第三屆即時與媒體系統研討會
Third Workshop on
Real-Time and Media Systems (RAMS' 97)
台北市南港中央研究院
1997 年 7 月 30 - 31 日

Issues and Design of Server for FlyingCloud - A Mobile Agent Service Network

Yao-Nan Lien, Yenlin Yin, and Tony Chan

Department of Computer Science
National Chengchi University
Taipei, Taiwan, R.O.C.
lien@cherry.cs.nccu.edu.tw, (02) 9393091-2275, Fax:(02)2341494

Abstract

To make information ubiquitously available to people in the world requires not only Information Superhighway, but also a non-traditional computing paradigm, such as the intelligent messaging, to overcome the intermittent connection problem inherited in a mobile environment. This paper describes the design of the server in the FlyingCloud mobile agent service network currently under development in the National Chengchi University. The main objective of this prototype is to simulate a real operational mobile agent service network, to analyze the network behavior and to exercise our solutions. The system is developed based on the previously proposed open architecture and hybrid mobility management infrastructure [7,8,9].

We designed a high level script language and a **script rewriting** *technology to carry out the most challenging task, agent migration, successfully.*

Keywords: mobile computing, mobile agent.

1. Introduction

1.1 Mobile Agent

Due to the advance of computer and communication technologies as well as the promotion of National Information Infrastructure (NII), the progress of *mobile computing* is accelerating to a revolutionary speed, making the dream of ubiquitous information service a reality [1,7,14]. The goal of a ubiquitous information service network is to provide information to users anytime anywhere. To accomplish this goal, the service network must be supported by some ubiquitously available communication networks and be able to conveniently access to various information resources. Currently, wireless communication networks such as AMPS or GSM cellular networks are able to support the ubiquitous communication requirement [12,16]. As for the convenient information services, distributed computing technology seems to be an ideal computing paradigm. In a distributed system, all servers in a network are integrated into a single logical server so that clients, that can be programs or users, can access the network resources transparently by interacting with a single server. Unfortunately, applying distributed computing technology in such a scale and heterogeneity may have to take a much longer time to accomplish in real world.

Therefore, clients will have to access network resources in a prescriptive fashion by interacting with individual servers probe by probe to accomplish a complicated task. However, in most mobile computing environments, the nature of communications is intermittent and the battery energy is limited. Thus, it is very difficult to accomplish a complicated task which requires its clients to interact intensively with multiple servers. A non-traditional computing paradigm, the *intelligent messaging*, that allows clients to interact with multiple servers in a dynamic fashion, has been brought up to cope with this problem [2,3,4,5,6,11,13,15].

Simply speaking, an *intelligent message* is an electronic message that carries a computer program, either procedural or declarative, which can be executed by the receiving servers on behalf of the originating client. The program in the message can also instruct a receiving server to forward automatically the message itself to another server, on which the program is executed continuously in a pipeline fashion. Such a message is also known as an *intelligent agent* in other fields [4]. For simplicity, it is called an *agent* in the rest of this paper. Good examples can be found in [7,8].

1.2 Mobile Agent Service Networks

To make a service network commercially viable, it is essential to have a high quality and cost effective operation, administration, and maintenance system (OA&M) in place to guarantee a certain QoS (Quality of Service). However, it will be impractical to acquire huge resources to modify existing telecommunication networks to support many prospective information services. All

1. This work is supported by NSC Grant (86-2213-E-004-001).

infrastructures and solutions must not require any change to the existing telecommunication network. To satisfy this constraint, we employed the open service network architecture proposed in [8], which separates service networks from transport networks to maintain the required physical network independency. It also allows services of any scale and any quality to be introduced into the network easily. Service providers can choose whatever operation model and QoS level based on the resources available to them. Readers are referred to [8,9] for details.

1.2.1 Hybrid OA&M Supporting Infrastructure

Centralized OA&M support is easier to achieve higher QoS. However, it suffers from higher operation cost and long deployment time duration. On the other hand, distributed support is usually more flexible in introducing new services and has a much lower operation cost, but it suffers from lower QoS. In [9], Lien proposed a hybrid OA&M supporting structure that can take advantages of both approaches. In that hybrid approach, all OA&M functionalities are classified into two categories: distributable and non-distributable. Distributable functionalities can be supported by any server in the network or even client users' own resources. Non-distributable functionalities must be supported by designated servers. Critical functions that are more appropriate to be managed centrally, such as security and billing, are classified as non-distributable and must be managed centrally.

1.2.2 Physical OA&M Facilities

In FlyingCloud, we assumed there is a central facility, called *Network Mangement Center (NMC)*, supporting all non-distributable management functions as well as distributable functions if it is needed. Typical OA&M functions are client and server registration, authentication, name server, coordination, billing, or user specific services.

We also assumed that most of the mobile computing users in the future will be able to access to the Internet from either their offices or homes. These facilities are called *Home Base Nodes* (HBNs). To further reduce operation cost, Lien proposed to use these personal facilities to help managing service networks [9]. A client user of the service network can choose to use the facilities provided by the service providers or his/her own HBN to manage the client and agent mobilities.

1.2.3 Logical OA&M Facilities

One important logical facility is the *status holder* of an agent, which is a place to store the status of an agent so that the client user or other authorized entities can access this information easily. It can be any node such as user's HBN or the NMC itself. A system may require each agent report its status to its status holder on the designated events such as arrive-a-node, suspended, frozen, etc.. A user can choose whatever the events he/she is interested in when he/she submits an agent. An alternative status holder might be needed when the availability of the original status holder is a concern. A user can even request an agent **not** to report its status to save communication cost. These all depend on implementation details.

1.2.4 Relationship Between Logical and Physical Entities

Physical entities such as NMC and HBNs are fixed with respect to the network address. Logical entities, such as status holder and computing support, can be dynamically assigned to some physical entities. An example is depicted in Figure 1.

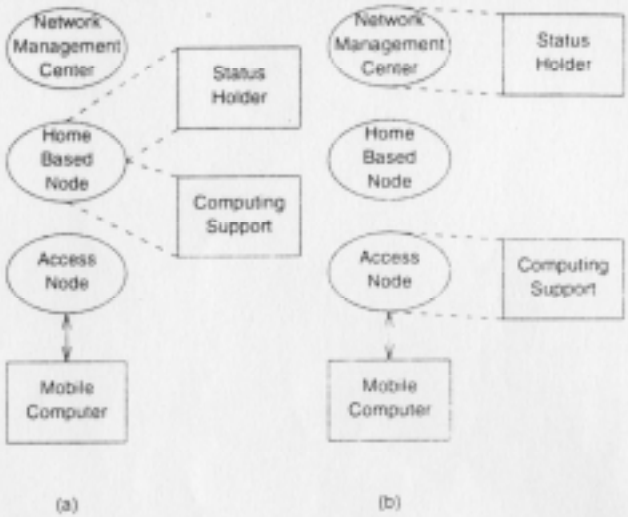

Figure 1. The designation of logical entities to physical entities, (a) both status holder and computing support are designated to the HBN, (b) status holder is designated to NMC and computing support is designated to the Access Node.

2. Current Implementation

The intelligent messaging paradigm is such a newly emerging area that we do not have sufficient knowledge about its behavior under all possible operation conditions, especially in a real operational system. Thus, there is a need to create a simulated environment to facilitate further study in various critical issues raised in [7,8,9]. A project is currently under development in NCCU to develop an agent mobility management network prototype, the *FlyingCloud* [10]. The platform will consist of a set of servers capable of intelligent messaging support, a script language, and an agent management system.

2.1 Transport Mechanism

To comply with our open architecture and to simplify the implementation, email system (MIME protocol) over the

Internet is adopted in the initial design. Servers, clients, and agents all communicate with each other through email. Every agent is wrapped within an email message. (However, the system will be designed with necessary flexibility that the underlying transport network can be easily replaced if a better mechanism is available.) In the immediate future, we will investigate the http protocol, which is widely available on the WWW information network over the Internet. The efficiency and security will be greatly enhanced.

2.2 Design Philosophy

In addition to the architecture principles we proposed in [8], the system was designed under the following guidelines:

1. In order to maintain the required QoS, system reliability is the main objective with the highest priority. (As a consequence, most components in our system will be as simple as possible.)
2. The system was designed with flexibility so that each of its components can be easily replaced.
3. Since designing a complete and sound system is by no means a trivial job, the system was designed evolutionally. Only the simplest platform was implemented initially and will be gradually evolved into a more complete system. In the initial stage, we decided to focus on the functionality of the system instead of a complete system. In other words, we designed and implemented a working prototype to facilitate further researches on other issues such as security, management, transaction control, etc.

2.3 Design Considerations

Based on the design philosophy mentioned above, our first design consideration is the selection of script language. We decided to design a high level language with limited capability instead of choosing a sound and complete low level language such as Java. Currently, there are a few similar researches in this area that choose to use Java as the basic programming language because Java is intended to be platform independent and network mobile. However, our current objective is not to design a sound and complete language. Instead, we only need a prototype language for us to write various testing scripts which may provide some guidances to the researches in a broader scope. A high level language would allow us produce testing scripts faster. Furthermore, the mobility of Java is not clear yet (i.e. to move a Java program from server to server.) Further researches are needed to make Java really mobile in a network. Nevertheless, the prototype language can be evolved to a more complete language in the future.

The second consideration is the choice of implementation language. We chose to use "perl" programming language for its portability and powerfulness. It is available on various types of Unix operating systems such as Solaris, HP UX, Linus, FreeBSD, etc. Theoretically, it can also be available on Windows NT.

2.4 Agent

A FlyingCloud agent is designed to be *self-contained* that the entire context is encapsulated in the script itself. When it visits a server, the server will execute the script until the script is terminated or it demands a move. When the agent is moving to another server, the current server will wrap the entire context into the script itself and forward it to the next server. The relationship between the server and the agent is then terminated. The server may record the external status of the script execution such as arrival and departure times, the termination status, and the next server it visits, etc.. However, it does not keep any internal context of that agent.

This self-contained property is quite different from the *Remote Procedure Call* (RPC) approach, where the server that issues an RPC to another server will keep the context of that agent until the agent finishes its execution at the remote server and returns to the originating server. If an agent visits a sequence of servers, the servers it had visited will all remain active until the agent terminates. Much resource will be tied up by the agent. Further, the concept of mobile agents will be violated, which will have a significant implication to the mobile agent paradigm. For example, the semantic of recovery will be quite different. (Nevertheless, it is yet to be studied which way will be better.)

This self-contained property also has another significant implication to the language design, which will be discussed in latter sections.

2.4.1 Agent Status

The status of an agent is in one of the following six different states:

- *running* - an agent is being executed by a server.
- *spinning* - an agent is active in a server, but is waiting for some local resource.
- *hopping* - an agent is being forwarded to another server.
- *terminated* - an agent is terminated.
- *suspended* - an agent is suspended in the middle of or before an execution by an authority external to that agent.
- *frozen* - an agent in a server is not being executed, but is waiting to be forwarded to another server.

The difference between "spinning" and "suspended" is that a spinning agent can resume its execution by itself without any external permission, while a suspended agent can't. (Even when a spinning agent is waiting for something, it can always resume itself if it decides not to wait.) The details can be found in [8].

2.5 Agent Control

The management system must be able to control the execution of each agent. The control functions available in FlyingCloud are termination, suspension, resumption, and freeze. The state transition diagram under the external control events and two internal events, hopto and arrive_a_node, are shown in Figure 2.

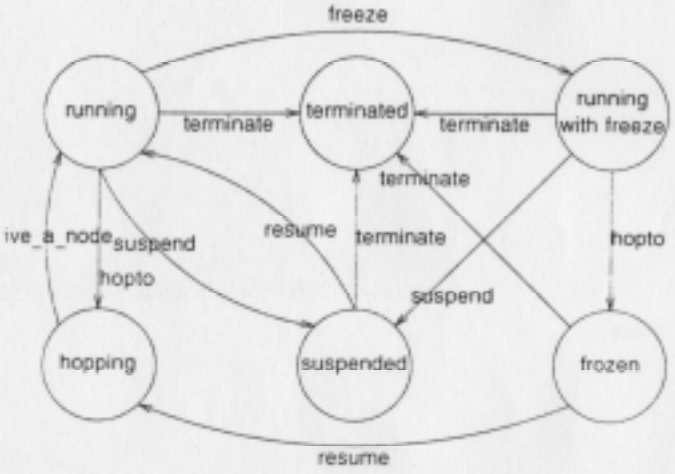

Figure 2. The control of agents.

2.6 Agent Script Language

Designing a sound and complete script language can never be a trivial task, especially for such a newly emerging paradigm. For the reasons mentioned in Section 2.2, we designed a high level script language for our platform.

The language we designed in FlyingCloud offers only *string type* and a derived type *list* with some simple forms of list operation. The control flow is much like an assembly language that has *sequential* and *jump* capabilities. It also has *if* construct and *loop*, but does not have *function* yet.

The following is an example script which sequentially searches a target agent (ID AD123) within a list of servers and terminates it.

```
 1  name: shanon
 2  passwd: xxxxx
 3  LIST={apple,kiwi,lemon,banana,orange}
 4  JUMP(6)
 5  hopto STATION
 6  STATUS=`exit AD123`
 7  IF(!STATUS) {
 8     NEW_LIST=cuttail(2,LIST);
 9     STATION=first(NEW_LIST);
10     JUMP(5);
11       }
12   terminate AD123
```

The section between Line 1 and 2 is the basic authentication information. The password will be encrypted in the real service network. The real script is in the section between Line 3 and 12. String variables are allowed.

One of the greatest issues in designing such a language is to fulfill the self-contained requirement. Initially the language did not allow *split-context* execution. For example, the following program construct **was not** allowed:

```
for i from 1 to 10 do {
  hopto next server
  compute
  }
```

In this example, the server that executes the "for" statement must maintain the script context including a counter to count how many times it executes the "do block". Within each do block, the script will visit another server to execute a computing job. Therefore, the context of the script may actively exist in two servers simultaneously. This problem has been solved by the *script rewriting environment migration technology*, which will be described in next section. In other words, the context of a control variable is wrapped up by the server and migrated to the next server when it should move during a loop context.

To offer more flexibility and extensibility to the language, we are considering to make the language a complete macro language by adding macro definition capability.

3. Server Design

In this section, we will discuss the design of mobile agent server in the FlyingCloud system.

3.1 Design Issues

Major issues considered in the current design of FlyingCloud servers are as follows:

1. A platform independent virtual machine that can execute the incoming agent.
2. The ability of moving an agent with the environment to another server.

   Actually, the environment migration is the greatest challenge to the design of FlyingCloud servers. In a homogeneous environment, a process can be migrated from a server to another by copying the process image and all associated stack, process status, etc., to another machine and restarting the process over there. Unfortunately, this method cannot be applied to a heterogeneous environment.
3. A mechanism to facilitate the communications between agents.

3.2 Components of Servers

Each agent server consists of four major components:

1. Dispatcher,
2. Agent Control Center (ACC),
3. Virtual Machine (VM), and
4. Client Server Center (CSC).

3.2.1 Dispatcher

When the mail daemon of a FlyingCloud server receives an agent, it will invoke the Dispatcher to process the incoming agent. The Dispatcher first carries out an authentication process to verify the agent. If this agent is sent directly by a valid client, the Dispatcher will assign a network-wise unique identification, called *agent ID*, to the agent. This agent ID will be associated with the agent throughout its life time. On the other hand, if the incoming agent is sent by a server, its agent ID and the associated security information will be authenticated. Finally, the Dispatcher will forward the agent to the Agent Control Center by sending a notification message to it.

There is also a special case that the Dispatcher may receive a *homecoming agent* that has finished its job and returns to its home. The Dispatcher will deliver the result to the client immediately or store the result in an appropriate place waiting for the original client to retrieve. The details will be discussed in the latter sections.

3.2.2 Agent Control Center (ACC)

The Agent Control Center (ACC) in a server is really the headquarter of a FlyingCloud server. ACC is a daemon process always running in the background. After ACC receives a new agent from the Dispatcher, it will fork a child process, called *Virtual Machine*, to execute the agent. As implied by its name, this child process acts as a virtual machine for the agent. The entire script execution environment is provided by this Virtual Machine.

ACC also provides external communication channels to its agents. ACC can manage and control agents through these communication channels. Agents can also communicate with each other through these channels. When an agent needs to communicate to ACC or another agent, it sends a message to ACC. ACC will process the message or forward it to the target agent. One important management function in the FlyingCloud system is the *look-ahead control* feature that allows a client or manager to place a control message in advanced on a server where the target agent hasn't reached the server yet. This communication mechanism is important to support such a feature. A client or manager can leave a message in the message queue of a server. When the target agent reaches the server, the VM that processes the target agent will receive this message and give it to the agent.

3.2.3 Virtual Machine (VM)

A VM executes the script of an agent sequentially like a conventional script language. It will check the message queue before the execution of each line. Any request in the message queue will be executed first. A script may contains several different types of instructions: passive instructions, loop, and active instructions. Passive instructions will only affect the internal state of the agent itself, while active instructions, such as "freeze" and "suspend", may affect other agents. For example, the following instruction

```
Suspend A123;
```

will suspend the execution of Agent "A123". The VM of this agent (the one that executes this instruction) will send a "suspend" message to ACC. ACC will forward the message to the message queue of the VM that is executing Agent A123.

3.2.3.1 Instruction Atomicity

Every instruction except a loop instruction and "hopto" must be executed atomically. In other words, it must be executed in the current server. However, different instructions within a loop can be executed in different servers. Indeed, allowing instructions within a loop to be executed in different servers is a great challenge to us. All inherent control variables must be migrated together with the agent. This is not trivial.

3.2.3.2 Agent Migration By Script Rewriting

The instruction "hopto" is used to migrate an agent from the current server to another. For instance, the following instruction

```
Hopto NEXT_STATION;
```

will request the current server to migrate the agent itself to NEXT_STATION, which is the name of another server. When VM executes this instruction, it will first check whether the supervised agent is in "frozen" state or not. If the agent is in "frozen" state, it will postpone the migration until receiving a "resume" or "terminate" message (from another agent.) Otherwise, it will proceed with the requested migration.

When proceeding with a migration, VM will wrap all the internal data belonging to this supervised agent and convert the script into a new script that represents the current state of the agent. The current state of the agent can be automatically restalled by executing the script in another server. The modified agent will then be sent to the next server by email. The agent migration is greatly simplied in this way since it does not require extra communications between the preceding and succeeding servers. The only things a server needs to do is to receive an agent, execute it and forward it to another server by email. The task of migration is completely done by rewriting the script itself.

3.2.3.3 Homecoming Agent

A special instruction "<script end>" denotes the termination of the agent. The VM will wrap the current state of the agent (i.e. the results of the execution of the agent), and send back to the client through another agent, called *homecoming agent*. Please note that a client can designate a server which is different from the originating

server to receive the results. For example, one can send an agent from his/her PDA and ask the agent to deliver the results to his/her HBN, instead of his/her PDA.

3.2.4 Client Service Center (CSC)

CSC provides the owner of an agent a tool to monitor and control the agent and to retrieve its results. To offer on-line services, CSC needs to be executed continuously. Considering the user friendliness and the popularity of WWW browers, we chose to use HTML/WWW format for client interface. Users can use any WWW browser to interact with the CSC of any FlyingCloud server. In order to provide the active display capability to automatically display the results to the client's browser, we implemented the "push" technology by using Java applets. A client can send out an agent and wait on-line through a WWW browser. The results will be "pushed" to the browser automatically by this active display capability.

4. Summary

This paper describes various design issues and the design of the server in the FlyingCloud, the prototyping experiment of a mobile agent service network undergoing in the National Chengchi University. This prototype is designed based on the previously proposed open architecture that is independent of physical communication networks and has greatest flexibility for introducing new services. It also employs the hybrid management mode proposed in [9] to incorporate personal Internet facility for reducing OA&M cost. The agent migration is the most challenging task in our server design. We designed a high level script language and a script rewriting technology to implement the agent migration. We believe this unique approach has opened a new direction for further researches in this area.

References

1. *T. Berners-Lee, et al., "The World-Wide Web," Communications of the ACM, vol. 37, no. 8, August 1994, pp. 76-82.*
2. *Wen-Shyen Chen and Yao-Nan Lien, "Intelligent Messaging for Mobile Computing over the World-Wide-Web", Proc. of the 2nd International Mobile Computing Workshop, Mar. 1996, pp. 42-51.*
3. *David Chess, Bejamin Grosof, and et. al., "Itinerant Agents for Mobile Computing", IEEE Communications, Oct. 1995, pp. 34-49.*
4. *Michael Genesereth and Steven Ketchpel, "Software Agents", CACM, July 1994, pp. 48-53.*
5. *T. Imielinski and B. R. Badrinath, "Mobile Wireless Computing: Challenges in Data Management," Communication of ACM, August 1994.*
6. *Ichiro Lida, Takashi Nishigaya, Koso Murakami, "DUET: An Agent-Based Personal Communications Network", IEEE Communications, Nov. 1995, pp. 44-49.*
7. *Yao-Nan Lien, "Perspective of Service Networks on National Information Infrastructure," CCL Technical Journal, No. 35, Dec 1, 1994, pp. 28-36.*
8. *Yao-Nan Lien, "Client and Agent Mobility Management," Proc. of the Second Workshop on Mobile Computing, Hsing-Chu, Taiwan, March 1996, pp. 141-152.*
9. *Yao-Nan Lien, "An Open Intelligent Messaging Network Infrastructure for Ubiquitous Information Service," Proc. of the First Workshop on Mobile Computing, Hsing-Chu, Taiwan, April 1995, pp. 2-9.*
10. *Yao-Nan Lien, Yenlin Yin, Chin-Hung Chen, Fuhan Liu and Yuli Hwang, "The Design of FlyingCloud: A Mobile Agent Service Network", Proceedings of the 3rd Workshop on Mobile Computing , March 1997, pp. 53-60.*
11. *P. Maes, "Agents that reduce work and information overload", CACM, July 1994, pp. 30-41.*
12. *Jay Padgett, Christoph Gunther, Takashi Hattori, "Overview of Wireless Personal Communications ", IEEE Communications, Jan. 1995, pp. 28-41.*
13. *Charles Perkins, Kevin Luo, "Using DHCP with computers that move", ACM Wireless Networks, vol. 1, 1995, pp. 341-353.*
14. *M. Weiser, "The computer for the 21st century", Scientific America, 1992, pp. 94-104.*
15. *James E. White, "Telescript Technology: The Foundation for the Electronic Marketplace", General Magic, Inc.*
16. *TIA/EIA IS-41, "Cellular Radio Telecommunications Intersystem Operations", Telecommunications Industry Association, Dec. 1991.*