관점 지향 프로그래밍을 이용한 컴포넌트의 테스트 프로시저 모듈화 방안

허승현 최은만
동국대학교 컴퓨터공학과
seknj@hanmail.net emchoi@dgu.ac.kr

Modularization of Test Procedures using Aspect-Oriented Programming

Seunghyun Heo, Eun Man Choi
Dept. of Computer Engineering, Dongguk university

요 약

소프트웨어의 재사용으로 인한 생산성 향상을 기대하면서, 컴포넌트 기반 개발(Component Based Development)에 관련한 연구가 지속적으로 이루어지고 있으며, 그 중 컴포넌트의 테스트 연구는 컴포넌트를 배포하고, 재사용하기 위해 검증하는데 기여하며 발전해 왔다. BIT(Built-In Test)와 컴포넌트 테스트를 위한 래퍼 클래스에 관한 연구가 대표적이다. 본 논문에서는 테스트 모듈의 산재를 방지하고, 유지보수성과 추적성 개선을 위해 테스트 프로세저를 모듈화하는 방안을 연구하였으며, 이를 위해 관점 지향 프로그래밍 개념을 도입하였다.

1. 서 론

컴포넌트 기반 개발(Component Based Development)은 생산성을 향상시키고, time-to-market을 단축시키기 위한 대안으로써 소프트웨어 재사용 연구 분야 중 가장 활발하게 진행되어 왔다.

컴포넌트는 소스코드가 공개되지 않는 특징 때문에 재사용을 위한 컴포넌트의 선택과 검증이 어렵다. CBD의 활성화를 위해 컴포넌트의 테스트에 관한 연구는 매우 중요하게 진행되어 왔다.

시스템 빌트 전의 컴포넌트의 테스트는 크게 유닛(Unit) 테스트와 통합(Integration) 테스트로 구분된다. 유닛 테스트는 시스템의 컴포넌트가 다른 컴포넌트와 분리된 상태로 내부만을 테스트하는 것이고, 통합 테스트는 시스템을 이루는 컴포넌트들이 시스템 구조 설계 명세에 나타난 것과 같이 작동하는가를 테스트하는 것이다[9].

테스트 모듈을 어떻게 구현할 것인가 하는 것은 컴포넌트 테스트의 유용성과 직결된 문제이며, 대표적인 유닛 테스트로는 기능 모듈과 테스트 모듈을 같은 컴포넌트에 배치시키는 BIT(Built-in Test)와 컴포넌트를 감싸는 형태인 래퍼(wrapper)클래스에 관한 연구가 있다.

그러나 BIT와 테스트 래퍼의 연구는 컴포넌트 배포자 입장에서의 테스트만을 고려한 형태이다. 여러 컴포넌트와 새로 개발된 모듈 간의 상호운용성 또는 호환성을 검사하기 위해서는 통합 테스트가 필수 적이며, 이는 유닛 테스트 모듈과 다르게 적은 수의 객체에 모듈화 되어 구현될 경우 유지보수를 쉽게 하고, 추적성을 향상 시킨다.

이를 위해 본 논문에서 제시하는 것이 애스펙트이다. 관점 지향 프로그래밍은 본래 핵심(core) 관심사와 횡단(cross-cutting) 관심사를 분리하여 구현함으로써 모듈화를 달성하려는 개발 패러다임이다. 2장과 3장에서는 본 연구의 기초가 되는 연구에 대하여 기술하며, 4장에서는 테스트 모듈을 횡단 관심사로 구현함으로써 모듈화를 모색하고, 5장에서는 테스트 프로시저의 모듈화를 실험을 통해 관찰한다.

2. 관점 지향 프로그래밍

관점 지향 프로그래밍(AOP)이란 1996년 Gergor Kiczales가 처음 만든 용어로, 시스템의 기본적이고 대표적인 업무처리 기능을 나타내는 핵심(core) 관심사와 여러 개의 모듈에 걸치는 사용자 인증, 로깅과 같은 횡단(cross-cutting) 관심사를 구분함으로써 시스템의 설계와 구현의 복잡성을 감소시키고 모듈화를 이루려는 패러다임이다. AOP의 개발 프로세스는 다음과 같다[8]. 핵심관심사는 기존의 구조물인 클래스에 구현되고, AOP의 핵심 구조물인 애스펙트(aspect)에는 횡단 관심사와 직조규칙이 구현되며, 이 직조규칙을 통해 클래스와 접합 또는 분리된다.

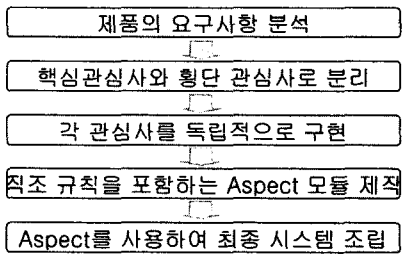

그림 1 AOP 개발 프로세스[7]

AOP는 OOP의 문법요소와 특성 등이 대부분 통용되나

AOP에 횡단관심사를 구현하기 위한 핵심 요소는 다음과 같다

1) 결합점(join point) - 프로그램의 실행 과정에서 구별 가능한 프로그램의 한 지점. 메서드 호출부나 객체의 멤버에 값을 할당하는 부분이 될 수 있다.

2) 교차점(pointcut) - 결합점에 대한 직조 규칙을 형성하기 위해 애스펙트내에 포함시키는 구조물.

3) 충고(advice) - 교차점에서 지정한 결합점에서 실행되어야 할 코드. 결합점 이전에 실행되는 이전(before) 충고와 결합점 이후에 실행되는 이후(after) 충고, 결합점을 대체하여 실행되는 대체(around) 충고가 있다.

3. 테스트 모듈의 구현 기법

3.1. 테스트 내장형(Built-in Test) 컴포넌트

컴포넌트는 소스코드 형태로 배포되지 않으며, 이를 검증하기 위한 테스트 방법도 블랙박스(Black Box) 형태의 테스트로 제한된다[5].

이로 인해 컴포넌트 기반의 소프트웨어 개발은 낮은 테스트 능력(testability)과 CBSE(Component Based Software Engineering) 행위자에 의한 낮은 유지 보수성, 런타임(run-time) 테스팅이 지원되지 않는 제약을 가지고 있다[1].

이를 극복하기 위한 방안이 BIT(Built-in Test)이다. 컴포넌트의 내부에 테스트를 위한 기능들을 내장하여 컴포넌트의 본래 기능을 제공하는 기능적 인터페이스(functional interface) 외에 시험에 필요한 테스트 데이터와 제어의 입출력을 위한 테스트 인터페이스를 추가로 제공하는 BIT 컴포넌트를 제작함으로써 테스트의 제약을 극복하는 것이다[5].

또한 컴포넌트에 내장된 BIT는 소프트웨어의 라이프사이클(life cycle)전반에 걸쳐 재사용될 수 있으며 이를 통해 소프트웨어의 유지보수성을 증가시킬 수 있다[2].

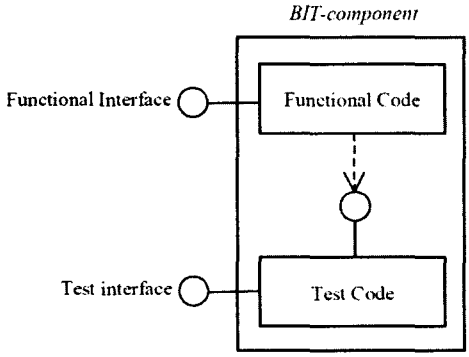

그림 2 컴포넌트의 개념[3]

3.2. 컴포넌트 테스트를 위한 래퍼 (Wrapper)

래퍼란 본래 컴포넌트를 둘러싸고 있는 또 다른 컴포넌트나 클래스로써 래퍼를 통해 기존의 컴포넌트를 직접 수정 없이 재사용 하거나 기능의 변경, 성능의 개선 등을 이룰 수 있다[4]. 소프트웨어 래퍼를 컴포넌트의 테스트에 이용하는 경우, 그림 3과 같은 모양을 하게 된다.

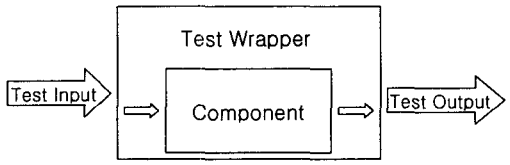

그림 3 컴포넌트 테스트를 위한 래퍼[6]

래퍼는 컴포넌트의 주위를 감싼 형태로써 오직 테스트를 위한 입력과 출력 인터페이스를 가지고 있다. 이러한 래퍼를 통해 본래의 컴포넌트로부터 테스트를 위한 검증 코드를 완전히 분리해 낼 수 있다[6].

4. 테스트 프로시저의 모듈화

컴포넌트의 통합 테스트를 위한 프로시저는 각각의 컴포넌트 또는 그와 연관하는 다른 클래스에 분산되기 마련이다. 이러한 테스트 프로시저의 분산은 테스트 결과의 분석에 어려움을 겪게 하고 유지보수에 필요한 노력을 증가시킨다. 본 연구에서는 테스트 프로시저의 모듈화를 위한 방안으로 관점 지향 프로그래밍(Aspect-oriented Programming)을 도입한다. 각 컴포넌트의 통합 테스트를 수행하기 위한 모듈을 제 3의 구조물에 구현한다.

AOP의 문법 구조 중 대표적인 것이 결합점(join point), 교차점(pointcut), 충고이다. 클래스 내에 정의된 결합점에 대하여, 애스펙트에서 교차점과 충고로써 원하는 처리를 구현한다. 이 구조는 소스코드가 공개되지 않는 컴포넌트의 특성에 잘 어울린다. 테스트하려는 부분을 결합점으로 가정하고, 컴포넌트의 명세에 나와있는 메소드 명, 리턴 타입, 파라메터 등을 이용해 애스펙트에서 테스트 모듈을 구현함으로써 컴포넌트를 테스트하고, 원하는 변경을 가할 수도 있다. 또한 컴포넌트의 속성이 공개된 경우에는 ' privileged' 라는 키워드로 애스펙트에게 특권을 부여함으로써 컴포넌트의 private 요소에 접근할 수 있다.

이와 같이 각 클래스에 산재되어 구현된 테스트 프로시저를 AOP의 횡단 관심사로 구현함으로써 모듈화를 달성하는 것이다. 애스펙트에 구현된 테스트 모듈을 도식화하면 그림 5와 같다.

왼쪽은 실질적 기능에 해당하는 핵심 관심사가 클래스에 구현되어 컴포넌트화 된 모습이고, 오른쪽은 컴포넌트를 테스트하기 위한 모듈을 애스펙트에 구현한 모습이다.

테스트 프로시저를 클래스에 구현할 경우 그림 4와 같이 한 컴포넌트의 클래스를 상속받거나 객체의 인스턴스를 생성해서 메소드를 호출하는 형태로 구현될 것이고, 이와 같은 컴포넌트의 기능을 호출하는 프로시저를 가진 클래스들이 산재되어 구현될 것이다.

테스트 프로시저를 애스펙트에 구현하는 경우 테스트 프로시저는 extends 형태로 컴포넌트와 연관하게 된다. AOP 컴파일러로 직조 후 테스트 모듈은 마치 컴포넌트 안에 구현된 것과 같은 기능을 하게 된다. 또한 컴포넌트의 기능에 추가할 기능이나 메소드의 파라메터 타입을 변경하는 등의 변경도 애스펙트에 구현함으로써 필요한 컴포넌트를 검증하기 위한 다양한 시도가 가능하다.

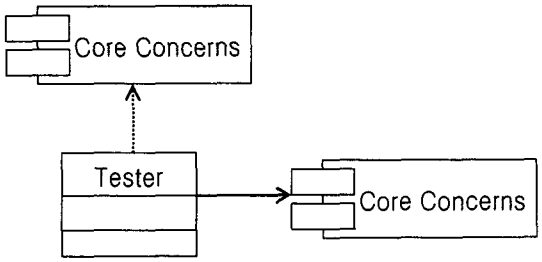

그림 4 클래스에 구현한 테스트 프로시저

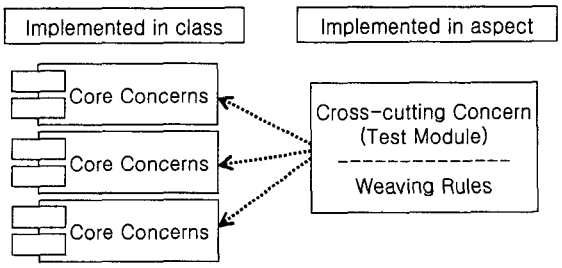

그림 5 애스펙트에 구현된 테스트 프로시저

5. 실험연구

실험 대상에 대하여 테스트 모듈을 클래스에 구현하는 경우와 애스펙트에 구현하는 경우를 비교하였다.

실험 대상은 학생관리, 강의 관리, 수강신청, 성적조회 등의 기능을 하는 각 컴포넌트들로 이루어진 J2EE 플랫폼 기반의 학사정보시스템이다. 테스트 프로시저를 포함하는 애스펙트는 Java의 확장인 AspectJ로 구현하였다.

그림 6은 애스펙트에 구현한 테스트 프로시져의 일부이다.

코드의 상단에서는 강의 등록 메소드를 테스트하고 있으며, 매개변수의 타입을 사용자가 수정한 새로운 클래의 타입으로 변경하여 테스트하고 있다. 다음은 학생 정보가 올바르게 얻어지는지 테스트하기 위하여 컴포넌트 사용자가 새로 정의한 printStudentInformation()으로 컴포넌트를 테스트하는 코드이다.

프로그램에 사용된 4개의 컴포넌트를 테스트하기 위한 프로시저를 구현하는데 클래스에 구현하는 경우 3개의 클래스가 필요했으며, 애스펙트에 구현하는 경우 하나의 애스펙트에 모두 구현 가능하였다.

6. 결론

CBD는 소프트웨어의 재사용을 목적으로 매우 활발하게 연구가 진행되어 왔으며, 실무로의 적용 또한 실효를

```
public aspect IntegrationTester {
        ...
        //**LectureManager**//
        pointcut registCourse(Course course):
        call(public                    Course
        UIS.lectureManager.addCourse(Course))
        &&args(course);
        void around(Course course)
        :registCourse(course){
                proceed(new NewCourse());
        }
        //**StudentManager**//
                pointcut getStudent():call(public
        Student
        StudentManager.Student.getStudent(..));
        after():
        getStudent{st.printStudentInformation();
        ...
        }}
```

그림 6 애스팩트에 구현된 테스트 코드의 일부

거두고 있다. 본 논문에서는 산재되어 분포하는 컴포넌트 테스트 프로시저를 적은 수의 객체에 구현하고, 어느 정도의 변경을 컴포넌트에 가할 수 있게 하기 위해 관점 지향 프로그래밍(AOP)를 도입하였다. 작은 실험으로 인해 방법은 고안하였지만 명확한 결과를 위해 더 많은 실험 케이스가 필요할 것이며, AOP의 활용 역시 앞으로 연구해 나아갈 방향이다.

참고 문헌

[1]Hakan Edler, " BIT in software component," EC IST 5th Framework

[2]Yingxu Wang, " A Method for Built-in Tests in Component-based Software Maintenance," Centre for Software Engineering.

[3]J Vincent, " Built-In-Test Vade Mecum - Part1, A Common BIT Archtecture," Southampton Institute/Bournemouth University, UK

[4]A. Bertolino, A. Polini, " A Framework for Component Deployment Testing," Proc. ACM/IEEE 25th International Conference on Software Engineering ICSE 2003

[5]송호진, 최은만, " 테스트 기능 내장 컴포넌트의 설계와 구현," 2003

[6]송호진, 최은만, " 컴포넌트 테스트를 위한 래퍼의 자동 생성에 관한 연구," 정보과학회논문지: 소프트웨어 및 응용 제 32권 제8호, 2005

[7]허승현, 최은만, " 관점 지향 프로그래밍의 문법 요소를 적용한 프로덕트 라인의 가변 기능 조합," 한국 소프트웨어공학 학술대회 제 8권 1호, 2006

[8]Ramnivas Laddad, " AspetJ in Action: Practical Aspect-Oriented Programming, Manning Publications," 2003

[9]최은만, " 객체 지향 소프트웨어 공학," 사이텍 미디어, 2005