Toward a Message-Oriented Application Model and its Middleware Support in Ubiquitous Environments

Chun-Feng Liao, Ya-Wen Jong, and Li-Chen Fu
Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan
{d93922006, r96922022, lichen}@ntu.edu.tw

Abstract

Context-awareness has become a distinguishing feature of Ubiquitous systems. Contrary to desktop and web applications, Ubiquitous applications gather environmental context and provide services without user intervention. In this paper, we propose a message-oriented application model that is capable of modeling both user and system initiative interaction paradigms. Systems designed based on this model are inherently loosely-coupled and scalable. Results of this research include a systematic development procedure and its supporting middleware. We show the feasibility of this model by constructing several Ubiquitous applications for two dissimilar demo sites, and discuss experiences when adopting this model.

1. Introduction

Traditional desktop and web applications tend to be "user-initiative" [1], which force application users to pay more attention to interact with system via graphical user interfaces. In contrast, the goal of Ubiquitous computing is to help application users focus more on the activities they are doing than on interacting with the system. This goal can be accomplished by providing appropriate services according to context information without user intervention so that the users feel the system is "invisible" [2]. Therefore, the "system-initiative" or "mix-initiative" [1] nature of Ubiquitous systems has become a critical characteristic that distinguishes themselves from traditional systems.

Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, time, or device that is relevant to the system [3]. In a typical context-aware Ubiquitous system, context is usually inferred from environmental data gathered by sensors. According to context information, applications infer user situations and then trigger services by sending commands to actuators, for example, to turn on a light or to play a media file. We can observe that the data flow discussed above forms a feedback loop between environment and the Ubiquitous system (see Fig.1). Note that as opposed to RPC-style distributed applications, applications in a Ubiquitous system are usually "event-driven". A natural abstraction of this loop is to model the data dissemination as sequences of "messages" passing among message handlers [4].

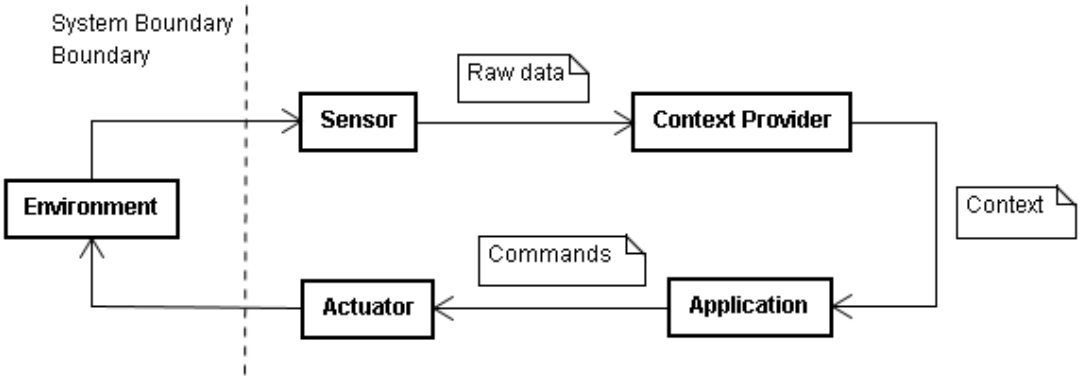

Figure 1. The feedback loop of a Ubiquitous system.

Messaging is a technology that enables asynchronous, loosely-coupled, and reliable communication. It has been widely used in integrating heterogeneous large-scale enterprise systems. When compared with other paradigms, messaging is more immediate than file transfer, better encapsulated than shared database, and more reliable than RPC-style invocation [5]. In [6], we proposed a service-oriented smart home architecture. This paper is primarily concerned with the system design and the infrastructure aspects, which were not addressed in our previous work. The objective of this paper is therefore threefold: (a) to propose message-oriented application model, which contains key abstractions and a systematic development procedure for Ubiquitous applications; (b) to present the supporting middleware of the proposed model; and (c) to report the lessons learned from constructing applications based on this model and its supporting middleware.

2. Related works

Many middleware have been developed since the rise of Ubiquitous computing. Several of them concentrate their works based on the service-oriented architecture, which has gained extreme popularity lately. The Context Toolkit [7] is the first attempt to address cross-cutting concerns in ubiquitous systems. In this toolkit, context are gathered by the context widgets (sensors) as well as from context aggregators, and then translated by the context interpreter. Messages between the above components are first encoded using the XML and then wrapped with HTTP. Context Toolkit supports RPC-based communication. The callback nature of context event is supported by implementing the Observer [10]. SOCAM [8] is one of the most popular service-oriented middleware. It aims to provide a rapid prototyping of context-aware services. The communication between SOCAM components are done through the Service Locating Service, the context consumers are able to locate and access the context providers through the Service Locating Service. MIRES [9] is one of the few message-oriented pervasive middleware developed to date. MIRES is built mainly for Wireless Sensor Networks (WSN) and is based on a publish/subscribe paradigm. MIRES focuses on the gathering of context information, and doesn't address the issues of application development.

3. Message-oriented application model

The communication mechanism of a message-oriented system is supported by the message-oriented middleware (MOM). The MOM creates a "software bus" for integrating heterogeneous applications. Most MOMs support both point-to-point (one to one) and

publish-subscribe (many-to-many) communication styles. Publish-subscribe style is more appropriate in a Ubiquitous environment. For example, sensors can not share their data among context providers with point-to-point mechanism. The logical pathways between publishers and subscribers are called "topics", which reside in the MOM. All software entities in MOM exchange messages via these logical pathways. Consequently, the MOM become Mediators [10] that make these entities loosely-coupled [5]. More specifically, dependencies between software entities are removed: they depend on topics instead of depending on each other. Hence, the message-oriented applications are also service-oriented. In the following, we use the term "service" to refer to the software entities reside in MOM.

As we shall see in the next two sub-sections, the proposed systematic procedure helps to transform the requirements of Ubiquitous applications into message-oriented design specifications. We can then take advantage of MOM when constructing and deploying Ubiquitous applications.

3.1. Key abstractions

In [11], the author proposed an abstract architecture for Ubiquitous systems. In this architecture, Ubiquitous systems consist of three service types: the Sensing, Thinking, and Acting subsystems. It is noteworthy that if we divide the Thinking subsystem into two parts, namely, Context Provider and Application, then the abstract architecture is consistent with the feedback loop illustrated in Fig.1

Table 1. Services and their associated operations

Service Type	Associated Topics	Operations
Sensor	SENSOR_DATA	Send gathered data to SENSOR_DATA topic
Actuator	COMMAND	Perform actions according to messages coming from COMMAND topic
Context Provider	SENSOR_DATA CONTEXT	Receive messages coming from SENSOR_DATA, transform them into context, and then send context messages to CONTEXT topic
Application	SENSOR_DATA CONTEXT COMMAND	Decide which action to take based on messages coming from SENSOR_DATA or CONTEXT topics. Send intended actions to COMMAND topic.

The Context Provider is responsible for the "inference" aspect of Thinking subsystem. It interprets raw data gathered by sensing subsystem and then publishes context information to a relevant topic. For instance, a "Fall-detection Context Provider" subscribes raw data from different types of sensors and fuses them with built-in statistical inference algorithms. The Application part represents the "process control" aspect of Thinking subsystem. In the previous example, when a fall event is detected, fall-detection module will publish context information to relevant topics. Upon receiving the context messages, the "Fall-detection Application" coordinates among actuators to perform expected actions, such as sending an alarm and calling out to an emergency center. In short, our work extends the abstract architecture in [11] by breaking its Thinking subsystem into Context Provider and Application. Table 1 summarizes the service types, their associate message topics, and the operations. Note that we have renamed the Sensing and Acting subsystems to Sensor and Actuator, respectively.

3.2. A Systematic development procedure

In this sub-section, we shall concentrate on a systematic procedure to construct message-oriented Ubiquitous systems. We divide the procedure into several phases: requirement gathering, system analysis, design, implementation and deployment. As sketched in Table 2, the procedure associates each phase with one or more mechanisms in order to produce the artifacts. The goal of requirement gathering phase is to collect user requirements. Many Ubiquitous application designers record the requirements using informal scenarios. Hence, we recommend User Story, which is an agile approach used to capture informal requirements [12]. For example, following is a portion of User Story describing a "Media follow me" application:

The system detects Bob presence in the living room and starts to play Bob's favorite music. After detecting Bob's movement to the kitchen, the system transfers the music to there without breaking off.

Table 2. A procedure for developing Ubiquitous applications

Development phase	**Mechanisms**	**Artifacts**
Requirement gathering	User Story [12]	Requirement documents
System analysis	TMSC [13]	analysis specifications
Design	Role mapping	Design specifications
Implementation	APIs provided by the middleware	Ubiquitous applications
Deployment	OSGi [14] or standalone	Ubiquitous system

In system analysis phase, the developer elaborates requirement documents using Triggered Message Sequence Chart (TMSC), which is a graphical, mathematically well-founded framework for capturing scenario-based system requirements of distributed systems [13]. There are many competitive methodologies such as Petri Net or UML Sequence Diagrams. We use TMSC here for three reasons. First, the graphical syntax of TMSC is extended from MSC (Message Sequence Chart) [15], which is designed to support the message-oriented systems. TMSC enhances MSC by its capability to model "event-driven" or "triggering" requirements. Second, TMSC supports compositional and evolutionary design, that is, developers can refine functional specifications incrementally and iteratively. Finally, the validity of new specifications can also be mathematically proofed. Fig. 2 depicts part of TMSC analysis specifications for the "Media follow me" application. A TMSC is composed of service entities representing software modules (SF, MFM App, Ds and Dt in Fig.2), the messages, local actions (the rectangle notations in Fig.2) performed by service entities, and the dashed line implying the trigger-action relationships. When the Smart Floor (SF) detects a user movement, it sends the user's location to the "Media follow me" Application (MFM App). MFM App checks if the cached user location has changed (local action *a*), if so, it sends a "Transfer" command to the Source Smart Display (Ds). The Ds gets the current playing position (local action *b*), sends the "Play" command with the playing position to the Target Smart Display (Dt) and stops the local media (local action *c*). Finally, the Dt starts playing the media from the playing position received (local action *d*).

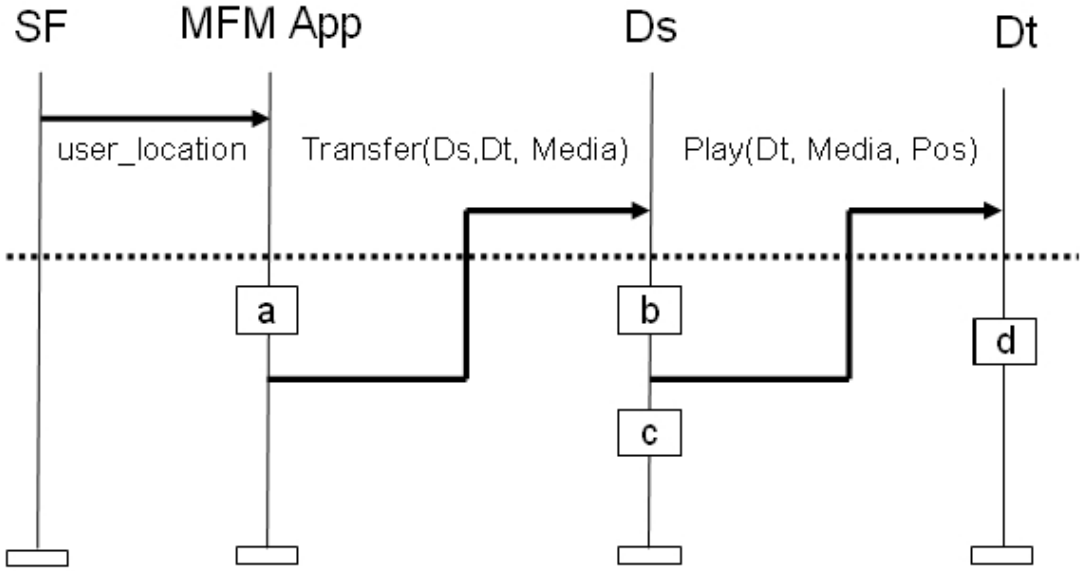

Figure 2. TMSC for the media follow me example

Next, we identify required middleware supports by performing role mapping. For each service entity in TMSC, we perform following steps:

1. Identify the service type
2. Do role mapping according to table 3.
3. Identify middleware support for each service entity.

For instance, the MFM App in Fig. 2 is with the service type "Application". According to table 3, MFM App is a Message Transformer in MOM, and it is a Platform Component. As a result, MFM App requires the middleware to provide message sending service and message receiving service.

In the implementation phase, developers build services by selecting appropriate API provided by the middleware. The API selection is closely related to the service's role in MOM. In the "Media follow me" example, MFM App is a Message Transformer in MOM, which is capable of both sending and receiving messages. As depicted in Fig. 3, MFM App can send and receive messages by implementing MessageListener interface and using MessageGateway interfaces.

Table 3. Role mapping of services

Service Type	**Role in MOM**	**Role in platform**
Sensor	Message Sender	Platform Adapter
Actuator	Message Listener	Platform Adapter
Context Provider	Message Transformer or Message Sender	Platform Component or Platform Adapter
Application	MessageTransformer	Platform Component

After building Ubiquitous applications, developers can either deploy them on an OSGi-based platform or as a standalone application. Experience shows that the standalone

deployment mode is more feasible in early stages of development, since applications will suffer from additional class loading issues induced by OSGi platform and make it harder to inspect the defects of applications.

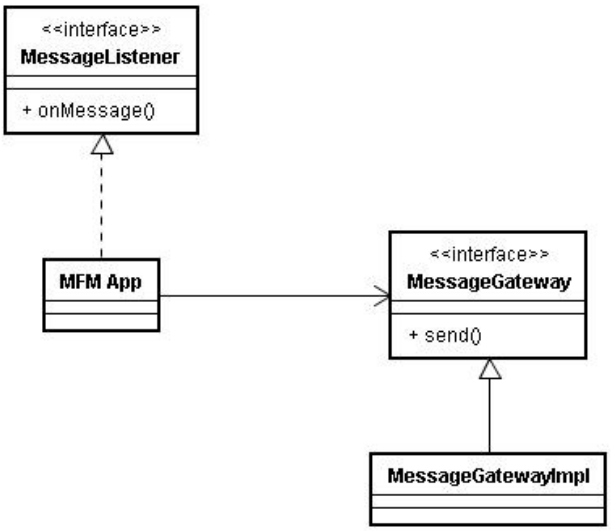

Figure 3. Simplified structure of the MFM App

4. The supporting middleware

We have built a middleware in order to provide infrastructural supports throughout the development lifecycle of a message-oriented Ubiquitous application. As revealed in Fig. 4, operations of the middleware are triggered by the events gathered by sensors at the bottom layer. Above this layer, we can obtain sensor data via vendor-specific control software modules. Integrating these heterogeneous modules requires developers to construct a Platform Adapter for each module. These adapters are analogous to the “drivers” of an operating system. Consequently, the construction of a Platform Adapter is a one-time effort, through which the sensor hardware can be shared among applications.

Platform Adapter converts sensed events to messages, and then publishes them to topics located in the Home Message Bus (HMB). In this middleware, HMB plays the role of a MOM, which is responsible for exchanging messages. Hence the modules reside in the top layer, that is, Applications, Platform Services, and System Tools, can receive, process, and send messages come from topics of HMB. We will discuss the detail of these services in section 4.1. The operations we have described so far also apply in principle when command messages come from the top-layer modules to the bottom-layer actuator hardware.

4.1. Platform Services and System Tools

Platform Services modules provide design time and runtime support for hosting application. The HSB (Home Service Bean) Framework is an application framework that provides design time supports with a set of reusable libraries, interfaces, and default implementations. At runtime, the Context Broker is the gateway to the context repository, and hosting applications query persisted context information through it. The Command Executer is essentially a Façade [10], by which the hosting applications can interact with Platform Adapters of actuators in a unified way. More specifically, applications can control smart

appliances from different vendors with a consistent set of message syntax, which can be interpreted by the Command Executer service. Since applications are separated by message topics, System Tools such as HMBM (HMB Monitor) and HMBS (HMB Shell) are useful when debugging and testing Ubiquitous applications. For example, developers can send testing messages using HMBS and then check the outputting results using HMBM before deploying applications to the Ubiquitous environments.

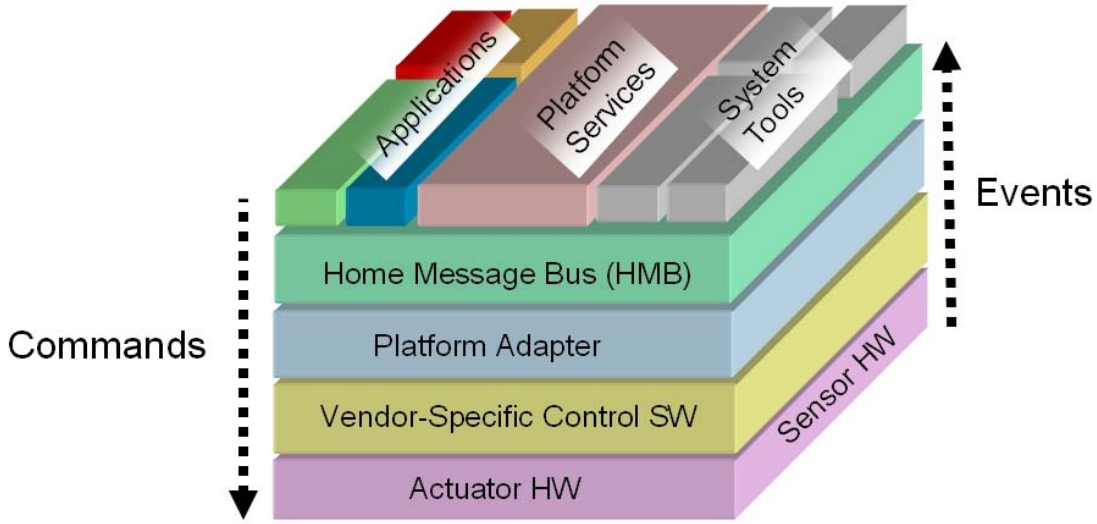

Figure 4. Architecture of the supporting middleware

4.2. Implementation

The realization of this middleware is mainly based on JDK 5.0 and Knopflerfish 2.0.0 [16], an OSGi R4 implementation. In the OSGi deployment mode (see section 3.2), the top-layer modules run as threads that are spawned from OSGi bundle activators. Otherwise, these modules run as standalone java applications. We use ActiveMQ 4.1.1 [17], an open source embeddable MOM, to implement HMB. ActiveMQ uses a cross-platform messaging protocol, and supports several programming languages such as C, C++, C#, and Java. The persisted context information is stored in MySQL 4.0.18 database. The middleware is hosted on a P4/1GHz mini-PC with 1GB memory.

5. Case study

We verify the feasibility of the proposed application model by developing Ubiquitous applications for two dissimilar demo sites (Fig. 5). They are different in size (NTU Attentive Home: 400 square feet; Open Lab: 1080 square feet.), partition (NTU Attentive Home is with 1 living room, 1 kitchen and 1 bedroom; Open Lab is with 1 living room, 1 kitchen, 1 toilet, 1 dining room and 2 bedrooms), appliances, and furnishing. Due to their dissimilarities, developers have to modify xml-based configuration files for each demo site in order to deploy the proposed middleware. However, we do not need to change the source code.

Figure 5. Two dissimilar demo sites. Left: the NTU Attentive Home; Right: Open Lab for NSC S2H Home Project

Table 5. The Web-based Monitoring and Control Application allows users to manipulate home services remotely.

Service Name	Service Type	Implementing Technologies
Web GUI		Java Servlets / AJAX
Context Listener	Application	Java
Smart Appliance Controller	Actuator	Java / OSGi

Table 6. The MFM Application enables media to migrate to the display that are closest to a nomadic user.

Service Name	Service Type	Implementing Technologies
Smart Floor Controller	Sensor	C#
MFM App Controller	Application	Java / OSGi
Smart Display Controller	Actuator	C#

Table 7. The Fall Detection Application keeps an eye on the inhabitants and sends alarm when fall events are detected.

Service Name	Service Type	Implementing Technologies
Fall Detector	Sensor	C++ / Open CV
Fall Detention App Controller	Application	Java / OSGi
Alert	Actuator	C#

Table 8.The Smart Air Conditioner Application operates air conditioners and fans automatically based on sensed information.

Service Name	Service Type	Implementing Technologies
Sensor Node Adapter	Sensor	Java
Smart Air Conditioner App Controller	Application	Java / OSGi
Smart Appliance Controller	Actuator	Java / OSGi

Tables 5 through 8 describe five applications we have deployed in both demo sites. These applications fall into two categories. The “user-initiative” applications such as the Web-based Remote Monitoring and Control and the Voice Command applications require user to direct system actions via user interfaces. On the other hand, the Media Follow Me, Vision-based

Fall Detection, and Smart Air Conditioner applications are typical "system-initiative" context-aware applications that do not need user interventions. It is important to note that in the proposed application model, an application can compose of components implemented by means of heterogeneous technologies. This ability is important in developing Ubiquitous applications. For instance, the real-time image-processing components are better implemented with C or C++ while server-side components are usually implemented with Java language. The cross-platform interoperability is an inherited nature of the message-oriented applications, which is very hard to achieve in other application model. Besides, to avoid the possibility of single-point-of-failure, most available MOM supports load-balancing as well as fail-over mechanisms. As a result, the MOM-based Ubiquitous applications are also scalable and robust.

6. Conclusions

We have presented a message-oriented application model, a natural abstraction of the data dissemination in Ubiquitous systems, and its supporting middleware. Based on the proposed approach, we developed several Ubiquitous applications for dissimilar demo sites. Applications constructed with this application model are loosely-coupled, cross-platform, and more scalable. There are still many important issues not addressed in this work, for example, the service composition / discovery mechanisms [18] [19], and the feature interaction problem [20]. In addition to the above topics, we are also investigating an RDF-based [21] communication protocol that directly supports ontological inference. Besides, we also noted that most available methodologies gathering requirement are not adequate for Ubiquitous environment. Further studies on an agile requirement analysis methodology for Ubiquitous applications are therefore obviously needed.

7. References

[1] N. Ramakrishnan, R.G.Capra III, and M.A.Perez-Quinones, "Mixed-Initiative Interaction = Mixed Computation," in Proc. ACM SIGPLAN Workshop PEPM'02, January 2002.

[2] M.Weiser, "The Computer for the 21st Century, ", Scientific American, 265(3), 66-75, 1994.

[3] A.K.Dey, "Understanding and using context," Personal and Ubiquitous Computing Journal, issue 5, vol. 1, 2001.

[4] E.Souto, G.Guimaraes, G.Vasconcelos, M.Vieira, N.Rosa, and C.Ferraz, "A Message-Oriented Middleware for Sensor Networks," in Proc. 2nd International Workshop on Middleware for Ubiquitous and Ad-Hoc Computing, Toronto, Ontario, Canada, 2004.

[5] G.Hohpe and B.Woolf, Enterprise Integration Patterns: Design, Building, and Deploying Messaging Solutions, Addison Wesley, MA, 2004.

[6] C.L. Wu, C.F.Liao and L.C.Fu, "Service-Oriented Smart Home Architecture based on OSGi and Mobile Agent Technology," IEEE Transactions on Systems, Man and Cybernetics - Part C, Special Issue on Networking, Sensing, and Control, vol.37, no.2, 2007.

[7] D.Salber, A.K. Dey and G.D. Abowd, " The Context Toolkit: Aiding the Development of Context-Enabled Applications, " in Proceedings of the 1999 Conference on Human Factors in Computing Systems (CHI '99), Pittsburgh, PA, May 15-20, 1999. pp. 434-441.

[8] T.Gu, H.K. Pung, and D.Q.Zhang, "Toward an OSGi-based Infrastructure for Context-Aware Applications," Pervasive Computing, Vol.3, No.4, pp.66-74, 2004.

[9] E.Souto, G.Guimaraes, G.Vasconcelos, M. Vieira, N.Rosa, and C.Ferraz, "A Message-Oriented Middleware for Sensor Network," Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing, pp.127-134, Toronto, Ontario, Canada, 2004.

[10] E.Gamma, R.Helm, R.Johnson, and J.Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[11] S.Loke, Context-Aware Ubiquitous Systems: Architectures for a New Breed of Applications, Auerback Publications, NW, 2007.

[12] K.Beck, Extreme Programming Explained: Embrace Change, Addison Wesley, 2000.

[13] B.Sengupta and R.Cleaveland, "Triggered Message Sequence Charts," IEEE Transactions on Software Engineering, Vol. 32, No. 8, 2006.

[14] OSGi Alliance, URL: <http://www.osgi.org>.

[15] Message Sequence Charts (MSC), ITU-TS Z.120, 1996.

[16] Knopflerfish, URL:<http://www.knopflerfish.org/>

[17] ActiveMQ, URL:<http://www.activemq.org>

[18] H.Pourreza and P.Graham, "On the Fly Service Composition for Local Interaction Environmnets, " in Proc. Ubiquitous Middleware Workshop 2006 (PerWare '06), Pisa, Italy, March 2006.

[19] C.Campo, M.Munoz, J.C.Perea,A.Marin, and C.Garcia-Rubio, "PDP and GSDL: a new service discovery middleware to support spontaneous interactions in Ubiquitous systems, " in Proc. Ubiquitous Middleware Workshop 2005 (PerWare '05), 2005.

[20] M. Kolberg, E. H. Magill, and M. Wilson, "Compatibility issues between services supporting networked appliances," in IEEE Communications Magazine, Vol. 41, Issue 11, Nov. 2003.

[21] Resource Description Framework, URL: <http://www.w3.org/RDF/>.

8. Acknowledgement

This research is supported by the National Science Council of Taiwan, under Grant NSC96-2752-E-002-007-PAE, and by the Chunghwa Telecom Co., Ltd. Telecommunication Laboratories, under Grant TL-97-9501.

Authors

Chun-Feng Liao received the B.S. and M.S. degrees in Computer Science from National Cheng-chi University in 1998 and 2004, respectively. He is currently a Ph.D. candidate in the Department of Computer Science & Information Engineering at National Taiwan University. His research interests are Intelligent Systems, Context-Aware Middleware, and Service-Oriented Software Architecture in Smart Living Space.

Ya-Wen Jong received a BS in Computer Science & Information Engineering from National Central University in 2007. She is currently a graduate student in the Department of Computer Science & Information Engineering at National Taiwan University. Her research interests include Dynamic Service Management and Service-Oriented Software Architecture in the Smart Living Space.

Li-Chen Fu received the B.S. degree from National Taiwan University in 1981, and the M.S. and Ph.D. degrees from the University of California, Berkeley, in 1985 and 1987, respectively. Since 1987, he has been on the faculty of and currently is a professor in both the Department of Electrical Engineering and Department of Computer Science & Information Engineering of National Taiwan University. He is now a senior member of both the Robotics and Automation Society and Automatic Control Society of IEEE, and he became an IEEE Fellow in 2004. His areas of research interest include robotics, FMS scheduling, shop floor control, home automation, visual detection and tracking, E-commerce, and control theory & applications.