

Reuse based refactoring tools

Raúl Marticorena, Carlos López
Area of Languages and Computer Systems
University of Burgos (Spain)
{rmartico, clopezno}@ubu.es

Yania Crespo, Francisco Javier Pérez
Department of Computer Science
University of Valladolid (Spain)
{yania, jperez}@infor.uva.es

Abstract

Current refactoring tools work on a particular language. Each time it is intended to provide refactoring support for new languages, the same refactoring operations are defined and implemented again from scratch. This approach ignores reuse opportunities in this matter. It is possible to define a way of collecting code information suited for several languages (a family of languages) and define refactoring operations over that representation. On the other hand, it is also possible to define and implement each refactoring operation by composing previously defined and developed elements. In this paper we show the current implementation of a reuse oriented refactoring engine and its specialization for a particular language.

Key Words: refactoring, reuse, composition, language-independence

1 Introduction

One of the open trends in refactoring [2] is the construction of language-independent refactoring tools. Language independence, or at least certain language independence, allows to reuse previous efforts in defining and implementing refactoring when support for a new language must be provided. It is also aimed at obtaining a rational solution to provide refactoring operations for development environments, specially for those which support several languages.

We present a refactoring tool, using a framework based on a Minimal Object-Oriented Notation, named MOON. The use of this minimal notation allows to abstract the main concepts over a set of object-oriented languages. Language particularities must be provided by framework specialization and extension.

A refactoring engine, based on the MOON core and extensions, is responsible of checking and executing the

refactoring elements on the code. Finally, the refactored code is generated. In order to provide more powerful reuse capabilities, refactoring operations are defined by composition. A refactoring is composed of preconditions, actions and postconditions (following [3], [4]). On the one hand, conditions allow to check applicability from the point of view of behavior preservation. On the other hand, actions transform the code, changing its current state through add, remove and rename operations. Pre and postconditions are functions and predicates that query the model and actions are model transformers. Each pre, postcondition or action is stored in a repository to be reused when defining new refactoring operations. We have built an extension of the MOON framework core to deal with Java code information, in order to manage all the information of the source code.

1.1 Refactoring Engine

The refactoring engine runs the refactoring definitions and obtains a new object model with the new state. A framework definition has been used to allow a simple scheme of reuse, as can be seen in Fig. 1.

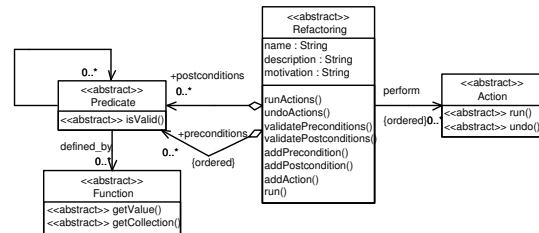


Figure 1. Refactoring Engine Framework

Using the Template Method Pattern Design [1], each refactoring has to be defined with stages, using the repository content.

1.2 Refactoring Repository

Refactoring elements are implemented as classes (see Fig. 2). These classes query or transform the current model instance. Although the model extension contains the information of real code (i.e. Java), most of the classes work with the MOON metamodel abstractions. This proposal allows to reuse the same query or action, when the related concepts are the same in several languages. For example, the precondition `ExistParameterWithName` or the action `MoveAttributeAction`, stored in the repository, are reusable for several languages.

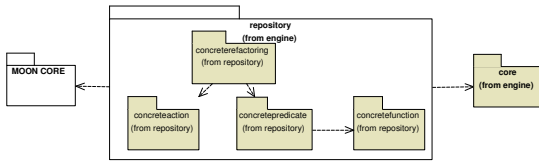


Figure 2. Repository Architecture Overview

2 Current State

The current version of the tool implements eleven refactoring operations (Fig. 3):

- add, rename and remove parameter.
- rename classes and methods.
- move attributes and methods.
- four refactoring operations, we have defined, on generic classes.

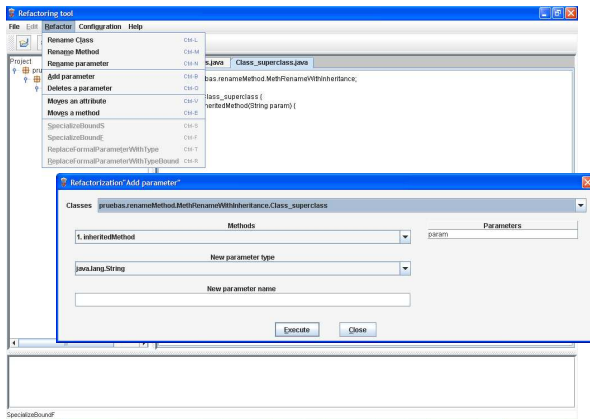


Figure 3. Refactoring Tool

Each one of these refactorings are implemented as concrete classes (extending the `Refactoring` abstract class as can be seen in Fig. 1). The refactorings are

built from instances of pre and postconditions classes and action classes, using the corresponding `add` methods of the template (Fig. 1).

The `repository` contains the implementation of these elements, allowing the programmer to compose the refactoring. If the element is not available, the programmer should add the new code needed to the repository. Hence, last refactorings to be added are implemented with a minor effort, because the complete set of their elements is already available in the repository in order to be reuse.

3 Future Works

We have presented a refactoring tool which intends to provide some advantages: certain language independence, which allows to reuse the same refactoring implementation (or a very similar one) for different languages and refactoring construction by composition, which allows to implement new refactorings from pieces already available from previously introduced refactoring operations. The current version of the tool allows to run the refactorings over a simple set of toy codes. The Java parser is being completed to support commercial code, and a C# parser (with its own framework extension) is under development to validate the solution.

We are also currently working on a declarative definition of refactorings using XML. This makes easy to compose refactorings from the repository elements using a graphical interface. Since specialization could be necessary, the declarative definition could be also specialized for different languages with a high degree of reuse.

References

- [1] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.
- [2] Tom Mens and Tom Tourwé. A survey of software refactoring. *IEEE Trans. Softw. Eng.*, 30(2):126–139, 2004.
- [3] William F. Opdyke. *Refactoring Object-Oriented Frameworks*. PhD thesis, University of Illinois at Urbana-Champaign, IL, USA, 1992.
- [4] Donald Bradley Roberts. *Practical Analysis for Refactoring*. PhD thesis, University of Illinois at Urbana-Champaign, IL, USA, 1999.