Monitoring of Distributed Processes with Mobile Agents

Ryan P. Kennedy

Computer and Information Science Department
New Jersey Institute of Technology
Newark, NJ, U.S.A.
Ryanpk@bellatlantic.net

Franz J. Kurfess

Computer Science Department
Concordia University
Montreal, Quebec, Canada
Franz.Kurfess@computer.org

Abstract

The Proliferation of networked and distributed systems presents a need for more tool development with regard to the monitoring and maintenance of distributed processes. The goal of this paper is to present a mechanism used to collect detailed process information from various remote Unix hosts on a network. The interface to this mechanism, a GUI applet, is accessible through a Java enabled browser such as Netscape Navigator. It presents the user with a menu of choices such as which host to view, and what process information to retrieve(I/O – bound processes, numbers of processes, individual/total process usage, etc.). The requested information is retrieved and displayed in a window presented by the GUI.

1. Project Overview

Much of the work concerning the management of computer networks, especially in large organizations, is rather tedious, routine work. Although network management systems offer substantial support, the network administrators are often burdened with small decisions that required the processing of a lot of detailed, relatively low-level information. Over the last two years, our group has investigated the use of intelligent agents for network management; this paper describes an enhancement of the user interface, and more realistic experiments.

The goal of the project described in this paper is to develop a Java GUI applet that will be used to collect detailed process information from various remote hosts on a network. The experimental setup for the system has been expanded and some implementation aspects have been revised. These are described in this paper.

2. Related Work

This paper is based upon previous work done by Klaus Holthaus, Felip Miralles, Franz J. Kurfess, and Dhaval P. Shah [1] in which a platform-independent application for the monitoring of distributed applications is presented. This application is to be effected through the use of mobile agents that travel throughout a network, enabling collection of a distributed application's process information over various (remote) hosts. A client-side applet GUI is used by an administrator to specify the criteria for information to collect. The result is returned and displayed in a window of the GUI.

The aforementioned project is ongoing and the work detailed in this paper is an extension and support effort, with particular emphasis on user interaction, and on practical tests in realistic setups.

3. Intelligent Agents

An intelligent agent is an autonomous entity that acts on behalf of the user. The key here is *behalf*; the agent autonomously behaves according to its own judgement (on *behalf* of the user), influenced by parameters that the user specifies. The benefit is that the user need not be directly concerned with the task the agent has been developed to perform. The task is being performed by the agent.

Intelligent agents usually begin or change operation when some recognizable event occurs. "In the context of intelligent agents, an event is anything that happens to change the environment or anything of which the agent should be aware. For example, an event could be the arrival of a new piece of mail, a timer going off at midnight, or a change to a web-page" [7]. When the event occurs, the intelligent agent must autonomously analyze the situation (or conditions) and act according to the new environment and its design specification. This is the *Events-Conditions-Actions* [7] behavior, and it is common to all intelligent agents.

Intelligent agents come in two types, stationary agents and mobile agents. Stationary agents do not leave their system or host of origination; rather, they operate using more traditional means such as Remote Procedure Calls. Though a stationary agent may be very helpful by its function, it does not provide the level of advantages that a mobile agent does.

A mobile agent enhances the abilities of intelligent agents. A mobile agent is not bound to the system on which it begins execution [2]. When it travels to another host on a network, the agent transports its execution state and execution code along with it. Upon arrival at the destination, the agent resumes its execution. This behavior provides several advantages. It helps reduce network traffic. The agent travels to the host where the object or data is located, and executes upon it there, afterwards returning to the origin host with the results. The data remains at its own location rather than being transmitted over the network for processing. Mobile agents are also very helpful with performing network management tasks. A number of agents disseminated across a network may execute autonomously and in parallel, aiding in network administration by performing management tasks at each host. Mobile agents are ideal for heterogeneous networks. Because they perform execution operations locally, they are mostly hardware and software independent; they execute in the context of the local environment. Distributed applications using mobile agents across a heterogeneous network are provided with seamless system integration [2].

Mobile agents aid in mobile computing (e.g. travelling salesperson with a laptop). The links that connect mobile computer users to a network are often unreliable or sporadic. This greatly reduces the effective use of distributed applications with respect to the mobile user. Mobile intelligent agents provide a means to reduce the penalty of mobile computing. First, they are location independent. The agent may begin a task when the user is at one location and return the results to the user at another location. Also, disconnection of the user from a network does not mean that the tasks being performed by the user are disrupted. For example, suppose a travelling salesperson begins a time-consuming interactive application on a local network, which requests information and prices on many products. Next, suppose that the task was initiated via a mobile agent. Then the salesperson, before the application has completed, disconnects from the network and travels to some other distant location, only to reconnect at a later time. During the duration of time that the salesperson has been disconnected (to travel), the mobile agent has been handling the task. Later, when the salesperson reconnects, the mobile agent is alerted of this event and will travel to the new location with the results.

The salesperson example also exemplifies the Events-Conditions-Actions [7] operation design of intelligent agents mentioned above. The (autonomously handled) *event* is the reconnection of the salesperson at a remote location ant the agent being alerted of this (e.g. a message sent to the agent). The *condition* is the new user location. The *action* is the response to the new user location. The agent must decide to transport itself with the data from the task it was performing (the product information application) to the user, for delivery of final output.

This section presents some of the advantages of mobile agents. There are many others; the intent of the section is to illustrate the potential of mobile intelligent agents.

4. Intelligent Agent Development

The development environment for the intelligent agents of the project is the IBM Agent Building Environment Developer's Toolkit (ABE), Level 6. The ABE Toolkit, Level 6 utilizes a rule-based reasoning system to impart intelligence to agents. It is available on several platforms: AIX Version 4.1.4, OS/2 Warp, Windows 95, Windows NT. The toolkit consists of five components as depicted in the diagram below:

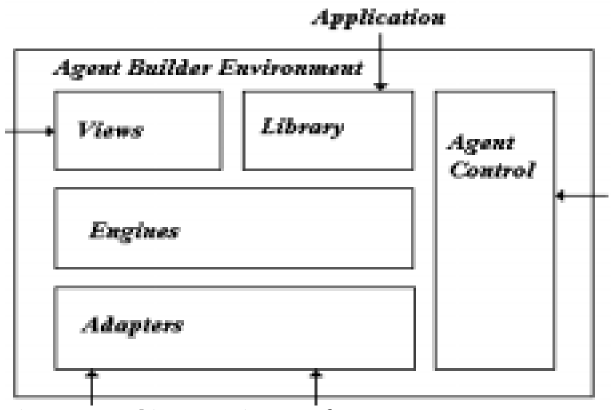

Figure 1 – Architecture Diagram of ABE

This diagram is taken from the documentation provided with the ABE Toolkit [8], which is also the source for the following description of the main components. Each of the five components has distinct functions which provide a developer with tools for agent development.

Views: "This component establishes and modifies the instruction for the agent." It contains a rule-editor, which allows the developer to enter, edit, and remove rules from the rule ABE. The rule-editor is an interface to views.

Library: "Manages the storage of the agent instructions (rules and facts)."

Engine(s): Provides a rule-based engine for interpreting instructions to the agent.

Adapters: "Provide the interface needed by the engine to trigger events and communicate with the engine." There are five adapters: files adapter, http adapter, nntp adapter, time (alarms) adapter, and util (mathematical utilities) adapter.

Agent Control: Initializes and provides overall control of the running agent. This component also provides an interface for designing agents.

The arrows origination from outside the development kit and pointing to components represent the existence of an interface. The interface is provided for the user to facilitate using a component to make changes, developments, etc.

Intelligent agents developed using ABE and Java are named aglets by IBM. The ABE Aglet Daemon is a server that needs to be run on computers that are to use aglets. The daemon provides a context or "place" which is a homogeneous environment in which aglets may travel to and from, and execute. The context is implemented in Java and is thus platform independent.

5. Project Components

The main component discussed here is a user-interaction component implemented as a Java applet, which is used to retrieve detailed process information from one or multiple hosts on a network. The application design is composed of the following components:

Client GUI: The GUI has one menu allowing the user to specify a host to view, and another menu to specify process criteria (type of processes to view, or all processes). There are two windows in the GUI. One to display the processes (as specified by the process criteria in the menu) that are running on a host, together with their states, usage, and priorities. The second window displays host information, IP address, idle-time, hostname, I/O-time, and processing-time.

Client GUI process window: Each process to be displayed in the window appears as a circle. When the user places the mouse pointer on a process circle, a window pops up, listing the information of the process (i.e. process id, name, usage, state, etc.) [1].

Client GUI host information window: This window displays the pertinent information for a specific host in a text format. It lists IP address, idle-time, hostname, I/O-time, and processing-time of the host.

The GUI applet is not standalone; rather it inter-operates with other components that actually facilitate the collection of information for the GUI. These comprise the following:

Helper Tool: This is local to a host and retrieves (via low-level calls) host information (IP address, hostname, processor idle-time, I/O-time, processing-time) and process information (process ID, process name, type(thread/process), state, kernel-running-time, user-running-time, sleeping-time priority). The Helper Tool is implemented in C, and based on a Unix platform. For other platforms such as Windows NT, separate helper tools must be developed.

Intelligent Agents: The agents are implemented using IBM's ABE Development Toolkit. The Java-based agents are called aglets. Aglets are used in this project because they may autonomously perform the tasks given them. Their ability to intelligently execute allows them to implement their goals efficiently and flexibly. Their ability to traverse a network, bringing their code and preserving their state helps reduce the network load that would otherwise be greater with a different implementation. The intelligent agents used in this project come in two forms: mobile and stationary[10].

Mobile Aglets: Intelligent agents that roam the network, collecting the information produced from the helper tools located on the various hosts and delivering it to a stationary aglet located on a "Management Server" [4], [1].

Stationary Aglet: Located on a management server with whom the GUI applet communicates. This aglet (developed with IBM's Agent Building Environment(ABE) is responsible for persistent storage of information delivered to it by the mobile aglets [4],[1].

6. Input and Output

As indicated above, the interaction with the user is required for the system to function. The user must dictate a host on the network to view, as well as any process criteria for viewing. This can be accomplished via the two pull-down menus. The user may manipulate the menus using a mouse or keyboard.

The applications output reflects the user's input. The host information window of the GUI displays the information of the host that pertains to the host chosen by the user: host name, address, etc. The information displayed in the process information window of the GUI reflects the process criteria as specified by the user (or the default, which in itself is a user's choice). The application retrieves the process information of a host from the stationary aglet, which received the information from one of the mobile aglets. Upon receipt from the stationary aglet, the GUI displays the information.

7. Project Functionality

Five steps summarize the overall functionality of the application:

1. Connection to the user's choice of a network host to view. This is implemented by access to a pull-down menu found on the GUI. Each choice on the pull-down menu will be a host that is located on the network. The user only needs to choose one to be connected to that host [1].
2. Record the user's process criteria (or set of information that the user wishes to have displayed). This action is implemented by access to a second pull-down menu, which enables the user to choose what information will be returned and displayed in the GUI process window. The default will be to display all processes running on the host, along with all pertinent information.
3. The display of process information in the GUI process window. This information will reflect the criteria that the user has specified regarding which processes to view, and on which host(s).
4. The display of host information, such as hostname and IP address, in the host information window.
5. Print the contents of the GUI's process and information window. This function is enabled by a print button provided as part of the GUI. A log-file is also created which will reflect the same data as the output of a print. Like the print function, this facility is to be enabled by a button which is found on the user interface.

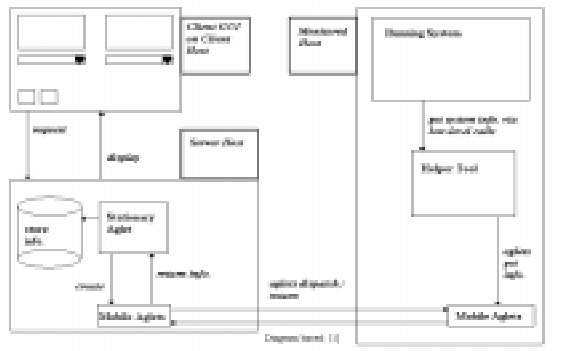

Figure 2 – Overall Project Schematic

8. System Organization

The system includes three different components that interact:

1. Network hosts from which information is needed; they are monitored.
2. The management server in which the stationary agent collects and stores information about a network host.
3. The client GUI, which accepts user input, collects information from the stationary agent residing on the server, and displays collected information for the user.

The client GUI is designed with JavaScript and a Java applet and will send a request to the management server for process and host information. The server will then retrieve that information from a network host using mobile aglets, and upon receiving the information, again from a mobile aglet, will make it available to the client GUI. The client GUI will then display the information for the user [1].

8.2 Part I – The Client (user) GUI

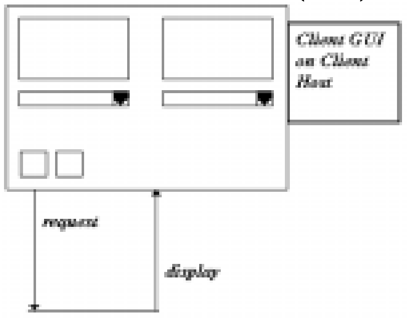

Figure 3 Part I – The Client (User) GUI

The client GUI applet receives user input that will specify the monitor information which is to be retrieved (i.e., from which host, process criteria). The client GUI sends this request to the management server. The management server will then dispatch a mobile agent to obtain the information. When the information is retrieved, the management server will alert the client GUI. The GUI will then display the retrieved information in its host information and process information windows.

8.3 Part II – Management Server

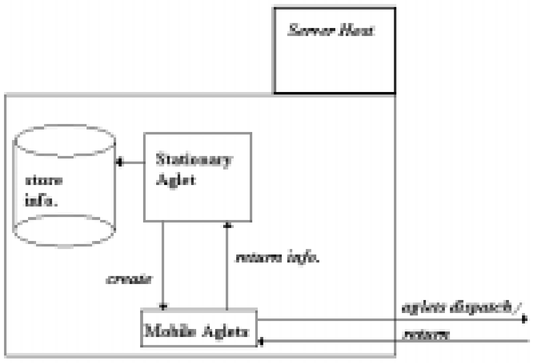

Figure 4 Par II – Management Server

The management server has a stationary aglet with the ability to create and dispatch mobile aglets. When the management server receives a monitor request from the client GUI, it dispatches a mobile aglet to the host to be monitored. The aglet will be autonomous and will act in accordance to the user-specified criteria of information retrieval. When the aglet returns from a monitored host with the required information, the management server will store the information and make it available to the GUI [1].

8.4 Part III – The Networked Host

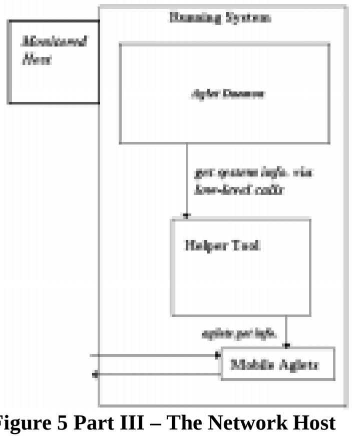

Figure 5 Part III – The Network Host

These are the computers on the network to be monitored. Each network host to be monitored must have the IBM ABE installed on it. This allows the Aglet Daemon to run on it. The Aglet Daemon in turn allows aglets to reside on the host by providing an aglet context or "place". It is in this context that the aglets reside and do their work. The aglets will have been created by the stationary aglet and dispatched from the management server, and will travel to the host to be monitored. The aglet will be able to autonomously retrieve the process information according to the process criteria specified by the user (through the help of the Helper Tool). After the information is retrieved, the aglet will travel back to the management server [1].

9. Equipment Configuration

The client GUI may run on any platform that supports a Java-enabled browser such as Netscape Navigator. The Management Server is designed to run on a Unix platform. The Network (Monitored) Hosts are Unix hosts.

10. Implementation Languages

Java language is used to code the client GUI applet and the Management Server because it is platform independent and is the preferred language for agent development. Through the use of Java object serialization, an aglet object may be persistently serialized or "stored" along with its execution state and associated values. The serialized aglet can then be dispatched over the network to another host where it is deserialized. The execution code and stored values allow the aglet to resume execution on the remote host in its previous execution state.

The Helper Tool installed on the network (monitored) hosts is implemented using C language because it is efficient for low-level calls to the system for process and host information collection.

11. Conclusion

The project is part of an ongoing endeavor and it enhances the interaction capabilities and the functionality of the work reported earlier. The purpose is to facilitate the monitoring and information retrieval of process information on networked hosts. The intent of the effort is to present a tool that may assist in network maintenance and monitoring.

12. References

[1] F. J. Kurfess, D. P. Shah, K. Holthaus, F. Miralles, “Monitoring Distributed Processes with Intelligent Agents”, *In Proceedings, Engineering of Computer-based Systems (ECBS’99)*, Nashville, TN, USA 1999.

[2] D. B. Lange, M. Oshima, “Seven Good Reasons for Mobile Agents”, *Communications of the ACM 42*, [(March 1999), pp. 88-89].

[3] D. Wong, N. Paciorek, D. Moore, “Java-based Mobile Agents”, *Communications of the ACM, 42*, [(March 1999), pp. 92-102].

[4] W. R. Cockayne, M. Zyda, *Mobile Agents*, Manning Publications, Connecticut, 1998.

[5] M. Knapik, J. Johnson, *Developing Intelligent Agents for Distributed Systems*, McGraw-Hill Publishing, 1998.

[6] J. Bloomer, *Power Programming with RPC*, O’Reilly & Associates, Inc., 1992

[7] J. P. Bigus, J. Bigus, *Constructing Intelligent Agents with Java*, Wiley Computer Publishing, 1998.

[8] International Business Machines (IBM), *Documentation distributed with the IBM Agent Building Environment Developer’s Toolkit, Level 6*, 1997.

[9] JavaWorld Web-Page with Java aglets information, http://www.javaworld.com/javaworld/jw-05-1997/jw-05-hood.html.

[10] Matisse, *Computing Glossary Web-Page*, http://www.matisse.net/files/glossary.html.