Generation of Use Cases for Requirements Elicitation by Stakeholders

Junko Shirogane(✉)

Department of Communication, Tokyo Woman's Christian University, Suginami, Japan
junko@lab.twcu.ac.jp

Abstract. Use case diagrams and scenarios are often used in the requirements elicitation phase in software development. It is difficult for developers to create them based on appropriate stakeholders' requirements. Meanwhile, stakeholders can survey existing applications to find functions and interactions that are similar to their requirements. This paper proposes a method to generate the bases of use case diagrams from the operation histories of existing applications. Operation histories are divided into operations of individual windows, and the entire window-switching sequence in an existing application is represented as a directed graph. Then the directed graphs are analyzed to extract the window-switching sequences that correspond to use cases. Finally, use case diagrams are generated.

1 Introduction

Requirements analysis of software development consists of requirements elicitation, analysis, specification, and validation [1]. Based on these phases, various requirements engineering processes have been proposed [2, 3], and these subphases are iteratively performed. Especially, more detailed processes for requirements elicitation phase have been provided [4, 5], however, few processes have been provided for other phases. In addition, many methods to elicit stakeholders' requirements have also been proposed [6]. Thus, because stakeholders often do not understand their requirements, and developers cannot extract their requirements enough [7], researches for requirements elicitation are more active than other phases, and appropriate elicitation of stakeholders' requirements by developers considered to be quite difficult.

In the requirements elicitation phase, use case diagrams and scenarios are often used to describe users' requirements and to communicate between stakeholders and developers. However, developers often have difficulty understanding the stakeholders' intent as use case diagrams and scenarios do not always represent the stakeholders' requirements appropriately. To describe accurately the stakeholders' requirements as use case diagrams and scenarios, it is desirable for the stakeholders themselves to describe the requirements. However, stakeholders without knowledge of software development often have difficulty describing use case diagrams and scenarios.

Meanwhile, in many cases, various applications already exist in the domain that stakeholders' require. Surveying existing applications is one important method to elicit

K. Kim and N. Joukov (eds.), *Information Science and Applications 2017*,
Lecture Notes in Electrical Engineering 424, DOI 10.1007/978-981-10-4154-9_65

requirements [8, 9]. Although it is difficult for stakeholders to describe use case diagrams and scenarios, they can use existing applications as references to realize use case diagrams and scenarios in new applications.

This paper proposes a method to generate the bases of use case diagrams from the operation histories of existing applications operated by the stakeholders. Concretely, the operation histories are divided into individual windows, and window-switching sequences are represented. Next the main window, which is the window where all functions originate, is identified. Then the entire window sequence in the application is represented. Finally, use cases are extracted from the entire window sequence, and use case diagrams are generated. Scenarios are generated from the events of the window-switching sequences in the use cases, and scenario generation method was proposed previously [10]. This paper describes details of use case diagram generation.

The rest of the paper is organized as follows. Section 2 compares this work to other research. Section 3 describes the basic concepts of this method, while Sect. 4 describes the details of the use case diagram generation. Section 1 evaluates this method. Finally, Sect. 6 concludes this paper.

2 Related Works

Several works support creating use case diagrams. They can be classified into three types; languages to describe use cases and requirements, generation of diagrams, and use case patterns.

For languages to describe use cases and requirements, Silva et al. proposed a pattern language to describe use cases [11]. The authors defined patterns to specify the use case type. Next they determined patterns to correspond between the use case and the main entity of a domain model. They also defined patterns to classify four types of interaction blocks, which represented whether scenario events were actor behaviors or system processes. Finally, they described patterns to classify the interaction blocks as detailed behaviors to apply Model Driven Development approaches. For another research, Aspect-oriented User Requirements Notation (AoURN) is a requirements engineering language combining goal-oriented, scenario based, and aspect-oriented [12]. User Requirements Notation (URN) [13] is extended. URN is also a requirements engineering language and consists of Goal-oriented Requirements Language (GRN) [14] for goal modeling and Use Case Maps (UCM) [15] for scenario modeling. By these researches, requirements and specific types of use cases can be described formally. However, it is difficult for stakeholders to use the notations. The method of this paper does not require any software development knowledge to stakeholders.

For generation of diagrams, A method to generate use case diagrams based on the results of analyzing problems by problem frames [16] was also proposed [17]. First, problems were analyzed by the problem frame approach, and subproblems were identified. Next, two types of formal context were created based on the Formal Concept Analysis (FCA) approach [18]. Finally, FCA concept lattices corresponding to relationships between use cases, such as include and extend, and relationships between use cases and actors were created. However, it is difficult for stakeholders to analyze by

problem frame approach. In the method of this paper, stakeholders can easily prepare operation histories of existing applications.

For use case patterns, Cruz focused on data oriented systems and proposed use case patterns [19]. In this method, use cases were classified into two types. One was "Independent use cases" that had interactions with actors and were not include and extend use cases of other use cases. The other was "Dependent use cases" that did not have interactions with actors and were include or extend use cases of other use cases. Then, five use case patterns were defined based on the typical operations for data management. Also, Ko et al. proposed a method to extract use case patterns for supporting completeness of software requirements [20]. First, this method extracts agents (actors) and verbs from event sentences of scenarios, and they are classified into high appearance group and low appearance group. Next, semantic distances between verbs are calculated, verbs in the low appearance group are linked to verbs in the high appearance group. The linked verbs become clusters. Then, use case flow graphs are generated. Finally, strength of the relationships between clusters are calculated. Use case flow patterns are extracted applying thresholds to the calculated values. Although these methods provide or extract use case patterns, support to create use cases are beyond.

3 Basic Concepts

To extract use cases, herein the operation histories that stakeholders try to use in existing applications are analyzed. The operation histories record sequences of users' events. Events indicate operations of widgets in windows and window switching.

3.1 Event Sequence of a Function

In most applications, functions begin from the main window or top page. For a desktop application, when users select a function in the main window, the process of the function corresponding to the menu item begins, and windows related to the function are displayed. Once the function process is complete, the related windows disappear, and only the main window is displayed.

For a web application, the top page has many links and buttons. To initiate the process of the corresponding link or button, users select a link or button and web pages are switched. Once the function process is finished, the top page is displayed again. Hereafter, both main windows and top pages are referred to as "main windows", while all other windows and web pages are called "windows". This method targets applications that functions start from the main window.

3.2 Role of Windows

To extract use cases, the whole window sequence is constructed from all the operation histories. Concretely, event sequences in the operation histories are divided into windows. Next identical windows are merged, and the entire window sequence of an application is represented. Use cases are extracted based on the identified main window.

Thus, identical windows must be identified. Window title indicates the contents assigned to a window [21]. Consequently, window titles can be used to identify windows. Windows with the same title are identical.

In this research, the locations of titles in 15 desktop applications and 15 web applications were surveyed. For desktop applications, titles of 12 applications were located in the title bar, and an application was the top label. For web applications, titles of 10 applications were located in the title bar, and thee applications were the first h1 tag. If titles were displayed in both the title bar and the h1 tag, only the title bar was counted.

Most of the titles were displayed in the title bar for desktop applications, whereas the titles were displayed in both the title bar and the h1 tags for web applications. Thus, the character strings in the title bar are identified as the window title in desktop applications. For web applications, if the character strings in the title bars differ by window, the character strings are identified as the title. On the other hand, if the character strings in the title bars of all windows are the same, the character strings in the first h1 tags of the windows are identified as the title. The titles are used as the window identifiers.

4 Use Case Diagram Generation

Use case diagrams are generated from the operation histories by five steps:

1. Divide the event sequences by windows
2. Merge windows
3. Identify the main window
4. Extract use cases
5. Generate scenarios

4.1 Divide the Event Sequences by Windows

The input and selection of a window consist of some events in a scenario, and a scenario can be represented by a sequence of windows. The operation histories record the events in the executing scenarios. In this research, the operation histories are assumed to record event sequences and window switching. Thus, the events in the operation histories are divided into windows using the entries of window switching. In this manner, the operation histories are represented by the window sequences (Fig. 1).

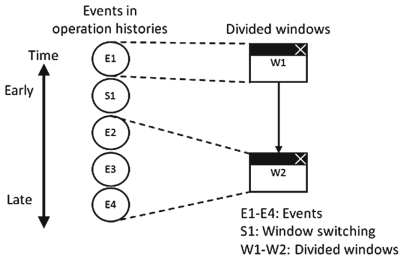

Fig. 1. Schematic of operation histories

4.2 Merge Windows

The operation histories record many scenario executions. Certain scenarios may be executed multiple times. That is, after dividing the operation histories into windows in Sect. 4.1, appearances of identical windows may be exist. As described in Sect. 3.2, window titles are used as identifiers. Thus, to specify a certain window sequence of a scenario, windows with the same titles are merged in this step. Window sequences are represented as a directed graph (Window Switching Graph, WSG). In a WSG, windows and window switches are represented as nodes and edges, respectively. The number of window switches from each window is represented by an edge.

Figure 2 depicts the merging windows. The left side shows the divided windows in Sect. 4.1, while the right side is the merged window sequences. The number "2" indicates that window "W1" switches to window "W2" twice in the operation histories.

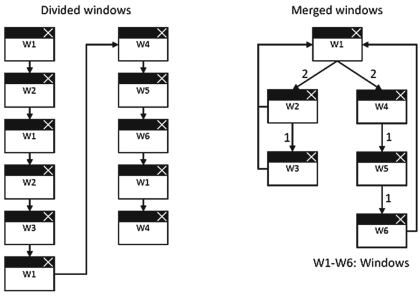

Fig. 2. Depiction of merging windows

4.3 Identify the Main Window

Because a use case consists of several scenarios, execution of a use case also starts from the main window. That is, it is necessary to identify the main window in the WSG. Window switching from the main window to another window occurs using any function, whereas window switching to other windows only occurs when using a specific function. That is, more window switches occur from the main window. Thus, the number of window switches from each individual window is calculated, and the window with the most switches is identified as the main window.

Table 1 shows the number of window switches from individual windows shown in Fig. 2. Because window "W1" has is most switches, it is identified as the main window.

Table 1. Identification of the main window

	W1	W2	W3	W4	W5	W6
Number of window switches	4	2	1	1	1	2

4.4 Extract Use Cases

After constructing a WSG, the use cases are extracted. As described in Sect. 3.1, because all functions start from the main window in this research, individual window sequences from the main window are extracted as use cases. Use cases often have precondition and include relationships [22]. Thus, three patterns are defined to extract use cases from a WSG; basic pattern, pre-condition pattern, and inclusion pattern.

Use case diagrams represent the interactions between actors, including users, and use cases. In this research, the operation histories record the interactions between applications and stakeholders. Thus, the name of the actor is "user", and the use case diagrams are generated as the interactions between the users and the extracted use cases.

4.4.1 Basic Pattern

This is the base for all patterns to extract use cases. For each use case, the main window is the starting point, and each branch of a window sequence from the main window is extracted as a use case. Because a use case often consists of several scenarios (e.g., main scenarios, alternative scenarios, and exceptional scenarios [23]), branches of window sequences from windows except the main window are recognized as different scenario executions in a use case.

Figure 3 shows an example of a WSG to extract use cases based on basic pattern where the two branches of window sequences ("M" -> "W1" and "M" -> "W6") are from the main window ("M"). Two use cases are extracted, such as U-B1 and U-B2. Figure 4 shows the generated use case diagram from the WSG in Fig. 3.

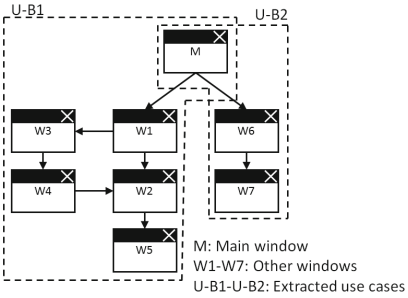

Fig. 3. Example of basic pattern

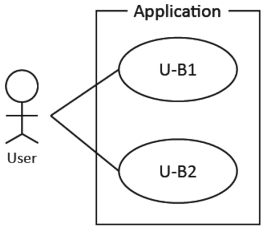

Fig. 4. Generated use case diagram from basic pattern

4.4.2 Pre-condition Pattern

Some interactions are performed before the main window is displayed. For example, in an internet banking application, users login to the application before using functions of the application. In this case, the login is always performed before the main window. Hence, completing the login is a pre-condition for functions in the internet banking application.

In this pattern, window sequences from the main window are extracted as use cases (post-use cases) based on basic pattern in Sect. 4.4.1. Window sequences before the main window are also extracted as use cases (pre-use cases). Then event sequences of the operation histories in the pre and post-use cases are analyzed. If all event sequences of the pre-use cases are executed before the post-use cases, the pre-use case is identified as a pre-condition of the post-use case.

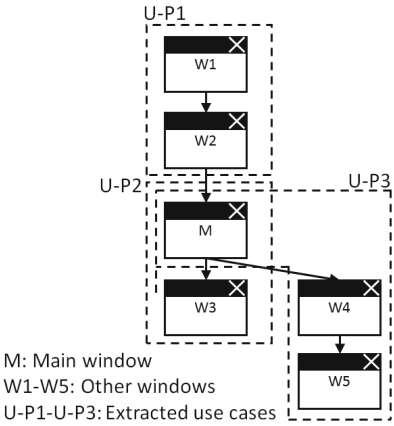

Fig. 5. Example of pre-condition pattern

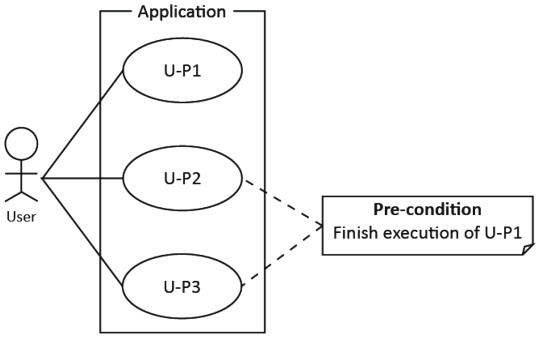

Fig. 6. Generated use case diagram from pre-condition pattern

Figure 5 shows an example of a WSG to extract the use case of a pre-condition and the extracted use cases. Because the window sequence (“W1” -> “W2”) is always displayed before the main window, it is extracted as pre-use case “U-P1”. Window

sequences after the main window “M” are extracted as post-use cases “U-P2” (“M” -> “W3”) and “U-P3” (“M” -> “W4” -> “W5”) based on the basic pattern. Then the pre-use case “U-P1” is identified as the pre-condition for post-use cases “U-P2” and “U-P3”. Figure 6 shows the generated use case diagram from the WSG in Fig. 5.

4.4.3 Inclusion Pattern

Different use cases sometimes have common event sequences, and the event sequences are often represented by dependent use cases. That is, different use cases can employ the same event sequences as dependent use cases.

Figure 7 shows an example where two use cases can be extracted based on the basic pattern in Sect. 4.4.1, such as the window sequences “M” -> “W1”-> “W2” -> “W3” -> “W4” (use case (a)) and “M” -> “W5” -> “W6” -> “W2” -> “W3” -> “W7” (use case (b)). Both use cases (a) and (b) include the common window sequence “W2” -> “W3”.

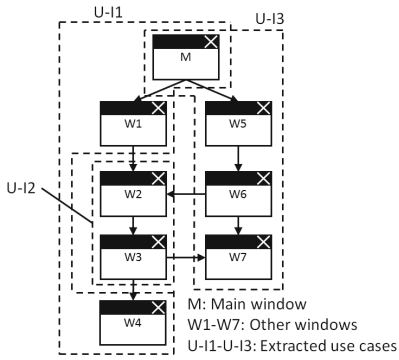

Fig. 7. Example of inclusion pattern

Thus, this common window sequence is separated from use cases (a) and (b) and extracted as a dependent use case. That is, U-I1, U-I2, and U-I3 are extracted as use cases. Use case U-I2 is included by use cases U-I1 and U-I3. Figure 8 shows the generated use case diagram from the WSG in Fig. 7.

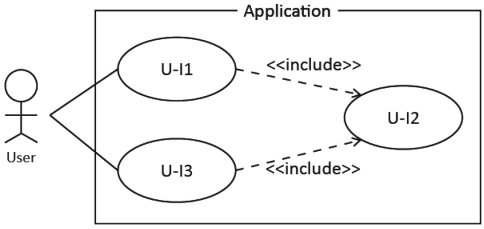

Fig. 8. Generated use case diagram from inclusion pattern

5 Evaluation

The appropriateness of the use case diagrams generation was evaluated using five applications; internet banking (App A), food delivery (App B), ticket reservation (App C), travel (App D), and e-Learning (App E). Table 2 shows the results of use case generation, where "Actual (Basic)", "Actual (Pre)", and "Actual (Include)" indicate number of use cases based on the extraction patterns of the basic, pre-condition, and inclusion patterns in Sect. 4.4, respectively by manual. Hereafter, the strategy specifying use cases by the author is called "actual". "Extract (Basic)", "Extract (Pre)", and "Extract (Include)" indicate the number of use cases extracted by this method.

Table 2. Results of use case generation

	Actual (Basic)	Actual (Pre)	Actual (Include)	Extract (Basic)	Extract (Pre)	Extract (Include)
App A	2	1	2	2	1	2
App B	1	0	1	1	0	1
App C	3	0	0	3	0	0
App D	1	0	0	2	0	2
App E	6	1	0	6	1	0

The number of use cases between the actual and this method differed in App D. For App D, the actual use case was only one, but this method extracted two use cases for the basic pattern for hotel selection. The window titles included the area names of hotels, preventing the windows from being merged in the step of Sect. 4.2. Thus, window sequences of the hotel selection from the main window were separated by area, and they were extracted as different use cases.

In addition, after the hotel selection, login and inputting customer information were required. This sequence consisted of two windows (e.g., login and inputting customer information), and the actual use case indicated that they were part of one window sequence in the hotel selection. However, the inclusion pattern in this method extracted two use cases for this sequence. Because this sequence was executed after the two use cases, they were extracted by the inclusion pattern. In addition, once login was completed for the first hotel selection, login for a second hotel selection was not required. To select a second hotel, only the customer information must be inputted. Because both extracted use cases included window sequences that executed/did not execute login, login and inputting customer information were extracted as different use cases.

For "App E", although the number of use cases between the actual and this method were the same, there were window sequences that this method did not use for use case extraction. In "App E", the main window was identified as the window of functions to manage a lecture. However, the window title included the lecture name. Thus, the title of the window differed by lectures and were not merged in the step of Sect. 4.2. Consequently, window sequences of one lecture but not the other are used in the use case extraction. In the actual case, the windows were merged because they were recognized as the same window.

These results reveal some problems using window titles. Parsing a natural language is required to merge windows. If the area names of "App D" and the lecture names of "App E" can be deleted by parsing a natural language, windows that should be merged in this method can be merged. However, words should not always be deleted. Thus, strategies to delete words must be carefully considered.

Although problems merging windows due to the window titles were elucidated, other problems were not identified. Thus, except the problems, this method could be confirmed that use cases could be extracted appropriately.

6 Conclusion

Because developers difficultly define stakeholders' requirements appropriately, it is desirable for stakeholders themselves to use case diagrams and scenarios that reflect their requirements directly. To facilitate the description of use case diagrams and scenarios, this paper proposes a generation method that uses the operation histories of existing applications.

To generate use case diagrams in our method, first, the event sequences in the operation histories are divided into windows. Then windows with the same titles are merged. Next the window with the most window switches is identified as the main window. Then the use cases are extracted based on the main window by three patterns: basic, precondition, and inclusion patterns. To confirm the appropriateness of the generation, five existing applications with different domains were operated. Then the use case diagrams and scenarios were generated from the operation histories. Although there were some problems, the use cases and scenarios were almost appropriately extracted and generated.

In the future strategies to merge windows to extract use cases must be considered. Because window titles included proper nouns, some windows were not merged, but should have been in the evaluations. Appropriately merging windows can realize more suitable use case extraction. To realize this, parsing window titles as a natural language could be considered.

Next, the types of users must be identified. Currently, the actors in the use case diagrams are only "users". However, use cases that users can execute often differ by the types of user, and the use case diagrams must represent each type of user. To realize this, recording users operating the applications and then analyzing the use cases that each user operates can be considered. Additionally, naming the extracted use cases should be considered. Although the names of buttons or links for window switching from the main window to the next window can often be used as the name of the use case, this is inappropriate in some cases.

References

1. Bourque, P., Fairley, R.E.: SWEBOK V3.0 Guide to the Software Engineering Body of Knowledge. IEEE (2014)
2. Lamsweerde, A.: Requirements Engineering: From System Goals to UML Models to Software Specifications. Wiley, Hoboken (2009)
3. Wiegers, K.: Software Requirements, 3rd edn. Microsoft Press, Redmond (2013)
4. Potts, C., Takahashi, K., Anton, A.I.: Inquiry-based requirements analysis. IEEE Softw. **11**(2), 21–32 (1994)
5. Christel, M.G., Kang, K.C.: Issues in requirements elicitation, Technical report CMU/SEI-92-TR-12 (1992)
6. Carrizo, D., Dieste, O., Juristo, N.: Systematizing Requirements Elicitation Technique Selection. Inf. Softw. Technol. **56**(6), 644–669 (2014)
7. Alexander, I.F., Beus-Dukic, L.: Discovering Requirements How to Specify Products and Services. Wiley, Hoboken (2009)
8. Laplante, P.A.: Requirements Engineering for Software and Systems, 2nd edn. Auerbach Publications, Boca Raton (2013)
9. Kotonya, G., Sommerville, I.: Requirements Engineering: Processes and Techniques. Wiley, Hoboken (1998)
10. Shirogane, J.: Scenario description method based on existing software operation history. In: Proceedings of 9th International Conference on Software Technologies (ICSOFT2014) (2014)
11. Silva, A.R., Savi, D., Vlaji, S., Antovi, I., Lazarevi, S., Stanojevi, V., Mili, M.: A pattern language for use cases specification. In: Proceedings of the 20th European Conference on Pattern Languages of Programs (2015)
12. Mussbacher, G., Amyot, D., Whittle, J.: Composing goal and scenario models with the aspect-oriented user requirements notation based on syntax and semantics. Moreira, A., Chitchyan, R., Araújo, J., Rashid, A. (eds.) Aspect-Oriented Requirements Engineering Part II, Springer, Heidelberg (2013)
13. Z.151: User Requirements Notation (URN) - Language definition. http://www.itu.int/rec/T-REC-Z.151/en. Accessed 14 Oct 2016
14. GRL. http://www.cs.toronto.edu/km/GRL/. Accessed 14 Oct 2016
15. Buhr, R.J.A.: Use case maps as architectural entities for complex systems. IEEE Trans. Softw. Eng. **24**(2), 1131–1155 (1998)
16. Jackson, M.: Problem Frames Analyzing and Structuring Software Development Problem. Addison Wesley, Boston (2000)
17. Imam, A.A., Hamza, H.S., Moneim, R.A.: Automated generation of use case diagrams from problem frames using formal concept analysis. In: Proceedings of 10th International Conference on Information Technology: New Generations (2013)
18. Wille, R.: Restructuring lattice theory: an approach based on hierarchies of concepts. In: Rival, I. (ed.) Ordered Sets, vol. 83, pp. 445–470. Springer, Netherlands (1982)
19. Cruz, A.M.R.: A pattern language for use case modeling. In: Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development (Modelsward 2014) (2014)
20. Ko, D., Park, S., Kim, S., Hwang, M.: Automatic use case flow pattern generation using verb clustering. Int. J. Softw. Eng. Appl. **9**(7), 201–212 (2015)

21. Guidelines. https://msdn.microsoft.com/ja-jp/library/windows/desktop/dn688964(v=vs.85).aspx. Accessed 14 Oct 2016
22. Object Management Group, OMG Unified Modeling Language TM (OMG UML) Version 2.5 (2013)
23. Schneider, G., Winters, J.P.: Applying Use Cases: A Practical Guide. Addison-Wesley, Boston (1998)