A Pattern-based Model Transformation Approach to Enhance Design Quality

Yong-Yi Fanjiang1  Nien-Lin Hsueh2  Jonathan Lee3
1Department of Computer and Information Science, National Taichung University
2Department of Information Engineering and Computer Science, Feng Chia University
3 Software Engineering Institute, National Central University

Abstract

Recently, the impact of design patterns on software quality has attracted a gradual attention since design patterns encapsulate valuable knowledge to resolve design problems and improve design quality. As an attempt towards the investigation of applying goals and design patterns to realize the model transformation, we proposed, in this paper, a goal-driven model transformation by applying design patterns to transform an analysis model into its corresponding design model with an emphasis on the non-functional requirements. The use of goals makes it easier to transform the functional and non-functional requirements into the software models, and derives the candidate design patterns to help satisfy non-functional requirements for resolving the design problems and improving software quality.

Keywords: Design patterns, model transformation, design quality, nonfunctional requirements.

1. Introduction

The software quality has long been recognized as an important topic since the early days of software engineering. In the past, researchers and practitioners alike have examined how systems can meet specific software quality requirements. Therefore, a growing number of practitioners have shown great interests in using design patterns towards high-quality software, since design patterns represent high-level abstractions that reflect the experiences of no other than skilled practitioners themselves. Design patterns have become a popular means to encapsulate object-oriented design knowledge. They capture successful solutions to recurring problems that arise when building software systems.

In this paper, we adopt the goal-driven use case (GDUC) [14] and fuzzy object-oriented modeling (FOOM) [15] served as the formal specifications to capture and specify imprecise requirements, and provide a set of pattern-based transformation rules to deal with the software quality issues (see Fig. 1 for an overview). In Fig. 1, the designer constructs the goals hierarchy based on goals analysis and goals interaction. Before applying the design pattern transformation rules, the designer selects a suitable design pattern from the candidate design patterns through matching the non-functional goals and intents of design patterns, and then transforms the analysis model into design model by applying the transformation rules incrementally. Our proposed approach has the following features:

- Goal-driven use cases and goals hierarchy are built according to the goals and use cases analyzing. The analysis model is constructed based on the goals hierarchy by using the FOOM notations and semantics to specify various alternative models.
- Design patterns serve as a supporting design knowledge base to help the development of a system. A goal-driven method is provided to facilitating the selection of candidate design patterns from the analysis model, and the design pattern transformation rules are used to helping the enhancement of non-functional requirements and improve the software quality.

Figure 1: An overview of our proposed approach
The organization of this paper is as follow. We first briefly describe the background knowledge about model transformation and applying design patterns to model transformation in the next section. The relationship between design patterns and non-functional issues is depicted in section 3, and the section 4 provides the transformation rules. A meeting scheduling system is provided as an example to illustrate our approach in section 5, and some concluding remarks are given in section 6.

2. Background

A number of researches in the area of model transformation and pattern based model transformation have made their marks on our goal-driven pattern-based model transformation method.

2.1. Model transformation approaches

**Graph transformation.** Varró et al [16] describe a system for model transformation based on Graph Transformations. This style of transformation is based on the application of an ordered set of rules. Operators available include transitive closure, and repeated application. Rules identify sub-graphs which define before and after states, and may refer to source and target model elements and introduce associations between.

**Relational transformation.** The basic idea is to state the source and target element type of a relation and specify it using constraints. Gerber et al. [9] explore the application of logic programming to implement transformations.

**Model transformation through high-level programming language.** Implementing a model transformation can be carried out by using a general programming language with a specific API. Indeed, nowadays, repositories such as dMOF [4] can be used to save models and metamodels that are MOF compliant. These tools allow an API to be generated and its basic implementation for each contained metamodels.

**Model transformation based on transformation metamodel.** Engineers build models to better understand the systems that are being developed. In a similar way, to understand existing models we may provide models of these as well. This activity is called meta-modeling. K. Duddy et al. [5] defined a transformation metamodel of which instances will be input parameters of a generator, which transforms a model into a program that implements the transformation process described by this model.

2.2. Pattern-based model transformation approaches

Work in a number of fields has made their marks on the pattern-based approach. These researches are organized into following categories: automatic application of design patterns, quality improvement by design patterns, pattern-based development methodology, and others. The first category of researches [1, 12] focuses on automatic pattern application and ignores the usage of design patterns. The second category of researches [3, 11, 13] focuses on how to restructure a legacy system to be more qualified for the assistance of pattern-based transformation skills. The third category [2, 8] attempts to providing a methodology to serve as a bridge between users’ requirements and design pattern technology. They put efforts on analysis of user needs rather than automation of pattern application.

3. Design patterns and non-functional requirements

Non-functional requirements are not easy to handle because they are subjective (they can be interpreted differently by different people), relative (their importance is depending on the system being considered) and interacting (their achievement may impair or help that of others) [6]. Design patterns [7] provide a possible way to deal with non-functional requirements since they provide solutions to satisfy functional requirements as well as better solutions to meet non-functional requirements. In particular, besides providing a basic, functional solution to a problem, a design pattern offers a qualified, non-functional improvement to that solution. For example, considering the original intent described in Observer design pattern:

```
Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
```

By elaborating the intent, we can understand the design pattern is designed to resolve the communication between a subject objects and its related observer objects. Viewing from the functional aspect, it requires the subject to notify all observers when the subject changes its state. Viewing from the non-functional aspect, it requires the notification should work automatically without knowing types of observers. In other words, Observer design pattern has a FR-intent (functional requirement intent) to address functional problems and an NFR-intent (nonfunctional requirement intent) to improve non-functional quality requirements. We thus transform a design pattern's
intent into functional-intent (FR-intent) and non-functional intent (NFR-intent) to highlight the quality contribution of the pattern.

The structure distributes the object model to achieve the goal described in the intent property. With respective to the FR-intent and NFR-intent, FR-structure and NFR-structure are developed respectively. The major difference of NFR-structure to FR-structure is it applies object techniques to resolve problems. These techniques, such as polymorphism, abstraction, encapsulation, delegation, dynamic binding and inheritance are keys to make object-oriented system more reusable, extensible and maintainable. More details about the relationships between intent and structure of FR and NFR please see [10].

4. Transformation Rule Schema

Each refinement process is based on the application of a design pattern. A transformation is described graphically by a schema called transformation rule schema. A transformation rule schema is parameterized by model elements to be specified by the designer, and is composed of two compartments. The first compartment describes the source model of the design while the second compartment shows its corresponding target model after application of a design pattern. Fig. 2 shows the transformation rule schema that we have defined for the Observer pattern.

After establishing the goals hierarchy for obtaining alternative models and constructing stable kernel and alternatives, the designer must specify the analysis model based on the goals hierarchy in an incremental fashion. Initially a goal in goals hierarchy is chosen that will serve as a start point for the design pattern transformation applying under our proposed approach.

5. Meeting scheduling example

In this section, we use an example - meeting scheduling system to demonstrate the idea of our approach. Goals are identified based on the proposed verb-based classification scheme. In this example, we have identified five goals and formed a goal-driven use case model (see Fig. 3).
the Observer design pattern is recommended and used. Observer pattern is used to resolve the inconsistency problem between a set of objects (said observers) which have a common interest on subject. By requiring the observers to register on the subject before operating, observers can be notified for keeping consistency whenever the subject changes its status.

Fig. 4 represents the final design model after applying the design patterns through our proposed approach.

**6. Conclusion**

In this paper, we propose an approach to providing a goal-driven method to transform the analysis model into design model for dealing with the software quality issues based on design patterns. With the aid of design patterns, in our approach, a goal can be achieved by a well-specified transformation rules, recovered by a proven solution, and enhanced by a known skill.

**Acknowledge**

This work was supported by the Ministry of Economic Affair (Taiwan) under grant 95-EC-17-A-02-S1-029

**References**


