

SlimWare: Dynamic Interfacing Method for Lightweight Middleware in Mobile Computing

Tae-Gyu Lee*, Gi-Soo Chung*

*Korea Institute of Industrial Technology, KITECH B-108-1, 143 Hanggaulro, Sangnok-gu, Ansan-si, Gyeonggi-do, 426-910, Korea

tglee@kitech.re.kr, gschung@kitech.re.kr

Abstract—we propose the dynamics and mobility interface method for light-weighting of middleware in mobile computing applications. The previous middleware did not properly support the lightweight and mobility of mobile user terminals. Therefore this paper presents the dynamic configuration method of mobile middleware components and the mobility interface strategy. The proposed middleware system called of “SlimWare” provides the component dynamic logic and the interface design.

Keywords— Interface design, Middleware, Mobile computing, Context awareness, Disconnected operation

I. INTRODUCTION

The various applications of mobile computing such as mobile cloud computing, ubiquitous computing, wireless sensor networking, and etc. based on a distributed system has highlighted the importance of middleware's flexibility and portability advantages.

The definition of middleware is variously expressed as follows. By Toni [1] and Kurt [2], “Middleware is the software that assists an application to interact or communicate with other applications, networks, hardware, and/or operating systems. The term middleware also refers to the software layer between the operating system—including the basic communication protocols—and the distributed applications that interact via the network. In a distributed computing system, middleware is defined as the software layer that lies between the operating system and the applications on each site of the system. By Linthicum [13], “Middleware is an enabling layer of software that resides between the application program and the networked layer of heterogeneous platforms and protocols. It decouples applications from any dependencies on the plumbing layer that consists of heterogeneous operating systems, hardware platforms and communication protocols”.

Dynamic changes in system configuration and operating context at runtime will be inherent characteristics of future computing environments. Middleware design and development must readily meet these challenges [2]. Recent trends of middleware are faced to challenges about wireless and mobile computing environments [3]. It is a major challenge to solve all problems of mobile distributed systems. This is true due to the high degree of dynamism in mobile environments. Especially, the strengths of this Component Based or Reflective middleware are that it is configurable and re-configurable. These components are selected either at

build-time or at run-time. Re-configurability can be accomplished at run-time, which offers a lot of flexibility to meet the needs of a large number of applications [1]. But current middleware platforms like CORBA cannot successfully run in such an environment. Hence, there is an urgent need for new solutions that support particular application requirements such as dynamic reconfiguration, context-awareness, and adaptation.

The focused points of this paper are lightweight and mobility interface strategies for mobile computing defined as follows: The lightweight is the building process which constructs the light running process on a mobile terminal. The mobility interface is a dynamic configuration interface which makes up the user interfaces dynamically according to the context of mobile users.

The contents of this paper will be described as follows. Section 2 will write about the mobile middleware and the related research works. Section 3 will illustrate the proposed mobility interface design for the lightweight mobile computing. Section 4 presents an example of the mobility interface design and analyses the lightweight (comparisons of transition ratio of the context module according to the degree of lightweight) and mobility interface (comparisons of fast interface replacement methods). Finally, Section 5 concludes this paper.

II. RELATED WORKS

Among the previous middleware, we consider *Reflective Middleware* and *Event-Based Middleware* to meet our research goals of the lightweight and the mobility interface. *Reflective or Component-Based Middleware* contributes to the flexibility of application systems. *Event-Based Middleware* has the advantages of the lightweight in the computing services.

A. Reflective Middleware

The reflection technique was initially used in the field of programming languages to support the design of more open and extensible languages [4]. Reflection is also applied in other fields including operating systems [5] and more recently distributed systems [6]. The principle of reflection enables a program to access, reason about and change its own behaviour. Smith [7] defined the concept of reflection in the following quote:

"In as much as a computational process can be constructed to reason about an external world in virtue of comprising an ingredient process (interpreter) formally manipulating representations of that world, so too a computational process could be made to reason about itself in virtue of comprising an ingredient process (interpreter) formally manipulating representations of its own operations and structures".

A reflective system consists of two levels referred to as meta-level and base-level. The former performs computation on the objects residing in the lower levels. The latter performs computation on the application domain entities. The reflection approach supports the inspection and adaptation of the underlying implementation (the base-level) at run time. A reflective system provides a meta-object protocol (meta-interface) to define the services available at the meta-level. The meta-level can be accessed via a concept of reification. Reification means exposing some hidden aspect of the internal representation and hence they can be accessed by the application (the base-level). The implementation openness offers a straightforward mechanism to insert some behaviour to monitor or alter the internal behaviour of the platform. This enables the application to be in charge of inspecting and adapting the middleware behaviour based on its own needs. Thus, a lightweight middleware with a minimal set of functionality is achieved to run on mobile systems.

The main motivation of this approach is to make the middleware more adaptable to its environment and better able to cope with changes. Examples of middleware systems that adopted the concept of reflection are OpenCorba [8], OpenORB [9], DynamicTAO [10], FlexiNet [11], and Globe [12].

B. Event-Based Middleware

Invocation-based middleware systems such as CORBA or Java RMI are useful abstractions for building distributed systems. The communication model for these platforms is based on a request/reply pattern: an object remains passive until a principle performs an operation on it. This kind of model is adequate for a local area network (LAN) with a small number of clients and servers, but it does not scale well to large networks like the Internet. The main reason is that the request/reply model only supports one-to-one communication and imposes a tight coupling between the involved participants because of the synchronous paradigm. This model is also unsuitable for unreliable and dynamic environment. The event-based communication paradigm [14] is a possible alternative for dealing with large-scale systems. Event notification is the basic communication paradigm that is used by event-based middleware systems. Events contain data that describes a request or message. They are propagated from the sending components to the receiver components. In order to receive events, clients (subscribers) have to express (subscribe) their interest in receiving particular events. Once clients have subscribed, servers (publishers) publish events, which will be sent to all interested subscribers. This paradigm hence offers a decoupled, many-to-many communication model between clients and servers. Asynchronous notification of events is also supported naturally. There are several examples of middleware based on the event-based systems, but not limited

to, Hermes [15], CEA [14], STEAM [16], JEDI [17] and ToPSS [18].

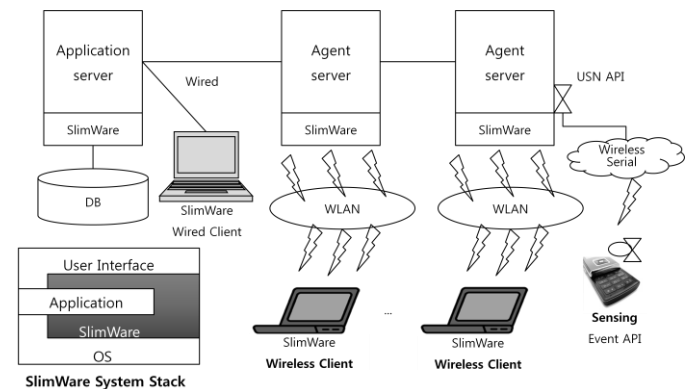


Figure 1. Lightweight Middleware System Architecture: SlimWare

C. Problem Issues and Contributions

Reflective or Component-Based Middleware contributes to the flexibility of application systems, but is vulnerable to the lightweight. *Event-Based Middleware* has the advantages of the lightweight, but are inadequate to the computing services on a large-scale network.

Existing mobile middleware did not consider the vulnerability of the terminal display size to build the mobile user interface. Thus, it has increased the redundant access and the click counts to the mobile user interfaces. In addition, the approach depth of interface was increased.

This study is to implement the slim middleware that realizes the flexibility of the component-based middleware and the lightweight of the event-based middleware at the same time. It is also to realize the simplicity and ease-of-use by building a dynamic reasonable interface considering the user's situation in order to build an interface that facilitates mobile users.

Figure 1 show the proposed lightweight middleware system Architecture called "SlimWare" in a mobile computing service infrastructure. The architecture is based on the different networking interaction such as the wired networks, the wireless LANs, wireless global networks, Ad-hoc networks, and so on. The communication infrastructure dynamically changes its connection bandwidth and safety depending on the location of mobile users. Therefore, the proposed mobile middleware should support the adaptive user interface and running context modules by considering the dynamics.

III. LIGHTWEIGHT MOBILE MIDDLEWARE AND MOBILITY INTERFACE DESIGN

The dynamic context switching is required for the lightweighting of mobile middleware on user terminals. Especially, the mobile user terminals such as smart phone or pad should load up the proper middleware components on its memory depending on the location of mobile users.

A. Lightweight Middleware Components and Operations

Middleware components and application execution processes can be dynamically operated in order to overcome the constraints of the client with the limited network and mobile terminal resources. Dynamic loading configuration of the execution processes on memory is required to ensure the lightweight of the mobile application computing.

The dynamic loading mechanism can be divided into the loading process and the unloading process. For the dynamic loading policy, the first policy preferentially loads the high-priority context module that needs in context environments of the mobile users. The second policy preferentially unloads the low-priority context module that less likely to use in terminal environments of the mobile users.

TABLE 1. CONTEXT-MODULE PARAMETERS

CX_T_ID	CXT_PAT	UR_CLASSES	LA_CLASSES	REF_CNT	LD_FLAG	IF_ID_X	CM_ID_X
CX_Txxx	CPxxx	Sxxx/Lxxx/Mxxx	Office/Ext/Plant mode	xxx (High/Low)	On/Off	Mxxx/Ixxx/Wxxx	CX_Txxx

As shown in Table 1, the application-context modules of mobile devices can be dynamically relocated depending on the user's business position("UR_CLASS"), the user's location area("LA_CLASS"), the interface reference count("REF_CNT"), the memory status of load on/off("LD_FLAG"), the interface index("IF_IDX"), and etc. Moreover, these dynamic context-module configurations can be maximized for the utilization and convenience by defining a set of user interface patterns ("CXT_PAT").

Figure 3 describes the context module switching logic for establishing a lightweight mobile execution process on a mobile terminal. When the mobile users move, it dynamically loads the execution context module ("CM_IDX") on the basis of the context parameters into memory by calling the context-id ("CXT_ID"). Also, depending on the movement situation of mobile user, it executes an unload process which removes the non-executable module that no longer be used among the existing context modules loaded on memory.

B. Mobility Interface Design

It should support the appropriate user interface to provide the optimal application services and interfaces for mobile users on the moving path and environments. To do this, it requires defining interface parameters of the user-specific ("UR_ID") as shown in Table 2.

As shown in Table 2, the user interfaces of mobile devices can be dynamically relocated depending on the user's business position("UR_CLASS"), the user's location area("LA_CLASS"), the interface reference count("REF_CNT"), the status of display on/off("DIS_FLAG"), the interface index("IF_IDX"), and etc.

Moreover, these dynamic interface configurations can be maximized for the utilization and convenience by defining a set of user interface patterns ("UI_PAT").

TABLE 2. USER MOBILITY INTERFACE PARAMETERS

UR_ID	UI_PAT	UR_CLASS	LA_CLASS	REF_CNT	DIS_FLAG	IF_ID_X
URxxx	UPxxx	Sxxx/Lxxx/Mxxx	Office/Ext/Plant mode	000 (High/Low)	On/Off	Mxxx/Ixxx/Wxxx

The displaying mechanism of mobility interface can be divided into the upward process and the downward process. For the mobility interface policy, the first upward policy preferentially should place the user interface into the upper-level that means a high referencing of use in context environments of the mobile users. The second downward policy preferentially should place the user interface into the sub-level that means a low referencing of use in the mobile terminals.

Figure 3 describes the mobility interface mapping logic to provide appropriate interfaces to mobile users in a mobile environment. It provides the interface pattern or interface real-time dynamic deployment on the basis of mobility interface parameters of Table 2 in the moving environment of the mobile user. Also, it changes the placement of the existing interface layout according to the movement environment of the mobile users.

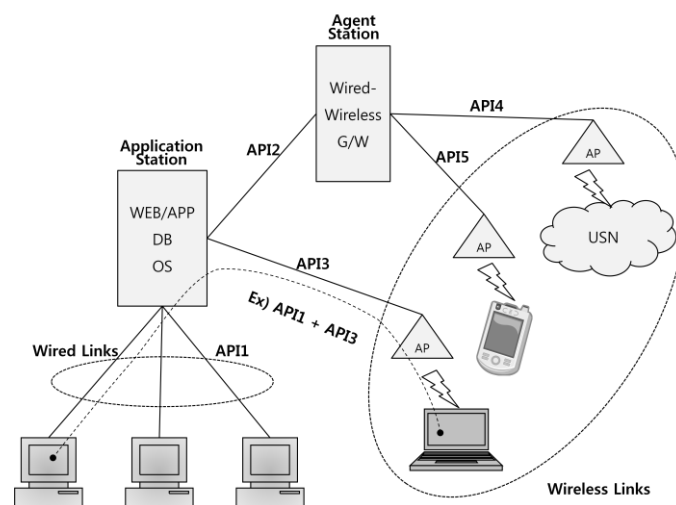


Figure 2. Mobile Architecture and Mobility Interface Scenarios

Figure 2 shows the configuration of the mobile networks to support mobile computing. It is consisted of a wired network, a wireless network, a wireless sensor network, and etc. as a middleware infrastructure. The mobile middleware system could be operated depending on the mobile network infrastructure environment, flexibly and dynamically. To build this flexibility, it requires each application interface (a set of APIs) to different respective link environments: That requires

API1 (wired link access for the terminals), API2 (wired link access for the agents), API3 (wired LAN link access for the computing terminals), API4 (wireless Ad-hoc link access for the sensor modules), API5 (wireless CDMA link access for the mobile phones) and so on. And these APIs can be selectively coupled for building the mobile application scenarios between the mobile terminals. For example, the merge of API1 and API3 can be bridged for the computing connection between the wired terminal and the wireless LAN-based terminal (Ex., API1 + API3).

```
Dynamic Load()
CXT_ID = context_sense();
if (current_CXT_ID < CXT_ID)
then CM_swap() by CXT_ID;
```

```
Interface Mapping()
UR_ID = event_sense();
if (current_UR_ID < UR_ID)
then employ UI_PAT or UI real-time replacement by UR_ID;
```

Figure 3. Dynamic Loading and Mobility Interface Mapping Logic

IV. SYSTEM ANALYSIS AND EVALUATION

This section evaluates the interface design dynamics method and the proposed lightweight method on an example of mobility scenario.

The Figure 4 shows a simulation map for proposing the mobility scenarios. The simulation map can be selective operated by the modes of random or sequential sequences. In this experiment, it is assumed that the user mobility pattern randomly appear in the simulation map.

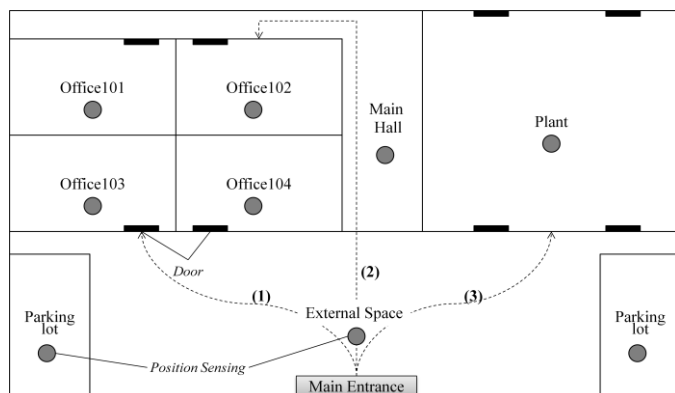


Figure 4. Map Example for Mobile Computing

Figure 4 shows an example of the application scenarios of mobile computing users: (1) the employee (or Member) movement path of Office103's finance department (Fxxx), and (2) the administrator (or Manager) movement path of Office102's research department (Rxxx), and (3) the director (or Staff) movement path of the Plant's manufacturing sector (Mxxx). The different paths of mobile application scenarios require different work schedules. Therefore, a different user interface to support these different application scenarios must be supported.

For the case (1), it should upload the application execution components to preferentially support the financial job and should relocate the user interface for the financial member. For the case (2), it should upload the research-related application components and should relocate the user interface for the research manager. For the case (3), it should preferentially upload the plant-application execution components and should relocate the oriented user-interface for the plant director.

V. CONCLUSIONS

We propose the lightweight method and the mobility interface design method in this paper.

It should realize space-effective mobile computing through the adaptive switching and the dynamic configuration. User interface convenience can be improved by minimizing the access count of interfaces.

Future researches will be performed for providing the efficient context switching in the dynamic configuration of mobile middleware components. Also, these dynamic configurations are unfamiliar to the first-time users and can be complicated. Thus, alternatives for these problems are needed.

REFERENCES

- [1] Toni A. Bishop and Ramesh K. Karne, "A SURVEY OF MIDDLEWARE".
- [2] Kurt Geihs, "Middleware Challenges Ahead", Computer, June 2001.
- [3] Salem Hadim and Nader Mohamed, "Middleware: Middleware Challenges and Approaches for Wireless Sensor Networks", IEEE DISTRIBUTED SYSTEMS ONLINE 1541-4922 © 2006 Published by the IEEE Computer Society, Vol. 7, No. 3; March 2006
- [4] Kiczales, G., des Rivieres, J., and Borrow, D., "The Art of the Metaobject Protocol", The MIT Press 1991.
- [5] Yokote, Y., "The Apertos Reflective Operating System: The Concept and its Implementation", In Proceedings of OOPSLA'92 (1992), pp. 414-434. ACM Press.
- [6] McAffer, J., "Meta-level Architecture Support for Distributed Objects", In Proceedings of Reection'96 (San Francisco, 1996), pp. 39-62.
- [7] Smith, B., "Reflection and Semantics in a Procedural Programming Language". Ph.D. thesis (January 1982), MIT.
- [8] Ledoux, T., "OpenCorba: a Reective Open Broker", In Reflection'99, Volume 1616 of LNCS (Saint-Malo, France, July 1999), pp. 197-214. Springer-Verlag.
- [9] Gordon S. Blair, Geoff Coulson, Anders Andersen, Lynne Blair, Michael Clarke, F' abio Costa, Hector Duran Limon, Tom Fitzpatrick, Lee Johnston, Rui Moreira, Nikos Parlavantzas, and Katia Saikoski, "The Design and Implementation of Open ORB 2", IEEE Distributed Systems Online, 2(6), 2001.
- [10] Fabio Kon, Manuel Roman, Ping Liu, Jina Mao, Tomonori Yamane, Luiz Claudio Magalhaes, and Roy H. Campbell, "Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB", In Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000), number 1795 in LNCS, pages 121-143, New York, April 2000. Springer-Verlag.
- [11] R. Hayton et. al., "FlexiNet Architecture Report", ANSA Phase III report, February 1999
- [12] M. van Steen, P. Homburg, and A. Tanenbaum, "Globe: A Wide-Area Distributed System", IEEE Concurrency, 7(1):70-78, March 1999.
- [13] Linthicum, D., B2B Application Integration: e-Business-Enable Your Enterprise, 2001
- [14] Jean Bacon, Ken Moody, John Bates, Richard Hayton, Chaoying Ma, Andrew McNeil, Oliver Seidel, and Mark Spiteri, "Generic Support for Distributed Applications", IEEE Computer, pages 68-77, March 2000.

- [15] Peter R. Pietzuch and Jean M. Bacon, "Hermes: A Distributed Event-Based Middleware Architecture", Submitted to the Workshop on Distributed Event-Based Systems (DEBS), 2002.
- [16] R. Meier and V. Cahill, "STEAM: Event-Based Middleware for Wireless Ad Hoc Networks", in Proceedings of the International Workshop on Distributed Event-Based Systems (ICDCS/DEBS'02). Vienna, Austria, 2002, pp. 639-644.
- [17] G. Cugola, E. D. Nitto, and A. Fuggetta, "The JEDI Event-Based Infrastructure and its Applications to the Development of the OPSS WFMS", IEEE Trans. on Software Engineering, 27(9): 827-850, Sept. 2001.
- [18] G. Ashayer, H. K. Y. Leung, and H.-A. Jacobsen, "Predicate Matching and Subscription Matching in Publish/Subscribe Systems," in Proceedings of the Workshop on Distributed Event-based Systems, 22nd International Conference on Distributed Computing Systems, (Vienna, Austria), IEEE Computer Society Press, July 2002.



Tae-Gyu Lee (BSc'92–MSc'96–PhD'06) He received the B.Sc. degree from Kunsan National University, Kunsan, Korea in 1992, the M.Sc. degree from Soongsil University, Seoul, Korea in 1996, and the Ph.D. degree from Korea

University in 2006. He is currently an Adjunct Professor in the Department of Computer Engineering, Korea Polytechnic University, Gyeonggi, Korea. Also, he is a Professional Researcher in Advanced Convergent Technology R&D Group, Korea Institute of Industrial Technology (KITECH), Ansan, Korea. He has also been a President in the JIGUNET Corporation, Seoul, Korea, from 1999. His research interests are in distributed systems, ubiquitous computing, middleware, networks, wearable and robot computing.



Gi-Soo Chung (BSc'88–MSc'91–Dr.Ing'00) He received BSc and MSc at Department of Textile Engineering in Kyung Hee University in Korea, and Dr.-Ing degree at Process Engineering, Stuttgart University in Germany in 2000. He is a principal researcher in Textile Fusion Technology R&D Group of KITECH (Korea Institute of Industrial Technology). His research interests include Digital Garment and Protective Clothing.