코드와 모델의 일치성을 위한 코드 저장소

오동은° 김경민 김태웅 김태공
인제대학교

The Code Repository for Consistency between Code and Model

Dong-Eun Oh° Kyung-Min Kim Tae-Woong Kim Tae-Gong Kim
Inje University

요 약

소프트웨어 개발 과정에서 산출된 설계 모델은 향후 시스템의 유지보수를 위해 꼭 필요한 산출물 중의 하나이며 시스템을 이해하는데 필수적인 요소이다. 그러나 다양한 요구사항의 변경에 따라 유지보수 단계를 거치면서 소스 코드에만 수정이 가해져 모델 정보와의 연계성이 없어지게 된다. 이에 본 논문에서는 소스 코드와 모델간의 일관성을 위한 코드 저장소를 제안한다. 모델 정보와 소스 코드와의 일관성을 지키기 위해서는 소스 코드에 있는 시스템의 행위적인 정보를 유지하는 것이 중요하다. 본 연구에서는 모델을 표현 할 수 있는 XMI를 이용하여 메타모델 기반의 코드 저장소를 구축함으로써 해결하고자 한다. 코드 저장소에는 소스 코드의 모든 정보를 추출하여 XMI 형태의 코드 모델로 저장되며 이 정보로부터 다시 실행 가능한 소스 코드를 생성할 수 있도록 한다.

1. 서 론

소프트웨어 개발 방법론은 좋은 품질의 소프트웨어를 최소의 비용으로 계획된 일정에 맞추어 개발함으로써 품질과 생산성을 향상시키기 위한 방법이다. 개발 방법론은 일반적으로 요구사항 분석, 설계, 디자인, 구현, 테스팅, 배치 단계로 이루어져 있다. 여기에서 시간 소비가 큰 요구사항 분석, 설계, 디자인 단계에서 산출되는 모델 정보는 대상을 종합적으로 이해하고 주어진 조건이나 입력에 대한 대상의 반응을 예측할 목적으로 시스템을 이해하고 분석하기 위한 중요한 정보이다.

하지만 이와 같은 전통적인 소프트웨어 개발 과정에서는 구현(Coding) 단계로 넘어오면서 이전 단계와의 연계성이 없어지는 문제점이 발생한다. 특히 시스템의 수정 및 보완처럼 유지보수가 필요한 경우에는 구현 코드에만 수정이 가해져 이전 단계와의 연계성이 더욱 없어지게 된다. 결국 시스템을 이해하고 분석하기 위한 중요한 모델 정보들이 버려지게 되는 것이다.

이것은 개발자가 수작업으로 모델 정보를 수정하고 이를 다시 코드에 반영시켜야하는 개발의 비효율성 때문이다. 이러한 문제를 해결하기 위해 기존의 CASE Tool들에서는 모델을 표현하기 위한 OMG(Object Managemnet Group) 표준인 XMI(XML Metadata Interchange)을 이용하여 모델 정보와 구현 코드 간의 자동화 기술들을 제공하지만 제한된 부분들이 많다.

이에 본 연구에서는 소스 코드와 모델간의 일치성을 위한 코드 저장소를 제안한다. 코드 저장소에서는 소스 코드에 포함되어 있는 시스템의 행위적인 정보를 유지하기 위해 소스 코드의 모든 정보를 추출한다. 그리고 그 정보를 바탕으로 메타모델 기반의 XMI문서인 코드 모델을 생성한다. 또 코드 모델의 정보를 기반으로 다시 소스 코드를 생성 있게 한다 이를 위해 소스 코드와 코드 모델간의 변환을 담당하는 'Code to Code Model Generator(C2M)'와 'Code Model Generator to Code(M2C)'를 구현한다. C2M을 통해 생성된 코드 모델의 XMI는 여러 CASE Tool에 사용되는 설계모델의 XMI로 Model Transformation과 같은 기술을 통해 상호 변환될 수 있다. 또 M2C를 통해 코드 모델에서 소스코드를 생성할 수 있다. 이것은 실제 개발자의 수작업에 의해 이루어졌던 설계 및 구현 단계에서의 작업을 자동화함으로써 개발의 효율성을 향상시킬 수 있을 것이다.

2. 소스코드 저장소

2장에서는 소스 코드 저장소 전체 구조와 프로세스 알아본다. 그리고 Java 코드에 대한 저장소의 구축을 설명한다.

2.1 코드 저장소의 전체 구조

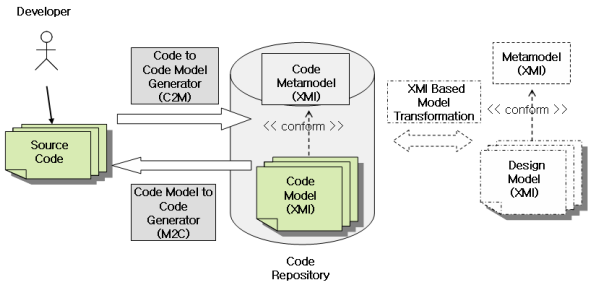

그림 1. 소스 코드 저장소의 구조

그림 1은 소스 코드 저장소의 전체 구조를 나타낸 것이다. 그림 1에서는 소스 코드가 저장소에 저장되었다가 다시 소스 코드로 생성되는 순환 구조를 확인 할 수 있는데, 이는 소스 코드 저장소를 구성하는 각각의 구성 요소를 통해 이루어진다. 소스 코드 저장소의 구성 요소는 다음과 같이 크게 4가지로 나눌 수 있다.

- Code to Code Model Generator(C2M)
- Code Model Generator to Code(M2C)
- Code Meta Model
- Code Model

C2M은 소스 코드에서 코드 모델로 생성되는 과정으로 소스 코드 저장소의 입력 부분에 해당되고 M2C는 코드 모델에서 소스 코드를 생성하는 출력 부분에 해당된다. 코드 메타모델은 입력되는 소스 코드의 모든 키워드를 표현할 수 있는 구조로 되어있어 코드 메타모델에 따라 구성되는 코드 모델이 소스 코드의 키워드들을 가지는 것이 가능하다.

그림 1에서 점선으로 표시 되어 있는 부분은 설계 모델과의 관계를 나타내는 것으로 XMI[1]로 표현된 설계 모델이라면 저장소에 저장된 코드 모델과 Model Transformation이 가능하다. 때문에 Model Transformation을 거쳐 코드 모델에서 설계 모델을 생성하거나 기존에 존재하는 설계모델과의 연계하는 과정을 통해 소스 코드와 설계 모델의 일치성을 유지할 수 있다.

2.2 저장 프로세스

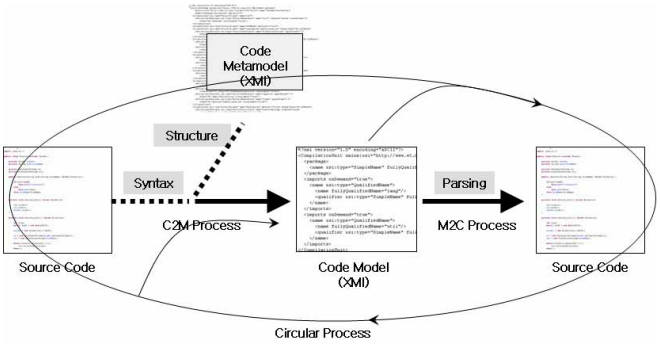

그림 2. 소스 코드 저장소의 순환 프로세스

소스 코드 저장소는 2.1절에서 설명한 C2M과 M2C의 과정을 기본 프로세스로 가진다. 소스 코드에서 코드 모델을 생성하는 C2M은 입력된 소스 코드를 파싱하여 문법 정보를 분석하는 과정으로 시작한다. 그리고 소스 코드에서 분석된 문법 정보를 코드 메타 모델에 정의되어 있는 구조대로 XMI문서를 구성하여 코드 모델을 생성하게 된다. M2C는 코드 모델에서 소스 코드를 생성하는 과정으로 코드 모델의 XMI문서를 파싱하여 문법 정보만을 추출하고 그 정보를 바탕으로 소스 코드를 생성한다.

소스 코드 저장소는 소스 코드를 코드 모델로 저장하는 것을 기본 프로세스로 가지지만 소스 코드와 설계 모델의 일치성을 위해서는 저장소의 입력과 출력이 소스 코드와 모델 중 어떤 것이 되더라도 상관없이 이루어져야 한다. 소스 코드 저장소는 그림 2에서 나타난 것과 같이 C2M과 M2C의 조합에 따른 순환 프로세스를 가질 수 있으므로 입력과 출력의 선택이 자유롭다.

2.3 소스 코드 저장소 구축

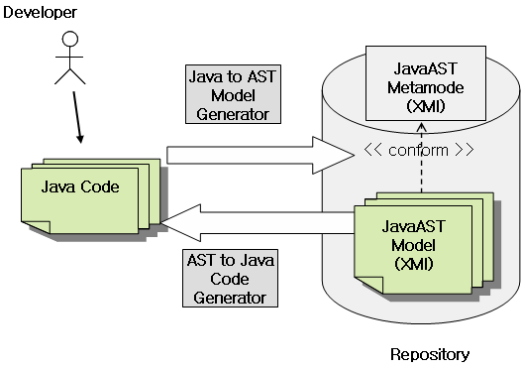

그림 3. Java 코드 저장소의 구조

2.3절에서는 본 논문에서 제시한 소스 코드 저장소의 구조와 프로세스를 Java 코드에 대해 적용시켜 Java 코드 저장소를 구현하는 것에 대해 설명한다.

2.1절과 2.2절에서 설명한 소스 코드 저장소의 구조와 프로세스를 구현하기 위해서 필요한 구현 요소는 Java 코드 파싱, 코드 메타 모델, Java 코드에서 코드 모델 생성(C2M), 코드 모델에서 Java 코드 생성(M2C) 4가지이다. 우선 Java 코드를 파싱하기 위해 Eclipse JDT/AST[2]를 사용하는데, Eclipse JDT/AST는 Java 코드에 대한 추상 구문 트리를 자동으로 구성해주고 그에 대한 API를 제공한다. 이를 이용해 Java 코드를 파싱하여 Java 코드의 문법 정보를 추출하는 기능을 구현한다.

Java 코드를 구문 정보를 표현하기 위한 코드 메타 모델은 Modisco[3]의 JavaAST MetaModel을 사용한다. Java 코드에서 추출해낸 구문 정보들을 JavaAST MetaModel의 구조에 맞게 구성하는 과정은 EMF[4] API를 이용해 구현하며 이를 이용해 코드 모델에 해당하는 JavaAST Model의 XMI문서를 생성한다. 또한 XMI문서를 파싱하여 구문 정보를 추출하
는 기능을 구현하고 그것을 바탕으로 Java AST Tree를 구성, Java 코드를 생성한다.

그림 3은 소스 코드 저장소의 각 요소를 Java 코드에 맞게 구성한 저장소의 구조를 나타낸다.

3. 소스코드와 코드 모델간의 변환

3장에서는 소스 코드의 구문을 추출하는 과정과 코드 모델의 구성 과정 그리고 코드 모델에서 소스 코드를 생성하는 과정을 설명한다.

3.1 소스코드 구문 정보 추출

코드 모델을 생성하기 위해서는 소스 코드에서 모든 구문 정보를 추출하는 과정이 필요하다. 소스 코드를 읽어 들이는 과정에서 코드의 문법에 따라 구문을 파악하고 그 구문을 추출해야하는데 본 논문에서는 Java 코드의 구문 정보를 추출하기 위해 Eclipse JDT/AST를 이용해 구현한다. JDT/AST에는 Java 코드에 대한 추상 구문 트리를 구성해주는데, 그림 4는 Java로 되어 있는 'Hello World'의 추상 구문 트리를 나타낸 것이다.

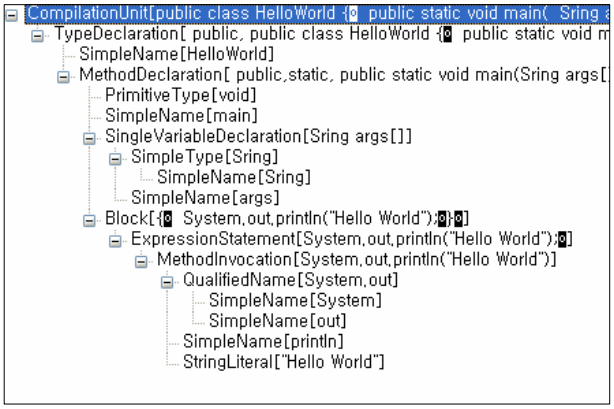

그림 4. 'Hello World'의 추상 구문 트리

Java 코드에서 구성된 추상 구문 트리에서 구문을 추출하기 위해서는 JDT/AST API의 Visitor 클래스 이용한다. Visitor 클래스는 추상 구문 트리의 노드를 루트노드부터 순차적으로 순회하면서 모든 노드를 한번 씩 방문하게 된다. 각 노드를 거쳐 갈 때 마다 노드의 정보를 가져오게 되고 이 작업이 반복되면서 추상 구문 트리에서 Java 코드 구문 정보를 추출하게 된다.

3.2 코드모델의 생성

소스 코드에서 추출된 구문 정보에서 코드 모델을 생성하기 위해서는 다음과 같은 2가지 요소가 필요하다.

- 코드 메타 모델
- 코드 모델의 XMI문서 구성

첫 번째 요소인 코드 메타 모델은 소스 코드 저장소의 기본 입력 요소에 해당되는 소스 코드의 구문 구조를 구성하고 있는데, Java 코드의 구문 구조를 표현하기 위한 메타 모델로는 Modisco에서 제공하는 JavaAST MetaModel이 있다. JavaAST MetaModel은 JDT/AST의 추상 구문 트리의 구조를 XMI문서로 구성하고 있기 때문에 Java 코드의 세부적인 구문 요소까지 모두 표현 가능하다. 그림 5는 JavaAST MetaModel의 XMI문서를 트리 구조로 나타낸 것이다.

그림 5. JavaAST MetaModel

두 번째 요소인 코드 모델의 XMI문서 구성은 소스 코드에서 추출한 구문 정보를 코드 메타 모델의 정의된 구조대로 XMI문서를 생성하는 것이다. 코드 메타 모델에 정의된 구문 구조를 알기 위서는 코드 메타 모델의 XMI 문서를 파싱해야 한다. EMF에는 XMI문서를 파싱하거나 생성하기위한 API를 제공하는데, 이를 이용하여 코드 메타 모델에 해당하는 JavaAST MetaModel을 파싱하여 구조를 파악하는 기능을 구현한다.

Java 코드 모델의 XMI문서를 구성하는 과정은 우선 Java 코드 파싱과정에서 추상 구문 트리의 각 노드에 대응되는 JavaAST MetaModel의 구문 요소를 찾는다. 그리고 추상 구문 트리에서 추출한 구문 정보를 JavaAST MeataModel에서 찾은 구문 요소의 구조대로 XMI문서를 구성해 나간다. 이 과정이 Java 코드의 추상 구문 트리를 구성하는 노드의 수만큼 반복되면서 XMI문서가 완성되면 입력된 Java 코드에 대한 JavaAST Model이 생성되게 된다. 그림 6은 4.1절의 Hello World Java 코드에 대한 코드 모델을 트리 형태로 나타낸 것이다.

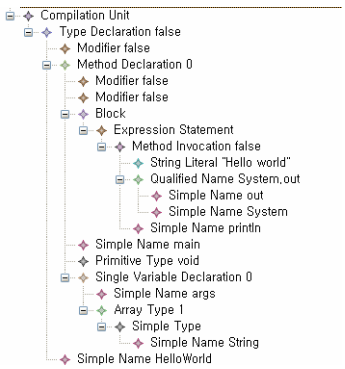

그림 6. 'Hello World' JavaAST Model

3.3 소스코드의 생성

코드 모델에서 소스 코드를 생성하기 위해서는 코드 모델의 XMI문서를 파싱하여 구문 정보를 추출해야 한다.

JavaAST Model에서 Java 코드를 생성하는 과정은 우선 EMF API 이용해 JavaAST Model의 XMI문서를 파싱하여 구문 정보를 추출하는 기능을 구현한다. 그리고 JavaAST Model의 XMI문서에서 추출된 구문 정보를 바탕으로 Java 코드에 대한 추상 구문 트리를 구성한다. 추상 구문 트리의 각 노드에는 Java 코드의 구문 정보가 포함되어 있으므로 추상 구문 트리를 구문 정보를 추출하여 Java 코드를 생성 할 수 있다. 그림 7은 그림 6의 Hello World에서 Java 코드를 생성하는 과정을 나타낸 것이다.

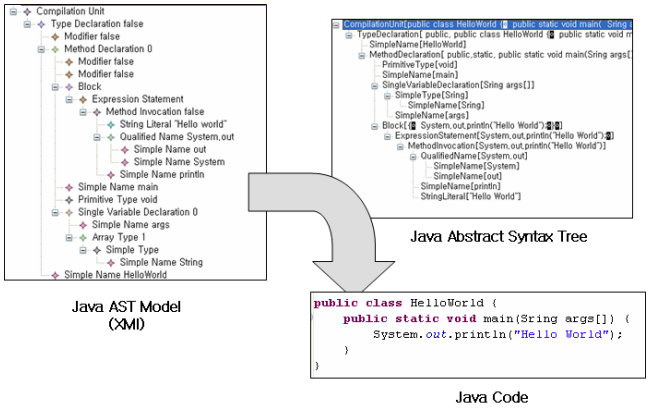

그림 7. Java Code의 생성 과정

4. 적용사례

4장에서는 Java로 구현된 Stack 코드를 코드 저장소에 입력시켜 입력된 코드에 대한 JavaAST Model이 생성된 모습을 사례로 들어 본다.

```
public class Stack1{
    private int array[];
    private int top;
    private int size;

    public Stack1(){
        array = new int[5];
        top = 0;
    }
    public Stack1(int size){
        this.size = size;
        init();
    }
    public void init(){
        array = new int[size];
        top = 0;
    }
    public void push(int data){
        if(!isFull()){
        array[top] =data;
        top++;
    }else
        System.out.println("Full");
    }
    public int pop(){
     if(!isEmpty()){
        top--;
        return array[top];
    }else {
         System.out.println("Empty");
         return 0;
        }
    }

    public boolean isFull(){
        return (top >= size);
    }
    public boolean isEmpty(){
        return (top <=0);
    }
    public int getUsedSize(){
        return top;
    }
    public int getSize(){
        return size;
    }
}
```

그림 8. Stack 코드

```
<bodyDeclarations xsi:type="FieldDeclaration">
  <modifiers xsi:type="Modifier" private="true"/>
  <fragments extraDimensions="1">
    <name fullyQualifiedName="array" identifier="array" declaration="true"/>
  </fragments>
  <type xsi:type="PrimitiveType" code="int"/>
</bodyDeclarations>
<bodyDeclarations xsi:type="FieldDeclaration">
  <modifiers xsi:type="Modifier" private="true"/>
  <fragments>
    <name fullyQualifiedName="top" identifier="top" declaration="true"/>
  </fragments>
  <type xsi:type="PrimitiveType" code="int"/>
</bodyDeclarations>
<bodyDeclarations xsi:type="FieldDeclaration">
  <modifiers xsi:type="Modifier" private="true"/>
  <fragments>
    <name fullyQualifiedName="size" identifier="size" declaration="true"/>
  </fragments>
  <type xsi:type="PrimitiveType" code="int"/>
```

그림 9. Stack 코드에 대한 XMI문서

그림 8은 코드 저장소에 입력되는 Stack 코드이다. 코드 저장소는 입력된 Stack 코드를 3장에서 설명한 과정을 통해 구문 정보를 추출하고 JavaAST Model의 XMI 문서를 생성한다. 그림 9에서는 최종 생성된 XMI문서의 일부분을 나타냈는데 중앙에 표시된 부분을 통해 입력된 Stack 코드의 구문 정보가 XMI에 저장되어 것을 확인할 수 있다.

5. 결 론

본 논문에서는 소스 코드와 코드 모델의 일치성을 위한 코드 저장소를 제안했다. 2장에서 코드 저장소의 전체 구조와 코드 저장소가 가지는 전체 프로세스에 대해 정의했다. 그리고 Java 코드에 대한 코드 저장소의 구축 과정을 설명했다. 3장에서는 코드 저장소 구축 시 필요한 구현 요소인 소스 코드와 코드 모델간의 변환 대해 살펴보았다. 입력된 Java 코드를 파싱하여 구문 정보를 추출한 뒤, 추출된 구문 정보를 바탕으로 XMI 문서를 구성하여 코드 모델에 해당하는 JavaAST Model을 생성했다. 그리고 JavaAST Model에서 다시 Java 코드를 생성함으로써, 소스 코드와 코드 모델 간의 일치성을 유지시켰다.

본 연구를 통해 소스 코드와 코드 모델간의 일치성을 유지시킴으로써 다양한 소프트웨어 개발에 활용될 수 있을 것이다. 시스템 변경 시 개발자의 수작업에 의해 이루어졌던 설계 및 구현 단계에서의 수정 작업을 자동화함으로써 개발의 효율성이 향상될 것이다. 그리고 모델 정보와 구현 코드 간의 상호 변환이 가능하게 됨으로써 모델과 코드 간의 일치성이 유지되어 소프트웨어 유지보수 비용을 줄일 수 있을 것이다.

참고 문헌

[1] Object Management Group(Hrsg): XML Metadata Interchange(XMI) Specification version 2.0, Object Managmenet Group, 2003, OMG Document formal/2003-05-02

[2] Eclipse Java Development Tools (JDT), http://www.eclipse.org/jdt/overview.php#JDT_CORE

[3] MoDisco, http://www.eclipse.org/gmt/modisco/doc/MoDisco_Overview_1.0.pdf

[4] Frank Budinsky, David Steinberg, Ed Merks, Ray Ellersick, Timothy J. Grose, Eclipse Modeling Framework(EMF), Addison Wesley Professional, 2004