Available online at www.sciencedirect.com
SCIENCE DIRECT
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 127 (2005) 113–128
www.elsevier.com/locate/entcs

Detecting Structural Refactoring Conflicts Using Critical Pair Analysis

Tom Mens[1]

Software Engineering Lab
Université de Mons-Hainaut
B-7000 Mons, Belgium

Gabriele Taentzer and Olga Runge[2]

Technische Universität Berlin
D-10587 Berlin, Germany

Abstract

Refactorings are program transformations that improve the software structure while preserving the external behaviour. In spite of this very useful property, refactorings can still give rise to structural conflicts when parallel evolutions to the same software are made by different developers. This paper explores this problem of structural evolution conflicts in a formal way by using graph transformation and critical pair analysis. Based on experiments carried out in the graph transformation tool AGG, we show how this formalism can be exploited to detect and resolve refactoring conflicts.

Keywords: refactoring, restructuring, graph transformation, critical pair analysis, evolution conflicts, parallel changes

1 Introduction

Refactoring is a commonly accepted technique to improve the structure of object-oriented software [2]. Nevertheless, there are still a number of problems if we want to apply this technique in a collaborative setting, where different software developers can make changes to the software in parallel.

[1] Email:tom.mens@umh.ac.be
[2] Email:gabi@cs.tu-berlin.de

doi:10.1016/j.entcs.2004.08.038

To illustrate these problems, consider the scenario of a large software development team, where two developers independently decide to refactor the same software. It is possible that these parallel refactorings are incompatible, in the sense that they cannot be combined together. As an example, assume that a *Move Variable* refactoring and an *Encapsulate Variable* refactoring are applied in parallel to the same variable in the same class. Both refactorings are clearly in conflict since they cannot be serialised as they both affect the same variable in different incompatible ways.

It is also possible that two parallel refactorings can only be combined in a particular order. As an example, assume that a *Rename Variable* refactoring and an *Encapsulate Variable* refactoring are applied in parallel to the same variable in the same class. One can decide to rename the variable first, and then encapsulate it, but not the other way round. The reason is that the encapsulation introduces an auxiliary setter and getter method whose names rely on the variable name.

To address the problems illustrated above, we propose to take a formal approach based on *graph transformation* and *critical pair analysis* [1,4,5]. We will perform a feasibility study using the AGG tool. As such, the contribution of our paper will be twofold:

- to show the feasibility of the technique of critical pair analysis for a new practical application;
- to support refactoring tool developers with a formal means to analyse the consistency of refactoring suites, and to allow them to identify unanticipated dependencies between pairs of refactorings.

2 The AGG tool

We decided to use the tool AGG (see `http://tfs.cs.tu-berlin.de/agg`) for our experiments. It is the only graph transformation tool we are aware of that supports critical pair analysis, a crucial ingredient of our approach towards the detection of refactoring conflicts.

2.1 Specifying graph transformations

To reason about object-oriented software evolution, we specify object-oriented programs as graphs, that have to respect the constraints specified by a *type graph*. This type graph acts as an object-oriented metamodel. The metamodel we expressed in AGG is shown in Figure 1. It expresses the basic object-oriented concepts (such as classes, methods and variables), their attributes (such as name and visibility), and their relationships (such as inheritance,