Regular Paper

Lightweight FIPA Compliant Agent Platform on Java-enabled Mobile Phone for Ubiquitous Services

Satoshi Nishiyama,† Gen Hattori,†† Chihiro Ono†† and Hiroki Horiuchi††

We discuss the design issues for lightweight and FIPA compliant agent platform for Java-enabled mobile phones and describe the project of developing such agent platform. This platform changes Java-enabled mobile phones to ubiquitous terminals by providing high-level communication mechanism. Combined with services specific to mobile phones such as the location service, it can be used for various ubiquitous applications, such as providing ITS information to pedestrians. In this paper, we present seven design issues in three categories, discuss our approaches to these issues and describe the detailed design and implementation of an experimental platform. We also show the remaining issues through the performance comparison with LEAP, another lightweight agent platform.

1. Introduction

Until a few decades ago, a governmental monopoly had provided fixed and uniform telecommunication services all over Japan. Most users used the telephone terminals of same and single type at home. This terminal is called "kuro-denwa" where "kuro" means black and "denwa" means telephone as it is colored black. After the monopoly was privatized and the competition was introduced into the Japanese telecommunication market, the competitors have been trying to gain their market shares by offering various services and by rapidly providing advanced terminals, especially in the mobile phone market. Current mobile phone terminals are feather-weighed, may have large color LCD displays, cameras, GPS receivers and additional communication media such as IrDA and Bluetooth, and offer mail and Internet access. Also, many are Java-enabled to provide more sophisticated services. Considering the rapid lifecycle of terminals, within a few years, more than half of total subscribers, or almost a quarter of people in Japan, will use Java-enabled phone. However the mobile phone operators are still wondering what are the killer applications using Java. We believe that with the Java capability combined with the GPS location service, such Java-enabled mobile phone is suitable for providing ubiquitous services. For example, this terminal may be used to provide ITS (Intelligent Transportation Systems) related service such as route navigation or travel planning to pedestrians. However current Java-enabled phones only offer limited and operator dependent functionalities, especially for data communication. Thus, middleware that provides advanced communication mechanisms such as P2P and asynchronous communication and hides the differences of Java functionalities is required to promote Java-enabled mobile phones as ubiquitous terminals. Agent platforms based on FIPA (Foundation of Intelligent Physical Agents) standards [1] are ones of potential candidates for such middleware. Many research projects already developed FIPA compliant agent platforms on standard Java environment [2)~5)] and some are aiming to implement them on small nomadic devices [6),7)].

The final goal of our research is to design a lightweight FIPA compliant agent platform for Java-enabled mobile phone suitable for developing ubiquitous agent applications. In this paper, we discuss the design issues to implement FIPA compliant agent platform on Java-enabled mobile phone and present our solutions to the issues through an implementation, named KDDIAP (KDDI Agent Platform).

The rest of this paper is organized as follows. Firstly, we present the related activities as the background of this work. We identify the design issues for developing lightweight agent platform suitable for Java-enabled mobile phone in the third section. Section 4 discusses our approach to solve the issues and describe the detailed design of KDDIAP. An experimental implementation is evaluated and discussed by comparing another platform based on similar design con-

† YRP Ubiquitous Networking Laboratory
†† KDDI R&D Laboratories

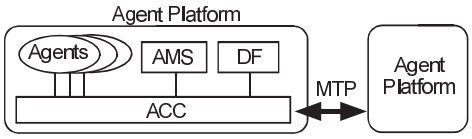

Fig. 1 FIPA agent platform model.

cept in Section 5.

2. Related Works

2.1 FIPA and FIPA Compliant Platforms

Providing agent platform as a "place" for agent is the first step to make the agent-based software popular. In the last decade, many researches concentrated on developing agent platforms [8),9)]. The next step is to increase the interoperability between agent platforms by standardization. FIPA [1)] is a non-profit organization to promote the interoperability of agent communication among heterogeneous agents. Currently FIPA produces a set of standards for agent communications, including the abstract architecture, agent communication protocols, agent management functions, agent message transports and typical applications of FIPA based platform.

More than 10 agent platforms have been developed based on FIPA standards. Typical implementations are JADE [2)], FIPA-OS [3)], Comtec Agent Platform [4)] and April Agent Platform [5)], all of them are publicly available. **Figure 1** shows the basic model of FIPA agent platform. ACC (Agent Communication Channel) acts as the message router in the platform. All the messages among agents are exchanged by ACC. AMS (Agent Management System) is for management of agent lifecycle. DF (Directory Facilitator) provides yellow page agent lookup services.

2.2 Java Environment for Nomadic Devices

JCP (Java Community Process), a SUN initiated consortia for developing Java standards, defines several Java specifications for Java 2: Java 2 Standard Edition (J2SE) is for typical client personal computers and other computers, Java 2 Enterprise Edition (J2EE) is for servers, and Java 2 Micro Edition (J2ME) for relatively small computers and devices. Furthermore JCP defines 2 configurations within J2ME: CLDC (Connected Limited Device Configuration) for devices with very limited resources and CDC (Connected Device Configurations) for relatively large (but still small) computers and devices.

The largest difference of J2ME/CLDC to J2SE is the lack of key mechanisms such as RMI and serialization, thread and event handling and dynamic loading. The lack of these mechanisms adds heavy burdens for implementing complicated and distributed programs on J2ME/CLDC. The Java virtual machine implementation for J2ME/CLDC provided by SUN is called as KVM. KVM runs on many PDAs in both PalmOS and WindowsCE. A profile for nomadic devices (PDAs and mobile phones) named MIDP (Mobile Information Device Profile) is defined. Recently mobile phone operators mostly in Japan, provide Java-enabled mobile phones to increase the flexibility of services (iAppli [10)], au EZPLUS [11)], J-Phone Java [12)]). KVM or equivalent virtual machines run on these phones. However iAppli is not MIDP compliant. EZPLUS and J-Phone Java add additional limitation for capabilities by omitting some optional parts of MIDP (e.g., datagram connection) and adding access control mechanism for resources within the phone for security reasons. Also all of them limit the resources available for the Java applications. For example, the maximum program size (i.e., the size of jar file) is 30 KB to 50 KB for iAppli and EZPLUS services. Only J-phone Java recently extended the size limitation to 256 KB. Though relaxing the limitation of program size may promote more advanced Java services, it also increases both the initial cost of the terminals as they require more memories and the communication cost for dowloading Java programs. Therefore, we think the operators will remain limiting the program size to some level, say 100 KB to 256 KB, until both of these costs are drastically reduced.

2.3 Agent Platforms for Lightweight and Nomadic Devices

Nomadic devices such as PDAs and mobile phones are suitable for providing anywhere services. Thus these nomadic devices are potential candidates as ubiquitous terminals. To promote agent based ubiquitous services, many researches have been developing agent platforms, which run agents on these nomadic devices.

Jumon [13)] is a commercial agent platform running both on J2SE and J2ME/CLDC + MIDP Environment. The J2ME version of Jumon is quite small. It requires only 3.5 KB heap memory for the platform. However, it does

not have the global identification mechanism for agents nor the directory mechanism for searching agent services. Therefore, Jumon is rather similar to remote object invocation middleware without directory services and is not suitable for providing general purpose multi-agent systems.

picoPlangent [14] is an extension of Plangent agent platform to the iAppli environment. It inherits the agent planning capability offered by Plangent. Most of the platform functionalities exist in the center side servers in the form of servlet to reduce the program size. Agents themselves do not exist in the iAppli environment. Only the components, i.e. classes which help agents to solve their goals, exist in the environment.

In addition to the non-FIPA agent platforms, two research platforms based on FIPA standards, LEAP [7] and Micro-FIPAOS [6], have been developed.

LEAP is developed by porting the container mechanism of JADE to the J2ME/CLDC + MIDP environment. It runs on the MIDP emulator and on PDAs both with PalmOS and with WindowsCE. Also there is a J2SE version of LEAP. Since RMI, used in JADE for intra-platform communication, is not available in the J2ME/CLDC environment, LEAP uses a dedicated protocol, named JITP, which transfers commands and responses directly over TCP/IP socket for communication. LEAP does not implement AMS and DF, the mandatory parts of the FIPA model, in the mobile side and depends these functionalities on the JADE platform running on another machines. So we may view that LEAP is a part of JADE platform running on nomadic devices.

Micro-FIPAOS is a lightweight version of FIPAOS ported by the University of Helsinki as a part of CRUMPET project [6]. It runs on J2SE and PersonalJava. As all the mandatory components in the FIPA model are implemented, creating a whole platform is possible if enough resources are available. However due to the resource limitation, partial platform without AMS and DF is recommended as the typical configuration for PDAs.

The design goals of LEAP and Micro-FIPAOS are running them on nomadic devices. However running these platforms on the actual Java-enabled mobile phone still needs further works. The program sizes of these platforms are over 300 KB, which exceed the size limitation by the current mobile phone operators. Also, the lack of mechanisms such as datagram connection has not been considered.

3. Design Issues of lightweight Agent Platform for Java-Enabled Mobile Phones

Currently, we think FIPA standards have two advantages to other agent designes: rich and well-defined functionalities, and good interoperability. The communication model of FIPA standards is based on well-defined theory. Its communication language, or abbribated as ACL, provides complex but rich capabilities for expressing the intention of agents. It is also akin to KQML [15], another popular agent communication language. For the interoperability, nearly ten experimental platforms are already developed and connected to each other worldwide for interoperability testing.

Therefore we decided to design the lightweight agent platform for Java-enabled phone based on FIPA standards, as in the cases of LEAP and Micro-FIPAOS. However, The problem is that FIPA compliant agent platforms tend to be large programs. Therefore, reducing the program size is one of the design issues and is already mentioned in LEAP and Micro-FIPAOS. In addition, when the platform runs on Java-enabled phones, it should also satisfy requirements specific to mobile phone, such as their unreliable wireless communication and about the start up speed.

Therefore, we classify the design issues for lightweight agent platform for Java-enabled phone as follows:

(1) Issues from requirements specific to mobile phone communication.
  (1-1) Unreliable wireless communication:
    Mobile phone communication tends to be unstable by fading or other reasons.
  (1-2) Local management capabilities:
    Users may wish to interact with agent software even when they are out of coverage areas such as in the underground.
  (1-3) Start up speed:
    Platforms should start quickly so that the users can use agent services easily anywhere and anytime.
(2) Issues on Java functionalities and on limitation of program size:
  (2-1) Java capabilities:
    As discussed in the previous section,

Table 1 Design issues on lightweight agent platform for java-enabled mobile phones and proposed solution.

Issues	Issues in Detail	Proposed Solutions (Section 4)
(1) Issues from requirements specific to mobile phone communication	(1-1) Instability of wireless communication	Implement buffer-based reliable communication mechanism
	(1-2) Needs for anytime agent management	Implement minimum local management functionalities and synchronization mechanism
	(1-3) Quick start up	Implement warm start up mechanism
(2) Issues on Java functionalities and on limitation of program size	(2-1) J2ME/CLDC+ MIDP functionalities	Design platform newly and dedicated for J2ME /CLDC+MIDP
	(2-2) Program size limitation	Leave minimum functionalities on mobile phone
(3) Management issues	(3-1) Scalability	Design to run center multiple sub-platforms parallelly using shared database
	(3-2) SPAM protection	Implement message filtering mechanism based on blacklist

J2ME/CLDC+MIDP lacks some important functionalities for implementing complicated communication programs. Additional restrictions by the mobile phone operators, especially the lack of datagram connection, may also have strong impact on the design. For example, Java programs on mobile phones can access to servers using HTTP protocol. However there is no way of establishing TCP/IP connection from servers to the mobile phones, which makes it difficult to sending information from server to mobile phones effeciently (we call this as "downlink problem").

(2-2) Program size:
For the size limitation, the current maximum program size is 30 KB to 50 KB (and 256 KB for J-phone Java), which is smaller than that of the existing platforms. Therefore based on the discussion in the previous section, the program size should be under 256 KB.

(3) Issues from management aspects:

(3-1) Scalability:
Mobile phone operators have order of million subscribers. Therefore scalability is one of key issues.

(3-2) Access control:
Sending SPAM mails to the mobile phones is one of hot social problems in Japan. Therefore, access control mechanism is needed.

We summarize the design issues in the left two columns of **Table 1**.

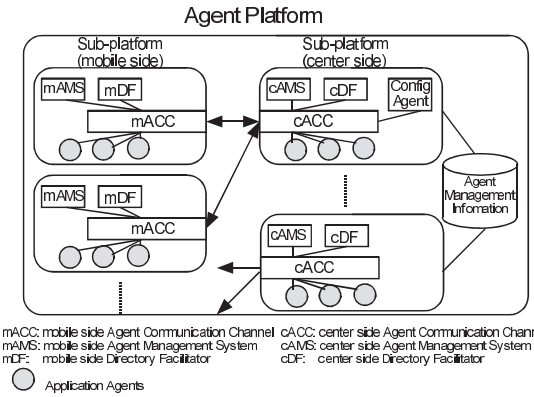

Fig. 2 Overview of architecture of KDDIAP.

4. Design of KDDIAP and Its Implementation

This section describes the detailed design of KDDIAP, a lightweight agent platform for Java-enabled mobile phone.

4.1 Overall Architecture

We choose a center-oriented type of platform as the overall design from the viewpoint of management (i.e., reliability, security and accounting). Each mobile phone or center server maintains a fragment of platform (we call it sub-platform) and the collection of sub-platforms are viewed logically as a single FIPA compliant platform (see **Fig. 2**).

4.2 Our Approaches to Design Issues

The followings show our approaches to the design issues discussed in the previous section, which are also summarized in the right column of Table 1.

(1) Solution to requirements specific to mobile phone communication

(1-1) Unreliable wireless communication:
To handle the instability of communication and to avoid the loss of messages, we place the message buffers both in the mobile phone sub-platforms and the center side sub-platforms. The buffered messages will be retransmitted when the communication becomes available later.

(1-2) Local management capabilities:
We design that the mobile sub-platform implements all the mandatory components (e.g., ACC, AMS and DF) defined in FIPA standards with minimum functionalities enough for communication and for local management. Therefore, even if a mobile phone is unreachable to a center sub-platform, the mobile sub-platform allows local management operations on the locally existing agents in the mobile phone to its user by using the local AMS. After the connection is reestablished, the new status will be propagated to the ones stored in the center sub-platform.

(1-3) Start up speed:
Generally, starting platform needs creating the above mandatory components, which may be time consuming processing. We design our platform to support the full lifecycle of an agent, defined by FIPA standards, which allows agents be "suspended" for mobility and other reasons such as reducing memory usage. To shorten the start up time, when the mobile sub-platform is going down, instead of destroying these components in the sub-platform, it just makes them suspend. When the sub-platform starts again, it just wakes these components up. Thus the start up time will be much shorter.

(2) Solution to limitation on Java functionalities and program size:

(2-1) Java capabilities:
Event Handling mechanism: typically, each agent is implemented as a thread of Java. J2ME/CLDC + MIDP does not provide the event handling mechanism between threads. So we choose the classical polling mechanism for intra-sub-platform communication in the mobile sub-platform.
RMI and Serialization: as J2ME/CLDC + MIDP does not support RMI, we introduce dedicated mechanisms for data serialization and for agent mobility. We will discuss the detail in Subsection 4.4.
Down link problem due to the lack of datagram connection: to avoid the downlink problem, we designed that KDDIAP can use asynmetric MTPs where messages from and to the mobile and center sub-platforms are sent separately in the uplink MTP and the downlink MTP. The detailed solution will be discussed in Subsection 4.5.

(2-2) Program size:
To accommodate the program size into 100 KB to 256 KB, as we discussed in (1-2), we designed that all the components in the mobile side have minimum functionalities enough for communication and for local management on the mobile phone. The difference of our approach to LEAP is that LEAP has only ACC in the mobile side and fully depends on the center, while KDDIAP is somehow independent to the center side so that it can handle off-line management operations. The detailed mapping of functionalities is described in the following subsection.
We also give up providing kinds of libraries, which may be useful for agent programmers, to aviod that all the libraries are included in the platform. For example,libraries for interaction protocols, generic ontology mechanisms and behaviours are not implemented.
Furthermore, to cope with the complex ACL syntax, KDDIAP does not support the XML representation for ACL and the envelope. We also limit the syntax of the semantic language to SL0.

(3) Issues from management aspects:

(3-1) Scalability:
To realize the scalability, our platform can invoke multiple instances of center side sub-platforms. The mandatory components in these sub-platforms share the management information using a relational database. So logically, these sub-platforms are viewed as a single sub-platform from mobile side sub-

platforms and other external platforms. Each mobile sub-platform is connected to one of center sub-platforms. When a mobile sub-platform starts, it firstly asks a special agent, named "configuration agent", for a center sub-platform to be connected by using the default sub-platform. The configuration agent selects one based on the current loads of the center sub-platforms and returns the name of the assigned sub-platform to the requester. Then the mobile sub-platform reconnects to the assigned sub-platform to start the normal message communication.

(3-2) Access control:
To protect user from SPAM messages, each user can register his/her blacklist to KDDIAP. Messages from the agents who are named in the list or whose owners are specified in it are filtered out by the center side ACC.

4.3 Mapping of Functionalities between Mobile Phone Side and Center Side

We divided the mandatory functionalities into the mobile phone sub-platform and the center sub-platform as follows:

- ACC: We omit the generic routing mechanism from the mobile ACC (mACC). mACC delivers messages only to the agents in the local sub-platform. Other messages are sent to the center ACC (cACC) to which it is currently connected. cACC has full routing mechanism, including routings between cACCs and to other external platforms.
- AMS: Normally, all the agent management information is controlled by the center AMSs (cAMSs) and the mobile phone AMS (mAMS) acts as a proxy for accepting local operations and for cashing the current status of local agents. If the mAMS can not communicate with the peer cAMS, it locally executes the operations. The results will be propagated to the cAMS when the communication is reestablished.
- DF: The mobile DF (mDF) is also just a cash proxy to the center DF (cDF) in order to reduce the amount of communication. Full yellow page functionalities are implemented in cDFs.

4.4 Mobility and Serialization

As for mobility, the Java environments for mobile sub-platform and for center sub-platform differ. Therefore, we use "weak mobility", where the agent program of the moving agent for the environment of the destination already exists at the destination and only the internal state of the agent is transferred and restored there. Sending internal states needs serialization mechanism as RMI is not available. Therefore, we provide dedicated serialize functions for basic data types (e.g., integer, floating numbers and string). It is the agent programmer's responsibility to provide serialize(), deserialize() interfaces for the agent running on KDDIAP. Suspend(), which stores the internal state of the agent into the memory, tells to the local AMS and stops the execution of the agent, and restart(), which restores the written internal state and restarts the execution of the agent, are also required for the agent. We also provides a template for creating agent programs which comply with the above interfaces. Finally, the serialized internal state of an agent is base64 encoded and sent as the content of ACL between the AMSs.

4.5 Downlink Problem

As for the MTP between the mobile side and the center side, LEAP assumes that datagram connection is available in the mobile side, but that the server socket is not supported (MIDP limitation). Therefore LEAP places a proxy program named Mediator in a center side server to accept message requests from the center side [16]. Messages are exchanged between the mobile side and Mediator bidirectionally through the datagram connection. Current implementation of KDDIAP, which we discuss in the following subsection, runs only on PDAs and MIDP emulator. Howevwe, when we are going to implement KDDIAP on actual Java-enabled mobile phones as the fure work, as all of iAppli, EZPLUS and J-phone Java do not provide datagram connection, this approach is not appricable.

For the uplink, sending message by HTTP protocol, which is a standard MTP for FIPA platforms, is possible as they offer HTTP client interface at the mobile phone. But for the downlink, as they offer neither HTTP server interface nor the datagram connection at the mobile phone, HTTP protocol is not appricable. Therefore we designed that KDDIAP can handle unidirectional message transports between

Fig. 3 Screenshots of KDDIAP on MIDP emulator and iPAQ PDA.

the center and mobile sub-platforms to separate the uplink messages and the downlink messages in addition to the normal bidirectional message transports. According to the current Java services on mobile phone, we think that the short messaging service is suitable for the future implementation on Java-enabled phones, where the expected communication delay is about a second. Polling to the center side using HTTP is not desirable due to the polling delay and communication cost.

4.6 Prototype Implementation

We implemented a prototype of KDDIAP. The center sub-platforms run under J2SE environment on Linux or Windows operating systems. The mobile phone sub-platform was implemented under J2ME/CLDC + MIDP environment, running on PalmOS and MIDP emulator over Windows PCs. **Figure 3** shows screenshots of KDDIAP running on MIDP emulator and iPAQ PDA (see Section 5). As for the DBMS which stores agent management information for ACCs and AMSs, KDDIAP accesses it through JDBC for portability. Currently Oracle and Postgress are already tested for the database server.

For the asymmetric message transport between mobile sub-platforms and the center sub-platforms, we currently run the KHTTP server [17], a small HTTP server running on J2ME environment, on both of the mobile sub-platform and the center sub-platform and use the HTTP protocol for both of uplink and downlink. Another version of message transport, which uses short messaging service as the downlink, is currently under development for actual mobile phones. We also use the HTTP based message transport package [18] for external communication to the other FIPA compliant platforms.

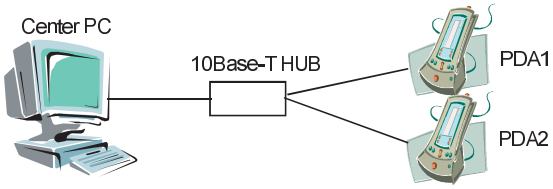

Fig. 4 Evaluation environment.

Table 2 Detailed hardware and software specifications for evaluation.

	Center	PDA1/PDA2
CPU	Pen-III 750 MHz	ARM 206 MHz
RAM	256 MB	64 MB
OS	Linux	SavaJe
Java Env.	J2SE	SavaJe (J2SE)

5. Performance Evaluation and Discussions

We evaluated the performance of the prototype and compared with LEAP to show the difference caused by the available Java functionalities. We also discuss about the scalability of KDDIAP in this section.

5.1 Evaluation Environment

In this evaluation, to run KDDIAP and LEAP on the same mobile device environment, we ported the mobile sub-platform of KDDIAP into SavaJe [19], J2SE compliant Java OS running on iPAQ PDAs and compared to the J2SE version of LEAP on iPAQ (plus JADE in the center side) for performance evaluation. Note that the ported version of KDDIAP is not optimized for J2SE environment. It just uses APIs available for J2ME. **Figure 4** shows the configuration of the evaluation. Two iPAQ PDAs and a desktop PC are connected by LAN (10base-T). For KDDIAP, a Postgress server runs on the center PC. **Table 2** shows the detailed hardware and the software for these machines. **Table 3** shows the functionalities of KDDIAP compared to those of LEAP.

5.2 Evaluation Results

5.2.1 Program Size

Firstly, we measured the program size (the size of Jar file) of the mobile sub-platform of KDDIAP and that of LEAP. **Table 4** and **Table 5** show the sizes of Jar files for J2SE and J2ME environments, respectively. As the J2SE version of LEAP contains functionalities that are not included in KDDIAP, i.e., IIOP support, interaction protocol handler, ontology support and wrapper support, we removed the codes for these functionalities from the Jar file

Table 3 Comparison of functionalities of KDDIAP with LEAP.

	KDDIAP	LEAP (on J2ME)
components		
Center side AMS	One or more per platform	one per platform
Mobile side AMS	Yes, act as proxy (allows local management operation)	None
Center side DF	One or more per platform	one more more per platform
Mobile side DF	Yes, act as proxy (reduce communication traffic)	None
Center side ACC	One or more per platform	one per platform
Mobile side ACC	Yes (limited routing capability)	Yes (limited routing capability)
Agent communication		
Content language syntax	Semantic Language (SL) 0	SL0 $\sim$ SL2
Support of Interaction Protocol	None	Yes
ACL representation	String	String, XML
Envelope representation	String	String, XML
Supported MTPs for Mobile Communication	Both of bidirectional and unidirectional MTPs. Current version uses HTTP for both uplink and downlink MTPs. Future version will use short messaging services for downlink MTP.	Bidirectional MTP only. JITP is currently supported.
Other features		
Mobility Support	Yes	None
Secuirty	Access Control by blacklist and whitelist	None

Table 4 Program size of mobile side in J2SE environment (in kilobytes).

KDDIAP (full)	KDDIAP (without GUI)	LEAP (full)	LEAP (without GUI and above)
209	185	1290	501

Table 5 Program size of mobile side in J2ME environment (in kilobytes).

KDDIAP (full)	LEAP (full)
220	374

before the measurement.

Also, as the GUI of J2SE LEAP is richer than that of KDDIAP, we removed the GUI codes from both platforms for comparison. The program sizes shown in Table 5 represent the sizes of Jar files of KDDIAP and LEAP for the J2ME environment. The program size of KDDIAP is about 200 KB on J2SE, or 220 KB on J2ME, almost one third or half of that of LEAP, respectively, and meets the requirement (2-2) in Subsection 4.2. The size difference between the J2ME version and the J2SE version of LEAP may due to the supported functionalities. The size difference between the LEAP and KDDIAP in J2ME version mainly comes from the following factors: (note: (+) represents factors which make KDDIAP larger. (−) represents factors which make KDDIAP smaller.)

(+) KDDIAP has mAMS and mDF, while LEAP does not,
(−) JITP is relatively large to the HTTP based MTP of KDDIAP, as it contains the interface mechanism to IIOP and the data compression mechanism,
(−) Handlers of interaction protocols and ontologies of LEAP are large, and
(−) while LEAP provides typical behaviour libraries, KDDIAP does not.

The last two factors mean that agent programmers need more codes to develope agents in KDDIAP than in LEAP. To reduce the above programming overhead, KDDIAP provides a program template for typical agents.

We also tested Retroguard [20], a compaction tool of Jar file. But we only obtain limited compression gain of about 5 percentages. Therefore in order to reduce the program size further, we needs additional design refinements. Currently parsers for ACL and envelope in each mandatory component (i.e., ACC, AMS and DF) are

Table 6 Start up time (in milliseconds).

KDDIAP		LEAP
Second time and later	First time	with JADE
2,898	17,792	1,615

relatively large compared to the code for implementing actual functionalities for the component, especially in DF. Therefore, limiting complex syntax and functionalities for these components, especially for DF, may be the next refinement area.

5.2.2 Start Up Time of Mobile Sub-platform

Secondly, we measured the start up time discussed in the requirement (1-3) of Subsection 4.2. **Table 6** shows the comparison of KDDIAP and LEAP with JADE in the center (we just refer this combination as LEAP in the later section for simplicity). At the first time, the mobile sub-platform of KDDIAP creates mACC, mAMS and mDF and registers them (cold start). At the second time or the later, it only wakes the "suspended" agents up from the local storage, which will takes much shorter time (hot start). Therefore, KDDIAP usually starts within 3 seconds. Considering that LEAP does not create AMS and DF in the mobile side, we believe that the hot start mechanism of KDDIAP is effective.

5.2.3 Performance of Message Communication

We think that in the actual mobile phone environment, as the communication delay of short messaging service is around a second, the performance bottlneck may exist in the communication but not in the processing delay by the limited computing power of the mobile phones. To clarify this point, we evaluate the performance of message communication. A test agent sends an Inform message that only contains the header part (about 320 bytes) to the peer and the latter returns the same message to the originator. **Table 7** shows the measured round trip delay in the J2SE environment for the message exchange in the following 5 cases:

a) Center local: Both the test agent and the peer in the center
b) PDA local: Both the test agent and the peer in PDA1
c) PDA → Center: The test agent in PDA1 and the peer in the center
d) Center → PDA: The test agent in the center and the peer in PDA1
e) PDA1 → PDA2: The test agent in PDA1 and the peer in PDA2

Table 7 Round trip delay for message exchange (in milliseconds).

	KDDIAP	J2SE LEAP with JADE
a) Center local	54	2
b) PDA local	416	18
c) PDA → Center	947	85
d) Center → PDA	948	104
e) PDA1 → PDA2	2,032	69

The difference of round trip delay between KDDIAP and LEAP mainly comes from the methods of communication. KDDIAP uses RMI within the center sub-platforms, the polling based mechanism within the mobile sub-platform due to the capability of J2ME/CLDC, and HTTP protocol by KHTTP server between the mobile and the center sub-platforms. On the other hand, LEAP directly uses socket for all the message communication. Therefore, the difference in case a) mainly comes from the performance of RMI and that of the socket. In case b), KDDIAP needs 4 times of polling. Currently the polling interval is set to 100 milliseconds, which means the average delay is 50 milliseconds. Therefore about 200 milliseconds are consumed by the polling. For case c) and case d), additional overhead by the KHTTP servers in both sub-platforms (2 times of polling in the mobile side and a thread creation for each sub-platform) exists. Case e) takes almost twice time of case c) or d), since all the messages are relayed by the cACC.

The performance of KDDIAP will be improved if we use shorter polling interval. However, as explained, when we use KDDIAP in the actual mobile phone, short messaging services will be used for downlink of the message transport, where the delay will be around a second. So we think that the trade-off of performance and the processing overhead should be customized for each environment, considering the delay of its short message service.

5.2.4 Performance of Agent Mobility

As KDDIAP supports agent mobility, we evaluated the performance by comparing to the J2SE version of LEAP. A test agent on a sub-platform requests to move to another sub-platform and then the same agent requests to return to the originating sub-platform. **Table 8** shows the measured round trip delay for the following 3 cases:

a) PDA → Center : Starts from PDA1, moves to the center and returns

Table 8 Round trip delay of agent mobility (in milliseconds).

	KDDIAP	J2SE LEAP with JADE
a) PDA → Center	4,308	2,891
b) Center → PDA	3,879	2,933
c) PDA1 → PDA2	9,459	9,079

b) Center → PDA : Starts from the center, moves to PDA1 and returns
c) PDA1 → PDA2 : Starts from PDA1, moves to PDA2 and returns

The number of messages needed for mobility in KDDIAP is larger than that in LEAP as an agent in LEAP can directly access its AMS, while that in KDDIAP accesses the center AMS via its mobile AMS. Therefore, currently KDDIAP is slower than LEAP due to the difference of number of messages needed for mobility and the polling mechanism used in the mobile sub-platform of KDDIAP. The performance of agent mobility can also be improved by shortening the polling interval. However, as discussed in Subsection 4.5, the downlink delay would still remain as the dominant factor for the performance in the actual mobile phone environment.

5.3 Scalability of KDDIAP

Current version of KDDIAP is written in Java, which is known as the heavy CPU comsuming language. KDDI is designed to be able to run more than one AMS, DF and ACC over multiple servers in a single FIPA platform to improve its scalability. However, all the agent management information is stored in a single relational database system and shared by these components. Therefore, the performance of the RDBMS may become the bottleneck.

According to the result by the Transaction Processing Performance Council [21], the best performance by a single non-cluster DB server is over 800 K TPmC for on-line transaction processing benchmark test(TPC-C). The processing overhead to the database for message routing in KDDIAP seems lighter than those defined in TPC-C. However, if we assume that the overhead of the former is equvalent to the latter, a high-performance RDBMS server can offer processing power needed for routing 800 K messages per minutes, or 16 K messages per second. Even if one tenth of the mobile phone subscribers actively uses agent applications in their Java-enabled phone at the same time and each agent application sends one message per second, the performance means that a single KDDIAP platform can accomodate 1.6 million subscribers. By using clustered RDBMS servers, we can further increase the muximum number of subscribers. Therefore, we think the scalability of KDDIAP is practically achieved.

6. Conclusion

In this paper, we discussed the design issues for FIPA compliant agent platforms for mobile phones and presented our solutions through the prototype development of KDDIAP. KDDIAP has features suitable for running on mobile phones, e.g. small implementation dedicated to J2ME/CLDC+ MIDP environment, scalability, reliable communication mechanism, offline management functions and warm startup mechanism. We evaluated the prototype by comparing to LEAP, a similar lightweight agent platform.

We are currently developing another message transport that uses the short messaging service as the downlink to run KDDIAP on the real mobile phone. Our future plan also includes further compaction of KDDIAP and an extension of mACC to support ad-hoc agent communication through Bluetooth and IrDA equipped in mobile phones.

Acknowledgments The authors wish to thank to Dr. Tohru Asami, President & CEO of KDDI R&D Laboratories, Inc. for his continous support for this study.

References

1) FIPA Home Page. http://www.fipa.org
2) Bellifemine, F., Poggi, A. and Rimassa, G.: JADE — A FIPA-compliant agent framework, *Proc. PAAM'99*, London, pp.97–108, The Practical Application Company Ltd. (April 1999).
3) Poslad, S., Buckle, P. and Hadingham, R.G.: The FIPA-OS agent platform: Open Source for Open Standards, *Proc. PAAM 2000*, Manchester UK, pp.355–368, The Practical Application Company Ltd. (April 2000).
4) COMTEC Agent Platform. http://ias.comtec.co.jp/ap/
5) April Agent Platform. http://www.nar.fujitsulabs.com/aap/
6) Poslad, S., Laamanen, H., Malaka, R., Nick, A., Buckle, P. and Zipf, A.: CRUMPET: Creation of User-friendly Mobile Services Personalised for Tourism, *Proc. 3G 2001*, London (March 2001). http://conferences.iee.org.uk/3G2001/
7) Bergenti, F. and Poggi, A.: LEAP: a FIPA Platform for Handheld and Mobile Devices,

Proc. ATAL (2001).
8) Aglet Agent Platform. http://www.aglets.org
9) Ohsuga, A., Nagai, Y., Irie, Y., Hattori, M. and Honiden, S.: Plangent: An Approach To Making Mobile Agents Intelligent, *IEEE Internet Computing*, Vol.1, No.4, pp.50–57 (1997).
10) I-mode Java Service. http://www.nttdocomo.com/top.html
11) au EZPLUS (in Japanese). http://www.au.kddi.com
12) J-Phone Java Service (in Japanese). http://www.j-phone.com/
13) http://www.e-jumon.com/
14) http://www2.toshiba.co.jp/plangent/
15) UMBC KQML Web. http://www.cs.umbc.edu/kqml/
16) Berger, M., Rusitschka, S., Toropov, D., Watzke, M. and Schlichte, M.: Porting Distributed Agent-Middleware to Small Mobile Devices, *Proc. Workshop on Ubiquitous Agents on Embedded, Wearable and Mobile devices* (2002).
17) KHTTP Server. http://khttp.enhydra.org/index.html
18) Java Agent Message Router. http://liawww.epfl.ch/JAMR/
19) SavaJe Operating System. http://www.savaje.com/
20) Retroguard. http://www.retrologic.com/
21) Transaction Processing Performance Council. http://www.tpc.org/

(Received May 9, 2003)
(Accepted December 2, 2003)

Satoshi Nishiyama received his B.E degree from the Department of Electrical Engineering, the University of Tokyo in 1984 and joined Kokusai Denshin Denwa Co. Ltd. (now KDDI). He received his M.A. degree from the Department of Computer Science, the University of Texas at Austin in 1991. Currently, he is the section head of Ubiquitous Networking Protocol Section, YRP Ubiquitous Networking Laboratory. His research interest includes database, network management, intelligent transportation systems, agent systems and ubiquitous network. He received the IEICE Research Promotion Award in 1993 and the Best Paper Award of the National Convention of IPSJ in 2001, respectively.

Gen Hattori received the B.E. and M.E. degrees of Electrical and Electronics Engineering from Kobe University, Japan, in 1996 and 1998 respectively. Since joining KDD R&D Laboratories Inc. in 1998, he has been working on network management systems, intelligent transportation systems and software mobile agent systems. He is currently a research engineer of Text Information Processing Lab. in KDDI R&D Laboratories Inc.

Chihiro Ono received the B.E. degree of Electrical Engineering, the M.S. degree of Computer Science from Keio University, Japan, in 1992 and 1994 respectively. Since joining KDD R&D Laboratories Inc. in 1994, he has been working on network management systems, database systems, and agent systems for electronic commerce. From 1999 to 2000, he was a visiting researcher at Stanford University. He is currently a research engineer of Text Information Processing Lab. in KDDI R&D Laboratories Inc. He received Best Paper Award for Young Researchers of the National Convention of IPSJ in 1996.

Hiroki Horiuchi received the B.E., M.E. and Dr. of Engineering from Nagoya University, Japan, in 1983, 1985 and 2000 respectively. Since joining KDD R&D Laboratories Inc. in 1985, he has been working on network architecture, formal description techniques for communication protocol, network management systems, distributed processing technologies and intelligent transportation systems. He is currently a senior manager of Ubiquitous Network Lab. in KDDI R&D Laboratories Inc. He received Young Engnieers Award of IEICE in 1992 and the Excellence Award in 52nd and 58th National Convention of IPSJ in 1996 and 2000, respectively.