AOP를 이용하여 진화된 프로그램의 회귀 테스트 기법

이미진, 최은만
동국대학교 컴퓨터공학과
e-mail : alexshel@naver.com, emchoi@dgu.ac.kr

A Regression Test Method for Program Evolving by Using AOP

Mee-Jin Lee, Eun Man Choi
Dept. of Computer Engineering, Dongguk University

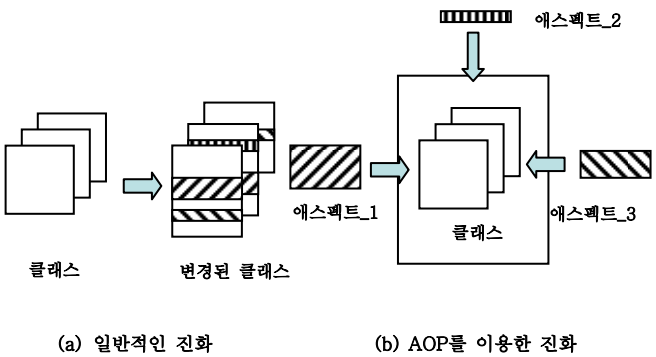
요 약

이미 개발된 소프트웨어를 확장, 진화시킬 때 AOP 기법을 이용하면 모듈화뿐만 아니라 동적으로 플러그인 시킬 수 있어 편리하다. 즉 객체 지향 방식으로 개발된 프로그램에 AOP 를 적용하여 확장하면 여러 모듈에 걸쳐 나타나는 횡단 관심사를 기존 프로그램의 수정 없이 기능을 추가할 수 있다. 이미 테스트까지 마친 소프트웨어에 AOP 를 적용하여 확장한 경우 AOP 특성에 맞는 회귀 테스트 방법이 필요하다. 본 논문에서는 AOP 를 이용하여 진화된 프로그램의 회귀 테스트 방법을 제안하였으며 사례 연구에 의하여 그 효용성을 보였다.

1. 서론

Aspect Oriented Programming(AOP)는 횡단 관심사를 분리하여 표현하는 애스펙트 개념을 제공함으로써 프로그램이 모듈화되고 변경을 로컬화할 수 있다는 장점을 가지고 있다. 또한 AOP 는 별도로 정의된 관심사들이 필요에 의하여 언제나 동적으로 플러그인 될 수 있다는 웨이빙 메커니즘을 가지고 있다. 이러한 장점으로 인하여 프로그램의 특정 관심, 예를 들면 보안과 같은 관점의 테스트를 애스펙트로 정의하여 동적으로 테스트 하는데 응용되기도 하고[1], 일반적인 모듈을 단위 테스트하는 데 사용되기도 하고[2], 컴포넌트의 빌트인 테스트에도 사용된다[3].

한편 AOP 의 웨이빙 메커니즘을 잘 이용하면 레거시 시스템을 손대지 않고 확장하거나 진화시킬 수 있다. 그림 1 에 나타낸 것처럼 순수한 확장이나 진화는 소프트웨어를 구성하는 각 요소 안을 수정하거나 추가하지만 AOP 는 레거시 부분을 손대지 않고 독립된 애스펙트를 정의하여 웨이빙에 의하여 시스템을 구성할 수 있다.



(그림 1) AOP 를 이용한 진화

AOP 는 횡단 관심사를 독립적으로 처리할 수 있는 방법을 제시해 준다. 그림 1 에서와 같이 AOP 는 이미 개발된 프로그램에 새로운 기능을 추가하기 위해 여러 모듈의 프로그램을 수정하지 않아도 되는 이점을 가지고 있다[6]. 레거시 시스템과 별도로 독립적으로 개발할 수 있고 별도의 모듈로 정의할 수 있는 애스펙트는 동적으로도 웨이빙이 가능하여 최근에는 실행중인 시스템의 확장이나 테스트에 대한 연구 사례가 발표되고 있다[13].

그러면 AOP 를 이용하여 확장한 프로그램은 테스트를 어떻게 하여야 하는가? 이것이 이 논문에서 다루고 있는 주제이다. 그림 1 의 (a)와 같은 일반적인 확장은 실행 경로를 분석하여 추가 또는 변경된 부분이 있는 곳은 다시 새로운 테스트 케이스를 만들어 시험하고 또한 변경이 없더라도 영향 받는 부분을 찾아 이를 다시 테스트 한다. 반면 (b)와 같은 애스팩트 형태로 확장될 때 과연 레거시 시스템의 어떤 부분을 다시 테스트 하여야 하는가 이것이 의문이다.

즉 본 논문에서는 AOP 를 이용해 객체 지향 방식의 프로그램을 확장할 경우의 회귀 테스트 방법에 대해 논한다. AspectJ 를 이용한 실험을 통해 제안한 회귀 테스트 방법의 적합성을 보이고 그 결과를 평가하였다.

2. 배경지식

2.1 회귀 테스트

유지보수 단계에서는 오류의 발견, 환경의 변화, 기능의 추가 등 여러 가지 변경요인에 의하여 원래의 프로그램에 수정을 가하고 이를 테스트하여야 한다. 이를 회귀 테스트라 하는데, 회귀 테스트는 두 가지 요소가 있다. 첫 번째는 오류를 발견하고 고치고 다시

원래 문제를 일으켰던 것을 테스트해 보는 것이다. 두 번째 요소는 수정된 부분이 다른 부분에 영향을 주지 않는다는 것을 확인하는 것이다.

본 논문에서 말하는 회귀 테스트의 의미는 후자의 것으로, 이러한 의미의 테스트는 수정 작업이 성공적으로 이루어졌는지 확인하는 것이 아니라 수정 후 프로그램의 통합이 제대로 이루어졌는지 확인하는 작업이다. 왜냐하면 AOP 로 확장하는 경우 확장하기 위하여 추가한 애스팩트 자체 프로그램의 테스트는 그렇게 큰 의미를 가지지 않기 때문이다. 확장을 위하여 추가된 애스펙트들은 원래의 레거시 시스템과 웨이빙 될 것이며 그 후에는 영향받는 부분과 인터랙션하여 그 결과가 나타나기 때문이다.

2.2 Aspect J

AspectJ 는 Java 언어에 Aspect Oriented 개념을 확장한 것이다. AspectJ 의 핵심적인 개념은 결합점이다. 결합점은 메소드의 호출이나 변수에 값을 할당하는 것과 같이 시스템에서 구별이 가능한 실행 지점으로 행단 행위를 삽입하는 장소이다. 교차점에서 선택할 수 있는 결합점을 노출된 결합점이라 한다. 충고는 교차점에서 지정한 결합점에서 실행되어야 할 코드이다. 이를 통해, AspectJ 는 노출된 결합점에서 충고를 통한 프로그래밍의 삽입으로 핵심 관심사 뿐만 아니라 횡단 관심사의 모듈화를 가능하게 해 준다[7, 9, 10]

2.3 AspectJ 결함 모델

결함 모델은 프로그램이 실행되는 동안 대부분의 결함을 찾기에 충분하도록 테스트쉬트를 디자인하기 위한 방법을 가지고 있다. AspectJ 는 결합점에 교차점과 충고로 프로그램이 확장된다. 이러한 AspectJ 의 특성으로 인해 새로운 결함이 생성된다.

AspectJ 의 특성으로 인해 생기는 결함의 종류를 알지 못하는 상황에선 테스터가 좋은 테스트 전략을 고안하는 기초가 부족하고, 적절한 테스트의 범위를 알 수 없게 된다. 결함 모델은 프로그래머로 하여금 결함을 어떻게 피할지 이야기해 주고 결함을 피하지 못할 경우 결함을 쉽게 발견하고 찾아주는 방법을 제시해 준다. 결함 모델은 또한 AspectJ 테스트 연구의 기초를 제공해 준다.

AspectJ 의 경우 부정확한 교차점 강도나 부정확한 Aspect 우선순위, 변화하지 않아야 하는 상태 보존의 불이행, 제어 흐름의 부정확한 변화 등이 결함 모델의 일부 카테고리가 될 수 있다[4, 5].

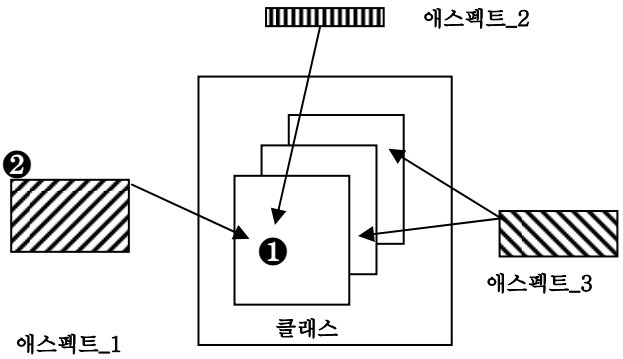
2.4 AspectJ 인스트루멘트

인스트루멘트의 사전적인 의미는 기계, 도구란 의미로 AspectJ 인스트루멘트는 원래의 레거시 프로그램에 특정한 목적으로 삽입하는 코드를 말한다. 본 논문에선 테스트를 위해 AspectJ 인스트루멘트를 사용할 것이다. 이를 이용하여 구현한 프로그램에 수정을 가하지 않고도 로깅이나 추적 등의 기능을 이용하여 테스트를 하고, 테스트를 마친 후 AspectJ 인스트루멘트만 제거하여 원래의 프로그램을 수행할 수 있도록 구현하였다.

3. AOP 확장을 위한 회귀 테스트 기법

AOP 의 여러 특징 중 하나는 기존의 객체지향 언어로 개발된 소프트웨어에 추가하여 사용할 수 있다는 것이다. 이미 개발된 소프트웨어에 여러 모듈을 횡단하여 나타나는 관심사를 객체지향 언어로 개발하기 위해선 어려 모듈에 걸쳐 프로그램을 추가하고 수정해야 하는 어려움이 있지만, AOP 를 적용하면 기존 코드의 변경 없이 기능의 추가가 가능하다.

테스트가 끝난 소프트웨어에 AOP 를 이용하여 확장한 경우, 확장된 소프트웨어의 테스트를 위한 회귀 테스트 기법이 필요하다. 이 때, 회귀테스트는 다른 언어와 차별된 AOP 특성에 맞도록 계획되어야 한다.



(그림 2) AOP 테스트의 관찰 요소

AOP 의 가장 핵심적인 개념은 그림 2 의 1 로 표시된 결합점이다. 노출된 결합점에 충고를 이용해 프로그래밍의 삽입이 가능하게 되는 것이다. 따라서 AOP 를 이용해 확장한 프로그램의 테스트를 위해 결합점을 유심히 관찰해야 할 필요가 있다.

결합점에서 중요하게 관찰해야 할 첫 번째 부분은 결합점의 선택이다. 올바른 결합점을 선택하여 프로그램을 적용하는 것이 중요하다. 올바른 결합점의 선택을 위해 레거시 시스템의 구조를 정확히 파악할 필요가 있다. 클래스 다이어그램이나 클래스 흐름 그래프 등을 이용해 시스템 확장을 위해 웨이빙 되어야 할 부위를 체크해 볼 수 있을 것이다. 원하는 결합점이 제대로 포착되고 있는지 확인하기 위해 AspectJ 인스트루멘트를 작성해 확인해 볼 수 있다. 결합점이 호출될 때 마다

두 번째 부분은 웨이빙 된 후 프로그램 실행 순서가 올바르게 진행되고 있는가를 살펴 보아야 한다. 웨이빙 된 후 프로그램의 실행 순서를 확인하기 위해 우선 프로그램의 실행 순서를 설계해 볼 필요가 있다. 애스펙트 웨이빙 프로그램의 실행 순서를 작성하기 위한 방법 중 하나로 애스펙트 흐름 그래프(aspect flow graph, AFG)를 적용해 볼 수 있다. 이는 ASSM(aspect scope state model)에 충고와 메서드 데이터

흐름 그래프를 결합한 것이다.[12]

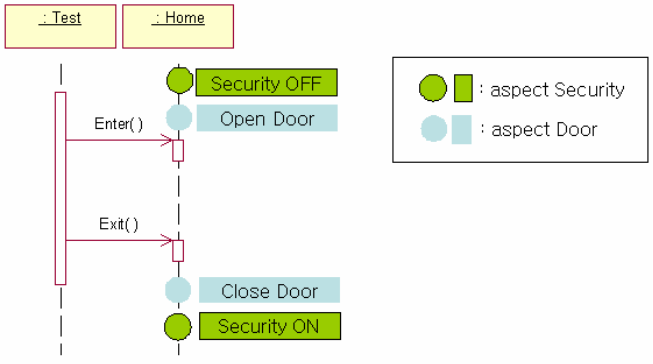
세 번째는 보존되어야 할 정보 혹은 상태가 제대로 유지되고 있어야 한다는 점이다. 간혹 바뀌지 않아야 할 정적인 정보들을 변화시킨 경우 프로그램의 논리적 에러를 부를 수 있다. 따라서 추적을 통해 변화되는 정보들을 확인할 수 있어야 한다.

앞서 살펴본 바와 같이 AOP 를 이용해 확장된 프로그램이 제대로 된 기능을 수행하고 있는지 살펴보기 위해 인스트루멘테이션에 의한 테스트를 적용하였다. 현재 확장되기 전의 테스트를 마친 프로그램의 기능을 실행시 로깅을 남기는 AOP 인스트루멘트를 만들어 실행을 추적한 결과를 놓고, AOP 확장 후 동일한 기능에 대해 동일한 로깅을 남기는지 확인할 수 있다. 또한 확장된 기능에 대해 인스트루멘트를 만들어 정확한 결과를 내는지 확인할 뿐만 아니라 로깅을 확인하여 원하는 바 대로 실행 되는지 확인할 수 있다. 이때 인스트루멘트는 확장된 AOP 의 결합점을 중심으로 환경변수의 수집을 이용해 프로그램의 실행 순서 및 변수의 추적 등을 이용하여 원하는 대로 프로그램이 확장되었는지, 잘못 되었다면 어느 부분에서 문제가 있는지 한눈에 알아볼 수 있도록 테스트를 계획할 수 있다.

4. 제안한 방법의 회귀 테스트 실험

4.1 객체 지향 프로그래밍 개발과 테스트

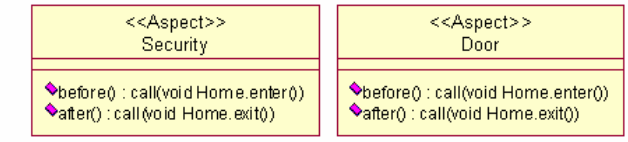
논문을 위하여 간단한 객체 지향 프로그래밍을 설계하였다. 그림 2 와 같이 사람이 들어오기 전에 보안을 끈 후 문을 열어주고, 사람이 나간 후 문을 닫고 보안을 설정해 주는 프로그램을 제작하였다. 우선 객체 지향 프로그래밍을 이용하여 Home 클래스에 사람이 들어오고 나가는 함수가 있는 프로그램을 제작하였다.



(그림 2) 실험 프로그램의 순서 다이어그램**[11]**

4.2 객체 지향 소프트웨어에 AOP 를 적용한 기능의 추가

그림 3 에서 보는 바와 같이 AspectJ 를 이용하여 보안 기능과 문의 열고 닫힘 기능을 추가하였다. 이를 이용하여 사람이 나간 후에 문이 닫히고 보안 기능이 설정되고, 사람이 들어오기 전, 보안 기능이 해제 되고 문이 열리도록 하였다.



(그림 3) Security 와 Door 애스펙트 프로그램

4.3 AOP 적용한 소프트웨어의 회귀테스트

테스트 결과 AOP 순서가 바뀌어 실행됨을 확인할 수 있었다. 이때 환경 정보를 이용하여 실행되는 결합점들을 포착하여 어느 부분에서 에러가 수행되는지 쉽게 확인할 수 있다. 표 1 은 에러를 확인하기 위한 AspectInstrument 애스펙트의 소스코드 부분이다.

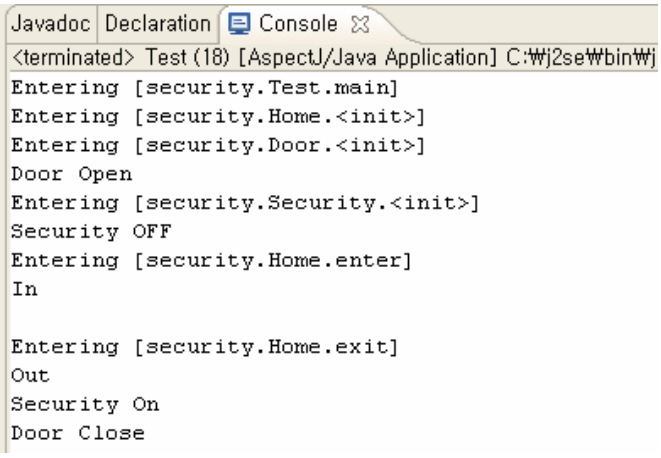
<표 1> AspectInstrument 소스 코드

```
public aspect AspectInstrument {
        pointcut traceMethods() :
execution(* *.*(..))||
execution(*.new(..))
&& !within(AspectInstrument);

        before():traceMethods(){
                Signature sig =
thisJoinPointStaticPart.getSignature();

        System.out.println("Entering
["+sig.getDeclaringType().getName() +
"." +sig.getName() + "]");
        }
}
```

4.4 실험 평가



(그림 4) AspectInstrument 실행 결과

4.3 의 간략한 AspectInstrument 애스펙트를 이용하여 AOP 로 확장된 프로그램의 동적으로 변하는 환경변수를 수집하여 어떤 부분에서 프로그램의 결함이 생성되었는지 쉽게 확인할 수 있다. 그림 4 에서 보는 바와 같이 포착되는 결합점의 이름을 통해 수행 순서를 확인하여 어떤 부분에서 수행이 잘못되었나 확인할 수 있었다.

5. 결론

빠르게 변화하는 사용자의 요구에 맞추기 위해 사용자가 원하는 기능을 빠른 시간에 추가하여 제품을 출시하기 위해, AOP 는 기능 추가 시에 기존의 소프트웨어를 수정하지 않고 가능하다는 큰 이점을 가지고 있다. 이렇게 AOP 를 이용해 확장한 경우의 회귀 테스트를 위해선 AOP 의 특징을 잘 이해하고 그에 맞는 회귀 테스트 방식을 적용해야 한다.

본 논문에선 AOP 에 맞는 회귀 테스트 방법을 AspectJ 를 이용한 실험을 통해 제시하였다. 현재 실험은 AOP 프로그램의 특징 중 하나인 우선순위를 잘 따져야 한다는 부분에 맞도록 회귀테스트를 적용하여 동적으로 실행되는 결합점을 환경변수로 수집하여 테스트를 진행하였다. 그러나 현재의 실험은 AOP 의 결함을 충분히 찾아내기에 규모나 내용면에서 많이 부족하다. 차후 실험에선 AOP 우선순위 분만 아니라 교차점에서 올바른 결합점을 취하고 있는지 테스트 하기 위한 인스트루멘트를 추가할 것이다. 또한 AOP 결함을 찾아내기 충분한 실험환경을 만들고, 그에 맞게 Aspect 인스트루멘트 부분을 확장하여 일반적인 AOP 프로그램의 결함을 찾기에 적합하도록 구현하여야 할 것이다. AOP 우선순위를 테스트 하기 위한 인스트루멘트 기법에 그래픽 적인 요소를 가미하면 확장된 프로그램의 흐름을 그래프로 만들어 설계한 부분과 비교해 볼 수 있을 것이다. Aspect 인스트루멘트 부분을 확장하여 AOP 뿐만 아니라 기타 프로그램의 특징에 맞는 Aspect 인스트루멘트 개발 방법을 구상해 보는 것도 하나의 연구 과제가 될 수 있다.

참고문헌

[1] N. Belblida, et. al, “AOP extension for security testing of programs”, *Proc. of IEEE Canadian Conference on Electrical and Computer Engineering*, Ottawa, May 2006, pp.647-650.

[2] G. Xu, Z. Yang, “A novel approach to unit testing: The Aspect-Oriented way”, International Symposium on Future Software Technology 2004, Xian China, October 2004, pp.

[3] J. Bruel, et al., “Using aspects to develop built-in tests for components”, *4th Modeling with UML workshop of 2003 UML conference*, San Francisco, Oct. 2003, pp.

[4] Jon S.Bekken, Roger T. Alexander. “A Candidate Fault Model for AspectJ Pointcuts” *17th International Symposium on Software Reliability Engineering (ISSRE’06)*, 2006, pp.

[5] R. T. Alexander, J. M. Bieman, and A. A. Andrews. “Towards the systematic testing of aspect-oriented programs” *Technical Report CS-4-105, Department of Computer Science*, Colorado State University, March 2004.

[6] Nadia Belblidia, Mourad Debbabi, Aiman Hanna and Zhenrong Yang. “AOP extension for security testing of programs” *Proc. of IEEE CCECE/CCGEI*, Ottawa, May 2006, pp.

[7] 허승현, 최은만 “관점 지향 프로그래밍을 이용한 컴포넌트 테스트 프로시저 모듈화 방안” *Proc. of Korea Computer Conference*, 정보과학회, 용평리조트, 2006, pp.

[8] 허승현, 최은만 “관점 지향 프로그래밍의 문법요소를 적용한 프로덕트 라인의 가변 기능 조합” *석사학위 논문*, 동국대학교, 2006.

[9] 이미진, 최은만 “AOP 를 이용한 테스트 관점의 분리 방법” *한국정보과학회 소프트웨어공학회지*, 제 19 권, 제 2 호, 2006, pp.

[10] Ramnivas Laddad, *AspectJ in Action: Practical Aspect-Oriented Programming*, Manning Publications, 2003

[11] 최은만, *객체 지향 소프트웨어 공학*, 사이텍미디어, 2005.

[12] Weifeng Xu, Dianxian Xu, Vivek Goel and Ken Nygard “ASPECT FLOW GRAPH FOR TESTING ASPECT-ORIENTED PROGRAMS”, laboratory research

[13] H. Song, Y. Yin, S. Zheng, “Dynamic aspects weaving in service composition”, *Proc. of the 6th International Conference on Intelligent Systems Design and Applications(ISDA’06)*, IEEE, 2006, pp..