Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

FlyingCloud: A Mobile Agent Service Network

by

Yao-Nan Lien, Yenlin Yin, Tony Chan, Fuhan Liu, Chi-Yung Chen, Chun-Shyan Hwang
Yuli Hwang, Chun-Ing Lee, Shin-Yi Leu, Chin-Hung Chen, and Yang-Fam Liu

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 the people in the world requires not only the Information Superhighway, but also a non-traditional computing paradigm, such as the intelligent messaging, to overcome the intermittent connection problem inherent in a mobile environment. This paper describes the development of a mobile agent service network prototype currently undergoing in the National Chengchi University, the FlyingCloud. 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 will be developed according to the previously proposed open architecture and hybrid mobility management infrastructure [15,16,17].

Keywords: mobile computing, mobile agent.

1. Introduction

1.1 Agent and Agent Mobility

Because of the advance of computer and communication technologies as well as the promotion of National Information Infrastructure (NII), the progress of *mobile computing* is accelerating into a revolutionary speed making the dream of ubiquitous information service a reality [2,3,15,22,23,24]. The goal of a ubiquitous information service network is to provide information to the 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 communications networks such as AMPS or GSM cellular networks will be able to support the ubiquitous communication requirement [1,18,20,26,27]. 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 such that clients, which 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 will have to take a much longer time to realize in the real world.

Thus, 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 that requires its client to interact with multiple servers intensively. 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 [4,5,6,8,9,10,12,13,19,21,25].

Simply speaking, an *intelligent message* is an electronic message that carries a computer program, whether procedural or declarative, that 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 [8,11]. For simplicity, it is called an *agent* in the rest of this paper. Good examples can be found in [15,16].

Since an agent may be traveling in a service network, the originating client may not be able to trace or control its operation directly. A service network must provide some mechanisms allowing its clients to trace and control these agents. This problem is referred to as the *agent mobility management*.

1.2 Mobility Management

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

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

Service). Followings are some critical OA&M problems with respect to the mobility management raised in [16]:

1. How to locate an agent in a service network?
2. How to locate a client user?
3. How to know the status of an agent?
4. How to control the execution of an agent that is traveling in a service network?
5. How to trace the execution of an agent (e.g. for debugging or auditing purposes)?

They are by no means exhaustive. Furthermore, there are other issues such as transaction and security supports needed to be addressed as well [16]. The initial version of our prototype will facilitate the study of the agent and client mobility management.

2. Mobile Agent Service Networks

2.1 Open Service Network Architecture

Traditional telecommunication networks such as PSTN and 800 Toll-Free service used to take considerable resources and long deployment duration to establish. One major resource drain in such networks is the OA&M (and provisioning in some cases). It will be impractical to demand the comparable resources to support the OA&M functionalities in many perspective information services. The computing community will have to rely on themselves, rather than the telecommunication community, to develop and deploy the demanded functionalities. All infrastructures and solutions must not require any change to the existing telecommunication network. To achieve this, we employ the open service network architecture proposed in [16], which separates service networks from transport networks. It also allows services of any scale and any quality to be introduced into the network easily. Service providers can choose whichever operation model and the QoS level based on the resources available to them. That architecture is summarized as follows:

Basic entities are servers each providing a specific information service. They are connected by various logical or physical communication networks such as the Internet, the PSTN, or the ARDIS radio network. The networks that provide connectivity services between servers and clients are referred to as the *transport networks*. Any number of server can be integrated together to form a *service network* providing higher level services to their client users. A server can participate into more than one service network. A *terminal* is a physical device that allows a client to interact with a transport network (and service networks). A terminal could be a telephone, a Personal Digital Assistant (PDA), a desktop PC, or a workstation, etc. (A terminal is associated with a transport network, while a client user is associated with a service network. There is no fixed relationship between a client user and a terminal.) This architecture provides the transport network transparency as well as the required flexibility to the service networks. Readers are referred to [16] for details.

2.2 Hybrid OA&M Supporting Mode

Centralized OA&M support is easier to achieve higher QoS. However, it suffers higher operation cost and long deployment time duration. On the other hand, distributed support is usually more flexible in introducing new services and has much lower operation cost, but it suffers lower QoS. The open service network architecture accepts both approaches. However, many perspective information services will not be able to afford the expensive centralized OA&M support, while they need certain level of QoS beyond what a distributed approach can offer. In [17], 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. Those critical and more appropriate to be managed centrally, such as security and billing functions, are classified as non-distributable and must be managed centrally. It is yet to be researched to define and to classify all OA&M functions. Many perspective services will be benefited from this hybrid OA&M supporting model.

Judging from the fact that the Internet users are growing in an exponential rate, most of the mobile computing users in the future will have stationary access to the Internet from 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 [17]. 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.

2.3 Infrastructure for Mobility Management

In order to reduce management cost, the infrastructure proposed in [16,17] makes use of users' facilities to share the OA&M workload. The most important facilities are briefly described in this subsection.

2.3.1 Network Management Center (NMC)

The NMC is a central facility 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. Although, it looks like a single node, it may actually be a number of nodes distributed over a network.

2.3.2 Home Base Node (HBN)

Even though a mobile user may change its locations from time to time, we assume most mobile users have their own home locations and each of them has a most frequently used Internet access point such as a personal account on an Internet-connected system or a PC, called *Home Base Node*. This infrastructure proposes to make use of HBN to share the OA&M workload. When a client user subscribes to a service network, he/she can choose to use or not to use his/her own HBN. A lot of overhead in managing a service network can be saved by using HBNs. A HBN can also offer auxiliary computing resources to help mobile terminals to cope with their residential resource limitation. Readers are referred to [17] for details.

2.3.3 Status Holder

The *status holder* for an agent 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 original access node, or even the NMC itself. A system may require each agent to 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 when he/she submits an agent. An alternative status holder may 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.

2.3.4 Relationship Between Logical and Physical Entities

The facility available on the network can be classified into 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.

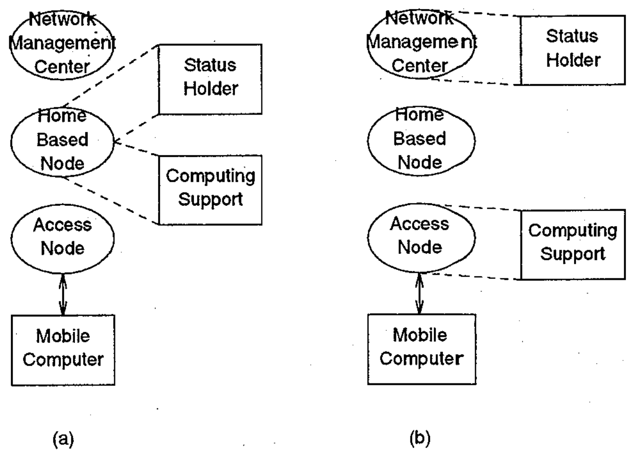

Figure 1. The designation of logical entities to physical entities, (a) both status holder and computing support are designated to the HBN, (b) status holder is designated to NMC and computing support is designated to the Access Node.

3. Client and Agent Management

A service network must be able to manage its client users. Managerial functions include client registration, authentication, tracking, billings, request submission, result delivery, etc. Another important issue is the trace of agents.

After an agent is submitted into a service network, the user or the network manager may need to know its current location in order to inquire its status, or to control its execution, etc. A simple way is to send another agent, called *search agent*, to search the original agent along the original path, or to send a message to every server where the agent might have visited. The first approach takes longer time, while the second one might consume too much wireless connection resources. There is a need to study better solutions that can take the advantage of both approaches with minimum compromise. In [14], various solutions are proposed. One approach is to have an agent to report its status to a status holder such that its approximate solution is immediately available from its status holder. For those that do not have or want a status holder, several blind search algorithms are proposed. If the execution time of each task in each server can be estimated, some statistic calculation can be taken to predict the location of the agent to reduce search effort. Interested readers are referred to [14] for details.

The main difference between agent search strategies and traditional data search strategies is that the target agent itself may be moving during the search. A successful search strategy must prevent a target agent from slipping through the search windows.

4. Current Implementation

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 operation system. Thus, there is a need to create a simulated environment to facilitate further study in various critical issues mentioned above. A project is currently undergoing in NCCU to develop an agent mobility management network prototype, the *FlyingCloud*. The platform will consist of a set of servers capable of intelligent messaging support, a script language, and an agent management system.

4.1 Transport Mechanism

To compliant 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.

4.2 Design Philosophy

In addition to the architecture principles we proposed in [16], the system will be designed with 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 will be designed with flexibility so that each of its components can be easily replaced.
3. The system will be designed evolutionally that only the simplest platform will be implemented initially and will be evolved gradually into a more complete system.

4.3 Agent

An 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 time, the termination status, 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 visited will all remain active until the agent terminates. Much resource will be tied up by the agent. Further, the concept of mobile agent will be violated, which will have a significant implication to the mobile agent paradigm. For example, the recovery semantic will be quite different. (Nevertheless, it is yet to be studied which way will be better.)

This self-contained property has also another significant implication to the language design, which will be discussed in the Section 4.5.

4.3.1 Agent Status

The status of an agent can be 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 if a spinning agent is waiting for something, it can always resume itself if it decides not to wait.) The details can be found in [16].

4.4 Agent Control

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

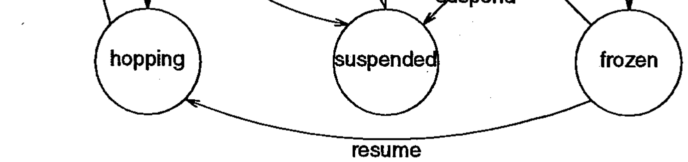

Figure 2. The control of agents.

4.5 Agent Script Language

Designing a sound and complete script language can never be a trivial task, especially for such a newly emerging paradigm. Currently, there are two competing standards, tcl/tk and Java. Neither of them is mature enough for the mobile computing. For the testing purpose, we designed a trivial language for our platform.

In order to fulfill the self-contained requirement, the language does not allow *split-context* execution. For example, the following program construct is **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. According to the self-contained requirement, this is not allowed.

The language we designed in FlyingCloud offers only *string type* and a derived type *list* with some simple form of list operation. The control flow is much like an assembly language that has only *sequential* and *jump* capabilities. It does have *if* construct but has no *loop*. Every statement including *if* must be executed in the same server completely. To avoid unintentional mistake, it does not offer *loop* construct yet. It will be considered in the future.

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

```
 1  name: danny
 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. The variables whose life time span across server boundaries must be exported and will be carried in the global variable section, which is not shown in this example script.

4.6 Server

When an agent (an email message) reaches a server, a daemon will be activated by the email daemon to process the incoming agent. This subsection will discuss the implementation of server.

The first design consideration is the management of sharable information such as user profiles, execution log, agent status, etc. Considering the reliability objective, distributed database technology is rejected and decentralized database will be used. Each server maintains its own local information such as execution log. Client information will be centrally maintained and distributed to all other sites. Considering its low update frequency, replicating client user information to all servers won't be too much a burden. Nevertheless, in our initial design, the authentication process will be carried out by the Home Base Node of each client user. To simplify the registration process, each client user except the network manager can only register to a server (and will have only one authentication server for each client user), all requests that need to be authenticated will be forwarded to its HBN first. In the future design, authentication will be distributed to all other servers in the same service network to improve the accessibility preventing a client from being rejected by the network due to a failure occurring in its HBN.

The second consideration is the environment migration for an agent to move from one server to another. The environmental objects belonging to the agent and the intermediate results must be carried with the agent, while server dependent environmental objects will be released. Since we do not assume any direct communication mechanism other than email between two servers, the migration must be carried out in the form of script. In other words, the same script language is used not only for the implementation of client's requests, but also for environment migration.

The network manager who maintains the agent network through a management system must be authorized to access all other servers (via the same agent transport mechanism). It can be seen as a *super-user* who has registration on all servers.

Normally, the path taken by an agent is dynamically defined by the script contained in the agent. The agents will travel automatically in the network while the script is being executed. However, an agent can also travel through a predefined path with much less overhead. A typical example is an agent that performs routine network management functions such as server status collection. (Note that although a network manager, with a risk of traffic congestion, can broadcast messages to all servers to collect their status simultaneously, it can also send an agent to collect the status of all servers sequentially without risking a traffic congestion.)

The communication structure between system manager, management server, clients, and regular servers is depicted in Figure 3.

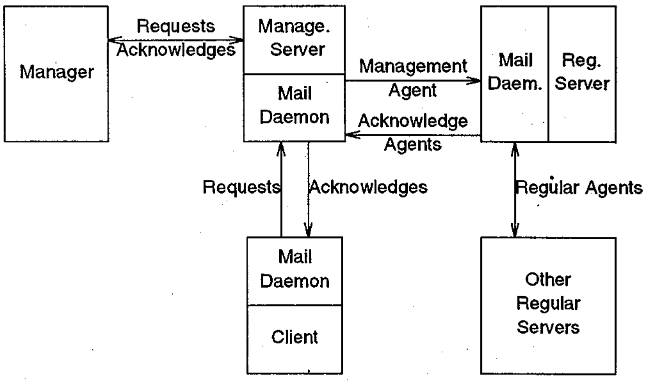

Figure 3. The communication structure of management system.

4.6.1 Agent Log

In order to trace the execution history, the execution of an agent will be logged in each server it visited as well as in the agent itself. Basic logged information in servers are: (1) user ID (optional if it is part of agent IDs), (2) agent ID, (3) arrival time, (4) departure time, (5) resource usage, (6) next node, and (7) termination condition.

4.7 Management System

The management system acts as a super user using the same email protocol to communicate with all servers. Basically, the management system is a facility provided to the service network managers to manage the network. Major managerial functionalities are agent control and network monitoring. Detailed functionalities are described in [16].

In this platform, users are assumed using Internet to access the service network. Therefore, a user can be reached by email address regardless his/her location if he/she is connected to the Internet through a location independent communication network such as cellular phone. In the future, the system will allow a user to access to the service network from various IP addresses. Under this circumstance, a client mobility management mechanism must be in place to assist the service network to track the Internet location of its client users. The operational infrastructure proposed in [16] will be implemented.

4.8 Resource Management

4.8.1 Agent Resource Control

Similar to a process in an operating system, each agent is allocated with some resources, such as size in byte and execution cycle time. There must be some control mechanism to prevent an agent from over runing the allocated resources. There are plenty of resource control policies that can be borrowed from operating system area. This system will focus on the problems that are unique in our environment.

One way to avoid information overflow in retrieving large amount of information from a server is to transfer the retrieved information back to the client immediately before traveling to the next server.

4.8.2 Garbage Collection

The memory leak problem may seriously hurt the reliability of a long running process. Similarly, the resource leak may freeze the entire system by exhausting all its resources. The potential sources of resource leak include:

1. run-away agents,
2. lost agents,
3. agents whose owners disengage with the network for various reasons, and
4. log and status databases.

A garbage collector will be implemented to reclaim the leaked resources. Because the service network allows long-running job, such as watch-dog agents that stay in the network without explicit termination condition, it is not easy to precisely distinguish a run-away agent and a long-running agent. One way to solve this problem is to have all long-running agents whose life time are longer than the network default to explicitly specify their expected life duration.

4.9 Atomicity

The execution of an agent may terminate before it runs to completion. Exceptional cases include server failure, agent failure, network failure, forced abort, etc. The integrity of the service network may be destroyed by these abnormal events and, thus, some atomicity and recovery mechanisms must be provided to protected the entire service network. It is yet to be researched to find appropriate atomicity definitions and associated recovery mechanisms. Apparently, the atomicity definition in the database area is not sufficient. In addition to the atomicity with respect to a database transaction, the atomicity with respect to a server visit and to the entire lifetime must also be defined. Recovery mechanisms must be provided to dear with agent, server, and network failures.

4.10 User Interface Design

Considering the portability and maintainability, the user interface is designed on top of the WWW platform so that managers can manage the network through Internet with a decent GUI interface. Java language will be used for graphic and dynamic information presentation. Some GUI interface frames are shown in Figure 4.

5. Summary

A ubiquitous information service environment needs to offer the client mobility allowing its client user to move across different logical networks. To overcome the intermittent connection problem inherent in mobile environments, it also needs to offer the agent mobility allowing its users to access network services by sending an intelligent message to cruise the network. To guarantee the QoS for a service network, an OA&M system is needed to manage both mobilities.

The OA&M system are often to be the most expensive and most complex system module in supporting a service network. Due to a lack of real experience in operating a ubiquitous information service network, there are many problems related to this new computing environment yet to be studied.

This paper describes the FlyingCloud, the prototyping experiment 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 [17] to incorporate personal Internet facility for reducing OA&M cost. This experiment will allow us to observe this new computing paradigm more closely.

References

1. J. F. Bartlett, "W4 - The Wireless World Wide Web," *Proc. of the IEEE Workshop on Mobile Computing Systems and Applications,* December 1994.
2. T. Berners-Lee, et al., "The World-Wide Web," *Communications of the ACM,* vol. 37, no. 8, August 1994, pp. 76-82.
3. T. Berners-Lee, R. Cailliuau, J.-F. Groff, and B. Pollermann, "World-Wide Web: The Information Universe," *Electronic Networking: Research, Applications and Policy,* vol. 1, no. 2, Meckler, Westport, CT., Spring 1992.
4. 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.
5. Wen-Shyen Chen, Yao-Nan Lien and Wen-Yee Hsien, "An Intrastructure for Mobile Computing with Intelligent Messaging: Implementation Issues", *Proc. of 1996 Workshop on Distributed System Technology and Applications*, May. 1996, pp. 270-277.
6. David Chess, Bejamin Grosof, and et. al., "Itinerant Agents for Mobile Computing", *IEEE Communications,* Oct. 1995, pp. 34-49.
7. G. H. Forman and J. Zahorjan, "The Challenges of Mobile Computing," *IEEE Computer,* April 1994, pp. 38-47.
8. Michael Genesereth and Steven Ketchpel, "Software Agents", *CACM,* July 1994, pp. 48-53.
9. T. Imielinski and B. R. Badrinath, "Mobile Wireless Computing: Challenges in Data Management," *Communication of ACM,* August 1994.
10. M. F. Kaashoek, T. Pinckney, and J. A. Tauber. "Dynamic Documents: Mobile Wireless Access to the WWW," *Proc. of the IEEE Workshop on Mobile Computing Systems and Applications*, December 1994.
11. K. D. Kotay and D. Kotz, "Transportable Agents," *Proc. of the Third International Conference on Information and Knowledge Management,* December 1994.
12. James Lee, "Intelligent Messaging Paradigm and Telescript," *CCL Technical Journal,* No. 35, Dec 1, 1994, pp. 37-41.
13. Ichiro Lida, Takashi Nishigaya, Koso Murakami, "DUET: An Agent-Based Personal Communications Network", *IEEE Communications,* Nov. 1995, pp. 44-49.
14. Yao-Nan Lien and Roger Leng, "On the Search of Mobile Agents", *in The Seventh IEEE International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC'96),* Oct. 1996, pp. 703-707.
15. Yao-Nan Lien, "Perspective of Service Networks on National Information Infrastructure," *CCL Technical Journal,* No. 35, Dec 1, 1994, pp. 28-36.
16. Yao-Nan Lien, "Client and Agent Mobility Management," *Proc. of the Second Workshop on Mobile Computing,* Hsing-Chu, Taiwan, March 1996, pp. 141-152.
17. 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.
18. Yi-Bin Lin, "Determining the User Locations for Personal Communications Services Networks", *IEEE Transactions on Vehicular Technology* , vol. 43, no. 3, Aug. 1994, pp. 466-473.
19. P. Maes, "Agents that reduce work and information overload", *CACM,* July 1994, pp. 30-41.
20. Jay Padgett, Christoph Gunther, Takeshi Hattori, "Overview of Wireless Personal Communications ", *IEEE Communications*, Jan. 1995, pp. 28-41.
21. Charles Perkins, Kevin Luo, "Using DHCP with computers that move", *ACM Wireless Networks,* vol. 1, 1995, pp. 341-353.
22. R. J. Vetter, C. Spell, and C. Ward, "Mosaic and the World-Wide Web," *IEEE Computer,* October 1994, pp. 49-57.

23. G. M. Voelker and B. N. Bershad, "Mobisaic: An Information System for a Mobile Wireless Computing Environment," *Technical Report, Department of Computer Science and Engineering*, University of Washington, September 1994.

24. M. Weiser, "The computer for the 21st century", *Scientific America*, 1992, pp. 94-104.

25. James E. White, "Telescript Technology: The Foundation for the Electronic Marketplace", General Magic, Inc.

26. Mohammed Zaid, "Personal Mobility in PCS", *IEEE Personal Communications*, vol. 1, no. 4, 4th Qtr. 1994, pp. 12-16.

27. TIA/EIA IS-41, "Cellular Radio Telecommunications Intersystem Operations", *Telecommunications Industry Association*, Dec. 1991.

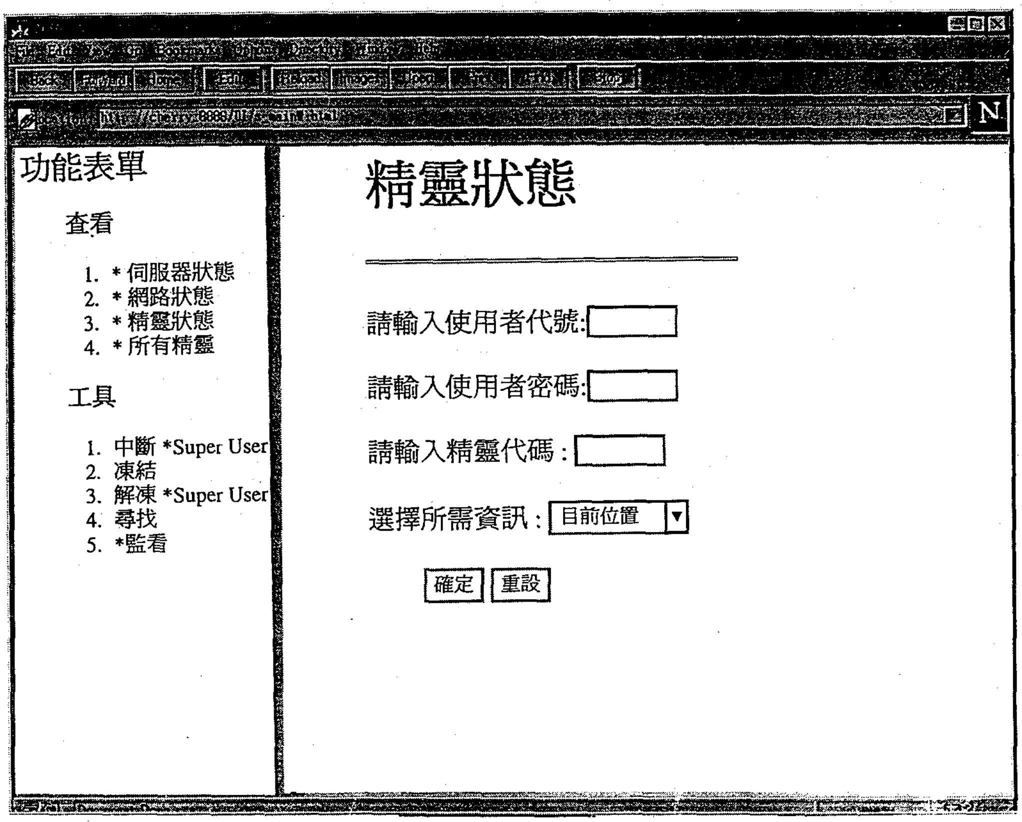

Figure 4. A user interface example for Management System.