JOURNAL OF KOREA MULTIMEDIA SOCIETY, VOL. 7, NO. 6, JUNE 2004(pp. 832-840)

A Jini-Based Ubiquitous Messaging System Supporting Context Awareness and User Mobility

Tae-Uk Choi†, Ki-Dong Chung††

ABSTRACT

In ubiquitous environments, *context* is any information that can be used to characterize the situation of an entity such as a person or an object. Many sensors and small computers collect contexts and provide applications with them. Thus, ubiquitous applications need to represent contexts and exploit them effectively. In this paper, we design and implement a context-aware messaging system, called UMS (Ubiquitous Messaging System), based on Java and Jini. UMS can represent various contexts using XML scripts, and communicate text messages regardless of user's location using the proxy mechanism of Jini.

Keywords: Ubiquitous Computing, Jini, Context Awareness, User Mobility

1. INTRODUCTION

Current computing environments are changing from desktop computing, where users compute at a desk, to mobile computing, where users compute while moving. Currently, computing environments are changing from mobile to ubiquitous computing, where users compute at anytime and anywhere. In ubiquitous environments, computers have been becoming much smaller with the progress of hardware technology. These small computers are embedded into all sorts of things such as appliances, cars and buildings. They sense and gather context such as the user's location, mood and room temperature, and give the information to ubiquitous applications.

In these ubiquitous environments, applications need to satisfy the following requirements. First, applications should have a context model to represent and communicate the context. Second, the user is free to move from one space to another. To support user mobility, ubiquitous applications need to cooperate with each other so that user tasks can migrate from one device to another[1]. Third, ubiquitous applications require a flexible, reusable, distributed and platform-independent software architectures. Sun Microsystems recently released Jini[2] technology that provides an environment for creating dynamically networked components, applications and services.

There have been several works related to context aware applications. Cybreminder[3] is a message transmission program that reminds users to do some action in the near future. It collects contexts, such as time and location, based on Context ToolKit[4]. Babble[5] is a CMC (Computer-Mediated Communication) program that simulates social activities in real world. Using the social proxy, it gives a sense of the size of the audience, the amount of conversational activity, and who it is that is coming and going. Hubbub[6] is an instant messenger program that can represent the percipient's activity state and mood using visual icons and voice messages. However, these works

※ Corresponding Author : Tae-Uk Choi, Address : (609-735) San-30, Jangjeon-dong, Geumjeong-gu, Busan, Korea, TEL : +82-51-518-7502, FAX : +82-51-515-2208, E-mail : tuchoi@pusan.ac.kr

Receipt date : Jan. 7, 2004, Approval date : March 30, 2004

† Ph.D student, Dept. Computer Science, Busan National Univ., Busan, Korea

†† Professor, Dept. Computer Science, Busan National Univ., Busan, Korea.
(E-mail : kdchung@pusan.ac.kr)

※ This work was supported by grant No. R05-2002-000-00345-0 from the Basic Research Program of the Korea Science & Engineering Foundation.

do not have a common context model but utilize limited contextual information depending on the applications, and moreover, they do not provide a direct solution for user mobility.

In this paper, we implement a ubiquitous messaging system based on the Jini platform. Compared to conventional context-aware programs, this system uses an XML-based context model to share and communicate contexts, and transmits messages to the user even when the user is moving. Section 2 presents the overview of Jini technology. Section 3 describes the context representation model, and Section 4 discusses the method to support user mobility. In Section 5, we implement the system architecture based on Jini and describe the interface and usage of the implemented Ubiquitous Messaging System (UMS). Lastly, we draw a conclusion in Section 6.

2. OVERVIEW OF JINI

Jini network technology is an open architecture that enables developers to create network-centric services that are highly adaptive to changes. By using objects that move around the network, the Jini architecture makes each service adaptable to changes in the network. The Jini architecture specifies a way for clients and services to find each other on the network and to work together to get a task accomplished. Fig. 1 shows the service, client and proxy in a Jini network[15]. Clients need to download the proxy to communicate with the service. The key concepts of Jini are as follows:

Service: A service is an entity that offers a specific function and is used by a person, a program or another service. Services in a Jini network communicate using a service protocol. Jini provides mechanisms for service construction, lookup, communication and usage. A service is discovered and resolved by a lookup service that acts as a central repository for the service.

Proxy: Services upload serialized Java objects

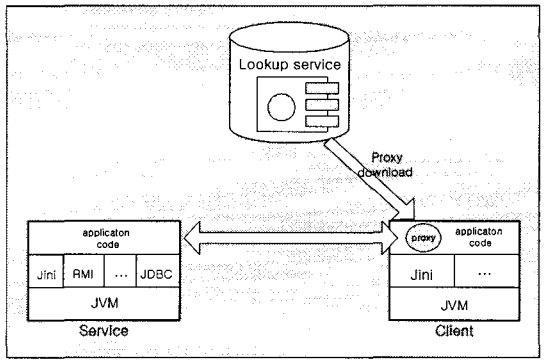


Fig. 1. Jini Service Architecture

(proxies) to the lookup service. This object can be downloaded to any client and invoked to access the service. The proxy encapsulates a protocol or interface for the actual communication between the proxy object and the service.

Discovery/Join: Jini services and clients should find a lookup service before they do anything else. The Discovery/Join protocol provides a built-in bootstrapping mechanism that enables Jini-enabled objects to find a lookup service and register their services.

Lease: For performance of Jini-based distributed systems, Jini allows clients to access services only during a restricted time, not permanently. That is, services are accessed via a valid lease. A lease is negotiated between the client and the service and is guaranteed access over a period of time. After the time expires, the lease is to be renewed if the client wants to use it longer.

3. USER CONTEXT MODEL

A common model for contexts is required to represent and transfer contexts between participants. ConChat[7] presents a context model based on first-order predicate calculus and Boolean algebra. The model covers a wide variety of available contexts and supports various operations such as the conjunction and disjunction of contexts. Jang[8] defined the 5W1H context model that can be applicable to all types of applications. Fig. 2 shows the message format of the Jang's context

Preliminary Context Integrated Context	who + \t + what + \t + where + \t + when + \t + how + \t + why + \0
If one of 5W1H is empty, it can be expressed as -	

Fig. 2 Message format of the 5W1H model

model.

It is not easy for applications to use these models to represent, store, retrieve and transmit complicated or combined contexts. That is to say, implementing the model-based applications is difficult. We define a User Context Model that can represent various contexts and allow applications to configure and communicate context messages easily and simply. The User Context Model is represented as follows:

Context (<User>, <Type>, <Value>)

<User> is the person with whom the context is concerned, <Type> is the type of context such as the user's location, mood, and activity. <Value> is a value associated with the <Type>. Example contexts include

Context (Tuchoi, location, room417)

Context (Shpark, mood, happy)

Context (Hansol, activity, idle)

These contexts means that Tuchoi is currently located in room417, Shpark's mood is happy and Hansol is idle. This model is simple and can express the basic and complicated context types. We represent the model as an XML message to store, retrieve and communicate contexts easily in Jini and Java applications. Fig. 3 shows an example of the context represented by XML.

```
<?xml version= 1.0 ?>
<!--DTD definition -->
<!DOCTYPE contextInfo [
<!ELEMENT contextInfo (ContextItem)*>
<!ELEMENT contextItem (user, type, value)>
<!ELEMENT user (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT value (#PCDATA)> ]>
<contextInfo>
<contextItem>
        <user>Tuchoi</user>
        <type>location</type>
        <value>room 417</value>
</contextItem>
<contextItem>
        <user>Hansol</user>
        <type>mood</type>
        <value>happy</value>
</contextItem>
</contextInfo>
```

Fig. 3. XML-based context message

4. USER MOBILITY

In a ubiquitous environment, a user is free to access specific services at anytime and anywhere. This means that the device that is serving the user could be changed, and task migration is required between devices. Most related works focused on host mobility[8-10], which means that a user carries a device that can access the same service while moving. However, we consider user mobility, which not only includes host mobility, but also includes the case where a user is free to switch from one host to another. This means that the real end point in ubiquitous communications is not the host but the person. Recently, there have been works that can support personal mobility in cellular phones and PDAs[11,12]. MPA[13] tackles the problem of personal mobility by introducing a person layer on top of the application layer. Gui et al.[14] studied the user-level handoff problem of multimedia service delivery in ubiquitous computing environments.

Based on the handoff scheme proposed in[14], we implement a modified handoff scheme that can be exploited in Jini/Java environments. Fig. 4 shows the modified handoff protocol. Upon the user

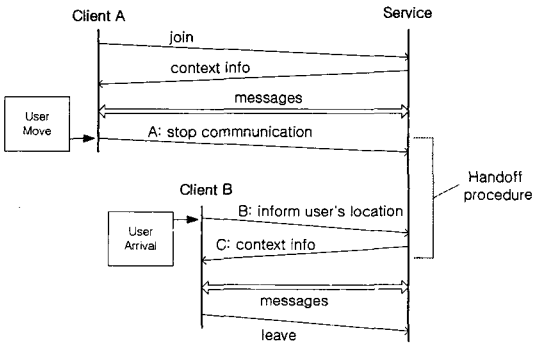


Fig. 4. Handoff protocol between Clients and Service

leaving, Client A stops message communication and informs the service of the user's moving (Step A). Upon detection of the user's arrival, Client B informs the service of the user's arrival, and the service updates the location of the user (Step B). As a relay, the service transfers the last messages and context of all users to be used for the next communication (Step C). After receiving the context, Client B is ready to resume message communications.

In Jini, the handoff protocol can be implemented easily because it utilizes the RMI mechanism, not sockets. Jini applications are free to access the service because they use a proxy that is free to move. Fig. 5 shows the mobility of the proxy. The service keeps track of the users as they move and direct the communication to the appropriate clients.

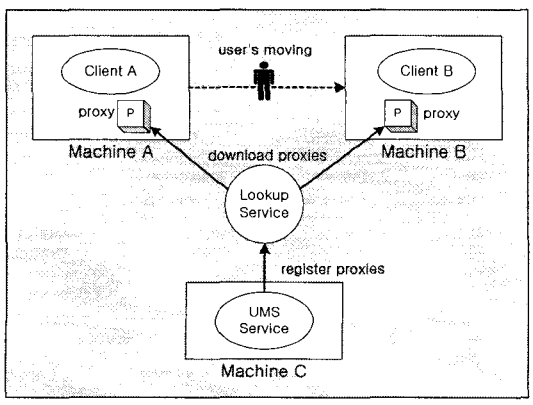


Fig. 5. Mobility of the proxy in the Jini system

The client processes the user commands and monitors user activity from pervasive sensors. The proxy is the interface to communicate messages and context between the client and the service. The client needs to download the proxy to communicate with the service. As a user moves from client A to client B, client B automatically downloads the proxy from the lookup service to get context from the UMS service, and then continue to communicate messages.

5. IMPLEMENTATION

5.1 System Architecture

We have implemented UbiChatter, a prototype Ubiquitous Messaging System, using Jini. It currently allows a user to send and receive only text messages to another user. Fig. 6 shows the class diagram of the prototype system, which consists of the service, client and communication subsystems.

- Service Subsystem

This subsystem maintains contexts and message routing information for user mobility. The UMSService discovers and joins the Jini's Lookup service, and registers the proxy to the Lookup Service. The ContextManager maintains the XML-based contexts. Contexts is stored in and retrieved from local buffer. The SessionManager maintains communication sessions to support user mobility. A session can be stopped and resumed by adding or deleting the listener of the session. The listener is an object of the RemoteEventListener class. As the user moves to another client, the listener for the session is updated by the new client.

- Client Subsystem

This subsystem collects contexts around users and processes user commands. The UMSClient searches a lookup service to download the proxy

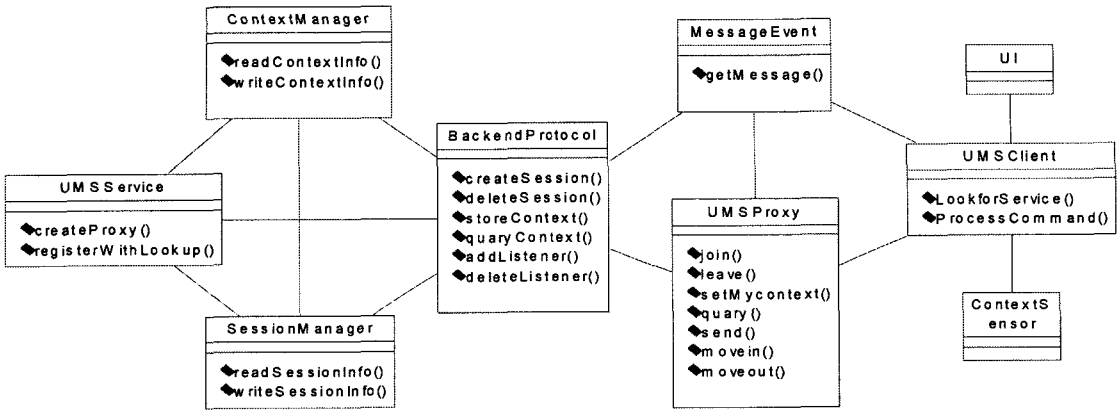


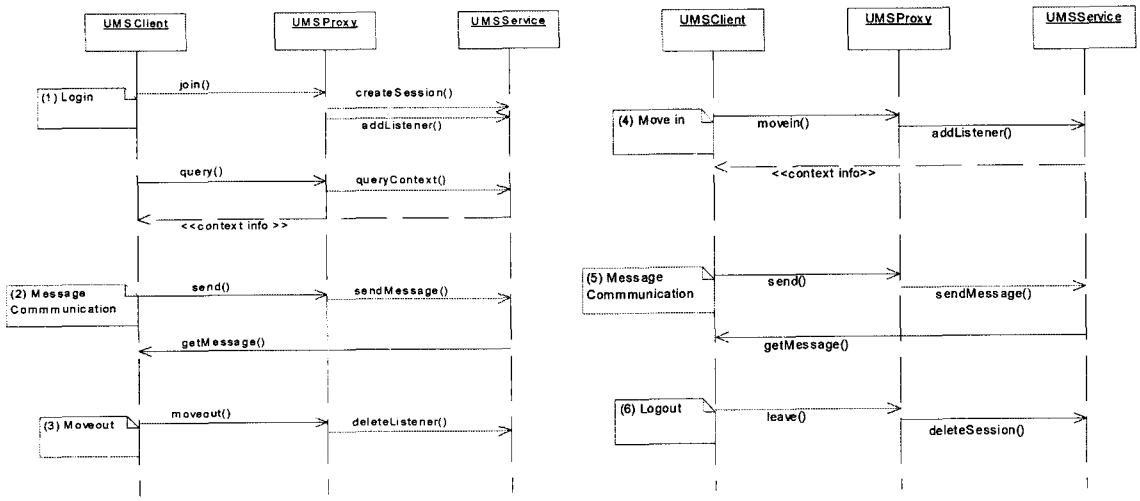
Fig. 6 Class Diagram of the prototype of UMS

of the UMSService and communicates with the service using the proxy. The ContextSensor perceives the contexts around users and transforms the contexts into the XML-based form. The UI displays the information of current connected users, their context, messages sent and received, etc.

- Communication Subsystem

This subsystem uses the RMI mechanism, on which Jini is built, to communicate messages. The client downloads the proxy of the service and then communicates with the service through the proxy. The proxy at the client needs RMI stubs to communicate with the service objects (ContextManager and PLRManager). The BackendProtocol encapsulates an interface for RMI communications. This means the proxy at the client side directly communicates with the BackendProtocl object at the server side. Conversely, when the service transmits a message, RMI can be used too. Fortunately, Jini provides the RemoteEvent and RemoteEventListener class. Once the client sends a RemoteEventListener object to the HandoffManager of the service, the service can send a message to the service using the MessageEvent object.

Fig. 7 shows sequential diagrams related to the



(a) Client A (b) Client B

Fig. 7. Scenario of user mobility

scenario of user mobility. The scenario is as follow:

Step(1): when a user logs into the service in client A, the client invokes the join method to create a new session and add a listener for client A. Then, the client invokes the query method to get the contexts of other users connected in the service.

Step(2): when the user communicates messages with the service in client A, the client calls the send method to transmit a message, and the service calls back the getMessage method as other user sends a message to the user.

Step(3): when the user moves out from client A, the client calls the moveout method to delete the listener connected to the client. So, the client cannot send a message any more.

Step(4): when user moves into client B, the client invokes the movein method to add a new listener for the user. As a reply, the service returns the context of the user.

Step(5): the user resumes message communication with the service in client B.

Step(6): when the user logs out of the service, the client invokes the deleteSession method to close the session for the user.

5.2 Experiments

In several Pentium4 PCs installed Windows 2000 Server and Windows XP, we implement the UMS application using JDK 1.3 and Jini Development Kit 1.2.1. To run Jini applications, the following services are required: HTTP servers to download RMI codes, a RMI demon, which is widely used by Jini services to manage the activation and deactivation of service objects, and a lookup service (reggie) to manage the proxies of application services. Fig. 8 shows the services that are required to run UMS. Fig. 9 shows the service and client of UMS. The service is displaying the message that denotes the service ID after registering the proxy in the lookup service. The client is displaying the message after finding the service and downloading the proxy.

Fig. 10 shows the user interface of the UMS

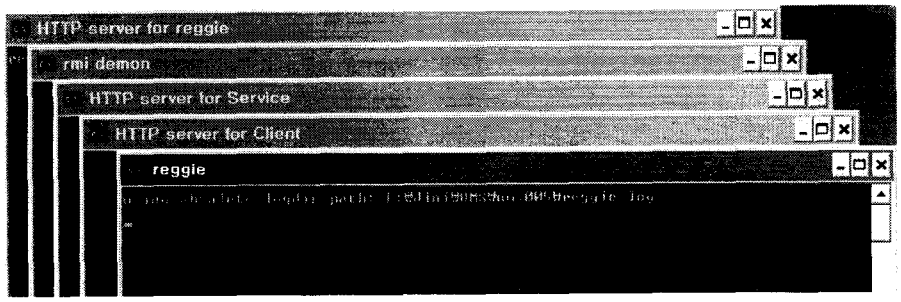


Fig. 8. Requisite services to run the Jini service

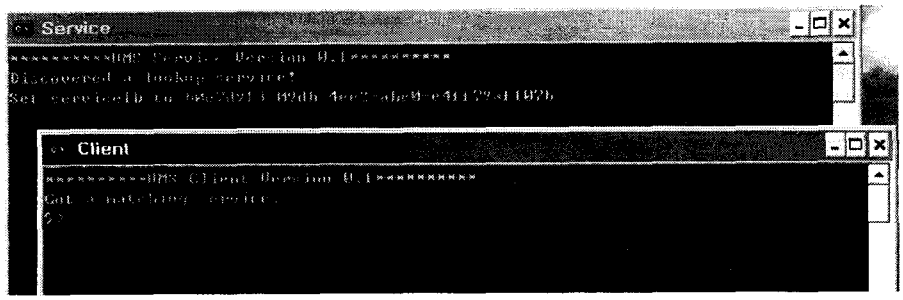


Fig. 9. The service and client of UMS

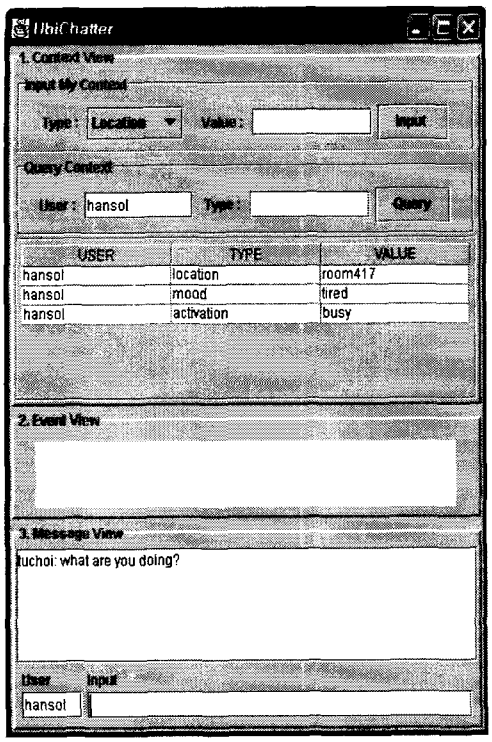
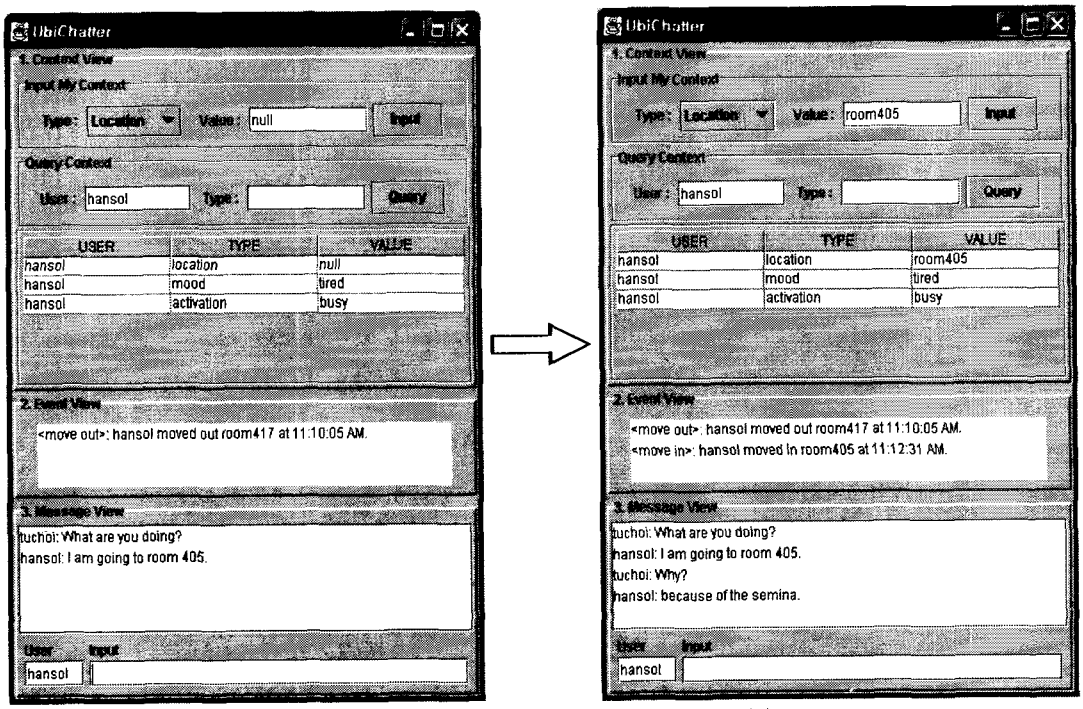


Fig. 10. User interface of UMS

client that consists of Context View, Event View and Message View. In Context View, the logged-in user can publish his/her contexts by selecting the context type and inputting a value. As well, the user can retrieve the contexts of other connected users by querying in terms of user name or context type. In this figure, we can see the result of the query to find the contexts related to the user hansol. Event view displays the events such as user movement when some contexts change. The Message View is similar to that of conventional chatting programs, but the user needs to input the other user ID to send a message.

Fig. 11 shows user mobility in UMS. Before a user moves, the user types the location value to *null* in the previous UMS client. Then the move out event occurs (see Fig. 11 (a)). After moving, the user inputs the new location value in the new UMS client. Then the move in event is displayed (see Fig. 11(b)). Like this, the user continues to communicate messages even though he/she moves.



(a) Before movement (b) After movement

Fig. 11. User mobility in UMS

6. RELATED WORKS

In terms of context and user mobility, we summarized the features of the several messaging systems in Table 1. MSN messenger[16] and ICQ[17] are the messenger programs many people use currently. But they employ only status information such as on-line, off-line and idle. Babble[5] transmits social cues such as audience size and how actively people are participating in a multiparty chat scenario. Hubbub[6] uses sounds to give awareness cues of other people and visualizes the other person's status during a conversation. Conchat[7] can represent various contexts but does not support user mobility. However, proposed UMS can represent various contexts as well as support user mobility.

Table 1. Comparison of the messaging systems

	Context Information	User Mobility
MSN Messenger	Status information (online, idle, offline)	No
ICQ	Status information (online, idle, offline)	No
Babble	Social cues	No
Hubbub	Sound, Icons and Location	No
ConChat	Various contexts	No
UMS	Various contexts	Yes

7. CONCLUSIONS

Ubiquitous applications need to exploit various contexts perceived from many sensors and small computers. This paper has presented a context-aware messaging system as a ubiquitous application. We have discussed a user context model and user mobility that should be considered in the design of ubiquitous applications. Then, we have implemented a prototype of UMS using Java and Jini. Through the test runs of UMS and the comparison to other messaging systems, we found out that the merits of UMS are as follows.

• **Awareness of various contexts**: UMS can provide a user with various contexts such as location, mood and activity. Knowing such contexts makes the communication richer.

• **Support of user mobility**: UMS employs Jini technology to move the Java object code so that a user can continue to communicate messages even though he/she moves.

However, this UMS has no sensors to perceive the contexts from real world, that is, it is dependent on the context information that the user inputs. So, we have plans to embed the sensors in the UMS to collect real contexts. Moreover, we will improve UMS to communicate multimedia data such as voice, music and video.

8. REFERENCES

[1] J. P. Sousa and D. Garlan, Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments, IEEE/IFIP Conference on Software Architecture, Montreal, 2002.

[2] Sun Microsystems, http://developer.java.sun.com/developer/products/jini

[3] A. K. Dey and G. D. Abowd, CybreMinder: A Context-aware System for Supporting Reminders, Proc. 2nd International Symposium on Handheld and Ubiquitous Computing, New York, 2000.

[4] A. K. Dey and G. D Abowd, The Context Toolkit: Aiding the Development of Context-Enabled Applications, Proc. Workshop Software Eng. for Wearable and Pervasive Computing, New York, 2000.

[5] T. Erickson et al., Socially Translucent Systems: Social Proxies, Persistent Conversation and the design of Babble, Proc. Human Factors in Computing Systems, New York, 1999.

[6] E. Lsaacs, A. Walendowski, and D. Ranganathan, Hubbub: A Sound-Enhanced Mobile Instant Messanger that Supports Awareness

and Opportunistic Interactions, Proc. Conf. Computer Human Interaction, New York, 2002.

[7] A. Ranganathan, R. H. Campbell, A. Ravi, and A. Mahajan, ConChat: A Context-Aware Chat Program, In IEEE Pervasive Computing, pp. 52-58, July-Sept 2002.

[8] David Johnson, Scalable Support for Transport Mobile Host Internetworking, Mobile Computing, 1996.

[9] D. A. Maltz and P. Bhagwat, MSOKS: An Architecture for Transport Layer Mobility, Proc. IEEE Infocom'98, 1998.

[10] Alex Snoeren and Hari Balakrishnan, An End-to-End Approach to Host Mobility, Proc. ACM/IEEE MobiCom'99, 1999.

[11] H. J. Wang et al. ICEBERG: An Internet-core Network Architecture for Integrated Communications, IEEE Personal Communications, Special Issue on IP-based Mobile Telecommunication Networks, 2000.

[12] B. Raman, R. H. Katz and A. D. Joseph, Universal Inbox: Providing Extensible Personal Mobility and Service Mobility in an Integrated Communication Network, in Workshop on Mobile Computing Systems and Applications (WMCSA'00), 2000.

[13] G. Appenzeller et al. The Mobile People Architecture, ACM Mobile Computing and Communication Review, Vol. 1, No. 2, 1999.

[14] Yi Cui, K. Nahrstedt, D. Xu, Seamless User-level Handoff in Ubiquitous Multimedia Service Delivery, Multimedia Tools and Applications Journal, 2003.

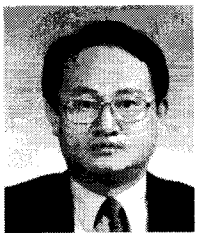
[15] Edwards, W. Keith, Core Jini, Prentice Hall PTR, 1999.

[16] MSN Messenger, http://www.msn.co.kr

[17] ICQ, http://www.icq.com

Tae-Uk Choi

He received the B.S degree in computer science & statistics from Dong-eui University, Pusan, Korea, in 1997, and M.S degree in computer science from Pusan National University, in 1999. He is currently pursuing the Ph.D degree in computer science at the same university. His research interests include Reliable Video Communications, Multimedia QoS, and Mobile & Ubiquitous Multimedia.



Ki-Dong Chung

He received the B.S degree from Seoul National University, Seoul, Koera, in 1973, and M.S and Ph.D degree in computer science from the same university, in 1975 and 1986. In 1990, he was a visiting scholar at Massachusetts Institute of Technology, Cambridge, MA. Since 1987, he has been a professor of computer science at Pusan National University, Pusan, Korea. His research interests include Operating System, Parallel Processing, VOD(Video On Demand), Multimedia Systems, and Wireless & Mobile Multimedia