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

Nanchaya Khrueahong 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 system 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 the 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.

Manuscript received December 23, 2011; revised January 09, 2012.

Nanchaya Khrueahong is with Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok Thailand (e-mail: nanchaya.k@student.chula.ac.th).

Wiwat Vatanawood is with Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok Thailand (e-mail: wiwat@chula.ac.th).

II. Related works

To manage the complexity of requirements captured by the UML use case diagrams for large complex system 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 = ACTOR ∪ USECASE and E = ASSOC ∪ REL ∪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 REL = {REL-INC} ∪ {REL-EXT} and GEN = {REL-GEN} to cope with the type of relationships and generalization.

Definition 2: <<include>> relationship, REL-INC. *An <<include>> relationship is 2-tuple REL-INC = (baseUC, 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 REL-EXT = (baseUC, 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 GEN = (superUC, 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 $DG_i$ which has number of nodes less
   than 3

     For each $DG_i$ ,

         If NumberOfNode($DG_i$) < 3 then delete $DG_i$
         from TDG

b) Repeatedly find the subsystems

   Do while  $TDG \neq \{\}$

     {
       Find the $DG_i$ that has the maximum number of
       nodes and call it MaxDG

       For each $DG_i$
         {
           If the set of nodes of MaxDG $\cap$ the set of
           nodes of $DG_i \neq \{\}$ then

           o The new MaxDG equals MaxDG $\cup DG_i$

           o Delete $DG_i$

         }

         Define MaxDG 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.

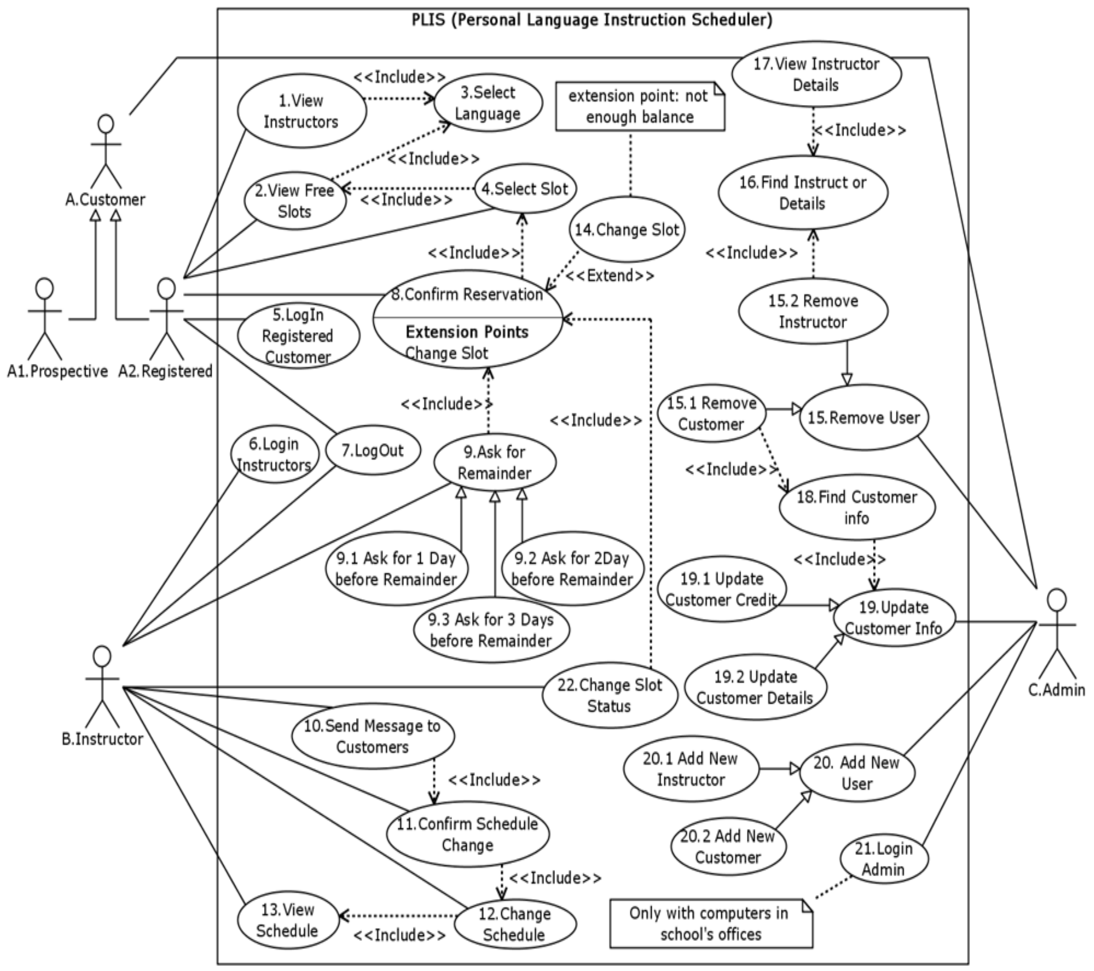

Fig. 1. PLIS use case diagram [12]

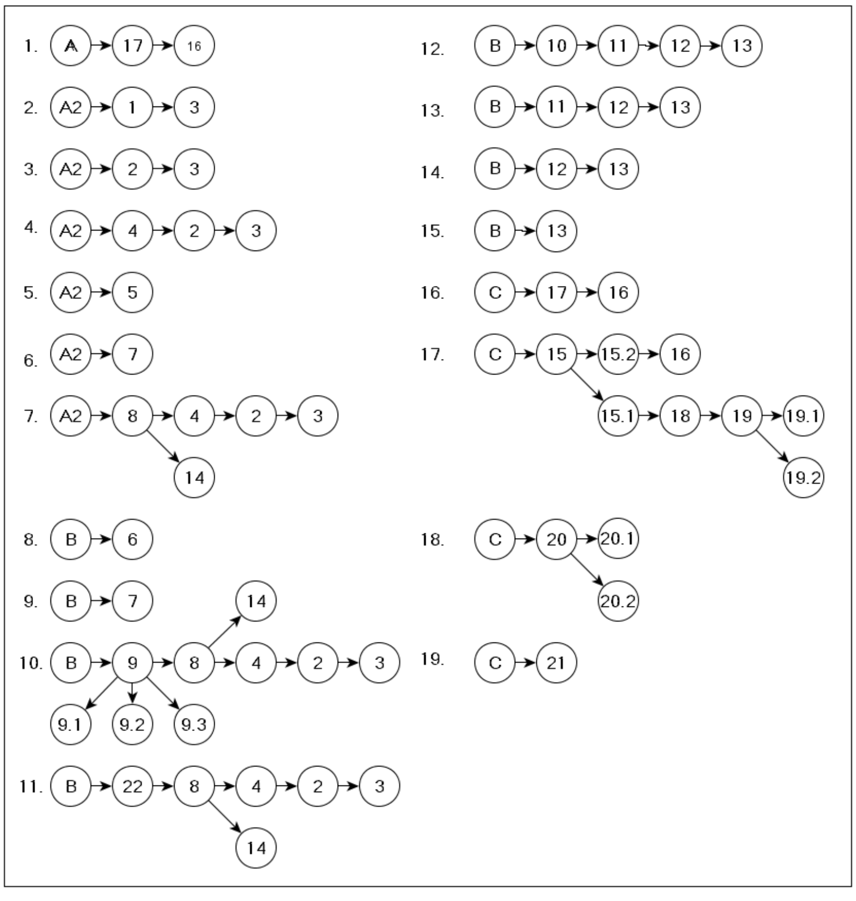
1. A → 17 → 16
2. A2 → 1 → 3
3. A2 → 2 → 3
4. A2 → 4 → 2 → 3
5. A2 → 5
6. A2 → 7
7. A2 → 8 → 4 → 2 → 3; 8 → 14
8. B → 6
9. B → 7
10. B → 9 → 8 → 4 → 2 → 3; 8 → 14; 9 → 9.1, 9.2, 9.3
11. B → 22 → 8 → 4 → 2 → 3; 8 → 14
12. B → 10 → 11 → 12 → 13
13. B → 11 → 12 → 13
14. B → 12 → 13
15. B → 13
16. C → 17 → 16
17. C → 15 → 15.2 → 16; 15 → 15.1 → 18 → 19 → 19.1; 19 → 19.2
18. C → 20 → 20.1; 20 → 20.2
19. C → 21

Fig. 2. Use Case Dependency graph

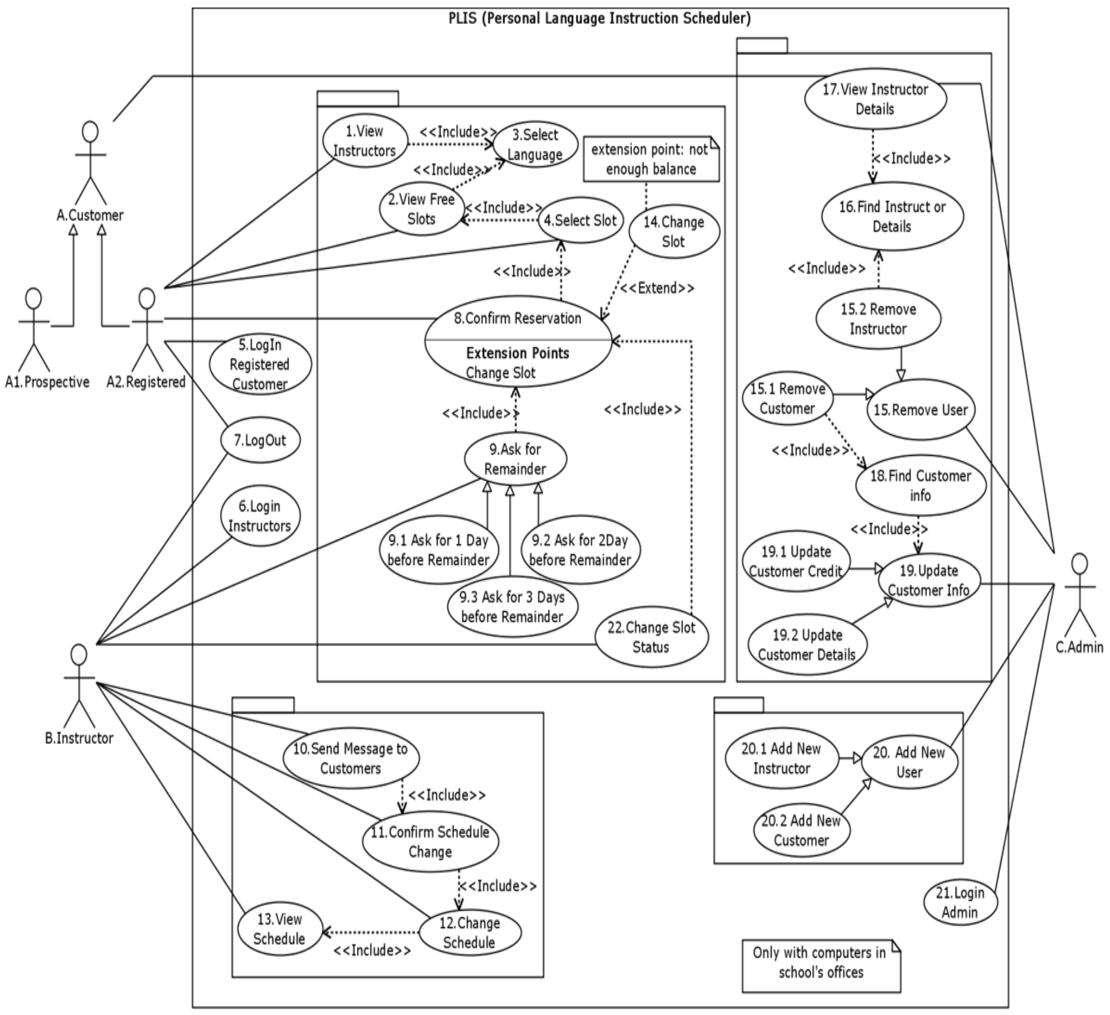

Fig. 3. PLIS use case diagram obtained Use Case Dependency Graph

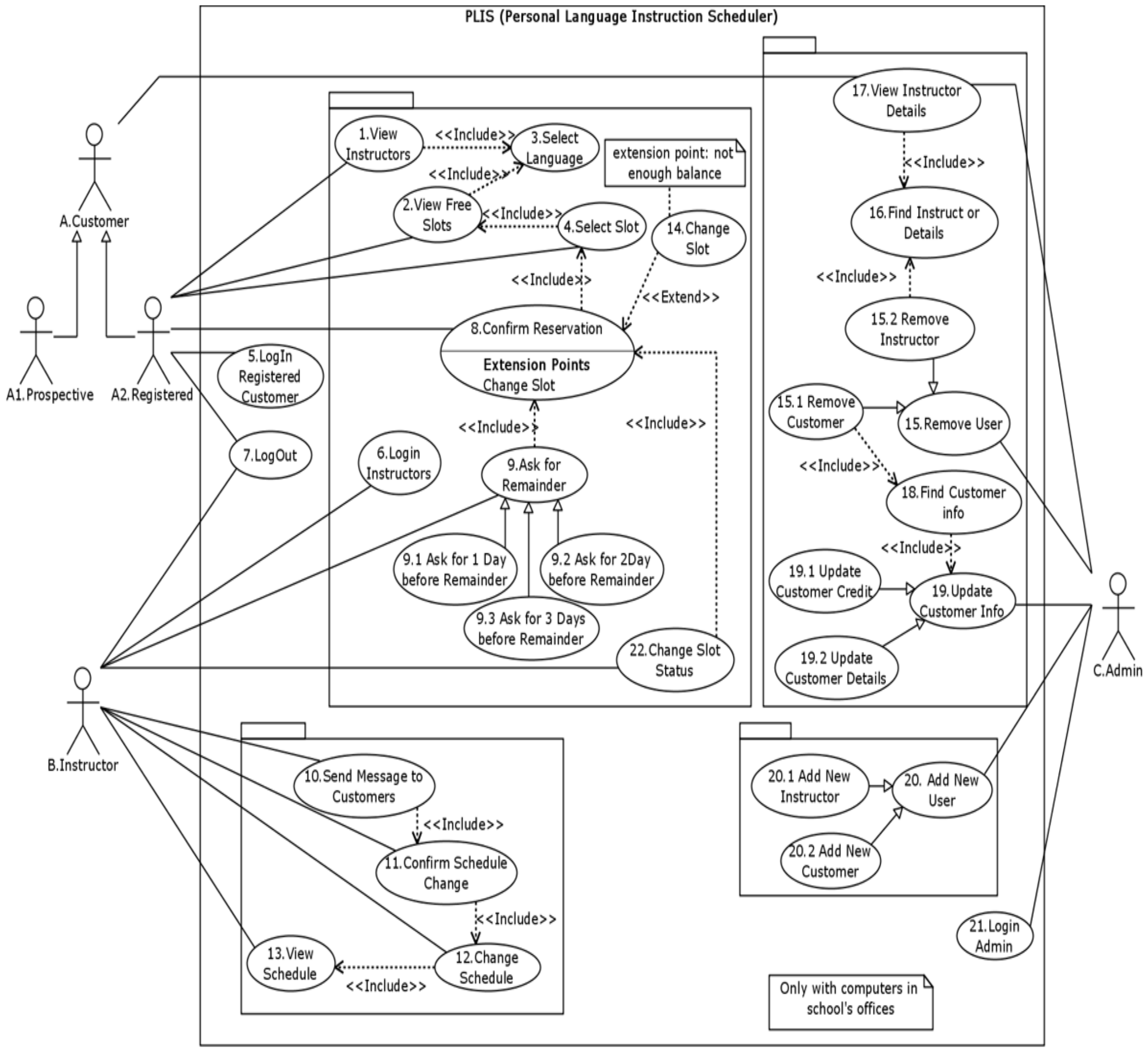

Fig. 4. PLIS Four subsystem of grouping PLIS use case diagram

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

[1] B. Dobing and E. Parson, "Dimensions of UML Diagram Use: A Survey of Practitioners," *Journal of Database Management 19(1)*, 2008, pp. 1-18.

[2] G. Booch, J. Rumbaugh, and I. *Jacobson. The Unified Modeling Langueage User Guide*. USA: Addison-Wesley, 2001.

[3] M. N. Alanazi, "Basic Rules to Build Correct UML Diagram," Proceeding of International Conference on New Trends in Information and Service Science (NISS'09), 2009, pp. 72-76.

[4] Y. Labiche, "The UML Is More Than Boxes and Lines," Chaudron, M.R.V. (ed) Models in Software Engineering, *Springer-Verlag Berlin Heidelberg*. LNCS, Vol. 5421, pp. 375-386, 2009.

[5] A. Cockburn, *Writing Effective Use Cases*. Addison-Wesley, 2001.

[6] G.A. Kohring, "Complex Dependency in large Software Systems," *Journal of Advances in Complex Systems*, 2009, Vol. 12, No. 6, pp. 565-581.

[7] C. R. Myers, "Software Systems as Complex Network: structure, function, and evolvability of software collaboration graphs," *Phys. Rev*. E 68, 046116, 2003.

[8] N. Ibrahim, R. Ibrahim, M.Z. Saringat, D. Mansor, and T. Herawan, "On Well-Formedness Rules for UML Use Case Diagram," Proceedings of the International Conference on Web Information System and Mining (WISM'10), *Springer-Verlag Berlin Heidelberg*, 2010, pp. 432-439.

[9] H. Eichelberger, "Automatic Layout of UML Use Case Diagrams," Proceeding of the 4th ACM symposium on Software visualization (SoftVis'08), 2008.

[10] K. S. Lew, T. S. Dillon, and K. E. Forward, "Software Complexity and Its Impact on Software Reliability," *IEEE Transaction on Software Engineering*, Vol. 14, No. 11, November 1988.

[11] E. Nasr, J. McDermin, and G. Bernat, "A Technique for managing Complexity of Use Cases for Large Complex Embedded Systems," Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISOEC'02), 2002, pp. 225-232.

[12] Farid. (2007, Nov 25). An On-Line Software System for Allocating Personnel Instruction Sessions [Online], Available: http://www.getacoder.com/projects/uml_class_seqence_diagram_63447_shortlist.html?ord=biddate.

Nanchaya Khrueahong
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.