行政院國家科學委員會專題研究計畫成果報告

※※※※※※※※※※※※※※※※※※※※※※※
※　　　　　　　　　　　　　　　　　　　　　※
※　建立一個在網際網路上執行的分散式計算平台　　※
※　　　　　　　　　　　　　　　　　　　　　※
※※※※※※※※※※※※※※※※※※※※※※※

計畫類別：V 個別型計畫　　□整合型計畫

計畫編號：NSC 89 –2213 – E – 009 – 138

執行期間：　89 年 8 月 1 日 至　90 年 7 月 31 日

計畫主持人：王豐堅

共同主持人：

本成果報告包括以下應繳交之附件：

□赴國外出差或研習心得報告一份

□赴大陸地區出差或研習心得報告一份

□出席國際學術會議心得報告及發表之論文各一份

□國際合作研究計畫國外研究報告書一份

執行單位：國立交通大學資訊工程系

中　華　民　國　90 年　12　月　31 日

行政院國家科學委員會專題研究計畫成果報告

計畫編號：NSC 89 –2213 – E – 009 – 138
執行期限：89 年 8 月 1 日至 90 年 7 月 31 日

主持人：王豐堅　國立交通大學資訊工程系
計畫參與人員：陳明豐等研究生

-. 摘 要

代理程式(agent programming)模式在過去幾年來日益發達，在網路上移動的代理程式一般稱為移動型代理執行緒(mobile agent)，它與單點上的智慧型代理執行緒(intelligent agents)及程式導向的代理執行緒(program-oriented agents)有可能成為未來程式撰寫與執行的主流。主要原因是它攜帶部分程式，具主動性質，可在網際網路上自行達成單一或多項目標。本期的工作主要是設計製作一個允許移動行代理執行緒交互溝通的平台。

關鍵字: 代理執行緒, 移動型代理執行緒, 交互溝通的平台。

關鍵字: 企業 Java Bean, 伺服器,交易系統,資料庫,分散式系統.

Abstract

Mobile agent paradigm is getting popular in the past few years. Agent programs communicate with each other or local user to accomplish its goal. Mobile agent(program)s roam in the network, and, there needs a mechanism to maintain the location of mobile agents in inter-network. It is difficult to find a target agent. Besides, Intelligent agent(program)s can learn continuously to modify its intension (behavior) based on the belief and desire. Our work describes a new design and the implementation of communication system for our mobile agent system in order for agents to collaborate with each other more effectively. Our communication system includes both a new name service system and a new message delivery based on standard naming/communication mechanisms. The name service system works with location tracking, while the message delivery system handles message deliveries.

Keywords: agent programs, mobile agents, communication service system, naming

1. Introduction

Recently, browsing information on World Wide Web is the most popular global service. Internet has more and more software applications to generate and transfer huge amounts of information. Mobile agent is a new paradigm proposed to reduce the programming efforts and network traffic. Mobile agents are programs, which can migrate among machines in a heterogeneous network; they can communicate between and cooperate for each other no matter whether they are inside a machine or not. Due to migration, an agent can access resource at another host like doing those locally in order to reduce the network bandwidth. Mobile agent models changes programming from the rigid client-server model to a more flexible peer-to-peer model, where programs communicate as peers in the same site, depending on their current needs.

Mobility and communication are important factors in cooperation of agents. The platform of mobile agents needs an efficient and reliable communication system. Such a communication system is usually based on a good procedure for locating agents roaming the network and a reliable message delivery mechanism. In general, both mechanisms are complicated because agents can move anytime. For example, the target agent might move to another host after the sending agent gets its location datum. Since the location datum is not correct now and the delivery mechanism needs additional efforts to deliver the message.

In the study, we designed a new communication system in our agent system platform, where the message delivery system is divided into two parts: a location system associated with the naming scheme and a message delivery system.

2. Related Works

In order to let code be executed on heterogeneous machines, there are many languages used for implementing agent system before Java was shown in the world [1]. For example, Agent Tcl [3] and Ara (Agent for remote access) [4] are based on the Tool Command Language, and Telescript [5] is from General Magic [10] Inc. JAVA is a better choice of language for agent systems, because it has some features not found in other language for mobile agent systems. With object serialization in Java, objects can be easily "serialized" and sent over the network or written to disk for persistent storage.

There are many commercial Java-based agent systems: IBM Aglets [6], Objectspace voyager3.1 [7], Mitsubishi's Concordia [8], …, etc. Existing location tracking system can be divided into two parts for discussion: a naming scheme and a location tracking system.

Existing naming schemes can be classified as in (table.1).

Scheme	Example	Transparency	Independence	Used by
Agent-site+ name	Current.host/MyAgent	No	No	Voyager, AgentTcl
Agent-site+id	Current.host/9999	No	No	Aglets, concordia
Name+Home-site	Myagent@Home.site	No	Yes	Our system

Table 1. Naming Scheme

A naming scheme is location transparent [12] if the agent name does not contain any site-specific information. For example, a name comprising the site to which the agent belongs plus an agent identifier (e.g. dssl.csie.nctu.edu.tw/MyAgent) is not location-transparent. On the other hand, an agent named according to its functions may be location transparent. Example is the name MySearchAgent. A naming scheme is location independent if an agent name can be used to reach the target agent, no matter where it is, and the name is not changed after being created. Note that "location-transparent" does not mean that an agent name cannot contain location-specific information.

Location-dependent naming schemes may allow simpler implementation of name service systems than location-independent ones. Platforms like Agent Tcl, Aglets and Tacoma use location dependent technique to name agents. In these systems a mobile agent is named based on hostname or port number. The name service is resolved using DNS. When an agent migrates, its name would change. For example, in Aglets, if the agent named ssss.csie.nctu.edu.tw/hello moves to another location dddd.csie.nctu.edu.tw, its name will change to dddd.csie.nctu.edu.tw/hello. However, the location-dependent naming schemes make the implementation of agent tracking cumbersome. On the other hand, a location-independent naming scheme requires a name service system to map the symbolic name to the agent's current location.

A good location tracking system may help deliver messages quickly and reliably in mobile agent systems. There are several solutions offered today. A simpler solution is to use a unique name server to keep track of all agents [6][15] instead. Cuurent Solutions have their drawbacks correspondingly. [6] [8] [15] [16]

3. Our Platform for Agents

The current version of our mobile agent system contains four main components: Agent, Place, Security Manager, and Agent System Management Server.

An agent in our platform, a basic component in the application system, is a JAVA thread in a place at a time. An agent consists of code and variable states, and auxiliary data as in Table 1. The states describe the agent's requirements. The dynamic state of an agent includes both site transfer information and state–execution condition. For a move, the agent transfers itself to a new node and continues its execution at some state. If there is a need to restart an agent at a particular execution point, it is done similar to the methodology in Aglets. All agent classes could inherit the basic Agent class, which provides elementary methods or functions for a mobile agent. The attribute agentname for an agent is a unique name controlled by name server that identifies a particular agent object.

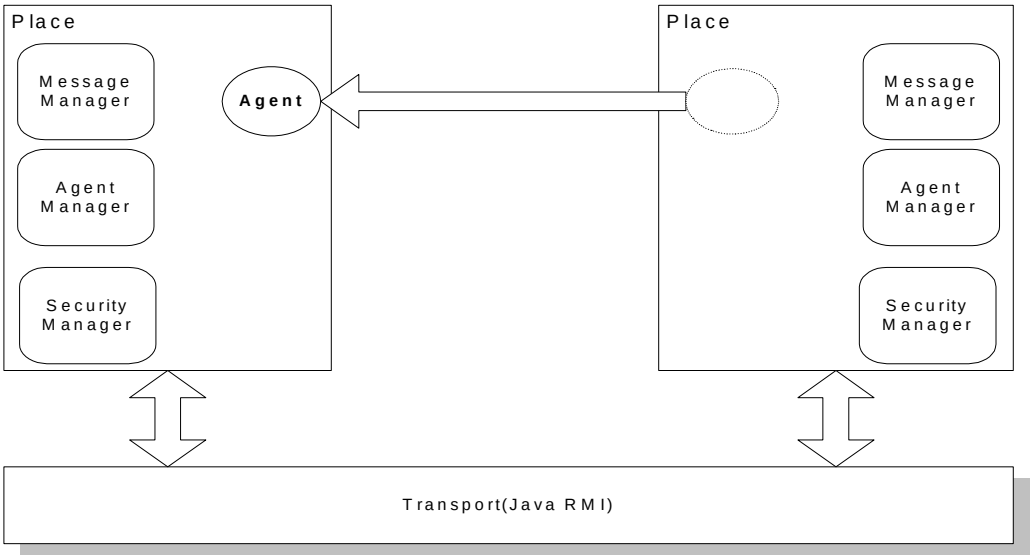

Figure 1 Our platform of mobile agents

State	Static Attribute Dynamic state Agent message queue
Code	Internal code Agent methods
Auxiliary Data	host-dependent executables

Table 2 overview of agent object

There are several other mobility patterns discussed in the literature, for example, itinerary, Start-shaped, and Branching [11] etc. Our system contains two basic patterns, sequential and parallel migrations, to derive others [11].

- Sequential Migration:

Here an **itinerary** object maintains a list of destinations, including the next the agent will move to, defines a routing scheme, and handles special cases such as what to do if a destination place does not exist. Objectifying the itinerary allows programmer to save and reuse it later. It is similar to use bookmarks.

- Parallel Migration:

The **Master-Slave** Pattern [2] allows an agent to spawn several *slave* agents, which move to the places of different locations for execution in parallel. A slave agent is delegated a task by the master agent. After finishing its task, the slave returns to the place created to report the results to the master agent.

The detailed mechanism of implementing a mobile agent in JAVA is skipped here.

For Internet programming, a **region** [9] is newly defined for a set of places and agents of the same agent authority. Each region has one log server to record all behaviors of agents inside, and several register servers (called Place Agent Registry, PAR), each covers several places, to provide information about service agents as in Aglets, voyager, … etc. The log server stores the behaviors in the database using JDBC. Each log entry of the database contains the place location, date, time, and the action of an agent. A PAR is queried by the agents in the same region for finding the agents for communication. A region also contains a Region Name Registry Server (RNR) as a backup server for its PARs.The relationships among places, PAR, Logserver, Region, and RNR are shown in Figures 3 and 4.

4 The Design of a Modified Communication System for Agents

As discussed in Section 2, a name service system provides naming scheme and location tracking services. The design of our modified communication system contains three parts: naming scheme, location tracking and message delivery. The naming scheme guarantees that an agent has a unique name. The major service of location

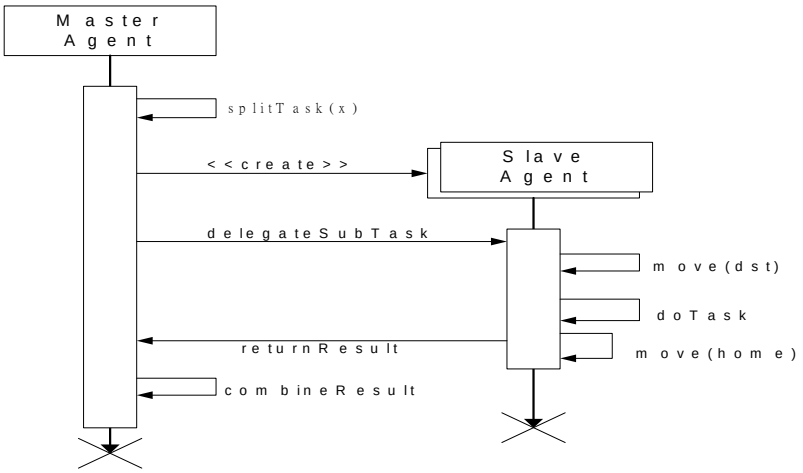

Figure 2 Master-Slave Interactive Diagram

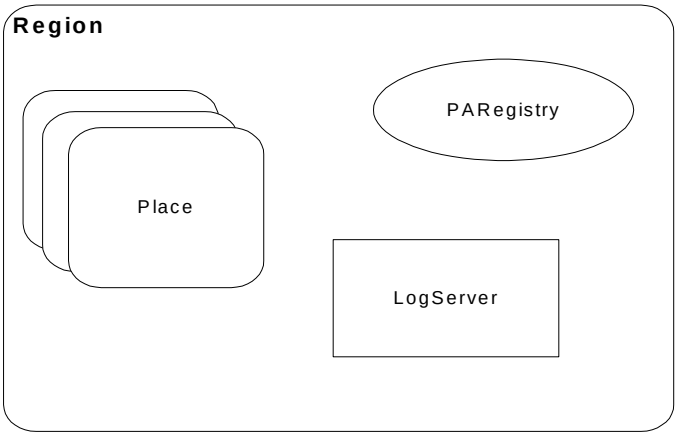

Figure 3 Agent server system view

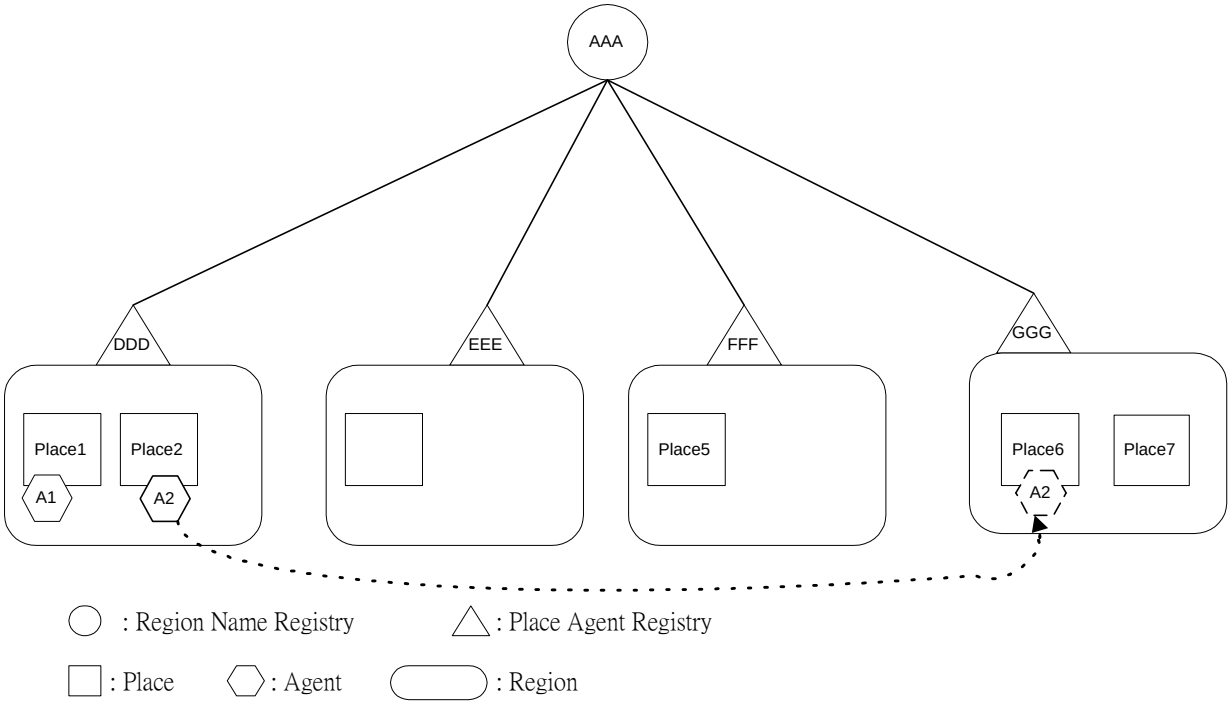

Figure 4 PAR and RNR relationship

tracking is to find out where an agent is currently. The message delivery mechanism then passes the message to the target agent.

4.1 Our Naming scheme

Our naming scheme contains three characteristics:

1. It allows users to name objects easily.
2. It maps the user-defined name of an agent to the birthplace so that users can find the location of an agent correctly based on the agent name.
3. The name of an agent is not changed after the agent is born.

In our naming scheme, we define the naming format as the following: Localname@PlaceName.regionName:port
For example, myHello@Place1.AAA:9999
PlaceName.regionName is the name of an agent's birthplace and Localname is the name of an agent defined by the generator or programmer. A Localname cannot be used for naming if the name is defined for another agent born at the same place and still alive. The port stands for the entrance of every place at the machine. The string expressing the characteristics can be used for agent's Localname for better readability.

To avoid name duplication for agents, the system is defined two parts. First, DNS (Domain Name System) [14] guarantees that the name (place name.region name) for each place has a unique name. In an intranet environment, a private IP could not have a public domain name at DNS. A RNR server is used to guarantee the uniquesness of region name. Second, each agent has one name only. The birthplace PAR of an agent guarantees the use of Localname when an agent is generated.

4.2 The Mechanism to Track an Agent

The management of agents' location information in our system could be described in three separate phases: *registration (deregistration)*, *migration* and *getLocation*. When an agent is generated, the agent is registered with the agentname in the name service system automatically. When the agent terminates, it reports to the name service system to clean the registration. When an agent moves, the agent's information in the name service system would be updated for request.

As discussed in Section 3, a PAR in a region stores and manages the location information of agents created in the region. Inside a PAR, an entry of the agent table contains five attributes: *agentname, valid bit, current location, message queue, logic clock.* Attribute *agentname* is the main key in a PAR; each *agentname* is unique in a PAR. Attribute *valid bit* is used to indicate whether the agent is moving out now. If *valid bit* is false, the agent is moving and *current location* is incorrect. The PAR would not reply the location for any agent entry whose *valid bit* is false. Attribute *Current location* stores the place where the agent stays currently. The *current location* is (correct and) replied, if *valid bit* is true. It would be updated once the agent moves successfully. Attribute *Message queue* stores the agent's input messages that arrived at the *current location* after the agent left. Attribute *Logic clock* is an integer value used to sequence input messages of an agent. The logic clock variable of an agent entry is increased one by the PAR, each time the clock is queried. The default value is 0.

When an agent is created, the home PAR is informed the information of agentname and current location for registration in the region. The home PAR then checks whether the agentname exists in this region. If no, the PAR adds an entry to store the agentname and location for registration. Otherwise, the registration fails and the system or user gives another name to register again. In case the registration fails too many times, the home PAR would offer an unregistered agentname for successful registration. Notice that the agent name cannot change in an agent's life. The home PAR deletes an agent entry and asks the RNR server to delete the entry when it receives the termination request. In case there is a message in the entry, it sends a "MessageExisting" message back to the agent additionally,

A successful move for an agent can be divided into five steps in Figure 8. Firstly, an agent notifies its home PAR that it wants to move out. The home PAR changes the *valid bit* in the agent entry to false. This is a synchronous step. Secondly, the agent puts the address of destination place into the cache of current place. Thirdly, the code of its code and state is moved to the destination place. Fourthly, if the agent is accepted

(activated) by the destination place, i.e., it first asks the home PAR to update the *current location*, and change the *valid bit* to true in the agent entry. Then, it asks the home PAR to move out the message(s) stored in the *message queue* to its SYN otherwise.

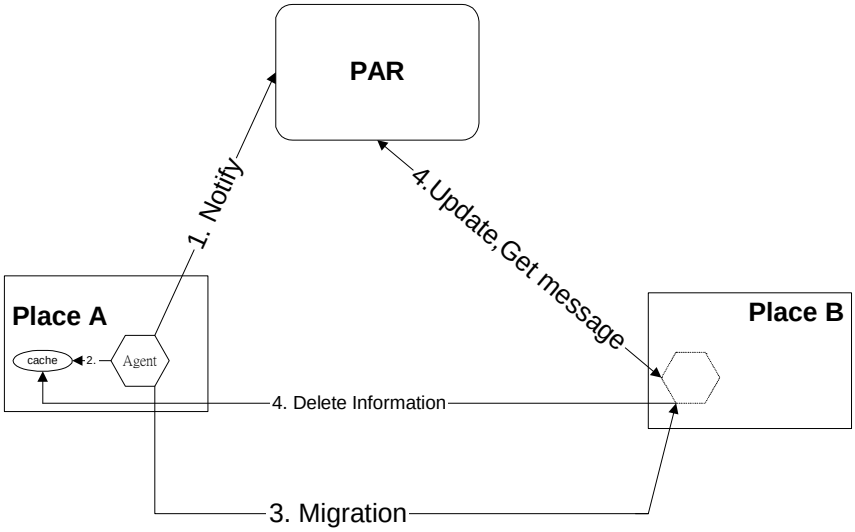

Figure 5 Successful agent migration phase

The location information request in our system is simple. Our naming scheme allows an agent to get the name of the home PAR of another agent (by the latter's agentname). Thus, an agent can get the current location of another one by asking the latter's home PAR. If the *valid bit* is true, the reply contains the location address. Otherwise, the reply is " AgentMigrating" by default. In case the agent does not exist, the reply is "The agent does not exist".

4.3 Message delivery mechanism

In an agent system, a receiving agent might move during the processing of message delivery. It makes message delivering more complicated than conventional distributed system. Based on our tracking system, the activities of message delivery are divided into synchronous, asynchronous and one-way, where each message is considered from sending agent, receiving agent, and delivery process correspondingly:

- **Synchronous messages**

For a receiving agent,

If there is no message, it keeps waiting until such a message arrives. Here, an agent is like a server in the client-server model. If there is a message in its SYN message queue, it processes the message. The queue guarantees first-in-first-use principle.

For a sending agent, when the message delivering completes, the delivering can either be done or fails (as described in step 1.b). In other word, the agent gets an "O.K." or "fail" message and goes to next step.

The delivery of a synchronous message can be described as follows:

1) The sending agent requests the current location of the target agent from the target agent's home PAR.

a) If the target agent does not exist or has been killed, the PAR returns a failure message.

b) If the target agent is at the migration stage, the sending agent would get a "AgentMigrating" message. Here, the sending agent will start a new request in a fixed time interval continuously until the sending agent gets the location or the time interval present for failure.

2) If the sending agent got the address, the message is sent out to the target agent directly. When the message arrives the place of the target agent,

a) If the target agent moved already, the message is sent to the home PAR of the agent.

b) Otherwise, the message would be

sent to the target agent.

(1) The message is the one the target agent is waiting for. The agent is then activated to start its execution.

(2) Otherwise, the message is put at the end of the SYN message queue of the target agent.

- **Asynchronous Messages**

For a receiving agent,

If there is a message in its ASYN message queue, it processes the message. The step then returns an 'O.K.' message. Otherwise, the step returns a 'no message'. No matter what the step returns, the agent goes to next step after the return.

For a sending agent,

Message sending can be treated Step 1.a and 1.b in the delivery process.

The delivery of an asynchronous message can be described as follows:

1. The sending agent requests the current location of the target agent from the target agent's home PAR.
   (a) If the target agent does not exist or has been killed, the PAR returns a failure message.
   (b) If the target agent is at the migration stage, the sending agent would get a "AgentMigrating" message. The sending agent now starts requesting a location until the PAR returns a failure message (in a fixed time interval).
2. When arriving message is the place of the target agent,
   (a) If the target agent exists in the current place, the message is sent to its ASYN message queue.
   (b) In case the agent moved and the information of its new place exists in the current place, the message is forwarded to the new place. If the information does not exist, the current place can be treated as the original sending agent and goes to step 1. Note that the current place is also in charge of forwarding the failure message back to the original sending agent.

- **One-Way Messages**

A one-way message is asynchronous and does not block the current execution of the sending agent. The sending agent will not retain a handle for this message, and the receiving agent will never have to reply to the sending agent.

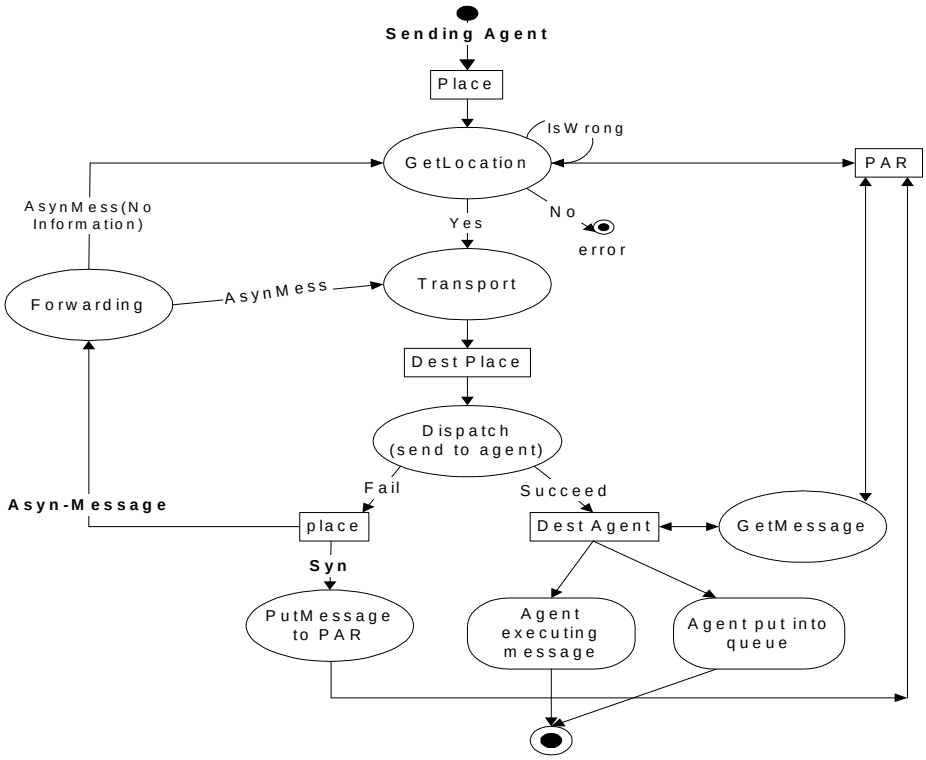

Figure 6 Message Delivery Flowchart

In distributed systems, network transporting has some special phenomena. Theoretically, the clock cannot be synchronized for all places in a distributed system, in other words, it is impossible to find out the order of all messages. Practically, a message sent earlier than another one may not arrive at the destination earlier because of network traffic. On the other hand, a server may have more than one entry points. The message arriving earlier may not be guaranteed for service earlier. Logic clock mechanisms provides $O(n^2)$ solutions to causal order.[13]
To solve the problem, our system provides a simple mechanism to help decide the execution order of arriving synchronous messages only. A PAR's message queue is used to store the message(s) which arrives at the destination place after the target agent left out in Figure 7. Each agent is defined a ***logic clock*** for the synchronous message in its home PAR. For the delivery of synchronous messages, when a sending agent asks the target agent's home PAR the latter's current location for a synchronous message, the agent also gets a timestamp from the PAR. On the other hand, the PAR increases logic clock variable by one at the same time.
In message handling aspect, an agent is defined a synchronous message queue and is also associated with a timestamp queue where each element indicates whether the corresponding message is "executed", "arrived but not executed", or "not arrived yet". The queues are implemented with arrays and two indices. The first index points to the oldest message that is not arrived and served. The second index points to the biggest timestamp to show the latest message being received. When processing a message, the agent searches the timestamp queue to find the oldest message for the corresponding entry. The search starts from the first index and completes at the second index. If there is no message, the agent waits in a fixed time interval, gets the message(s) from its home PAR, and then starts next searching. If such a message exists, the agent modifies the corresponding element in timestamp queue as "executed" and processes the message. If the first index equals second, no message needs to be served and the agent repeats above (wait, get, start) activities till a candidate message arrives.
When the agent receives the messages, it also lets the second index be modified to get the newest timestamp and updates the elements added in timestamp queue if a timestamp of the message is bigger than the second index. Here, if an element whose message exists already, the element is not changed. If an element whose message is received now, the element is set "arrived but no executed". If an element whose message is not in, the element is set "not arrived".
Our method has some defects. For example, an agent might crash when it waits for the candidate message that is lost. An agent asking earlier than another one may not get a smaller timestamp than another one because of network traffic. In case the target agent doesn't move, the sending agents get the same location and timestamps from the home PAR. It wastes the PAR's time execution and network traffic.

5. Conclusion and future work

The report presents the design of a communication platform for mobile agents and a system has been implemented in our lab. It supports the agent location tracking and offers a more reliable and efficient mechanism for delivering messages to mobile agents than previous versions [16][17].

There is a defect for synchronous message passing in the system: the message queue in a PAR. When a message reaches the destination place, the target agent may leave already, and thus the message is sent to the agent's home PAR. In case that the target agent does not move, it will not ask the home PAR and thus the message will stay in the home PAR too. This may not be efficient because the target agent might not move for a long.

One solution to this problem is to design a mechanism that allows a PAR to check whether its registered agent stays at the same location for a fixed time. The PAR sends the messages to the agent if the answer is "the same place". Another solution to this problem is to let the agent gets the message from the home PAR when it finds out some message not received for handing. Both mechanisms introduce additional problems for the system still and no discuss further.

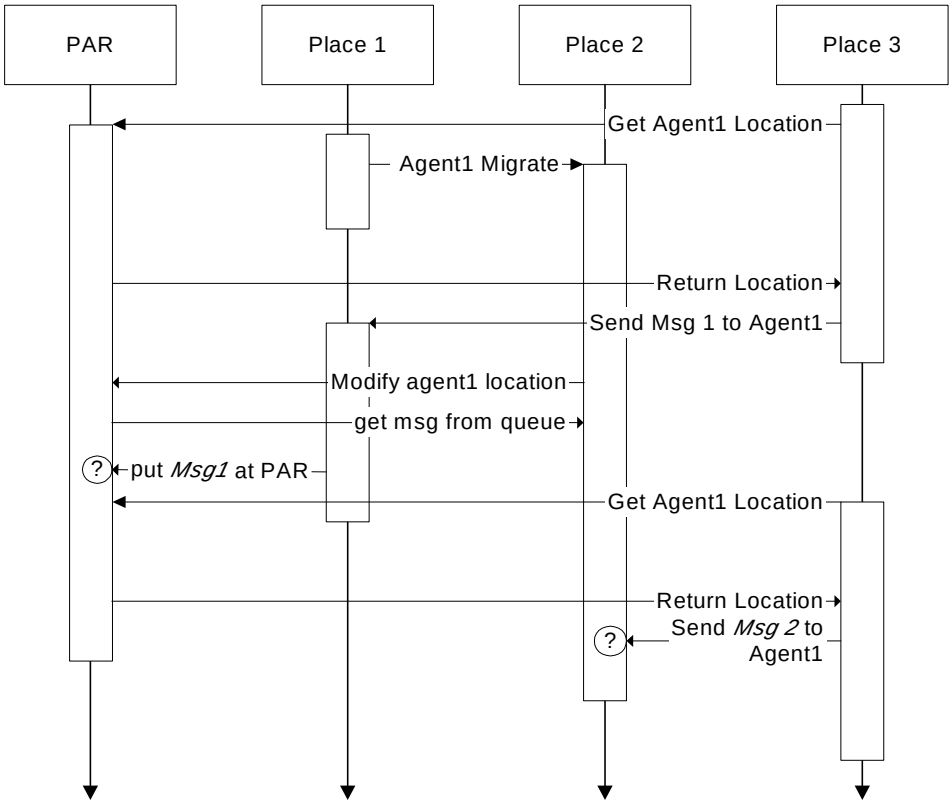

Figure 7 Situation of Message Arriving

Reference

[1]"Software Agents: A review", Shaw Green, Leon Hurt etc.

[2] Programming and Deploying Java Mobile Agents with Aglets, Danny B. Lange and Mitsuru Oshima

[3] Agent Tcl was developed by Robert S. Gray and colleagues at the Dartmouth College Computer Science Department .

[4] Ara is a project within the Distributed Systems Group in the Computer Science Department of the University of Kaiserslautern, Germany. http://www.uni-kl.de/AG-Nehmer/Projekte/Ara/index_e.html

[5] James E. White; Telescript technology: The foundation for the electronic market place; General Magic White Paper.

[6] IBM Aglets, http://www.trl.ibm.co.jp/aglets

[7] Voyager 3.1, http://www.objectspace.com

[8] Concordia, http://www.meitca.com/HSL/Projects/Concordia

[9] MASIF-The OMG Mobile Agent System Interoperability Facility; Mobile Agents–Second International Workshop, MA'98 (Stuttgart, Germany, September 1998); Published as Kurt Rothermel and Fritz Hohl, editors, Lecture Notes in Computer Science, 1477. Springer, September 1998.

[10]Mobile Agents White Paper, General Magic, http://www.genmagic.com/technology/techwhitepaper.html/

[11] Agent system Development Method Based on Agent Pattern; Yasuyuki Tahara, Akihiko Ohsuga and Shinichi Honiden; 21st International Conference on Software Enigeering, 16-22 May 1999.

[12] Distributed Systems: concepts and Design; George Coulouris, Jean Dollimore, and Tim Kindberg; second edition 1994

[13] Distributed Operation Systems & Algorithms; Randy Chow and Theodore Johnson at university of Florida; publisher Addison Wesley 1997

[14] HOSTNAME Server; Tech. Report RFC 953; ftp://nic.ddn.mil/user/pub/RFC

[15] Mobile Objects and Agents (MOA); Dejan S., William LaForge and Deepika Chauhan; The Open Group Research Institute