JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 22, 1485-1504 (2006)

The Context Driven Component Supporting the Context Adaptation and the Content Extension*

HOIJIN YOON AND BYOUNGJU CHOI
Department of Computer Science and Engineering
Ewha Womans University
Seoul, 120-750 Korea
E-mail: {hjyoon; bjchoi}@ewha.ac.kr

The context aware applications should focus on the Context adaptation, where the context change is reflected in the application, and the Content extension, where a new content of context is added without rebuilding the whole application. This paper defines Context Driven Component, which implements behaviors required by a context. An application is developed through composing the context driven components. It supports the context adaptation through replacing components or the content extension through adding components implementing behaviors relevant to the extended contents. The development using the context driven components will be analyzed in the following respects; the scale of context, the vertical decomposition compared to the existing way, and the implementation in Ubicomp.

Keywords: context, adaptation, ubiquitous computing, component, context awareness

1. INTRODUCTION

The real world where Ubiquitous computing is applied is changing more dynamically than the traditional domains of computing. Ubiquitous computing (Ubicomp) needs the ability to modify behavior of an application based on knowledge of its context of use [1]. The ability is called *context-awareness*. *Context-awareness* is a critical feature for the ubiquitous computing system where there are frequent changes of context. Context-aware application *adapts* according to the location of use, the collection of people nearby, hosts, and accessible devices, as well as to changes to such things over time [2]. There are two types of context-aware applications; a discrete one and a continuous one. 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 [3]. This kind of actions is called *Context adaptation* where the context-aware application modifies its behaviors or parts according to context changes.

For the context adaptation, a context aware application typically has rules governing how the application should respond to context changes. The rules are simple IF-THEN rules used to specify how the context-aware applications should adapt. The common approach to deal with the application adaptation is based on IF-THEN rules [4, 5]. The fa-

Received September 29, 2005; revised February 22, 2006; accepted June 14, 2006.
Communicated by Sy-Yen Kuo.
* This research was supported by the Ministry of Information Communication (MIC), Korea, under the Information Technology Research Center (ITRC) support program supervised by the Institute of Information Technology Assessment (ITTA).

mous context aware applications, for instance, Active Badge based "Watchdog" and PARC's tab based "Contextual Reminder," are the rule based systems [6]. From a software engineering perspective, the rule based programming is a failure because the systems developed in it are not able to maintain, test and unreliable [7]. In addition, Ubiquitous systems are getting bigger and mission-critical, as the real world is getting involved in the ubiquitous environment. Therefore, instead of the rule-based system, this paper proposes a new way to build a context aware application. The idea was originated from Component Based Software Development (CBSD). CBSD has many contributions from reuse, encapsulation, and modularity that the rule-based system sacrifices to obtain simplification [7].

The new way this paper proposes enables an application to adapt to the dynamic changes of context by loading only the behaviors representing the current content of context, and extends the contents of context by adding the behavior of the added content as a component without recoding the application. The former is called Context Adaptation, and the latter is called Content Extension. For supporting these qualities, this paper introduces a development of a component model sensitive to context, which is called *Context Driven Component*.

In section 2, the context-related works are mentioned. In section 3, it describes Context Driven Component and its Context Adaptation and Content Extension. Section 4 illustrates the application development using the Context Driven Components in detail by an example. Section 5 evaluates how useful the context driven component is by the scale of contexts, compares the context driven components to the existing approaches to develop context aware applications, and lists some advantages by implementing context driven components in Ubicomp devices. Finally, section 6 makes some conclusions and mentions some future work.

2. RELATED WORK

2.1 Context Aware Application

Context refers to the physical and social situation in which computational devices are embedded. The use of context allows an application to be tailored to a user's specific situation [8]. One goal of context-aware computing is to acquire appropriate to the particular people, place, time, events, *etc*.

A context aware application definition was given by Schilit and Theimer [2], in 1994, as *software that adapts according to its location of use*, *collection of nearby people and objects*, *as well as changes to those objects over time*. Since then, the definitions of context-aware applications were related to applications' adaptation, reactivity, responsiveness and sensitiveness to context. For instance, Pascoe *et al*. [9] define context-aware computing to be the ability of computing devices to detect and sense (sensitiveness), interpret and respond to (reactivity) aspects of a user's location environment and computing devices. In [2], context-aware applications are defined as applications that dynamically change or adapt their behavior based on the context of the application and the user. Fickas *et al.* [10] define context-aware applications (called environment-direct applications) to be applications that monitor changes in the environment and adapt their opera-

tion according to predefined or user-defined guidelines. Finally, Dey *et al.* [8] claim it was necessary to give a more generic definition, which is not bound to a specific characteristic; it is adaptation, reactivity, responsiveness or sensitiveness). Dey's definition states that a system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task. According to this definition, to be a context-aware application, the only requirement is to respond to context and thus, detection, interpretation and adaptation are not mandatory characteristics. This definition makes sense; for instance, in applications which do not adapt to context but simply reflect the context to the user or the ones that do not detect or sense context. Detection and interpretation can be performed by other computing entities. We consider this informal definition of context-awareness as a reference for our work.

2.2 Context Aware Application Design

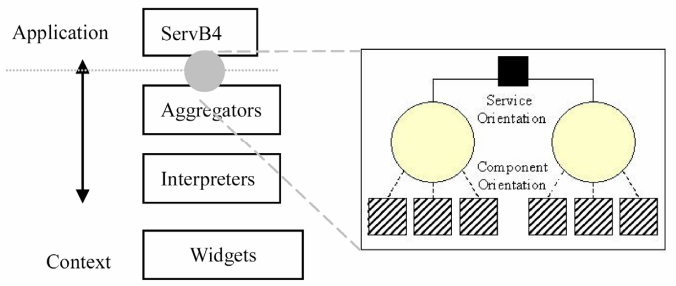
Most of the context-related researches focus on capturing and modeling context. Few of them focus on how an application accepts contexts and adapts to contexts. As mentioned before, ad-hoc approach for developing a context aware application does not meet what Ubicomp requires handling dynamic context changes. Therefore, more sophisticate approaches are needed. Some of them are Context Toolkit [11], Gravity [12], Multifacet [13] approach and so on.

2.2.1 Context toolkit

Context Toolkit [11] makes the distributed nature of context, transparent to the context-aware applications. The applications do not need to know whether these context components are executed remotely or locally. All of the components share a common communications mechanism (XML over HTTP) that supports the transparent distribution. These components run independently as a single application, allowing them to be used by multiple applications.

The Context Toolkit aims to separate the concerns of applications from the concerns of context related environments effectively. It makes an application consists of layers representing the separated concerns, such as Context Widgets, Interpreters, Aggregators, Servers, and Discoverers. The layers are arranged horizontally from a context-biased layer, Context Widget, to a service-biased layer, Server. The Context Driven Components can be located between the Aggregators and the Service if the Context Toolkit adopts the context driven components proposed in this paper.

The Context Toolkit arranges the layers horizontally from a context-biased layer to a service-biased layer as shown in Fig. 1 (a). From top to bottom, the layer is getting context-biased, and from bottom to top, the layer is getting service-biased. The context-based layer means the layers handling the context as just data, and the service-biases layer means the layer handling the context as input of the application. The Context Toolkit starts from receiving the context from the sensors, and the context is refined, and finally the context is applied to a service. The Context Toolkit is a framework to build the context aware applications that execute the whole process from receiving the data from the sensor to handling the services. How to handle the context adaptation is not dealt in Context Toolkit, however, this paper proposes the way to extend the scale of context as



(a) Context toolkit. (b) Context driven components.

Fig. 1. Comparison of context toolkit and context driven components.

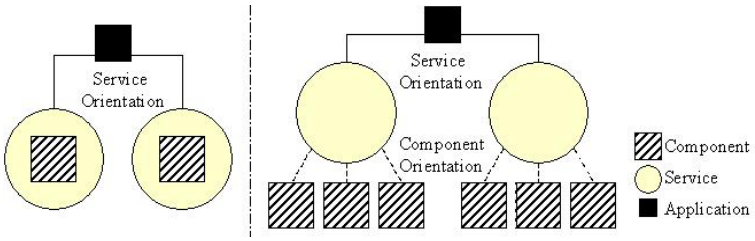
well as to adapt context changes. But this paper does not consider how the context is captured how the contents of the contexts are refined for the applications and so on. Therefore, it is conceptually located between the Aggregators layer and the Services layer of Context Toolkit as shown in Figs. 1 (a) and (b).

2.2.2 Gravity

Gravity [12] defines a component model, where the components provide and require the services and all the component interactions occur via the services. The goal is to support the construction and the execution of component-based applications that are capable of autonomously adapting at run time due to the dynamic availability of the services provided by constituent components. Dynamic availability refers to a situation where the services may become available or unavailable at any time during the execution of an application.

Gravity adopts the service to compose the components on demand at run time. The dynamic adaptation using the context driven components also adopts the service to make the application adapt to the context changes dynamically. Gravity focuses on developing the component-based applications, while this paper focuses on the context aware applications handling the dynamic adaptation and the context scalability in the ubiquitous environment.

Both of the Context Driven Components and the Gravity use both the components and the services for supporting adaptation. But there are some differences between them. Gravity makes the services of the components by wrapping the components with some descriptions supporting service orientation. It aims to adapt at run time due to the dynamic availability of the services provided by constituent components. This paper uses the components and the services separately on the different integration levels. It defines the context driven components and the context services, and then it makes a context service by integrating the context driven components. The point is the range of components and services that are defined according to context. The details are mentioned in section 3. Fig. 2 shows the difference of Gravity and the way using context driven components. The difference is the context driven component considers the context as an important criterion for defining the range of a component or a service. Therefore, it serves the adaptation and the scalability caused by the context in the ubiquitous computing.



(a) Gravity. (b) Context driven components.

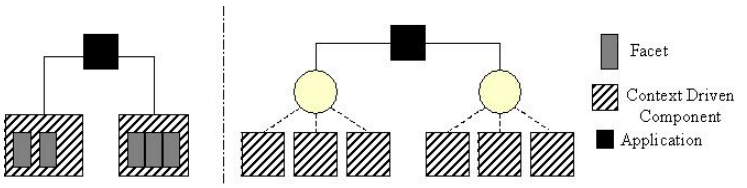
Fig. 2. Comparison of gravity and context driven components.

2.2.3 Multifacet approach

Multifacet [13] approach is to present an adaptation mechanism for context aware applications. It is able to centrally coordinate the triggering of the actions. Thus, an action gets triggered based on both its set of conditions and some other factors such as priority or the existence of some other actions currently triggered. It is based on the idea that an application consists of both components that are context sensitive and components that do not depend on the context, and a context sensitive component can be seen as an item with many facets. In an application, a facet has a condition that behaves like a switch; in the sense that if the condition is true the facet is exposed otherwise the facet is hidden.

Switching facets is our switching context-driven components for the adaptation of context changes. A facet included in a context sensitive component is on the same level of our context specific component. It means that the unit switched for the adaptation is implemented as a part of a component, which is a facet, not as a component. The multifacet approach solves the adaptation through reorganizing facets of a component, while our approach uses the composition of CBSD when solving the adaptation problem.

Fig. 3 shows the differences of the multifacets and the context driven components.



(a) Multifacet. (b) Context driven components.

Fig. 3. Comparison of multifacets and context driven components.

2.3 Ubiquitous Computing

Ubiquitous computing will help organize and mediate social interactions wherever and whenever these situations might occur. The idea of such an environment emerged more than a decade ago in Weiser's seminal article and its evolution has recently been accelerated by improved wireless telecommunications capabilities, open networks, continued increases in computing power, improved battery technology, and the emergence of

flexible software architectures. Consequently, during the next five to ten years, ubiquitous computing will come of age and the challenge of developing ubiquitous services will shift from demonstrating the basic concept to integrating it into the existing computing infrastructure and building widely innovative mass-scale applications that will continue the computing evolution.

The movement into the ubiquitous computing realm will integrate the advances from both mobile and pervasive computing. Mobile computing is fundamentally about increasing our capability to physically move computing services with us. As a result, the computer becomes a taken for-granted, ever-present device that expands our capabilities to inscribe, remember, communicate, and reason independently of the device's location. This can happen either by reducing the size of the computing devices and/or providing access to computing capacity over a broadband network through lightweight devices. Another dimension in making the computer invisible is the idea of pervasive computing. This concept implies the computer has the capability to obtain the information from the environment in which it is embedded and utilizes it to dynamically build models of computing [14].

The main challenges in ubiquitous computing originate from integrating large-scale mobility with the pervasive computing functionality. In its ultimate form, ubiquitous computing means any computing device, while moving with us, can build incrementally dynamic models of its various environments and configure its services accordingly.

3. CONTEXT DRIVEN COMPONENT

The main problem, which the existing ways have, is that they depend on static programming that cannot handle dynamic changes of contexts. This paper proposes an enhanced model to support such dynamic changes. It is Context Driven Component that enables context adaptation and content extensibility. Starting from introducing some essential concepts such as context, content and behavior, this section describes how the context driven component is implemented and how the component contributes to context adaptation and content extensibility.

3.1 Context, Content, and Behavior

Context is any kind of information that can be used to represent the status of an entity. An *entity* is a person, a place, or an object relevant to the interaction between a user and an application, including the user and the application themselves. By its nature, a context has specific content at a certain point of time. There is a pair of context and its content at every certain point of time. The pair is called *CCpair*.

CCpair determines what the service should behave to satisfy the condition of the pair. The service is named *Behavior*. Each *CCpair* has its own behavior. There are three types of behaviors; they are B_{simple}, B_{single}, and $B_{multiple}$. First, B_{simple} is a behavior decided by only one *CCpair*. B1, B3, and B5 in Fig. 4 are included in this type. Second, B_{single} is a behavior decided by more than one *CCpair* within a single context. B2 in Fig. 2 is an example of B_{single}. Third, $B_{multiple}$ is a behavior derived from more than one *CCpair* from multiple contexts. B4 in Fig. 4 is an example of $B_{multiple}$. Fig. 4 shows the relationship of contexts, contents, and behaviors.

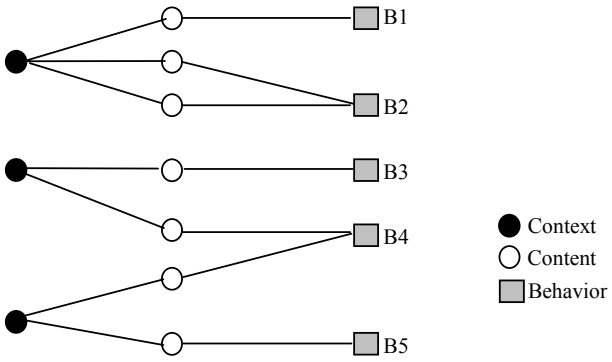
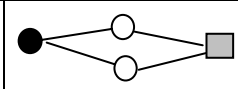
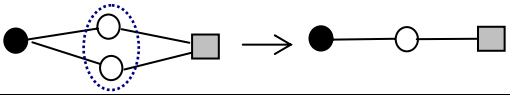
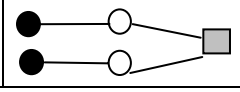
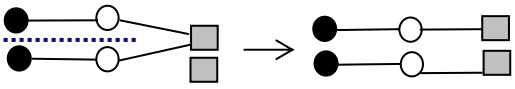


Fig. 4. *CCpair*s and behaviors.

No matter which one of contents associated to the B_{single} is activated, only one behavior, B_{single}, is selected and executed. It means B_{single} can be reformed to B_{simple} by merging all related contents into one group of content. $B_{multiple}$ also can be reformed to B_{simple} by dividing the behavior into multiple behaviors each of which is assigned to one content of a context or by duplicating the behavior and assigning each duplicated behavior to an associated content of multiple contexts. Table 1 shows the solutions described above graphically.

Table 1. The solutions to reform B_{single} or $B_{multiple}$ to B_{simple}.

Types of Behavior		Solution to make it B_{simple}
B_{single}		
$B_{multiple}$		

In this way, two non-simple types, B_{single} and $B_{multiple}$, need to be simplified to B_{simple} first, and then context driven components are extracted from the simplified behaviors. This approach may reduce the complexity of building the context drive components.

3.2 Context Independent Component and Context Specific Component

A context aware application consists of a context-independent part and a context-specific part. Therefore there are two types of context driven component. One is Context Independent Component and the other is Context Specific Component. Their definitions are as follows.

Definition 1 *Context Independent Component* (*CIC*).

The Context Independent Component, shortly *CIC*, is a component which implements a behavior not related to contexts. It does not have to consider any changes of contexts.

Definition 2 *Context Specific Component* (*CSC*).

The Context Specific Component, shortly *CSC*, is a component which implements a behavior related to contexts. It has to consider all changes of contexts.

UML sequence diagrams are used to build *CIC*s and *CSC*s. A sequence diagram is a kind of interaction diagrams, which describe how the objects are grouped to collaborate in some behavior. Typically, an interaction diagram captures the behavior of a single use case [15]. The diagram shows a number of example objects and the messages that are passed between theses objects within the use case. An application consists of use cases, and it consists of a group of sequence diagrams. Therefore, this paper uses a sequence diagram as a basic unit building a Context Driven Component.

This paper proposes a table called Sequence and Context Table (SCT). SCT shows relations between *CCpair*s and sequence diagrams. For each *CCpair* in SCT, all sequence diagrams related to the *CCpair* are marked gray as illustrated in Table 2.

Table 2. Sequence and context table.

	Sequence Diagram							
CCpair	1	2	3	4	5	6	…	*n*
$CCpair_{11}$			■					
$CCpair_{12}$				■				
$CCpair_{13}$					■			
$CCpair_{14}$						■		
$CCpair_{21}$			■	■	■	■		
$CCpair_{22}$								
$CCpair_{23}$								■

$CCpair_{11}$ in the table represents a pair of the context #1 and its content #1. And each sequence diagram is also numbered as 1, 2, 3, …, *n*. From SCT, all sequence diagrams related to the content of a specific context can be easily identified. All sequence diagrams related to one *CCpair* is implemented as a *CSC*. Therefore, the total number of contents of all contexts is equal to the total number of *CSC*s. The other sequence diagrams not related to any *CCpair*s are implemented in a *CIC* since they are all context-independent. For example, the sequence diagram 3 will be implemented in a *CSC* related to $CCpair_{11}$, and the sequence diagram 1 will be included in a *CIC*. As described in section 3.1, all *CCpair*s have been simplified to B_{simple} before the SCT is created.

The modularity of the context driven component must be considered since an application is made up of *CSC*s and a *CIC*. The better modularity has the higher cohesion. According to the cohesion levels of *G*. Myers [16], the context driven component is on the level of *Communication Cohesion,* the third highest level of the seven cohesions.

3.3 Context Adaptation

*CSC*s are located in a storage separated from Service as shown in Fig. 5. The separate storage is *CSC* storage in Fig. 5. In this figure, Service is the execution version of a context aware application, and it consists of more than on *CSC* and *CIC*. *CSC*s related to the new contexts are selected and transported to Service whenever the new contexts are arrived inside the range of service. Not all of an application, but only a part of the application is located in Service as a group of context driven components. Compared to *CSC*s, *CIC*s are not dependent on any contexts; therefore, all *CIC*s keep staying in Service. CSC Directory in Fig. 3 maintains the list of *CSC*s located in CSC storage. Manager allocates *CSC*s to Service according to the changes of contexts. It is a kind of main program to handle the context adaptation. Fig. 5 includes two kinds of XML documents; Service.xml and CSC.xml. Service.xml describes which *CSC*s are executed in Service currently. CSC.xml is a directory of *CSC*s. *CSC* is located in *CSC* storage physically. In an implementation model, *CSC* is as an executable unit such as *dll* in case of implementing it in .NET framework or *jar* in case of implementing it in Enterprise Java Beans (EJB) platform. Service has the instruction to call the executable unit, which is assigned to the service according to CSC.xml and Service.xml.

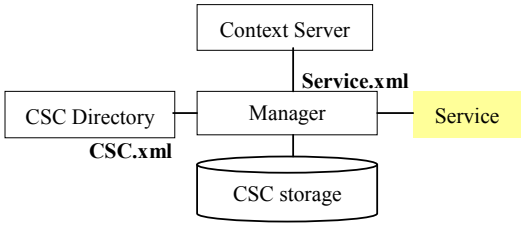


Fig. 5. Architecture for supporting context driven components.

At a certain point of time, an application has *CIC*s and *CSC*s activated as a Service of Fig. 3. When contexts are changed, the application should accept the changes by modifying some parts of the application. For this adaptation, rather than embedding hard-coded implementation for the changed contents, this paper uses an enhanced way only to replace the *CSC*s with the *CSC*s implementing the behaviors for the *CCpair*s of the changed contexts and the changed contents. The following describes the detailed steps.

ContextAdaptation
Input: Context information, Service.xml, CSC.xml
Output: Service.xml
Step 1. Get the context information from the Context Server
Step 2. If (Service.xml does not include the new context information
then
2.1 Register the new context information to Service.xml
Step 3. Find the id of the *CSC* execution unit implementing the new context in CSC.xml
Step 4. Register the id to Service.xml

3.4 Content Extensibility

In the enhanced way of implementation, an application can be easily extended by adding new behaviors for the changed contents of a context, with the help of concept of composing components in CBSD. This feature is *content extensibility*. With the help of content extensibility, when the content of a context is changed, rather than the changed content is hard-coded in the application, only new *CSC*s are developed and registered to CSC directory. Also when the content of a context is extended, the traditional method modifies the overall part of the application. But the new adaptive way implements the context driven component only for the behaviors of added contents and put the component in CSC storage, and then the behaviors of the added context can be implemented without changing the application itself.

Suppose a context, *People*, needs a new content, *Kate*, when the application with the context, People, has been already developed based on Context Driven Components. The way using the context driven component adds the behavior of the new content to the application without modifying the source code. It only makes a *CSC* that implements the behavior of the content, Kate, and then registers the *CSC* on *CSC* storage and *CSC* directory. The following describes the detailed steps.

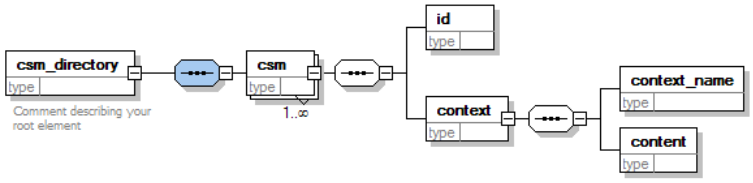
If a context, not a content, is added to an application, the algorithm described above runs repeatedly with all the *CCpair*s relevant to the added context. Suppose a context, time, is inserted to the application and the contents of the context are Morning, Afternoon, Night. In this case, there could be three *CCpair*s, and they are applied to the algorithm. Finally, three different *CSC*s are registered to *CSC* directory, then the *CSC*s for the added context, time, are available in constructing a Service or modifying it for the adaptation.

ContextExtension
Input: *CCpair*, Sequence Diagram
Output: *CSC*, CSC.xml
Step 1. If (the *CCpair* is not B_{simple})
then
1.1 Reorganize it B_{simple} through Table 1.
Step 2. Get the sequence diagram for the *CCpair*.
Step 3. Make a *CSC* implementing the sequence diagram.
Step 4. Put the *CSC* in *CSC* storage.
Step 5. Register the *CSC* to CSC.xml

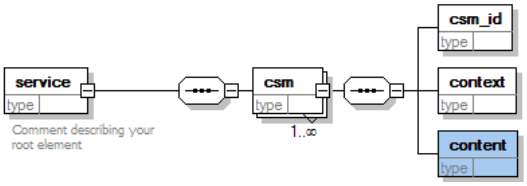
3.5 CSC.xml and Service.xml

*CSC*s are described in CSC.xml, and the configuration of the current services is described in Service.xml. The XML schemas for these xml documents were built on XML Spy. Fig. 6 shows the XML schemas.

CSC.xml has <csm_directory> as a root element, and one or more <csm>, which represents information of a *CSC*, can be located in a csm_directory. Each *CSC* has its own id in <id>, and it has information of the context and the content associated with



(a) CSC.xsd



(b) Service.xsd

Fig. 6. XML schemas of CSC.xml and Service.xml.

the *CSC*. Service.xml has <service> as a root element, and it has one or more <csm> representing information of *CSC*s that are currently loaded in Service. Service.xml is modified when the application is adapted to the context changes, and CSC.xml is modified when *CSC*s implementing new content are added for supporting the content extensibility.

4. AN EXAMPLE

This section shows the full steps to build the context driven components and adapt to context changes with Call Forwarding [17].

4.1 Call Forwarding

Call Forwarding was published in 1992. It was based on the Active Badge system. The location context is presented to the receptionist who routinely forwards the telephone calls to the nearest phone of the destination user (location context is used passively to display to receptionist). Recently, with the help of a PBX that has the digital interface designed for computer-integrated telephony, the location context becomes active to help automatically forward the phone calls. This application proves to be useful to both the staff and the telephone receptionist. It was observed, however, that people want to express more control over when calls are forwarded to them, depending on their current context; for example, they prefer not to take unexpected calls when having a meeting with their boss.

We have added a context related feature to the original call forwarding system in order to handle the variety of available contexts. The added context is the destination user. Depending on the destination user, the ringer of a forwarded call is different. If the destination user is Tom and Tom's favorite ringer is Bell 2, the call is forwarded to

Tom's nearest phone with Bell 2. It helps Tom recognize that the call is for himself. As mentioned earlier, all of the *CCpair*s need to be reformed to B_{simple} before making SCT. All the behaviors in Table 3 are in B_{simple}. The two contents of Destination User, Austin and Tim, have the same behavior. The relation was B_{single}. It is changed to B_{simple} by merging the contents.

Table 3. Contexts, contents, and behaviors in call forwarding.

Context	Content	Behavior
Location	A	Forward the call to the phone in the area A
	B	Forward the call to the phone in the area B
	C	Forward the call to the phone in the area C
	D	Forward the call to the phone in the area D
Destination User	John	Ring the bell type 1
	Brian	Ring the bell type 2
	Austin, Tim	Ring the bell type 3

4.2 Context Driven Components

Sequence diagrams are used to divide an application into Context Driven Components. Call Forwarding was designed through Rational Rose Enterprise Edition, and 17 sequence diagrams were generated as shown in Table 4. The diagrams are grouped by SCT and then a group is assigned to a *CCpair*. Absolutely, we use a sequence diagram as a basic unit building a context driven component, but we don't manipulate or analyze the

Table 4. Sequence diagrams in call forwarding.

#	Sequence Diagrams	#	Sequence Diagrams
1	Collect a Call	10	Forward Calls_Brian&PhoneD
2	Get Information	11	Forward Calls_AustinTim&PhoneA
3	Forward Calls_John&PhoneA	12	Forward Calls_AustinTim&PhoneB
4	Forward Calls_John&PhoneB	13	Forward Calls_AustinTim&PhoneC
5	Forward Calls_John&PhoneC	14	Forward Calls_AustinTim&PhoneD
6	Forward Calls_John&PhoneD	15	Get the phone_John
7	Forward Calls_Brian&PhoneA	16	Get the phone_Brian
8	Forward Calls_Brian&PhoneB	17	Get the phone_AustinTim
9	Forward Calls_Brian&PhoneC		

Table 5. *CCpair*s in call forwarding.

CCpair	Context	Content	*CCpair*	Context	Content
$CCpair_{11}$	Location	A	$CCpair_{21}$	Worker	John
$CCpair_{12}$	Location	B	$CCpair_{22}$	Worker	Brian
$CCpair_{13}$	Location	C	$CCpair_{23}$	Worker	Austin or Tim
$CCpair_{14}$	Location	D			

information of the diagram. It means that the contents inside the diagram are not affected to the building process of context driven components. To make the sequence diagrams and *CCpair*s expressed for short in SCT, the sequence diagrams are numbered in Table 4, and *CCpair*s are named in Table 5.

The SCT is made of the sequence diagrams and *CCpair*s in the two tables described above. Table 6 is the SCT built from Call Forwarding.

Table 6. SCT of call forwarding.

	Sequence Diagram																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$CCpair_{11}$			■				■				■						
$CCpair_{12}$				■				■				■					
$CCpair_{13}$					■				■				■				
$CCpair_{14}$						■				■				■			
$CCpair_{21}$			■	■	■	■									■		
$CCpair_{22}$							■	■	■	■						■	
$CCpair_{23}$											■	■	■	■			■

```
<csm_directory>
<csm>
<id> 11 </id>
<context>
<context_name> location </context_name>
<content> A </content>
</context>
</csm>
<csm>
…
<csm>
<id>21 </id>
<context>
<context_name> Destination User </context_name>
<content> John </content>
</context>
</csm>
<csm>
…
<id>23 </id>
<context>
<context_name> Destination User </context_name>
<content> Austin </content>
</context>
</csm>
<csm>
<id>23 </id>
<context>
<context_name> Destination User </context_name>
<content> Tim </content>
</context>
</csm>
</csm_directory>
```

Fig. 7. *CSC*.xml of call forwarding.

Each *CCpair* is a *CSC*, and the *CSC* is implemented by coding the sequence diagrams that are marked as gray in the *CCpair*'s low. For example, *CSC* of $CCpair_{11}$ is a component executing the sequence diagrams 3, 7, and 11. *CIC* implements the other sequence diagrams that are not included in any *CSC* of the *CCpair*s. Finally, seven *CSC*s and one *CIC* are made from SCT of Table 6. The seven *CSC*s are registered to CSC Directory of Fig. 5, and *CIC* stays in Service of Fig. 5. To mange CSCs located in in CSC storage, CSC.xml is created as followed in CSC directory. Fig. 7 is CSC.xml of Call Forwarding based on the XML schema of Fig. 6 (a).

Suppose that the current service that is implemented by *CIC* and some *CSC*s is to forward a call to John in location A. Service.xml for the current service description reads as follows. It is based on the XML schema of Fig. 6 (b).

4.3 Context Adaptation

According to the steps for context adaptation described in section 3.3, this example would adapt when a content of contexts is changed. In this case, the location of John is changed A to B. The context adaptation steps are as followed. The current *CCpair*s are, for example, (Location, B) and (Destination User, John), which means the service forwards the call to John in location B. Manager compares the changed content to the content in Service.xml, and updates Service.xml if there is founded any differences. As the current content of the context, location, in Service.xml of Fig. 8 was *A* (<context> Location</context><content>A</content>). The new content of the context is *B*. Service.xml is updated to "<context>Location</context><content>B</content>". Manager retrieves the id of CSC implementing newly updated content, and add it to Service. The changed Service.xml reads in Fig. 9. After the context adaptation, Service forwards the call to John in location B not in location A.

```
<service>
<csm>
<csm_id> 11 </csm_id>
<context> location </context>
<content> A </content>
</csm>
<csm>
<csm_id> 21 </csm_id>
<context> Destination User </context>
<content> John </content>
</csm>
</service>
```

Fig. 8. One of the *Service.*xml describing call forwarding.

```
<service>
<csm>
<csm_id> 12 </csm_id>
<context> location </context>
<content> B</content>
</csm>
<csm>
<csm_id> 21 </csm_id>
<context> Destination User </context>
<content> John </content>
</csm>
</service>
```

Fig. 9. Service.xml updated for adapting the context change.

5. DISCUSSION

This section discusses the context driven component in four subjects; they are the scale of context, a comparison to Context Toolkit, the advantages of the Context Driven Component, and the implementation in Ubicomp. The first subject says that the bigger the size of the context is, the more the context-oriented component is worth. The second subject is the comparison to the existing way based on the Context Toolkit. The third one is the advantage of the Context driven components compared to the rule-based systems.

5.1 The Scale of Context

The context driven component proposed in this paper adapts context changes by reconfiguring the service dynamically. If the number of *CSC*s is considerably bigger than the number of *CIC*s in an application, then handling the context driven component could rather make overload in transferring data. There are two criteria to evaluate whether an application is adequate to use the context driven component or not. One of them measures how much the application is related to context, which is defined as *CSCScore*. The other criterion measures how many kinds of contexts are related to the application, which is represented as |*CCpair*|. Namely, *CSCScore* is the number of sequence diagrams related to at least one *CSC* over the number of all sequence diagrams, and |*CCpair*| is the total number of *CCpair*s. The higher the *CSCScore* and |CCpair| are, the more the application is adequate to use the context driven component. As shown in Fig. 8, applications are categorized into our different cases depending on whether the values of *CSCScore* and |*CCpair*| are high or low.

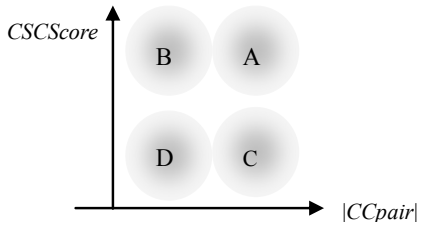


Fig. 10. Four cases of applications by *CSCScore* and |*CCpair*|.

Applications included in the area A or the area C of Fig. 10 has the high |*CCpair*|, which means they consider various contexts. For instance, the application considers the context information of who entered, where the person is located, and what time it is simultaneously. And applications included in the area A or the area B of Fig. 10 has the high *CSCScore*, which means they are much specific to contexts according to the definition of *CSCScore*. For instance, the application has completely different behaviors depending on who entered the range. The portion of *CIC* of the application is almost nothing compared to the portion of *CSC*. The other area D has low |*CCpair*| and low *CSCScore*; it can be said to be definitely not a context-sensitive application.

The context driven component is good to be applied to an application having high CSCScore and high |*CCpair*|, since it focuses on how to build and maintain the behaviors

relevant to context. Therefore, the context driven component is getting worthy on context adaptation and content extensibility as applications are getting closer to the area A of Fig. 10.

Moreover, the applications in Ubicomp are required to accept various and delicate contexts existing in the real world. In this point, the applications should be going to the area A of Fig. 10. It means the approach using the context driven component also can contribute to develop Ubicomp applications.

5.2 Vertical Viewpoint Compared to Context Toolkit

Context Toolkit [11] arranges the layers horizontally from a context-biased layer to a service-biased layer as shown in Fig. 11. From top to bottom, the layer is getting context-biased, and from bottom to top, the layer is getting service-biased. Context Toolkit starts from receiving context from sensors, and the context is refined, and finally the context is applied to a service. Context Toolkit is a framework to build context aware applications that execute the whole process from receiving data from sensor to handling services. It did not suggest how to handle the context adaptation. However, this paper has developed Context Driven Component to build context aware applications with focusing on the context adaptation and the content extensibility. It does not consider how the context is captured, how the contents of the contexts are refined for applications and something like that. Therefore, the context driven component is conceptually located between the Aggregators layer and the Services layer of Context Toolkit as shown in Fig. 11.

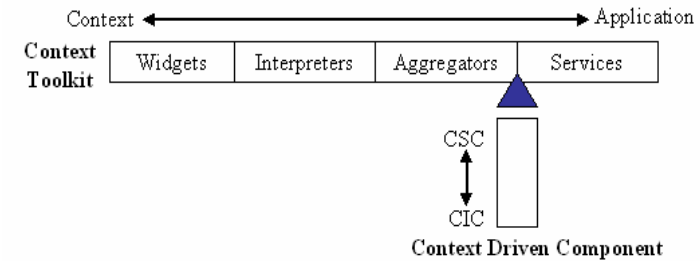


Fig. 11. Horizontal layers vs. vertical layers

*CSC*s are arranged vertically just over *CIC*s. If the context driven components can be inserted as a layer between Services and Aggregators in Context Toolkit, a new context-based software development would be created with two diverse viewpoints; one is horizontal for covering all process to handle context awareness, the other is vertical for handling context sensitive components to accept context changes. It means that Context Toolkit and Context Driven Component complement each other.

5.3 Advantages Compared to Rule Based System

A context aware application typically has rules governing how the application should respond to context changes [8]. It is a rule-based system. From a software engi-

neering perspective, the rule-based programming is a failure because the systems developed in it are not able to maintain, test and reliable [7]. Instead of the rule-based system, this paper proposed the context driven component to build the context aware applications. The proposed content is based on the concept of existing component-based software development. The CBSD is planning to assemble the existing software component and produce a larger piece of software. The treatment of components as plug-replaceable units is a simplistic view of system evolution. It helps to maintain the application by replacing of outdated components with new ones. This paper explains that replacing of components by new ones results in the context adaptation and the content extension. In practice, replacing a component may be a non-trivial task, especially when there is a mismatch between the new component and old one. But the context driven components result from the design artifacts for one application. The components are also implemented under the same development environment. Therefore, replacing components are not difficult that much. Moreover, CBSD promises to reduce development costs by enabling rapid development of highly flexible and easily maintainable software systems. Using the context driven components expects the advantages taken from CBSD.

5.4 Implementation in Ubicomp

According to the architecture shown in Fig. 5, the service is only for the current point of time, and the rest parts of the behaviors are stored in CSC Storage. For Ubiquitous computing, an application needs to have both Embeddedness and Mobility [14]. For Embeddedness, the whole set of components in Fig. 5 need to be embedded onto Ubicomp devices (or appliances). They can be implemented using the traditional Client/ Server way, but it does not meet the rule of Embeddedness that Ubicomp requests. Embeddedness means all software elements should be located in the devices. To support Mobility, the *battery usage* should be considered in the hardware side, and *instant-on* should be supported in the software side. *Instant-on* means that services or applications must be always on, always accessible, but must not demand explicit attention of users. The service in Fig. 5 will be loaded on memory space as PDA loads its operating system on ROM to enable instant-on.

Only the service including some Context Driven Components stays in memory for the instant-on, and the other part of the application is on disk not on memory because it does not need to be ready to run. It helps the size of memory reduced, and it consequently results in saving battery. The higher $|CCpair|$ could induce the battery saving effect higher.

6. CONCLUSION AND FUTURE WORK

This paper has developed Context Driven Component to handle the context changes dynamically. The applications using the context driven component can be adapted to context changes by replacing the components implementing old contents to components implementing new contents, and they can be extended by adding a component implementing a new content. The context driven component supports the context adaptation and the content extension. These two qualities are requested especially in Ubicomp since

the real world Ubicomp is running can be changed more dynamically than the traditional software domains.

The development using the context driven component has some contributions as mentioned in section 5. The context adaptation and the content extensibility through the context driven components are worthy in the applications having the high *CSCScore* and |*CCpair*|. And it focuses on handling context changes with slicing applications vertically depending on contents of contexts, while Context Toolkit focuses on developing applications handling the whole process from capturing contexts to using them in services with slicing application to some layers horizontally. In addition, the context driven component is a module of an application according to CBSD paradigm, which makes the maintenance of the application easier than the rule based system that the traditional context aware application is in. The context adaptation and the content extension can be sorts of the maintenance. Also, the approach holding a part of components in memory not all the components of an application helps instant-on and battery-saving requested by Ubicomp environment.

The experimental study on the scale of context introduced in section 5.1 is the first future work. If the scale of context is small, the static development using rule-based systems, which contains all behaviors of contexts in an application, may be better than the development through the context driven component in considering the cost of transferring components. The static development does not need any specific process for accepting context changes. It may have a low adaptation complexity. If the scale of context is getting bigger, the codes for the extended contexts of the static application grow. The static development with large scale of contexts may trace the whole codes of the application knottily, and it increases the adaptation complexity more than the approach of the context driven component. If the scale of context is getting bigger, while only the *CSC*s of extended contexts are added to CSC storage in the case of using the context driven components. It does not increase the adaptation complexity even when the size of the context grows, only the *CSC*s for the extended contexts and the size of CSC storage are increased. It still adapts with following the same steps having the same complexity. Through the experimental experiment, the value, x, in Fig. 12 will be founded. The value will help developers decide whether the application is adequate to use the context driven component proposed in this paper.

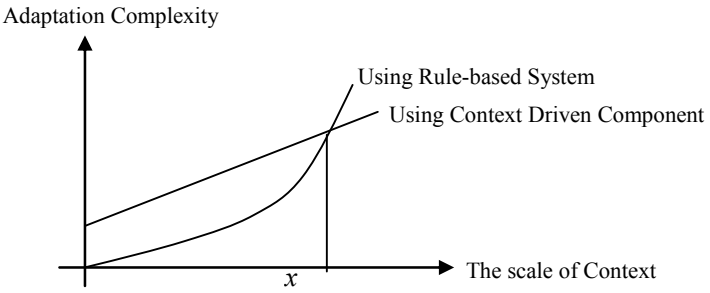


Fig. 12. Adaptation complexity vs. the scale of context.

REFERENCES

1. G. D. Abowd, "Software engineering issues for ubiquitous computing," in *Proceedings of International Conference on Software Engineering*, 1999, pp. 75-84.
2. W. N. Schilit, N. I. Adams, and R. Want, "Context-aware computing applications," in *Proceedings of the 1st International Workshop on Mobile Computing Systems and Applications*, 1994, pp. 85-90.
3. P. J. Brown, J. D. Bovey, and X. Chen, "Context-aware applications: from the laboratory to the marketplace," *IEEE Personal Communications*, Vol. 4, 1997, pp. 58-64.
4. T. Gu, K. Pung, and D. Q. Zhang, "A service-oriented middleware for building context aware services," *Journal of Network and Computer Applications*, Vol. 28, 2005, pp. 1-18.
5. S. S. Yau and F. Karim, "An adaptive middleware for context-sensitive communications for real-time applications in ubiquitous computing environments," *Real-Time Systems*, Vol. 26, 2004, pp. 29-61.
6. B. Schilit, *et al.*, "Disseminating active map information to mobile hosts," *IEEE Networks*, Vol. 8, 1994, pp. 22-32.
7. X. Li, "What's so bad about rule-based programming," *IEEE Software*, Vol. 8, 1991, pp. 103-104.
8. A. K. Dey, G. D. Abowd, and M. Futakawa, "The conference assistant: combining context-awareness with wearable computing," in *Proceedings of the 3rd International Symposium on Wearable Computers*, 1999, pp. 21-28.
9. J. Pascoe, "The stick-e note architecture: extending the interface beyond the user," in *Proceedings of the 2nd International Conference on Intelligent User Interfaces*, 1997, pp. 261-264.
10. S. Fickas, *et al.*, "Software organization for dynamic and adaptable wearable systems," *First International Symposium on Wearable Computers*, 1997, pp. 155-160.
11. A. K. Dey and G. D. Abowd, "The context toolkit: aiding the development of context-aware applications," in *Proceedings of Human Factors in Computing Systems*, 1999, pp. 434-441.
12. H. Cervantes and R. S. Hall, "Autonomous adaptation to dynamic availability using a service-oriented component model," in *Proceedings of the 26th International Conference on Software Engineering*, 2004, pp. 614-623.
13. A. Rarau, K. Pusztai, and I. Salomie, "Multifacet item based context-aware applications," *International Journal of Computing & Information Sciences*, Vol. 3, 2005, pp. 10-18.
14. K. Lyytinen and Y. Yoo, "Issues and challenges in ubiquitous computing," *Communications of the ACM*, Vol. 45, 2002, pp. 63-65.
15. M. Fowler, *UML Distilled*, Addison-Wesley, 1997.
16. G. Myers, *Reliable Software through Composite Design*, Mason and Lipscomb Publishers, New York, 1974.
17. R. Want, A. Hopper, V. Falcao, and J. Gibbons, "The active badge location system," *ACM Transactions on Information Systems*, Vol. 10, 1992, pp. 91-102.

Hoijin Yoon is a full time lecturer in the Department of Computer Science and Engineering at Ewha Womans University. She received the B.S. degree in Computer Science from Ewha Womans University and the M.S. and Ph.D. degrees in Computer Science and Engineering from Ewha in Korea. Her research interests are in software engineering with particular emphasis on testing component-based software, development of context aware software, and ubiquitous software engineering.

Byoungju Choi is a Professor in the Department of Computer Science and Engineering at Ewha Womans University. She received the B.S. degree in Mathematics from Ewha Womans University and Computer Science at Purdue University in USA; and the M.S. and Ph.D. degrees in Computer Science from Purdue University in U.S.A. Her research interests are in software engineering with particular emphasis on software testing, embedded software testing, software process improvement.