Journal of Knowledge Information Technology and Systems

ISSN 1975-7700

http://www.kkits.or.kr

A Study on an Object-orientation and Extensibility in Context-aware Systems

Jongmyung Choi[1], **Iksu Kim**[*2]

[1]*Department of Computer Engineering, Mokpo National University*
[2]*School of Computer Science and Engineering, Soongsil University*

ABSTRACT

There has been a lot of research on context-aware computing, but software extensibility in the development of those systems has not gotten much attention in spite of its importance in software engineering. In this paper, we introduce some extension requirements for context-aware systems, and identify four extension types for them: sensors, context inference algorithms, contexts, and context-aware services. For those extension requirements, we propose four extension mechanisms based on object-oriented technology: separation between abstraction and implementation of context, separation of context from sensors, modular and separate model for context-aware functions, and overloading model for context-aware functions. By adopting those mechanisms, developers or maintainers can add new sensors, context inference algorithms, contexts, or context-aware services without modifying the source code after deployment. Those mechanisms are all based on object-orientation. Our approach represents a context as a class, and context services as methods with context parameter. Context inference is a class with a method which understands sensor values and infers the current context from the values. This approach will reduce costs, time, and efforts in context-aware system maintenance which requires new context-aware features after deployment because it will increase the software reusability and extensibility. In this paper, we also specify a case study which shows how to extend a context-aware system with the extension requirements.

KEYWORDS: Context-aware, Extensibility, Extension mechanism, Object-orientation, Context-model

ARTICLE INFO: Received 16 March 2017, Revised 3 April 2017, Accepted 7 April 2017.

[*]Corresponding author is with the School of Computer Science and Engineering, Soongsil University, Sangdodong, Dongjak-gu, Seoul, 06978, KOREA.
E-mail address: iksuplorer@ssu.ac.kr

1. Introduction

"Extensibility" is one of the most important properties in modern software [1], because nowadays software is considered as "evolving entity" which meets continuously changing user requirements [2]. In software development and maintenance, "extensibility" allows developers or maintainers to add new features or modify the existing features for user requirements without modifying the source code [3]. This feature extension or enhancement happens all the times in software lifecycle, especially in maintenance phase. According to Glass [4], about 40 to 80 percent (average 60 percent) of software costs is consumed for the maintenance, and about 60 percent of activity in the maintenance is about feature enhancement rather than bug or error correction. Before Glass's study, Lients and Swanson [5] studied on 487 IT organizations, and they also found that about 51% of maintenance activities were caused by new functional or nonfunctional requirements. Because of its importance, there has been research on "extensibility" mainly in software engineering and programming language. Furthermore, a lot of software product including even open source projects [6][7] adopt extension mechanism, for example extensible architecture, for their future requirements.

There has been research on various topics for context-aware systems and computing. However most of the research has focused on context modeling, context reasoning, or prototype systems, and they have not paid much attention to "extensibility" issues [8]. However, in order to prepare the future where context-aware services are used everywhere, we have to pay attention to extensibility, and identify context-aware system specific extensibility issues. We believe that there are context-aware system specific extensibility issues and extension mechanisms for them. For example, we have some extension related questions as follows. 1) What happens if a new type of context is required to be added to context-aware systems? 2) What if new context-aware services are required to be added to the systems? 3) What if new types of sensors are required to be added for detecting contextual data more efficiently? 4) What if traditional services should be converted into context-aware services? Those questions are related with contexts or context-aware services, and they should be answered to increase extensibility and to reduce maintenance cost.

In this paper, we introduce some software extension requirement scenarios caused by "context", and several extension mechanisms for context-aware systems. We classify the context related extension scenarios into four types: sensor extension, context inference extension, context extension, and context-aware service extension. Those extension requirements happen frequently in software development and maintenance phase. We also propose our extension mechanisms. For context, we propose the separation between abstraction and implementation for context, and the separation of context from sensors. This approach allows us to define contexts to be highly abstract enough to be understandable by

people. Furthermore, it decouples context from sensors, so sensors can be added or replaced without changing the contexts. For context-aware services, we propose Separate Model and Overloading Model. Those models mean that a context-aware service has one name and it has multiple implementations with overloading methods which have contexts with their parameters. Those models allow new contexts and context-aware services to be added after deployment without modifying the source code. The extension mechanisms proposed in this paper are based on the features of Object-Oriented Programming. Therefore, we do not need any specific libraries or platform to apply our proposal.

The contribution of this paper is the definition of characteristics of software extension mechanisms in context-aware systems. Additionally, this paper gives examples of existing software with extension mechanisms and gives guidelines for introducing them in an own product. Furthermore it introduces different extension mechanisms and gives an overview on important aspects when introducing an extension mechanisms in own software. Software extension mechanism in context-aware systems will reduce time, efforts, and costs in software maintenance when new context-related features or modification are required.

The remainder of this paper is structured as follows: In Section 2 we will survey the existing research which are related to software extension and context-aware systems considering extension. In Section 3 we introduce some extension scenarios and extension types in context-aware systems. After then, we propose our extension mechanisms for context-aware systems in Section 4. Furthermore, we introduce some example scenarios for software extension with a sample context-aware system in Section 5. Finally, in Section 6, we reveal our conclusions and sum up the results of this paper.

2. Related Works

As far as our knowledge, no paper has been found that directly deals with software extension mechanisms for context-aware systems. In this section, we summarize research on software extensibility and introduce some context-aware systems which consider their extensibility.

2.1 Extensibility in Software Engineering

Software extension is very common in development and maintenance phase and it requires huge cost. Therefore, “extensibility” is one of the most important properties of modern software. According to Lients and Swanson’s study [5] on 487 IT organizations, about 51% of maintenance activities were caused by new functional or nonfunctional requirements. Glass [4] also argued that about 40 to 80 percent of software cost was consumed for the maintenance, and about 60 percent of maintenance costs was consumed in software extension. Those studies showed two things. First, the main activities in the maintenance are about adding new functions or modifying the existing software. Second, the

activities consume huge costs. Notkin [9] also argued that software enhancement is the most costly activity in software development lifecycle. To make things worse, typical software applications have a number of releases ahead of their retirement, and this increases the need of software maintenance. They need to maintain properly so that a subsequent evolution can easily modify a baseline version to fulfill the new or modified requirements.

How can we increase the extensibility? Due to the importance of software extension, there have been studies to increase the extensibility. In 1970s, Pan [10] introduced the concept of software extension and proposed some design guidelines for software extension. In 1990s, Object Oriented Programming (OOP) became popular and some mechanisms for software extension were introduced. Those mechanisms include abstraction, subtyping, and polymorphism. Meyer [11] proposed "Open-closed Principle" in his book, and Liskov proposed "Substitution Principle" [12]. Furthermore, design patterns were introduced, and many of them were elaborated to enhance software extension. Ng [13] introduced the result of his experiment for identifying the relationship of design pattern and the extension. Remco [24] introduced a software platform, named BEAST 2, which allows new third party features to be added to the system. It supports "post-deployment extensibility" by managing its features as modules.

From the existing research, there is some known knowledge for extensibility. The knowledge will help developers design and maintain the system efficiently.

F1. Software extensibility has been considered in various aspects in software development. For example, there has been research on software architecture, frameworks or toolkits, and programming models to improve extensibility.

F2. Software extension mechanism has close relation to programming paradigm. In structured programming, Pan[10] proposed extension mechanism. However, in object oriented programming, the features of object oriented programming such as polymorphism and "Substitution Principle" are used as software extension mechanisms.

F3. Design pattern is helpful for extensibility because it provides flexibility for unexpected new requirements [14]. By applying design patterns, maintenance efforts can be lowered.

F4. If the future can be predicted, we can design the system to be prepared for the future requirements.

F5. Extensibility promotes software reuse. Software enhancement means that reusing the existing software. Therefore, they have much in common: abstraction, separation of concerns, modularity, and high cohesion.

2.2 Extensible Context-Aware Systems

Until now, there have been various researches on context-aware computing and systems. However, most of the researches have focused on context modeling, context inference, or prototype systems. In addition, most of context-aware systems have focused on their services, but they

have not paid much attention to extensibility [8]. Even though there has been no research which deals with software extension mechanisms for context-aware systems, but there has been some research which provided extensibility properties to context-aware systems. Those researches included extensible software architectures, frameworks, or programming models.

Software architecture is important in system design, and there has been research on software architecture for extensible context-aware systems. [15] proposed an extensible the Pipe-and-Filter architecture for context-aware framework. [16] proposed WCAM (Watcher, Controller, Action, and Model) architecture to reduce context-aware system's complexity and increase extensibility. The architecture decoupled context concerns and service concerns by dividing a system into four sub-components: watcher, controller, action, and model.

Frameworks or toolkits for context-aware systems have considered extensibility because they have to consider requirements for various context-aware services. Fahy [17] suggested a context-aware middleware (CASS) that is flexible and extensible in its context handling. CASS separates context management from the context-aware application code. Therefore, this allows the context inference and its behavior to be changed without re-compilation. Athanasopoulos[18] proposed "dynamic discovery of context sources" in their middleware, CoWSAMI. Henricksen [19] argued that addition of new "fact" types and "situations" was possible with small amount of code change.

Some literatures provide good reviews on context-modeling and implementation of the systems. The survey by Kapitsaki [20] focused on "source code level programming/language extensions", "model-driven approaches", and "message interception."

2.3 Context Models

Modeling contexts is an important issue in the context-aware systems because the implementation of context-aware services is inevitably based on the context models. Therefore, context models are directly related to important sub-components including context inference, context data management [21], and context-aware services. There has been much research on context modeling, and Strang and Linnhoff-Popien [22] surveyed the existing context models and classified them according to their data structures. They classified context models into six types as follows:

- Key-value Model: This model utilizes key-value pair for contextual information and its value. This model is very simple, and its key-values are directly used in implementation. However it is hard to extend the system’s features for new requirements without modifying the source code.
- Markup Scheme Model: This model represents contextual information with XML style markup scheme. Therefore it has hierarchical structure and it is possible to represent recursive data. Normally the structure and vocabulary for the markup scheme are

fixed, but some allow flexibility. Markup processing modules are used to interpret contextual information.

- Graphic Model: This is rather conceptual model for context modeling. The typical models in this category are UML extension, Object-Role Model, and Entity-Relationship Model. This model is good for context model but it should be converted into other data structure for implementation.
- Object-oriented Models: This model utilizes the advantages of object-oriented paradigm. Therefore contextual information processing is performed in the object and is hidden to outside of the component. The access to the contextual information is performed through the open interface. Object-oriented models are good for extension because of its inheritance mechanism.
- Logic-based Models: This approach is based on mathematical predicate logic and it models contexts as fact, expression, or rules. The context inference is performed by formal inference model, and the inference is based on the rules. The inference algorithm is fixed but the rules can be added dynamically.
- Ontology based Models: This model is good for describing concepts and interrelationship among them. Ontology model is flexible to describe the basic concepts, arbitrary amount of sub-concepts and facts. Furthermore, ontology is inferred by using ontology inference engines. Meditskos[26] adopted ontology model for their system, and interpretation techniques for contexts.

In context model, there are some important properties which context model should have: abstraction, concrete, reuse, programmable, and extensibility. Context model should be both abstract and concrete. It should be abstract enough for people to understand, and it also should be concrete enough to access detailed information about the situation. Context model should be reusable, because the modeling and inference technologies are important reusable components. The context model should be extensible. Developers want to add or modify the exiting context models to enhance their systems. Finally, context model should be naturally and easily implemented in programming language. <Table 1> shows the properties of each context model.

표 1. 상황 모델 특성

Table 1. Context model's properties

Models	Abstraction	Concreteness	Reuse	Programmability	Extensibility
Key-value	X	◎	X	◎	X
Markup	X	O	O	O	O
Graphic	◎	◎	X	X	X
Object	◎	◎	◎	◎	◎
Logic	◎	◎	◎	◎	O
Ontology	◎	◎	◎	O	◎

◎: Very good / O: Good / X: Not support

3. Extension Types

3.1 Context-aware System Model

The existing research has proposed various

architectures and components for context-aware systems. Those systems are different but they have some components in common: contextual data sensing, context management, and context-aware services. <Figure 1> shows a context-aware system model which consists of three sub-components.

그림 1. 상황인지 시스템 모델

Figure 1. A Context-aware system model

Sensor Management component acquires contextual data such as user's location, the current time, and other sensor data. With the popularity of sensors or small devices, interesting contextual information is acquired. The number of sensors tends to increase to identify user's situations or behaviors.

Context Management component is the part which gathers sensor data, stores the data, and infers the current context. In this component, the inference algorithm and its implementation are important. The inference algorithm is related to the context model and programming model used in the context-aware system implementation.

Context-aware Services component is the services which the systems provide to users according to the current context. The services may be predefined at the system design time or dynamically adaptive at runtime.

3.2 Extension Requirements Scenarios

In order to understand the requirements for software extension in context-aware systems, we adopt a context-aware service example. Kapitsaki [20] illustrated a tourist guide system in order to compare programming techniques for context-aware systems. Actually there have been several studies on context-aware tour guide systems or exhibition guide systems, but Kapitsaki's tour guide system will be a good starting point to understand context-aware services and systems. The tour guide system is a simplified version, but it has the features which are required for context-aware systems: context information and context-aware services. <Figure 2> shows Kapitsaki's tour guide system, which has two context-aware services: greeting and attractions recommendation. In the system, the greeting service considers only user's language for contextual information, and attractions recommendation service utilizes two types of contextual information: user's location and weather information.

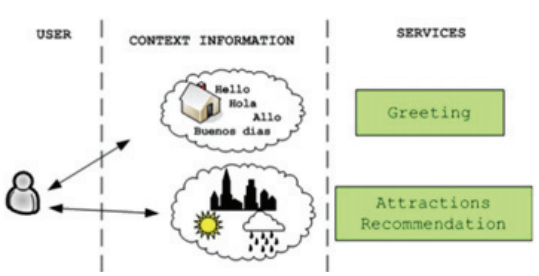

그림 2. 여행 가이드 시스템

Figure 2. A tour guide system(from [20])

As the traditional systems have requirements for software extension, the context-aware systems also have similar requirements. In addition, the context-aware systems have extra extension requirements, which are related to the context. Let us identify the extension requirements using the tour guide system. The following extension scenarios can be applied to development phase or/and maintenance phase of the system. We need to extend the system after the scenarios.

Scenario 1.

The CEO of CA Tour Inc. is happy with its context-aware tour guide services. He wants to add time information for greeting, so it can greet more time properly, for example, "Good Morning" or "Good Afternoon". Furthermore, he also wants to replace the current location sensors with GPS based sensors.

Software enhancement requirements for scenario 1 are very common and frequently happen. In order to provide more enhanced services, more contextual information (i.e. current time) may be required. Furthermore, the existing contextual data source (i.e. sensors) can be replaced with other types.

Scenario 2.

The CEO of CA Tour Inc. wants to expand its service for French tourists. Furthermore, he also wants to convert its existing "Take Picture" service into context-aware service.

As software system evolves, the requirements similar to Scenario 2 also frequently happen. In order to provide service to new target users, we have to develop a new service for French. Furthermore, we also have to convert the traditional service "Take Picture" into a context-aware version without modification of the existing source or with minor modification.

3.3 Extension Types

Software extension issues which are related to context locate three parts in the context-aware system model in <Figure 1>. This model-based approach will be helpful in scrutinizing issues in the realistic systems. For maintenance or upgrading systems, the extension of the existing systems always happens. In context-aware systems, there are four kinds of extension with respect to context-awareness: sensors, context inference algorithms, contexts, and context-aware services.

- Sensor Extension: Sensors are the basic parts in context-aware systems because they perceive contextual information from the environment. However, the sensors can be replaced or added because new types of sensors are being developed.
- Context Inference Extension: The context inference algorithm varies according to context models, quality of services, and the fields of the services. Simple services will adopt simple algorithm such as value matching, but high quality services will utilize complex algorithm such as machine learning. Inference algorithm extension may happen when the algorithm should be

replaced with new one or enhance the existing algorithm and new context inference rules. This extension may influence the quality of context-aware services.

- Context Extension: Context-aware services are specially targeted services for specific contexts. Those targeted services increase the quality of services. Naturally, the need to increase the number of contexts or to define the existing contexts in more details will arise as the time goes. There are three types of context extension: to increase number of contexts, to add new information to the existing contexts, and to define the existing contexts in more details. Ferreira[25] proposed ad-hoc context extensions.
- Context-aware Service Extension: In context-aware systems, some services are context-sensitive but the others are context independent. As the services are provided, new context-aware services are needed. Some of the new services are completely new, but others are the context-aware version from the existing context independent services.

The requirements for those four different types of extension arise often in context-aware system development and maintenance phase. Therefore, context-aware system designers should pay attentions to cover those four different extension requirements.

4. Extension Mechanisms

4.1 Extensible Context Model

4.1.1 Abstract and Implementation

What is “context” in the context-aware computing? There have been many definitions on “context” in this field, but among them, the mostly accepted one is Dey's and Abowd’s definition [23]:

“*Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*”

This definition gives researchers good hints on context modeling and context-aware services, but it seems vague. Therefore, we will define “context” in more detail and to take care of extensibility issues.

Context should be understandable by people because context-aware services should be reasonable to users.

Context should be high level abstraction [17]. Let us take an example. Assume that we are in the room #516 to sign $10 billion contract with Samsung. There are ten people in the room for final reviewing the contract. What would you say the situation? Most of us call the situation as “important business meeting” or other similar terms. We do not say detailed information such

as room number, people identities in the room, or the time to describe the situation. If people do not specify the situation explicitly, the system guesses the situation using the detailed information. The detailed information is sensed by location sensor, camera, microphone, clock, and others. It means that sensor values are used to determine the context, but the summation of the values is not context itself. Context should be abstract and understandable by people.

Furthermore, the context should have detailed information at the same time. Therefore we define "context" as "class" because it is abstract and it has detailed information. Definition 1 shows the definition of "context".

Definition 1: Context

Context is high level, human understandable information which describes the current situation. It may have internal values that describe the situation in more detail.

We suggest a context to be represented as an object in object oriented programming. An object model is good for representing context in high level and presenting contextual attributes in it. Object-oriented programming provides many mechanisms for software extension including Substitutions Principle, and design patterns.

<Figure 3> shows our context model in Object-oriented approach. The conceptual context is represented as an "interface" named IContext, and any specific contexts will be represented as classes which implement the interface. The interface has three basic methods: getCntxName(), get(), and put(). The getCntxName() method is to access to the context name that is understandable by people. The get() method is to access detailed information of the context. Developers can access the information by passing specific information name as its argument. The put() method is used to store any specific information with its name and value. The stored information can be accessed by the get() method. Those get() and put() methods allow any information and any number of information to be managed in the context model.

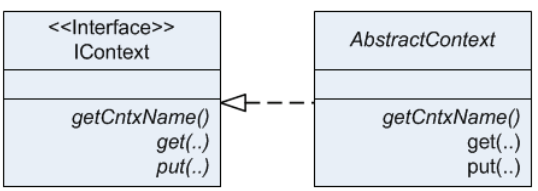

그림 3. 객체지향 상황 모델

Figure 3. Object-oriented context model

A context-aware system may have multiple contexts in its lifecycle, and we call the set of contexts in the system as its context set. The context set is normally identified at requirement analysis phase, and it is determined at the system design phase in the development lifecycle.

Definition 2: Context Set

Context set, C, is the set of contexts used in a context-aware system for its life time.

The size of context set can be changed because new contexts can be required at system design or maintenance phase. This is an example of context extension, and this extension will be

handled at Section 5.4.

4.1.2 Separation of Context from Sensors

We call the sensor whose value is part of context as “context attribute”. With similar way, if system’s internal state whose values are a part of context, and then it is also “context attribute”. Definition 3 shows the definition of context attribute.

Definition 3: Context Attribute

Context attribute is a sensor (physical or virtual) or an internal state. The values of context attributes determine “context” in a context-aware system.

Sensor value is a part of context, but it is not context. When we look into the existing context-aware systems, there are many systems that adopt multiple sensors for their services. In those systems, we cannot say that they have multiple contexts at any specific time because they have multiple sensors.

Rule 1.

A context-aware system must have a single context at any specific time, or the system cannot operate correctly.

We need to separate contexts from sensors. Most of research believes that contexts are related to sensor values, and a context is derived from the sensor values. However, as mentioned above, the summation of sensor values are not the context itself. A context can be derived from various types of sensors. In other words, we can use various sets of sensors to determine a context, rather than only one set of sensors. Furthermore, a type of sensor may have multiple subtype versions of sensors. For example, for location sensor, even though we do not mention manufactures, there are GPS based sensors, RFIDs, Beacons, WiFi based sensors, and others.

<Figure 4> shows our model for context and sensors. There is SensorContainer class, which manages multiple sensors. SensorContainer can keep sensors regardless sensor types or the number of sensors. ISensor is an interface for sensors and it has getValue() method to access the sensor value. The getValues() method is designed to access multiple values which are accumulated for a specific time duration in the sensor.

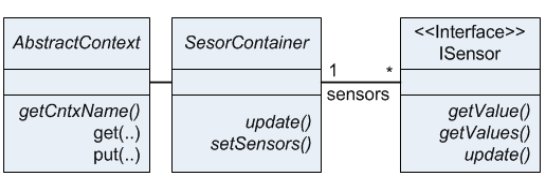

그림 4. 센서 정보와 상황의 분리

Figure 4. Separation of context from sensors

The separation of context from sensors has at least three advantages. First, it decouples between context and sensors. Second, it reduces the system complexity. Third, it increases extensibility, so it allows developers to add new sensors or to replace sensors without or minimally modifying the source code.

4.2 Extensible Context Inference Algorithms

How can context attribute values become a context? The system should gather sensor values, and guess the context using the values according to some algorithm. We call this algorithm as context inference algorithm. It receives context attribute values as input, performs some inference algorithm, and returns the context as the result. Definition 4 shows the definition of context inference function.

Definition 4: Context Inference Function

Context Inference Function, f_x is a function or an algorithm that receives context attribute values at or during the time $t_0..t_k$, and returns the context, C_i , where a_i represents the value of ith context attribute.

$$C_i = f_x\ (a_0,\ a_1,\ ..a_n)t_0..t_k$$

Context inference extension has two types: extension of inference algorithm and extension of rules. The inference algorithm extension is important because context inference may be changed when the new algorithm is developed or new contexts are introduced. Context inference function is implemented as a class which implements ICntxInference interface as shown in <Figure 5>. The interface has interpret() method, which infers the current context using sensor values, and returns a context instance.

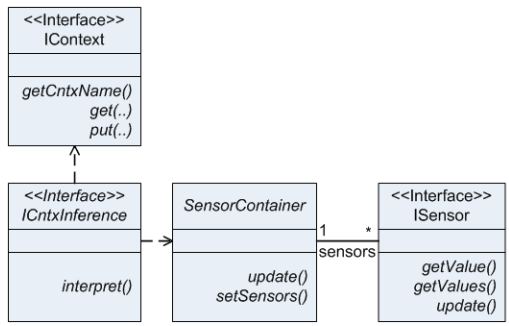

그림 5. 상황 추론

Figure 5. Context inference

Another type of extension, i.e. rule extension, happens when inference data or inference rules are added or modified. Rule extension depends on the context model and inference algorithms. Rule-based systems are good for extension of context inference because they allow inference rules to be added at runtime. Ontology-based systems are also good for extension. However, key-value model is not good for rule extension.

4.3 Extensible Context-aware Services

4.3.1 Modularity and Separate Model

Context-aware services are the services that use contexts to provide intelligent services to users. Context-aware services are activated by context change, or they are the services that perform their services according to contexts. Definition 5 shows the definition of context-aware service.

Definition 5: Context-aware Service

Context-aware services are services that are activated by context change or that perform operation according to the current context.

There are two kinds of services in context-aware systems: context-aware services and context independent services. Context-aware service is complex because it manages business logic, and manipulates the logic according to the contextual information. Then how can we manage both business logic and contexts in programming model. There are two combination models for business logic and contexts: mixed model and separate model.

- Mixed Model: In this model, business logic is mixed with multiple contexts within a function. It means that a function can handle multiple contexts. It is simple when the number of contexts is small.
- Separate Model: In this model, a function manages business logic with only one context. Therefore, when there are multiple contexts, there should be multiple versions of functions according to the number of contexts. It is simple when the number of contexts increases, because each function handles only one context.

The "Mixed Model" is good for simple application, but when the number of contexts increases, it will be error prone and hard to extend. On the other hand, "Separate Model" is good for extensibility. [16] and [17] also suggest the separation of context from business logic. From the two models, we choose the "Separate Model", and a function handles only one context. The separate model increases modularity and cohesion, and reduces complexity. Because we adopt class model for context, each context-aware service is represented as a class method.

4.3.2 Operation Abstraction and Overloading

The rule of separation between abstraction and implementation reduces complexity, and increase flexibility, reusability, and extensibility. This rule can be applied to context-aware services. As mentioned above, we adopt "Separate Model", and a method handles a context. Therefore a context-aware service has a single name for abstraction and semantics, but there are multiple implementations according to the number of contexts. This approach can be managed by "Overloading Model". In other words, a context-aware service has one method name, but its implementation is done with overloading methods with different contexts as their parameters. <Figure 6> shows the method overloading model for context-aware services. For abstraction-implementation rule, "doX()" is an abstraction for a services, and "doX(CntxtA con)" and "doX(CntxtB con)" are implementations with different contexts.

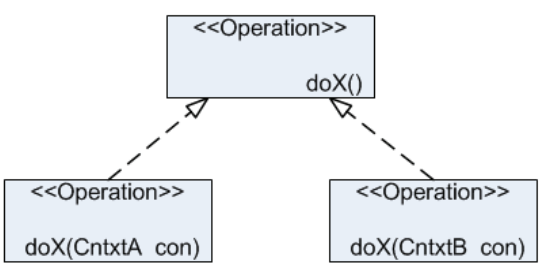

그림 6. 메소드 오버로딩 모델

Figure 6. Method overloading model

With the “Overloading Model”, a context-aware method has the following format as shown in <Figure 7>. There can be a representative method, which has no context or default context as its parameter. And there are overloading methods with different contexts with their parameters. The caller invokes method() without considering the context, but the representative method checks the context and invokes a proper method.

```
<visibility> <returnType> method(args) {
    // generic method
  IContext con = getContext();
  return method(con, args);
}

<visibility><returnType>method(ContextA con, args) {
    // method for ContextA
}

<visibility><returnType>method(ContextB con, args) {
    // method for ContextB
}
```

그림 7. 오버로딩 모델 코드 예

Figure 7. Overloading model code sample

In context-aware systems, Branching Model [19] is about how to determine which method to invoke according to the current context. By applying Overloading Model, the Branching Model issue is addressed easily. When the method() is invoked in <Figure 7>, it determines the current context by invoking getContext(), and invokes one of its overloading methods. The invoked method is selected from its multiple overloading methods by matching its context parameter with the current context. In <Figure 7>, ContextA and ContextB implements IContext interface. Therefore, the current context type determines the best-fit method by overloading method mechanism in OOP.

Overloading model allows developers to add new context-aware services with new context to the context-aware system without modifying the existing source code.

4.3.3 Overriding Approach

Our “Separate Model” and “Overloading Model” have one important property, i.e. they can take advantage of OOP mechanism. The context-aware service implementations can be added without modifying the existing source code by utilizing OOP’s features such as method overriding and overloading.

These features allow two types of extensions: context-aware service extension and conversion of the traditional services into context-aware services. The context-aware services can be extended by overriding to enhance their algorithm or business logic. Furthermore, whenever the traditional services are required to be context-aware, developers can override the existing methods to check the current context and invoke their overloading context-aware implementation.

5. Software Extension Case Study

In this section, we will introduce how to extend a context-aware system with four different extension requirements. First, we will introduce Kapitsaki’s tour guide system implementation following our proposal. After then we will show

how to extend the system to meet the extension requirements.

5.1 Context-aware Tour Guide System

As seen in <Figure 2>, Kapitsaki's tour guide system has two context-aware services: greeting and attractions recommendation. Those services can be implemented as classes and methods according to our approach as shown in <Figure 8>. TourGuide system consists of Greeter and Recommender, and it utilizes three context attributes (tourist's language, location, and weather information). It has Reasoner class which inferences the current context from the context-attributes, and returns the current context. For "greet" context-aware service, we define two contexts (EnglishContext and KoreanContext) according to tourist's language.

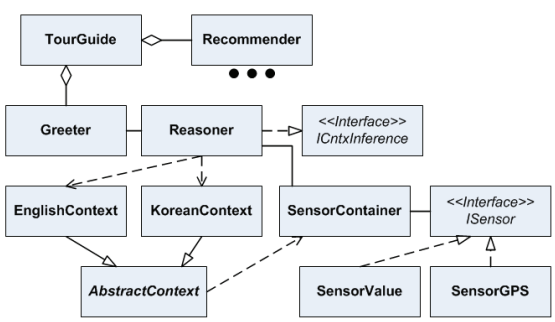

그림 8. 여행 가이드 시스템 클래스도

Figure 8. Tour guide system class diagram

<Figure 8> shows the class diagram of the system. Greeter class has Reasoner to infer the context, and Reasoner itself also utilizes SensorContainer to access sensor values. SensorValue represents a sensor which manages system's internal data with the sensor API (i.e. ISensor).

<Figure 9> shows how Greeter sub-system looks like. It has context-aware functions named "greet". A context-aware service is implemented as overloading methods with their context parameters.

```
public class Greeter extends CObject implements
IContextListener {
....
  public void greet(EnglishContext c, People p) {
       System.out.printf("Hello. %s\n", .get("name"));
  }
  public void greet(KoreanContext c, People p) {
       System.out.printf("안녕. %s\n", p.get("name"));
  }
```

그림 9. Greeter 클래스

Figure 9. Greeter class

5.2 Sensor Extension

By adding new sensors, the context-aware services can be customized to current situation. For our TourGuide system, we can add a time sensor, so that it can greet according to the time. Furthermore, it also can suggest different attraction places according to the time. For example, in day time it recommends good viewpoints or outdoor places, but at evening, it recommends nice restaurants or cafes.

Adding new sensors leads two possible cases: no context change or introduction of new contexts. In this section, we will deal with the first case, because the second case will be handled at 5.4 Section. On adding new sensors, if

the contexts are not influenced, the only parts to be modified are context-sensitive classes (Greeter and Recommender) and the main class (TourGuide) for setting sensors. <Figure 10> shows how to extend the existing classes with new sensors. Utilizing inheritance and method overriding, we can add new sensors without modifying the existing source code.

```
public class GreeterTime extends Greeter {
...
   public void greet(EnglishContext cx, People p)
   {
      String time = (String) cx.get("time");
      ...
      if(hour >= 5 && hour <= 12) {
         System.out.println("Good morning.");
      } else if(hour > 12 && hour <= 18) {
         System.out.println("Good afternoon.");
      }
      ...
```

그림 10. 새로운 센서에 의한 데이터 도입

Figure 10. New sensor data introduction

5.3 Context Inference Extension

Until now several algorithms and techniques for context inference have been developed, and well adopted techniques will improve maintainability and extensibility of the context-aware systems [21]. As the inference technologies advance, the needs for replacing inference algorithm in the context-aware systems increase. Therefore, context inference algorithm should be enhanced or replaced according to new requirements.

<Figure 11> shows an example of context inference extension. The new context inference class (RuleReasoner) inherits from Reasoner. The new inference class adopts predicate logic inference algorithm, and it has RuleManager class for managing fact and inference rule information.

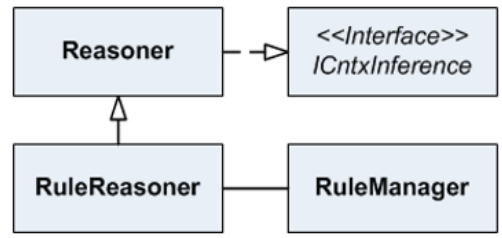

그림 11. 상황 추론 확장

Figure 11. Context inference extension

5.4 Context Extension

In the evolutionary development or the maintenance process, it is required to add more contexts for some context-aware services. It is natural because the system scope and the target user scope are extended after the initial version of software product. For example, the tour guide system is initially targeted to Korean and English users, but in the second version, it is necessary to extend it to accept French tourists. Therefore, the context-aware system should have to be flexible enough to accept new contexts.

Introduction of new contexts means that the context set of the system increases. A new context influences three parts of the system: context itself, context inference engine, and context-aware services. When we add a new context, we have to three tasks for the context. First, we have to define new context as a context class. Second, we have to redefine Reasoner's interpret()operation to support the new context. Third, we have to define a new context-aware

service which runs on the new context.

<Figure 12> shows how to define a new context. The new FrenchContext is defined as a subclass of AbstractContext class.

```
public class FrenchContext extends AbstractContext
{ … }
```

그림 12. 새로운 상황 도입

Figure 12. New context

<Figure 13> shows how to define a new context-aware service with the new context, FrenchContext. A new context-aware service can be implemented by using method overloading approach.

```
public class GreeterFrench extends Greeter {
....
public void greet(FrenchContext c, People p) {
      System.out.printf("Bonjur. %s\n", p.get("name"));
}
…
```

그림 13. 새로운 상황 인지 서비스

Figure 13. New context service

Context extension with inheritance mechanism has two important meanings. First, it allows new contexts to be introduced without modifying the source code. It is helpful to introduce new contexts. Second, it allows developers or maintainers to partition an existing context into several sub-contexts. Those sub-contexts are required to provide specifically targeted services. For example, we may want to divide context-aware services into age specific services for Korean tourists. <Figure 14> shows how to partition an existing context into more specific sub-contexts. KoreanContext is partitioned into two different contexts: KoreanKidContext and KoreanAdultContext. The context partition is also a kind of context extension, and some common services provided regardless of ages are used without any change. Only age specific services should be (re)defined.

```
public class KoreanKidContext extends KoreanContext
{ … }

public class KoreanAdultContext extends KoreanContext
{ … }
```

그림 14. 상황 파티션

Figure 14. Context partition

5.5 Context-aware Service Extension

The extension of context-aware services may happen very frequently. There are four cases for context-aware service extension: 1) new context-aware services due to the introduction of new contexts, 2) new context-aware services for new requirements, 3) enhancement of the existing context-aware services, and 4) conversion of the traditional services into context-aware services.

Using method overriding, we can extend the traditional methods to context-aware methods. In our example, public String getName() method, which returns tourist's name, defined in Greeting is extended to a context-aware method in order to return the name in tourist's own language. <Figure 15> shows an example code that converts a traditional service into a context-aware service.

```
public class NewGreeting extends Greeting {
   public String getName(People p) {
      IContext con = getContext ();
      return getName(con, p);
   }
   public String getName(EnglishContext c, People
p) {
       // check p's title and sex information
       return "Mr. " + super.getName();
   }
   …
```

그림 15. 기존 서비스를 상황인지 서비스로 변환

Figure 15. Conversion of a traditional service into a context-aware service

6. Conclusions

Until now, most of context-aware systems have been developed and tested in the laboratory, and only a few systems have been used in the real world. However, in the future, context-aware systems will be widely used in the real world, and then software engineering issues including developing and maintenance cost will be important. Software extensibility will be one of the important issues because it is related to the cost, software reusability, and development or maintenance time.

Even though the software extensibility is an important issue, researchers in the context-aware computing field have not paid much attention on this topic. In this paper, we identified four types of extension requirements: sensor extension, context inference extension, context extension, and context-aware service extension. Furthermore, we also proposed some extension mechanisms including object model for context and overloading model for context-aware services. We showed that some scenarios for software extension, and showed how to extend the existing context-aware systems without changing the existing source code.

Extensibility is very important property, and we contribute extensibility for context-aware systems. It will enable developers to extend the existing context-aware systems with minor efforts with less time and less cost.

References

[1] F. P. Brooks, *No silver bullet, essence and accidents of software engineering*, Computer, Vol. 20, No. 4, pp. 10-19, 1987.

[2] M. Zenger, *Programming language abstractions for extensible software components*, PhD Thesis, EPFL, Switzerland, 2004.

[3] D. Notkin, and W. G. Griswold, *Enhancement through extension: the extension interpreter*, Proc. of SIGPLAN'87, ACM SIGPLAN Notices, Vol. 22, No. 7, pp. 45-55 1987.

[4] Robert L. and Glass, *Frequently forgotten fundamental facts about software engineering*, Software, Vol. 18, No. 3, pp. 110-112, 2001.

[5] B. P. Lientz, and E. B. Swanson, *Software maintenance management*, Addison-Wesley Longman Pub., 1980.

[6] OSGi, http://www.osgi.org/, accessed in Mar. 2017.

[7] O. Gruber, B. J. Hargrave, J. McAffer, P. Rapicault, and T. Watson, *The eclipse 3.0 platform: Adopting OSGi technology*, IBM Systems Journal, Vol.

44, No. 2, IBM, pp. 289-299, 2005.

[8] M. Baldauf, S. Dustdar, and F. Rosenberg, *A survey on context-aware systems*, Int. J. Ad Hoc and Ubiquitous Computing, Vol. 2, No. 4, pp. 263-277, 2007.

[9] D. Notkin, *Extension and software development*, Proc. of the 10th Int'l Conf. on Software Engineering, pp. 274-283, 1988.

[10] D. L. Parnas, *Designing software for ease of extension and contraction*, In Proc. of the 3rd Int'l Conf. on Software Engineering, pp. 264-277, 1978.

[11] B. Meyer, *Object-oriented software construction*, Prentice Hall, 1988.

[12] B. H. Liskov, and J. M. Wing, *A behavioral notion of subtyping*, ACM Trans. Program. Lang. Syst. Vol. 16, No. 6, pp. 1811−1841, 1994.

[13] T. H. Ng, S. C. Cheung, W. K. Chan, and Y. T. Yu, *Toward effective development of design patterns for software extension: A case study*, Proc. of the 2006 Int'l Workshop on Software Quality, pp. 51-56, 2006.

[14] L. Prechelt, B. Unger, W. F. Tichy, P. Brossler, and L. G. Votta, *A controlled experiment in maintenance: comparing design patterns to simpler solutions*, IEEE Trans. Software Eng. Vol. 27, No. 12, pp. 1134-1144, 2001.

[15] A. A. Sabagh, and A. Al-Yasiri, *An extensible framework for context-aware smart environments, architecture of computing systems-ARCS 2011*, LNCS, Vol. 6566, Springer, pp 98-109, 2011.

[16] J-M. Choi, *Software architecture for extensible context-aware systems*, Proceeding of Convergence and Hybrid Information Technology, pp. 811-816, 2008.

[17] P. Fahy, and S. Clarke, *CASS-middleware for mobile context-aware applications*, Workshop on Context Awareness, MobiSys 2004.

[18] D. Athanasopoulos, A. V. Zarras, V. Issarny, E. Pitoura, and P. Vassiliadis, *CoWSAMI: interface-aware context gathering in ambient intelligence environments*, Pervasive and Mobile Computing, Elsevier, Vol.4, No. 3, pp. 360-389, 2008.

[19] K. Henricksen, and J. Indulska, *A software engineering framework for context-aware pervasive computing*, Proc. of the 2nd Advent IEEE Annual Conference on Pervasive Computing and Communications (PERCOM'04), IEEE, 2004.

[20] G. M. Kapitsaki, G. N. Prezerakos, N. D. Tselikas, and I. S. Venieris, *Context-aware service engineering: A survey*, Journal of Systems and Software, Elsevier, Vol. 82, No. 8, pp. 1285−1297, 2009.

[21] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni, *A survey of context modelling and reasoning techniques*, Pervasive and Mobile Computing, Elsevier, Vol. 6, No. 2, pp. 161-180, 2010.

[22] T. Strang, and C. Linnhoff-Popien, *A context modeling survey*, Proc. of the 1st Int'l Workshop on Advanced Context Modelling, Reasoning And Management at UbiComp 2004, 2004.

[23] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, *Towards a better understanding of context and context-awareness*, Int'l Symposium on Handheld and Ubiquitous Computing, pp. 304-307, Springer, 1999.

[24] R. Bouckaert, J. Heled, D. Kühnert, T.

Vaughan, C-H. Wu, D. Xie, M. A. Suchard, A. Rambaut, and A. J. Drummond, *BEAST 2: A software platform for mayesian evolutionary analysis*, PLOS Computational Biology, 10:e1003537, 2014.

[25] D. Ferreira, V. Kostakos, and A. K. Dey, *AWARE: mobile context instrumentation framework*, Frontiers in ICT 2, 6, 2015.

[26] G. Meditskos, S. Dasiopoulou, and I. Kompatsiaris, *MetaQ: A knowledge-driven framework for context-aware activity recognition combining SPARQL and OWL 2 activity patterns*, Pervasive and Mobile Computing, Elsevier, Vol. 25, No. 1, pp. 104-124, 2016.

상황인지 시스템에서 객체 지향성과 확장성에 관한 연구

최종명[1], 김익수[2]

[1] *목포대학교 컴퓨터공학과*
[2] *숭실대학교 컴퓨터학부*

요 약

상황인지 컴퓨팅에 대한 많은 연구가 있었지만, 소프트웨어 공학의 중요성에도 불구하고, 상황인지 시스템 개발에서 소프트웨어 확장성에 대한 연구는 상대적으로 많지 않았다. 본 논문에서는 상황인지 시스템에 필요한 확장요구사항을 소개하고, 상황인지 시스템의 확장유형을 네 가지 형태(센서, 상황 추론 알고리즘, 상황, 상황인지 서비스)로 구분하여 소개한다. 본 논문에서는 이러한 확장요구사항을 위해 객체지향 기술에 기반을 둔 네 가지 확장메커니즘을 제안한다. 즉, 상황에 대한 추상화와 구현의 분리, 센서와 상황 분리, 상황인지 함수의 모듈화, 상황인지 함수의 오버로드 모델을 제안한다. 이러한 메커니즘을 채택함으로써 개발자 또는 유지보수 관리자는 배포 후 소스 코드를 수정하지 않고, 새로운 센서, 상황 추론 알고리즘, 상황, 상황인지 서비스를 추가할 수 있다. 본 논문에서 제안하는 방법은 새로운 상황인지 특징이 요구되는 상황인지 시스템 유지보수에서 비용, 시간, 노력을 절감시킬 것이다. 이러한 확장메커니즘은 모두 객체 지향을 기반으로 하였다. 즉, 상황을 클래스로 표현하고, 상황인지 서비스를 상황을 매개변수로 갖는 메소드로 표현한다. 상황추론은 센서 값을 이해하고, 이를 이용해 현재 상황을 추론하는 메소드가 있는 클래스로 표현한다. 논문에서는 또한 확장요구사항이 있는 상황인식 시스템을 확장하는 구체적인 사례를 소개한다.

Acknowledgement

본 연구는 “(재)전남정보문화산업진흥원 2016년 수요창출형 R&D 지원 사업”의 지원을 받았음.

Jongmyung Choi received the Bachelor's degree, Master's degree, and Ph. D. in computer science from Soongsil University, South Korea, in 1992, 1996, and 2003 respectively. He is currently a professor in the Department of Computer Engineering, Mokpo National University, South Korea, since 2004. He did research as a visiting scholar at Georgia Institute of Technology, USA, from Aug. 2010 to Dec. 2011. His research interests are human computer interaction, context-aware systems, social computing, and healthcare.

E-mail address: jmchoi@mokpo.ac.kr

Iksu Kim received the B.S., M.S., and Ph.D. in Computer Science from Soongsil University, South Korea, in 2000, 2002, and 2008, respectively. He worked at SKYCOM as a manager until January 2009. He is currently an associate professor in the School of Computer Science and Engineering at Soongsil University since September 2009. His research interests include system security, network security, and mobile security.

E-mail address: iksuplorer@ssu.ac.kr