A Reactive Planner-Based Mobile Agent System

Whang-Hee Seok[a] and In-Cheol Kim[b]

[a]*Department of Computer Science, Kyonggi University*
San94-6, Yiui-Dong, Paldal-Gu, Suwon, 442-760, KOREA
Tel : 82-31-249-9670, Fax : 82-31-249-9673

[b]*Department of Computer Science, Kyonggi University*
San94-6, Yiui-Dong, Paldal-Gu, Suwon, 442-760, KOREA
Tel : 82-31-249-9669, Fax : 82-31-249-9673

Abstract

Mobile agents have the unique ability to transport themselves from one system in a network to another. The ability to travel allows mobile agents to move to a system that contains services with which they want to interact and then to take advantage of being in the same host or network as the service. But most of conventional mobile agent systems require that the users or the programmer should give the mobile agent its detail behavioral script for accomplishing the given task. And during its runtime, such mobile agents just behave according to the fixed script given by its user. Therefore it is impossible that conventional mobile agents autonomously build their own plans and execute them in considering their ultimate goals and the dynamic world states. One way to overcome such limitations of conventional mobile agent systems is to develop an intelligent mobile agent system embedding a reactive planner. In this paper, we design both a model of agent mobility and a model of inter-agent communication based upon the representative reactive planning agent architecture called JAM. An then we develop an intelligent mobile agent system with reactive planning capability, IMAS, by implementing additional basic actions for agent moves and inter-agent communication within JAM according to the predefined models. Unlike conventional mobile agents, IMAS agents can be able to adapt their behaviors to the dynamic changes of their environment as well as build their own plans autonomously. Thus IMAS agents can show higher flexibility and robustness than the conventional ones.

Keywords:

Mobile Agent, Reactive Planner, JAM

Introduction

A software agent that migrates from one remote host in a network to another along with its data, codes and state information to perform a given task is called mobile agent. Recently, interests in mobile agent have emerged in various fields, and consequently a variety of systems has been implemented. Major examples are Telescript [5], Agent Tcl (Tool Command Language) [2], Ara (Agent for Remote Action) [11], TACOMA (Troms And COrnell Moving Agents) [12], Aglets Workbench [9], Voyager [14], Odyssey [15], and Concordia [13]. Also, application systems employing mobile agents have been developed and they include Mobile Computing Devices, Network and Distributed System Management, Remote Diagnosis, Electronic commerce, Internet Information Retrieval, Distributed Simulation, and Virtual Enterprise.

Most of the existing mobile agent systems require users or programmers to enter extensively detailed behavior script to perform assigned tasks. Such an agent system is limited in its functionalities in that it simply acts as described in this fixed script at runtime. Therefore, it is impossible that an agent dynamically formulates and executes a plan taking into account the goal and the environment at execution time.. In order to enter the detailed behavior script prior to runtime, users or programmers need to acquire complete information about dynamic working environment that the agent will face with. However, in most of environment adopting mobile agents that leave users and migrate to work in various remote heterogeneous hosts, predicting all possible situations that these agents might face with is extremely difficult, if not impossible. The network and system faults, or internal composition changes in the platforms, interruption and delay in network transfer due to overloads, and others are all possible incidents. Therefore, the existing mobile agent systems that follow fixed script given on the basis of incomplete prior knowledge are incapable of dealing with the changes effectively when these unexpected incidents occur.

One way to address the problem of mobile agent system in that it is restricted in the aspects of autonomy, reactivity and robustness is to develop an intelligent mobile agent system based on reactive planner [7][8]. In general,

whereas the plan generation and plan execution phases are two separate, sequential processes in traditional deliberative planners, reactive planners have the advantage that they can respond to changes in the environment dynamically and sensitively, because plan generation, execution, and detection of changes in the real world and the consequent plan modifications are integrated into one process and performed in turn. In order to implement an intelligent mobile agent system that possesses mobility and communication functionality on the basis of reactive planner, we need to establish an efficient model for mobility and communication agent, to implement primitive actions for mobility and communication on the basis of these models, and provide the agent with an internal reactive planner. An intelligent mobile agent system like this requires users to declaratively enter the goal and domain knowledge for the agent conforming to a unique method of expression, instead of programming specific behavior script of mobile agent directly in specific programming or script language. The agent itself can establish dynamically plans necessary to achieve the goal and execute them. The plans of a mobile agent contain mobility and communication actions basically provided by the system as well as the actions related to domain knowledge entered by the user. Also a reactive planner which is the core of a mobile agent architecture monitors the real world while executing the plans, and if it detects the situation changes and as a consequence of the event it cannot perform the remaining plans, it immediately attempts to generate new plans or modify without direct interruption by the user.

On the basis of JAM [7], well-known reactive plan agent architecture, we designed a mobility model and a communication model between agents. By applying these models and implementing additional primitive actions for mobility and communication of agent on JAM, we developed an intelligent mobile agent system called IMAS. IMAS is a mobile agent that can establish and execute the plans autonomously. If IMAS is provided with declarative domain knowledge for application from a user, it can be used as an agent development tool that one can easily develop various application mobile agents. Behaviors of mobile agents created using these IMAS will be guided by plans containing mobility and communication actions we extended for this research. Unlike the existing mobile agents, IMAS mobile agents will display more flexibility and robustness, because they can efficiently adapt to the changes in dynamic environment and establish or modify their own plans for action with respect to the goal.

In the following sections, we will briefly introduce JAM [7], a major reactive plan agent architecture, on which we implemented IMAS. We also describe the design and implementation of IMAS in detail, and then conclude.

2. Architecture of Reactive Plan Agent

2.1 Basic Concepts

In general, an agent architecture that adopts a planner as the core of internal components is called *plan agent architecture*. The conventional artificial intelligent planner and/or planning system regards plan generation and plan execution as two separate, independent processes. That is, in plan generation phase, the goal to be achieved by the agent, current state, and possible actions are expressed and presented to the system in symbolic logic. Based on this knowledge, the system determines the sequence of actions to be taken to achieve the goal. For the plan execution phase, a sequential control architecture is expected to control a separate runner that executes these generated plans sequentially. These conventional artificial intelligent planners are called *deliberative planners*. It is assumed in the traditional deliberative planners that the planner acquires all complete and correct information about real world by the plan generation phase. Thus, the system can not modify the plans or even re-plan promptly in response to the changes in the surrounding conditions when unexpected events or changes occur in the middle of plan generation or plan execution process. This has been noted as a weakness of the conventional planning system. In contrast with deliberative planners, the planners that promptly respond to environmental changes by performing plan generation, plan execution and plan modification within a planner simultaneously or in parallel, are called *reactive planners*. Examples of these reactive planners are PRS (Procedural Reasoning System)[16], TouringMachine and InteRRaP.

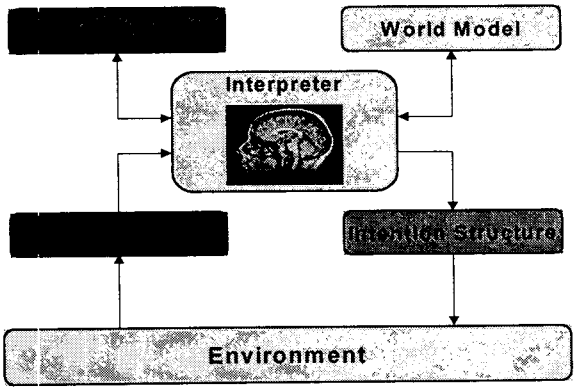

Figure 1 - Components of JAM

2.2 JAM

JAM [7] is an intelligent agent architecture which grew out of many researches descending from PRS (Procedural Reasoning System) which was proposed by Georgeff [8]. Several functionalities have been added and extended onto PRS plan agent architecture, and JAM is a re-implementation of PRS in Java language. JAM consists of five components as shown in *Figure 1*. Through the interaction of these five components, JAM responds to the real world. It brings about cyclic effect by repeating the internal processing triggered by the changes imposed from external environment, and then influencing the external environment applying the result of the process.

(1) World Model: Information held by an agent regarding the real world. Facts or beliefs held by the agent are represented as follows:

```
FACTS:
    FACT robot_status "Ok";
    FACT robot_position 10000 10000 0;
```

(2) Plan Library: A set of abstract plans, which are given by the agent designer in the initial phase of development. An abstract plan is described as follows:

```
Plan:
{
    GOAL: [goal specification]
    NAME: [string]
    BODY: [procedure]
    <DOCUMENTATION: [string]>
    <PRECONDITION: [expression]>
    <CONTEXT: [expression]>
    <UTILITY: [numeric expression]>
<FAILURE: [non-subgoaling procedure]>
    <EFFECTS: [non-subgoaling procedure]>
    <ATTRIBUTES: [string]>
}
```

(3) Interpreter: The interpreter creates sub-goals and specific plans, and executes them after the agent designer provides declaratively the agent with the world model, the goal, and a set of available abstract plans.

(4) Intention Structure: The Intension structure represents subgoals that the agent is pursuing in order to achieve its goal, and states of specified plans that agent is currently executing.

(5) Observer: During reasoning cycles of interpreter, the observer monitors changes in the external world. The resulting observation is described as given below:

```
OBSERVER:
{
    RETRIEVE initialized $VALUE;
    WHEN: TEST(== $VALUE "False"){
        EXECUTE setShowAPL 1;
        EXECUTE setShowGoalList 1;
        EXECUTE setShowWorldModel 1;
        EXECUTE setShowIntentionStructure 1;
        UPDATE (initialized) (initialized "True");
    };
}
```

Following is an example of JAM program that contains each of the five components mentioned above:

```
GOALS: ACHIEVE cycle_tested;
FACTS: test_done    "False";
       system_init   "False";
       cycle_number   0;
OBSERVER {
    RETRIEVE cycle_number $N ;
    EXECUTE print "\n Cycle#" $N "\n";
    UPDATE (cycle_number $N) (cycle_number (+ $N 1));
}
PLAN {
NAME: "Test CYCLE"
GOAL: ACHIEVE cycle_tested;
CONTEXT: FACT test_done "False";
BODY:
    EXECUTE print "\n Normal execution started. \n";
    UPDATE (system_init) (system_init "True");
    WHILE: TEST (== 1 1) {
        EXECUTE noop;
    };
}
```

In order to facilitate its application development, JAM provides basic actions within each plan that can be triggered using "EXECUTE" command. In addition, JAM allows programmers to easily define additional actions in Java language. One of the important functions provided in JAM is check-point functionality that is to record execution states of the agent. This check-point accumulates information which becomes an important basis for cloning, restoring, and migrating of the agent.

3. Design of Intelligent Mobile Agent System

For this study, in addition to develop an intelligent agent equipped with a planner based on JAM, we intend to implement mobility and communication functions of an agent, two of the most important functionalities required for intelligent agent systems. Initially, it is necessary to design major components necessary for mobility and communication of an agent and then to formulate the models defining the relations and interaction among these components. In this section, we describe the design of mobility model and communication model that IMAS, the intelligent mobile agent system we implemented, is based on.

3.1 Mobility Model of Agent

In order to implement an intelligent mobile agent system, we designed mobility model of agent that consists of three components -- place, mobile agent, mobility server. The

function of these three components is depicted in *Figure 2*.

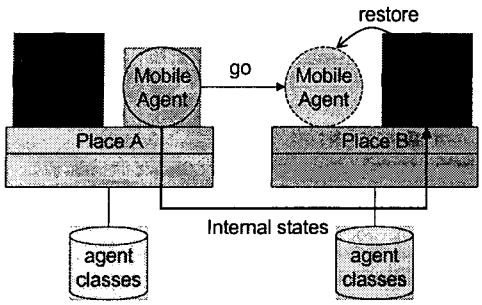

Figure 2 - Mobility Model of Agent

(1) Place: The place is a virtual work area that mobile agents run in a distributed network. The place can be one or more locations within one host computer physically connected to a network. Each place must contain one mobility server that runs all the time. All the mobile agents reside in distributed places in a network, constantly performing their tasks, except when they migrate between different places.

(2) Mobility Server: A mobility server is assigned to each place. It provides assistance to a mobile agent when it attempts to migrate into the local machine. Each mobility server maintains an address consisting of the port number and the host name. When a remote mobile agent wants to migrate into the location where the mobility server resides, the mobility server restores the mobile agent in the local machine after receiving its current internal state information through socket connection to the agent.

(3) Mobile Agent: A mobile agent is autonomous software that performs useful tasks for a user, migrating to many places on a distributed network, aided by the mobility server. In order to move to certain remote place, a mobile agent establishes socket connection by using physical address of mobility server running on that place. Once connected to the mobility server of destination, the agent transfers serialized internal state through the socket. With the assistance of mobility server in the destination, the mobile agent is restored to the state prior to migration, and resume execution in the destination place.

We present the process of agent migration in more detail as follows. First, there should be a mobility server running constantly in the destination place. While running, the mobility server waits for internal state information from mobile agents, monitoring the designated port. The mobile agent then executes migrating action, "Go" command, within its execution plan. When this migration action is triggered, the agent attempts to connect through socket to the mobility server in destination place. When the connection is established, it causes the internal state information to be serialized and sent to the mobility server of destination through the socket. Finally, the mobility server of destination that receives internal state information of the agent through the socket, restores the mobile agent using the received state information and class files for this mobile agent in the destination file system.

3.2 Communication Model between Agents

In this section, we present our design of a communication model between agents as shown in *Figure 3*. It comprises two basic actions for communication, MessageServerAgent and MessageClientAgent whose functions are displayed in *Figure 3*.

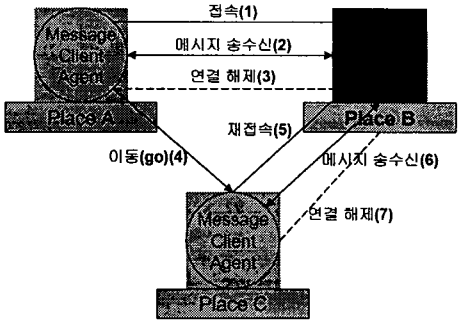

Figure 3 - Communication Model between Agents

(1) Message: We assume that communication between agents is primarily composed of actions of sending and receiving messages in character string data type. We also assume that the message exchange is enabled by the socket connection between server side and client side as it is the case for agent migration. Asynchronous messaging of disconnection style, like email, is also available, but in the system for this research, only synchronous messaging through the socket connection is supported. Through the socket connection, byte stream converted from character string, various multimedia information, or objects can be transferred, but in the model for our research, only messaging of most primitive character string type such as standard communication language between agents, called KQML, is supported.

(2) Message Server: The message server resides in one place in a distributed network, and exchanges the messages with certain remote mobile client through the socket connection. For this, each message server agent holds host name and address, and it communicates with many clients that attempt to connect to using the address.

(3) Message Client: Unlike a MessageServer agent that resides in one place, the MessageClient agents constantly move to different places in a distributed network and can communicate with MessageServer agent. In order to resume communication after they move to another place, MessageClient agents must terminate the connection to MessageServer agent before moving and then restore the connection to MessageServer agent through the socket after reaching the destination

Let's suppose that a MessageClient agent is communicating with a MessageServer agent in a remote host and a need to move to a new location arises, and it also needs to communicate with other mobile agents while migrating to the destination. The communication messages that take place between agents can be decomposed into several step units. First, a MessageClient agent running in place A requests the socket connection by triggering "connectToAgentAsClient" action in order to communicate with MessageServer agent in

a remote place B. If this succeeds, the agent communicates with MessageServer agent through the socket. In order to move to a new remote place C, in the middle of communication with MessageServer agent in place B, the MessageClient agent disconnects the socket. And after disconnecting, the MessageClient agent moves to the remote destination place C by executing "Go" command. If this migration succeeds, again MessageClient agent performs "connectToAgentAsClient" action to communicate with MessageServer agent and tries to establish socket connection again. Once it is connected, the MessageClient agent communicates normally as before. When it needs to terminate communication, or needs to move again to a new place, it disconnects itself from MessageServer agent. By performing the process described above repeatedly, the agents can communicate while migrating.

4. Implementation of Intelligent Mobile Agent System

4.1 Implementation of Agent Mobility Model

For this study, we implemented in Java mobility servers that reside in individual hosts and help migration of agents based on the mobility model of agents described in Section 3. In addition, we extended JAM architecture so as to create mobile agents by implementing "Go" command which is a basic action for agent migration, on this reactive planning agent architecture.

Figure 4 shows an execution scenario of an agent performing tasks moving along three different distributed hosts. The plans of the agent that this execution scenario is based on are triggered by "Go" action and listed in *Table 1*.

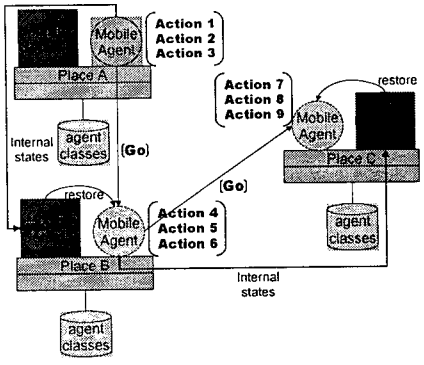

Figure 4 - Mobility Scenario of one Agent

Figure 5, *Figure 6*, and *Figure 7* show the results of tasks performed, which a mobile agent accomplished moving to the remote hosts one after another following the plans

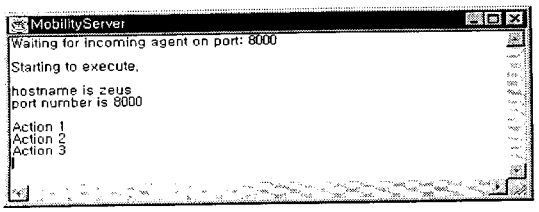

Figure 5 - Execution on Host A

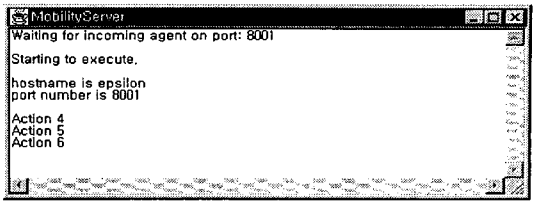

Figure 6 - Execution on Host B

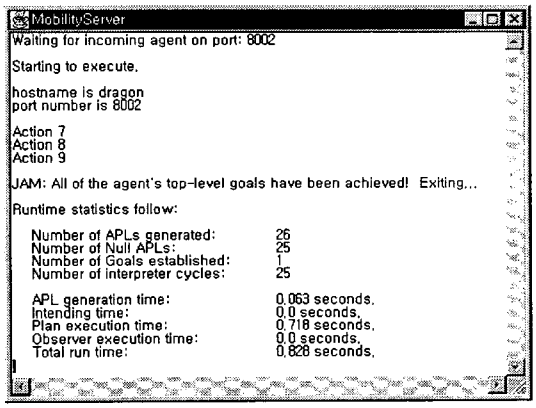

Figure 7 - Execution on Host C

Table 1 - Mobility Plan of Agent

```
BODY:                    . . . .
EXECUTE com.irs.jam.primitives.GetHostname.execute
                $hostname;
ASSIGN $port 8000;
EXECUTE print "hostname is " $hostname"\n";
EXECUTE print "port_number is " $port "\n";
EXECUTE print "Action 1\n";
EXECUTE print "Action 2\n";
EXECUTE print "Action 3\n";
// Performe to given host name and port number.
ASSIGN $hostname "epsilon.kyonggi.ac.kr"
ASSIGN $port (+ $port 1)
EXECUTE Go $hostname $port;
EXECUTE com.irs.jam.primitives.GetHostname.execute
                $hostname;
EXECUTE print "hostname is " $hostname "\n";
EXECUTE print "port_number is " $port "\n";
EXECUTE print "Action 4\n";
EXECUTE print "Action 5\n";
EXECUTE print "Action 6\n";
                         . . . .
```

4.2 Implementation of Communication Model between Agents

We also implemented in JAM [7] architecture several primitive actions. These are, "connectToAgentAsServer, connectToAgentAsClient, sendMessage, recvMessage, disconnect" for communication between agents based on communication model that we designed in chapter 3. This

implement enabled JAM agents to generate the plan for migration and communication using these primitive actions, and play the roles of message server agent and message client agent. *Figure 8* shows an example of communication scenario between agents. This example depicts a communication scenario that a mobile agent in place B moves to C and D in turn, and exchange messages with other agent in place A. The mobile agent in the example, called MessageClient, disconnects itself in the middle of communication with MessageServer agent through socket located in place B, moves to another place, and establishes again the connection to the MessageServer.

The primitive actions defined for communication between agents are given below:

(1) Connection Action of MessageServer: This resides in one place in a network waiting for a request for connection from a remote mobile MessageClient through the socket. For this functionality, a MessageServer agent has an address consisting host name and address, and using the address, it is able to communicate with many MessageClients requesting connections.

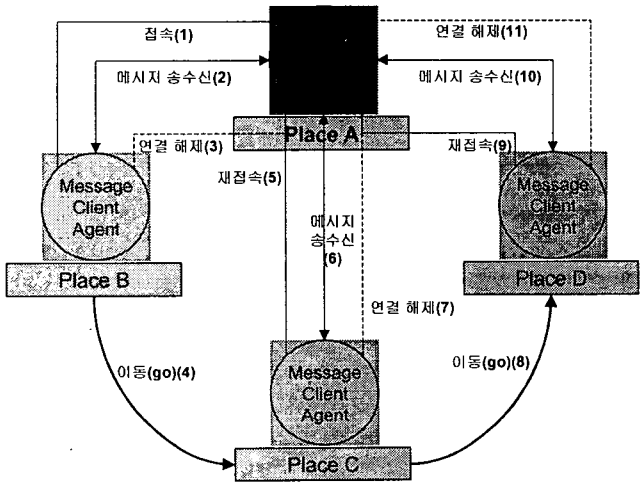

Figure 8 - Communication Scenario between Agents

(2) Connection Action of MessageClient: This is an agent that moves to different places in a network and can communicate with a MessageServer agent. In order to communicate with a MessageServer agent when a MessageClient agent moved to different places, the MessageClient agent must terminate the connection to the MessageServer agent that it was communicating with before moving, and then re- connect to the MessageServer agent through the socket after reaching the destination.

(3) Message Sending Action: This is an action for sending messages in character string data type from a MessageServer agent or MessageClient agent in one place to MessageServer agent or MessageClient agent in other place through the socket.

(4) Message Receiving Action: This is an action for receiving messages of string type which are sent from MessageServer agent or MessageClient agent in other place through the socket.

(5) Disconnection Action: This is an action executed when an agent does not need to communicate any further, or when an agent attempts to move to a new place. This action causes the socket of MessageServer agent to be disconnected.

Table 2 shows the communication plans between agents that we implemented for research, and specifically a part of plans for a MessageClient agent. It displays the process that an agent repeatedly communicates with the MessageServer agent, disconnects itself from the MessageServer agent, and then reconnects to it.

Table 2 - Communication Plan of MessageClient Agent

```
BODY:                    ...    ...    ...
ASSIGN $x 0;
WHILE: TEST (< $x 5) {
   EXECUTE print "Sending server a message of
         \"I will quit!!!!!!!!!!\"." "    " $x "\n";
   EXECUTE com.irs.jam.primitives.sendMessage.execute
         $OUT "I will quit !!!!!!!!!!";
ASSIGN $x (+ $x 1); };
EXECUTE print "Waiting for server reply.\n";
EXECUTE com.irs.jam.primitives.recvMessage.execute
      $IN $MSG;
EXECUTE print "Server reply was:\""$MSG"\".\n";
EXECUTE      com.irs.jam.primitives.disconnect.execute
8000 "localhost" $socket $IN $OUT;
EXECUTE Go "epsilon" 8000;
EXECUTE
com.irs.jam.primitives.connectToAgentAsClient.execute
8000 "localhost" $socket $IN $OUT;
...    ...    ...
}
```

Figure 9 and *Figure 10* present the results of communication between a MessageServer agent and a MessageClient agent.

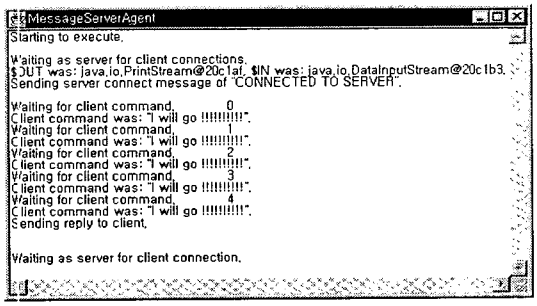

Figure 9 - Execution Result of MessageServer Agent

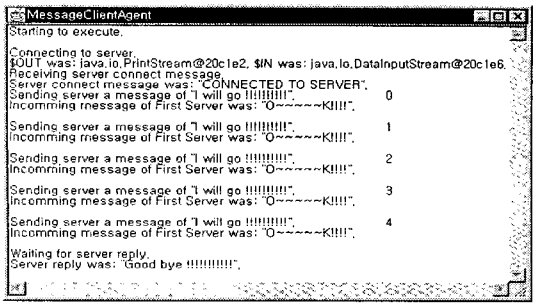

Figure 10 - Execution Result of MessageClient Agent

5. Conclusion

In this paper, we present out implementation of an intelligent mobile agent system equipped with planning capability. In order to develop the system, we designed and implemented mobility actions of agent and communication actions between agents on the basis of an existing reactive plan system, called JAM [7]. The main feature of this system is that it can establish and execute the given tasks requiring mobility actions and communication actions through reasoning on the basis of a internal reactive planner, and that it can modify the plans or re-plan promptly responding to the changes of external environment that might occur in the middle of execution. An intelligent mobile agent system developed for this research can be used as a development tool that can be easily developed into various application mobile agent systems, if provided with application domain knowledge by the user. However, the mobility and communication models of this system have several weaknesses. Firstly, when there are several mobile agents that attempt to enter simultaneously into the same place, a MobilityServer is not capable of handling the problem by providing concurrent services to these mobile agents. Secondly, the system is less reliable in that it provides no alternative means to correct the situation when an agent fails to migrate, for example, due to disruptions in serialization or restoration phase. For communication models, it is first noted that, as in the case of mobility model, it provides no functionality for concurrent message processing on the part of MessageServer. Secondly, it does not provide communication mechanism with asynchronous agents as email. Thirdly, the messages between agents do not take the format of a interagent communication language which is characterized by its rigid grammar and semantics. The area of future work will be to mitigate these limitations of IMAS, the intelligent mobile agent we implemented and described in this paper.

Reference

[1] C. G. Harrison, D. M. Chess, A. Kershenbaum, "Mobile Agents : Are they a good idea?", Technical Report, IBM T.J. Watson Research Center, 1995.

[2] Robert S. Gray. Agent Tcl: A transportable agent system. In Proceedings of the CIKM Workshop on Intelligent Information Agents, Fourth International Conference on Information and Knowledge Management (CIKM 95), Baltimore, Maryland, December, 1995.

[3] J. Baumann., F. Hohl. K. Rothermel., and M. StarBer., "Mole-Concepts of a Mobile Agent System," World Wide Web, 1(3):123-137, September 1998.

[4] Jaeho Lee, Edmund H. Durfee, and Patrick G. Kenny, Marcus J. Huber, UM-PRS : An Implementation of the Procedural Reasoning System for Multirobot Applications, Proceedings of CIRFFSS 94, pp. 842 - 849, 1994.

[5] J. E. White, "Telescript Technology: Mobile Agents," General Magic White Paper, Appeared in Bradshaw, J., Software Agents, AAAI/MIT Press, 1996.

[6] J. E. White, "Mobile Agents", Software Agents, MIT Press and American association for Artificial Intelligence, 1997.

[7] Marcus J. Huber, "Jam Agents in a Nutshell", http://members.home.net:80/marcush/IRS, 1998.

[8] Michael Georgeff and Amy L. Lansky, Reactive Reasoning and Planning, In Proceedings of the Sixth National Conf. on Artificial Intelligence (AAAI-87), pp. 677 - 682, Seattle, WA, 1987.

[9] Mitsuru Oshima, Guenter Karjoth, and Kouichi Ono, Aglets Specification 1.1,

[10]http://www.trl.ibm.co.jp/aglets/spec11.html, 1998.

[11]S. Hyacinth Nwana, "Software Agents : An Overview", Intelligent Systems Research, Advanced Applications & Technology Department, Knowledge Engineering Review, Vol. 11, No 3, pp. 205 - 244, October/ November 1996.

[12]H. Peine, and Stolpmann, T., "The Architecture of the Ara Platform for Mobile Agents," Proceeding of the First International Workshop on Mobile Agents(MA'97), Berline, Springer Verlag, LNCS 1219, pp.50-61, April 1997.

[13]Dag Johansen, Robbert van Renesse and Fred B. Schneider, Operating system support for mobile agents , Proceedings of the 5th. IEEE Workshop on Hot Topics in Operating Systems, Orcas Island, Wa, USA(4th-5th May, 1995), Published by: IEEE Computer Society, NY USA, May 1995.

[14]Walsh, T., Paciorek, N., and Wong, D., "Security and Reliability in Concordia," Proceedings of the 31st Hawaii International Conference on Systems Sciences, VII:44-53, January 1998.

[15]http://www.objectspace.com/products/voyager/Index.asp

[16]http://www.generalmagic.com

[17]http://www.cs.bham.ac.uk/~sra/Thesis/index.html