Internet Agents for Effective Collaboration

Vidya Renganarayanan, Amar Nalla and Abdelsalam (Sumi) Helal[1]

Computer and Information Science and Engineering Department
University of Florida, Gainesville, FL 32611, USA
[1]helal@cise.ufl.edu
http://www.harris.cise.ufl.edu/projects/azimas.htm

Abstract. With the ever-growing Internet, distributed applications are gaining popularity. Going "online" to browse, to send/receive emails is an integral part of our daily lives. In fact, the majority of Internet users rely on email and web browsers as the *de-facto* tools for collaboration. Such popular tools, however, provide limited and inefficient mechanisms for collaboration, especially when several collaborators and resources are involved. Surprisingly, collaboration tools such as NetMeeting and Netscape Collabra are only used in limited applications despite their rich collaboration support. What is lacking is an invisible infrastructure that extends email and web browsing tools – to which most Internet users are accustomed and addicted – to intuitive interfaces of a full-fledged Web collaboration system. In this paper we present WAPM, a Web Agent Programming Model that uses mobile agents as the tool for coordination and describes a model that uses familiar tools such as the web and email for effective asynchronous collaboration. We introduce WAPM and give details of a simple scripting language used to direct agent actions, a pre-processor to check the validity of the script, an agent manager that creates and manages agents according to the script and a defined class hierarchy for developing application agents in WAPM. The mobile agents are based on the aZIMAs (almost Zero Infrastructure Mobile Agents) system that integrates mobile agents with web servers by providing mobility using HTTP.

1 Introduction

Many routine activities require coordination among several people and the input from multiple users; deciding venue, date and time for a group activity, taking a survey, distributed voting mechanisms and decision making that require the consensus of more than one person, to name just a few. These activities form a class of applications that are currently handled by sending emails and a personal assistant (human) collecting the responses and processing them or alternatively writing a cgi script to process input collected using a form. In today's world of the Internet, where ubiquitous computing is actively being sought, the applet-servlet communication scheme is also being used. Internet has increased our web of contacts; more collaboration is possible and desired, yet we only have the same old tools. Chat software like Yahoo Messenger [14], MSN

Messenger [10] and NetMeeting [8] that provide synchronous collaboration are prevalent.

A major drawback of all the schemes mentioned above is the necessity of a client-server communication link or the burden on the personal assistant handling these tasks. An asynchronous mode of collaborating between multiple users and a purely automated processing of the user inputs would be a more attractive option. Such a technique would also be a more appropriate fit for the ubiquitous computing model of the Internet.

This paper describes a programming model that primarily focuses on applications that require collaboration or coordination between multiple users. The Internet, which is our primary means of communication and collaboration today, has been used as the foundation for this model.

The model aims at seamlessly exploiting agent technology to increase coordination and achieve more effective collaboration by using familiar and popular tools.

Mobile Agents are software modules that can migrate from machine to machine carrying out user-defined actions. Web Agents represent their user on the Internet. Agents perform a wide range of actions, from information access, being personal assistants – assisting users in daily computer-based tasks to semi-intelligent agents that adapt to the user behavior and environment so as to mimic the user and carry out tasks on behalf of the user.

The proposed model is based on the mobile agent paradigm. Mobile agents bring many advantages to this class of applications. They do not require an active connection to interact with the user, hence can work with low bandwidth, weakly-connected or disconnected environments. Mobile agents move to the user machine and once they are done, leave the machine hence freeing up memory for other applications. This provides users with low memory devices also, to access and interact with these applications. By their very nature they provide asynchronous communication.

There are many **Issues** when handling mobile agents. Mobile agents are migratory; they hop to the user machine, interact with the user and then hop to the next host. However, typical agent architectures require a server to be running at every host that is to be agent enabled. The server implements the agent protocol that is used to send and receive agents. This requirement adds an overhead on the user machine.

The agents need to interact with users who may be using different devices and different system configurations. The agents need to be written in an interpreted language so that it is more portable. Since the class of applications under consideration here are interactive, the interface rendering mechanism will have to be chosen carefully, to ensure optimum portability and compatibility.

The Internet Agent Programming Model presented in this paper will address the above-mentioned issues. It is not a model that defines a new architecture, neither is it concerned about agent issues such as agent location, naming, inter-agent communication to name a few. Many architectures [[1], [5], [6], [9]] have been defined and many research papers have addressed these issues.

The goal of this paper is to describe an effective programming model that provides an easy, quick and intuitive way of building applications that require user interaction. Some applications that we can build using this programming model are applications that schedule meetings, conduct surveys or implement a distributed voting mechanism.

The rest of the paper is organized as follows: Section 2 will describe the proposed agent programming model. Section 3 presents the agent primitives developed using the model; these act as building blocks for more involved interactive applications. Section 4 gives the conclusions. Finally, Appendix A briefly describes aZIMAs (mobile agent system), the agent system used by this model.

2 Web Agent Programming Model (WAPM)

The proposed Web Agent Programming Model (WAPM) is based on a simple mobile agent system called almost Zero Infrastructure Mobile Agents (aZIMAs) (see Appendix A) [11], developed at the University of Florida. The aZIMAs system, which is summarized in Appendix A, is an agent system that provides mobility to agents by using the Hyper Text Transfer Protocol (HTTP). It enhances the Apache Web Server to provide the means to send agents from client to the web server, run the agents at the web server and move the agents between web servers.

WAPM enables development and deployment of Web Agents over aZIMAs. **Web Agents** are agents that use the Internet as their roaming ground and jump from browser to browser carrying out activities for the user. Primary focus is on agents that interact with users to obtain information and process that information to retrieve meaningful results for the owner. WAPM does not address inter-agent communication or general agent applications. It is a model specifically focused on asynchronous collaborative applications involving multiple users and information on the Internet.

WAPM provides a web agent development environment. It provides guidelines for agent developers on how to put the agent system to work. It also enhances the architecture by providing agent developers with a means to express in a more intuitive way, what they want to do with their agents. Many agents have been developed and if we could harness the functionality of these, to do what we desire rather than re-invent the agent every time; that will provide for very effective code reuse.

The WAPM addresses two issues associated with any agent architecture. One, it provides a development environment specifically geared towards building agents that need to interact with multiple users. Two, given any architecture, a means to treat agents at an abstract level and coordinate the actions of the agents as per the logic of the specific algorithm for that application.

The agents built using WAPM can actually reside in the user's web browser interacting with the user, providing the information, getting feedback and then move on, to the next user performing the collaborative application's logic.

WAPM 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.
- **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.

2.1 Client Side Scenario

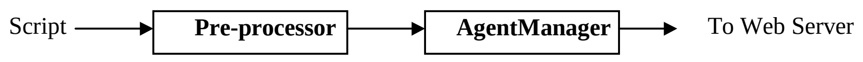

Fig. 1 Client-Side environment of the model. The *script* written in the restricted language of the model is run through a *pre-processor* and the syntax-error free script is parsed by the *Agent-Manager*. The *AgentManager* then moves to the first web server (as read from configuration)

A script written in the agent specification language is input to a pre-processor. The pre-processor ensures correct syntax and performs type checking. The pre-processor is run at the client-end so that the agents may be lightweight. Agent Manager is the interpreter for the Agent Specification Language.

Various agents written for the WAPM are pre-registered with the Agent Manager. The pre-processor gets the information and performs type checking for the agents invoked from the script. The pre-processed script is then given to the Agent Manager.

Agent Manager is the module that interprets the script written in the agent specification language. It executes the body of the script, which might require jumping to a web server or to a user. The agent manager maintains state between agent calls so that it can resume execution once the agent completes its job.

2.1.1 Agent Specification Language

The agent specification language describes the language features that can be understood by the agent manager. It supports a limited set of data-types and language constructs to effectively relate activities of multiple agents.

The script can have two sections DECLARE and BODY sections. The DECLARE section specifies the name value mappings to be used in the script. The data types supported are String, array of Strings and integer.

Language constructs supported are if-then-else and assign statements. The if-then-else construct can have an agent invocation as part of its condition and the result of the agent action can be compared to a value to produce a boolean valued expression that evaluates to true or false. The then-stmt and else-stmt parts can have agent actions or assign statements. The assign statements can assign the return value of an agent action or the results of an agent action to a name (results of agent actions are registered with the Agent Manager, see 2.2.1) and use the name elsewhere.

Every agent to be used in this script must register with the agent manager(see 2.1.2.1 Agent Manager APIs). At the time of registration, it will specify the RESULTS that it produces. The pre-processor will type-check agent invocation using this information.

2.1.2 Agent Manager

Agent Manager extends the `AzimasAgent` class provided by the architecture. Hence agent manager is also an agent conforming to the aZIMAs architecture. It implements the abstract *run* method of the `AzimasAgent` class provided by aZIMAs. This is to allow it to be launched as a thread.

The agent manager is also an interpreter for the script written in the agent specification language. It saves state [7] when an agent is to be invoked and stays dormant until the agent has completed its job. Every agent is aware of its agent manager and once the agent has done its action, it will re-start the agent manager. The agent manager will then restore state and resume execution of the script.

Upon startup the agent manager will automatically discover the agents in the current directory that are written for this architecture. It will then invoke static method *initialize,* to be provided in every conforming agent, to get agent information such as agent name and agent results.

2.1.2.1 APIs Provided by Agent Manager

- *registerApplicationAgent* – The agent developer in his initialize method, to register the agent with the agent manager, should call this method. It takes as arguments the application name, namely the function that this agent can provide users.
- *registerResults* – The agent developer in his initialize method, to register the results of agent action with the agent manager, should call this method. It takes as arguments the name of the result and the result type, namely String String array or integer.
- *setResult* - The agent developer in his *doAction* method must use the *setResult* method, upon completion of agent action, to give the result values to the agent manager.
- *setReturnValue* – The agent developer in his *doAction* method must use the *setReturnValue*, upon completion of agent action, to give the return value to the agent manager. The return value must be an integer and the its meaning must be clearly documented by the agent developer.

2.2 Web Server Scenario

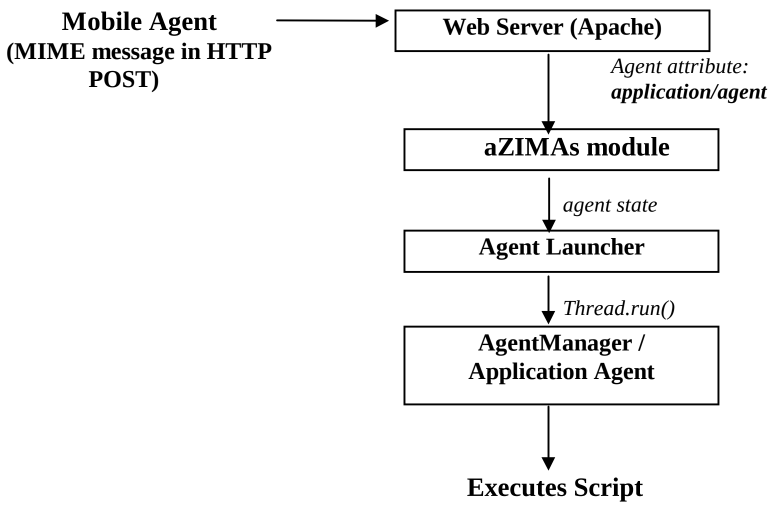

Fig 2. WAPM – Server Side. The *Mobile Agent* sent as a multi-part MIME message in an HTTP POST message is received by the *Apache Web Server*. The web server invokes the *aZIMAs module* for applications of type application/agent, indicated in the HTTP header. The module parses the MIME message and passes control to the *Agent Launcher*. The agent enabled by the *Agent Launcher* could be an *Agent Manager* or an *Application Agent*.

The Agent Manager reads from configuration at the client side and uses aZIMAs's *go* API to move itself to the first web server. The *go* API [(see Appendix A), [5]], composes a multi-part MIME message consisting of the agent-attributes, agent state and the various class files required by the agent. This MIME message is then sent as a HTTP POST request. At the web server, the application type is read from the HTTP header; an application/agent type of agent is handed over to the aZIMAs module of the Apache web server. This module stores the class files and transfers the agent state to the Java Run-time support environment, which is a Java program for launching agents, running on the web server. The launcher creates a separate thread of execution for each agent [Fig. 2].

Agents migrate from one machine to another by using the aZIMAs infrastructure's *go* method or using applet communication provided by a combination of *sendEmail* and the WAPM. Depending on the application the agent may also migrate from one web server to another using aZIMAs *go* API.

2.3 Interaction with User

Applets [7] have been chosen as the means of communicating with users on their browsers. Applets activate the Internet. Most popular browsers are already Java-Enabled. Users are familiar with the process of downloading applets off the web and running them. Web Servers already have a way of handling applets and security related issues could be met. Hence applets provide an established mechanism to interact with users.

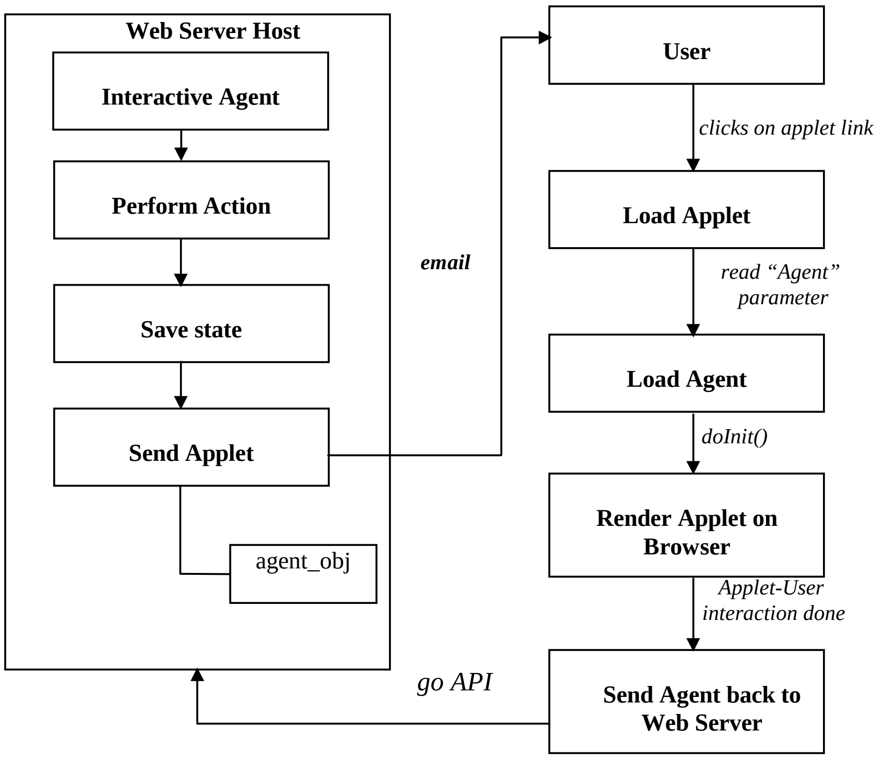

Fig 3. WAPM – Interactive Agent. At the *Web Server* the interactive web agent, to contact the user it wishes to interact with, *dumps its state* to a file and *sends applet* as email to user. The user on clicking the applet link activates the process of bringing the agent to his/her web browser. Once interaction is over, the agent *returns to the web server*. This process may be repeated as many times as required by the agent.

WAPM provides applet-agent technology for agent developers to develop agents that need to interact with users. Agents send the link to the applet using the *sendEmail* API in aZIMAs. The agent state is dumped into a file. The applet will contain a parameter that indicates the name of the file containing the agent state.

aZIMAs receives the agent and starts it off on a separate thread. The currently executing agent still has control and can now decide if it needs to contact other users or if has obtained the results. Contacting other users follow the same process as outlined above[Fig.3]. Once the agent has collected all the results, it returns control to the agent manager by activating the agent manager's thread of execution.

2.3.1 Dialog Applet

`DialogApplet` is a generic applet provided by the infrastructure to wrap the user interface provided by the application agent. `DialogApplet` reads the parameter from the HTML file, gets the agent state from the web server and calls *doInit* on the agent. The agent's *doInit* method has the look-and-feel of the applet as coded by the agent developer.
The applet has now been rendered on the web browser. The user actions are processed and once the agent has completed its work, it goes back to the web-server by using the aZIMAs API, *go* method [Fig. 3].

2.4 Developing a Mobile Agent

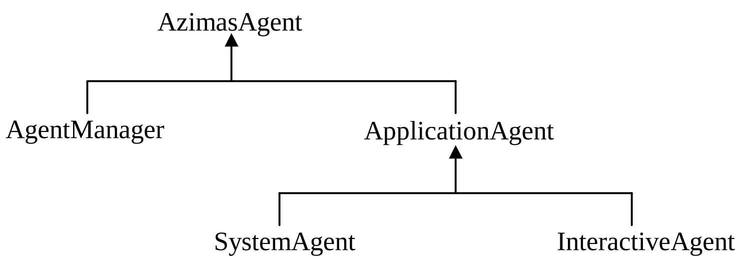

Fig. 4 Class Hierarchy of the Programming Model – Two kinds of application agents are possible, *Interactive* and non-interactive or *SystemAgents*.

An agent in the WAPM is an agent that attains its mobility from the aZIMAs architecture and is an agent that can be used in the agent specification language to be linked in a pipeline of agent activities.

A set of classes is provided by the WAPM infrastructure for mobile agent developers. These include `ApplicationAgent, SystemAgent, InteractiveAgent` [Fig. 4] and the `DialogApplet` class.

Steps to be followed to develop a mobile agent:

Step 1: Extend the `ApplicationAgent` class. The `ApplicationAgent` class extends from the `AzimasAgent` class provided by the architecture. This class also holds a reference to the `AgentManager`. Every agent has an agent manager associated with it; the reference is set when the agent is instantiated.

Step 2: Implement the abstract methods in the `ApplicationAgent` class, namely

- initialize()

  This method is a static abstract method specified in the `ApplicationAgent` class. It is used by the `AgentManager` to discover and get information about the Application Agent developed. This method must invoke two methods on the associated agent manager:

- registerApplicationAgent()

  This method registers the name of the application agent. Once registration is done, the user can use this application agent in the agent specification language.
- registerResults()

  This method informs the agent manager of the results produced by this application agent. Each result is given a name, the value for which will be set once the agent executes. The result name, as specified, can also be used in the agent specification language.
- doAction()

  This method is an abstract method in the `ApplicationAgent` class. It is invoked when the agent manager comes across the agent name in the agent script. This implements the application logic of the agent.

A single application agent may perform more than one action. The *doAction* method takes String name as a parameter. Depending on the name passed the appropriate logic may be executed. To provide for this functionality, *registerApplicationAgent*, which associates a name with the agent, may be called more than once in the initialize method.

Step 3: The infrastructure provides additional facilities for agents that intend to interact with users. Applets are the means of interacting with users. If the agent being developed is to interact with the user, then it should over-ride the *doInit* method of the `ApplicationAgent` class. The infrastructure applet, `DialogApplet`, invokes the agent's *doInit* method to render the applet on the web browser.

The template for an application agent to be developed using this model is given below:

```
class AnApplicationAgent extends ApplicationAgent
{
 public void initialize()
 {
  // The following two calls are repeated as many times
  // as the number of functions this agent provides.
  getAgentManager().registerApplicationAgent(
   “Function”);
  getAgentManager().registerResults ({
   “Result1”, Result2”,…,”Resultn”});
 }

  public void doAction (String name, Object[] params)
  {
   // application logic for the function passed in
   // parameter ‘name’
  }

   // Optional method: doInit
   // To be provided if the agent needs to interact with
   // user on the Web browser.
  public void doInit()
{
 // Application logic to render applet on web browser
 // and process user input.
}
}
```

WAPM provides some agent primitives that may be useful for users in writing more effective scripts using the agent specification language. Operations that are very commonly used, such as: sending information via email to a list of people, confirming agent actions that obtained user input, canceling agent actions that required interaction with the user. These primitives may be accessed in the agent specification language by the names: SEND, CONFIRM and CANCEL. These are SystemAgents and extend from the `SystemAgent` class. They do not require the applet model, since they do not interact with users. System agents are typically used to send information or perform action at the web server.

3 Application Agents Developed Using WAPM

Application Agents were built using the Web Agent Programming Model to demonstrate the features and to be able to present, visually, this new scheme of mobile agents and their interaction with users.

3.1 Multi-User Collaboration Application Agent

Frequently we come across scenarios where it is required to collect input from numerous users; namely surveys, feedback forms, scheduling meetings to name a few. To send out emails to multiple users and then track their replies, co-relate them and process the information to release meaningful data is a very involved process. To expedite this process, the Query agent can be programmed with the necessary logic to process user information.

The agent can jump from web browser to web browser, getting user input for users in a domain. It will jump from web server to web server to reach users in different domains. Identification of users is done using email addresses. The agent code is written using the model described in Chapter 4. The steps to develop a mobile agent using WAPM were followed and the application agent written conforms to the template.

The query application needs to interact with the user, hence the applet interaction model was used and a *doInit* method was implemented in the query application agent. The WAPM provides the facility to freeze the agent's state by serializing it and then re-starting it at some pre-defined point. This fits in with the requirement of the query application agent to be able to stay quiet until the user picks it up and interacts with it and then, return to the web server to find out the next user and continue collecting input.

The query application agent needs inputs of the form: from address, to addresses, subject and body (actual query). It can be invoked in the script using the application agent name registered with the agent manager; ASK. This information is specified to the agent manager in the *initialize* method as per the model specification. The syntax and data types are enforced when the AgentScriptPreProcessor runs through it.

An example of the query application agent, if used in a script as per the agent specification language:

```
DECLARE
FROM = vidyare@ufl.edu
LIST = { abhinav@ufl.edu , chl@ufl.edu , gu@ufl.edu }
SUB = Survey for the Lab
QUERY = “ Would you like card access to the lab? “
END DECLARE
BODY
if ASK ( FROM , LIST , SUB , QUERY ) > 1
then
         CONFIRM (  FROM , LIST , SUB , QUERY )
end_then
END BODY
```

This agent can be used to conduct surveys, implement a distributed voting system and many other application. This simple API which provides a mechanism to receive yes or no responses from multiple users, can be used as a building block to develop more involved applications

3.2 True Crawler/Filter Application Agent

The Internet is a treasure house of resources, resources in the form of information and in the form of the multitude of users who log-on everyday to browse or check email. We could tap these resources to obtain useful information; use a specialist in a field to filter out a huge list of information to something more appropriate for your needs.

An application agent that could take a set of input and interact with the user to filter it and reduce it to a more meaningful set was developed. This agent is a true crawler since it carries only the essential and most pertinent information. It extends from the `InteractiveAgent` class to provide an applet interface to the user.

The agent is registered with the name FILTER; it takes a list of information, sends it to users and returns the filtered results. This agent uses the `DialogApplet` to render the applet on the web browser. The `AgentManager`, activates the agent by invoking the *doAction* method. This method does some processing to initialize the state of the agent, and then invokes the *sendApplet* method, provided in the `InteractiveAgent`, to send email containing the applet link to the user indicated in the script.

The *sendApplet* method as described earlier, will dump the agent state into a file, indicate the name of the file as a PARAM in the applet’s html file and send the email. The agent is now in an inactive state, waiting to be loaded by the applet, once the user clicks on the link.

The agent once loaded by the applet, sits on the web browser of the user, renders the interface via its *doInit* method and interacts with the user.

The Filter application agent is to be used as a building block. It makes possible applications such as scheduling meetings; where a user may be given a number of choices and he chooses the ones convenient for him, applications to specialize search results by sending it to a live person and many others that involve a selection process.

4 Related Work

Applying the mobile agent paradigm to the Internet model is an area of active research. Many techniques have been proposed to enable collaboration applications.

TANGO [2] provides web-based collaboration using the JAVA applets feature. However it does not use the mobile agent model. The clients wishing to use an application in TANGO need to install the client-side environment. TANGO addresses collaborative applications where multiple users interact with each other simultaneously such as chat, whiteboard conferencing or a common drawing board.

WASP [4] uses the mobile agent paradigm and integrates it with the web server. It provides for users to interact with agents installed at the web server. The model however, does not address the issue of dynamic agents, created by users and which are not already installed on the web server.

Migratory Applications [3] describe a programming model for agent applications that hop from site to site, carrying information in *briefcases* and they receive a *briefing* at each site. The briefing contains advice for the agents such as site is busy or try an alternate site and site-dependent data. This model requires that an *agent server* be present at every site that is to be agent enabled.

Collaborative Interface Agents [7] address routine tasks that users perform on a day-to-day basis, but this talks about collaboration between the agents to assist the user better in performing these tasks.

Collagen [12] is an agent model that interacts with people, however the aim is to interact with only one user and present information from various sources to this user in a user-friendly manner.

aZIMAs [5] is a mobile agent system that uses the established HTTP infrastructure to mobilize agents. It has been used to enhance the Apache Web Server to accept and send mobile agents. It provides APIs for building interactive and non-interactive agents, however there is no application that fully utilizes the potential of such a system.

5 Conclusions

In this paper we described a Web Agent Programming Model (WAPM) that applies the mobile agent paradigm to building collaborative web-based applications. The model was developed over the aZIMAs architecture, which offers mobility to agents by transporting them between web servers using HTTP. The communication mechanism to other hosts on the Internet is through the popularly used electronic mail (email).

The WAPM provides the environment to build collaborative applications that are deployed over the Internet. It also clearly defines the interface an agent must conform to, hence allowing easy integration of existing applications with WAPM.

This model takes familiar, widely accepted tools such as email, web browsers and JAVA applets and combines them to provide a powerful environment for web-based collaborative applications.

We have demonstrated the use of WAPM by implementing two application agents, the Multi-User Collaboration Agent and the True Crawler/Filter Application Agent.

References

1. Ajanta – Mobile Agents Research Project, http://www.cs.umn.edu/Ajanta, University of Minnesota (Apr 1998).
2. Beca, L., Cheng, G., Fox, G.C., Jurga, T., Olszewski, K., Podgorny, M., Sokolowski, P., Stachowiak, T., and Walczak, K., *TANGO – a Collaborative Environment for the World Wide Web*, Northeast Parallel Architectures Center, Syracuse University, 111 College Place, Syracuse, New York (1997)
3. Bharat, K.A., and Cardelli, L., *Migratory Applications,* ACM Symposium on User Interface Software and Technology '95, Pittsburgh, PA, November 1995. pp. 133-142 (1995)
4. Funfrocken, S., *How to Integrate Mobile Agents into Web Servers,* Proceedings of the WETICE'97 Workshop on Collaborative Agents in Distributed Web Applications, June 18-20, Boston, MA (1997)
5. Gray, Robert S., *Agent Tcl: A flexible and secure mobile-agent system*, In Proceedings of the 4th Annual Tcl/Tk Workshop (TCL'96) (July 1996).
6. Karjoth, G., Lange, D., and Oshima, M., *A Security Model for Aglets*, IEEE Internet Computing, Jul-Aug 1997, pp. 68-77 (1997).
7. Lashkari, Y., Metral, M., and Maes, P., *Collaborative Interface Agents*, Huhns, Michael N., and Singh, Muninder P., editors, Readings in Agents. (1997).
8. Microsoft NetMeeting, http://www.microsoft.com/windows/netmeeting
9. Mitsubishi Electric., *Concordia: An Infrastructure for Collaborating Mobile Agents*, Proceedings of the 1st International Workshop on Mobile Agents (MA'97) (April 1997).
10. MSN Messenger, http://messenger.msn.com
11. Nalla, A., Renganarayanan, V., and Helal, A., *aZIMAs – almost Zero Infrastructure Mobile Agents*, Available at http://www.harris.cise.ufl.edu/projects/publications/azima.pdf (To be Submitted to SAINT 2002)
12. Rich, C., and Sidner, Candance L., *COLLAGEN: When Agents Collaborate with People*, Huhns, Michael N., and Singh, Muninder P., editors, Readings in Agents. (1997).
13. Sun Microsystems, Java, http://www.java.sun.com.
14. Yahoo! Messenger, http://messenger.yahoo.com/

Appendix A

The goal of the aZIMAs system is to make mobile agents pervasive on the Internet. This is achieved by developing a mobile agent platform that can be easily deployed on the existing infrastructure present in the World Wide Web. Web servers are important components in the architecture as they host the mobile agents and enable mobility of agents. The architecture targets simple information retrieval agents and interactive agents that enable collaboration of people over the Internet. These applications are enabled by a framework that enables mobile agents to jump from web server to web server for information retrieval and allows agents to transport themselves between the user's browser and the web server. Agents are transferred between web servers as encapsulated multipart MIME message in a HTTP POST request.

The server is an extended Apache web server that has an agent runtime layer to execute agents. The server side infrastructure consists of two components, an Apache module and an agent runtime layer written in Java. Figure 1 shows the overall architecture of the aZIMAs server.

The module written for Apache reads the multipart HTTP request containing the agent and parses it 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 with a security policy to reject agents having certain names, or coming from certain hosts. If the agent does not meet the security requirements 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 present in the agent object.

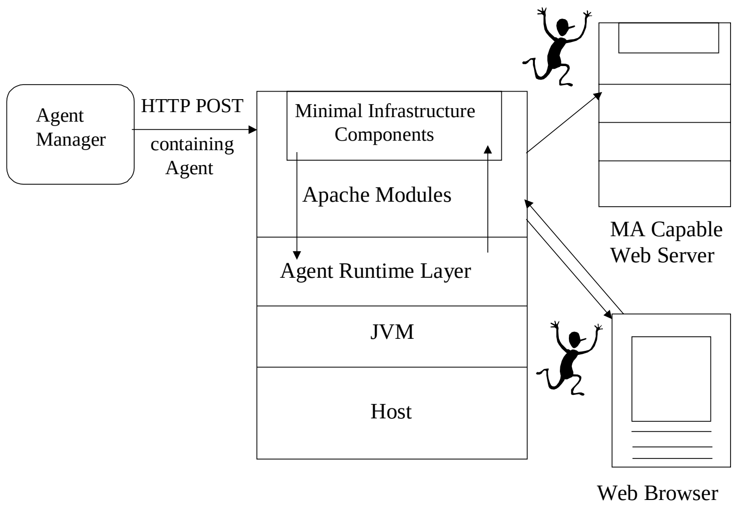

Fig 1. Architecture of aZIMAs platform. The figure depicts both the client and server components in the system.

Thus, functionality needed by a machine to host mobile agents is provided by extending the Apache web server by adding a server extension module and including 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 provides the infrastructure is activated and provides support for the agents.

The web server allows for secure execution of the agents on the web server and also protects the host web server from rogue agents. Agents migrate between web servers carrying their entire code (Java class files) and their state with them. Details of the aZIMAs mobile agents system can be found in [11].