A Polymorphism Implementation of Web Services for Context Adaptation and Performance

Hoijin Yoon† and Youngcheol Park ††,

Hyupsung University, Korea†
Hankuk University of Foreign Studies††

Summary
Context aware applications in Ubiquitous Computing (UC) require the dynamic adaptation to the context and the interoperability across heterogeneous platforms, as a distributed system. Service-oriented Architecture (SOA) enables the dynamic adaptation through using loosely-coupled services, and it also supports the interoperability through using XML messaging. For the adaptation, the orchestration service of SOA implements the adaptation steps inside the source code with some APIs. It also causes more messaging between services, but the XML messaging is the bottleneck of the performance. However, UC requires the frequent messaging between various sites within its boundary according to the frequent changes of the context. Finally, the context adaptation causes lots of messaging, and the messaging causes the decrease of the execution performance. This paper proposes a polymorphism implementation of SOA services to satisfy both of the context adaptation and the execution performance. It also measures how well the polymorphism model satisfies them with an implementation of J2EE.

Key words:
Web Services, Polymorphism, Service-Oriented Architecture, Context-Awareness, Ubiquitous Computing.

1. Introduction

Context aware applications in Ubiquitous Computing (UC) is better to be built under Service-Oriented Architecture (SOA) from the following three reasons. First, UC devices want a little size of codes since the device is getting tiny and the memory space is getting small [1]. That's why the applications in UC had better be built as distributed systems, where only the client side is put on the UC device. SOA fits to the distributed systems as architecture allowing more flexible connections between the services. Second. UC applications should be adapted to the changes of context on the run-time [2]. The software architecture in the traditional system is fixed before the run-time. In contrast, SOA does not compose services by combining their interfaces statically. It only sends SOAP messages to the wanted-services and receives the reply from them dynamically. SOA guarantees that each service is autonomous as well as loosely-coupled to other services, and the service does not consider how other services communicate. Before the run-time, SOA specifies only the work flow of services in the orchestration service. Namely, it is not fixed which message is sent to which service. Therefore, SOA allows the orchestration service to switch the called-services to others that implement the right behaviors of the updated context. Third, UC should guarantee the interoperability of various platforms. The applications of UC are ported on the various platforms, and they often communicate each other across the platforms. SOA enables the heterogeneous platforms to communicate by using XML technology, which aims at the platform-independent messaging. Therefore, XML-based messaging of SOA supports the interoperability between the platforms in UC.

The web service is counted as one of the SOA implementation technologies [3]. However, the use of web services when building UC applications has the following considerations, even though SOA is able to meet the UC requirements. First, the web service system should implement the *Context Adaptation*, where the service reacts to the updated context on the run-time[4]. An UC application dynamically recognizes the changes of context such as locations, time, user's preferences, and so on. And then it should be adapted to the changes by finding appropriate services and binding to them on the run-time. The orchestration service codes most of the steps from finding services to sending messages with iterative loops and variables of services' URLs or SOAP message's instances. Second, the performance should be considered because the SOAP messaging is one of the main causes to drop the performance. Some empirical studies and experiments [5,6,7,8] show that SOAP messaging could be a bottleneck in web services. However, UC anticipates a frequent SOAP messaging of services. Therefore, cutting down the messaging time of an UC application would have a good influence on the performance.

Although we should consider both of the context adaptation and the performance, the implementation of the context adaptation would drop the performance because it causes the frequent messaging. As one of the solutions of this dilemma, this paper applies the polymorphism concept,

Manuscript received March 5, 2008
Manuscript revised March 20, 2008

"one interface multiple implementations", to the web services running in UC. The polymorphism is expected to shorten the steps of the context adaptation, and the short steps cut down the messaging time. It would result in the better performance.

2. Polymorphism Implementation Model

Figure 1 harmonizes the UC architecture[9] of the context-awareness with SOA configuration[10] with putting the orchestration service on a central junction. In the SOA side of Figure 1, services located in each layer communicate each other only through the orchestration service, instead of binding each other. In the UC side of Figure 1, the orchestration service is deployed separately from the part, where the raw context information is caught and refined. The lower layer is sensor-biased, and the orchestration is not binding directly to a sensor. This section describes how to apply the polymorphism to the mixed architecture of Figure 1 for supporting both of the context adaptation and the performance.

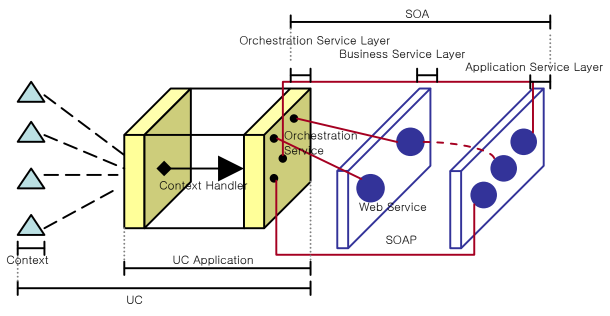

Figure 1. Integration of UC and SOA [11]

2.1 Polymorphism for the Context Adaptation

Figure 2 shows how the polymorphism operates the context adaptation under Figure 1. The web service, C or D, which implements an appropriate behavior of the changed context, has the same interface as the previous service, B. In Figure 2, the filled circles of the web service, B, C, or D, is the interface, while the implementation of the service is drawn as a different shape inside the service. The same interface results in the almost same SOAP messages, and then it also results in the reduction of steps of sending messages including writing SOAP messages. Through the polymorphism, the orchestration service adapts to the updated context without rebuilding SOAP from the scratch since it has already knew the interface of the service wanted for the updated context.

Suppose that the polymorphism is applied to the *in-car information system*[7]. It is one of the context aware applications in UC, and it informs drivers of weather information, traffic condition, local information, and so on. The system is installed inside a car, and the car is moving around. The service, B in Figure 2, provides the general weather information in the downtown area, and the service, C or D, could inform the driver of the weather information with some tips for driving in the mountain area. In Figure 2, ① shows that the service, C or D, has been built to implement the function of telling the drivers the weather information with some tips of driving in the mountain-area. It is about the same context type - the location - and the different context value - the mountain-area or the downtown-area. In ② of the figure, the context is changed as the car has moved to the mountain area. The context value that was changed from the downtown area to the mountain area is passed to the orchestration service with expecting that the service behaves the right function that the updated context requests. According to the expectation of the right behavior, the service, C or D, is selected through ③. The orchestration service does not need to know which service of B, C, or D is acting, since all of them have the same interface. That is one of the advantages coming from the polymorphism.

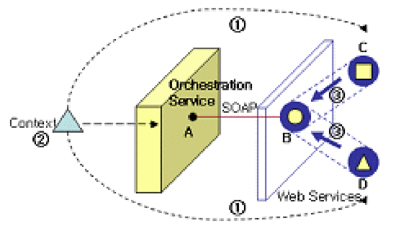

Figure 2. Context Adaptation through Polymorphism

2.2 Polymorphism for the Performance

There are two different forms of the implementation of the polymorphism in Object-Oriented Programming; one is *Overloading*, and the other is *Overriding*. Each form of the polymorphism cuts the steps down through its own way. Table 1 shows how well the form, the overloading or the overriding, handles four factors; The context adaptation and the performance are the main considerations to be focused on in this paper, and the implementation of a web service and the implementation of an orchestration service are two different views in developing web services. In the preference column of

Table 1, << or < shows that a left one is superior to a right ones, and vice versa. << has a stronger meaning than <.

Table 1. Comparison of the Overloading and the Overriding

	Overloading	*Preference*	*Overriding*
Context adaptation	OK	=	OK
Implementation of a web service	weak restrictions	>	strong restrictions
Implementation of an orchestration service	complex	<<	simple
Performance	long steps for the adaptation	<<	short steps for the adaptation

In Table 1, *the preference of the context adaptation* in the overloading is equal to that in the overriding, because both of them implement the context adaptation with making the steps shorter. In case of *the implementation of a web service*, the overloading allows some more variations than the overriding. That's because a web service implemented with the overloading is allowed to have a different type of arguments, while that with the overriding is not. Therefore, the preference of *the implementation of a web service* in the overloading is better than that in the overriding. However, *the implementation of an orchestration service* in the overloading is much worse than that in the overriding. In case of the overriding, an orchestration service only send a message, while an orchestration service in the overloading should write a new message and then send it as long as the context is updated. This is since the strong restriction, that keeps the name of the web service same, does not have a good influence on *the implementation of a web service*. *The performance* depends on how many steps the orchestration service has for handling the context adaptation. Therefore, the overriding is better than the overloading as much as *in the implementation of an orchestration service*. In conclusion, the overriding is superior to the overloading in the performance with handling the context adaptation, and it is because the overriding simplifies the orchestration service more than the overloading.

3. Performance of the Context Adaptation

This section explains some contributions with an empirical study based on J2EE, because J2EE was evaluated to guarantee the better performance than .NET through the experiment [12].

The orchestration service works with some APIs for handling the context adaptation. First of all, it gets WSDL from UDDI by accessing to UDDI and getting the URL of WSDL through JAXR API. After downloading WSDL, it writes SOAP messages. It uses JAX-RPC API and JAXP API for generating a DOM tree to treat WSDL files and SOAP messages, which are XML documents, and it also uses SAAJ API to write and send SOAP messages. After sending messages, the orchestration service receives the messages replied from the web services with SAAJ API. The orchestration service is finally deployed on J2EE container. The sequence described above is written as the steps in Figure 3.

Usually, ① and ② of Figure 3 are done before executing the orchestration service. It means that the orchestration service only sets some real values to a SOAP message and sends it to the web service that has been selected before. WSDLs and SOAP messages has been already fixed when implementing the orchestration service. In this usual case, only ③ and ④ need to be implemented as codes inside the orchestration service. However, suppose that the orchestration service is in UC. It is the orchestration service that should handle all the steps, from ① to ④, inside itself. That was one of the requirements of the application in UC as mentioned before.

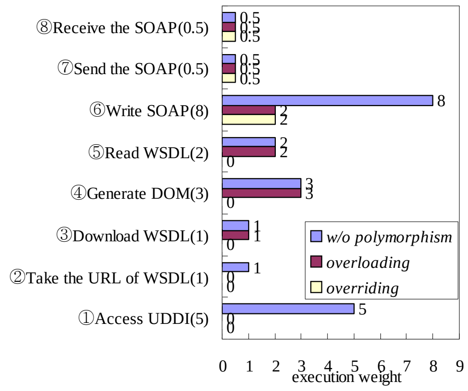

Figure 3. Execution Weight of the Orchestration Service

Figure 3 shows how many weights each step executes in each case of *without-polymorphism*, *overloading*, and *overriding*. The *weight* is calculated by counting the lines of the algorithms [13] of each step. For instance, "Write SOAP" in ② of Figure 3 has the weight, 8, where it consists of all the eight steps of Figure 4. The orchestration service of *without-polymorphism* executes all the 8 lines, while that of the *overloading* or the *overriding* needs only two steps - the line 7 and the line 8 in Figure 4. That's why the chart draws the two-length bar on it. All the full weight of each step is described inside the parenthesis of Figure 3. From the chart, the total weight, where the orchestration service executes for the context adaptation, is calculated as 21 for without-polymorphism, 9 for overloading, or 3 for overriding. The overriding carries the less weight than the overloading, and the use of the overriding could reduce the weight more considerably if compared to without-polymorphism, as we expected in Section 3.

```
WriteSOAP
1. Create SOAPMessage
2. Create SOAPPart
3. Create SOAPEnvelop
4. Create SOAPFactory
5. Get an object of SOAPBody
6. Create BodyEntry
7. Create a child of BodyEntry
8. Create SOAPConnection
```

Figure 4. Steps of "WriteSOAP"

The difference of the execution weight of the cases would be getting considerable depending on how many context changes are recognized by the orchestration service. Suppose that the changes of context have happened three times during the run-time, the adaptation would be repeated at three times. It also means that the steps from ① to ⑧ of Figure 3 are repeated. The orchestration service developed through *without-polymorphism* would be heavier than that of the *overriding* by (21-3)*3. The difference between *without-polymorphism* and the *overloading* would be (21-9)*3. Figure 5 shows how much heavier the execution weight would be on each case as the number of the iterations of the context adaptation goes big. As seen in the figure, the line of *without-polymorphism* is rising more sharply than the others, and the line of *overriding* is rising more slowly than that of *overloading.*

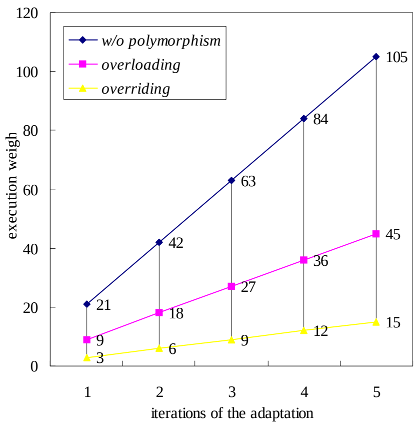

Figure 5. Increasing of the Cases

The first conclusion is that *with-polymorphism* is better than *without-polymorphism*, and the second one is that the *overriding* is better than the *overloading*, in the execution weight of handling the context adaptation from the view of the orchestration service. The decrease of the execution weight would result in the decrease of the messaging cost, which is one of the factors of the web service performance.

4. Conclusions

The characteristics of SOA satisfy the requirements of the applications running in UC from the three reasons explained in Section 1. However, the usual way to implement the orchestration service can hardly handle the context adaptation, where the application in UC updates itself to do the right function that the updated context expects on the run-time. The orchestration service needs to implement all the steps, shown in Figure 3, inside its own codes in order to handle the context adaptation. It causes the orchestration service to have lots of SOAP messaging and it finally has a bad influence on the performance. The performance of the web service is one of the main issue the web service should solve practically [5,6,7,8], however UC expects fast reactions as the real-time services. That's why this paper proposes the polymorphism model, which cuts the steps of the orchestration service down.

The empirical study of Section 3 showed that the polymorphism implementation model contributes to simplifying the orchestration service that handles the context adaptation and it would have a positive influence on the performance. As a compensation of simplifying the orchestration service, it constrains the web services to use the fixed interface. However, it is not a big weak point because the polymorphism model focuses on the development of applications running in UC not on the development of web services. Section 3 concluded that *with-polymorphism* is better than *without-polymorphism* and the *overriding* is better than the *overloading* from the view of the orchestration service's performance of executing the context adaptation.

References

[1] 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

[2] Gregory D. Abowd, "Software Engineering Issues for Ubiquitous Computing," Proceedings of ICSE '99, pp.75-84, May 1999, LA, CA, USA

[3] Thomas Erl, *Service-Oriented Architecture -A Field Guide to integrating XML and Web Services*, Prentice Hall, 2004

[4] Ian Sommerville, 'Software Engineering', 8th edition, Addison Wesley, 2007

[5] K.Chiu, M.Govindaraju, and R.Bramley, "Investigating the limits of SOAP performance for scientific computing," Proceedings of IEEE International Symposium on High Performance Distributed Computing, pp.246-254, 2002

[6] D.David and M.Parashar, "Latency Performance of SOAP implementations," Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid," pp.407-412, 2002

[7] R.Elfwing, U.Paulsson, and L.Lunberg, "Performance of SOAP in Web Service Environment Compared to CORBA,"

Proceedings of the 9th Asia-Pacific Software Engineering Conference, 2002

[8] Wei Jun et al, "Speed-up SOAP Processing by Data Mapping Template," Proceedings of the 2006 International workshop on Service-oriented Software Engineering, 2006

[9] 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

[10] Thomas Erl, *Service Oriented Architecture - Concepts*, Prentice Hall, 2005

[11] Hoijin Yoon, "A Convergence of Context-Awareness and Service-Orientation in Ubiquitous Computing," IJCSNS International Journal of Computer Science and Network Security, Vol. 7 No.3, pp.253-257, 2007

[12] Sun Microsystems Inc. Web Services Performance comparing Java2 Enterprise Edition and .NET Framework, http://java.sun.com/performance/reference/whitepapers/WS_Test-1_0.pdf, 2004

[13] Mincheol Shin, *XML Web Services*, Freelec, 2004

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. After the degree, she stayed in Georgia Institute of Technology as a visiting scholar and then worked at Ewha Womans University. She has been teaching at Hyupsung University as a full-time lecturer since 2007, She is interested in Software Testing, Service Oriented Architecture, and Context awareness in Ubiquitous Computing.

Youngcheol Park completed his B.S. in 1992 and M.S. in 1994, both from Yonsei University. In 2004, he completed his Ph.D. in Electrical Engineering at Georgia Tech. In March 2007, Dr. Park joined the faculty at Hankuk University of Foreign Studies, where he is Assistant Professor in Dept. of Electronics Engineering. Dr. Park has over 12 years of industrial experience of such as CDMA, mWiMAX mobiles as well as RF circuits and systems.