Towards a Pattern-based Model Transformation Approach for Design Quality Enhancement

Yong-Yi Fanjiang\(^1\)  
Nien-Lin Hsueh\(^2\)  
Jonathan Lee\(^3\)

Recently, the growing popularity of model driven frameworks and methodologies, as well as the Model Driven Architecture (MDA) initiated by Object Management Group (OMG) has implied an increasing focus on model transformation. Meanwhile, 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 nonfunctional requirements for resolving the design problems and improving software quality.

**Keywords:** 
Design patterns, model Transformation, goal-driven approach, design quality.

\(^1\) Department of Computer Science and Information Engineering, Fu Jen Catholic University  
\(^2\) Department of Information Engineering and Computer Science, Feng Chia University  
\(^3\) Department of Computer Science and Information Engineering, National Central University
1. Introduction

Recently, the growing popularity of model driven framework and methodologies, as well as the Model Driven Architecture (MDA) [10] initiated by Object Management Group (OMG) has implied an increasing focus on model transformation. Model transformation takes a set of model fragments, which conform to the source metamodel, as input and applies a set of rules to them, resulting in a set of models, which conform to the target metamodel [21]. Meanwhile, 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.

Generally, a transformational approach has two major steps. First, a formal specification is derived from user requirements. Then, a set of transformations is applied to these formal specifications to obtain an implementation system. In this paper, we adopt our previous work on the goal-driven use case (GDUC) [19] and fuzzy object-oriented modeling (FOOM) [20] 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. As an attempt towards the investigation of applying goals, design patterns, and model transformation approach, a goal-driven model transformation by applying design patterns approach has been proposed (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 analyzed. 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 help the enhancement of non-functional requirements and improve the software quality.

Fig. 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 nonfunctional issues is depicted in section 3, and the section 4 provides the transformation rules. A meeting scheduling application 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. Goal-based approach

In recent years, goal-based requirements analysis methods have attracted an increasing attention in the area of requirements engineering, as goals information are valuable in identifying, organizing and justifying software requirements [6, 18, 21, 22, 23]. The tenet of goal-based approaches is focusing on why systems are constructed, which provides the motivation and rationale to justify software requirements. Other benefits include: (1) helping to acquire requirements by elaborating what requirements are needed to support the goals; (2) making easy the justification and explanation of the presence of requirements in a progressive way by starting from system-level and organizational objectives from which such lower level descriptions are progressively derived [6]; and (3) providing the information for detecting and resolving conflicts that arise from multiple viewpoints among agents [6, 18].

Subsequently, a number of researchers have reported progress toward the improvement of goal-based techniques [6, 21, 1]. In particular, Dardenne et al. [6] have advocated a goal-directed approach to models acquisition. Mylopoulos et al. [21] have proposed a framework for representing nonfunctional requirements in terms of goals, which can be evaluated in order to determine the degree to which a nonfunctional requirement is supported by a particular design. Moreover, they advocated that object-oriented modeling approach can then be used to model functional requirements to compensate the goal-oriented approach [22]. Meanwhile, Anton [1] has proposed a goal-based requirement analysis method to identify, elaborate and refine goals for requirements specifications.

To provide a systematic way to handle non-functional requirement, and model the interactions between requirements, our previous work on GDUC [19] proposes an approach to extending use case models with goals to guide the derivation of use cases, and analyzing the interactions among non-functional requirements on the basis of interactions between goals and use cases.

2.2. Goal Driven Use Case (GDUC)

An important advantage of use case driven analysis is that it helps to manage complexity, as it focuses on one specific aspect of usage at a time. Use cases focus the expression of requirements on users, starting from the very simple viewpoint that a system is built first and foremost for its users. The organization of the approach is performed with respect to the interaction of a single category of users at a time; this participating of the set of requirements considerably reduces the complexity of requirements determination. However, the current use case approaches are somewhat limited in supporting a formalization of use cases, and in structuring and managing large use case models.

Our previous work on goal-driven use cases (GDUC) [19] extends use cases with goals to alleviate the above problems:

- goals are used to structure use case models and to derive use cases;
- soft goals are differentiated from the rigid ones to handle non-functional requirements that are imprecise in nature; and
- goals information are embedded in the use cases, and relationships between requirements are analyzed by investigating the interactions between goals and use cases.

After constructing the use case model, the interaction between goals are evaluated and the goals hierarchy is constructed for software modeling. To better characterize the relationships between use cases and goals, we adopted predicates proposed by Mylopoulos et al [21]: satisfied, denied, satisfiable,
Towards a Pattern-based Model Transformation Approach for Design Quality Enhancement

and deniable for the relationships between goals and use cases called the goals interaction in the use cases level. Then, the relationships between goals in the system level are analyzed based on the use case models and relationships between goals interaction in the use case level.

Instead of putting heavy efforts on resolving conflicts at the beginning of system design which may overload analysts, we start with constructing a model to meet a set of conflict-free requirements. To this end, we organize the goals into several alternatives based on the interactions analyzed such that each alternative contains a set of goals that are not conflicting with each other. In other words, no conflict is allowed in an alternative. The designer can choose the most important alternative as stable kernel for software modeling first, and composes the other alternatives incrementally by applying predefined conflict resolution mechanism. More detailed method about GDUC please refers [19].

2.3. Fuzzy Object Oriented Modeling (FOOM)

Fuzzy Object Oriented Modeling (FOOM) [20] is a modeling approach to analyzing imprecise requirements which extends the traditional OOM along several dimensions: (1) to extend a class to a fuzzy class which classifies objects with similar properties, (2) to encapsulate fuzzy rules in a class to describe the relationship between attributes, (3) to evaluate fuzzy class memberships by considering both static and dynamic properties, and (4) to model uncertain fuzzy associations between classes. Several kinds of fuzziness that are required to model imprecise requirements are identified:

- classes with imprecise boundary to describe a group of objects with similar attributes, similar operations and similar relationships;
- rules with linguistic terms that are encapsulated in a class to describe the relationships between attributes;
- ranges of an attribute with linguistic values or typical values in a class to define the set of allowed values that instances of that class may take for the attribute;
- the membership degree (i.e. ISA degree) between an object and a class, and between a subclass and its super class (i.e. AKO degree) can be mapped to the interval [0,1]; and
- associations between classes that an object instance may participate to some extent.

FOOM uses fuzzy inclusion technique to compute the compatibility degree between a class and an object, and the class membership between a class and its subclass (i.e. perceptual fuzziness). Instead of using fuzzy inclusion technique to clustering data, the focus is to determine the compatibility between objects based on their properties (i.e. attribute, operation, and association).

2.4. Model transformation approaches

Graph transformation. Varró et al [26] 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 the before and after states.

Relational transformation. The basic idea is to state the source and target element type of a relation and specify it using constraints. In its pure form, such specification is non-executable. However, declarative constraints can be given executable semantic, suchlike in the case of logic programming. In fact, logic programming with its unification-based matching, search, and backtracking seems a natural choice to implement the relational approach, where predicates can be used to describe the relations. Gerber et al. [13] 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 [5] 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. Thereafter, programmers can use these APIs to create, manipulate and modify models that conform to a metamodel contained by the repository.

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. [7] 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. These generated programs are based on the dMOF repository.

Table 1 A Summary Of The Benchmark Programs

<table>
<thead>
<tr>
<th>Objective</th>
<th>Engineering</th>
<th>Pattern selection</th>
<th>Pattern application</th>
<th>Pattern usage</th>
<th>Abstraction level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Budinsky et al. 1996 [2]</td>
<td>auto. Pattern application</td>
<td>forward eng.</td>
<td>n/a</td>
<td>script language</td>
<td>n/a coding level</td>
</tr>
<tr>
<td>Eden et al. 1997 [9]</td>
<td>auto. pattern application</td>
<td>forward eng.</td>
<td>n/a</td>
<td>meta-programming</td>
<td>n/a coding level</td>
</tr>
<tr>
<td>Jeon et al. 2002 [16]</td>
<td>auto. pattern selection</td>
<td>forward eng.</td>
<td>structure-based (hot-spot)</td>
<td>rule-based transformation</td>
<td>n/a coding level</td>
</tr>
<tr>
<td>O’Cinneide and Nixon 2000 [4]</td>
<td>re-structure legacy system to provide flexibility</td>
<td>re-eng.</td>
<td>evolution based (precursor)</td>
<td>layered transformation</td>
<td>improve flexibility coding level</td>
</tr>
<tr>
<td>Tahvildari and Kontogiannis 2002 [25]</td>
<td>re-structure legacy system to improve quality</td>
<td>re-eng.</td>
<td>requirement based</td>
<td>layered transformation</td>
<td>improve quality coding level</td>
</tr>
<tr>
<td>Khriss et al. 2000 [17]</td>
<td>provide traceability by using patterns</td>
<td>forward eng.</td>
<td>n/a</td>
<td>layered transformation</td>
<td>refine model design level</td>
</tr>
<tr>
<td>Huston 2001 [15]</td>
<td>examine the compatibility between design patterns and metrics</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>improve quality n/a</td>
</tr>
<tr>
<td>Gross and E. Yu 2000 [12]</td>
<td>provide a development methodology</td>
<td>forward eng.</td>
<td>requirement based</td>
<td>n/a</td>
<td>deal with nonfunctional reqt. requirement and design level</td>
</tr>
<tr>
<td>Chung et al. 2003 [3]</td>
<td>provide a methodology for developing adaptable system</td>
<td>forward eng.</td>
<td>requirement based</td>
<td>n/a</td>
<td>Improve adaptability architecture level</td>
</tr>
</tbody>
</table>

2.5. 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.

Table 1 makes a comparison of these researches from the perspectives of their objectives, engineering direction, pattern selection, pattern application and pattern usage. The first category of researches [2, 9, 16] focuses on automatic pattern application and ignores the usage of design patterns. The second category of researches [4, 15, 17, 25] focuses on how to restructure a legacy system to be more qualified for the assistance of pattern-based transformation skills. The third category [3, 12] attempts to provide a methodology to serve as a bridge between user’s 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) [8, 21]. Design patterns [11] 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 [14]. The FR-intent describes what does the pattern do, and the NFR-intent concentrates on the extension to the FR-intent to describe what quality attribute the pattern can achieve, such as reusability, maintenance, or extensibility.

3.1. FR- and NFR-structure of design 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. Figure 2 illustrates the NFR-structure and its corresponding FR-structure of Abstract Factory design pattern. Essentially, NFR-structure is an extension of FR-structure to satisfy the associated NFR-intent. In our approach, the extension plays an important guideline to help us transfer a basic model to an extension model that is compatible with the NFR-structure.
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. 3 shows the transformation rule schema that we have defined for the Observer pattern.

The procedure of applying design patterns to transform the analysis model into design model through transformation rules is depicted in Fig. 4.

5. Meeting Scheduling System

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. 5).

- **Schedule**: [initiator, meeting date and location, initiator, participants, ϕ, rigid]
- **Handle**: [system, plan meetings, initiator, ϕ, in distributed, rigid]
- **Support**: [initiator, meeting date and location, initiator, participants, support flexible and consistent date format , soft]
- **Support**: [system, meeting date, initiator, participants, support flexible and consistent date format , soft]
- **Provide**: [system, performance, ϕ, ϕ, an appropriate level, soft]

A goal in our approach is formulated as following: 
*Action*: [Actor, Target, Source, Destination, Condition, Competence]

*Action* is the verb from requirement documents that a goal intends to achieve. *Actor* refers to people who use this system or system itself that performs the action. *Target* are entities affected by the goal. *Source, destination, and condition* are optional. There are two types of directions: *source* and *destination*, to identify respectively the initial and final location.
Towards a Pattern-based Model Transformation Approach for Design Quality Enhancement

Fig. 5 Goal-driven use case model for the meeting scheduling system

Fig. 6 Interaction between goals in the system level and goals hierarchy
of objects to be communicated. Condition describes the situation prior to performing an action, which can also be used to describe the invariant that the system must keep before and after performing the action. Competence can be either rigid or soft to identify whether the goal must be satisfied utterly or to some extent.

The goal driven use cases model is shown in Fig. 5. In this model, each goals identified above will be achieved, or optimized by a use case. For example, in order to deal with the goal Support Reusable Schedule, we need perform a use case “Support reusable schedule” to achieve it. After the use case model is constructed, the analyses between the goals interaction in use case level and in system level are evaluated. According to the GDUC method, the goals interaction in the system level is built as well as the goals hierarchy based on the interaction is constructed shown in Fig 6(a) and Fig 6(b), respectively. Notices that, the goal Support Various Data Format (GSFC) and the goal Appropriate Performance (GAP) is conflicting in the system level, since the flexible data format provided will decrease the performance. In this goals hierarchy, two alternatives are identify (Fig. 6(c)) and

the initial analysis model is obtained from the goals hierarchy and the construction of the design model is proceeded based on stable kernel in an incremental fashion from G_MS to G_GSC.

According to the pattern matched procedure (Fig. 4), the goal G_MS is choice in the stable kernel for constructing the initial analysis model shown in Fig. 7 by FOOM notations. In this model, the Meet Scheduler class provides the central role for performing the scheduling function. Because the degree that a person belongs to the class Important Participant depends on his status and his role (i.e., student, staff, or faculty) in the meeting he attends, the Important Participant class is specified as a fuzzy entity class and inherited from the Participant class. Additional fuzzy rules attached in the Registration class to describe the rules for fuzzy membership function.

Since the G_MSHID represents the system can be scheduled and handled in distributed manner, a new class “Remote Object” is added to provide this requirement (Fig. 8). After browsing the design patterns, the designer chooses the remote proxy pattern to support the distributed management and
Fig. 8 Goals considered: \( \{G_{M1}, G_{M2}\} \) before applying Proxy pattern

Fig. 9 Goals considered: \( \{G_{M3}, G_{M4}\} \) after applying Proxy pattern
apply the transformation rule to refactoring the initial mode into Fig. 9. The dotted elliptic notation named “proxy” is an UML extension means that applying the proxy pattern and the dotted lines connected the corresponding classes generated from the transformation rules. In this figure, the stereotype depicts the relationships between the model and the used name in design pattern for understanding the mapping relation. For example, the <<RealSubject>> means that the MeetingScheduler class plays the role “real subject” in the remote proxy pattern as well as the Meeting Scheduler Proxy class is mapping to the proxy in the same pattern.

By considering the G_{SRS} and G_{SFC} to realize the goal Support Reusable Schedule and the goal Support Various Data Format by applying the abstract factory and the observer patterns are shown in Fig. 10 and Fig. 11, respectively. Fig. 10 illustrates the resulting model after applying Abstract Factory pattern in the meeting scheduler system to enhance Reusable Schedule defined by G_{SRS}. Instead of creating the product objects directly, the Scheduler uses Meeting Scheduler Factory for creating the product objects of Participant Preference, Meeting Date and Meeting Loc. The scheduling strategy can be simply reused in a course scheduling system in the way of designing a Course Scheduler Factory which creates objects of Teacher Preference, Class Date and Class Loc.

The G_{SFC} represents the meeting schedule system could support various data format and keep consistence between different data views, i.e., a change on one data view must make the same change on the others. To resolve the inconsistency problem, the Observer design pattern is recommended and used in Fig. 11. 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.

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,
Towards a Pattern-based Model Transformation Approach for Design Quality Enhancement

and enhanced by a known skill. Our approach offers two benefits: (1) deriving candidate design patterns from analysis model based on the goals hierarchy; and (2) transforming analysis model into design model incrementally based on the design pattern transformation rules for dealing with the non-functional requirements.

References


Yong-Yi FanJiang

Yong-Yi FanJiang received M.S. and Ph.D. degrees in Computer Science and Information Engineering from National Chiao Tung University and National Central University in 1996 and 2004, respectively. He is currently an Assistant Professor in the Department of Computer Science and Information Engineering of Fu Jen Catholic University, Taiwan. His research interests include agent-based software engineering, web-based computing and software engineering for embedded system. He is a member of the Software Engineering Association of Taiwan (SEAT). Contact him at yyfanj@csie.fju.edu.tw

Nien-Lin Hsueh

Nien-Lin Hsueh is an assistant professor in the Department of Information Engineering and Computer Science at Feng Chia University in Taiwan. His research interests include object-oriented methodologies, design pattern, software process improvement, and software quality assurance. He was the technology consultant in the CMMI®-based process improvement project of OIT at Feng Chia University. He received his PhD in computer science from National Central University in Taiwan. Contact him at nlhsueh@fcu.edu.tw.

Jonathan Lee

Jonathan Lee is a professor in the Computer Science and Information Engineering at National Central University (NCU) in Taiwan, and was the department chairman from 1999 to 2002. He is currently the director of Computer Center at NCU. His research interests include agent-based software engineering, service-oriented computing, and software engineering with computational intelligence. Professor Jonathan Lee has published more than 100 refereed publications papers and three edited books. He received his Ph.D in computer science from Texas A&M University. He is a senior member of the IEEE Computer Society and a member of the ACM. Contact him at yjlee@selab.csie.ncu.edu.tw.