# Automated Elaboration of Refactoring Plans

F. Javier Pérez García

Universidad de Valladolid
jperez@infor.uva.es

## 1 Introduction

Current approaches to refactoring automation [1] are focused on finding and applying each individual transformation step. When decided which and where a refactoring opportunity exists, development tools are used to automatically apply the refactoring and test its effects. This kind of methodologies are design improvement oriented. If deeper redesigning is needed, for example when starting a framework based product-line from legacy software, refactorings must be directed to a full system new design proposal. Assuming that a sketch of the desirable system design exists, this work explores how to check if the new design could be derived from the original system using the available set of defined refactorings. A small example is presented to show how the full transformation step sequence could be obtained in case it exists.

The transformation sequence is understood as a chain of refactorings that builds up a "Refactoring Plan". This work aims to support the automation of this refactoring process.

## 2 Refactoring Plan Issues

The problem of finding the transformation steps that turn a state into another, is seen as a formal language problem. Representing software, specially its structural relationships, is straightforward with graphs, so graphs, Graph Grammars (GG) and Graph Rewriting Systems (GRS) seems a natural way to formalize the problem.

The Refactoring Plan can be worked out with a graph grammar where the software system is the starting node, refactorings are the production rules and the desirable design is a “word” formed by the available symbols (classes, fields, methods, . . . ). While the research being at a first stage, this “word” is expected to be a new design, that includes all elements from the current one, where the Refactoring Plan has already been executed. Validating if the desirable design state could be derived from the original system, using the available set of defined refactorings, is similar to the membership problem. Obtaining the Refactoring Plan, is the same problem as obtaining the derivation sequence.

## 3 Approach and Example

To formalize refactorings, a formal graph representation of Object-Oriented (OO) software is needed. It must support elements of OO paradigm and their structural relationships. The chosen formalism [2], presents a GRS based on directed typed graphs and is

already aimed at formalization of refactorings. This formalism defines a representation of refactorings as graph transformation rules and it is designed to represent also key concepts such as positive application conditions, negative application conditions and behavior preserving invariants.

An approach to the membership problem and to the finding of the derivation sequence, is searched in the world of GGs. The main purpose of this work is to find a GG for which the membership problem is solved and represent refactorings, with the other chosen formalism [2], within this grammar. A possibly adequate GG is defined by Layered Graph Grammars [3](LGGs), and their extension Contextual Layered Graph Grammars [4] (CLGGs). In CLGGs, the set of available rules are divided in different subsets classified in ordered layers. To parse a graph, rules are restricted to be applied in the order given by the layers. In addition, application of rules layer by layer, reduce the graph and consequently, the space of search is also reduced in each layer step.

The approach presented in this work is inspired by the way that CLGGs behave. A set of refactorings are ordered and assigned different layers. To be sure that these transformations lead to a some kind of reduction of the graph, "normal" refactorings are decomposed into micro-refactorings, based on Opdyke's low-level refactorings [5]. Micro-refactorings perform small behavior preserving operations such as creation, deletion, renaming and moving of classes, fields and methods. Micro-refactorings are layered to state an application order. To avoid infinite application of canceling micro-refactorings, for example, moving a method back and forth between two classes, some restrictions are applied. For example, a micro-refactoring that moves a method can only be applied once on that method.

## 4 Results and Future Work

This work present a starting step in researching to support the automated elaboration of Refactoring Plans. Small examples are used to explore the formalisms that can be applied to support this process.

Next steps include extending the set of refactorings supported. It must be checked if it is true that every refactoring can be represented as a chain of micro-refactorings. It must be proven that composing micro-refactorings is equivalent to executing "normal" refactorings. The interpretation of the proposed technique as a CLGGs must be formalized too. Validating the process through tools like AGG [6] or FUJABA [7], will be a main issue to support automated elaboration of Refactoring Plans. If needed, extensions to available tools must be developed.

## References


1. T Mens and T. Tourwé. A survey of software refactoring. *IEEE Trans. Softw. Eng.*, 30(2):126–139, 2004.
2. Tom Mens, Serge Demeyer, and D Janssens. Formilising behaviour preserving program transformations. In *Graph Transformation - 1st International Conference on Graph Transformation*, volume 2505 of *Lecture Notes in Computer Science*, pages 286–301. Springer-Verlag, 2002.

3. J. Rekers and Andy Schurr. Defining and parsing visual languages with layered graph grammars. *Journal of Visual Languages and Computing*, 8(1):27–55, 1997.
4. Paolo Bottoni, Gabriele Taentzer, and Andy Schürr. Efficient parsing of visual languages based on critical pair analysis and contextual layered graph transformation. In *VL '00: Proceedings of the 2000 IEEE International Symposium on Visual Languages (VL'00)*, page 59, Washington, DC, USA, 2000. IEEE Computer Society.
5. W.F. Opdyke. *Refactoring Object-Oriented Frameworks*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1992. also Technical Report UIUCDCS-R-92-1759.
6. Gabriele Taentzer. Agg: A graph transformation environment for modeling and validation of software. In *AGTIVE*, pages 446–453, 2003.
7. Ulrich Nickel, Jörg Niere, and Albert Zündorf. The fujaba environment. In *ICSE*, pages 742–745, 2000.