동적 조립을 위한 OCL 기반의 리팩토링 명세

장필재O 김경민 김은지 김태공
인제대학교 전산학과
prericpil@naver.com, kmkim@cs.inje.ac.kr,
ejkim8511@naver.com, ktg@cs.inje.ac.kr

The Refactoring Specification for Dynamic Composition based on OCL

Pil-Jae Jang0 Kyung-Min Kim Tae-Woong Kim Un-Ji Kim
Dept. of Computer Science, Inje University

요 약

최근의 소프트웨어 개발에서는 리팩토링이 일반적인 요소로서 활발하게 이용되고 있다. 리팩토링에 대한 관심이 높아지면서 리팩토링의 자동화와 리팩토링 재사용을 위한 조립에 대한 연구가 많이 진행되고 있다. 기존의 연구들에서는 리팩토링과 리팩토링 조립에 대해 선/후행조건을 각각 명세하고 있다. 하지만 리팩토링 조립의 경우에 대한 선/후행조건은 개별적인 리팩토링들의 선/후행조건들과 중복된 기능들이 대부분이며, 사용 가능한 리팩토링의 조립이 정적으로 고정되어 있음을 의미한다. 이에 본 연구에서는 리팩토링 조립의 경우가 고정되어 있지 않고 조립의 경우에 따른 선/후행조건의 추가적인 정의 없이 동적으로 다양하게 조립여부를 확인할 수 있도록 하고자 한다. 이를 위해 리팩토링의 동적 조립을 위한 전체 프레임워크를 제안하며 OCL 기반으로 리팩토링을 명세한다.

1. 서 론

리팩토링(Refactoring)은 시스템의 기능을 유지하면서 시스템의 이해도를 높임과 동시에 유지보수를 보다 쉽게 할 수 있도록 내부 구조를 변경하는 것이다[1]. 이러한 리팩토링의 결과로 코드의 확장성, 모듈화, 재사용성, 유지보수성 같은 품질을 개선하여 개발의 속도를 높이고 코드 복잡도를 낮출 수 있다[2].

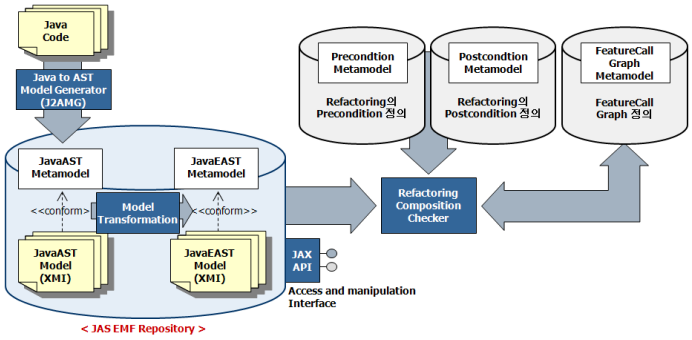
리팩토링에 대한 관심이 높아지면서 리팩토링의 자동화와 리팩토링 재사용을 위한 조립(Composition)에 대한 연구가 많이 진행되고 있다[3,4]. 기존 연구들에서는 리팩토링 자동화를 위해 각 리팩토링의 적용 가능한 경우를 정의한 선행조건(Precondition)과 리팩토링의 적용 후 만족해야하는 후행조건(Postcondition)을 명세하고 있다. 그리고 리팩토링의 조립에 대해서도 조립 가능한 경우에 대한 선/후행조건을 각각 명세하고 있다.

이것은 리팩토링 조립 시에, 첫 번째 리팩토링 적용 후 두 번째 리팩토링을 적용할 경우 첫 번째 리팩토링의 후행조건이 두 번째 리팩토링의 선행조건에 영향을 미치기 때문에 리팩토링들의 조립의 경우에 맞춰 각각의 선/후행조건들이 필요하게 되는 것이다. 하지만 이것은 개별적인 리팩토링들의 선/후행조건들과 중복된 기능들이 대부분이며, 사용 가능한 리팩토링의 조립이 정적으로 고정되어 있음을 의미한다. 그리고 새로운 종류의 리팩토링 추가 시 리팩토링의 다양한 조립의 경우에 대해서도 각각 선/후행조건을 정의해야 하는 문제점을 가지게 된다.

이에 본 연구에서는 리팩토링의 조립의 경우가 고정되어 있지 않고 조립의 경우에 따른 선/후행조건의 추가적인 정의 없이 동적으로 다양하게 조립여부를 확인할 수 있도록 하고자 한다. 리팩토링의 후행조건을 실제 리팩토링이 적용된 것과 같은 동일한 효과가 반영되도록 명세함으로써 리팩토링을 조립 가능한 요소로써 정의하고자 한다. 이를 위해 리팩토링의 동적 조립을 위한 전체 프레임워크를 제안하며 OCL 기반으로 리팩토링을 명세한다.

2. 전체 프레임워크

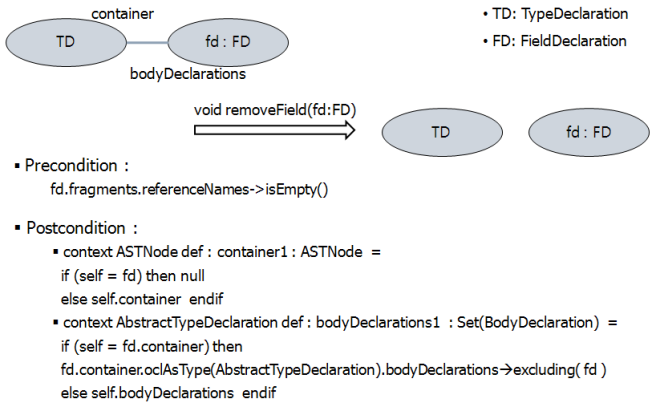
본 장에서는 리팩토링의 동적 조립을 위한 전체 프레임워크를 그림 1과 같이 제안한다. 자바로 작성된 소스코드를 MoDisco[5]에서 제안한 JavaAST(Java Abstract Syntax Tree)[6]로 산출(generate)하고(본 연구에서 구현한 Java to AST Model Generator를 이용), 이를 본 연구에서 확장한 JavaEAST(Java Extended Abstract Syntax Tree)기반의 XMI(XML Metadata Interchange)[7]문서로 변환(transformation)하여(본 연구에서 구현한 Model Transformation를 이용) 분석한다. 동적 조립이 가능하도록 리팩토링을 명세한 후 그에 대해 OCL(Object Constraint Language)[8]을 이용하여 작성한다. 작성된 OCL은 Eclipse[9] 기반의 OCL 번역기(OCL Interpreter)[10]를 통하여 실행된다. 변환된 XMI문서에 리팩토링의 선/후행조건을 이용하여 Refactoring Composition Checker를 통해 동적으로 리팩토링의 조립여부를 판단하게 된다.



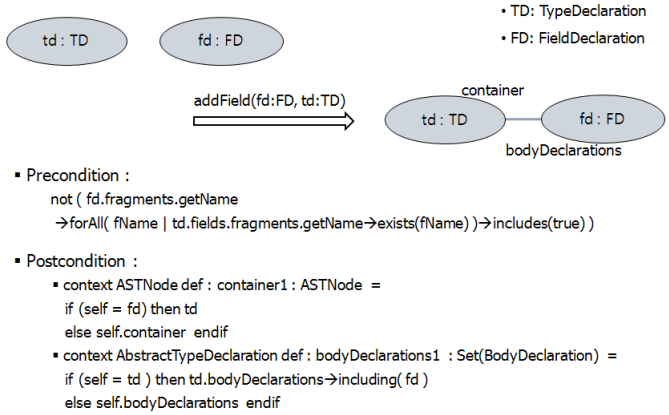
(그림 1) 리팩토링의 동적 조립을 위한 전체 프레임워크

3. OCL 기반의 리팩토링 명세

리팩토링을 동적으로 조립하기 위해서는 리팩토링 종류마다 선/후행조건을 이용하여 정확한 명세가 선행되어야 한다. 특히 리팩토링의 후행조건은 실제 리팩토링이 적용된 것과 같은 동일한 효과가 반영되도록 명세하는 것이 중요하다. 본 장에서는 'RemoveField'와 'AddField'를 그림 2, 3과 같이 OCL 기반으로 명세해본다.



(그림 2) RemoveField 명세



(그림 3) AddField 명세

'RemoveField'와 'AddField'에서 공통적으로 사용되는 Common Def는 다음과 같다.

```
context TypeDeclaration
def : fields : Set(FieldDeclaration) =
self.bodyDeclarations->select
(oclIsTypeOf(FieldDeclaration)).oclAsType(FieldDeclaration)
->asSet()

context VariableDeclarationFragment
def : getName : String =
self.name.identifier
```

4. 결 론

본 연구에서는 리팩토링의 동적 조립을 위해 전체 프레임워크를 제안했으며 OCL 기반으로 리팩토링을 명세했다. 이를 위해 소스코드를 AST 메타모델을 확장한 EAST 메타모델 기반의 XMI문서로 변환하여 분석했다. 동적 조립이 가능하도록 리팩토링의 후행조건을 실제 리팩토링이 적용된 것과 같은 동일한 효과가 반영되도록 OCL을 기반으로 명세했다. 작성된 OCL은 Refactoring Composition Checker를 통해 동적으로 리팩토링의 조립여부를 판단하게 된다. 이것은 새로운 종류의 리팩토링 추가 시, 조립에 대한 다양한 경우의 명세를 필요로 하지 않으며 해당 리팩토링의 명세만으로 다양한 조립을 가능하게 한다.

앞으로 다양한 종류의 리팩토링 명세와 리팩토링 조립 사례를 통해 본 프레임워크의 활용성에 대한 검증이 필요하다.

참고문헌

[1] Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D., Refactoring: Improving the Design of Existing Code. Addison Wesley, 1999.
[2] Mens, T., Tourwe, T. A Survey of Software Refactoring. IEEE Transactions on Software Engineering, February, 2004, 30-2 : 126~139
[3] Yoshio Kataoka, Michael D. Ernst, William G. Griswold, and David Notkin, "Automated Support for Program Refactoring using Invariants", In Proc. Int. Conf. On Software Maintenance, pages 736-743, 2001
[4] Mel O Cinneide, B.Sc., M.Sc., "Automated Application of Design Patterns: A Refactoring Approach", PhD thesis, University of Dublin, Trinity College, 2000
[5] MoDisco, MoDisco Tool- Java Abstract Syntax Discovery Tool, http://www.eclipse.org/gmt/modisco/toolBox/JavaAbstractSyntax/
[6] Eclipse, Abstract Syntax Tree, http://www.eclipse.org/articles/article.php?file=Article-JavaCodeManipulation_AST/index.html
[7] Object Management Group, MOF 2.0/XMI Mapping Specification, V2.1.1, http://www.omg.org/technology/documents/formal/xmi.htm
[8] Object Management Group, Object Constraint Language Specification, Version 2.0, http://www.omg.org/technology/documents/formal/ocl.htm
[9] Eclipse, OCL Library, http://www.cs.kent.ac.uk/projects/ocl/tools.html#documentation
[10] Eclipse Consortium, Eclipse Version 3.0.1, 2004, available at http://www.eclipse.org.