Java 코드 분석기법을 이용한 UML 클래스 다이어그램 생성 방법

한우희*, 김경수, 김현수
충남대학교 정보통신공학부 컴퓨터공학 전공
(mhhan, kskim, hskim}@ce.cnu.ac.kr

Building a UML class diagram using Java code analysis techniques

Moo-Hee Han*, Kyoung-soo Kim, Hyeon-soo Kim
Dept. of Computer Engineering, Chungnam National University

요 약
본 연구에서는 자바 코드로부터 UML 클래스 다이어그램을 추출하는 역공학방법을 제시하였다. 파서를 이용하여 자바 코드로부터 AST를 생성하고 이를 순회하면서 클래스다이어그램 생성에 필요한 정보를 추출하였다. 이를 위해 구조정보와 관계정보를 정의하였는데, 구조정보에서는 클래스를 묶는 정보를 표현하였다. 관계정보에서는 클래스들 간의 연관관계를 결정하기 위해 필요한 정보를 표현하였다. 이에 여러 관계정보를 통해 연관관계를 유추하는 방법을 제시하였다. 즉 클래스들간의 연관관계를 추출하기 위한 규칙들을 정의하고, 이를 통해 얻어진 관계정보를 이용하여 연관관계를 유추하는 과정을 설명하였다.

1. 서론
1980년대 중반 이후 소프트웨어 개발과정에 도입된 객체지향 개
념은 1990년대 중반 Java 언어와 UML 이란 개체지향 모델링 언어가
등장하면서 개발자들로부터 엄청난 관심을 받으면서 널리 사용되게
되었다. 이를 계기로 기존의 설계중심의 개발방법과 객체지향 중심 개
발형태로 상당수 변화되었으며, 설계중심 개발방법의 단점인 소프트
웨어에 대한 수정 및 확장의 어려움을 극복할 수 있으면서 기대를 하
였다.
그러나 소프트웨어의 기능 확장이나 변경작업은 여전히 소스코드
에 대한 이해를 필요로 하지만, 많은 경우 도큐먼트와의 통일성에 의
해 이는 쉽게 되지 않았다. 이를 극복하기 위한 방법으로 등장한 것이 순환
공학이다. 이는 앱데이터를 통해 기존의 레거시 코드로부터 소프트웨어
시스템의 구조를 얻어내고 이에 대해 수정을 재공학과정을 통해 소스
코드에 반영하는 것이다.

본 연구에서는 자바 소스코드로부터 UML 클래스 다이어그램을 추
출하는 역공학 방법을 제시하고자 한다. 파서를 이용하여 자바 소스
코드로부터 AST(abstract syntax tree)를 얻어낼 수 있고, 이를 순회
하면 UML 다이어그램을 표현하기 위해 필요한 정보를 추출해낼 수
있다. 이에 필요한 정보들이 무엇인지 결정할 필요가 있는데, 자바가
클래스를 기본단위로 하는 객체지향 프로그래밍 언어이기 때문에 시스
템의 정적 정보를 표현하는 클래스 다이어그램을 그리는 것을 가
장 적합한 단계로 보였다.

 특히 클래스 다이어그램의 요소들 중 가장 유사하기 어려운 것이 클래스들 간의 연관관계를 결정하는 일인데 이는 자바 코드로부터 명
시적으로 표현되지 않기 때문에 이를 추출하기 위한 방법들에 대해
이 논문의 초점을 두었다.

2. 선행연구
Fujaba 프로젝트에서는 기존의 그래프 모델을 확장한 story
diagram이라는 메타모델 형태의 그래프 제작언어를 정의하여
UML을 지원하는 순환공학 CASE 도구를 구현하였다[1]. 여기서 클
래스 다이어그램을 추출하는 방법은 파서를 통해 얻어진 AST로부터
클래스와 클래스 메서드들을 찾아낸다. 메소드 목록을 분석하는 과정에
서 to-many관계를 얻기 위해 JFC 라이브러리를 사용하는 컨테이
너 클래스에서 바로 정의된 클래스들만을 활용하거나, 양방향 연관관계
를 얻기 위해서 annotation engine이라는 코드 클리네트를 사용한다.
 이를 위해서 fuzzy reasoning 기술을 적용하여 유사한 패턴에 매칭시
켜 앱데이터를 구현한다. 그러나 자바에서 컨테이너 클래스는 포함된 개
체들의 타입정보를 얻지 못하기 때문에 클래스에 대한 접근 메소드의
정보를 사용자가 미리 마련을 한다.

두 번째는 그래프에 적용한 방법이다[2]. 클래스, 인터페이스, 메
소드로 노드임항 요소로 사각하고, 이들간의 확장, 구현, 상속, 소유,
호출관계 등을 노드간의 에지로 간주하는 그래프 형태로 소스코드 정
보를 표현하고 이들간의 관계를 포함하여 클래이 다이어그램에 필요
한 정보들을 추출하였다. 패업타임의 어드러피트나, 백터 또는 해이
t을 같은 어드러피트를 통해 다중성 정보를 얻어낸다. 각각의 조
합을 통해 호출관계 같은 기본적인 패턴을 정의하고 연관관계에서 집
합연산을 자동적으로 분류할 수 있다. 그러나 소스코드가 결실 경유
그래프가 담긴 정보가 지나치게 컷릴 수 있어서 정보를 추출하고 관
리하는데 시스템의 부가량이 많이 걸린다는 단점이 있다.

세 번째 방법인 데이터베이스를 위한 ODMG 표준을 매
3. 클래스 다이어그램 생성을 위한 규칙

이 장에서 클래스 다이어그램을 그리기 위해 필요한 정보들은 클래스 문제를 구성하는 정보들과 클래스들의 관계를 표현하는 정보들로 구성된다. 그러나 개념, 명세에서 구현할 수 있는 정보들은 실제 코드에서 정보를 얻기 위한 규칙들이 정한다.

3.1 클래스 다이어그램을 표기하기 위한 정보

UML 클래스 다이어그램을 표기하기 위한 정보들은 클래스 문제를 구성하는 정보들과 클래스들의 관계를 표현하는 정보들로 구성된다. 그러나 개념, 명세에서 구현할 수 있는 정보들은 실제 코드에서 정보를 얻기 위한 규칙들이 정한다.

3.2 관계관련: 클래스 다이어그램과 소스 코드 관점

UML 클래스 다이어그램에서 관계(association)는 클래스의 인스턴스 간의 관계를 보여주는 것으로서 각 관계는 두 개의 연관 관(association end)이 존재한다. 각 연관에는 역할 이름(ROLE name)과 다중성(multiplicity)을 표기할 수 있다. 다중성이 참여관계의 핵심이 되는 관점의 분류에 따라 관계의 유형이 다름이다. 또한 화살표를 통해 운동방향(navigability)을 표기할 수 있다. 관계의 유형은 관계의 방향성에 따라 다음과 같이 나눌 수 있다:  
- 공집합: 연관 관계가 두 클래스 간에만 존재한다.  
- 공합집합: 연관 관계가 두 클래스 모두를 포함한다.  
- 공합집합: 연관 관계는 두 클래스 중 하나를 포함한다.  
- 공합집합: 연관 관계는 두 클래스 모두를 포함한다.  

3.3 소스코드로부터 연관관계 추출

단순 클래스의 객체를 참조하는 관계는 일반적으로 호출관계를 통해 얻을 수 있으므로, 선언문(2)에서는 호출관계를 몇 가지 문제에 따라 보다 세부적으로 다음의 값이 분류하였다.

- local: 메소드 내에서 선언된 객체 호출  
- result: 메소드 결과값 참조  
- static: 정적 메소드 호출  

이 방법에서는 연관관계 생성에 도움이 되지 않는 관계들은 제거하였는데 이는 UML의 코드를 참조하는 관계모델(recursive association) 관계를 포함하기 위해서 이 방법도 사용되어야 하며, static관계 역시 사용자 정의된 관계로 제거되는 경우가 된다. 비록 클래스 다이어그램이 시스템의 정적인 구조를 표현하기 위한 것임에도 불구하고, 클래스간 연관관계는 실제로 해당 타입의 객체가 동일하게 표현되기 때문에, 단순히 호출관계만 고려하기 보다는 코드 내에서 실제로 객체가 생성되거나 참조 또는 호출하기 위해서는 정보를 모두 포함하여 연관관계를 유지하는 것이 더 바람직하다.

이런 관점에서 연관관계를 유지하기 위한 규칙을 제정하는 데 다음과 같다. 다음에 열거하는 각각의 경우에는 연관관계가 있음을 전부한다.

1. 클래스A의 어트리뷰트나 혹은 메소드 내에서 사용자 정의된 클래스 타입의 객체를 참조하는 관계 A에서 B 객체의 메소드 호출을 포함하는 경우  
   `public class Course {...}  
   public setCourseName(Course c) {...}  
   public class Teacher {...}  
   public void createCourse(String cName) {  
     Course c = new Course();  
     c.setCourseName(cName);  
   }`  

   그림 1. 클래스 객체의 생성 및 메소드 호출

2. 클래스B에 속한 메소드의 아규먼트로 클래스B 타입의 객체

물 전달하는 경우
예를 들어 그림 2에서 Student 클래스와 애구먼트로 넘겨지는 Course 클래스와 관계가 있음을 유추 할 수 있다.

```java
public class Test {
    public static void main(String args[]) {
        Teacher t = new Teacher("Harry");
        Course c = l.createCourse("Math", 10);
        Student s = new Student(t);
        s.referee(c);
    }
}
```

그림 2. 메소드의 애구먼트로 계체 전달

3. 전역적으로 참조가 가능하도록 선언된 클래스의 메소드나, 어트리뷰트를 직접 참조할 경우
예를 들어 그림 3에서 Student 클래스에서 Course 클래스 타입의 객체를 선언하지 않고 hours 메소드를 직접 참조한다. 역시 두 클래스간에 어떠한 관계가 있음을 유추 할 수 있다.

```java
class Student {
    int hours;
    public int calcHours() {
        this.hours = Course.addHours();
    }
}
```

그림 3. 전역 메소드 참조

4. 클래스 다이어그램 생성
4.1 구조정보 및 관계정보의 정의
클래스 다이어그램에 표현되는 정보들은 클래스 이름, 메소드 정보, 필드 정보 등 클래스 자체에 대한 정보와, 상속, 인터페이스의 구현, 위존, 연관관계 등 클래스간의 관계를 표현하는 정보들로 구분할 수 있다. 클래스 자체에 대한 정보는 ‘구조정보’, 클래스들간 관계들에 대한 정보를 ‘관계정보’라고 정의한다.

구조정보는
- 클래스: 이름
- 생성자: 이름, 어트리뷰트
- 메소드: 가시성(visibility), 이름, 어트리뷰트, 리턴타입
- 어트리뷰트: 가시성, 타입, 이름, 초기값

물론 생성자는 사용자에 의해 정의된 것을 말한다. 이 정보들은 AST를 순회하면서 바로 얻을 수 있다.

관계정보는
- 상속 / 구현관계
- 연관관계: 연관(association), 집합관련(aggregateation)

들이다.

4.2 클래스 다이어그램 생성
자바 코드로부터 클래스 다이어그램을 그리기 위한 정보를 추출하는 방법은 다음과 같다.
1. 구조정보를 얻기 위해 AST를 순회하면서 해당정보를 저장한다.
   - 이름: 클래스이름
   - 동작: 생성자, 메소드 정보
   - 속성: 어트리뷰트 정보
2. 관계정보 중에서 상속 및 인터페이스 구현관계를 찾는다.

‘extends’란 예약어 다음에 나오는 클래스로부터 상속한 것이며 인터페이스 구현은 ‘implements’란 예약어 다음에 나오는 인터페이스로부터 구현 된 것임으로, 해당 예약어를 검색하고 있으며 그 후에 나오는 클래스와 인터페이스 이름을 저장한다.

3. 관계정보 중에서 유추할 수 있는 정보를 추출한다.
   - 메소드 내부를 순회하는 도중 다른 클래스 타입의 객체를 생성, 참조(호출), 수정, 제거하는 작업이 발견될 경우, 해당이 되는 객체 이름과 이들의 타입이 되는 클래스 이름, 이 객체에 대해 메소드가 수행한 작업을 메소드 이름과 함께 탐을 형태로 저장한다. 예를 들어 Student 클래스를 순회하는 동안 두 메소드에서 Course 클래스를 각 차 IDb하더라도 그림과 같이 표기할 수 있으며, 이를 통해 두 클래스가 연관관계가 있을음을 유추 할 수 있다.

```
참조된 객체 이름 - C
참조되는 어트리를부 -
참조되는 메소드 addHours getGrade
참조된 클래스 타입 Course Course
```

그림 4. 연관정보 테이블

4. 연관관계 정보를 얻는다.
   모든 메소드에 대한 순회를 마치면 구조정보를 통해 클래스 몇체를 생성하고, 연관정보 테이블을 통해 자신 클래스와 연관이 있는 클래스들을 알 수 있게 된다. 탐색에 저장된 클래스들은 이 클래스와 연관관계가 있는 것들이며, 다중성정보도 한 클래스타입으로 new 연산에 의해 생성되는 객체의 숫자를 통해 파악할 수 있다.

5. 결론
   본 논문에서는 자바 코드로부터 얻은 AST 정보를 순회하여 클래스 다이어그램을 표현하기 위해 필요한 정보를 추출하고 이를 정리하여 다이어그램을 표현하는 과정을 설명하였다.

그러나 단순하게 연관관계가 있는 클래스들로 다중성 정보만 추출할 수 있으므로 여기저기 집합연관관계의 후보가 되는 클래스들을 추출하고 한정연관, 연관 클래스 등 연관관계를 보다 세부적으로 표현할 수 있는 정보추출에 대한 연구를 더 진행해야 하리라 본다.

6. 참고문헌