支援行動性及管理功能之行動代理人架構
Pathfinder: A Mobile Agent Infrastructure with Mobility and Management Support

陳文賢　　林敬育　　連耀南
Wen-Shyen E. Chen*†, Ching-Yu Lin** and Yao-Nan Lien***

*國立中興大學資訊科學研究所
*Institute of Computer Science
National Chung-Hsing University
Taichung, Taiwan, ROC

**國立台灣大學電機工程學系
**Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan, ROC

***國立政治大學資訊科學系
***Department of Computer Science
National Chengchi University
Taipei, Taiwan, ROC.

摘　要

網際網路具有提供一個可分享資源、服務及計算能力的全域性與整合性計算平台之潛力。但在完全開發網際網路上的計算能力之前，我們需要一個新的計算行為模式來使網際網路的具有可設定性、可擴展性及可自訂性等特性。行動代理人程式架構在網際網路上提供了一個可設定的、可擴展的並且主動的計算平台，因此被視為具有在網際網路上提供計算能力的希望性。然而，在目前所提出的行動代理人程式架構中，使用者無法對所派出的行動代理人程式進行控制。因此在行動代理人程式架構可被真正在商業上使用之前，包含行動代理人程式的管理功能之相關問題必須被廣泛的研究。

在本論文中，我們設計一系統架構，提供軟體代理人程式行動性及管理功能的機制及便利性，使得軟體代理人程式具有可在網際網路環境下發展分散式應用程式之能力。這些獨具的特性增強了行動代理人系統的開放性、可使用性及容錯能力。我們將描述系統架構之實作並討論其問題。另外，我們亦發展了一名為RayTracing的行動代理人應用程式作為在網際網路上提供分散與平行計算能力的展示。

關鍵詞：行動代理人程式、行動式計算、管理、行動性、架構、網際網路計算、Java

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. They offer a configurable, scalable and active computing platform over the Internet. 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 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 enhance openness, usability, and fault-tolerance of mobile agent systems. An implementation of the infrastructure is then described with the implementation issues discussed. A RayTracing mobile agent application is also developed to explore the distributed parallel computing power available in the Internet.

Keyword：Mobile Agents, Mobile Computing, Management, Mobility, Infrastructure, Internet Computing, and Java

† Corresponding author.

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

should provide a mechanism to bridge the gap between the requests in User Agents and the heterogeneous information offered by Agent Servers.

6. Mobile Supporting Server: The Mobile Supporting Server accepts the request from the mobile user and invokes an instance of the user agent on behalf of the mobile user to carry out the requests. The Mobile Supporting Server holds the profiles of the mobile users it serves. It also provides non-volatile storage for the returned results and notifies the mobile users when the results come in.

3.2 Related Issues

For the infrastructure to be feasible in providing the needed services, some issues need to be addressed.

Transport protocol for Mobile Agents: A transport mechanism is needed for the Agent Servers to move the agents to their destinations. The HTTP [20] protocol might not be feasible, as the client-server protocol does not meet the peer-to-peer interaction requirements for the agent interactions. As a result, a new protocol for agent transport is needed. One possible solution is to adopt the Agent Transfer Protocol (ATP) [10] from IBM Tokyo Research Laboratory as the specification as it seems more complete and is more suitable for agent transport. However, the management functions provided by ATP are not complete, leaving rooms for improvement.

Management Functions: The Management Server needs to hold the status of the mobile agents and to provide control functions to the user. For this to be possible, the management server needs to have a way to locate the mobile agents first. To summarize, the functions it needs to provide are as follows.

Agent Location: How to locate a mobile agent?

Agent Status Report: How to find out effectively if an agent is **running**, **suspended**, **stopped**, **aborted**, or **completed**?

Agent Control: How to apply control functions to the target mobile agents?

In addition, possible control actions that should be provided are:

Launch: Request the creation of a mobile agent.

Dispatch: Sends a mobile agent to the specified Agent Server.

Fetch: Retrieve agent-related information from the specified Agent server.

Terminate: Stop execution of a mobile agent and release the resource held by the agent.

Suspend: Suspend the execution of a mobile agent until a resume message is received.

Resume: Resume the execution of a suspended agent.

Retract: Remove an agent from its current context and move it into the context from which the retraction was requested.

Report: Continue the execution of the mobile agent but request the agent to send back the intermediate results currently available.

With the aforementioned management functions, the mobile agent can be in one of the following states:

Running: The mobile agent is running.

Suspended: The mobile agent has been suspended.

Stopped: The execution of the mobile agent has been canceled and the results sent back to the mobile supporting server.

Aborted: The execution of the mobile agent has been aborted.

Completed: The mobile agent has been successfully completed.

The life cycle of the agent, after it arrives at an Agent Server, is depicted in Fig. 3. Note that the mobile agent will be instantiated with the accompanying arguments when it first arrives at an Agent Server. The lifecycle of the mobile agent is very similar to that of a *process* in a UNIX environment. The result is intentional, as we believe that providing user with a familiar perception of the environment will increase the acceptance of Pathfinder.

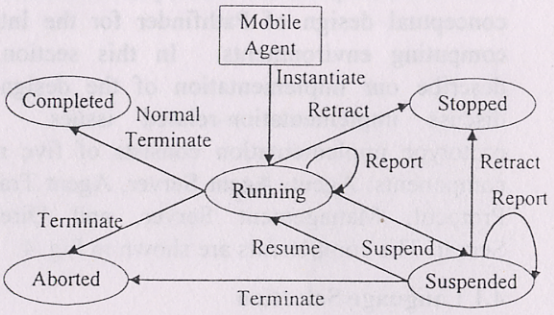

Figure 3. The Life-Cycle of a Mobile Agent.

3.3 A Sample Scenario

4.2.1 Mobile Agent

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. 5. 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.

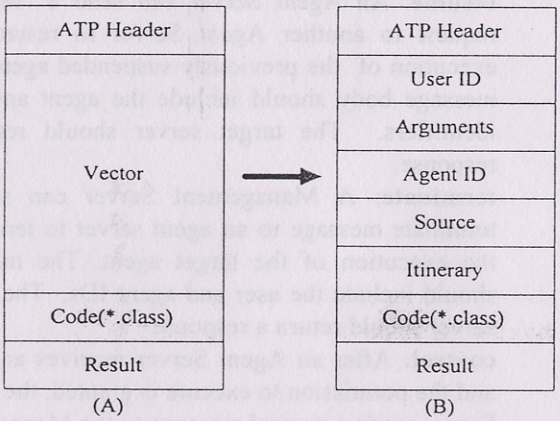

Figure 5. Encapsulate a Mobile Agent in the dispatch message of the Enhanced Agent Transfer Protocol.

4.2.2 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.

4.3 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. 6, 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.

4.4 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

4.7 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 4.5 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. They are listed in Appendix A for reference. 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. The prototype implementation has been run on a mix of machines running Solaris, Windows NT, and Windows 95.

In this section, in order to demonstrate the feasibility of Pathfinder, 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 Appendix A, an application developer needs the following steps to build a mobile agent application:

1. extend Agent.class
2. provide the InitDoJob() and DoJob() methods for the application
3. get user arguments from UserArgs
4. call method writeBack() provided by AgentContext to report results
5. 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. 7. 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. The code fragment for the RayTracing agent is omitted here due to space limitation.

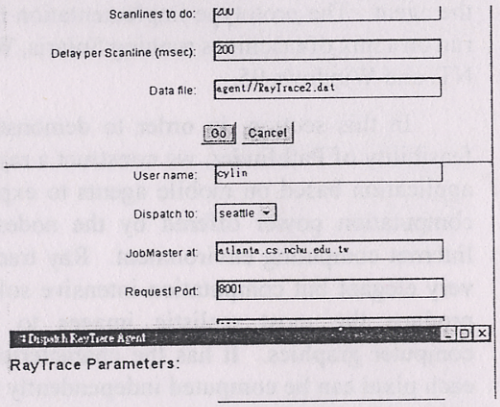

Figure 7: The User Interface of the RayTrace Agent.

5.2 Agent Server and Management Server

Figure 8 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 9 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, while the management table window at the bottom shows the status of the mobile agents being managed.

Figure 10 shows the intermediate results reported by the RayTracing Agents. (Therefore, there are three sections showing the progress of the execution.) Note that there are three dispatched mobile agents and they can be individually controlled to achieve better load balancing and fault-tolerance. Figure 11 shows that the three RayTracing Agents have been suspended by the request of the user. The suspended agents can later be "resumed" by the management server. Figure 12 is the result reported by the RayTracing Agents.

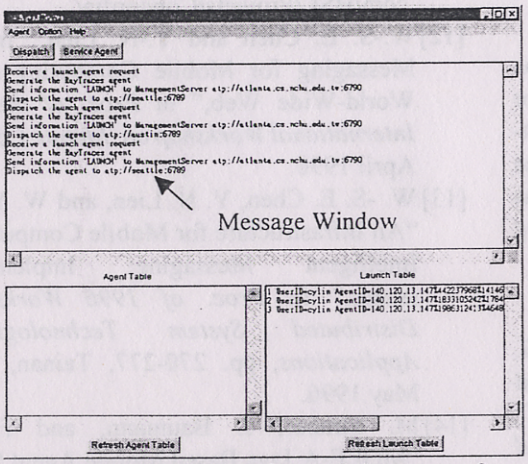

Figure 8: The User Interface of the Agent Server with Agent Table and Launch Table.

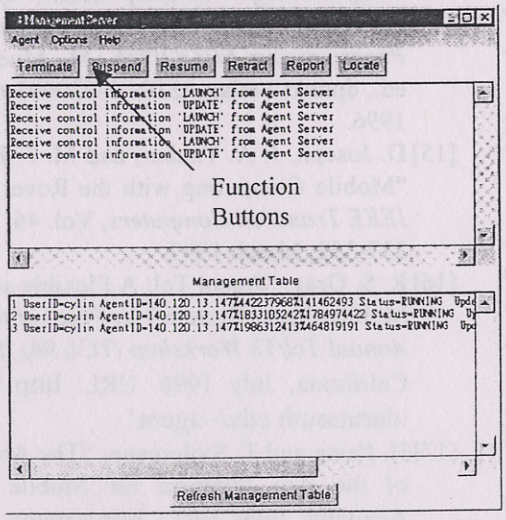

Figure 9. The User Interface of the Management Server.

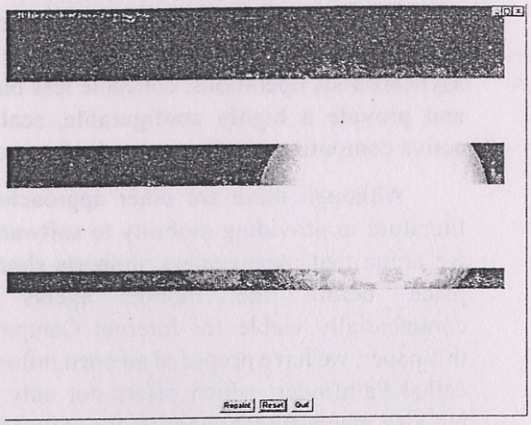

Figure 10: The Intermediate Results Reported by the RayTraacing Agents.

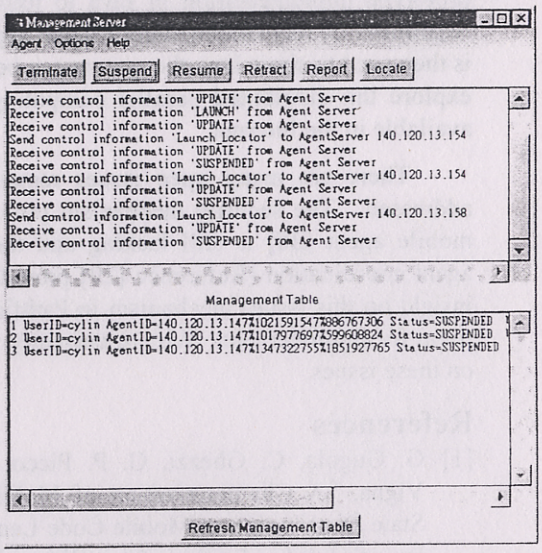

Figure 11: Suspend the Execution of RayTracing Agents.

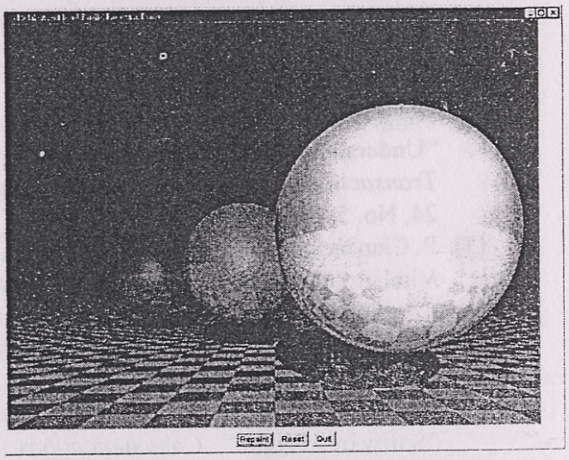

Figure 12: Final result reported by the RayTracing Agents.

6 Conclusion

[18] W. Li and D. G. Messerschmitt, "Java-to-Go," Available from http://ptolemy.eecs.berkeley .edu/dgm/javatools/java-to-go/

[19] J. Kiniry and D. Zimmerman, "A Hands-On Look At Java Mobile Agents," *IEEE Internet Computing*, Vol.1, No. 4, pp. 21-30, July/August 1997.

[20] R. Fielding, *et al.*, "Hypertext Transfer Protocol -- HTTP/1.1," Available from http://www.w3.org/Protocols/History.html#Rev06, November 1998.

[21] "The Java Language Environment: A White Paper," Sun Microsystems Computer Company, May 1995.

[22] Java Development Kit, Available from http://www.javasoft.com/products/jdk/1.1/index.html

[23] R. V. Guha, "Meta Content Framework," Available from http://mcf.research.apple.com/hs/mcf.html

[24] Y. -N. Lien and C. -W. R. Leng, "On the Search of Mobile Agents," in *Proc. of the 7th IEEE Symp. Of Personal, Indoor, and Radio Communications,* pp. 703-707, October 1996.

[25] W.-S. E. Chen, C.-W. R. Leng, and Y.-N. Lien, "A Novel Mobile Agent Search Algorithm," in *Proceedings of ICCCN'97*, Las Vegas, September 1997.

[26] L. Vanhelsuwe, " Create Your Own Supercomputer with Java," in *Java World*, January 1997. Available from http://www.javaworld.com/javaworld/jw-01-1997/jw-01-dampp.html

[27] Raytracer written in Java, by Frederico Inacio de Moraes.

[28] G. Karjoth, D. Lange, and M. Oshima, "A Security Model for Aglets," *IEEE Internet Computing,* Vol. 1 No. 4, pp.68-77, July/August 1997.

[29] P. Ciancarini, *et al.,* "Coordinating Multiagent Applications on the WWW: A Reference Architecture," *IEEE Transactions on Software Engineering*, Vol. 24, No. 5, May 1998.

[30] W.-S. E. Chen, S. T. Su, H. T. Shu, Y.-N. Lien, and H., Liu, "Mobility and Management Support for Mobile Agents," in *Proceedings of Autonomous Agents '98*, Minneapolis/St. Paul, May 1998.

`public void setArguments()`
Set Agent arguments to default values (NULL).

- **setArguments**
  `public void setArguments(Vector ar)`
  Set Agent arguments by parsing `ar`. The variables `UserArgs`, `AgentID`, `Source`, and `Itinerary` are assigned separately by the result of the parsing.

- **ConnectToServiceAgent**
  `public void ConnectToServiceAgent(String sname)`
  Connect to the specified Service Agent (sname) for services

- **SendMessage**
  `public void SendMessage(RequestMessage requestm)`
  Send information to the Service Agent using `RequestMessage` format.

- **ReceiveMessage**
  `public ResultMessage ReceiveMessage()`
  Receive results from the Service Agent using `ResultMessage` format.

- **Disconnect**
  `public void Disconnect()`
  Close the connection between the Mobile Agent and the Service Agent.

- **getLocalAddr**
  `public String getLocalAddr()`
  Get the IP address of the server where the Mobile Agent currently resides.

- **getLocalHostName**
  `public String getLocalHostName()`
  Get the domain name of the server where the Mobile Agent currently resides.

- **nextTarget**
  `public void nextTarget()`
  Move to the next server. If it is a final destination, return results.

- **nextTarget**
  `public void nextTarget(String atpAddress)`
  Move to the next server (`atpAddress`).

- **reportResult**
  `public void reportResult()`
  Report the results back to Agent Server where the Agent first started.

- **InitDoJob**
  `public void InitDoJob()`
  For the initialization of the Mobile Agent. Provided by application developer.

- **DoJob**
  `public void DoJob()`
  Provide the main functions of the Mobile Agent. Provide by application developer.

Class agent.ServiceAgentProcess

```
java.lang.Object
   |
   +--java.lang.Thread
         |
         +--agent.ServiceAgentProcess
```

```
public abstract class ServiceAgentProcess
extends Thread
```

The `ServiceAgentProcess` is an abstract class. It is instantiated when called by a Service Agent. `ServiceAgentProcess` communicates with other Agents by the socket passed from Service Agent.

Variables

- **AgentID**
  `protected String AgentID`
  Agent ID.

- **socket**
  `protected Socket s`
  Passed from Service Agent and used for communicating with other Agents.

Constructor

- **ServiceAgentProcess**
  `public ServiceAgentProcess(Socket client_socket, String aid)`
  The default constructor of `ServiceAgentProcess`. The variable `client_socket` receives the socket passed from the Service Agent, and the variable `aid` is an Agent ID.

Method

- **SendMessage**
  `public void SendMessage(ResultMessage resultm)`
  Send the result (in `ResultMessage` format) to a Mobile Agent using.

- **ReceiveMessage**
  `public RequestMessage ReceiveMessage()`
  Receive request (in `RequestMessage` format) from a Mobile Agent.

- **Disconnect**
  `public void Disconnect()`
  Close the connection between the Service Agent and the Mobile Agent.