An Automatic Subsystem Grouping Scheme using Use Case Dependency Graph for Large Complex Systems

Nanchaya Khruenahong and Wiwat Vatanawood

Abstract—As the software system becomes large and complex, the UML use case diagram which is widely used to capture their requirements is consequently difficult and complicated. According to the huge numbers of UML elements found in the initial use case diagram, the subsystem grouping could be used to alleviate the complexity.

In this paper, an automatic subsystem grouping scheme is proposed. The use case dependency graph is introduced as an alternative technique to identify the structural cohesion of the use cases and their relations. In our approach, the prerequisite preparation of the initial use case diagram is needed to ensure the well-formedness style and proper naming convention beforehand.

Moreover, the refinement of the subsystem grouping is also proposed using the use case naming convention approach. The final version of the result use case diagram with the relevant subsystems has been reviewed to ensure the improvement of the readability and understandability.

Index Terms—UML Use Case Diagram, Large Complex System, Complexity, Use Case Dependency Graph

I. INTRODUCTION

Currently, the UML Use Case diagram is one of the popular and standard tools for object-oriented modeling [1], [2], [3], [4]. It is a visual modeling language that can be used to capture the high level views of the behavioral requirements of the system [5]. A system analyst usually draws a use case diagram accordingly to represent the first draft of the expected target system’s behavior. Unfortunately, the current software system becomes larger and more complex, and the result huge complex figure of use case diagram seems to be difficult for reading and understanding.

Information hiding technique and subsystem design are focused as the key issues in this paper. They help the system analyst organize and simplify the huge numbers of the UML elements within the use case diagram [6], [7]. However, the competency on how to specify a relevant subsystem is still not common for the analyst. It should be helpful if there is a guideline regarding that capability. In this paper, we propose an automatic subsystem grouping scheme, using use case dependency graph, to ease the drawing of the very first draft of a total use case diagram. The boundary of the subsystems would be recommended by the proposed scheme to increase the readability and understandability of the use case diagram. The paper is organized as follows. Section 2 reviews related works. Section 3 describes our proposed automatic subsystem grouping scheme. In section 4, we demonstrate the case study. Finally, section 5 concludes the paper.

II. RELATED WORKS

To manage the complexity of requirements captured by the UML use case diagrams for large complex systems is not well addressed in general. However, several best practices on how to draw the quality UML use case effectively are concerned to alleviate the complexity. At the beginning, some templates and well-formedness rules [8] were formally defined, using set theory and logic, to ensure the syntactical constraints among use case elements and some guidelines are proposed to be followed. Moreover, the visualization and Aesthetics of the layout of use case diagram apparently increases the readability and understandability. [9] proposed the deterministic layout algorithm to support drawing use case diagram nicely. Whilst, [5] also suggested how to do the use case naming convention in order to communicate the proper semantic of the target system.

The graphics will stay in the “second” column, but you can drag them to the first column. Make the graphic wider to push out any text that may try to fill in next to the graphic.

For the large complex system, some best practices on the top-down approach are still the effective way to compromise with the complexity and the information coverage needed in the modeling. Some examples in [10], [11] show the evidences of how to use a hierarchical framework for use case diagram of large complex embedded systems.

III. OUR PROPOSED AUTOMATIC SUBSYSTEM GROUPING SCHEME

A. Preparation of The Initial Input Use Case Diagram

In our approach, a raw input use case diagram, probably
with a huge number of elements, should be initially prepared to conform to the well-formedness rules (WFR) of use case diagram [8] in order to ensure the syntactical consistence and completeness of the relations and their constraints among elements in the use case diagram. The WFR includes the techniques how to draw Actor, Use Case, Association, Generalization, <<include>> and <<extend>> relationship, etc.

In addition, we expect that all use cases should be named according to the recommendation in [5], in which an alternative the best practices of use case naming convention has been defined. In practice, a use case name is comprised of “active verb” and immediately followed by “direct object”. For example, a use case named “View Customer Information” has an active verb - “View”, followed by a direct object - “Customer Information”.

With the given raw input as mentioned above, we would be ready to perform the subsystem grouping in two passes: (Pass I) The subsystem grouping scheme using use case dependency graphs and (Pass II) The refinement of the subsystem grouping using use case naming convention.

B. Well-Formedness Rules [8] Revisited

According to [8], well-formedness rules are a set of syntactical constraints of UML elements and their relations, especially for the UML use case diagram. The following sentences show some examples of the WFRs written in the natural language:

An actor must have a name and must be associated with at least one use case. Actors are not allowed to interact with other actors. A use case must have a name and every use case is involved with at least one actor. The <<include>> relationship links the source use case to the destination use case. The rest of the WFRs are described in [8].

C. Part I: The Subsystem Grouping Scheme using Use Case Dependency Graphs

The raw input use case diagram, which is prepared according to section 3A, will be transformed into a set of use case dependency graphs and the first version of the result subsystems is then identified using Algorithm 1. The definitions and algorithm are shown as follow:

Definition 1: Use Case Dependency Graph, DG. A use case dependency graph is tuple $DG = (N, E)$. We define $N = \text{ACTOR} \cup \text{USECASE}$ and $E = \text{ASSOC} \cup \text{REL} \cup \text{GEN}$. ACTOR is a set of actors and USECASE is a set of use cases in the diagram. ASSOC is a set of edges on ACTOR x USECASE, REL is a set of edges on USECASE x USECASE, and GEN is a set of edges on USECASE x USECASE.

We also define $\text{REL} = \{\text{REL-INC}\} \cup \{\text{REL-EXT}\}$ and $\text{GEN} = \{\text{REL-GEN}\}$ to cope with the type of relationships and generalization.

Definition 2: <<include>> relationship, REL-INC. An <<include>> relationship is 2-tuple $\text{REL-INC} = (\text{baseUC}, \text{incUC})$, where baseUC is a set of the base use cases, and incUC is a set of the included use cases.

Definition 3: <<extend>> relationship, REL-EXT. An <<extend>> relationship is 2-tuple $\text{REL-EXT} = (\text{baseUC}, \text{extUC})$, where baseUC is a set of the base use cases, and extUC is a set of the extending use cases. For <<extend>> relationship, we intentionally define the direction of the edge starting from the base use case to the extending use case.

Definition 4: Generalization relationship, REL-GEN. A generalization relationship is 2-tuple $\text{GEN} = (\text{superUC}, \text{subUC})$, where superUC is a set of the parent use cases, and subUC is a set of the child or subordinate use cases. For generalization relationship, we intentionally define the direction of the edge starting from the parent use case to the child use case.

Algorithm 1: Subsystem Grouping

Input: A set of dependency graphs $TDG = \{DG\}$ generated by definition 1-4

a) Dropout the DG, which has number of nodes less than 3

\[\text{For each } DG_i, \text{ If } \text{NumberOfNode}(DG_i) < 3 \text{ then delete } DG_i \text{ from } TDG\]

b) Repeatedly find the subsystems

\[\text{Do while } TDG \neq \{\}\]

\[\text{Find the } DG_i \text{ that has the maximum number of nodes and call it } MaxDG\]

\[\text{For each } DG_i, \text{ If the set of nodes of } MaxDG \cap \text{the set of nodes of } DG_i \neq \{\} \text{ then}\]

\[\text{o The new } MaxDG \text{ equals } MaxDG \cup DG_i\]

\[\text{O Delete } DG_i\]

\[\text{Define } MaxDG \text{ as a subsystem.}\]

D. Pass I: The Subsystem Grouping Scheme using Use Case Dependency Graphs

As we mentioned earlier, the raw input use case diagram should be prepared using both in the well-formedness styles and properly naming convention.

The intention of the refinement is to reconsider the dropout use cases during step a) in the algorithm 1 (Subsystem Grouping) and include them into the relevant subsystems. We propose that the use cases with the same “direct object” should be in the same subsystem.
Fig. 1. PLIS use case diagram [12]

Fig. 2. Use Case Dependency graph
Fig. 3. PLIS use case diagram obtained Use Case Dependency Graph
Fig. 4. PLIS Four subsystem of grouping PLIS use case diagram
IV. THE CASE STUDY

This section demonstrates the automatic subsystem grouping scheme by using a case study called “The personal language instruction scheduler (PLIS)”. The raw input use case diagram of the PLIS is shown in fig. 1. We follow the preparation steps in section 3A so that the use case is now in the well-formedness styles and the proper naming convention is also ensured.

With the raw input use case diagram, the set of dependency graphs are defined and shown in fig. 2. We found that 19 dependency graphs are generated. The graph number 10, 17, 12 and 18 are respectively selected as a MaxDG in step b) to form each subsystem. The graph number 5, 6, 8, 9, 15 and 19 have been dropout and will be reconsidered in the refinement processing.

As the result, four subsystems are identified and shown in fig. 3. Each subsystem shows the appropriate use cases and their relationship. However, the refinement of the subsystems has been conducted and some of the dropout use cases in the prior step are included as shown in fig. 4.

The final use case diagram with the relevant subsystems has been reviewed and successfully accepted.

V. CONCLUSION

To read and understand the use case diagram for large complex system is a hard work for system analyst. The automatic subsystem grouping scheme using use case dependency graph is proposed. We found that the initial use case diagram should be prepared in the systematic way. The well-formedness rules and the proper naming convention mentioned earlier are still recommended in the stage of preparation of the initial use case diagram. The refinement of the subsystem grouping is needed to ensure the completeness of the final result.

Practically, the metadata of the UML use case diagram would be represented in .XMI file format and the file could be automatically processed using our proposed scheme.

REFERENCES


Nanchaya Khruahong
She is a graduate student of Computer Engineering at Faculty of Engineering, Chulalongkorn University. Her research interest is Software Engineering.

Wiwat Vatanawood
He is currently an associate professor of Computer Engineering at Faculty of Engineering, Chulalongkorn University. His research interests include formal specification methods, software architecture.