aZIMAS: Web Mobile Agent System

Subramanian Arumugam, Abdelsalam (Sumi) Helal and Amar Nalla

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

helal@cise.ufl.edu

Abstract. Mobile Agents technology opens up new avenues in personalizing and customizing the web experience of users. It provides new possibilities for deploying distributed applications using existing web infrastructure. One of the reasons why mobile agents are not yet popular on the web is due to the lack of an easily deployable framework that would facilitate their existence. Existing mobile agent systems usually require heavy infrastructure that lacks interoperability if deployed on the Internet. In this paper, we describe aZIMAS (almost Zero Infrastructure Mobile Agent System) - a framework that will enable the execution of lightweight mobile agents on the Internet and remove some of the constraints imposed by existing systems. aZIMAS uses existing platform independent protocols like HTTP to achieve code mobility and agent interaction. Our approach involves adding a minimal infrastructure layer, called Agent Environment (AE), over existing web servers and using web browsers as clients. By basing our framework firmly on existing web servers and browsers, we hope to leverage the pervasiveness of web browsers and servers and achieve similar pervasiveness for mobile agents.
Keywords: *Mobile Agents, Agent Systems, World Wide Web*

1 Introduction

Mobile Agents technology opens up new avenues in personalizing and customizing the web experience of users. It provides new possibilities for deploying distributed applications using existing web infrastructure. There are two important factors that we believe will make mobile agents a critical part of the Internet in the near future. First, due to the explosive growth of content on the web, the average user is subjected to overwhelming amount of information. It has become increasingly necessary for the underlying software to take an active role in presenting useful information. This involves information processing tasks like data filtering based on user preferences, automating common tasks etc. Many commercial software attempt to alleviate this problem [11]. With the advent of web services, more resources are available in a structured and easily accessible form than ever before. The second phenomenon is the

increasing number of mobile users. Unlike connected computers, mobile users often have to deal with a weak network connection, limited resources (power, memory, screen area). Hence, it is attractive to consider options that would move resource and bandwidth consuming processes to remote locations. Some of the problems arising out of both these trends can be attributed to the passive role played by existing software tools, which merely display retrieved information. Whereas through the use of mobile agents one can develop tools that actively participate in a user's web interaction process. Mobile users can benefit immensely in areas like remote processing and data rendering, whereas the average web user can tap on agents to cope with the information overload. With the Internet having been identified as the most desirable platform for distributed applications, agents can also help realize richer and dynamic forms of collaboration and cooperation. Given these advantages, the goal of our work is provide a framework for agents to operate on the Internet and access common web services. The framework should integrate seamlessly with existing infrastructure tools realizing agents as a natural part of the World Wide Web.

Our previous work [5] investigated the merits of an agent system that is based upon existing web infrastructure software like web servers. The work resulted in the project aZIMAS (almost Zero Infrastructure Mobile Agent System), a mobile agent system based on the apache web server. The system had a very simple architecture and provided no support for features like agent communication and collaboration. Since the system was custom designed for Apache, deploying it in other web servers required extensive modification. We realized the need to extend the aZIMAS framework. This resulted in our current improvised version, which consists of a simple framework called Agent Environment (AE) that can plug-in to a web server through means a server extension module specific to a web server.

The contributions of this paper can be summarized as follows:

(1) We demonstrate a new agent system that is based on existing web technologies and which requires no additional client-side components
(2) We present a new technique for integrating agent systems with web servers that is based on a lightweight web server-specific module. This approach enables the deployment of the agent system in any web server with no modification.
(3) We investigate the performance implications of deploying an agent system along with a web server.

The rest of this paper is organized as follows: In section 2, we give an overview of research that deals with web integration of agent systems. In section 3, we present the architecture of the aZIMAS System with a detailed description of its Agent Environment (AE). Section 4 provides an overview about aZIMAS agents and describes how issues like agent communication, mobility, and security are handled. Section 5 deals with implementation issues. In section 6, we present some preliminary performance results. We then describe usage examples and application scenarios in Section 7. Section 8 concludes the paper with a note on our current work.

2 Related Work

There are a number of research projects that deal specifically with the issue of integrating mobile agents into web servers. The approach taken by these projects fall in one of the following two categories:

- *Custom Web Server based*: Develop a custom web server integrating an agent execution component. The agent component handles agent related requests. Projects like WASP [1] (Web Agent-based Service Providing), and Agent Server [2] take this approach.

- *Script based*: Make use of server-side facilities like common gateway interface scripts to launch applications that handle agent requests. M&M framework [3] and WebVector [4] fall under this category.

In the *custom web server* approach, agent system architecture is usually bound tightly to a specific web server. The web server normally consists of an embedded agent server environment which enables it to host mobile agents. Agent specific requests are exchanged via HTTP POST messages with unique MIME-types. The web server recognizes messages with specific MIME types as agent related and passes it on to the agent server for further handling. The problem with this approach is that existing web sites may not want to replace their current web infrastructure in favor of a custom-made web server. Custom-made web servers may not match the power, reliability, and efficiency of production-quality commercial web servers. Thus, it is likely that any agent enhancement solution that gains acceptance among web content providers is based on existing web infrastructure software. Another issue with this approach is that the tight coupling between the web server and the agent architecture makes it difficult to deploy the solution in a wide variety of web servers uniformly.

In projects that make use of server-side scripts, a script normally handles the task of receiving agents, and supporting their operation. In M&M agent system, web deployment is achieved by making use of servlet technology. WebVector uses common gateway interface scripts to receive and transport agents. If the custom server based approach makes the level of coupling tight, this approach tends to decouple the web server and the agent system completely. This has some important performance consequences:

- The web server will incur a significant overhead if the agent system is bulky.
- Since the web server has no awareness of the agent system, neither the agent system nor the web server can take advantage of internal optimizations.
- To access local resources, the agent system will have to act like any other normal HTTP client.

Our approach seeks to strike a compromise between the two approaches discussed above. Our solution consists of an agent environment with a well-defined interface, so

that it can easily plug-in into existing web servers. In our approach, the agent environment remains the same for all web servers. The agent environment is integrated into a web server through use of a simple server module developed specifically for the server. Though this approach makes the server module specific to each web server, we believe this is not a constraint to the deployment of the aZIMAS AE. This is because the only task performed by the server module is to forward an agent-specific HTTP request to the AE, get the reply back from the AE and communicate it to the client. Hence, developing a server module for any given web server should be a relatively easy task.

We have developed an extension module for the popular apache web server to route requests to the agent environment. We are currently investigating if a similar architecture can be achieved on an IIS Server through the use of ISAPI extensions. For IIS server, an ISAPI extension that could interface with the AE would be needed. Developing a server module for other web servers should be a relatively easy task as almost all commercial web servers provide a library of routines to help developers in creating extension modules.

Using a server module to integrate an AE offers number of other advantages. The web server usually loads server modules during start up, so server modules incur little or no overhead in processing a request. In addition, all important server data structures are usually available for the server modules; hence clever optimizations can be done to improve request processing. Finally, by moving web server specific tasks to the server module, we can keep the agent environment the same for any web server.

3 aZIMAS System architecture

Figure 1 shows the high-level design consisting of both the client and server components of the aZIMAS System.

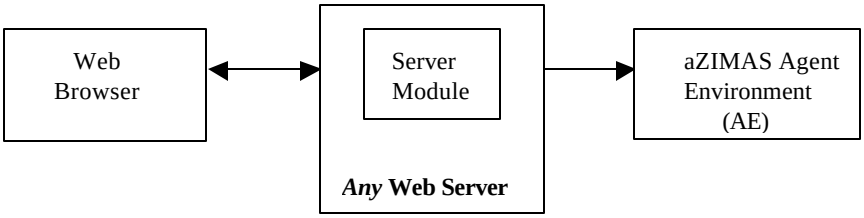

Figure 1. Architecture of aZIMAS

3.1 Client Side components

Client components in agent systems are typically used by users to launch new agents, check the status of launched agents, and in general to monitor and direct the actions of an agent. In contrast to many available agent systems and to the previous version, the current aZIMAS system does not require a separate client side component for launching agents. Instead, users interact with the aZIMAS Agent Environment using a web browser. Azimas *WebInterface* module exposes the functionality of the aZIMAS System. Interaction with the AE takes place through means of HTTP POST requests. The request structure requirements in AE for HTTP POST messages are well defined, making it easy for user applications to issue requests to the AE.

We have also developed a preliminary programming model called Web Agent Programming Model (WAPM) that would enable even non-programmers use the functionality of the agent system. WAPM defines a simple scripting language that can be used to direct and define an agent's action in a very high-level language.

We have adopted HTTP POST as the basic communication mode to interact with the AE since it is easy to define a request structure using POST. Also, POST enables us to deal with arbitrarily large data blocks.

3.2 Server Side Components

The server side components consist of the server extension module and the aZIMAS Agent Environment (AE) (figure 2).

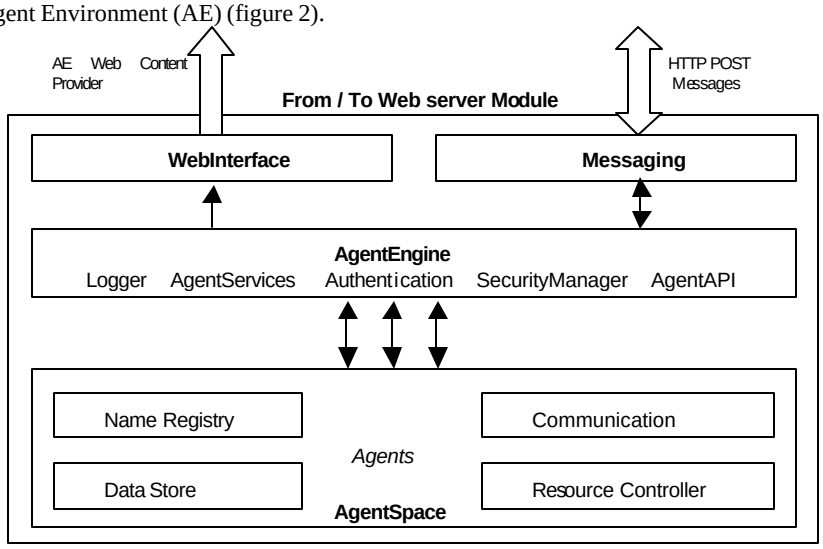

Figure 2. aZIMAS Agent Environment (AE)

The server extension module captures HTTP requests that are intended for the AE (identified by a specific request extension) and routes it to AE. aZIMAS AE is a minimal layer framework that consists of a *Messaging* and *WebInterface* component, an *AgentEngine* component, and an *AgentSpace* component.

The *Messaging* component acts like a gateway in the AE. It sends and receives messages between the web server module and the *AgentEngine*. *WebInterface* component acts like a content publisher. It exposes the functionality of the agent environment through static and dynamic web pages and provides the World Wide Web interface to the AE.

The *AgentEngine* provides services like verification of request structure related to agent messages, and authorization and authentication of incoming messages. The *AgentEngine* passes incoming agents to the *AgentSpace* after verification. The *Agent Engine* also includes a *Logger* to log system activities. The *AgentEngine* also has an integrated *SecurityManager* to control access to the Azimas *AgentAPI*. The AE exposes its services via the Azimas API.

The *AgentSpace* component forms the heart of the Azimas Agent Architecture. AgentSpace deals with supporting the lifecycle of an agent directly. It provides all necessary functionality needed by the agents like migration, agent communication, persistence, etc. On receiving the agents from *AgentEngine*, *AgentSpace* registers them in a *Name Registry* and then loads their state/code for execution. *AgentSpace* keeps track of the various agents in the AE and provides the interfaces to the *Agent API*. To some extent, *AgentSpace* also controls usage of resources like memory and disk space by agents through a *Resource Controller*. A *Communication Manager* handles message transfer among agents as well as transporting agent packets as HTTP messages to other hosts. Storage requirements of agents are handled using a *Data Store*.

4 Agents in aZIMAS

We define an azimas agent as a program that acts on behalf of an entity, and which has the ability to move autonomously from one host to another. The entity, which owns the agent (owner), can be a user, program or an organization. Agents in azimas can only exist within the context of the *AgentSpace* in Agent Environment (AE), which provides the functionality necessary for the agents to operate. In aZIMAS, every agent is associated with a *home base*, which refers to the AE at which the agent was first created. For an agent, hosts other than the home base are referred to as *foreign bases*.

An azimas agent can be described by the following attributes:
-ID, which uniquely identifies it within an AE
-*Home base*, the AE at which the agent was first created
-Credentials to prove its authenticity
-Data (like program state and output, itineraries, agent type etc.,) and
-Code that forms the agent program

These various attributes are packaged and represented as an *AgentTravelPack* when an agent is transferred between azimas hosts.

In aZIMAS, agents are classified as either *Interactive* or *Non-Interactive*:

- *Interactive Agents* have the ability to respond to the activity of their owners, and possibly other agents. Interactive agents can cooperate with other agents in realizing a common goal. These agents have the ability for synchronous and asynchronous collaboration and can be used to build distributed Internet Applications. Interactive agents can optionally specify *AppletContexts*, which provides an applet interface for their owners to direct their actions.

- *Non-Interactive Agents* are primarily concerned with information sources available at different hosts on the Internet and are suitable for information search and filtering applications. These agents have the ability to replicate or clone themselves. A non-interactive agent and its clones form a group, within which synchronous and asynchronous communication is possible. Non-interactive agents normally do not interact with agents outside their group and typically communicate results and messages to their owners in an asynchronous manner.

Every incoming agent is expected to identify its type (interactive/non-interactive). The purpose of this classification is to aid the aZIMAS AE in decisions of load balancing. When the number of agents in the system exceeds a given threshold (available system memory, disk space), the AE's *Resource Controller* gets invoked. The *Resource Controller* typically suspends some agents to ease the strain on the AE. Usually low priority agent threads get suspended in no particular order, with preference given to interactive agents and agents belonging to the AE's domain (*home base* agents).

4.1 Agent Communication

For cooperation and collaboration between agents to succeed, we need an effective communication medium. A communication medium facilitates agent interaction with other agents and its owners. aZIMAS provides support for both synchronous and asynchronous agent communication. Asynchronous communication is implemented through use of mailboxes. Every agent is allocated a mailbox, which stores incoming messages for the agent. A message sent by an agent is stored in the receiver's mailbox. An agent can do a blocking or non-blocking wait for messages. When an agent blocks

waiting for a message, it is notified upon the arrival of a message. If the agent is non-blocking then it has to explicitly check its mailbox to retrieve the messages. Synchronous communication is established through a rendezvous point established by the aZIMAS system. A rendezvous point opens up a connection between two agents in an AE, through which data transfer is possible. aZIMAS provides limited location transparency support, made possible through use of a forwarding service provided by every AE. Thus, when an agent receiving messages in an asynchronous manner chooses to migrate to a new location it will continue to receive messages at its new location.

4.2 Event Management

aZIMAS uses a type of publisher-subscriber model for event management. In this model, a publisher acts as an event generator. Subscribers receive events generated by publishers. In aZIMAS, an event manager facilitates event management and acts as the deliverer of events to the subscribers. Publishers register themselves with the event manager and deliver events to it. Agents can query the event manager for available publishers and can subscribe to specific publishers. An agent can act as both a publisher and a subscriber simultaneously. A subscriber in an aZIMAS AE will continue to receive event notification even after it has moved to a different host. Events are Java objects in aZIMAS, which provide a general description of the event along with its details.

4.3 Mobility and Agent Persistence

Mobile agents, by their very nature, need to have the ability to move from one host to another. aZIMAS supports only weak migration for Agents. In weak migration, only the code and state of the agent is transferred as opposed to strong migration, where the execution state is transferred. Attempts have been made to provide for strong migration [6] by either extending the Java Virtual Machine or by capturing the execution state through use of a backup object. Though strong migration has a number of positive aspects, the high overhead introduced by these techniques makes it unattractive to our cause. In aZIMAS, agent migration is handled by the AE. Agents are passive participants in the migration process and cannot react to events that take place during migration. An agent issues a migration request through a move method call. The move method when invoked never returns and takes an itinerary string as a parameter. An itinerary string is an ordered triplet *(destination_host: method: parameter)*, indicating the destination host to which the agent wishes to migrate, the method to be invoked upon arrival at the destination along with parameters that needs to be passed to the method. As of now, only simple parameters like strings and primitive types can be passed as method parameters. Agent persistence is supported through use of Java's serialization mechanism. An agent's state and code, once serialized, persist physically at the *home base*. At *foreign bases*, serialized agent state and code is maintained only in memory.

The agent's *home base* always keeps track of the location information of all agents belonging to its domain (home agents). This is made possible by an update to the location information at the *home base* whenever the agent moves between hosts. When an AE receives a foreign agent it sends the agent's *home base* AE a status message indicating the agent's new location.

4.4 Security

A mobile agent system needs to define proper security framework before it can be trusted and deployed widely since they often will have to execute arbitrary code from unknown agents. In addition to the security issues related to the agents in general, new issues arise when mobile agents are deployed on the web. Fortunately, dealing with web security is not as hard as that of agent security and acceptable solution exists to many problems [8]. To prevent unauthorized web use, aZIMAS AE requires each user to follow a registration process before being allowed to access agent services. This enables the AE to identify each request with a specific aZIMAS user. Since HTTP is a plain-text protocol it is possible for snoopers to gain access to confidential information. Use of secure HTTP where possible should eliminate this problem. It should be noted that in the current system no provision has been made to share user information across AE's. Thus, a user registered at a particular AE can use services of other AE's only indirectly through use of agent programs. To make registration information common across AEs, one can adopt a central registration server.

A detailed analysis of security issues in agent systems in provided in [7]. Broadly, there are three main classes of threats in agent systems:

(1) ***Agent-to-Platform***: The agent exploits weakness in the agent system to gain unauthorized access to resources, or otherwise launch attacks on the agent system. Numerous techniques have been developed to protect an agent system from harmful agents like safe code interpretation, signed code, proof carrying code, state appraisal etc.,

(2) ***Platform-to-Agent***: This category of threats represents situations in which the agent's security is compromised by attacks from the agent system. Preventing this type of attack is generally difficult since the agent system controls the execution environment of an agent. There are some techniques to prevent attacks from the agent system, but they tend to fall more towards detection that prevention.

(3) ***Agent-to-Agent***: In this category, an agent launches an attack against another agent possibly exploiting security weakness of the other agent. These types of attacks can usually be tackled using the same techniques that are employed for protecting the Agent platform.

The current system does not have any provision to prevent attacks by the platform. We assume that aZIMAS AEs are hosts trusted by the agents. To prevent attacks by the agents, aZIMAS employs a combination of techniques. Every agent is expected to carry credentials identifying its owner and the home base. As like many other agent systems, aZIMAS makes use of Java's inbuilt security mechanisms to prevent runtime attacks by providing a *Security Manager* that controls access to potentially harmful system libraries. aZIMAS offers some security features to prevent agent-to-agent attacks. This includes agent blocking, in which an agent can block messages from another agent to it by requesting the AE.

5 Implementation

We have implemented most of the framework that has been described in the previous two sections. We have developed the basic agent environment component, which supports the lifecycle of the agents. We have implemented a server extension module for the popular Apache Web server. As part of our work, we have also developed a simple web server called Pluto, which interfaces with the AE. Pluto makes it easy to see the project in action even on systems with no installed web servers. The Agent Environment is implemented using Java. Java provides a number of features useful to agent systems like serialization mechanism, on-demand code loading, relatively strong security framework etc., which makes it an excellent language to base an agent system upon. Some popular agent systems based on Java [9] include Aglets, Concordia, Voyager, Odyssey, Ajanta etc. The web server dictates the choice of language for the server extension module. In our implementation of the apache module, we have used the C language.

The core of the system is accommodated in five Java classes: `AzimasMessaging`, `AzimasWeb`, `AzimasAgentEngine`, `AzimasAgentSpace`, and `AzimasSecurityManager`. The implementation is modeled according to the aZIMAS design described in section 3.

All aZIMAS agents are derived from a basic agent class called `Azimas`, which helps enforce certain minimum guidelines for the agent programs. Agents are represented as threads in the AE. The base agent class has an attribute called `AzimasSpaceInterface`, which needs to be set by the Agent Environment. This is a Java object by means of which the agent accesses many services from the AE. To facilitate invocation as a thread, the system mandates that the agents provide a `run()` method, from where the agent code normally starts executing. Apart from these requirements, the AE's *SecurityManager* constrains the agent programs to make calls only to safe library methods.

The aZIMAS API provides a safe way to access many useful functions. The method names model closely after the Java library convention. For example, to invoke a request to an external URL the agent just needs to make a call to the `getLocation` method,

which takes an URL string as a parameter. The method call if successful returns the data retrieved from the URL.

6 Performance Evaluation

In this section, we present some of the performance results obtained after deploying our AE on Apache web server. The performance of a web server is critical and should not be adversely affected by the presence of additional components.

We carried out our simulation by sending HTTP requests at a gradually increasing rate using OpenLoad, a command line tool that can be used to test web server performance.

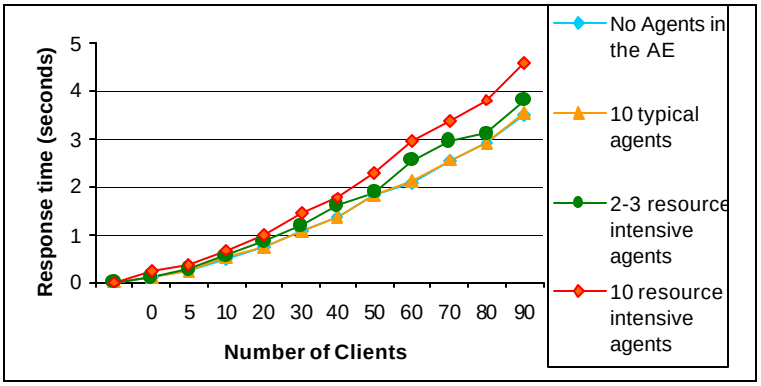

Figure 3. Impact on web server's response time

Figure 3 shows the affect on the response time of the web server for various cases. The base case represents the situation when no agents were present in the AE. The second case represents the situation when 10 non-resource intensive agents were present in the AE. These agents typically accessed some site on the Internet and communicated the result back to their owners. Since we expect this type of agents to be more common on the web, we call them as *typical web agents*. The agents were uploaded in to the AE from a web browser and the base test case is repeated. From the figure, we can observe that there is only a negligible difference in the response time of the web server due to the presence of *typical web agents*, this is due to the fact that these agents did not make use of excessive CPU or system memory. However, when resource intensive agents were uploaded to the AE, we could observe a noticeable difference in the response time. The effect is very obvious when we repeated the base test case with 10 resource intensive agents in the AE. The response time of the web server worsened by as much as 30 percent! Resource intensive agents, in this case, performed the computation of an N x N matrix multiplication where N = 300. The drastic change in response time occurs due to the fact that resource intensive agents take up much of the available CPU time thus affecting the web server processes.

We observed a similar drop in available system memory when resource intensive agents were present in the AE (figure 4). This effect is however easier to control than the previous one as it is easy for the AE's *Resource Controller* to make sure that the memory used by agent programs does not exceed the threshold set by the AE.

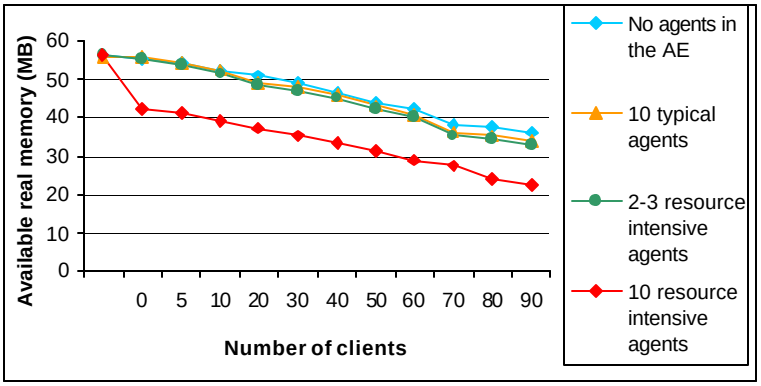

Figure 4. Impact on available system memory

Though there is only a negligible impact on the web server when *typical agents* were present, it is not advisable to co-locate the AE and a busy web server on the same machine. This is due to the fact that it is often not possible to know beforehand whether an agent is resource intensive or not. In our opinion, it is best if the AE is maintained in a machine different from the web server. To test this scenario, we repeated our simulation with the web server and AE located on different machines in the same network and subjecting the AE to large number of resource intensive agents. As expected, the response time of the web server was similar to the base (no agents) case and the performance implications were felt only in the machine where the AE was located.

7 Usage

Message exchange to the AE takes place using HTTP POST. A user accesses the services of the aZIMAS system through an account. The AE exposes its services to the user through web pages served by the *WebInterface* module. If the user intends to create a new agent, he/she selects the createAgent option and then uploads the compiled class file of the agent. The web server's server extension module captures the request and passes it on to the *Messaging* component of the Agent Environment (AE). The user is informed about the successful transmission of request and provided a link to query about the agent's status, and output. The status page provides information about the current location of the agent, its trust level etc. After eliminating

HTTP headers, the *Messaging* component transfers control to the *AgentEngine*. Verification of the client's request structure takes place in the *AgentEngine*. If verification is successful, the *AgentEngine* passes the request to the *AgentSpace* for further processing.

A valid HTTP POST request intended for the AE needs to be in conformance to the request structure expectations of the AE. A `requestType` is a basic field that needs to be present in all messages directed to the aZIMAS AE. It helps the AE in deciding how to process a request. Depending on the type of the request, the AE may expect additional fields. For example, a message containing agent code is handled differently based on the request type:

- If the request type is `createAgent`, it implies that a new agent needs to be created at the AE. The AE stores the code of the agent in a separate folder created for the user. The code is then loaded and the agent execution begins, provided necessary resources are available. The AE expects fields like `userName`, `userPassword`, `agentName`, `agentType` etc.,
- If the request type is `agentPack`, then it implies a visiting agent. The AE allocates only temporary space for the agent under a `visitors` folder. The AE also informs the home AE about the successful migration and then restores the state of the visiting agent. The AE expects fields like `agentTravelPack`.

The request structure expectations are well defined in aZIMAS. This enables user applications to access aZIMAS services by following the correct request structure. An example POST request illustrating a `createAgent` request is shown below.

```
----------------------------30881262316024
Content-Disposition: form-data; name="requestType"
createAgent
----------------------------30881262316024
Content-Disposition: form-data; name="agentName"
Matrix1
----------------------------30881262316024
Content-Disposition: form-data; name="agentType"
Non-Interactive
----------------------------30881262316024
Content-Disposition: form-data; name="userName"
sa2@cise.ufl.edu
----------------------------30881262316024
Content-Disposition: form-data; name="userPasswd"
*********
----------------------------30881262316024
Content-Disposition: form-data; name="agentFile"; filename="matrix.class"
[……
 Class Bytes
……]
----------------------------30881262316024--
```

7.1 Application Scenarios

We now describe some application scenarios where our system can be of use.

Search Agent:

aZIMAS provides a means for agents to search information available on the Internet. For example, the search results from the popular search engine Google are available as a Web Service. aZIMAS provides a search routine that relies on Google to obtain the results. An agent's call to the aZIMAS search routine results in a service query to the Google search engine. An agent program can make use of this useful feature and perform automated processing and filtering of the search results to fit a particular criterion. The agent can follow an itinerary, visiting and collecting information from various websites. For example, one can write an agent that indexes all results from .edu sites for a given search phrase. The agent can further refine these results to match specific keywords in the document. By its nature, this agent is non-interactive. This application can be useful when there is a need to process a large amount of information on the Internet. By possibly moving the agent application near to the source, the user can save bandwidth and at the same time free his/her machine for other activities. Moving processing to remote locations is also attractive for machines which are weakly connected, or which are poor in resources like low memory devices.

Info Agent:

An Info agent acts like an information repository for a user. It keeps track of the user's personal and contact information, bookmarks, and address book. The agent provides an interface on the Internet so that anyone can add new entries to the user's address book. The agent returns to the user's host machine at periodic intervals and synchronizes with the address book maintained locally by the user. An advantage with this agent application is that it provides owners with access to their address book, and bookmarks anywhere, anytime. The agent can also be used to set alerts, reminders based on date or time. The agent will then remind the user about the impending event through email or other communication means. This application qualifies easily as an interactive agent.

8 Conclusion and Future Work

In this paper we described aZIMAS, a novel framework that realizes mobile agent systems within the context of the World Wide Web. The approach interfaces the agent system (Agent Environment) with web servers through means of a minimal server specific extension module and makes use of web browsers as clients. By making only the extension module as server specific it is possible to deploy the agent system

uniformly in a wide variety of web servers. Our framework provides a reasonably secure environment for mobile agents to function on the Web.

We are currently working on abstracting the functionality of common web applications and making them available as part of the aZIMAS Agent API. Another area of interest is to realize aZIMAS agents as providers of web service. We are also investigating ways to incorporate proactive security features like verification of mobile code to check for harmful intent. We are fine-tuning the various features of the system and soon plan to have a public release of the AE along with web server modules for popular web servers.

References

1. Funfrocken, S., How to integrate mobile agents into Web Servers, 6th Workshop on Enabling Technologies Infrastructure for Collaborative Enterprises, 1997
2. Anselm Lingnau and Oswald Drobnik and Peter, An HTTP-based Infrastructure for Mobile Agents, Proceedings of the 4th International WWW Conference, 1995
3. P. Marques, R. Fonseca, P. Simões, L. Silva, J. Silva, Integrating Mobile Agents into Off-the-Shelf Web Servers: The M&M Approach, Proc. of the International Workshop on Internet Bots: Systems and Applications, 2001
4. T. Goddard, V.S. Sunderam, WebVector: Agents With URLs, Proceedings of the 6th Workshop on Enabling Technologies Infrastructure for Collaborative Enterprises, 1997
5. Amar Nalla, Abdelsalam (Sumi) Helal and Vidya Renganarayanan, aZIMAs: Almost Zero Infrastructure Mobile Agents System, Proceedings of the IEEE Wireless Communications and Networking Conference, March 2002
6. S. Bouchenak, Pickling threads state in the Java system. Third European Research Seminar on Advances in Distributed Systems (ERSADS'99), Madeira Island, Portugal, April 1999
7. Wayne Jansen and Tom Karygiannis, Mobile Agent Security, National Institute of Standards and Technology, Special Publication 800-19, August 1999
8. Stefanos Gritzalis and Diomidis Spinellis, Addressing threats and security issues in World Wide Web technology, In Proceedings of 3rd IFIP TC6/TC11 International joint working Conference on Communications and Multimedia Security, 1997
9. Damir Horvat, Dragana Cvetkovic and Veljko Milutinovic, Mobile Agents and Java Mobile Agents Toolkits, Proceedings of the 33rd Hawaii International Conference on System Sciences, 1998
10. David Kotz, Robert S Gray, Mobile Agents and the Future of the Internet, ACM Operating Systems Review, 33(3), August 1999
11. Internet Agents, http://cws.internet.com/agents.html