small, whereas Interactive agents tend to be much larger. However, it is our expectation that in actual deployment of the system over the Internet, the servers will be hit more frequently by lightweight Non-Interactive agents rather than by relatively bulkier Interactive Agents.

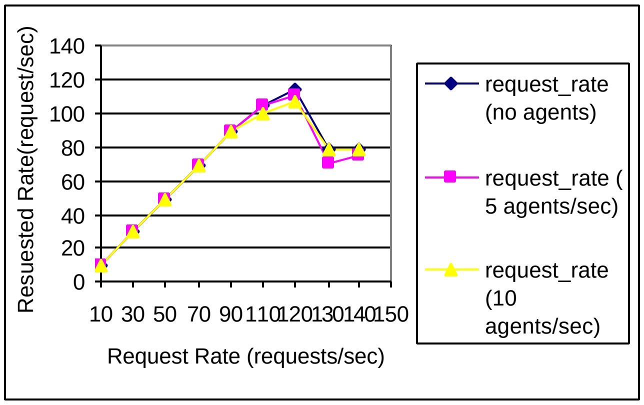

Figure 9(a). Requests served by Web Server

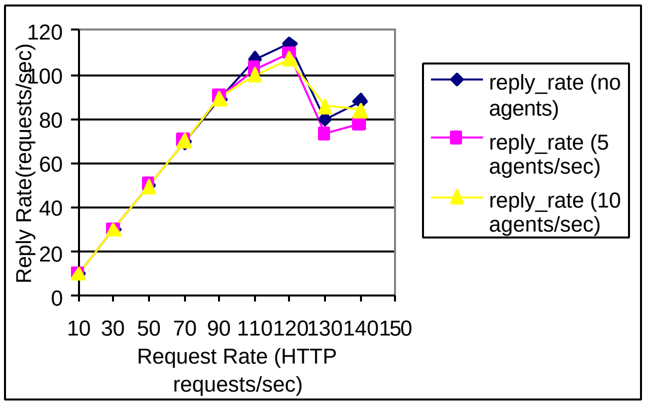

Figure 9(b). Replies sent by the Web Server

Figures 9 (a) and (b) show the affect of the aZIMAs extensions on the saturation level of the web server.

In this figure, Request Rate refers to the number of HTTP requests that are sent by the client tool, whereas Reply Rate refers to the number of replies successfully sent back by the web server. The web server is saturated if it is not able to process all the HTTP requests and some of the requests are dropped. The saturation level of the test web server was reached for 130 requests/sec for HTTP requests with no agents being sent at the same time. Simulation results with agents arriving 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 a little earlier due to the presence of the agents but this can be attributed to additional load of the agents on the server.

Simulation results have showed us that the extensions proposed by us do not affect the performance of the web server in a drastic manner. The difference in performance that is observed can be attributed to the minimal load generated by the extensions to the web server proposed as part of the aZIMAs system.

6. APPLICATION SCENARIOS

We describe two applications that we developed and implemented above the aZIMAs system to demonstrate its capabilities. The first one is a meeting scheduling application that deploys agents that interact with multiple users to achieve their goal. The second application is an information search and filtering application that accesses the resources provided by the web server. We briefly describe each application below. Full details of the applications can be found in [16].

6.1 Scheduler Application

A meeting scheduler agent visits a web server and contacts users using e-mail and provides a link to its user interface applet that resides on the web server. The user is able to interact with the agent through his or her web browser. The agent collects input from each user, processes it and moves to another web server to resume the process. After processing input from all the users the agent arrives at a suitable meeting time that is convenient for all the users. The meeting scheduler application is used to demonstrate the concept of *Interactive Agents*. These agents can be used conduct surveys, implement a distributed voting system and other applications needing collaboration between multiple users.

6.2 Information Search and Filtering Application

The agents developed as part of this application visit a web server and access the information sources that are available at the web server. Only those files and other sources that are explicitly published by the web server for the agents are available to the agent. The agent cannot access other files or resources on the web server thus demonstrating the security policies of the aZIMAs system. Such agents represent mobile web crawlers that actually move between a number of web servers to access the latest and the most relevant information. These agents fall under the category of *Non-Interactive Agents*.

7. RELATED WORK

Several research projects deal with implementation of general-purpose mobile agents. For most of the agent systems developed until now, specialized “agent servers” are needed if the agent system is to be deployed on the Internet. Our approach is different, as we are trying to incorporate agents on the web by using HTTP and making minimal changes to existing web server technologies to support the execution of mobile agents.

Research is also being done that looks into creating a web-based infrastructure for mobile agents, especially [13] presents a customized web server that has components that support agents written in a variety of languages. The WASP project [6] proposes an infrastructure for mobile agents on the web servers. The goal of the WASP system is to provide services on web data using mobile agents to implement these services. The approach is different since the agents are owned by servers and are later sent to the users so that they can request the desired service using the agent. As part of this project a customized, Java based HTTP server was developed that supports the functionality of agents. We also believe that this is the first time that an attempt is being made to use an existing popular web server (Apache) to support the functionality of mobile agents.

8. CONCLUSION AND FUTURE WORK

We have presented the design and implementation of a server platform for the aZIMAs mobile agent system. We have also developed applications that use the platform to build both interactive and non-interactive agents. Simulation results show the affect of the system on the performance of the web server. We are working to extend the system to include additional features such as keeping track of resource consumption by agents and facilitating negotiation policies regarding usage of resources. Currently, we are also exploring the features provided by other systems like SOMA [2] and J-SEAL2 [4] that enable the mobile agent system to keep track of the resources consumed by a mobile agent and thus prevent denial of service attacks. We are also adding security features like encryption of agents and protection of agents from hosts. We are particularly interested in reducing the overhead of the Interactive agents. We are also building a suite of *Interactive* and *Non-Interactive* applications that use the features provided by the platform. The aZIMAs system could actually make it possible to deploy mobile agents on the Internet because it is based on minimal extensions to existing web servers and does not need the presence of a new system.

9. REFERENCES

[1] Aglets Software Development Kit, http://www.trl.ibm.co.jp/aglets

[2] Bellavista, P., et al, Monitor and Control of Mobile Agent Applications, *ACM OOPSLA Workshop on Experiences with Autonomous Mobile Objects and Agent Based Systems*, Minneapolis, USA, October 2000.

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

[4] Binder, W., Design and Implementation of the J-SEAL2 Mobile Agent Kernel, *Symposium on Applications and the Internet (SAINT 2001),* San Diego, USA, January, 2001

[5] Borenstein, N., Freed, N., MIME(Multipurpose Internet Mail Extensions) , Part Two: Media Types, RFC 2046, November 1996

[6] Fuenfrocken S., How to integrate Mobile Agents into Web Servers, *Proceedings of Wetice97 Workshop on CADWA*, MIT, Cambridge, MA, June 1997

[7] General Magic Odyssey, http://www.genmagic.com/agents/odyssey

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

[9] Gray R., Agent Tcl, Department of Computer Science, Dartmouth College, http://agent.cs.dartmouth.edu/general/agenttcl.html

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

[11] Karnik Neeran M, Tripathi Anand R, "Design issues in Mobile Agent Programming Systems, *IEEE Concurrency*, 1998

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

[13] Lingnau A., Dronik O., Doemel P., An HTTP based Infrastructure for Mobile Agents, *WWW Journal – 4th International WWW Conf. Proc.*, Boston, USA, Dec 11-14, 1995

[14] Mole, http://mole.informatik.uni-stuttgart.de/, May 2001

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

[16] A. Nalla, A. Helal, and V. Renganarayanan, "aZIMAs and a WAPM", Submitted to IEEE Transactions on Computers, Special Issue on Data Base Management & Mobile Computing.

[17] 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

[18] The Apache HTTP Server Project, http://httpd.apache.org