An approach to verifying consistency of software refactoring process

Thi-Huong Dao, Hong-Anh Le, Ninh-Thuan Truong
College of Technology, Vietnam National University, Hanoi

Contact Information:
Information Technology
College of Technology, Vietnam National University, Hanoi
144 Xuan Thuy, Cau Giay, Hanoi, Vietnam

Email: huongdt.di12@vnu.edu.vn

Abstract

The combination of refactoring and design patterns has established revolution approach in software re-engineering. However, it raises the problem of inconsistency between original model and its evolution. This paper proposes an approach to dealing with those aspects, especially preserving behavior of the system. To this aim, we employ Unified Modeling Language to modeling system, and Object Constraint Language to represent the system requirement. In additional, we construct a set of consistent rule which uses to checking whether system can preserve its behavior or not. Furthermore, we illustrate the checking approach in a case study of Adaptive Road Traffic Control System.

Introduction

In software engineering, software evolution is the process of developing software initially, then repeatedly updating it. Techniques used in this process are commonly re-engineering and/or refactoring of software code or models.

Refactoring is a powerful technique, it is used to improve the quality of the software by changing the internal structure of software without altering its external behavioral properties, but it needs to be performed carefully. The main danger is that errors can inadvertently be introduced, especially when refactoring is done manually.

A design pattern [1] is a general reusable solution to a commonly occurring problem within a given context in software design.

Any evolution software system should be optimized by discovering opportunities to use design patterns to improve its source code or design model.

However, a big problem with design patterns in refactoring process is that we can not assure the consistency between the existing design model and its evolution.

Main Objectives

1. Constructing a set of consistent rules.
2. Checking static and dynamic behavioral consistency between initial model and it evolution.

Approach to checking consistency

Framework Overview

There are three constituent processes which have been identified in the framework overview, indicates in Fig. 2, mainly (1) modeling, (2) refactoring, and (3) checking consistency process.

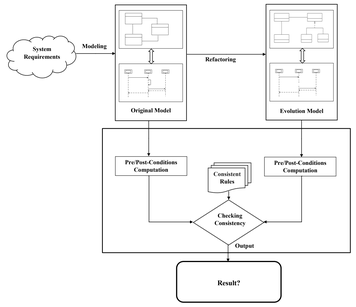

Figure 1: The framework of checking consistency

Formal representation of a UML model

- Clarify some concepts: Invariant, Precondition, Postcondition.
- Define new concepts: Scenario precondition, Scenario postcondition.
- Formalize these concepts: Model, Class, Abstract operation precondition, Abstract operation postcondition, Scenario, Scenario operation, Scenario precondition, Scenario postcondition, Refactor.
- Propose a set of consistent rule: Static preservation, Total dynamic consistency, Partial dynamic consistency, Model inconsistency.

Results

Applying the approach on a case study: Adaptive Road Traffic Control (ARTC) System which use to optimize the traffic congestion in towns and cities. The model before and after refactoring as follows:

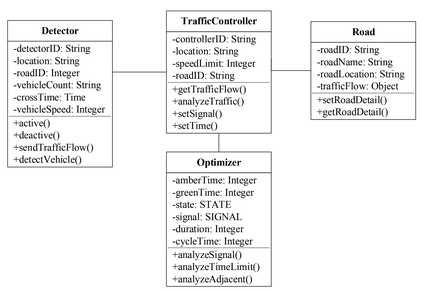

Figure 2: Initial class diagram of ARTC system

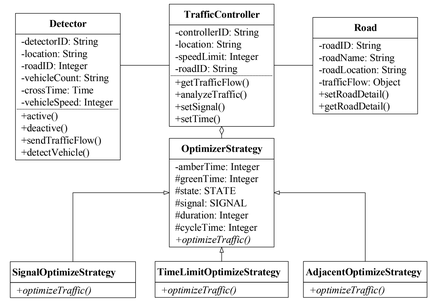

Figure 3: Class diagram of ARTC system after applying Strategy pattern

From the values of preconditions and postconditions, we conclude that the scenario is strongly preserved all the attributes of the model.

Conclusions

- We have proposed an approach to checking the consistency in refactoring process which is performed by design patterns in software models.
- We have also proposed consistent rules to verify if the specification of the initial model is consistency with the new one (after applied design patterns) in evolution software process.
- To demonstrate the approach, we have constructed a model of an ARTC system in UML. In the case study, we just illustrated only the consistency verification when applying Strategy pattern in the only a pair of scenario, respectively in the both models, others scenarios may be done in a similar way for the more complex system. The last result of checking consistency external behaviors between models depends on the conjunction of all the checking result in every pair of scenario in these models. If just only exist a proven inconsistent, as the sequence, the checking consistency is considered to inconsistency.
- As illustrated in the case study, we can see that the formalization process and the process of verifying the consistency are time-consuming and error-prone if we check by manually.

Forthcoming Research

- For the future works, we will adopt tools to help formalize and verify automatically the model evolution process.

References

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[2] J. Kerievsky, *Refactoring to Patterns*. Pearson Higher Education, 2004.

[3] C. Zhao, J. Kong, and K. Zhang, "Design pattern evolution and verification using graph transformation," in *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pp. 290a–290a, Jan 2007.

[4] P. Bottoni, F. Parisi-Presicce, and G. Taentzer, "Coordinated distributed diagram transformation for software evolution," *Electronic Notes in Theoretical Computer Science*, vol. 72, no. 4, pp. 59 – 70, 2003. Workshop on Software Evolution Through Transformations - Toward Uniform Support Throughout the Software Life-Cycle (First International Conference on Graph Transformation).

[5] R. Van Der Straeten, T. Mens, J. Simmonds, and V. Jonckers, "Using description logic to maintain consistency between uml models," in *UML 2003 - The Unified Modeling Language. Modeling Languages and Applications* (P. Stevens, J. Whittle, and G. Booch, eds.), vol. 2863 of *Lecture Notes in Computer Science*, pp. 326–340, Springer Berlin Heidelberg, 2003.

[6] J. Dong, Y. Sheng, and K. Zhang, "A model transformation approach for design pattern evolutions," in *Engineering of Computer Based Systems, 2006. ECBS 2006. 13th Annual IEEE International Symposium and Workshop on*, pp. 10 pp.–92, March 2006.

[7] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[8] A. Angeli, "A pattern language," 1977.

Acknowledgements

This work is partly supported by the research project entitled Verification of equivalence between software models, No. QG.14.07 granted by Vietnam National University, Ha Noi.