A Convergence of Context-Awareness and Service-Orientation in Ubiquitous Computing

Hoijin Yoon,

Department of Computer Science and Engineering, Ewha Womans University, Seoul, Korea

Summary
Ubiquitous service needs to react to context as well as normal input. Therefore, services in ubiquitous computing would be implemented as context aware applications, which handle two different types of input; one is explicit as a normal type of input and the other is implicit as context.
However, Ubiquitous Computing requires the seamless adaptation and extension of context aware applications. Some approaches to fit the two requirements, the adaptation and the extension, are based on Component-Based Software Development paradigm instead of Service-orientation, even though the service-orientation has the abstraction, which could support the seamless adaptation, and it consists of loosely-coupled services, which could support the seamless extension. That's because the know-how of Component-Based Software Development has longer history and is more proved than the service-orientation. This paper proposes a convergence of context-awareness and service-orientation according to the principles of service-orientation, and explains its contribution compared to other work of combining the context awareness and the service-orientation.

Key words:
Context-awareness, Service Oriented Architecture, Ubiquitous Computing, Abstraction, Loosely Coupled.

1. Introduction

The real world as a Ubiquitous Computing (UC)'s domain is more dynamic than the traditional computing domains. That's because ubiquitous services should react to the context as well as to the normal input, where the ability to adapt to the behaviors based on knowledge of its context is called *context-awareness* [1]. The *Context-awareness* is a critical feature where there are frequent changes of context. A context aware application treats the context as an implicit input as well as the normal input as an explicit input. From the handling-context of the context aware application, services in UC tend to be implemented as context-aware applications.

The common approach to deal with the context awareness is based on IF-THEN rules [2, 3]. The rules are simple IF-THEN rules used to specify how the context-aware applications should adapt. The famous context aware application, for instance, *Active Badge*-based "*Watchdog*" and PARC's *tab* based "*Contextual Reminder*," are the rule based systems [4]. However, from a software engineering perspective, this rule based programming is a failure because the systems developed in it are not able to maintain or test, and then it is unreliable [5]. Therefore, various new approaches instead of rule-based systems have been proposed, and the approaches more focus on what UC requires of the context aware applications.

One of the requirements of UC is *adaptation*. There are two types of context-aware applications; a discrete one and a continuous one [6]. The discrete application triggers actions at every well-defined point of time. In contrast to it, the continuous application continuously updates the parts that are dependent on context. This kind of actions is called the context adaptation where the context-aware application modifies its behaviors according to the context changes. For supporting the adaptation, Context Toolkit [7] proposed a layered model of developing a context aware application. Each layer has its own abstraction level from the sensor-biased to the application-biased. It could be said that the abstraction is the key for supporting the adaptation.

The other requirement of UC is *extension* without rebuilding the application from the scratch. Applications in UC are often requested to be extended for accepting new context information. Also, they are composed or separated each other over and over depending on the scenario of using the context. To rescale or expand applications could overload the adaptation, and it causes the cost of developing context-aware applications high. That's why the cost-effective extension should be considered in developing context-aware applications. Some approaches solve the extension by composing software components, which is more loosely-coupled than traditional components. It said that one of the key for solving the extension is to use the loosely-coupled.

The two requirements, the adaptation and the extension, have been answered with Component-based Software Development (CBSD) paradigm, which was applied to the Context Driven Component Model [8], Gravity [9], Multifacet [10], instead of the service-orientation, even though the service-orientation allows a more abstract level and more loosely-coupled component than CBSD. That's because the know-how of CBSD has

longer history and is more proved than the service-orientation. This paper proposes a convergence of context-awareness and service-orientation according to the principles of service-orientation, and explains its contribution compared to other work of combining the context awareness and the service-orientation. It is explained with principles of the service-orientation not with the techniques or the solutions.

Section 2 explains how the principles of service-orientation support the adaptation and the extension with Service-oriented Architecture (SOA). Section 3 mentions the contributions with compared to other studies of converging context awareness and SOA, and concludes this paper with some on-going future work.

2. Context-Awareness on Service-Orientation

Abstraction is the key in SOA, which has layers of abstraction such as the application service layer, the business service layer, and the orchestration service layer as shown in Figure 1[11]. The application service layer establishes the ground level foundation that exists to express technology-specific functionality. While application services are responsible for representing technology and application logic, the business service layer introduces a service concerned solely with representing business logic, called the business service. The orchestration service layer introduces a parent level of abstraction, and the orchestration brings the business process into the service layer, positioning it as a master composition controller.

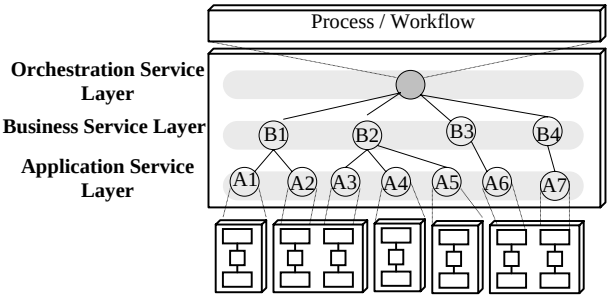

Fig.1 SOA Configuration [11]

Through the abstraction implemented distinct service layers described above, key SOA characteristics can be realized. There is no official set of service orientation principles yet. There are, however, a common set of principles most associated with service-orientation; *Services are reusable, Services share a formal contract, Services are loosely coupled, Services abstract underlying logic, Services are composable, Services are autonomous, Services are stateless, and Services are discoverable*. Of these eight, autonomy, loosely coupling, abstraction, and the need for a formal contract can be considered the core principles that form the baseline foundation for SOA. How to narrow down from the eight principles to the four core ones is explained in [11].

Table 1 Points of convergence

Context Awareness	SOA core principles
Adaptation	*Abstraction*
Extension	*Loosely coupled* *Need for a formal contract* *autonomy*

The four core principles mentioned above are supporting the adaptation and the extension needed in context aware applications as shown in Table 1. The points of their convergence in ubiquitous computing are where *abstraction* in service-orientation supports the *adaptation* in context-awareness, and where the *loosely coupled* with need for a formal contract and autonomy in SOA supports the *extension* in context-awareness. The detail about them will be in Section 2.1 and Section 2.2.

2.1 Adaptation of Context-Awareness and Abstraction of Service-Orientation

The abstraction principle supports the adaptation of context awareness. For adapting applications to updated context, applications consist of layers abstracted by context sensitivity. The application service layer in Figure 1 links application services and implementation units only by writing the link information as a service description language, for instance, Web Service Description Language (WSDL). It means services are abstract since they do not embed implementation unit like codes. That's why application services are not specific to a technology, a domain, or an implementation platform. This kind of approaches has their roots in a software engineering theory known as "separation of concerns." This theory is based on the notion that it is beneficial to break down a large problem into a series of individual concerns. This allows the logic required to solve the problem to be decomposed into a collection of smaller, related pieces. Each piece of logic addresses a specific concern.

SOA can be viewed as a distinct manner in which to realize a separation of concerns [11]. This characteristic is the *abstraction*. Services in SOA abstract underlying logic. The only part of a service that is visible to the outside world is what is exposed via the service contract. Underlying logic, beyond what is expressed in the descriptions that comprise the contract, is invisible and irrelevant to service requestors. This abstraction supports

the adaptation needed by context awareness in ubiquitous computing as seen in Table 1.

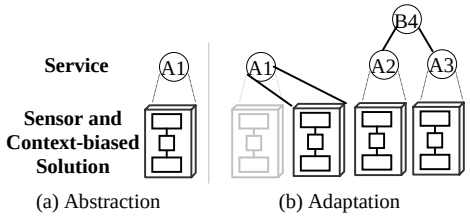

Fig. 2 Adaptation through the Abstraction

Figure 2 shows how to support the adaptation through the abstraction. SOA configuration in Figure 1 is for implementing a business process not for an application. Till now, SOA is generally targeting a business process not a traditional application. Therefore, context aware applications are mapping to the application service layer. In Figure 2 (a), A1 has only the description of itself, and it is linked to the specific parts like context-biased solution by describing the relation in the service description inside itself. As seen in Figure 2 (b), A1 can use other solutions depending on updated context information needs simply by modifying its link information in the description. Suppose a business service, B4, needs both of A2 and A3, it also does not embed the contents of the lower layers like the application service layer or specific solutions. It only has its own description about its using application services. If other service, for example A4, is wanted by B4, B4 would only update the description. All above the adaptation is enable from the abstraction of SOA.

2.2 Extension of Context-Awareness and Loosely-Coupled of Service-Orientation

For extension, software components associated with context information in a context aware application need to be composable more simply than software components of CBSD. Unless, a whole of the context aware application should be rebuilt from the scratch whenever new context information is needed. The *loosely coupled*, one of the principles of SOA, makes the components more composable. Services in SOA are loosely coupled. It means that services must be designed to interact without the need for tight, cross-service dependencies.

The loosely coupled is caused from the need for the formal contract and autonomy of the principles described in Table 1. For services to interact, they need not share anything but a formal contract that describes each service and defines the terms of information exchange. It is the *need for the formal contract*. Also, the logic governed by a service resides within an explicit boundary. The service has control within this boundary and is not dependent on other services for it to execute its governance. It is the *autonomy*.

From the loose coupled, context aware applications can be reorganized according to the need of new context information. An application service, A1 in Figure 3, implements behaviors for context information, and it is autonomous as a service. If the context aware application wants to handle some more context information, it gets other pre-built services associated with the added context. If there are no matched services in its range of architecture, it needs the developer to implement a new autonomous service for the context added. Anyhow, a wanted new service, A2 in Figure 3, is ready to compose. The services, A1 and A2 in Figure 3, are composed by updating a formal contract, the filled circle in Figure 3, as they share nothing but the formal contract according to the principles of service orientation. The formal contract can be more in the orchestration service layer than in the business service layer. The context-driven component model [10] implemented this concept for the extension within CBSD paradigm.

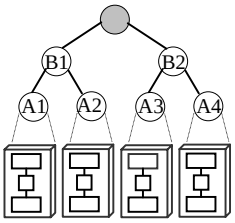

Fig. 3 Extension through the Loosely Coupled

2.3 Integration of UC and SOA

UC and SOA can be integrated with the orchestration service as a central junction as seen in Figure 4. Figure 4 harmonizes UC architecture [12] of context awareness with SOA configuration [11]. The left side of the orchestration service in the figure is the UC architecture, and the right side is SOA. In SOA side of Figure 4, services located in each layer communicate each other only through the orchestration service instead of binding each other directly as in CBSD. In UC side of Figure 4, the orchestration service is deployed separately from the part, where the raw context information is sensed and refined. The lower layer is sensor-biased, and the orchestration is not binding directly to a sensor. It is the orchestration service deployed in the center of Figure 4 that finds appropriate services in the SOA and requests the service by sending messages. This section describes how to apply the polymorphism to the mixed architecture of Figure 4 for supporting both of the context adaptation and

the performance, which are the two considerations mentioned in Section 1.

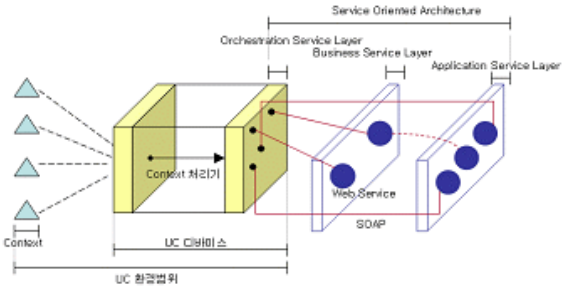

Fig. 4 Integration of UC and SOA

3. Contributions and Future Works

Recently, some studies coordinate the context-awareness and the service-orientation. *SOA for context aware applications* [13] and *Web services enabling context aware application* [14] are published recently, and they consider the UC environment like this paper. The former developed four basic services used in implementing location based services and to publish them as web services. If a developer want to use the services, he could subscribe them according to *publish-subscribe* mechanism of Web service. The latter proposed some web services, which handle context with its proposed context format. Once context is described as the format, the context could be handled with the proposed services. It is also based on Web service mechanism.

Table 2 compares *A Convergence…* proposed in this paper with two other work described above in some factors; "*SOA*" in Table 2 mentions if one considers SOA principles or not, "*Adaptation*" or "*Extension*" expresses if one solves the adaptation or the extension needed by context awareness or not, "*Specific Application*" means if one is implementation technique for a specific application or not.

Table 2 explains two contributions of our work as followed.

Table 2 Comparison with Other Work

	SOA	*Adaptation*	*Extension*	*Specific Application*
SOA for Context-aware applications	X	O	X	O
Web services enabling context-aware applications	X	O	X	O
A Convergence…	O	O	O	X

First, our work has derived a solution for the adaptation and the extension that the ubiquitous computing requires of context-awareness, from service-orientation. Of the requirements, the adaptation is generally more focused than the extension by other work as shown in Table 2. Our work tries to solve the requirements through the service orientation, and shows that the principles of SOA support them in Section 2 step by step.

Second, our work has applied the contemporary SOA to the convergence. Most of the solutions of SOA are evolving into accepting the contemporary SOA from the first-generation web service; Fiorano is launching Aqualogic™ as the next version of Fiorano BIS™, and IBM also releases the Websphere™ with a modeling process using Rational Rose™. As seen in Figure 5, the contemporary SOA is based on the principles of service orientation described in Section 2, and it includes web service concepts as an implementation technology. Also it adopts WS* specifications that support the principles of service orientation. However, the other work in Table 2 defined services within the web service boundary. Our work proposes the convergence of context awareness and service orientation from the level of the contemporary SOA.

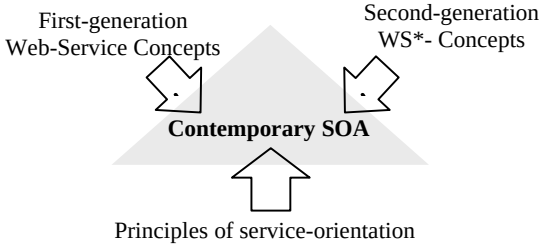

Fig. 5 Contemporary SOA

As a future work, we are studying a development process of context aware applications targeting ubiquitous computing. Ubiquitous systems are getting bigger and mission-critical and the real world is getting involved in the ubiquitous environment. Nevertheless, traditional developments of context aware applications do not consider software engineering approaches. This paper shows that SOA meets the requirements of context aware applications with the principles of contemporary SOA. Based on this work, the development process for context-aware applications will be specified to SOA.

References

[1] Gregory D. Abowd, "Software Engineering Issues for Ubiquitous Computing," in Proceedings of International Conference on Software Engineering, 1999, pp.75-84

[2] Gu T., Pung K., Zhang D.Q., "A service-oriented middleware for building context aware services," Journal of Network and Computer Applications, Vol.28, No.1, 2005, pp.1-18

[3] Yau S.S., Karim F., “An Adaptive Middleware for Context-Sensitive Communications for Real-time Applications in Ubiquitous Computing Environments,” Real-Time Systems, Vol.26, No.1, 2004, pp.29-61
[4] Schilit, B., et al., Disseminating Active Map Information to Mobile Hosts. IEEE Networks, Vol.8, No.5, 1994, pp. 22-32
[5] Xiafeng Li, “What’s so bad about rule-based programming,” IEEE Software, Vol.8, No.5, 1991, pp.103-104
[6] Brown, P.J., Bovey, J. D. and Chen, X., “Context-Aware Applications: From the Laboratory to the Marketplace,” IEEE Personal Communications, Vol.4, No.5, 1997, pp.58-64
[7] Anind K. Dey and Gregory D. Abowd, "The Context Toolkit: Aiding the Development of Context-Aware Applications," in Proceedings of Workshop on Software Engineering for Wearable and Pervasive Computing, 2000.
[8] Hoijin Yoon, Byoungju Choi, "The Context Driven Component Supporting the Context Adaptation and the Content Extension," Journal of Information Science and Engineering, Vol. 22 No.6 pp.1485-1504, 2006
[9] Humberto Cervantes and Richard S. Hall, “Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model,” in Proceedings of the 26th International Conference on Software Engineering (ICSE’04), 2004
[10] Anca Rarau, Kalman Pusztai, and Ioan Salomie, “MultiFacet Item Based Context-Aware Applications,” International Journal of Computing & Information Sciences, Vol. 3, No. 2, 2005, pp.10-18
[11] Thomas Erl, ‘Service Oriented Architecture - Concepts', Prentice Hall, 2005
[12] Karen Henricksen and Jadwiga Indulska, "Developing context-aware pervasive computing applications : Model and approach," Pervasive and Mobile Computing, Vol.2, No.1, pp.37-64
[13] Damiao R. Almeida, et al., “Using Service-Oriented Architecture in Context-Aware Applications,” GEOINFO 2006.
[14] C.H. Jardim, et al., “Web services enabling context-aware application: Lessons learned by integrating e-learning applications, ” in Proceedings of the international conference on next generation web services practices, 2005.

Hoijin Yoon received the B.S. and M.S. degrees in Computer Science and Engineering from Ewha Womans University in 1993 and 1998, respectively. She also received her ph.D with the dissertation about software component testing from Ewha. During 2004-2005, she stayed in Georgia Institute of Technology as a visiting scholar. She is interested in Software Testing, Service Oriented Architecture, and Context awareness in Ubiquitous Computing. She has worked in Ewha Womans University as a full-time lecturer since 2006.