Available online at www.sciencedirect.com
SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 130 (2005) 281–300

www.elsevier.com/locate/entcs

Refactoring Towards a Layered Architecture

Márcio Cornélio[1]

Departamento de Sistemas Computacionais
Escola Politécnica — Universidade de Pernambuco
Recife, PE, Brasil

Ana Cavalcanti[2]

Department of Computer Science
University of York
York, England

Augusto Sampaio[3]

Centro de Informática
Universidade Federal de Pernambuco
Recife, PE, Brasil

Abstract

In this paper we present how refactoring of object-oriented programs can be accomplished by using formal refinement. Our approach is based on the use of refactoring rules designed for a sequential object-oriented language of refinement (ROOL) similar to Java. We define a strategy that aims at structuring programs according to a layered architecture that involves the application of refactoring rules, object-oriented programming laws, and data and algorithm refinement. As the laws are proved in a weakest precondition semantics of ROOL, correctness of refactoring is ensured by construction.

Keywords: Refactoring, Formal Refinement, Refinement Calculus

[1] Email:mlc@upe.poli.br
[2] Email:Ana.Cavalcanti@cs.york.ac.uk
[3] Email:acas@cin.ufpe.br

doi:10.1016/j.entcs.2005.03.015

1 Introduction

Object-oriented programming has been acclaimed as a means to obtain software that is easier to modify than conventional software [20]. However, changing an object-oriented program often requires structural changes such as moving attributes and methods between classes, and partitioning one complex class into several ones. Such modifications should change just the internal software structure, without affecting the software behaviour as perceived by users. This activity is called *refactoring* [16]. Work on refactoring usually describes the steps used for program modification in a rather informal way [16,23,25].

In our approach, formal refactoring is achieved by the application of programming laws that deal with commands as well as with object-oriented features like methods and classes [3,4]. These laws were proposed for ROOL [8,7], an acronym for Refinement Object-Oriented Language, which is a subset of sequential Java with classes, inheritance, visibility control for attributes, dynamic binding, and recursion.

Programming laws are the basis for the derivation of refactoring rules, along with laws that lead to data refinement of classes [12]. These laws precisely indicate the modifications that can be done to a program, with corresponding proof obligations. Using laws, program development is justified and documented. Program transformations accomplished by the use of refactoring rules and programming laws preserve program behaviour [12]. Our language has a weakest precondition semantics, which supports the formal justification of the laws we use and, consequently, of our strategy. The proof that of soundness of all laws proposed for ROOL [3,12,4] with respect to a weakest precondition semantics [8,7] is presented in [12].

A system structured according to an architecture composed of independent layers of software that deal, in an orthogonal way, with database access, GUI, distribution and functional requirements, has classes with purposes clearly separated [6]. Well-structured programs are essential to improve reuse and extensibility. Using a layered architecture, we can, for instance, integrate Object-Oriented Programming Languages and Relational Databases without compromising software quality factors like reusability and extensibility [26].

In this paper we show how refactoring of object-oriented programs can be accomplished by using refactoring rules [12] and programming laws [3,4]. We present a refactoring strategy, exemplifying its application with the use of template classes. Using this strategy, we refactor a program that is representative of a number of real applications.

Our case study was first reported, and informally developed, in [26] and concerns the integration of object-oriented programming languages with re-