Widget Integration Framework for Context-Aware Middleware

Eun-Seok Ryu, Jeong-Seop Hwang, and Chuck Yoo

236, Department of Computer Science and Engineering, Korea University,
Anam-dong, Sungbuk-ku, Seoul, S.Korea
{esryu, jshwang, hxy }@os.korea.ac.kr

Abstract. Widget Integration Framework (WIF) is a framework that covers diverse widgets and their interactions with higher layers above. The framework offers several advantages of supporting a programming abstraction over widgets, supporting high reliability, run-time widget binding to the middleware and augmenting a service discoverer with available widget state information. This paper explains the processes of designing the WIF, including implementation, and applying WIF to the middleware. As an example, we explain a location-based service that uses a location positioning widget in a middleware employing WIF.

1 Introduction

There are several researches [15,16,17,18] for sensing environment changes and offering the most suitable context information to users. However, they were designed and implemented depending on restricted situations, only in several scenarios, only focusing on sensing an environment and managing context information with service discoverers and context manage modules, and are heavy to be applied to the mobile device for the future light-weight mobile middleware.

This paper presents WIF that addresses how middleware integrates and manages widgets effectively, binds appropriate widgets to the service discoverer, and supports mobile environment. The goal of WIF is to design and implement a light-weight widget abstraction layer for pervasive middleware. The focus is on widget integration, and such integration requires a programming abstraction over widgets. This framework encapsulates widgets, transmits sensed context information from widget to the upper middleware layers, registers and deregisters widget in real time and checks the registered widget's state for service discoverer.

The WIF has following advantages. First, it is light. Existing researches for pervasive environment(e.g. UIUC's Gaia and ASU's RCSM) provide CORBA-based services. However, this way makes offering dynamic service to handheld devices hardly. WIF is designed to support light-weight context-aware middleware platform for a mobile device. Second, WIF helps service discoverer to select a proper widget in the middleware by using our SDP(Service Discovery Protocol)-like protocol. Because WIF has an internal mechanism that can find available widgets and that delivers these widgets to the upper middleware layer, dynamic reconfiguration is supported. Third, WIF makes a middleware application fault-tolerant. Current middleware applications may impose a fatal obstacle on the entire system caused by an error from various hardware sensors. WIF monitors error occurrences by administrating widgets and disallowing the widget to operate in services. Through this, the WIF can support strong fault-tolerance. Finally, WIF has excellent interoperability with other middleware layers. It can interact effectively with other modules in middleware using the *Service Interaction Broker*[20] offering basic communications as well as an interface for interaction between service objects. The WIF that was developed through our research and related individual functions was verified by actual demonstration.

This paper is composed as follows. Section 2 explores existing systems aimed to support pervasive environments with their own visions. In Section 3, we describe the goal of this paper and the WIF design considerations. Section 4 describes the design of the WIF and its components in detail. In Section 5, we show how this WIF works in real pervasive environments by applying it to the middleware. In Section 6, we finally conclude our framework research for active surroundings and propose practical usage.

2 Related Work

2.1 Middleware for Pervasive Computing

Thorough research regarding middleware application that support pervasive environments includes projects, GAIA from the University of Illinois at Urbana-Champaign, AURA from Carnegie Mellon University, Oxygen from MIT and Dey's Context Toolkit from Georgia Institute of Technology. This paper addresses each of these research projects.

2.1.1 GAIA - UIUC

GAIA[1,2,3,15] is intended to coordinate software entities and heterogeneous networked devices contained within a physical space. GAIA is composed of four major building blocks, the Gaia Kernel, the Gaia Application Framework, the QoS Service Framework, and the Applications. The Gaia kernel contains a management and deployment system for distributed objects and an interrelated set of basic services that are used by all applications. The Application framework provides mobility, adulteration, and dynamic binding. The QoS service framework not only provides resource management for the QoS sensitive applications, but also dynamically adapts applications based on the QoS constraints and determines appropriate nodes for service instantiation. However, small mobile devices cannot cooperate autonomously without the infrastructure.

2.1.2 Aura - CMU

Aura[4,5,16] aims to provide a distraction-free computing environment where people can get services or perform their jobs without interventions from the system or environments. Aura is composed of five main components, Intelligent networking, Coda, Odyssey, Spectra and Prism. Odyssey supports resource monitoring and application-aware adaptation, and Coda provides support for nomadic, disconnect-able, and bandwidth-adaptive file access. Spectra is an adaptive remote execution mechanism that uses context to decide how to best execute the remote call. And, Prism is responsible for capturing and managing user intent.

2.1.3 Oxygen - MIT

Oxygen [17] aims to provide human-centered computing environments which help people automate repetitive human tasks, control a wealth of physical devices in the environment, find the information people need, and enable us to work together with others through space and time. To support dynamic and varied human activities, the Oxygen system focuses on the pervasiveness of the system or devices, supporting nomadic users, and representing system adaptability.

2.1.4 Context Toolkit – Gatech

The Context Toolkit[10,18] provides designers with the abstractions they have described — widgets, interpreters, aggregators, services and discoverers - as well as a distributed infrastructure. The Context Toolkit was developed using the Java programming language, programming language independent mechanisms were used, allowing the creation and interoperability of widgets, interpreters, aggregators and applications in any language.

2.2 Protocols for Service Discovery

Service Discover Protocol(SDP) finds a way software and network resources are configured, deployed, and advertised, all in favor of the mobile user researches on the emerging technologies of service discovery in the context of wireless and mobile computing are increasingly important. Some of these emerging SDPs include SLP, Jini, and UPnP(Universal Plug and Play). In these SDPs, UPnP uses SSDP(Simple Service Discovery Protocol)[23] for service discovery for announcing a device's presence and capabilities to others as well as discovering other devices or services.

3 Key Considerations

WIF is an individual framework that has ability to support applications without any other components or resources.

WIF is located in middleware's bottom layer in whole hierarchical system layers by integrating widgets which wrap sensors and it interacts with upper middleware through SOAP-based interface S*ervice Interaction Broker*[13]. If context-aware middleware uses WIF in it, it may manage service discoverer just only without caring about managing all sensors. In this chapter, we describe the key considerations of designing WIF.

- **High reliability** : The application expects data received from middleware layer to be reliable. Therefore, to support reliability, the middleware application needs functions that manage a widget's registration and deregistration, finds error occurrences and ignores targeted modules. This also includes filtering of abnormal sensor values.

- **Dynamic binding in mobile environment** : To support several services required from applications, middleware should be reconstructed dynamically according to situations of the context and of available hardware. Therefore, WIF needs to support run-time widget binding so that middleware can act plug-and-play. Some of previous systems which were developed for general purposes are heavy to apply to mobile device, because they are using XML-based protocol and CORBA for general interaction. However, we are trying to design a light-weight framework by choosing efficiency instead of generality. Of course, this way may cause system dependency to WIF, because it has to know all widgets will be used and has functions of those widgets in advance.

- **Common interface over different widgets** : WIF has interoperability with other middleware layers. It can interact effectively with other modules in middleware using the *Service Interaction Broker(SIB)*[20] offering basic communications as well as an interface for interaction between service objects. By this reason, other middleware components are allowed to just using a SIB-level interface. Consequently, it makes WIF offer a common interface over different widgets.

- **Augmenting a service discoverer by offering status information of available widgets** : If applications request necessary services from a middleware application's Service Discoverer, the Service Discoverer must be able to find and deliver available widgets among whole widgets, which support required services.

4 Proposed System Architecture

Fig. 1 shows the WIF system architecture and its internal components. It consists of Widget Information Manager, Fault Detector, Service Matcher, Widget Register and Widgets. In this section, each component is explained.

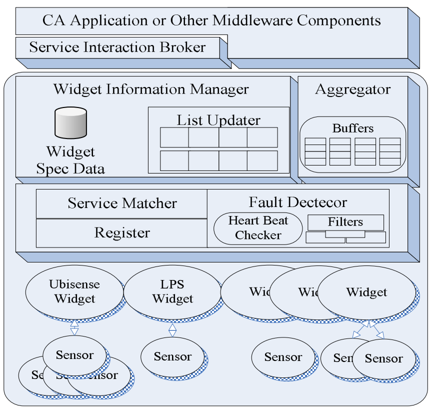


Fig. 1. Widget Integration Framework (WIF) Architecture

4.1 Widget

We use the concept *widget* as the Context Toolkit. Context widgets encapsulate information about a single piece of context, such as location or activity, for example. They provide a uniform interface to components or applications that use the context, hiding the details of the underlying context-sensing mechanisms [24]. In our research, we implemented Ubisense widget for location based service and it is treated in section 5.2.

4.2 Widget Information Manager

Widget Information Manager (WIM) role is to manage widget state information. It manages state information of widgets received from the Widget *Register* or *Fault-Detector* with *List updater*. It also keeps specification information of several widgets in order to find a suitable widget through *Service Matcher* regarding services required from the *Service Discoverer*. Also it exchanges all information from widgets with other modules in the middleware application through *Service Interaction Broker*.

4.2.1 List Updater

List updater (LU) belongs to WIM and manages the widget status information table. *Widget Type Number* in Table 1 divides widgets by the core unit role. For example, number 0 means that those widgets have relations with the Location Positioning System and number 1 means that those widgets are related to the sound/video input system. Within such classification, the *widget number* is granted to the widget, which is actually installed or is going to be installed. Then, the *Widget Type Number* and the *Widget Number* are combined and used as *Widget ID*. LU communicates whether relevant widget operates, encoded as 1 or 0 to the *Live Information* field. And, *State Information* expresses the state of widget (1: Occupied, 2: Requested, 0: available (non-used)). That is, value 0 means that the widget is not used and required at all, value 1 means that it is being used and value 2 means it was required to use if the relevant one is linked to the system. LU's role is table management. Therefore, other modules can use the necessary widget or unavailable widget in services using this table representation.

Table 1. Example of Widget State Table

Widget Type Number	Widget Number	Live Information	State Information
0	001	1	1 (Occupied)
(Location widgets)	002	1	1 (Occupied)
1	001	1	1 (Occupied)
(Sound Level /	002	0	2 (Requested)
Camera widgets)	003	0	0 (Available)
2	…	…	…
(…)	…	…	…

4.3 Fault Detector

Fault Detector is designed to check the state of the widget through a periodic heartbeat checker. It updates the *Live Information* value and the *State Information* value regarding the widget state table in *Widget Information Manager* if an error happens to the sensor or widget. LU informs *Service Discovery Manager* of these events. This guarantees high reliability by removing problematic widgets or replacing them with a new one. *Fault Detector* has various filters. Data, which is input from sensors, is transferred from the widget, and filtered. For example, a position revision filter corrects context location information retrieved from the Ubisense widget.

4.4 Service Matcher

Service Matcher finds a suitable widget for services that are required from *Service Discoverer*. For example, *Service Discoverer* in middleware applications requests a user's position information service from applications and must understand those requests, find a suitable widget and bind to it. *Service Matcher* references the widget state table as in Table 1 and finds an available widget, which can service position information. Detailed process steps are explained in Section 4.6.

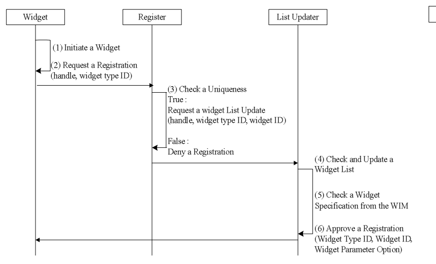
4.5 Widget Register

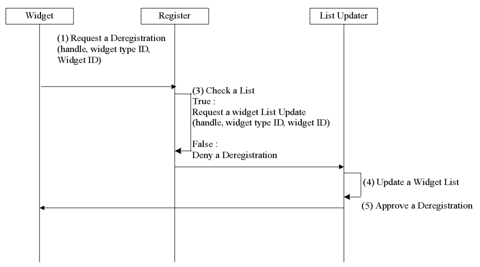
Widget Register takes charge of registration and deregistration of a widget. The widget helps *Widget Information Manager* by passing the state event to LU when it is registered/deregistered to the system. Though other research systems receive data only from pre-registered widget, Widget Register can support dynamic environment where widgets are registered/deregistered in real time by accepting a new incoming widget and notifying it to *Widget Information Manager*. In the result, *Widget Register* makes WIF bind a new incoming widget.

4.6 Interaction Diagram Among Components

The components of the WIF run through frequent interaction among them. This section explains 3 cases to show dynamic binding process using interaction diagrams [11] below.

(1) Widget registration and deregistration


Widget
Register
List Updater
(1) Initiate a Widget
(2) Request a Registration (handle, widget type ID)
(3) Check a Uniqueness True : Request a widget List Update (handle, widget type ID, widget ID) False : Deny a Registration
(4) Check and Update a Widget List
(5) Check a Widget Specification from the WIM
(6) Approve a Registration (Widget Type ID, Widget ID, Widget Parameter Option)


Widget
Register
List Updater
(1) Request a Deregistration (handle, widget type ID, Widget ID)
(3) Check a List True : Request a widget List Update (handle, widget type ID, widget ID) False : Deny a Deregistration
(4) Update a Widget List
(5) Approve a Deregistration

(2) Widget assignment by service request (when needed widget was pre-registered)

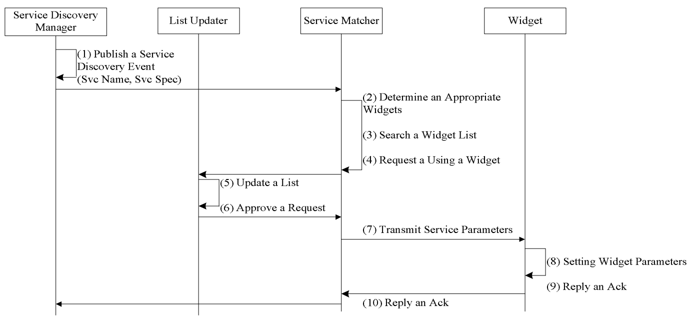

Service Discovery Manager
List Updater
Service Matcher
Widget
(1) Publish a Service Discovery Event (Svc Name, Svc Spec)
(2) Determine an Appropriate Widgets
(3) Search a Widget List
(4) Request a Using a Widget
(5) Update a List
(6) Approve a Request
(7) Transmit Service Parameters
(8) Setting Widget Parameters
(9) Reply an Ack
(10) Reply an Ack

Fig. 2. Interaction Diagram among Components

(3) Widget assignment by service request

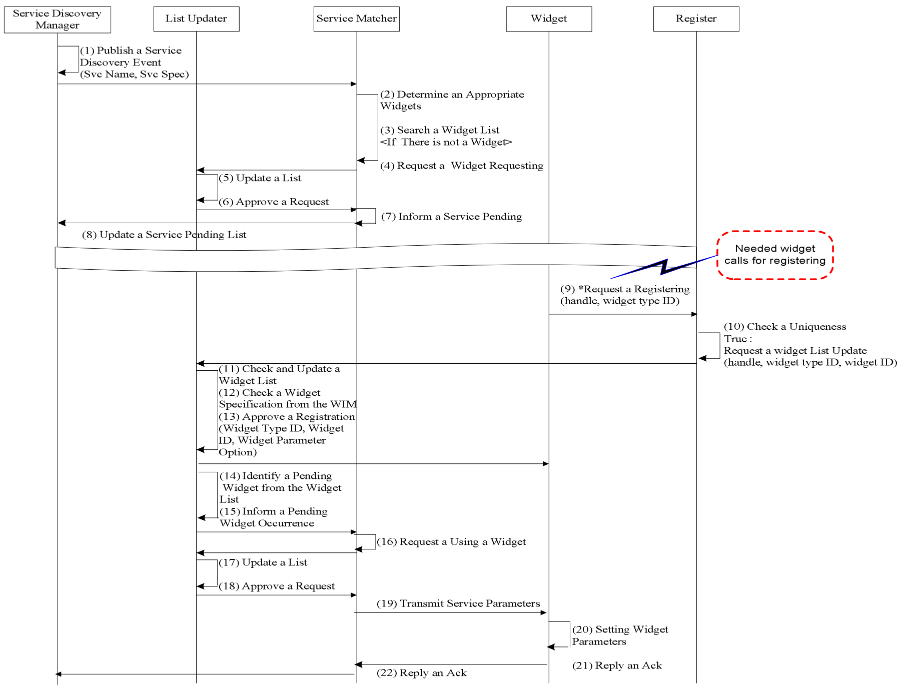

Service Discovery Manager
List Updater
Service Matcher
Widget
Register
(1) Publish a Service Discovery Event (Svc Name, Svc Spec)
(2) Determine an Appropriate Widgets
(3) Search a Widget List <If There is not a Widget>
(4) Request a Widget Requesting
(5) Update a List
(6) Approve a Request
(7) Inform a Service Pending
(8) Update a Service Pending List
Needed widget calls for registering
(9) *Request a Registering (handle, widget type ID)
(10) Check a Uniqueness True : Request a widget List Update (handle, widget type ID, widget ID)
(11) Check and Update a Widget List
(12) Check a Widget Specification from the WIM
(13) Approve a Registration (Widget Type ID, Widget ID, Widget Parameter Option)
(14) Identify a Pending Widget from the Widget List
(15) Inform a Pending Widget Occurrence
(16) Request a Using a Widget
(17) Update a List
(18) Approve a Request
(19) Transmit Service Parameters
(20) Setting Widget Parameters
(21) Reply an Ack
(22) Reply an Ack

Fig. 3. *(continued)*

5 Applying the WIF to Middleware

In this section, we address how to apply WIF to the middleware for a location-based service to show its usability. We currently developed several widgets including the *Ubisense widget*, *Fault-Detector* with filters for correcting location information, and implementing remainder parts according to our design. And our WIF is applied to the group-aware middleware[20] that was developed by ICU[13] to manage low level widgets. In the following sections, we briefly describe the group-aware middleware.

5.1 Group-Aware Middleware

Group-Aware Middleware(GAM) is researched to make middleware understand group-context in ubiquitous home. And this middleware infrastructure for *Active Surrounding*(a kind of ubiquitous home)[14] that focuses on group-context awareness and processing is studied by the pervasive computing group of Information and Communications University[11, 12, 13]. They defined *Active Surrounding* as a pervasive environment where entities (devices or services) actively response to user actions or help users to perform their jobs without intrusion to the users. We tried to apply our WIF to this GAM implementation so that it manages all sensor-related works. As a result, GAM with WIF consists of 4-units (*Context Manager, Dynamic*

Reconfiguration Supporter, Context-aware Service Discoverer and *Widget Integration Framework*(WIF)). Each component communicates with other components in the form of platform and location independence by the *Service Interaction Broker.*

Whole group-aware ubiquitous middleware research consists of unification of research results. The CAService(Context-Aware Service) can merge with required context information and publish events through the *Service Interaction Broker* API or join other events using the *Context Management* API by *Active Surroundings* middleware. *Context Management* relates to context information registered from the CAService by various sensor inputs that are input from the WIF. Also, it delivers the relevant CAService using event appearance context information. The *Dynamic Reconfiguration Framework* delivers events drawn from the CAService to CAServices that join to the events. In this situation, if it cannot find the appropriate CAService it conducts a semantics search of services through context-aware *Service Discovery*, searching for the CAService. Context-aware *Service Discovery* evaluates requests from the *Dynamic Reconfiguration Framework* and the CAService based on semantic characteristics of registered services. The Service Discover returns, finding the most suitable service to present context. The WIF focused in this paper integrates and manages widgets and allocates suitable widgets according to the service request.

5.2 Location-Based Service by Ubisense Widget

This paper explains location-based service by WIF-integrated middleware to show WIF's usability. This research uses Ubisense hardware sensor[21] devices to get context location information and develops Ubisense widgets for controlling those sensors. The characteristics of our implemented Ubisense widget are as follows.

- Passes location information (ID, timestamp, x, y, z) to the *Service Manager*
- Supports interaction between *Service Manager* and widgets
- Supports platform independence using JNI (Java Native Interface)

In the following sections, we examine the Ubisense widget embodied in this research in detail. For an explanation for the Ubisense marketing product, refer to the web site, this paper will not handle this in detail. Ubisense could be used regardless of development language because it supports COM (Component Object Model). Our middleware was developed using the Java programming language, because of this, we put a Java-COM wrapper over the Ubisense COM and developed the widget over the top. This Ubisense widget delivers sensed data to the upper middleware layers using a requested ratio from the middleware application. The system can achieve registration/deregistration of specific widget requests. Also, it corresponds to the Fault-Detector's heartbeat checking which is explained from lower middleware layers and has functions that transmit live messages. Of course, this ubisense widget achieves status management and message delivery with interaction using several components in the WIF. Context information is exchanged with other frameworks of a group-aware middleware using *Service Interaction Broker*.

The next figure shows the concept that the Ubisense widget delivers sensing information to upper middleware layer service manager using the *Service Interaction Broker (SIB)* and SOAP (Simple Object Access Protocol) interface.

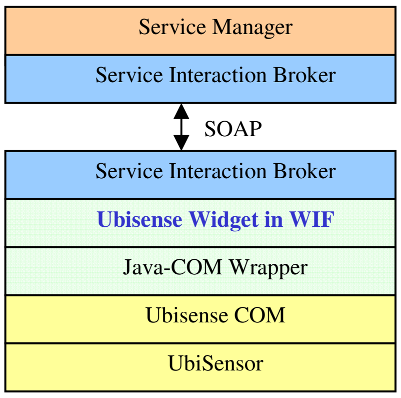


Fig. 4. The Hierarchical Architecture for Ubisense Widget

The Ubisense widget role is to deliver this context information to the upper layer by chasing tag-attached person's position from the actual Ubisensor. Therefore, it supports user's position chase, which is a basic requirement for all ubiquitous environments. This system was installed and demonstrated with sensors in a practical *'Active Home*' implementation. The demonstration scenario is that *'a group-aware middleware'* turns into a TV when user stands in front of it, shows broadcasting what a user wants and changes indoor temperature and lighting for more than two persons. The video clip for this demonstration is referenced in this paper [19].

6 Conclusion and Discussion

This paper explained several advantages that context-aware middleware can achieve by managing widgets that wrap sensors effectively. We describe the process of designing and implementing framework internals for managing widgets that support these advantages.

Many current middleware applications support a situation-aware application, integrating and analyzing primitive sensed information from widgets and delivering these to the middleware upper layer. However, these middleware applications cannot correspond properly in situations of widget malfunctioning and registering/deregistering during their operation. Therefore, we designed and implemented the *Widget Integration Framework* (WIF), located in the middleware lower layers. This framework encapsulates widgets, transmits sensed context information from widget to the upper middleware layers, registers/deregisters widgets in real time and checks a registered widget's state. With this processing, the WIF has exceptional error resilience properties and widget dynamic binding properties. The framework also has good interoperability because it communicates context information with other components of entire middleware applications *through the Service Interaction Broker (SIB),* using a SOAP interface. In addition, this framework is independent of middleware platforms and can be deployed using any development programming language on any operating system.

References

1. Manuel Román, Christopher K. Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt, “Gaia: A Middleware Infrastructure to Enable Active Spaces”, IEEE Pervasive Computing, pp. 74-83, Oct -Dec 2002.
2. Christopher K. Hess, Manuel Roman, and Roy H. Campbell, “Building Applications for Ubiquitous Computing Environments”, In International Conference on Pervasive Computing (Pervasive 2002), pp. 16-29, Zurich, Switzerland, August 26-28, 2002.
3. Renato Cerqueira, Christopher K. Hess, Manuel Roman, Roy H. Campbell, “Gaia: A Development Infrastructure for Active Spaces”, In Workshop on Application Models and Programming Tools for Ubiquitous Computing (held in conjunction with the UBICOMP 2001), September 2001.
4. David Garlan, Dan Siewiorek, Asim Smailagic, and Peter Steenkiste, “Project Aura: Towards Distraction-Free Pervasive Computing”, IEEE Pervasive Computing, special issue on Integrated Pervasive Computing Environments, Volume 1, Number 2, April-June 2002, pages 22-31.
5. Sousa, J.P., Garlan, D., “Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments”, Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture 2002, Montreal, August 25-31.
6. S. S. Yau, F. Karim, Y. Wang, B. Wang, and S. K. S. Gupta, “Reconfigurable Context - Sensitive Middleware for Pervasive Computing” IEEE Pervasive Computing, July-September 2002, IEEE Computer Society Press, Los Alamitos, USA, pp. 33-40.
7. Sergio Marti and Venky Krishnan, "Carmen: A Dynamic Service Discovery Architecture", Technical Report, August 2002.
8. Andry Rakotonirainy, Jaga Indulska, Seng Wai Loke, and Arkady Zaslavsky, “Middleware for Reactive Components: An Integrated Use of Context, Roles, and Event Based Coordination”, Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms, pp. 77-98, Heidelberg, Germany, November 12-16, 2001.
9. W. Zhao, H. Schulzrinne, E. Guttman, C. Bisdikian, W. Jerome, IETF RFC 3421, “Select and Sort Extensions for the Service Location Protocol (SLP)”, November 2002.
10. Anind K. Dey, Daniel Salber and Gregory D. Abowd, “A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications”, Human-Computer Interaction (HCI) Journal, Volume 16 (2-4), 2001, pp. 97-166.
11. Insuk Park, Soon Joo Hyun, Donman Lee, “Context-Conflict Management for Context-aware Applications”, Ubiquitous Computing Workshop 2004.
12. Dongman Lee, Soon J. Hyun, Young-Hee Lee, Geehyuk Lee, Seunghyun Han, Sae-Hoon Kang, Insuk Park, and Jinhyuk Choi, "Active Surroundings: A Group-Aware Middleware for Ubiquitous Computing Environments", Ubiquitous Computing Workshop 2004.
13. Pervasive Computing Group, “A Middleware Infrastructure for Active Surroundings”, TR-CSPG-2003-004-28-002. http://cds.icu.ac.kr/druid/res/TR-CSPG-2003-004-28-001.pdf
14. Dongman Lee, “Active Surroundings: A Group-Aware Middleware for Embedded Application Systems”, Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC ’04).
15. Gaia, web site: http://choices.cs.uiuc.edu/gaia/
16. Aura, web site: http://www.cs.cmu.edu/~aura/
17. Oxygen, web site: http://oxygen.lcs.mit.edu/
18. Context Toolkit, web site: http://www.cs.berkeley.edu/~dey/context.html
19. WIF, web site: http://os.korea.ac.kr/mediateam/WIF.htm
20. The Final Report of the Operation Digital Media Lab, February 2005.

21. Ubisense Company : http://www.ubisense.net
22. Simple Object Access Protocol (SOAP) : http://www.w3.org/TR/soap/
23. Simple Service Discovery Protocol (SSDP) Internet Draft
: http://www.upnp.org/download/ draft_cai_ssdp_v1_03.txt
24. Yen-Wen Lin and Hsin-Jung Chang, "Service Discovery in Location Based Services for Low-Powered Mobile Clients", http:// jitas.im.cpu.edu.tw/2004-2/5.pdf
25. Anind K. Dey, Daniel Salber and Gregory D. Abowd, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications", Human-Computer Interaction (HCI) Journal, Volume 16 (2-4), 2001, pp. 97-166