King Saud University

Journal of King Saud University – Computer and Information Sciences

www.ksu.edu.sa
www.sciencedirect.com

Pattern-based model refactoring for the introduction association relationship

Boulbaba Ben Ammar [*], Mohamed Tahar Bhiri

Faculty of Sciences of Sfax, Sfax University, Tunisia

Received 9 October 2013; revised 12 March 2014; accepted 5 June 2014
Available online 16 April 2015

KEYWORDS

Model refactoring;
UML;
B;
CSP;
Association relationship

Abstract Refactoring is an important software development process involving the restructuring of a model to improve its internal qualities without changing its external behavior. In this paper, we propose a new approach of model refactoring based on the combined use of UML, B and CSP. UML models are described by class diagrams, OCL constraints, and state machine diagrams. We detail a refactoring pattern that allows for the introduction of an association relationship between two existing classes. We illustrate our proposal by giving a case study involving the SAAT (Software Architecture Analysis Tool) system.

1. Introduction

Refactoring is a reorganization activity that aims to improve the internal structure of an existing body of code while maintaining its external behavior. This activity enhances the quality characteristics of a software system, including extensibility (during evolutionary maintenance), reusability, and efficiency. Various approaches have been proposed in the literature on the code refactoring technique. Fowler (1999) has, for instance, offered a catalog of refactoring rules applicable to the static part of a Java program, including "RenameClass", "ExtractClass", "MoveOperation", "MoveAttribute", and "RenameOperation".

More recently, the refactoring technique has also been adopted by several Agile software development methods (Shore and Warden, 2007) such as XP (Baumeister and Weber, 2013) and Scrum (Schwaber and Sutherland, 2013). In fact, they involve a Test-Driven Development (TDD) which is quick cycle consisting of three phases: test, coding and refactoring.

Refactoring tools are also available for most object-oriented languages, including Java, Smalltalk, C++, C#, Delphi and Eiffel, and for integrated development environments, such as Eclipse, NetBeans, and Oracle JDeveloper. These code refactoring rules have, however, often been defined informally, with no relationship being established between model quality and the rules. Several attempts have recently been made to overcome this inadequacy, with special focus on the application of the refactoring technique on standard models, including UML (Mens et al., 2007).

In this paper, we provide a new approach of model refactoring based on the combined use of UML, B (Abrial, 1996),

* Corresponding author.
E-mail addresses: Boulbaba.Ben-Ammar@live.fr (B. Ben Ammar), Tahar_Bhiri@yahoo.fr (M.T. Bhiri).

Peer review under responsibility of King Saud University.

http://dx.doi.org/10.1016/j.jksuci.2014.06.012

and CSP (Hoare, 2004). UML models are described by class diagrams, OCL constraints, and state machine diagrams. Specifically, we propose a catalog of refactoring patterns that are described in the same framework and formalized into B and CSP. These refactoring patterns cover the basic concepts of the object-oriented approach: conceptual relationships between classes (association and generalization/specialization), polymorphism, redefinition, abstract and generic class, and delegation.

The preservation of behavior after the application of refactoring is assigned to the tools associated to B (the prover of the ***Atelier B*** (Engineering, 2009)) and CSP (the model-checker ***FDR2*** (Goldsmith, 2005)). In fact, several researchers have defined systematic rules for the translation of UML into both B (Idani et al., 2009) and CSP (Rasch and Wehrheim, 2003) languages. Several studies have previously reported on the successful application of the B method in the development of various complex real-life applications, including the first driverless metro in the city of Paris, METEOR project (Behm et al., 1999). This method represents one of the few formal methods that has robust commercially available support tools for the entire development lifecycle, from specification down to code generation. Although this method is highly recommended for the verification of static properties such as safety, it is not used for checking dynamic properties such as liveness. For this reason, we have opted for the use of the CSP language.

The remaining parts of the paper will be structured as follows. Section 2 will provide an overview of related works on the topic under investigation. Section 3 will define our proposed approach. In Section 4, we will give a general description of the refactoring pattern. In Section 5, we will detail the pattern of association relationship introduction. Section 6 will be devoted to illustrating our proposal through the use of the SAAT system. Finally, the conclusion will summarize the major findings and provide new perspectives on model refactoring research.

2. Related works

Table 1 summarizes the major features characterizing the refactoring approaches so far proposed for the UML model using a set of evaluation criteria that are commonly cited in the literature. According to the MDE approach, refactoring can be considered a transaction processing system (Mens and Gorp, 2005) that introduces changes (without adding details) to the structure of a model. Unlike refinement, which is considered as a vertical model transformation, refactoring is a horizontal model transformation. In other words, the refactoring process does not lead to a change in the level of abstraction: the source model (before refactoring) and target model (after refactoring) remain in the same level of abstraction.

In software-driven engineering models, refactoring techniques are very limited (Allem and Mens, 2007). Several researchers (Gorp et al., 2003, Mens, 2006, Mens et al., 2007, Mens and Tourwe, 2004) indicate that taking the whole model refactoring process into consideration remains one of the challenging tasks. This process involves six major activities:

1. Identify which parts of the model should be refactored.
2. Decide on which refactoring rules to be applied to which areas.
3. Ensure that once applied refactoring would preserve model behavior and consistency.
4. Automate the application of refactoring.
5. Assess the impact of refactoring on software quality criteria (complexity, legibility, adaptability) or process (productivity, cost, effort).
6. Synchronize the refactored model and other artifacts, such as source code, documentation, specifications and tests.

The work described in Markovic (2008) offers a catalog of refactoring operations inspired by the list of operations previously described by Fowler (1999). The proposed operations are applicable on class diagrams and expressed by a QVT-based formalization of model transformation. The impacts of a refactoring operation on OCL constraints and object diagrams have also been described.

Other researchers (Gorp et al., 2003) proposed an extension of the UML meta-model that allowed for a better specification of two pre/postcontion operators in Refactoring: “Pull Up Method” and “Extract Method”. This extension also conferred tools with other abilities: check pre/post-conditions,

Table 1 Summary of related works on the refactoring approaches for the UML model.

Approach of	Markovic (2008)	Mens (2006) and Mens and Gorp (2005)	van Kempen et al. (2005)	Mens et al. (2007)	Marković and Baar (2008)	Sunyé et al. (2001)	Correa and Werner (2007)
Consideration of class diagram	Yes	Partial	No	Yes	Yes	Yes	No
Consideration of state machine diagram	No	No	Yes	Yes	No	Yes	No
Consideration of OCL constraints	Yes	Yes	No	No	Yes	No	Yes
Behavior preservation	Transformation of model formalized into QVT	Meta-modeling	UML to CSP process	UML to graphs	Graph grammars	Rewriting	Rewriting
Tool	Supporting QVT	OCL query engine	Supporting CSP	Fujaba for the graph transformation	Formalism based on graph grammars	No	No
Detection of refactoring	No	Design smells	No	Best suited refactoring	No	No	OCL smells