

A Refactoring Discovering Tool based on Graph Transformation

Javier Pérez, Yania Crespo
Departamento de Informática
Universidad de Valladolid
{jperez,yania}@infor.uva.es

Abstract

One of the problems of documenting software evolution arises with the extensive use of refactorings. Finding and documenting refactorings is usually harder than other changes performed to an evolving system. We introduce a tool prototype, based on graph transformation, to discover refactoring sequences between two versions of a software system. When a refactoring sequence exists, our tool can also help reveal the functional equivalence between the two versions of the system, at least, as far as refactorings can assure behaviour preservation.

1 Introduction

Efforts to include refactorings as a regular technique in software development have led refactoring support to be commonly integrated into development environments (*i.e.* Eclipse Development Platform, IntelliJ® Idea, NetBeans, etc.). Finding refactorings, now they are extensively used, is one of the problems of software evolution [2, 3]. For version management tools, for example, refactorings are modifications more difficult to deal with than any other kind of changes.

Our tool prototype implements a method, based on graph transformation [5, 8], to discover refactoring sequences between two versions of a software system. In case a refactoring sequence exists, our tool can also help reveal the functional equivalence between the two versions of the system, at least, as far as refactorings can assure behaviour preservation.

2 Graph parsing approach

To search refactorings we use graph transformation as an underlying formalism to represent Object-Oriented software and refactorings themselves. Refactorings involve modification of the system structure,

so we believe that graph transformation, which focuses on description and manipulation of structural information, is a quite appropriate formalism.

In [4, 7] the graph transformation approach is shown to be valid for refactoring formalisation. In these works, programs and refactorings are represented with graphs, in a language independent way, using a kind of abstract syntax trees with an adequate expressiveness level for the problem. This representation format is claimed to be language independent and very simple, with the purpose of making it easy to use and as flexible as possible. Therefore, as suggested in [7], it was necessary to extend the graph format to represent ‘real’ programs containing specific elements and constructions of a particular language. We have developed an extension to represent Java programs which we have named ‘Java program graphs’.

Once programs have been represented with graphs, and refactoring operations have been described as graph transformation rules, we apply graph parsing algorithms to find the refactoring sequence between two different versions of a software system. We address the problem of finding a transformation sequence from one version of the system to another as a state space search problem. With this approach we identify: **the original/old system** as a start state, **refactoring operations** as state changing operations (edges), **the refactored/new system** as the goal state, **the problem of whether a refactoring sequence exists** as a reachability problem, and **a refactoring sequence** as the path from the start state to the goal state.

We propose a basic search algorithm to look for refactoring sequences. In order to allow some kind of guided search, we base our solution in the use of refactoring preconditions and postconditions. So our approach needs refactoring definitions which include preconditions and postconditions.

The main idea of our algorithm is to iteratively modify the start state applying refactoring graph transformation rules. The set of selectable refactorings at each

iteration is composed just of refactorings whose preconditions are held in the current state and whose postconditions are held in the goal state. When no more refactorings are selectable, the algorithm backtracks to the last transformation applied. The algorithm ends up in success when the current state graph is isomorphic to the goal state graph. The refactoring sequence is the path found to the goal state. The algorithm ends up in fail when no more refactorings can be executed, and the current and goal states are not isomorphic.

3 Our tool so far

Our refactoring discovering tool consists mainly of a **refactoring searching graph grammar** and a **plugin for the Eclipse Development Platform** (see Fig. 1), which has a strong refactoring support.

We have developed a sample implementation of some searching rules to test the validity of our approach. Up to date, the refactoring searching graph grammar searches for *pullUpMethod*, *renameMethod* and *useSuperType*, supported by the Eclipse refactoring engine, *removeMethod* and *removeClass*. Using graph representation for source code enables to adjust the detail level by adding or removing elements from the graph model. The set of searchable refactorings can be easily extended by adding more searching rules to the grammar.

We use the AGG graph transformation tool [1] as the back-end of our prototype implementation. AGG is a rule-based tool that supports an algebraic approach to graph transformation, and allows rapid prototyping for developing graph transformation systems. We have chosen AGG mainly because it supports graph parsing. Graph parsing can be used to perform depth first search with backtracking, and our algorithm can be partially implemented that way.

Our initial Eclipse plugin [6] obtains the Java program graph representation from the source code of the two versions of a system, launches the graph transformation parser and shows a part of the output dumped by the parser. From this raw information, we are able to identify the refactoring sequence found, but this is only valid for the purpose of testing our approach. There is a clear need to improve the front-end to show up the search results in a more convenient way.

4 Results and future work

We have developed a tool prototype based on graph transformation to find whether a refactoring sequence exists between two versions of a software system or

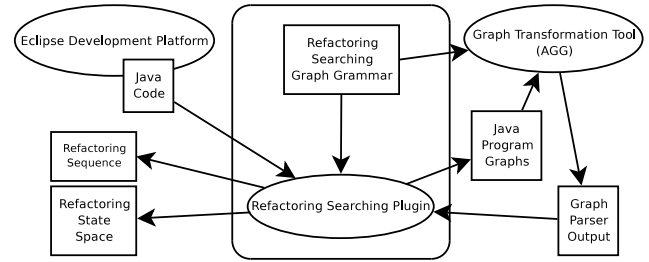


Figure 1. Outline of our tool

not. Our implementation is a proof of concept that offers very promising results.

Our immediate objectives are to improve the visualisation of results, to implement refactoring searching rules to support more refactoring operations and to measure the scalability of our technique over industrial-size systems. This will include improving rule descriptions to take benefit of new features being added in the newest versions of the AGG tool or even testing other graph transformation tools for the back-end.

References

- [1] Agg home page, graph grammar group, Technische Universität Berlin. <http://tfs.cs.tu-berlin.de/agg>.
- [2] S. Demeyer, S. Ducasse, and O. Nierstrasz. Finding refactorings via change metrics. In *OOPSLA*, pages 166–177, 2000.
- [3] D. Dig, C. Comertoglu, D. Marinov, and R. Johnson. Automatic detection of refactorings in evolving components. In *ECOOP 2006 - Object-Oriented Programming; 20th European Conference, Nantes, France, July 2006, Proceedings*, pages 404–428, 2006.
- [4] N. V. Eetvelde and D. Janssens. Refactorings as graph transformations. Technical report, Universiteit Antwerpen, 2005.
- [5] H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume II: Applications, Languages and Tools*, volume 2. World Scientific, 1999.
- [6] B. Martín Arranz. Conversor de Java a grafos AGG para Eclipse. Master’s thesis, Escuela Técnica Superior de Ingeniería Informática, Universidad de Valladolid, September 2006.
- [7] T. Mens, N. Van Eetvelde, S. Demeyer, and D. Janssens. Formalizing refactorings with graph transformations. *Journal on Software Maintenance and Evolution: Research and Practice*, 17(4):247–276, July/August 2005.
- [8] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume I: Foundations*, volume 1. World Scientific, 1997.