第四屆行動計算研討會
The 4th Mobile Computing Workshop
National Chiao Tung University, Taiwan, R.O.C.
March 25-26, 1998

A Mobile Raytracing Agent for Internet Computing†

Wen-Shyen E. Chen, C.Y. Lin*, Yao-Nan Lien**, and Huiling Liu***.*

*Institute of Computer Science
National Chung-Hsing University
Taichung, Taiwan, ROC
email: echen@cs.nchu.edu.tw

**Department of Computer Science
National Chengchi University
Taipei, Taiwan, ROC
email: lien@cherry.cs.nccu.edu.tw

***Institute for Information Industry
Taipei, Taiwan ROC
email: huiling@iii.org.tw

Abstract

The Internet has the potential to provide a global, integrative computing platform to share the resources, services and computing power. To explore the full potential of the Internet computing, a new computing paradigm is needed to exploit the Internet infrastructure in a configurable, scalable, and customizable way. Mobile agents have been shown to be promising in the Internet computing environment. However, in the currently proposed approaches, users cannot retain controls after the mobile agents are dispatched. As a result, an extensive study of the issues involved in the management support is needed before the approaches can be commercially viable.

In this paper, we describe the architecture and implmentation of *Pathfinder*, which provides facilities and mechanisms for the mobility and management support of software agents that enable distributed application in the Internet environment. These unique features enable openness and better fault-tolerance. A RayTracing mobile agent application is then developed based on Pathfinder to explore the distributed parallel computing power available in the Internet.

Keywords: Internet Computing, Mobile Agents, Management, Network Infrastructure, Java

1 Introduction

With the advent of network technologies and the popularized Internet, computers are no longer viewed as independent nodes accessing local resources. The Internet has the potential to provide a global, integrative computing platform to share the resources, services and computing power. To explore the full potential of the Internet computing, a new generation of distributed applications, called *mobile code applications* (MCAs) is envisioned. One distinctive feature of MCAs is the provision of "code mobility" to exploit the Internet infrastructure in a configurable, scalable, and customizable way [1-3].

Mobile agents [2, 4-19] are shown to be promising in addressing many issues in constructing MCAs. In this approach, an agent, which is composed of code and data, is submitted by the user and can navigate autonomously through heterogeneous networks. The agent is capable of interacting with servers or other agents, moving to another server while carrying the intermediate results, and resuming execution when it reaches the destination. After the agent is submitted, the user can be disconnected from the network. The user will be notified when the agent finishes its task or is aborted. With this approach, the user is decoupled from the servers in the sense that instead of getting intermediate results many times, the user interacts with the network only when it submits the agent and when the agent returns with results. The scenario is illustrated in Fig. 1.The technical advantages of mobile agents are many: higher bandwidth utilization, support for disconnected operation, support for weak clients, ease of distributing individual service clients, semantic routing, scalability, lower overhead for secure transactions, and robust remote interaction. A non-agent system can exhibit these same features with some work. But the mobile code paradigm supports the transfer of executable code to a remote location for asynchronous execution from the start.

Much effort can be found in the literature to provide mechanisms to enable mobility of software agents [4-19]. The reader is referred to [2] and [19] for taxonomy and review of the approaches. To the best of our knowledge, although the related approaches in the literature have advantages of their own, they either do not provide management functions to the agents at all, or the provision is not complete. In order to be commercially viable, a mobile agent system needs methods for getting information about the agent's current location, applying control functions, etc. Also, important events need to be handled by the agent so that it may react gracefully to exceptional conditions.

†This research was sponsored by MOEA and supported by Institute for Information Industry, R.O.C.

In this paper we describe the architecture of *Pathfinder*, which provides facilities and mechanisms for the mobility and management support of agents to support application in the Internet environment. The Java-based implementation of Pathfinder provides agent execution environment, agent transport protocol, agent-agent interaction, and agent management functions. Based on the implementation of Pathfinder, we then design a ray tracing mobile agent system that exploits the computation power of the Internet, and provides fault-tolerance and management functions at the same time.

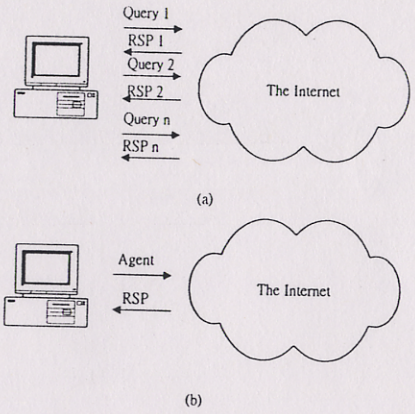

Figure 1. The interactions between the User and the Internet: (a) The Conventional Approach, (b) The Mobile Agent Approach

The rest of the paper is organized as follows. Section 2 gives a review of the related work. Section 3 briefly introduces the open infrastructure we propose to support mobile agents in the Internet computing environment. Section 4 describes the prototype implementation of the infrastructure. Section 5 presents a ray tracing mobile agents system to show the feasibility of the Pathfinder. Concluding remarks and possible future research topics are given in Section 6.

2 Related Work

Much effort can be found in the literature to provide mobility to software agents [4-19]. In this subsection, we survey only four of the most related work.

2.1 IBM Aglets Workbench

Aglets Workbench [10], developed by IBM Tokyo Research Laboratory, is an environment for building mobile agents based on Java [20,21]. An aglet is a program written in Java. It can be executed on one host, be dispatched to a remote host, and resume execution there. The aglet moves with its code, state, and data to another hosts. The major components of Aglets Framework are Java Aglet Application Programming Interface (JAAPI), Agent Transfer Protocol (ATP), Java Agent Transfer and Communication Interface (J-ATCI), and Tahiti.

JAAPI is an interface between aglets and their execution environments. It provides initialization, message handling, dispatching, retracting, suspending, resuming and disposing of the aglet. The programmers can develop platform independent aglets and run them on the hosts that support JAAPI. ATP is an application layer protocol for agent-based system. While mobile agents may be written in different languages and executed on different platforms, ATP provides a standard way for agent transportation. J-ATCI is an interface between agents and their communication protocols for movement and communication in the network. J-ATCI provides JAAPI a communication interface for dispatching agents to another hosts or agent interactions. J-ATCI can use ATP or HTTP as application layer protocol, TCP/IP as underlying protocol. Tahiti is an aglet server and an execution environment for aglets. In addition, Tahiti provides a user interface for users to dispatch or retract aglets.

Implemented in Java, the Aglets Workbench is object-oriented, platform independent, and supports multithreads and exception control. Its ATP provides an open protocol for agent transportation between different hosts. Programmers can develop agents easily based on JAAPI.

2.2 MOLE

MOLE is developed by a research group at Stuttgart University [14]. Implemented in Java, MOLE uses *location* as the place for agent execution. A location can belong to more than one host, and there can be more than one location in the same host. MOLE also implements a class server to provide the classes for agent execution. It offers an engine to manage locations, to provide the services of class server, and to support inter-location communications.

There are two kinds of agents in MOLE: system agent and user agent. System agents can access system resources and provide security control for protection. User agents can execute on different locations, while system agents can execute on a location. They must access system resources or services via system agents. This feature results in better system security. Remote Procedure Call (RPC) is used for the communications between agents. However, network disconnection will result in communication failure.

2.3 Rover Toolkit

MIT's Rover Toolkit [15] offers Relocatable Dynamic Object (RDO) and Queued Remote Procedure Call (QRPC) to build mobile applications. RDOs have four components: mobile code, encapsulated data, an interface,

and the ability to make outcalls to other RDOs. By moving RDOs across the network, applications can move data and/or computation from client to server and vice versa. The QRPC can establish non-blocking RPC for applications when network is disconnected. The message exchange proceeds when network is reconnected. Rover applications can link RDOs dynamically. Therefore, the capacity requirements of memory and disk can be smaller. This is critical for mobile devices, such as PDA. However, a RDO is not an agent in a strict sense --an agent must be equipped with autonomy, and can be controlled. In addition, RDOs managed by normal file system now may become a problem when the amount of RDOs increases.

2.4 Agent Tcl

Agent Tcl [16] is a mobile agent system under development at Dartmouth College. The Agent Tcl architecture has several levels. The lowest level is a transport mechanism, such as TCP/IP protocol stack. The second level is a server that runs on a host. The tasks performed by the server are: keeping track of agents running on its host and answering queries about their states, accepting and authenticating incoming agents, passing the authenticated agent to the appropriate interpreter, and providing a flat namespace for agents communication. The third level of Agent Tcl is an interpreter for available languages. The interpreter provides a security module for security issues, an API for agents to interact the server to handle migration and communication, and a state-capture module for capturing and restoring the state of the agent.

In order to support transparent migration at arbitrary points of agent execution, Agent Tcl modifies the core of Tcl language to provide facilities for capturing the complete internal state of executing agents. Furthermore, it provides authentication and digital signature to enhance the security of the system.

2.5 The Need for Management Functions

To the best of our knowledge, the aforementioned related work, though have advantages of their own, either do not provide management functions at all, or the provision is limited. In order to be commercially viable, a mobile agent system needs methods for getting information about the agent's current location, applying control functions, etc. Also, important events need to be handled by the agent so that it may react gracefully to exceptional conditions. For example, if the user who created the agent cancels the task before it is completed, the agent should abort and return with the intermediate results it has. In the next section, we will propose an infrastructure that takes these factors into account to support Internet computing.

3 The Infrastructure

The Mobile Raytracing Agent is based on an open mobile agent infrastructure, called Pathfinder. The infrastructure is open in the sense that no proprietary protocols are used and anyone who are interested in the infrastructure can participate in using or providing services. The structure of Pathfinder is depicted in Fig. 2.

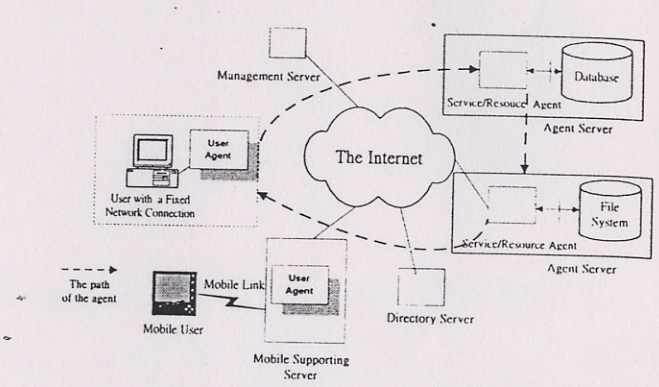

Figure 2. The Structure of Pathfinder.

3.1 Entities in the Infrastructure

The Pathfinder has a modular design, with each entities performs specific functions. The major entities in the infrastructure include Agent Server, Directory Server, Management Server, User Agent, Service/Resource Agent, and Mobile Supporting Server. For a detailed description of Pathfinder, please refer to [22].

3.2 A Sample Scenario

With the infrastructure in place, the system creates a User Agent for the user to carry out the assigned tasks. The User Agent carries the requests for services of the user and consults first with the Directory Server for the Agent Servers that can fulfill the goals or provide the requested services.

With the stored user and agent profiles, and the results from the query, an itinerary is created for the User Agent. The agent is then moved to the Agent Servers according to its itinerary and either being executed in the execution environments provided by the Agent Servers or interacts with their Service/Resource Agents. The User Agent might have to visit more than one Agent Server to have the requested work done. When the requested services are fulfilled or when being called back, the User Agent will return to where it was started.

After the User Agent is submitted, the user who created it can request management functions from the Management Server to retain control over it. With the Mobile Supporting Server, the same architecture can support both

the mobile use and the user with a fixed network connection, as illustrated in Fig. 2.

4 Implementation of the Infrastructure

We have presented in the previous section the conceptual design of Pathfinder for the Internet computing environments. In this section, we describe our implementation of the design and discuss implementation-related issues. The prototype implementation consists of five major components: Agent, Agent Server, Agent Transfer Protocol, Management Server, and Directory Server. The components are shown in Fig. 3.

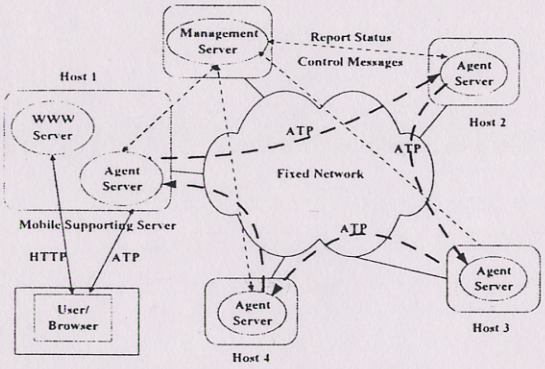

Figure 3: A Prototype Implementation.

4.1 Agents

There are two types of agents in Pathfinder: mobile agent, which moves about the fixed network to carry out the assigned task, and service/resource agent, which stations in various servers to provide services. They are described in the following.

4.1.1 Mobile Agent

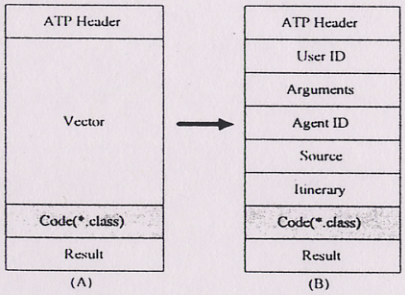

Figure 4. Encapsulate a Mobile Agent in the dispatch message of the Enhanced Agent Transfer Protocol.

The mobile agent in our infrastructure consists of code, itinerary, intermediate results, and related arguments. The encapsulation of the mobile agent in the ATP format is as shown in Fig. 4. Note that in the original ATP specification, the *dispatch* message format only specifies a vector for carrying agent-related information. We further refine the data structure to include UserID, Arguments, AgentID, Source, and Itinerary. The UserID and AgentID specify the unique IDs of the user and the mobile agent, respectively. The Source field is to record the address of the server where the agent was first created (to send back the final results). The Itinerary field stores the location information of the Agent Servers that the mobile agent should visit to carry out the assigned task.

4.1.2 Service/Resource Agent and Agent-Agent Interaction

This type of agents resides in Agent Servers and does not need to have mobility. The Agent Server can dynamically load them to provide services to the mobile agents. The agent-agent interaction is currently implemented with Java's Input/OutputStream classes and Java Object Serialization. The format of the exchanged stream object is not specified in the implementation to provide better flexibility.

4.2 Agent Server

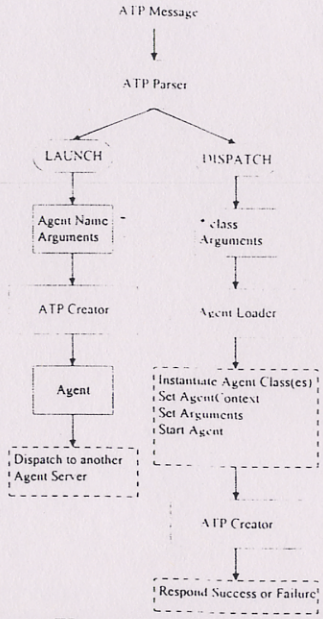

Figure 5. How an ATP message is handled by the Agent Server.

The implementation of the Agent Server consists of the following components.

- **ATP Handler**: send/receive data to/from the network
- **ATP Parser**: Parse the incoming information received from ATP Handler according to the enhanced ATP specification. Obtain the code and arguments of the mobile agent and instantiate the mobile agent code.
- **ATP Creator**: Create a message conforming to the format specified in the enhanced ATP. The message contains ATP header, arguments, agent code, and intermediate results.

- **Agent Security Manager**: Check the action requested by the mobile agent to see if it is legal.
- **Agent Context**: Provide an execution environment for the User Agents.
- **Agent Loader**: Load the instantiated mobile agent and start executing the agent code.

As shown in Fig. 5, whenever an Agent Server needs to dispatch an agent to another server, it first passes the agent to the ATP Creator to package it into ATP format; the resulting message will then be passed to the ATP Handler to be forwarded to its destination. On the other hand, an ATP message received from the network will first be processed by the ATP Handler, and then parsed by the ATP Parser to retrieve the agent code, parameters, and the intermediate results. The agent code is instantiated, sent to the Agent Loader to set Agent Context and parameter values. It then starts execution in the Agent Context.

4.3 Directory Server

The directory server provides the information about which agent servers provide the requested services. It accepts register, modification, and de-register request from the agent servers and answers queries from mobile supporting server. Note that the Directory Server itself is an agent server and it shares the common parts, such as ATP Handler, ATP Parser, and ATP Creator, with the Agent Server. In addition, Request Processor accepts the service requests and interacts with the directory information stored in the database through the JDBC interface.

The mobile agent will first consult with the Directory Server to get the information about which servers to visit in order to accomplish the requested services. However, since the requested services might have to be performed at the Agent Servers in some specific order, an itinerary will be constructed according to the results of the query to the Directory Server and user profiles. The itinerary will be a part of the parameters of a mobile agent.

In a heterogeneous network, the information or services offered by the Agent Servers are diverse and the keyword-based directory service is not appropriate. A content abstract-based directory might be more suitable, and the meta content frame work [23] provides a good implementation direction.

4.4 Management Server

The Management Server is needed to provide management supports, as discussed in Section 3, for the mobile agents. In order to retain controls over the dispatched mobile agents, the user can request management functions from the Management Server. To enable the management support in a mobile agent system, the target agent needs to be located first.

The problem in locating a mobile agent lies in that the agent may move after it reports its current location. In Pathfinder, we have developed mechanisms for agent location: one is to require the mobile agent to periodically report its status, and another is to locate the agent base on the execution time information. The detailed description of the agent location methods is beyond the scope of this paper and the reader is referred to [24, 25]. In the prototype implementation, the first approach is used.

The Management Server also maintains a management table for bookkeeping purpose. The control functions provided by the Management Server are:

1. **launch**: With Pathfinder, a mobile user will first contact the Mobile Supporting Server for an applet, which is used to "launch" a request to the Agent Server. The server will then generate a corresponding agent for the user.
2. **suspend**: An Agent Server can suspend the execution of an agent located in another Agent Server by sending a "suspend" request to that server. The message body should include the agent and user identifiers. The target server should return a response.
3. **resume**: An Agent Server can send a "resume" request to another Agent Server to resume the execution of the previously suspended agent. The message body should include the agent and user identifiers. The target server should return a response.
4. **terminate**: A Management Server can send a terminate message to an agent server to terminate the execution of the target agent. The message should include the user and agent IDs. The target server should return a response.
5. **control:** After an Agent Server receives an agent and the permission to execute is granted, the Agent Server sends a control message to the Management Server, if needed, to report the location of the agent.
6. **report**: A Management Server sends the report message to an Agent Server to ask the target agent to report the intermediate results obtained so far.

4.5 Mobile Supporting Server

The Mobile Supporting Server is the entry point of the mobile user to the fixed network. It consists of a WWW server and an Agent Server with additional functions. In our implementation, the user interface provided to the user is Java-enabled WWW browsers as it is a familiar interface. The user first connects to the WWW server residing in a Mobile Supporting Server through the HTTP protocol and requests a menu (in HTML format) of available services.

The WWW server returns a Java applet for the user to fill in the parameters that are needed for creating a corresponding mobile agent. The WWW browser then sends an ATP message to *launch* the mobile agent to the agent server. (Note that the WWW server does not need to be contacted for this action.)

4.6 Enhanced Agent Transport Protocol

To support agent mobility, a protocol to transport the agent in between servers is needed. We implement this application-layer protocol base on the ATP [10] specification (with some enhancements) developed by IBM Tokyo Research Laboratory. However, the request types currently specified in ATP do not meet the requirements to support the management functions defined in Section 3. For example, an agent should be able to be suspended or terminated by the user, and Agent Servers should be able to deliver an agent to other Agent Servers without the intervention of a centralized controller. In addition, we allow the user to *dispatch* an agent directly into the network -- a function not defined in the original ATP protocol. We have extended ATP by adding the control actions specified in Section 4.4 to the message body. Due to space limitation, the specification of the enhanced ATP is omitted here.

5 Example Agent Application Implementation

Three abstract classes, namely, Agent, ServiceAgent, ServiceAgentProcess, are created for application developers to implement their mobile agent applications and service agents in Pathfinder. Note that the user interface is a WWW browser and the application will present an applet for the user to fill in needed parameters of the application. As a result, the user is shielded from the complexity of creating the agent. We run the prototype implementation on a mix of machines running Solaris, Windows NT, and Windows 95.

In this section, we construct a ray tracing application based on mobile agents to explore the computation power offered by the nodes in the Internet computing environment. Ray tracing is a very elegant but computation-intensive solution to produce the most realistic images to date in computer graphics. It has the characteristics that each pixel can be computed independently from all others. The data independency makes ray tracing a good candidate to be adapted to the Internet computing environment that offers abundant autonomous computers. In [26] the author described a system that employs Java applets to perform the ray tracing computation in Java-enabled WWW browsers. However, the fault-tolerance provided by this approach is weak and there's no management functions offered. We believe that with our mobile agent approach, better performance and quality of services can be achieved.

5.1 Constructing a Mobile Agent

With the abstract classes specified in Pathfinder, an application developer needs the following steps to build a mobile agent application:

- extend Agent.class
- provide the InitDoJob() and DoJob() methods for the application
- get user arguments from UserArgs
- call method writeBack() provided by AgentContext to report results
- call method nextTarget() provided by AgentContext to move to next Agent Server

A RayTracing agent, which inherits the Agent abstract class, is used to demonstrate how Pathfinder works. The RayTrace is a ray tracing code written in Java and is due to [27]. It is being encapsulated in our mobile agent and is dispatched to Agent Servers to be executed in the execution environments provided. When a user first contact the WWW server to request a RayTrace service, the WWW server will return with a Java applet, asking the user to enter needed parameter values. The user interface is as shown in Fig. 6. After the form is filled and the "Go" button is pressed, the arguments for the RayTracing agent is set and the agent is "launched" to the specified agent server.

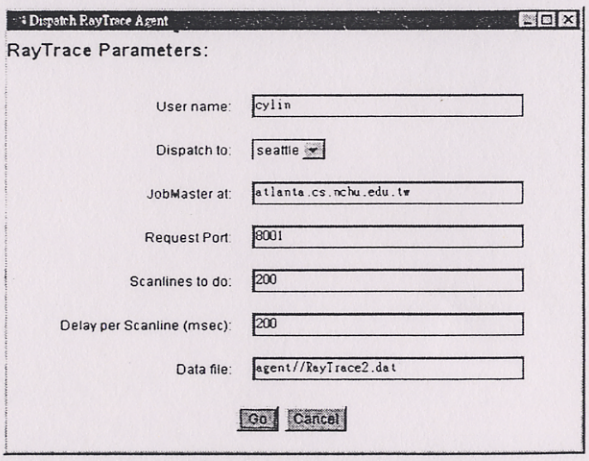

Figure 6: The User Interface of the RayTrace Agent.

5.2 Agent Server and Management Server

Figure 7 shows the interface used by the Agent Server. It shows the contents of Launch Table and the Agent Table. The window on the top shows the related messages when serving mobile agents. Figure 8 depicts the user interface of the Management Server. As can be seen on the top window, the management server provides five control functions: Terminate, Suspend, Resume, Retract, and Report. In addition, it provides the Locate function to request to search for the target agent. The message window shows the control messages received by the management server.