Conducting Requirements Evolution by Replacing Components in the Current System

Haruhiko Kaiya Kenji Kaijiri
Department of Information Engineering, Faculty of Engineering
Shinshu University
4-17-1 Wakasato, Nagano City, 380-8553, Japan
kaiya@cs.shinshu-u.ac.jp kaijiri@cs.shinshu-u.ac.jp
http://www.cs.shinshu-u.ac.jp/~kaiya/

Abstract

As new software components become available for an existing system, we can evolve not only the system itself but also its requirements based on the new components. In this paper, we propose a method to support requirements evolution by replacing a component with another component, and by changing the current requirements so as to adapt to the new component. To explore the possibilities of such a replacement, we use the technique of specification matching. To change the current requirements, we modify the structure by following the concept of Design by Contract.

1. Introduction

Requirements of a software system are originally acquired from users in several ways, e.g., interviews or questionnaires. When a system already exists and several new components become available, analysts can explicitly compare components in the current system with the new components, so as to let the users explore the possibilities of their new system. Therefore, the emergence of new available components can encourage the evolution of the system requirements. In this paper, we propose a method for conducting requirements evolution based on the replacement of current components.

Many techniques for supporting requirements acquisition focus on the behavior of users in their daily work [1], because we intend to replace several parts of current tasks with computer systems. Such techniques alone are not enough to encourage users to explore the new possibilities that are created by introducing computer systems, because users do not know enough about what and how the systems affect the current task.

Recently, many software components are introduced everyday. In practice, the method for selecting a suitable component from libraries has become one of the most important topics in requirements engineering [2]. Also, the number of such components seems to be larger than the number of user requirements. Therefore, we can stimulate user requirements by showing the possible replacements of current components and the effects of such replacements. We call such replacements and their effects *requirements evolution*.

To conduct requirements evolution, we find three technical issues:

1. How to clarify the differences between the old components and new ones.
2. How to explore new possibilities of the system with new components.
3. How to select the most suitable replacement among the possible replacements.

In this paper, we explore the first issue by using specification matching [3]. In the specification matching technique, each software component is specified by a signature and pre/post-conditions, which is the traditional way of specification. The differences between the components are characterized using the signature and the conditions. To resolve the second issue, we use the concept of Design by Contract [4]. In Design by Contract, each software component is also specified in the traditional way. In addition, the relationship between a component and the other parts of the system is modeled as a *contract*, so that the responsibilities of a component and the other parts are clearly disjointed. We do not take up the last issue in this paper, but discuss the possibilities in the last section.

The rest of this paper is organized as follows. In Section 2, we introduce the outline of our method, and Section 3 presents examples for explaining this method. In the last section, we discuss future directions of this method.

2. Method

In Figure1, we outline our method. We regard a sequence of *activities* as a requirements specification. Here we represent such a sequence using an *Activity Diagram* in UML.

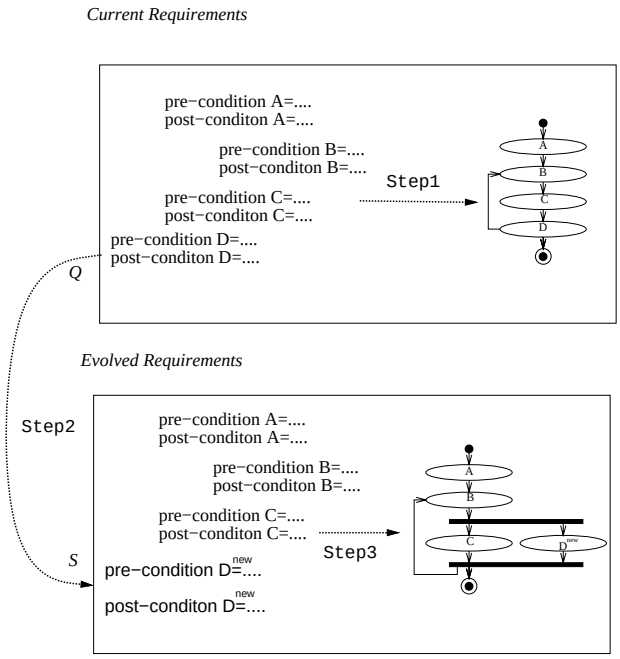

Figure 1. Outline of this method

We regard each activity shown by an ellipse as a component used in the system, and each activity is specified by pre/post-conditions. Because each activity is specified by conditions and the connections between the activities adhere to the principle of Design by Contract, the topology of the diagram is constrained by the conditions (Step1).

Step2 in Figure1 shows a replacement of a component, which can be a trigger of the requirements evolution. We call the set of components which are directly connected by arcs to the replaced component *the client* of the replaced component.

We use specification matching [3] for comparing the replaced components. The study of specification matching includes a method for finding a specification S which matches another specification Q. One type of matching is called pre/post match and defined as follows:

$$match_{pre/post}(S, Q) = (Q_{pre}\ \mathcal{R}_1\ S_{pre}) \wedge (S'\ \mathcal{R}_2\ Q_{post})$$

where S' is either S_{post} or $S_{pre} \wedge S_{post}$ and $\mathcal{R}_{\mathrm{i}}$ is either $\Rightarrow$ or $\Leftrightarrow$. In the context of our method, S corresponds to a candidate of the replacing component, and Q corresponds to the current (replaced) component. In our method, we introduce the principle of Design by Contract, which argues that the responsibilities of a component and its client should be clearly disjointed. Therefore, pre/post match is suitable for our comparison.

Specification matching deals with "matching" of alternative components. Therefore, the client of the component is normally fixed. As a consequence, the pre-condition may be weakened and the post-condition should be preserved, so the logical relationships of $\mathcal{R}_{\mathrm{i}}$ are either $\Rightarrow$ or $\Leftrightarrow$. In contrast, because its client does not need to be fixed in our study, the relationships could be $\Rightarrow$, $\Leftarrow$ or $\Leftrightarrow$. When $\mathcal{R}_{\mathrm{i}}$ is $\Leftarrow$, which is not supported in specification matching, its client should be modified.

Because each pre/post-condition normally consists of several predicates, pre/post match cannot simply cover all kinds of replacements. In this method, we will decompose a component so as to be able to apply the pre/post match, and we do not handle a component which cannot be decomposed adequately. In this paper, we use Z notation for describing pre/post-conditions.

In the context of Design by Contract, the conjunction of post-conditions which specify the precedent components should satisfy the pre-condition of the replacing component. The post-condition of the replacing component should satisfy each pre-condition of succeeding components. Therefore, if another contract can or should be made by replacing a component, we explore the requirements evolution

- by modifying the topology of the Activity Diagram
- or by replacing precedent and/or succeeding components.

Though we should manually find where and how to modify and to replace, we introduce a kind of strategy for modifying the topology as follows:

Rule1 [$\mathcal{R}_{1\mathrm{or}2}$ in $match_{pre/post}(S, Q) = \Rightarrow$]: an activity of the new component is moved forward in the sequence of activities.

Rule2 [$\mathcal{R}_{1\mathrm{or}2}$ in $match_{pre/post}(S, Q) = \Leftarrow$]: an activity of the new component is moved backward in the sequence of activities.

Note that this strategy is only valid when the conditions are gradually strengthened.

Then we can formally check whether the modifications and replacements are valid or not by checking the contracts between the components. Step3 in Figure1 shows one such resolution. Because we normally have several possibilities of evolution, we leave the selection of a suitable evolution to users now.

3. Examples

In this section, we introduce two examples of requirements evolution. From these examples, we will show that the replacement of components can encourage the requirements evolution.

3.1. Coaching Presentation Techniques

When you have a plan to give a presentation in the near future, you will practice your presentation and have comments from your colleagues, and refine your presentation.

In this example, we show a requirements specification for supporting such a task so as to discuss the replacement which seems to refine the task.

Step1: Suppose you use static materials of presentation, e.g., transparencies in your presentation. You cannot modify your material during the discussion for comments. This can be formally described as a pre-condition of an activity "Modify Material" as follows:

$$\begin{array}{|l} \textit{pre.Modify\_Material} \\ c? : \mathbb{P}\, \textit{Comments} \\ m : \textit{Material} \\ \hline \textit{exhausted}(c?) \\ \textit{finish\_trial}(m) \\ \hline \end{array} \quad \ldots\ldots Q_{pre}$$

The post-condition is as follows.

$$\begin{array}{|l} \textit{post.Modify\_Material} \\ c? : \mathbb{P}\, \textit{Comments} \\ m, m' : \textit{Material} \\ \hline m' = \textit{modify}(m, c?) \\ \hline \end{array} \quad \ldots\ldots Q_{post}$$

Because the pre-condition "Modify Material" has not been satisfied until finishing the activity "Give Comments", which is not described here but can be easily imagined, we may have a requirements specification as shown in Figure2.

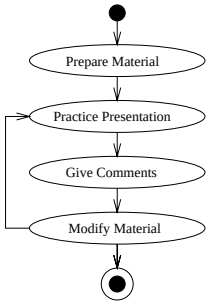

Figure 2. Current Requirements

Step2: Suppose you can use presentation software with which you can modify the materials during the discussion on the fly. e.g. Microsoft's PowerPoint, you can modify your material without waiting for the end of the comments. This can be formally described as follows.

$$\begin{array}{|l} \textit{pre.Modify\_Material}^{new} \\ m : \textit{Material} \\ \hline \textit{finish\_trial}(m) \\ \hline \end{array} \quad \ldots\ldots S_{pre}$$

That is to say, the pre-condition "Modify Material" is weakened according to the new presentation tool. Note that the post-condition is not changed.

In summary, the replacement of this component is classified as

$$\textit{match}_{pre/post}(S, Q) = (Q_{pre} \Rightarrow S_{pre}) \wedge (S_{post} \Leftrightarrow Q_{post}).$$

Step3: The former part of $\textit{match}_{pre/post}(S, Q)$ tells that responsibilities of precedent components may be relaxed because the pre-condition of the replaced component is weakened. According to rule1 in Section2, the activity "Modify Material" does not have to wait for the end of "Give Comments", but it has to wait until "Practice Presentation" finishes because the condition "$\textit{finish\_trial}(m)$" still remains in its pre-condition. Figure 3 shows one possible evolution. This illustrates that the new presentation tool can change the way of doing the task.

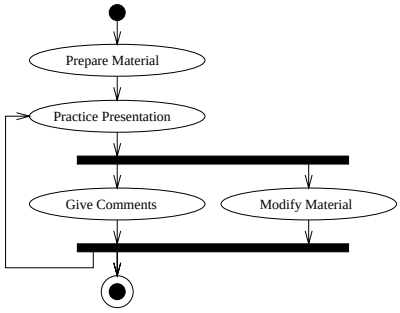

Figure 3. Evolved Requirements

3.2. Assigning Reviewers of a Conference

If you become a program chair of APSEC'99, you should organize the committee of the conference from all over the world, call for papers and assign the reviewers of each submitted paper. In this example, we show a requirements specifications for supporting the task of program chair so as to discuss a replacement which seems to degenerate this task.

Step1: Suppose the following conditions are satisfied:

- The committee members have suitable ways to share and to read the contents of all submitted papers, though the members of the committee are geographically distributed all over the world. For example, the papers are submitted in electronic format, e.g., PostScript or PDF, to a multi-casting address.
- They can have a meeting easily even if the time differences among the locations of each member is large. For example, they have an asynchronous communication tool like email.

Under such conditions, you can appoint new members of the committee until the submission deadline, if no suitable researcher is a member for a submitted paper. Reviewers of each paper can be assigned simultaneously.

This fact can be formally described as a pre-condition of "Assign Reviewers" as follows:

$$\begin{array}{|l} \textit{pre.Assign\_Reviewers} \\ \hline \text{true} \\ \hline \end{array} \quad \ldots\ldots Q_{pre}$$

This means that the client of "Assign Previewers" takes no responsibility. The effect of this activity can be described as follows:

$$\begin{array}{|l} \textit{post.Assign\_Reviewers} \\ \hline p' : \mathbb{P}\, \textit{Paper} \\ c' : \mathbb{P}\, \textit{Committee} \\ m' : \textit{Paper} \times \textit{Committee} \\ \hline m' = \textit{assign}(p', c') \\ \hline \end{array} \quad \ldots\ldots Q_{post}$$

As a result, you may have a requirements specification as shown in Figure4.

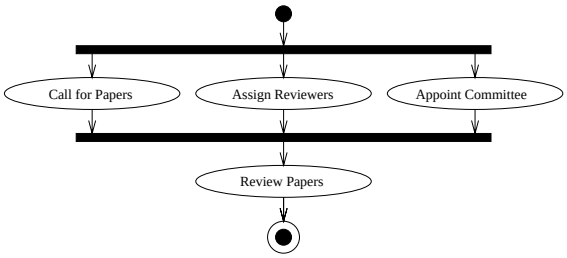

Figure 4. Current Requirements

Step2: Unfortunately, if the members of the committee cannot share the submitted papers easily and quickly, e.g. papers may be submitted to one or a few members of the committee as printed matter, you cannot assign the reviewers simultaneously.

This can be described formally as follows:

$$\begin{array}{|l} \textit{pre.Assign\_Reviewers}^{new} \\ \hline p : \mathbb{P}\, \textit{Paper} \\ \hline \textit{fixed}(p) \\ \hline \end{array} \quad \ldots\ldots S_{pre}$$

The post-condition is not changed.

In summary, the replacement of this component is classified as

$$\textit{match}_{pre/post}(S, Q) = (Q_{pre} \Leftarrow S_{pre}) \wedge (S_{post} \Leftrightarrow Q_{post}).$$

Step3: The former part of $\textit{match}_{pre/post}(S, Q)$ tells that the responsibilities of precedent components should be strengthened, because the pre-condition of replacing component, i.e., Q_{pre} is strengthened. Therefore we should modify the requirements according to rule2 in Section2 so that the new pre-condition of "Assign Reviewers" has been satisfied.

Figure5 shows one possible evolution where the component "Assign Reviewers" is preceded by the component "Call for Papers".

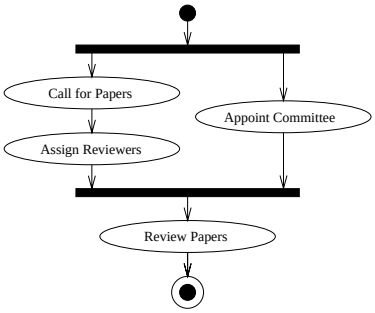

Figure 5. Evolved Requirements

4. Discussion

In this paper, we present a method for conducting requirements evolution according to the replacement of current components. However, the following issues still remain.

1. How to decompose each component so as to apply pre/post match.
2. How to handle the components which can not be decomposed nor pre/post match be applied.
3. Where and how to replace the other components of the current requirements so as to adapt the new component.
4. How to select the most suitable replacement from the possible ones.

We will explore the solution of the first three issues by using case studies. The fourth issue will be handled by Design Rationale Systems [5], with which we can compare and evaluate the alternatives of a design.

Acknowledgments

The authors would like to thank the members of Requirements Engineering Working Group, SIGSE of IPSJ.

References

[1] M. J. Muller, S. Kuhn, D. M. Wildman, and E. A. White. Participatory Design: Introduction. *Commun. ACM*, 36(4):24–28, Jun. 1993.

[2] M. A. Maiden and C. Ncube. Acquiring COTS Software Selection Requirements. *IEEE Software*, 15(2):46–56, Mar. and Apr. 1998.

[3] A. M. Zaremski and J. M. Wing. Specification Matching of Software Components. *ACM Trans. Software Eng. and Methodology*, 6(4):333–369, Oct. 1997.

[4] B. Meyer. *Object-oriented software construction, 2nd edition*. Prentice Hall, 1997.

[5] J. Lee. Design Rationale Systems: Understanding the Issues. *IEEE Expert/Intelligent Systems & Their Applications*, 12(3), May-Jun. 1997.