A Conceptual Model of Version Control in Method Engineering Environment

Motoshi Saeki Takafumi Oda

Dept. of Computer Science, Tokyo Institute of Technology
Ookayama 2-12-1-W8-83, Meguro-ku, Tokyo 152, Japan
Tel. +81-3-5734-2192 Fax +81-3-5734-2917 E-mail saeki@se.cs.titech.ac.jp

Abstract. Unlike line-based version control of source codes, version control of model descriptions should be based on logical components such as Class and Association in a class diagram. In addition, if an adopted method is changed to a new version during a development, the product being currently developed should be consistent to the new method. We proposed three-dimensional conceptual model to clarify version control in method engineering context.

1 Introduction

Method Engineering is the discipline for exploring techniques to build project-specific methods for information system development, called situational methods [1]. Computer Aided Method Engineering (CAME) is a kind of computerized tool for supporting the processes to build them [3, 5].

Although we can have a powerful situational method, another difficulty originating from frequent changes of a product still remains. A product is frequently changed by various reasons, e.g. customer's requirements changes, even during its development. Developers should have various versions of a product and manage them in their project. In this situation, the techniques for version control are significant to support their tasks by using computerized tools. Many version control techniques and supporting tools for source codes such as CVS [6] have been studied. These tools store the current version of a product and the differences between the adjacent versions in a repository, so that it can recover all of the older versions by applying the stored differences to the current one (backward difference). However, they are for text documents and line based management, i.e. the granularity of version control is a "line" and the difference is generated line by line. Since we use diagram documents such as class diagrams, we should manage the changes on the diagrams, not in the granularity of a line, but of a logical component, e.g. "Class", "Association", "Attribute" etc. in the case of Class Diagram. The targets of version control should be logical components and they depend on methods.

Another issue on version control in method engineering environment is the support for version control of methods themselves. The point that we have to consider is that the adopted method can be changed to its higher version during a development project. If the product that was developed following the older

Proceedings of the CAiSE'05 Forum - O. Belo, J. Eder, J. Falcão e Cunha, O. Pastor (Eds.)

method becomes inconsistent to the new method, some supports of the change of the product so as to make it consistent to the new method, including the detection of inconsistency, is necessary. In [4], the changes of methods were classified into a set of patterns, but it did not mention any support for the version control of methods themselves.

This paper proposes a conceptual model for version control in method engineering environment in order to clarify and solve above issues.

2 Conceptual Model of Version Control

We have a following simple scenario of a development as an example, which will be used throughout this paper. A method engineer constructed a new method assembling Class Diagram and Sequence Diagram of UML. Figure 1 illustrates a part of Lift Control System developed following this method. Each diagram is a unit of configuration management, i.e. configuration item.

2.1 Product Version v.s. Configuration Item

Firstly, consider what the target of version control is. A engineer completes the diagram shown in the left part of Figure 1, and commits it as the version 0. After that, he adds the object "Door" to the sequence diagram as shown in the right part of Figure 1, and commits it as the version 1. Note that the changes of layout information on a diagram such as moving the positions of graphical components and resizing them are not essential. Logical information such as "adding a Door object" should be held as the target of version control. What the targets of version control should be made depends on an adopted method. The targets should be method elements, in this example, Class, Association, Object, Message and so on. Thus we have to extract method elements from a method description of the adopted method, i.e., a meta model.

By using method assembly technique [2], a method is obtained as a composition of more primitive methods (method fragments). In fact, our example method consists of Class Diagram and Sequence Diagram. Thus a product includes a class diagram and sequence diagrams, each of which can be the target of version control, i.e., a configuration item (see Figure 1). In other words, we have to control both versions of the class diagram and of the sequence diagrams individually and independently. However, to keep consistency between the class diagram and the sequence ones, we control their versions together, not separately. Each configuration item in a version has a common version number. When a configuration item is changed to a new version, the same new version number is attached to the other items that the product comprises, even if they are not changed. As shown in Figure 2, we assume that all configuration items are put on a version plane. The addition of Door object and Open message is the difference between the two version planes. This figure includes two axes: product version axis and configuration item axis.

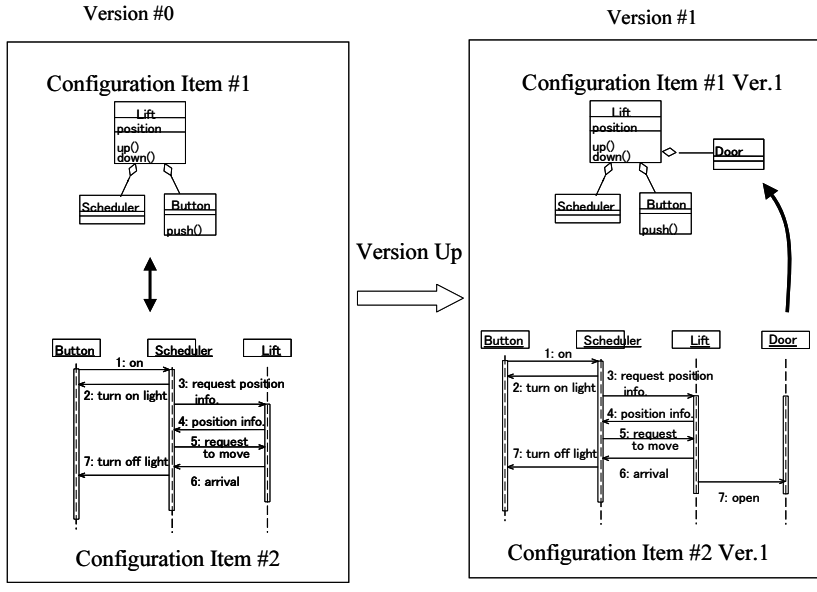

Fig. 1. Lift Control System

2.2 Product Version v.s. Method Version

The second point is on the changes of the adopted methods. As shown in Figure 3, the engineer finds that Lift Control System has real-time property, and extends the current method so that he can model timing constraints in sequence diagrams. He modifies the meta model of Sequence Diagram (M0 : version 0 of the method) by adding the method concept "Timing Constraint", and gets a new version 1 (M1). The version control of meta models is called "method version control" to distinguish it from usual version control of products (called "product version control"). Now, the engineer continues his activities following the new method M1. The change of the adopted method to M1 does not impact on the current product (version 1 in Figure 1), or does not violate consistency between the method and the product. When a method is changed, we consider that the current product is evolved to a new version, i.e., version-up, even if it is not really changed.

Assume a subsequent scenario that will occur inconsistency. As shown in the upper part of Figure 3, the engineer adds a timing constraint "b-a< 2 min.". The engineer returns back the method to the older one M0 after this addition. Since M0 does not include "Timing Constraint", "b-a<2min." in the current product causes inconsistency. The version control system should detect inconsistency when the currently adopted method is changed. If the version-up of the method causes any inconsistency on the product, the engineer changes the product so

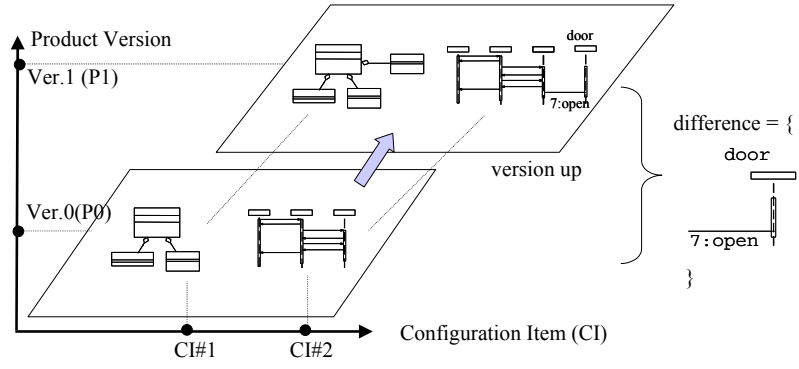

Fig. 2. Version Up of A Product

that it is consistent to the new method, or he stops the version-up of the method.

Since method version control is independent on product version control, as shown in Figure 4, there exists two axes: product version axis and method version axis in our conceptual model. After completing the version 1 (where Door object has been just added) following the method M0, the engineer evolves the current method into a higher version M1. The method M1 is the version where timing constraints of the system to be developed can be specified, as shown in Figure 3. Since the current version of the product P1 is consistent to the method M1, the engineer does not need to change it, so difference={} (an empty set) is obtained shown in Figure 4. However the version of P1 goes up to the version 2 (P2), because we should distinguish the current product based on M1 from M0's. After that, the engineer adds the timing constraint "b-a<2 min." to P2 and gets the version 3 (P3). Then he returns the method back to M0 after completing P3 by some reasons. Since M0 does not have the method element "Timing Constraints", he should delete "b-a<2 min." from P3, Finally, the current method becomes M0 and he gets the version 4 (P4). As shown in Figure 4, we can project product version and method one to the corresponding axes respectively, and it is helpful to understand them.

Consistency checking can be automatically done by examining whether the current version of a product includes the components belonging to a deleted method element. Deleted method elements can be identified from the differences of method changes. In the example, the differences from M1 to M0 includes the deletion of "Timing Constraints" and P3 includes the instance "b-a<2 min." of "Timing Constraints". The engineer is guided to pay attention to it.

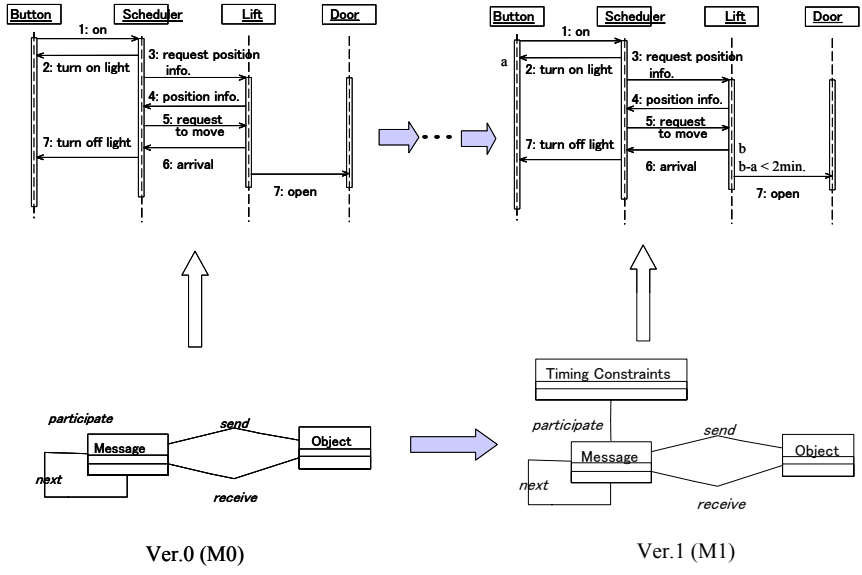

Fig. 3. Changing a Method

2.3 Three Dimensional Model

Summarizing the discussions above, we can capture a conceptual model of version control with a three dimensional space, as shown in Figure 5. Each lattice point represents a version of a product to be managed.

3 Conclusion and Future Work

This paper discusses the problems of version control in method engineering environments, and proposed a conceptual model to solve them.

Based on the conceptual model, we are implementing a version control system wherein backward difference are represented using XMI technique [7] will embed it into our CAME tool. As a result, we can generate diagram editors having version control mechanism that deal with diagrams in granularity level of their logical components. As for version control of meta models, we do not consider propagation of version change of method fragments. If a method engineer changes a method fragment MF to a new version, what happens to the method M that is composed of MF? Should we automatically change the method M to a new method that uses the new version of MF? This is also one of future work.

References

1. S. Brinkkemper. Method Engineering : Engineering of Information Systems Devel-

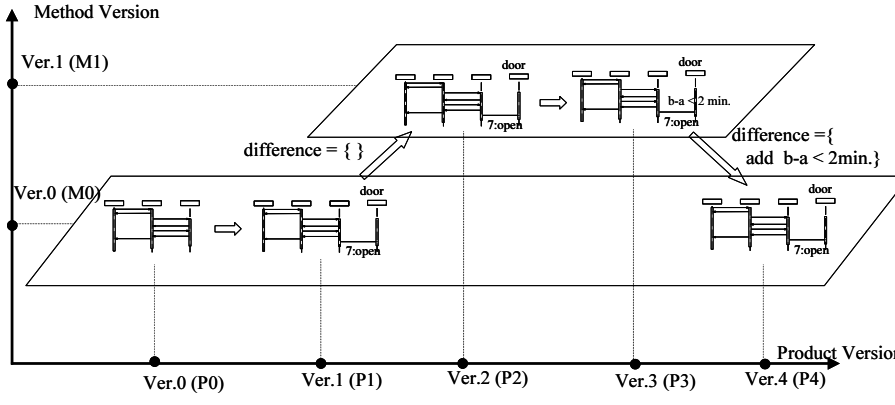

Fig. 4. Version Up of A Method

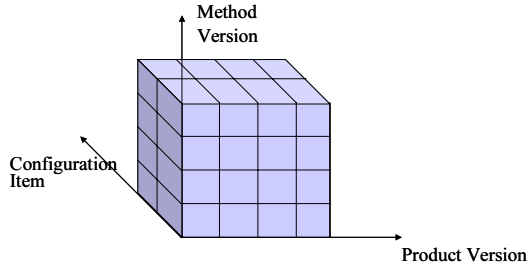

Fig. 5. Three Dimensional Model

opment Methods and Tools. *Information and Software Technology*, 37(11), 1995.
2. S. Brinkkemper, M. Saeki, and F. Harmsen. Meta-Modelling Based Assembly Techniques for Situational Method Engineering. *Information Systems*, 24(3):209–228, 1999.
3. F. Harmsen. *Situational Method Engineering*. Moret Ernst & Young Management Consultants, 1997.
4. J. Ralyte, C. Rolland, and R. Deneckere. Towards a Meta-tool for Change-Centric Method Engineering: A Typology of Generic Operators. In *CAiSE 2004*, pages 202–218, 2004.
5. M. Saeki. Toward Automated Method Engineering: Supporting Method Assembly in CAME. In *Engineering Methods to Support Information Systems Evolution - EMSISE'03, OOIS'03* http://cui.unige.ch/db-research/EMSISE03/, 2003.
6. Concurrent Versions System, http://www.cvshome.org/.
7. XML Metadata Interchange, http://www.omg.org/.