Middleware for Ubiquitous and Seamless Computing Environments

● Morihisa Kawai ● Ikuo Takekawa ● Yuji Wada
● Nobutsugu Fujino

(Manuscript received February 2, 2004)

The wide distribution of the Internet and wireless communication technologies is creating ubiquitous environments in which many users can access a variety of services anytime and anywhere. In this paper, we describe middleware that provides users with seamless and ubiquitous environments for computing and communication. This middleware automatically provides users with the optimum computing and communication facilities for their current environments. We introduce three key technologies for realizing this middleware: seamless roaming, Mobile IP, and Plug-and-Service.

1. Introduction

Recently, many hardware facilities for realizing a ubiquitous computing environment, for example, wireless Local Area Networks (LANs) and the third-generation (3G) public mobile infrastructure, have been rapidly developed. Now, the Internet can be accessed with ease from almost anywhere using a variety of methods.

However, these hardware facilities only provide a path to the Internet for e-mail and Web browsing. They do not provide any intelligence to help users find a network or set up a user access environment.

The drawbacks of the current system include the following:

1) Setting up the functions for making network connections is complicated.
2) Services are interrupted when users change location.
3) There is no easy way to determine which kind of services a network supports.

In this paper, we introduce several mechanisms for connecting and receiving services from a network automatically and securely, without requiring any knowledge about the network. The main features of these mechanisms are as follows:

1) Automatic selection of the most suitable access media among currently available networks; for example, wireless LAN or cellular phone (optimum network selection).
2) Seamless continuation of applications, even when the terminal transfers from one network to another (seamless roaming).
3) Automatic discovery and execution of locally provided services from the currently accessed network (Plug-and-Service).

Section 2 of this paper describes the functions of optimum network selection and seamless roaming. Section 3 describes seamless roaming using Mobile IP[1),2)] technology. Section 4 describes the Plug-and-Service mechanism, which enables a user to dynamically and securely use local services that suit the user's preferences.

2. Optimum network selection

To select the most suitable network and automatically connect to it, the currently available networks must be identified and it must be decided how to connect to the most suitable network for the user's preferences.

1) Currently available networks

To select a network, it is necessary to determine which networks are currently available. This includes determining which network devices are attached to the mobile node (note PC, Personal Digital Assistant [PDA], etc.) and whether the node is within the network's service area.

2) User's preferences

When selecting a network, the type of network the user wants to connect to must be considered. For example, some users want to connect to a fast network, while others want to connect to a low-cost one.

3) How to connect to a network

The methods available for connecting to a network may be limited depending on the user's location and the target network. For example, to connect to a company's intranet with a dial-up, a user can call directly to the company's access point or to an Internet Service Provider (ISP). However, connecting to an intranet via the Internet usually requires Virtual Private Network (VPN) software.

2.1 Network selection with an agent

A software agent can select the most suitable network more effectively than a user. We have developed seamless roaming software that features automatic network selection with agents. **Figure 1** shows the configuration of the software in a mobile node. The software is composed of two agents: Personal Agent and Network Agent.

Personal Agent is an agent for controlling connections to a network and the load of applications instead of users. It holds a user profile containing user preferences and authentication information for network access. It monitors user applications and requests Network Agent to access a network when those applications are loaded.

Personal Agent informs Network Agent about the user's preferences such as the network connection priority (e.g., select the fastest network first or the cheapest network first) and requests Network Agent to select the most suitable network. Likewise, it receives information about the network that Network Agent has selected and starts an application if the network qualities match the ones that an application requires.

Network Agent is an agent for making a network virtual and hiding complicated network set-up options and operations from users and applications. Network Agent stores the bandwidth that a network can provide, accounting information, and access-point information in a network profile, and monitors which network devices are attached to a mobile node and whether the node is in a service area.

When connectivity to the currently accessed network is predicted to be discontinued, Network Agent looks for another suitable network using the network profiles and user profile stored in Personal Agent.

For example, if a user prefers a low-cost network, Network Agent uses its network profiles to select the network with the optimum combination of network cost and ISP cost.

Likewise, Network Agent uses its network profiles to detect the network to which a mobile node currently belongs. If a mobile node needs to connect to an intranet, Network Agent judges whether VPN software must be used, loads the software if necessary, then connects to the intranet.

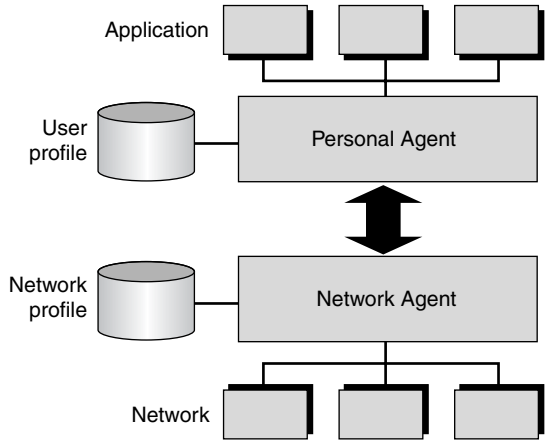

Figure 1
Configuration of terminal-side software.

2.2 Automatic connection with an agent

After a network has been selected as described above, in most cases, to connect to a network a user will need to provide authentication information such as an ID and password. Sometimes, additional information will be required to use VPN software. However, if users need to input information at each network roaming, the service is not seamless.

Our seamless roaming software can automatically connect to a network by using the automatic authentication mechanisms of a software agent.

Currently, public wireless LAN services have their own authentication mechanisms and employ different VPN software from each other. Personal Agent can automatically cooperate with these mechanisms to realize smooth network connections and seamless network roaming.

Usually, authentication information is securely encrypted and stored inside a terminal, but it is possible to store this information in a secure external device such as a Universal Serial Bus (USB) key token or a Subscriber Identity Module (SIM) card.

3. Seamless roaming

An important technology for realizing seamless roaming is Mobile IP. When a mobile node transfers from one network to another, its IP address changes. If this address change occurs while the mobile node is communicating, the communication is terminated. In many cases, an application running on the mobile node will have to be restarted because of this address change. As its name indicates, Mobile IP enables a mobile node to transfer within an IP layer. It provides a mechanism that makes the movement of a mobile node invisible from upper layer applications. Mobile IP makes it unnecessary to restart an application and enables a mobile node to continue communicating with another node even if the mobile node transfers to another network.

Figure 2 shows the general procedures of Mobile IP. The network that a mobile node usually connects to is called the home network. The home network allocates one fixed IP address to each mobile node, which is called the home address. In a home network, there is at least one node that controls the movements of each mobile node, and these nodes are called home agents. In a network to which a mobile node transfers (called a foreign network), there is usually a node called a foreign agent that handles the mobile node. The processes of Mobile IP are described below.

1) A mobile node transfers to a foreign network.
2) An IP address is allocated to the mobile node in the foreign network. This address is called a care-of-address.
3) The mobile node registers the care-of-address with its home agent every time the care-of-address is changed.
4) The home agent controls the care-of-address of the mobile node and receives all packets for the mobile node at its home address instead of the mobile node.
5) The home agent encapsulates the packets with the care-of-address of the mobile node and sends them to the foreign agent.
6) The foreign agent receives and decapsulates the packets and sends them to the mobile node.

A foreign agent can be implemented inside a mobile node, and this mode of operation is called co-located mode. Co-located mode operation is essential for Mobile IP, because a foreign agent is not required in a foreign network. In addition, there is another operation mode in which a mobile node registers its care-of-address with not only a home agent but also the target node for the intended communication. In this case, the mobile node can directly communicate with the target node.

By making Mobile IP cooperate with Network Agent, a mobile node can keep communicating when it transfers to another network.

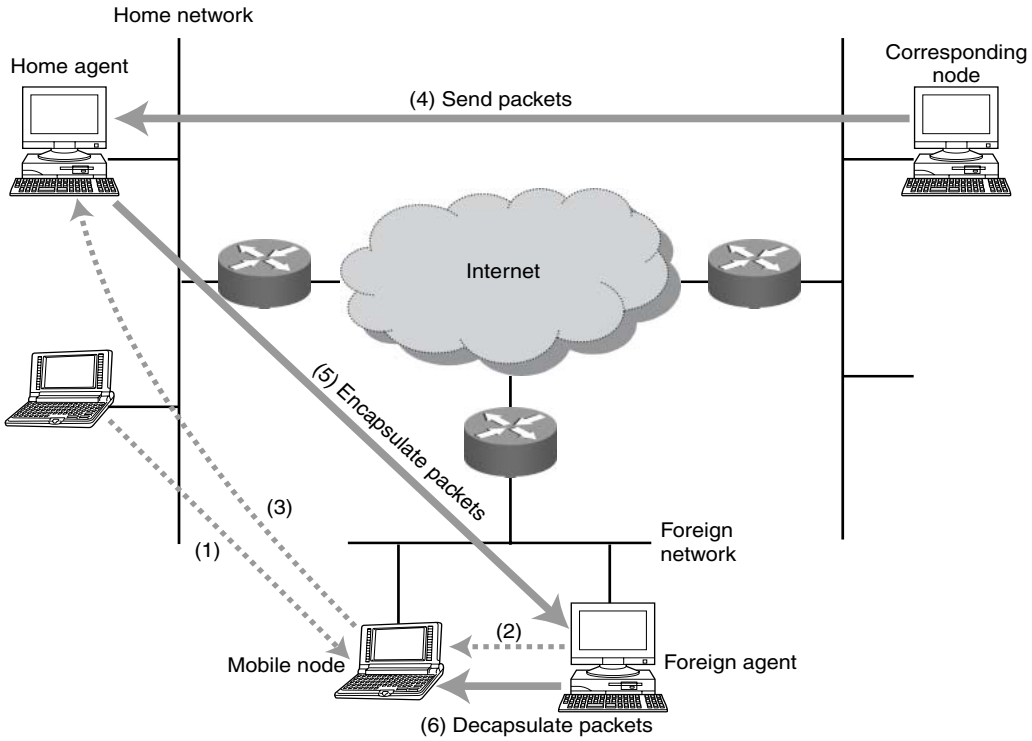

Figure 2
Outline of Mobile IP.

4. Plug-and-Service

One of the most important technologies for achieving ubiquitous services in ubiquitous networks is Plug-and-Service. Jini[3] and OSGi[4] provide a Plug-and-Play function for services. However, in these frameworks, the services need to be implemented in Java. Because of this restriction, ubiquitous service providers cannot freely choose the implementation language of their services. Our Plug-and-Service technology allows providers to implement their services in many different ways by encapsulating all the components needed to use a set of services in a self-contained service package. Users can then use the services simply by obtaining the service package. For example, a service package can encapsulate the programs and data that implement the services.

4.1 Service packages

Ubiquitous services should be available in all ubiquitous networks. Moreover, these services should be usable even when the user is disconnected from a network. As mentioned in the previous section, in our framework, ubiquitous services are distributed in the form of self-contained service packages. After obtaining a service package, a user can use the encapsulated services anytime and anywhere.

As shown in **Figure 3**, a service package consists of three descriptions. The use condition description defines which types of users can access the services encapsulated in the service package. The use condition description can be used, for example, to define that users who have a specific member license can use a specific set of the encapsulated services. The service startup description gives the service codes, initialization

procedures, and other information needed to launch the services. The service content description provides the service codes and service data that are required to offer the services.

The use condition description can be regarded as a mapping from the users' profiles to the service names. In Figure 3, the first mapping specifies that a user can use service "Service1" only if the user has attribute "AttributeA." By using this description, a service provider can control who uses the services encapsulated in a service package without a central server.

The service startup description can be regarded as a mapping from service names to the methods used to provide the corresponding services. In Figure 3, the description specifies that service codes "svc.Service1," "svc.Service2," and "svc.Service3" must be executed for services "Service1," "Service2," and "Service3," respectively. By using this description, a service provider can specify which service codes must be executed to offer a service, which method or function must be invoked to initialize them, and which values must be passed as arguments for the initialization.

The service content description specifies what is needed to offer the encapsulated services. In Figure 3, the description states that service code "service.jar" and service data "service1.html," "service2.html," and "service3.html" are required for the encapsulated services. By executing the service code, the services encapsulated in the service package can be used and the service data can be used by the service code.

Figure 4 shows an example description of a service package. The use condition description defines that a user can use service "Print" if the user has attribute "Print"; otherwise, the user can use service "NoPrint." The service startup description states that service "Print" requests that service code "Print.PrintSvc" be executed and initialized by invoking method "setFileName" with "print.htm" as its argument. It also states that service "NoPrint" requests that service code

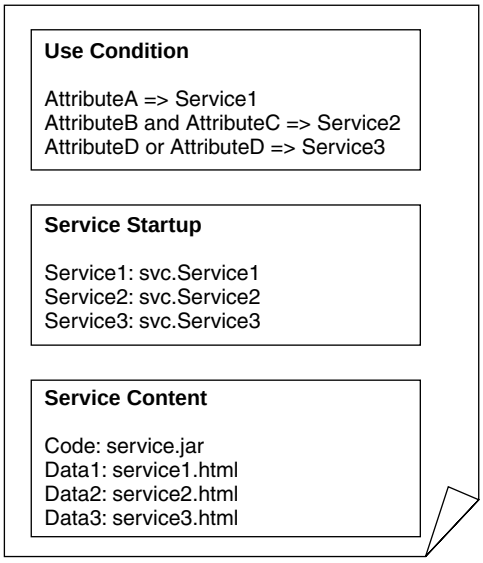

Figure 3
Service package structure.

```
<!-- Service Package -->
<POLICY NAME="Print">
 <RULES NAME="Print rules">
  <RULE>
   <CONDITION>
    <EQ ATTRIBUTE="Print" VALUE="yes"/>
   </CONDITION>
   <ROLE_NAME NAME="Print"/>
  </RULE>
  <RULE>
   <CONDITION>
    <NOT> <EQ ATTRIBUTE="Print" VALUE="yes"/> </NOT>
   </CONDITION>
   <ROLE_NAME NAME="NoPrint"/>
  </RULE>
 </RULES>
 <ROLES NAME="Print roles">
  <ROLE>
   <ROLE_NAME NAME="Print"/>
   <CLASS_PATH PATH="Print.PrintSvc"/>
   <INIT METHOD_NAME="setFileName">
    <ARG>print.htm</ARG>
   </INIT>
  </ROLE>
  <ROLE>
   <ROLE_NAME NAME="NoPrint"/>
   <CLASS_PATH PATH="Print.NoPrintSvc"/>
  </ROLE>
 </ROLES>
 <CONTENTS NAME="Print contents">
  <CONTENT NAME="print.htm" PATH="print.htm"/>
  <CONTENT NAME="ng.htm" PATH="ng.htm"/>
  <CONTENT NAME="code" PATH="print.jar"/>
 </CONTENTS>
</POLICY>
```

Figure 4
Example description of service package.

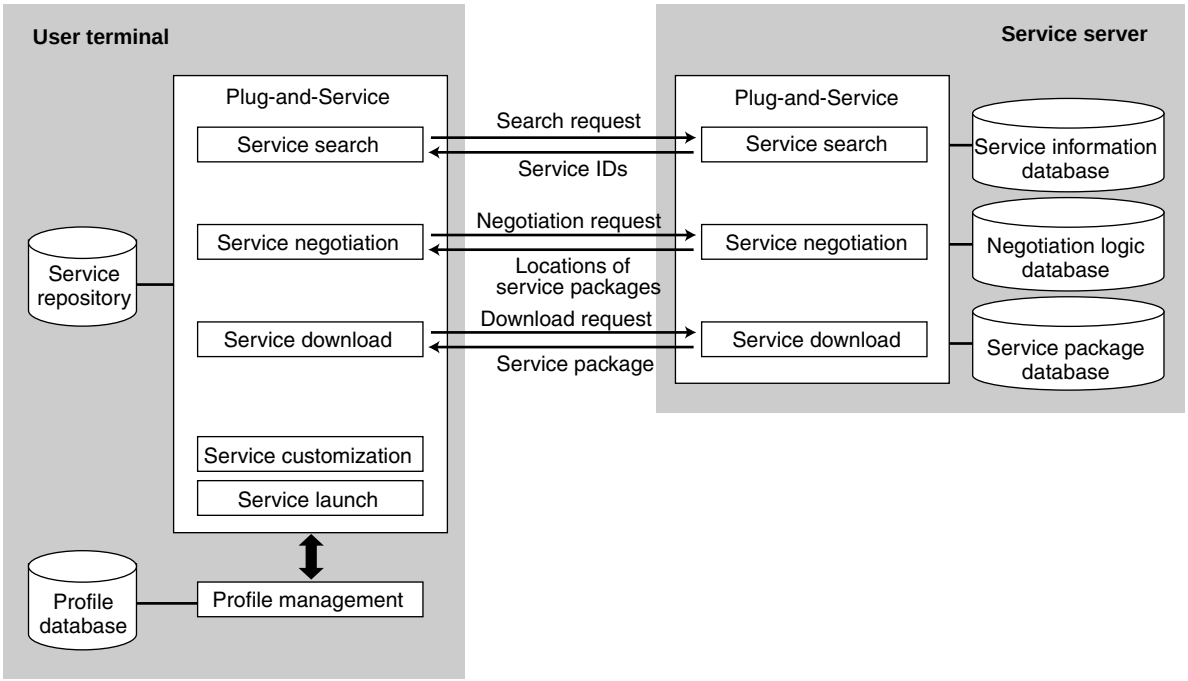

Figure 5
Functional architecture of Plug-and-Service.

"Print.NoPrintSvc" be executed.

4.2 Plug-and-Service architecture

The functional architecture of Plug-and-Service is shown in **Figure 5.** The architecture consists of five functions: service search, service negotiation, service download, service customization, and service launch.

The service search function searches for services that satisfy the user's preferences. The service negotiation function determines which of those services can be executed on the user's terminal. The service download function obtains the service packages that encapsulate the executable services. These three functions require cooperation between the user's terminal and the service server. As shown in Figure 5, the Plug-and-Service platforms of the user's terminal and the service server have these three functions.

After downloading a service package, the Plug-and-Service platform on the user's terminal performs the remaining two functions. The service customization function selects the appropriate services for the user based on the user's profile. The service launch function starts the services selected by the service customization function. The service codes for the selected services can be on the user's terminal beforehand or encapsulated in the downloaded service packages.

5. Conclusion

In this paper, we described a middleware solution for ubiquitous computing environments. Our solution can help users obtain services anytime and anywhere, without needing to perform any operation.

We are currently supplying a middleware product called "Seamlesslink," which is a middleware platform that enables easy and seamless use of networks. The Plug-and-Service function described in this paper extends "Seamlesslink" to enable easy and seamless use of ubiquitous services anytime and anywhere. In the near future, we will release a new version of a front integra-

tion middleware that integrates Seamlesslink and the Plug-and-Service function.

Part of the Plug-and-Service function described in this paper is supported by the New Energy and Industrial Technology Development Organization (NEDO) of Japan.

6. References

1) C. Perkins et al.: IP Mobility Support for IPv4. RFC3344, August 2003.
2) D. Johnson et al.: Mobility Support in IPv6; draft-ietf-mobileip-ipv6-24.txt. December 2003.
3) Jini.org.
http://www.jini.org/
4) OSGi Alliance.
http://www.osgi.org/

Morihisa Kawai received the B.S. degree in Physics from Rochester Institute of Technology, NY, U.S.A. in 1996. He joined Fujitsu Ltd., Yokohama, Japan in 1997, where he has been engaged in research and development of software for mobile computing.

Ikuo Takekawa received the B.S. degree in Mathematics from Nihon University in 1985. He joined Fujitsu Ltd., Yokohama, Japan in 1985, where he has been engaged in research and development of software for Internet and mobile networks.

Yuji Wada received the B.S. and M.S. degrees in Information and Computer Systems Engineering from Osaka University, Osaka, Japan in 1990 and 1992, respectively. He joined Fujitsu Laboratories Ltd., Kawasaki, Japan in 1992, where he has been engaged in research and development of algorithm animation, parallel systems, distributed systems, and agent/multi-agent systems. He is a member of the Information Processing Society of Japan (IPSJ).

Nobutsugu Fujino received the B.S. and M.S. degrees in Electronics Engineering from Osaka Prefecture University, Osaka, Japan in 1984 and 1986, respectively. He joined Fujitsu Laboratories Ltd., Kawasaki, Japan in 1986, where he has been engaged in research and development of radio communication systems and mobile computing technology. He is a member of the Information Processing Society of Japan (IPSJ).