Software Evolution Support Using Traceability Link between UML diagrams

Hidekazu Omote, Kohta Sasaki, Haruhiko Kaiya, Kenji Kaijiri
Faculty of Engineering, Shinshu University
4-14-1 Wakasato Nagano 380-8553 japan
{comote,ckota,kaiya,kaijiri}@cs.shinshu-u.ac.jp

Abstract. Evolution is indispensable process in software development, so the systematic treatment of evolution is very important. UML becomes a de facto standard for notation in object oriented development, but the relations between each models are not defined clearly in UML, and further a usecase model is not defined in UML. In this paper, in order to support the evolution in object oriented development, we defined a usecase model, an object model, and traceability links between these models. Using these models and links, we proposed a new evolution method. Evolution in design patterns are also considered.

1 Introduction

Evolution is indispensable process in software development, so the systematic treatment of evolution is very important. UML becomes a de facto standard for notation in object oriented development, but the relations between each models are not defined clearly in UML, and further a usecase model is not defined in UML, so the evolution in upper models cannot be easily transfered to lower models, even if these models are properly maintained. In this paper, in order to support the evolution in object oriented development, we defined a usecase model, an object model, and traceability links between these models. Using these models and links, we proposed a new evolution method. Evolution in design patterns is also considered.

There are several kinds of evolution[15, 18], so we define **evolution** as “changes in use case model”. In this paper, use case models consist of the following elements:

- use case diagrams
- activity diagrams which describe scenario for each use case

and object models consist of the following elements:

- class diagrams
- sequence diagrams

We also define **traceability link** as “the relation between the elements of use case models and those of object models”. We consider the use case model as an initial model, propagate the evolution using traceability, and do the required change in object model. We supposed that the traceability link between each models are established during development process. We maintain this link using proposed method. The advantages of our method are as follows:

1. The traceability link is maintained throughout the development process, because we consider the maintenance of traceability link as one of the fundamental development processes.

2. Software evolution can be propagated systematically. We can traverse each model elements using these links and effected elements can be identified easily.

In section 2, we described about software evolution, design pattern, and traceability link, related researches in section 3, evolution using traceability link in UML in section 4, and traceability link in design pattern in section 5. We showed an example in section 6, and concluded in section 7.

2 Software Evolution, Design Pattern, and Traceability Link

2.1 Evolution

We show our image of evolution in Fig.1. We consider two models: use case mode (UM) and object model (OM). We suppose that the traceability link between UM and OM is established during development.

After the first stage of development, some evolution will occur. We suppose that the traceability link was already established. We need to reconstruct the overall software according to this evolution. We must do the following steps:

1. To reflect the requirement change into UM

2. To identify the model elements in OM which must be changed using traceability link

3. To reflect required change in OM

4. To reestablish the traceability link

In order to realize the above steps, the followings become needed:

- development process integrated with traceability
  Traditional development process does not consider the traceability explicitly during development, so traceability link must be established afterward, but in this case some informations are missed, so the establishment of precise traceability link becomes difficult. On the other hand, traceability link is easily established during development process because the link is IN/OUT relation between model elements in our method.

- traceability link between each model elements
  We need to define each model and traceability link between several elements for each model. We need to consider fine grained traceability link.

2.2 Use Case Model and Object Model

There are several techniques about modeling use case and transformation between use case and object model. We surveyed these researches in section 3. But these researches are not adequate for our purpose from the granularity. We defined the use case model using use

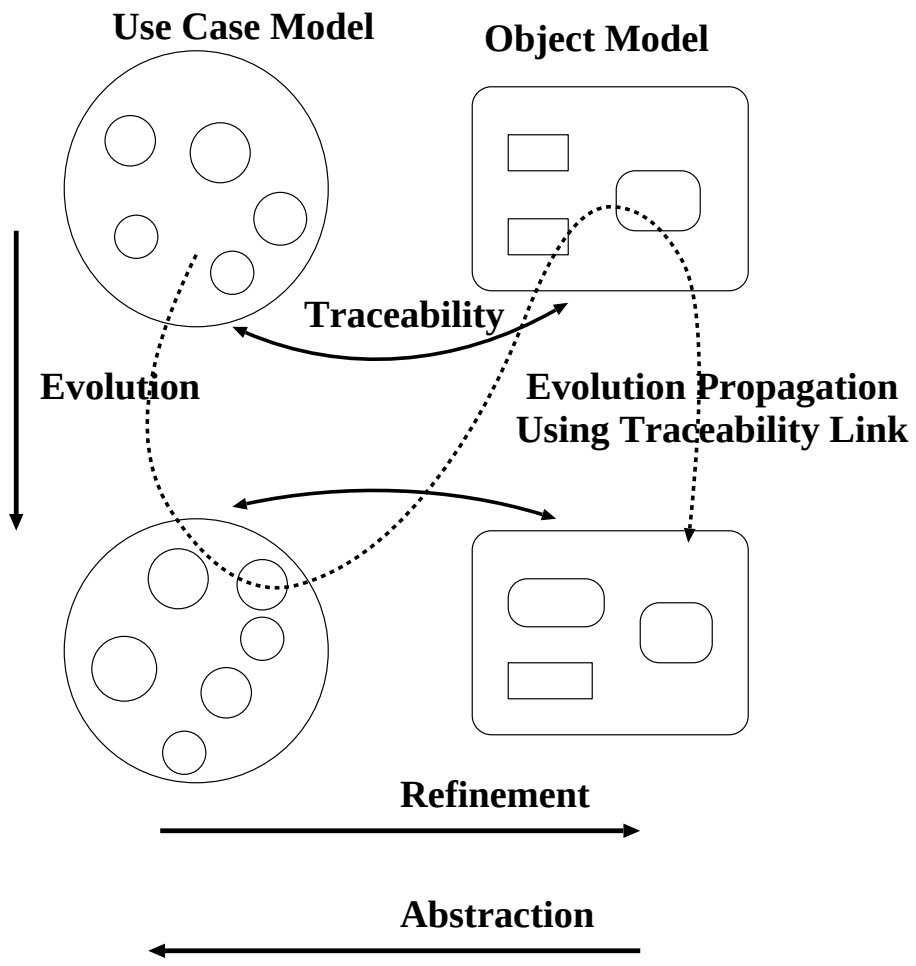

Figure 1: Use Case Evolution

case diagrams and activity diagrams, and describes scenarios using activity diagrams. We restrict the statements in activity diagrams as the following three forms in order to make the transformation from activity diagrams to sequence diagrams easy:

- *An actor action* which describes an action from an actor to a system
  actor *does something* to **system**
- *A system action* which describes an action from a system to an actor
  system *does something* to **actor**
- *A system inner action*
  system *does something*

We use two kind of a sequence diagram to describe dynamic behavior: **a system sequence diagram** which does not consider inner objects and **a detailed sequence diagram** which considers inner objects. A system inner action will be refined in a detailed sequence diagram.

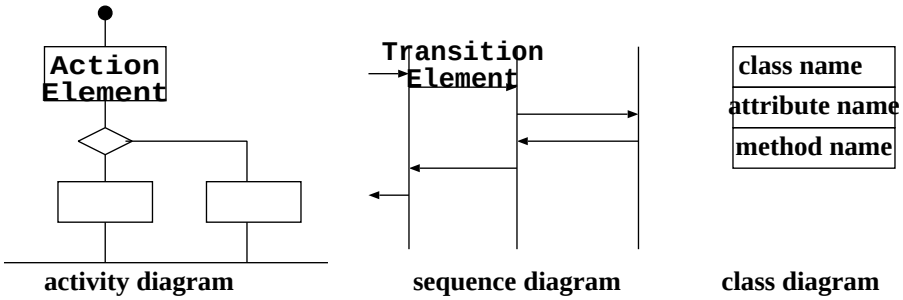

Figure 2: Model Elements

2.3 Development process

We suppose the following development process which refers the elements in Fig.2. We use a stereotype notation for traceability link.

1. Problem description
2. Use case diagram description
3. Scenario description using activity diagram
   Make link between use cases and activity diagrams using ≪refine≫ link
4. Class diagram description
   Make clear the responsibility and make link between action elements and classes using ≪responsibility≫ link
5. System sequence diagram description using activity diagrams
   Make link between action elements and transition elements using ≪action≫ link
6. Detailed sequence diagram description using system sequence diagrams
   Make link between transition elements in system sequence diagrams and those in detailed sequence diagrams using ≪action_refine≫ link. If some transition is delegated to other transition, then make link using ≪delegate≫ link
7. Class diagram refinement using detailed sequence diagrams
   Make link between transition elements in detailed sequence diagrams and method names in class diagrams using ≪method≫ link
8. Detailed sequence diagram refinement
   Make link between transition elements in detailed sequence diagrams and class names in class diagrams using ≪class≫ link
9. Implementation
   Make link between transition elements and methods using ≪implement≫ link.

This process may be too simple, so more elaborated process must be needed for real applications.

3 Related Researches

There are several researches about the structuring and formalizing use cases. A.Cockburn[2] introduced a goal concept and proposed structuring method for use cases using this concept. There are also several researches about the traceability between UML models. Glinz[11] proposed the traceability between scenario and class diagrams. He introduced the class link to scenario and the scenario link to class. Egyed[10, 13] established the traceability between scenario and system using dynamic profiles, which can be obtained using the scenario based on test data. Rausch[14] proposed the test method for the total integrity when the external requirement of some component is changed. Antoniol, et al[6, 9] proposed the traceability between two versions of the same OO software. In their method, high level design models for each version is compared, so coarse granularity is used. Koesters, et al[4, 16] proposed the method which maps the class model concept into a use case model, and implemented a support tool. For this purpose, they defined a formal use case model. Mittermeir, et al[7] proposed the reflection method of the evolution in specification level into the evolution in implementation. R.France, et al[17] proposed the software evolution method based on several stakeholders' multiple views.

The above researches are categorized into the following two categories:

- To formalize use case and introduce the concept of class diagrams into use cases (Koesters, et al[4])
- To introduce concepts in one model into the another model and vice versa (Glinz, et al[11]).

In our method, traceability will be established easily during the development process, and evolution propagation can be considered in fine granularity level.

4 Evolution Using Traceability Link in UML

In this section, we consider the evolution propagation using the traceability link between UML diagrams, which does not consider design patterns. In the next section, we propose the traceability link between UML diagrams which use design patterns. We propose the following steps:

1. From the changed problem description, we identify the use case that will be affected.
2. We identify action elements that will be affected using ≪refine≫ link.
3. From the ≪action≫ link between activity diagrams and system sequence diagrams, we identify the transition element in the system sequence diagram and modify it.
4. From the ≪action_refine≫ link between system sequence diagrams and detailed sequence diagrams, we identify the transition element in the detailed sequence diagram and modify it. If a ≪delegate≫ link is defined in this transition, the linked transition need be modified.
5. From the ≪implement≫ link between detailed sequence diagrams and codes, we identify the code segment which needs to be modified and modify it.

Traceability will be reestablished during this process. There may be another evolution which cannot be processed using this method, but if evolution can be processed using this method, semi-automatic evolution propagation becomes possible.

5 Traceability Link in Design Pattern

Design patterns (DPs) have several additional knowledges which characterize each pattern[1], and these knowledges diffused in several diagrams, so the links connecting these knowledges are useful. We considered three knowledges, and links between these knowledges and concrete UML models as shown in Fig.3: Structure, Participants, and Collaboration. We consider the virtual model about DP which describes recurring design knowledges, and make traceability links between that model and concrete models (UM and OM). France, et al[19] proposed UML based design pattern specification techniques. They used a role model in order to characterize design patterns and to describe these design knowledges. According to their terms, Structure corresponds to Structural Pattern Solutions, Participants corresponds to Role Name, and Collaboration corresponds to Feature Role.

DPs have specific evolution patterns and for these patterns, regulated evolution becomes possible, so if the required modification matches with the specific modification in a DP, the predetermined modification of that DP can be applied automatically using traceability link. If the required modification upsets the required condition of some DP, refactoring becomes needed for that DP realization, but the part to be refactored can be identified easily using these links.

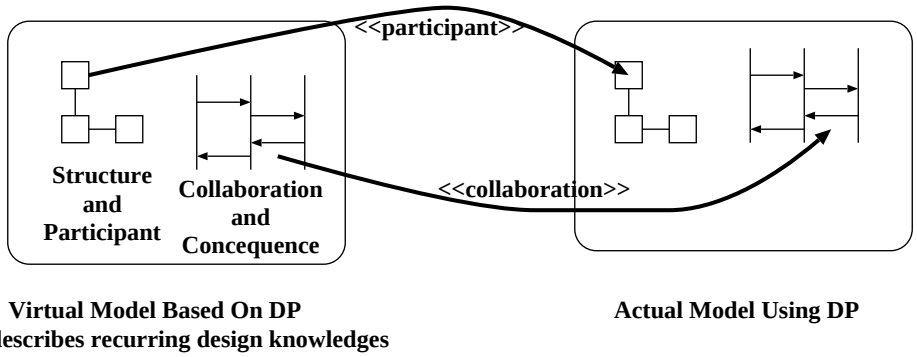

Figure 3: Traceability Link in Design Pattern

6 Examples and evaluations

In this section, we show the concrete example of traceability link maintenance. We suppose simple library system, and the problem description is as follows:

Users have ID cards. When they borrow a book, they indicate their ID card and the book. When they return the book, they only present the book. Users can borrow three books simultaneously, and can borrow within two weeks. If users already borrowed three books, or they borrowed over two weeks, they can not borrow further. There are two kinds of books in library: borrowable and unborrowable. In the library counter, the book to be borrowed will

be checked and unborrowable books can not be borrowed. Once a day, the system checks for each user the period of borrowing and urges users who borrowed over two weeks to return as soon as possible.

We show the use case diagram for this problem in Fig.4(a). Next we describe the scenario for "Borrow a Book" using a activity diagram as shown in Fig.4(b). According to the activity diagram in Fig.4(b) we construct a system sequence diagram as shown in Fig.4(c). We refine the system sequence diagram in Fig.4(c) into the detailed sequence diagram as shown in Fig.4(d). We showed the part of traceability links in Fig.4 using a thick line with arrows in both side.

We consider the following evolution; addition of reservation function for the borrowable book. We can propagate the evolution through traceability link as follows(description within a parenthesis shows the identified element):

1. Identify the use case (borrow a book)
2. Identify the action element (system checks whether the book is borrowable)
3. Identify the transition element in system sequence diagrams (system checks whether the book is borrowable)
4. Identify the transition element in detailed sequence diagrams (Clerk asks the Library whether the book is borrowable)
5. Identified element is related with the transition element (Library asks the Book whether it is borrowable) with delegate relation, so only the latter element needs to be modified.

   As a result, traceability links are reestablished.

7 Conclusions

We proposed the systematic method to process software evolution using the traceability link between UML models. It is only an idea, so it is necessary to further consider the following problems:

- Refine the development methodology which establishes the traceability link. In ordinary methodologies, several diagrams are developed from different viewpoint and concurrently, so establishment of traceability link is not so easy even if we consider it during development.
- Add the knowledge about evolution within design pattern catalogs. Design pattern catalogs do not include such informations. We need to write scenario templates and their evolution patterns for each design pattern.
- Refine traceability link and formalize it. Our traceability link is too simple and we need to elaborate the kind of link and to formalize it in order to realize a support tool
- Realize a support tool
- Do experiments and validate the effectiveness

References

[1] Erich Gamma, et al: *Design Patterns*, Readings, Addison-Wesley (1994)

[2] A. Cockburn: *Structuring Use Case with Goals*, html version (1995)

[3] M. Sefika, et al: *Monitoring Compliance of a Software System with Its High-Level Design Modes*, 18th ICSE (1996)

[4] G. Koesters, et al: *Coupling Use Cases and Class Models*, BCS FACS EROS workshop (1997)

[5] G. Koesters, et al: *Animated Requirements walkthroughs based on Business Scenarios* (1997)

[6] G. Antoniol, et al: *maintaining Traceability During Object-Oriented Software Evolution: a Case Study*, ICSM (1998)

[7] R.T.Mittermeir, et al: *Object Evolution by Model Evolution*, ICSM (1998)

[8] karin K. Breitman, et al: *A Framework for Scenario Evolution*, ICRE 98 (1998)

[9] G. Antoniol, et: *Evolving Object Oriented Design to Improve Code Traceability*, 7th IWPC (1999)

[10] Alexander Egyed, et al: *Automatically Detecting Mismatches during Component-Based and Model-Based Development*, ASE(1999)

[11] Martin Glinz: *A Lightweight Approach to Consistency of Scenarios and Class Models*, ICRE 2000

[12] Karin Breitman, et al: *Scenario Evolution: a Closer View on Relationships*, ICRE'2000

[13] Alexander Egyed: *A Scenario-Driven Approach to Traceability*, ICSE2000

[14] Andreas Rausch: *A Proposal for Supporting Software Evolution in Componentware*, CSMR (2000)

[15] T. Mens, et al: *Automating Support for Software Evolution in UML*, Automated Software Engineering, 7, 1 (2000)

[16] G. Koesters, et al: *Coupling Use cases and Class Models as a Means for validation and verification of requirements Specification*, RE Vol.6 (2001)

[17] R. France, et al: *Multi-View Software Evolution: A UML-based Framework for Evolving Object-Oriented Software*, ICSM (2001)

[18] Antje von Knethen: *A Trace Model for System Requirements Changes on Embedded Systems*, IWPSE (2001)

[19] Robert B. France, et al: *A UML-Based Pattern Specification Technique*, IEEE Trans. on Soft. Eng., 30, 3 (2004)

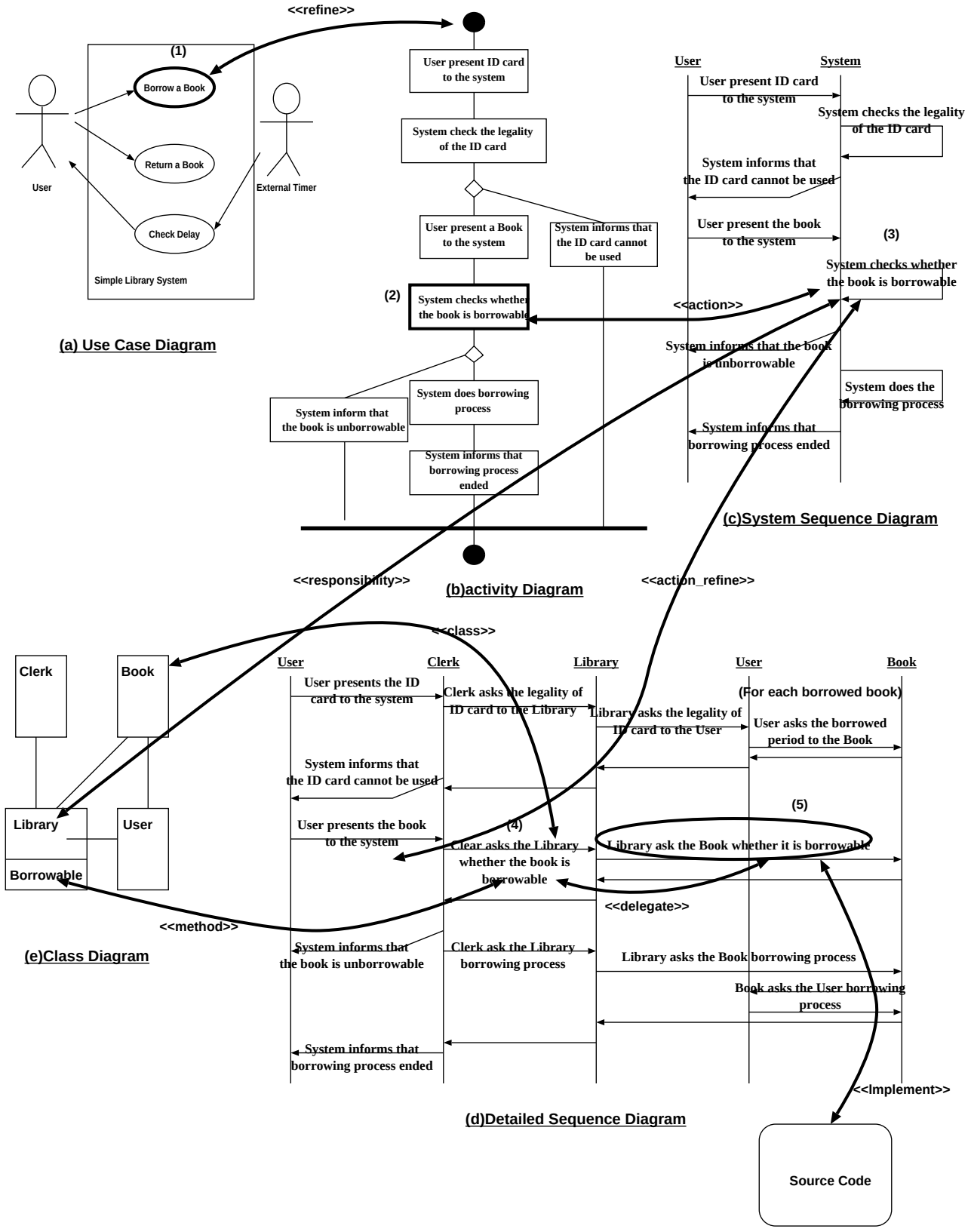

Figure 4: Traceability Link