

fUML Refactoring with EMF[★]

Sebastian Geiger (1127054) and Kristof Meixner (9725208)

Business Informatics Group
Vienna Technical University

Abstract. In this work we will present some ideas and concepts for the refactoring of fUML. The main contribution of this work is the extension of existing UML refactorings to cover not only the static aspect of UML such as class diagrams but to also include refactorings for dynamic parts such as activity diagrams. In this work we will present basic concepts for refactoring with EMF and show how model semantics can be preserved through the use of OCL constraints. In the remainder of the paper we then present our tool chain and the used technologies such as EMF and Ecore and how we used them for refactoring. We also present a discussion of EMF.Refactor, which shows how such refactorings can be made available in the Eclipse GUIs such as the EMF tree editor or Papyrus.

[★] This work has been created in the context of the course “Advanced Model Engineering” (188.952) in SS14.

Table of Contents

1	Introduction.....	1
2	Motivation	1
3	Refactorings Examples.....	2
4	Refactoring of fUML diagrams.....	2
5	Tool chain and implementation	3
	5.1 Model refactoring	3
	5.2 GUI Integration	3
6	Related Works.....	3
7	Conclusion	3
	References	4

1 Introduction

The concept of refactoring is an important part of software development which should be included in every cycle of iterative and evolutionary software development. Its main goal is to reorganise software components to achieve better design, lower coupling, higher cohesion, and other attributes which make up good software design. The important aspect of refactoring is that no additional features are introduced during a refactoring step and the semantics of the software are maintained; no additional bugs or errors are introduced. Refactoring can not only be applied to source code but as well to models such as UML or fUML models. In the context of models such as class diagrams, it means that the structure of the diagram is rearranged to achieve a better design. This included changing superclass relationships, moving attributes and methods to different classes or adding new interfaces. A catalogue of possible UML refactorings can be found in .

cite markovic

fUML is an extension of UML which builds on a subset of UML classes with the purpose of adding semantics to UML models such that they can be executed on the model level. The dominant concept for this is the activity diagram. If a refactoring is performed on a UML model such as a class diagram, then any activity diagram which is related to the class diagram, has to be checked and possibly changed as well. In section 4 we will present some examples of fUML activity diagrams and present the implications that result from changing class diagrams.

Since a refactoring changes the structure of a model it is important to ensure that all changes maintain the original semantics of the model. Violating this requirement can result in models with either a different behavior, or in models which can no longer be executed. To ensure semantic preservation, two main techniques can be used. First, the refactoring can be broken down into smaller steps, each of which either guarantees to preserve the semantics of the model or makes it easier to verify that this is the case. Second, logical constraints can be used to limit refactorings on models to only those cases where semantic preservation can be ensured. For this purpose pre- and postconditions are specified with OCL constraints. A refactoring is then only applied if the original model satisfies the precondition before the refactoring is applied and the postcondition after the refactoring has been completed. Such constraints must be individually specified for each kind of refactoring that is to be performed, as such a part of this paper will discuss different OCL constraints for the refactorings that we introduce.

2 Motivation

In this paper we will present a set of refactorings and give examples of how each refactoring can be applied to a concrete model.

3 Refactorings Examples

reference insurance figure

For this we use an example for the insurance domain. Figure X. shows the class diagram of an insurance policy. The policy allows to insure cars and trucks. Both Car and Truck have some attributes that are similar (weight, registration) and others that are different (dangerousGoods, passengers).

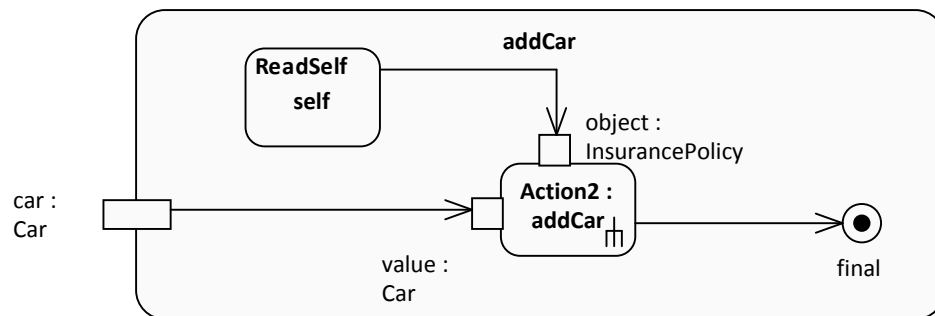
- Extract class
- Extract superclass
- Rename Class
- Rename Method
- Rename Variable
- Add / Remove Parameter
- Encapsulate Field
- Pull up field

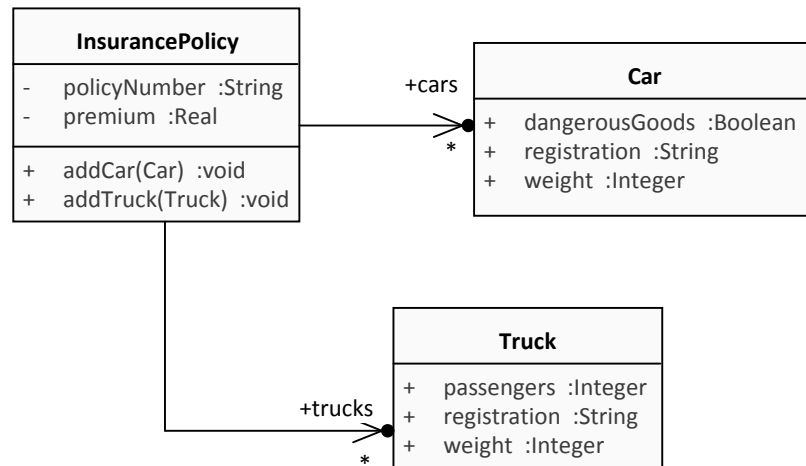
Fig. 1. Refactoring examples

In this section we will present some general refactorings such as the “extract superclass” refactoring. A list of example refactorings is presented in figure 1.

4 Refactoring of fUML diagrams

In this section we will present some general refactorings





5 Tool chain and implementation

For our tool chain we have relied on the moliz¹ repository, mainly for the ability to execute the fUML models with a virtual machine. The models are stored as XMI

5.1 Model refactoring

Describe our tool chain, how we created models, how we load them, what information of the abstract syntax we use for refactoring, etc.

5.2 GUI Integration

Describe what we did with EMF.Refactor.

6 Related Works

We have compared our works with several other available papers. In [...] there is a discussion of uml refactings which covers

some related works such as ...

[1]

7 Conclusion

We conclude this paper with...

¹ source...

References

1. ROBERTS, D. Practical analysis for refactoring. Master's thesis, University of Illinois, 1999.