Model-Based Approach for Engineering Adaptive User Interface Requirements

Kibeom Park[1(✉)] and Seok-Won Lee[2]

[1] Graduate School of Software, Ajou University, Suwon, Republic of Korea
pkb@ajou.ac.kr
[2] Department of Software Convergence Technology, Ajou University, Suwon, Republic of Korea
leesw@ajou.ac.kr

Abstract. Although Model-Based User Interface (MBUI) design approaches have been suggested and researched over a long period of time, the advantages of adopting them into the development of Adaptive User Interface (AUI) have not stood out. We believe that it is due to the lack of an integration of the Requirements Engineering (RE) process, and methodologies for Model-based AUI development. Since RE provides a solid base to the development of software, requirements of AUI have to be preceded appropriately in the development process. Previously, we suggested a RE method for AUI reflecting the viewpoint of Self Adaptive System (SAS). In this paper, we elaborate on our previous method grounded on a model-based approach. The proposed method is illustrated with an example scenario, which makes adaptations of the user interface at run-time by conforming to the context of users. Finally, an evaluation of our method is provided by a case study at the end of the paper.

Keywords: Adaptive User Interface (AUI) · Model-Based User Interface (MBUI) · Requirements Engineering (RE) · Self-Adaptive System (SAS) · User Interface (UI)

1 Introduction

Due to the widespread popularity of mobile devices, demands of end users to be provided with personalized and customized services have been tremendously raised these days. For achieving such a personalization of services, a system should fulfill dynamic requirements varying in different contexts at runtime. One of these attempts to support such runtime personalization is realizing the use of Adaptive User Interface (AUI).

AUI is a user interface (UI) that has the ability to adapt itself by reasoning a suitable presentation of the service according to a situation at run-time. Realizing AUI is considered a difficult challenge because a system needs to monitor and analyze the context data of a user and his or her environment at run-time without any decisions made by human. It demands several pre-defined rules and a knowledge base to make proper adaptations of UI to satisfy diverse user demands.

L. Liu and M. Aoyama (Eds.): APRES 2015, CCIS 558, pp. 18–32, 2015.
DOI: 10.1007/978-3-662-48634-4_2

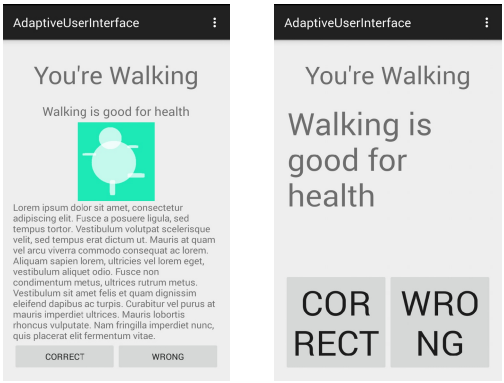

Fig. 1. UI adaptation prototype when user is standing (left) and running (right)

Fig. 1 illustrates a simple prototype of AUI in healthcare mobile application. The left side, which represents a normal version of UI, has several UI components; a title, sub-title, picture, texts with a small-sized font, and two buttons. One of the requirements is that the screen displayed to the user should be adequately changed with different user activities. For example, the normal version of UI is no longer usable when the user is running, since the user finds it hard to recognize contents in the screen due to the wobbling motion produced.

In this case, the UI can be adapted by increasing the font size and hiding some of the unimportant content. One possible adaptation is provided on the right side. If we assume that the subtitle and two buttons at the bottom are important, we can not only hide picture and texts with a small-sized font (considered unimportant content), but also increase the size of the subtitle and two buttons (important content). By doing so, we now can preserve usability, readability and legibility. Fig 1. will be used again for describing our method in section 3.

It is obvious that we cannot consider AUI just as a change of 'views', since it requires considerable thoughts in 'logic' for monitoring context data, reasoning suitable versions of UI adaptation, etc. Therefore, it brings us to look at AUI from the domain of software engineering.

For making a design of AUI, Model-based User Interface (MBUI) design approaches such as CAMELEON-RT [1] and several User Interface Description Languages (UIDLs) like UIML[2] and UsiXML[3] have been proposed for decades in the Human Computer Interaction (HCI) research area. Even though these approaches enable software engineers to develop infrastructure for AUI, it is still hard to apply them to actual AUI development so far. One of the major reasons is the lack of consideration of Requirements Engineering (RE). Since RE provides a solid base to the development of software, requirements of AUI have to be preceded appropriately in the development process.

In our previous work [4], we proposed requirements elicitation guidelines using general concepts of a Self-Adaptive System (SAS). However, we did not adopt existing MBUI design research from the HCI area. There continues to be a gap between RE for AUI in the software engineering area and MBUI in the HCI area.

In this paper, we extend our work by introducing the viewpoints of model-driven development. We exploit the MBUI research, and accordingly, the benefits of MBUI approaches are adopted into the RE method for AUI.

The rest of the paper is organized as follows: In section 2, we review the related literature in three parts. The definition and the characteristics of AUI are surveyed in the first part, and the previous MBUI design approaches are described in the second part. We then review our previous requirements elicitation guidelines for AUI in the last part. In section 3, we proposed an extended RE method for AUI by adopting a model-based approach and show how our method works by illustrating an example from the mobile application domain. Section 4 provides an evaluation of our method by application to a case study. At last, we conclude the paper with the summary of contributions and future works in section 5.

2 Literature Review

2.1 Adaptive User Interface

As we mentioned above, AUI is the UI that has an ability to adapt itself to the change of the context of user and device. The term 'context' as used above can vary by many design decisions of the developers. It can be the static context of individual user or device, including user characteristics [5], the user's physical capability, user preference, device type, etc. It can be a dynamic context considering spatial or temporal factors as well.

The term 'adaptive' should be distinguished from the term 'adaptable' [6]. One of the main differences between Adaptable UI and Adaptive UI is that Adaptable UI uses only the static context of the user while Adaptive UI uses both the static and dynamic contexts of the user. Fig. 2 portrays the difference between those two approaches and their relationships with static context and dynamic context. This shows that AUI is much broader concept than Adaptable UI.

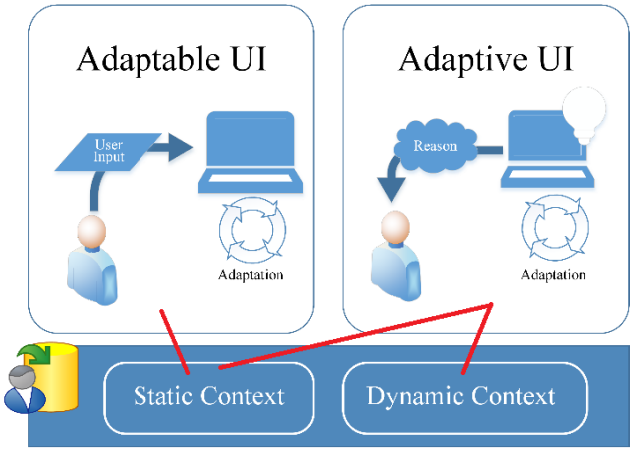

Fig. 2. Difference between Adaptable UI and Adaptive UI

Adaptable UI requires user input on such things including preference, physical capability, etc. The system then makes suitable UIs based on the user inputs. For instance, imagine a system that requires user input on whether the user is visually impaired or not, and then uses the response as a parameter for the proper adaptation. If the user has difficulty seeing, a voice-guided UI might be provided to the user in this case. Another example is that the UI that changes itself depending on the device type, such as display size. The system may choose a suitable version of UI for current display size of the device amongst already designed UIs. Even though the system could figure out the display size of the current device, it would be categorized as making a choice based on user inputs among the design templates during the design time.

AUI, on the other hand, contains the intention of generating a design at run-time rather than choosing a design at the design time. Instead of requiring user inputs, it examines the context of the user itself. Therefore, AUI reduces the burden of the user to respond in advance, unlike Adaptable UI. Furthermore, it makes reasoning about the situation and adapting the UI at run-time possible.

2.2 Model-Based User Interface (MBUI) Design

MBUI approaches have been proposed for decades as a way of achieving a technical basis for realizing AUI. The key concept is to separate UI development in multiple layers. One insightful paper addressing this concept is [7], which suggested the model-based UI framework with three layers: an Abstract UI, Concrete UI and a Final UI.

Abstract UI describes what the user actually works with the system. Currently, in the interactive system research area, researchers use a Task model and Domain model for description of Abstract UI. A Task model represents a task flow of user interactions, and a Domain model describes the knowledge of UI components which later can be used for adaptation.

Concrete UI describes more specific graphical representation based on Abstract UI. In this layer, the UI is dependent on platform or devices. It can be represented as a high-level user interface description languages (UIDL) such as UIML[2], UsiXML[3], etc. It can contain several UI design alternatives since it is written in high level.

Final UI describes final UI design decision. In this layer, high-level Concrete UI design is specified in more details. One of the examples of Final UI is a choice between ‘radio button’ or ‘select box’, in case the Concrete UI denotes ‘graphical selection’ as a high-level design decision.

Although an MBUI approach has been suggested and researched over a long period of time, the advantages of adopting it in the development of AUI did not stand out enough. A model-based approach in UI development focuses on solving device compatibility, but does not mention how they could be used in the development of AUI. We insist that this problem is caused by a lack of the RE process and methodologies for AUI. Currently, the requirements are addressed only in Abstract UI and the models in this layer provide much too abstract requirements. For this reason, detailed requirements that should exist according to specific domains are ignored and not addressed properly.

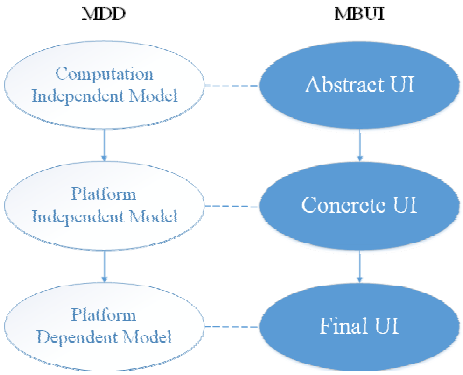

Fig. 3. A diagram mapping the relationship between Model-driven Development (MDD) and Model-based User Interface (MBUI)

In this paper, we strongly believe that the strong points of adopting model-based approach are also found in the development of AUI. A RE method, especially, can be elaborated with such a kind of approach. The following are the benefits of MBUI approach, which are normally addressed in the UI development. We will use these criteria to evaluate our method, which benefits from these in section 4.

A. Reusability

Model-based approaches in UI development enable the reuse of models, meta-models and transformations of UI components by separating the concerns of each level. When the needs are changed, developers can easily choose another way of specific adaptation alternatives easily without changing the high-level design. In this case, developers are able to reuse designs available in high-level layers.

B. Run-time Adaptivity

Making different adaptations at run-time is most suitable since the layered model explicitly structures a higher level goal and its related lower level design alternatives accordingly. This makes a system knowledgeable to what to adapt in a specific way to satisfy the high-level goal.

C. Modifiability

One of the benefits of a MBUI approach is that it enables a choice between an alternative design or to change the current design. The reason is that the designs are separately made with three layers according to the level of abstraction, and then it separates the concerns in the design. With layers, the detailed adaption logic can be modified easily.

2.3 Requirements Elicitation Method for Adaptive User Interface

Requirements elicitation is the first step in drawing the needs of users and various stakeholders. Since AUI requires motives triggering UI adaptation and several logics for monitoring the context at runtime, it is obvious that the requirements for AUI should be clearly elicited and well defined.

In fact, UI development has not been considered in the RE area well, as it has been regarded more in the domain of HCI area. Also, researchers of MBUI have developed different methodologies and terms that make it difficult to integrate that research into a software engineering perspective. One example is that researchers of developing Interactive System use different methodologies like a 'Task model' and 'Domain model' instead of RE for handling the goal and flow of UIs. Although there exist some research using those methodologies as a basis for RE [8], they do not expand its coverage towards AUI. There continues to be a lack of research addressing adequate RE methodologies for AUI.

For this reason, our previous work focused on how to elicit initial requirements of AUI [4]. We proposed guidelines for eliciting AUI requirements using well-known concepts from SAS research.

Our previous paper introduced a 3-step requirements elicitation process including 'AREA – BASE – CONSEQUENCE', focusing on AUI development in the domain of the mobile application. In the first step 'AREA', we considered context, property and constraints of the domain. Then we re-defined the term 'MAPE-K Loop' [9], 'Self-* Properties' [10], and their elements considering the contents of AREA for making them proper to the domain of mobile application. During the second step

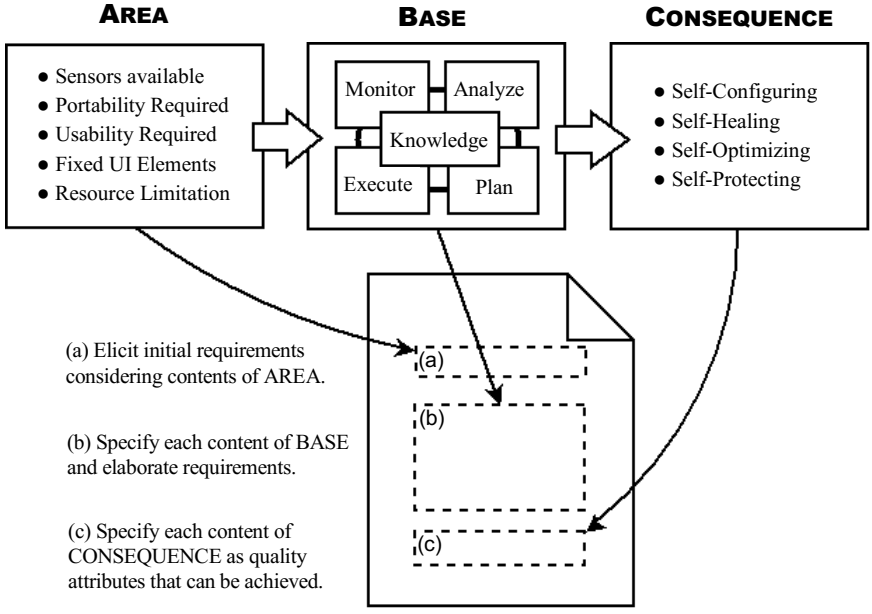

Fig. 4. Requirements Elicitation Method for AUI

'BASE', developers should use each element of MAPE-K Loop 'Monitor', 'Analyze', 'Plan', 'Execute' and 'Knowledge' as criteria for eliciting functional requirements. In this way, requirements could be elaborated more upon according to different domains. During the last step, 'CONSEQUENCE', re-interpreted Self-* Properties are used as a guideline for eliciting quality attributes. These elements could be the minimum quality attributes for achieving an adaptivity. That is, Self-Configuring, Self-Healing, Self-Optimizing, Self-Protecting should at least be elicited.

Previously, our work did not address how to bridge elicited requirements to MBUI design approaches. If we integrated the merits of MBUI, we could exploit many benefits of an MBUI approach in the development of AUI. In this reason, we extend our previous method by introducing a model-based approach in the paper.

3 Model-Based Engineering for AUI Requirements

3.1 Adopting Model-Based Approach to AUI Requirements

Fig. 5 shows how we can adopt the notion of model-driven development into RE. Three layered modeling from [7] is adopted to RE, respectively: Abstract Requirements for AUI, Concrete Requirements for AUI, and Final Requirements for AUI.

Abstract Requirements for AUI represent domain-independent AUI requirements while Concrete Requirements for AUI represent domain-specific requirements of AUI. Final Requirements for AUI mean detailed variation on UI elements.

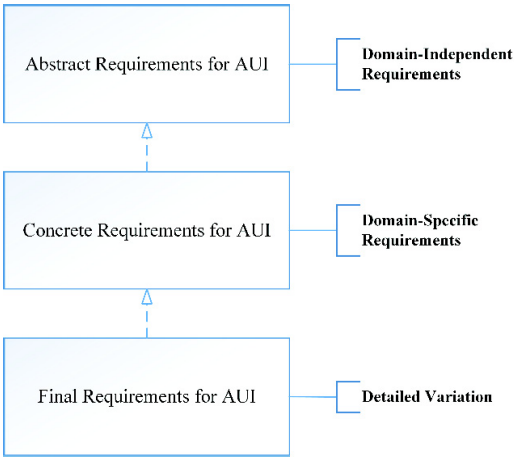

Fig. 5. Model-based Requirements Engineering for AUI

3.2 Proposing an Extended Method for AUI

In this section, we propose an extended requirements elicitation method for AUI by bringing a model-based approach into RE. Our previous work is restructured with a three-layered model-based notion in this paper. Fig. 6 gives an explanation of our approach. Our method guides software engineers to specify AUI requirements with three layers according to the level of abstraction.

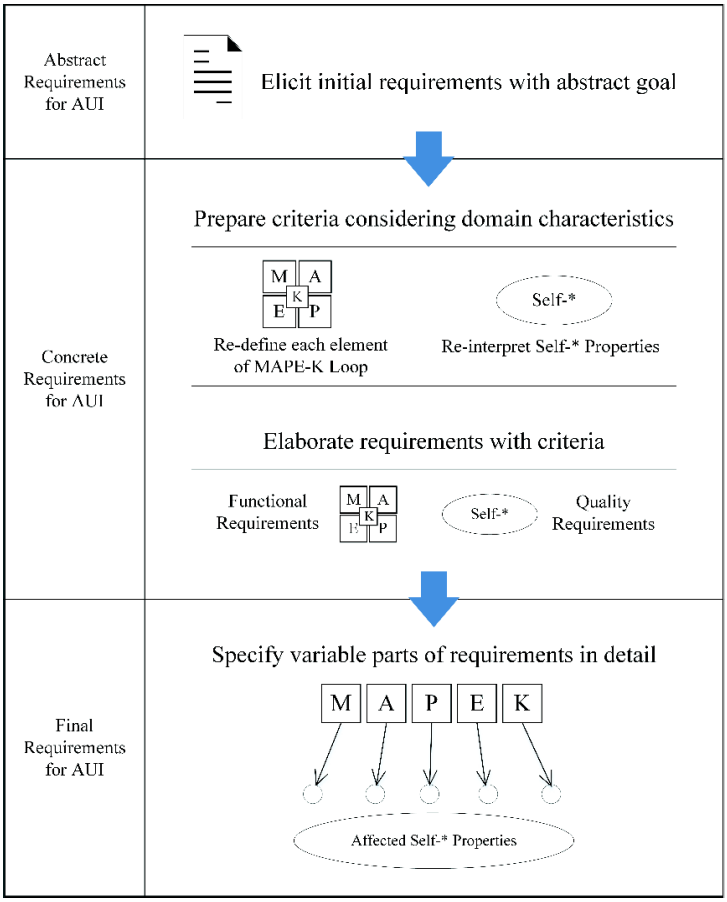

Fig. 6. Extended Requirements Engineering Method for AUI

A. Abstract Requirements for AUI

Abstract Requirements for AUI are described with the highest upper-level goals. These requirements might correspond to the initial rough requirements with goals and intentions. The requirements at this point are general, and not tailored to specific machine or domain. Therefore, we can say that they are kept independent on the specific domain, and it can be reused when the AUI should be adopted into other domains or machines with similar purposes.

B. Concrete Requirements for AUI

Concrete Requirements for AUI are dependent on the certain areas where software belong. In this layer, software engineers should consider the characteristics, unique features, and standards of the specific domain area. When the software engineer finds out the application domain area of software, the next thing to do is to define each element of the MAPE-K Loop and interpreting each property of Self* Properties, according to the domain. The elements of MAPE-K Loop are Monitor/Analyze/Plan/Execute, and these are used as the criteria to verify whether functional requirements are elicited well or not, in the perspective of a self-adaptive system. The elements of Self-* Properties are Self-Configuring, Self-Healing, Self-Optimizing, Self-Protecting, and these are consequences of elicited requirements which can be used to verify whether quality attributes are elicited in the perspective of self-adaptive system well or not. Each element of Self-* Properties also should be specified in the step.

Following the criteria based on MAPE-K Loop and Self-* Properties, software engineers can elicit AUI requirements easily. In addition, these requirements would be elaborated on to comply with the criteria. In this step, requirements should be more concrete than the previous step, but must not be too specific as to disturb the modification of adaptation rules. Concrete requirements for AUI should make room for the changeable part for the adaptation so that it preserves modifiability.

C. Final Requirements for AUI

We mentioned that the requirements that are specified in the previous steps do not contain too specific of requirements, such as detailed adaptation rules or UI elements to be adapted, in order to make a room for changeable parts. In this final layer, detailed requirements that can be expected to be changed later are specified and analyzed. Final Requirements for AUI are domain/machine-dependent requirements, and these can also cause changes of the quality attributes. Therefore, quality attributes specified complying with Self-* Properties might be changed accordingly. Related quality attributes are connected and traced through higher layers.

The three steps above, which are divided by the level of abstraction, enable the separation of the concerns in the requirements. Not only does it take the strong points of a model-based approach, it also makes it easy to trace the related requirements and change parts without changing the whole requirements. Moreover, this can be extended to each design level. We will later show the strengths of our method.

3.3 Method Illustration with an Example Scenario

For a deeper understanding of our method, we now illustrate our method with an example scenario. We elicit, analyze, and specify AUI requirements by following our model-based RE method for AUI. The results correspond to the requirements for implementation of the right side of Fig. 1.

The AUI scenario in our example is in the situation of the development of a healthcare mobile application. We assume that users might be everywhere, including both inside and outside of their home. Also, users might have different characteristics and ages, genders, characters, jobs, etc. The application can satisfy the personalized demands of users by adopting AUI in the application, considering both the user's situation and environments.

We assume that the initial goal to achieve from AUI is to make adaptations of UI at runtime to conform to the context of users. For achieving our initial goal, initial requirements from various stakeholders should be elicited. As one example, we would use the following requirement in the Abstract Requirements for AUI layer: *If the user finds it hard to see UI clearly because of the changes of environment, AUI increases its usability by changing its UI.*

In the Concrete Requirements for AUI layer, in order to make unclear requirements clear, the characteristics and constraints in the domain area of the software are considered. For example, Table 1 represents such considerations in the development of the mobile application. In our previous paper, we elicited considerations of the mobile domain from [11].

Table 1. Considerations of Mobile Domain

Consideration	**Description**
Sensors Available	Several sensors including an accelerometer are available.
Portability Demand	It should be compatible among multiple platforms/machines.
Usability Demand	Personalized user-centric service should be provided.
Fixed UI Elements	It should use a UI library that already exists.
Limited Resource	Resources such as battery, CPU, storage are limited.

Table 2. Re-defined Descriptions of MAPE-K Loop Elements

MAPE-K Loop	**Description**
Monitor	Data and its monitoring method for recognizing situation.
Analyze	Rule for analyzing situation.
Plan	Rule for adaptation.
Execute	Actual Adaptation Behavior.
Knowledge	Knowledge required for recognizing situation and adaptation.

Table 3. Re-interpreted Descriptions of Self-* Properties Elements

Consideration	**Description**
Self-Configuring	Personalization according to user characteristics and situation.
Self-Healing	The ability to recover the usability when unexpected increase of UI complexity occurs,
Self-Optimizing	UI optimization according to machine profile and resource(Battery, CPU, etc.) situation.
Self-Protecting	Preventing UI crashes or defending when UI crashes.

MAPE-K Loop and Self-* Properties are re-defined or re-interpreted according to the domain. In our case, we analyze them in the mobile domain. Table 2 represents the re-defined descriptions of MAPE-K Loop elements considering the mobile domain. Table 3 represents the re-interpreted descriptions of elements in the Self-* Properties.

The next step for software engineers is to specify concrete requirements according to the criteria that we have constructed. Table 4 contains the requirements specification of this step. Detailed types of sensors or rules for adaptation are not specified in this step. That is because those parts have high possibility to change.

Detailed rules that have high chances of change are specified in the next step, the Final Requirements for AUI. In this layer, related quality attributes are revisited and modified. The specification of Final Requirements for AUI is represented in Table 4. These contain adaptation rules and situation recognition rules which can be modified often due to the demands of stakeholders.

Table 4. Abstract, Concrete and Final Requirements of AUI

<table>
<tr><td>Abstract Req.</td><td colspan="4">If the user finds it hard to see UI clearly because of the changes of environment, AUI increases usability by changing its UI.</td></tr>
<tr><td rowspan="4">Concrete Req.</td><td>Monitor</td><td>Analyze</td><td>Plan</td><td>Self-* Property</td></tr>
<tr><td>The AUI monitors wobbles (shakes) through sensors that Android phone provides.</td><td>The AUI analyzes data and determines whether the user is interrupted by current wobble or not.</td><td>The adaptation rule is a UI simplicity rule that is performed when AUI determines that usability is getting too lower.</td><td rowspan="4">Self-Configuring, Self-Healing, Self-Optimizing</td></tr>
<tr><td colspan="2">Execute</td><td>Knowledge</td></tr>
<tr><td colspan="2">Plan is actually performed when the decision is triggered.</td><td>Sensor monitoring interval, usability metrics, UI simplicity rule, Adaptation rule.</td></tr>
<tr><td rowspan="4">Final Req.</td><td>Monitor</td><td>Analyze</td><td>Plan</td></tr>
<tr><td>The type of sensors that is used: Gyroscope</td><td>Usability evaluation algorithm: When gyroscope data is above a certain level, usability value decreases.</td><td>UI simplicity rule: Increase text size and hide unimportant UI elements.</td></tr>
<tr><td colspan="2">Execute</td><td>Knowledge</td></tr>
<tr><td colspan="2">Trigger rule: When usability value is under a certain level, adaptation triggers.</td><td>Sensor monitoring interval, usability metrics, UI simplicity rule, Adaptation rule.</td></tr>
</table>

4 Evaluation

In this section, we design a case study for evaluating our method by following the case study design methodology in [12].

4.1 Study Questions

Our model-based RE method for AUI in this paper also uses the advantages of using model-based approach in the UI development that we described in the section 2.2. We previously described three benefits of using a model-based approach: Reusability, run-time adaptability and modifiability. Our proposed method also introduces the traceability. In addition, this makes requirements to be easily extended to the design. To prove that these advantages are obtained by using our method, following questions are discussed.

- *Q1. Does it support reusability*?
- *Q2. Does it support run-time adaptivity*?
- *Q3. Does it support modifiability*?
- *Q4. Does it support traceability*?
- *Q5. Is it extendable to design level*?

4.2 Case Study

Table 6 shows the evaluation of our method by answering each question. Each answer shows whether it is enough to support each question, and if so, the evidence that supports it.

Table 5. Method Evaluation

Method Evaluation Table			
	Study Question	***Support***	***Evidence***
Q1	Does it support reusability?	Yes	When the domain or machine targeting is changed, requirements of upper layer can be used.
Q2	Does it support run-time adaptivity?	Yes	It controls UI elements separately since it can have knowledge about adaptation.
Q3	Does it support modifiability?	Yes	When adaptation rules should be changed, software engineers can change only local parts, instead of changing the whole part, because it is only related to the Final requirements.
Q4	Does it support traceability?	Yes	Requirements among layers and their affected quality attributes are easily found.
Q5	Is it extendable to design level?	Yes	Design and implementation are performed from requirements.

A. Reusability

It is reusable because it separates the concerns into three layers, and upper layers are still preserved when changes need to occur locally. For example in our scenario, when software engineers decide to support a desktop application, they can reuse Abstract Requirements for the desktop application.

B. Run-time Adaptivity

It supports run-time adaptivity because it allows AUI to have knowledge about the adaptation. Unlike traditional adaptable UI, which only allows for a few versions of whole UI, our method supports the development of a lot of versions of AUI. That is because our method can control each UI element individually and it is possible to adopt more detailed adaptations. For example, UI simplicity rules contain knowledge about the importance of each UI element, which enables separate control of each element.

C. Modifiability

It is modifiable because it separates the concerns into three layers, and without changing the whole AUI, software engineers can change only the local part in question. For example in our scenario, when software engineers decide to change the UI simplicity rule, they can specify only the Final Requirements for AUI again.

D. Traceability

Traceability is ensured because it has been derived from requirements from the upper layer. For example, when Final requirements should be changed, corresponding Concrete Requirements of AUI and their quality attributes are easily found.

E. Extendibility to Design

The model-based requirements can be easily extended to design and implementation. We designed the logic of the simple prototype of Android application, and it is shown in Fig. 1. In our design, Concrete Requirements for AUI are made to a Java Interface that includes Monitor, Analyze, Plan, Execute as the methods, and Knowledge as variables. Then, we implement FinalUI, which satisfies the ConcreteUI.

```
package pkb.nise.ajou.ac.kr.adaptiveuserinterface.aui;

/**
 * Created by AJOU on 2015-06-25.
 */
public interface ConcreteUI {
    public void startMonitoringSensor();
    public void stopMonitoringSensor();
    public void analyzeUserMoveStatus();
    public void planToSimplifyUI();
    public void executeAdaptation();
}
```

Fig. 7. Extending Concrete Requirements to the Design of Java Interface

Through the design and implementation, we found that our method allows for extending requirements easily to design and implementation. Fig. 7 shows the possibility of extending the design.

5 Conclusion and Discussions

Requirements elicitation process and methodologies for AUI development have not been defined well so far. Although our previous work suggested a guideline for AUI requirements, it did not reflect the advantages of MBUI design approaches.

In this paper, we proposed an extended RE method for AUI by adopting a model-driven approach. We showed how our method works by illustrating case study examples. By using our method, software engineers can effectively elicit requirements in AUI development step by step.

However, this paper contains several points of discussion, which can be seen as follows. First, the approach we suggest is still at a very high level or can be considered too general, in that many details are omitted. There remain questions about how to transform requirements in different layers. Furthermore, RE processes after the elicitation is not addressed well. In the future, we plan to focus on this question to elaborate our method.

Second, the evaluation is not enough since we conducted only one case study. In addition, the metrics for evaluation hold some threats to its validity. We will conduct more case studies for a better evaluation in the future.

Lastly, it is not clear how existing RE approaches can be integrated into the proposed framework or design approaches. For example, there is already much research that address the model-based AUI design approach such as that in [13]. We will bridge our RE method to the design method of AUI by further research.

Acknowledgment. This research was supported by the Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (2013M3C4A7056233).

References

1. Balme, L., Demeure, A., Barralon, N., Calvary, G.: CAMELEON-RT: a software architecture reference model for distributed, migratable, and plastic user interfaces. In: Markopoulos, P., Eggen, B., Aarts, E., Crowley, J.L. (eds.) EUSAI 2004. LNCS, vol. 3295, pp. 291–302. Springer, Heidelberg (2004)
2. Abrams, M., Phanouriou, C., Batongbacal, A.L., Williams, S.M., Shuster, J.E.: UIML: an appliance-independent XML user interface language. Computer Networks **31**(11), 1695–1708 (1999)
3. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., López-Jaquero, V.: USIXML: a language supporting multi-path development of user interfaces. In: Feige, U., Roth, J. (eds.) DSV-IS 2004 and EHCI 2004. LNCS, vol. 3425, pp. 200–220. Springer, Heidelberg (2005)

4. Park, K., Lee, S.W.: Requirements Elicitation for Mobile Adaptive User Interface based on Concepts from Self-Adaptive Software. In: Korea Conference on Software Engineering 2015. Korean Institute of Information Scientists and Engineers Software Engineering Society (2015)
5. Kühme, T.: A user-centered approach to adaptive interfaces. In: Proceedings of the 1st International Conference on Intelligent user Interfaces, pp. 243–245. ACM (1993)
6. Stephanidis, C., Paramythis, A., Sfyrakis, M., Stergiou, A., Maou, N., Leventis, A., Karagiannidis, C.: Adaptable and adaptive user interfaces for disabled users in the AVANTI project. In: Trigila, S., Mullery, A., Campolargo, M., Vanderstraeten, H., Mampaey, M. (eds.) Intelligence in Services and Networks: Technology for Ubiquitous Telecom Services., vol. LNCS, pp. 153–166. Springer, Heidelberg (1998)
7. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A unifying reference framework for multi-target user interfaces. Interacting with Computers **15**(3), 289–308 (2003)
8. Reichart, D., Forbrig, P., Dittmar, A.: Task models as basis for requirements engineering and software execution. In: Proceedings of the 3rd annual conference on Task models and diagrams (pp. 51-58). ACM (2004)
9. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**(1), 41–50 (2003)
10. Salehie, M., Tahvildari, L.: Self-adaptive software: Landscape and research challenges. ACM Transactions on Autonomous and Adaptive Systems (TAAS) **4**(2), 14 (2009)
11. Wasserman, A. I.: Software engineering issues for mobile application development. In: Proceedings of the FSE/SDP Workshop On Future Of Software Engineering Research, pp. 397-400. ACM (2010)
12. Lee, S. W., Rine, D. C.: Case Study Methodology Designed Research in Software Engineering Methodology Validation. In: SEKE, pp. 117-122 (2004)
13. Akiki, P. A., Bandara, A. K., Yu, Y.: Adaptive model-driven user interface development systems. ACM Computing Surveys, 47(1), In-press (2015)