# AOP를 위한 동적 결합 메커니즘

김태웅°, 김경민, 김태공
인제대학교 전산학과
{twkim°,kmkim, ktg}@cs.inje.ac.kr

# Dynamic Weaving Mechanism for Aspect-Oriented Programming

Taewoong Kim°, Kyungmin Kim, Taegong Kim
Dept of Computer Science, Inje University

요 약

영역지향 프로그래밍(Aspect-Oriented Programming)은 소프트웨어의 성능을 향상시키고 유지보수에 많은 이점을 가지는 새로운 프로그래밍 방법론이다. 하지만 영역지향 프로그래밍 방법으로 소프트웨어를 개발하기 위해서는 Aspect를 지원하는 새로운 영역지향 프로그래밍 언어를 사용하거나 레거시 시스템에 Aspect를 적용할 경우에 소스코드의 재 컴파일등과 같은 문제점을 가지고 있다. 이에 본 논문에서는 레거시 시스템에 Aspect를 동적으로 결합할 수 있는 동적결합 메커니즘을 제안한다. 이를 위하여 Aspect의 행위와 결합 정보를 가지는 Connector를 생성하고, 생성된 Connector는 코어클래스의 위임자로서 역할을 수행하게 되는 동적결합 가능하게 하는 메커니즘을 제안한다.

## 1. 서 론

소프트웨어를 개발할 때 견고한 디자인을 바탕으로 개발된 소프트웨어는 모듈화가 잘 되어 있다. 잘 정의된 모듈화는 소프트웨어의 성능을 향상시키게 되고 차후 유지보수에도 많은 이점을 준다. 하지만 여러 가지 이유로 인하여 모든 소프트웨어들이 좋은 모듈화를 유지할 수 있는 것은 아니다.

객체지향 프로그래밍에서는 클래스 혹은 오퍼레이션(Operation)이라는 단 하나의 모듈화 방법만을 제시하고 있기 때문에 여러 클래스나 오퍼레이션에 걸쳐서 구현되어야 하는 보안이나 결함내성과 같은 부가기능들의 모듈화에는 실패하였다[1]. 클래스는 클래스 자체의 정의뿐만 아니라 함께 연동하는 다른 클래스들과 관련된 행위를 혼합된 형태(tangling)로 가지고 있다. 때문에 구성하는 클래스들이 다른 프로그램 구성요소로 사용될 경우나, 지속적인 요구사항의 변경으로 유지보수가 빈번하게 일어나게 될 때 소프트웨어는 복잡성과 같은 문제를 야기 시킬 수 있다. 특히, 스케줄링, 로깅, 문맥의존형 에러처리, 트랜잭션, 보안과 같은 기능들은 여러 다른 모듈들에 걸쳐져서(scattering) 구현이 되어야 하기 때문에 한 파일이나 한 클래스 등으로 모듈화 될 수 없으며, 이 기능들을 구현한 코드들은 흩어져서 존재할 수밖에 없다. 이처럼 여러 클래스에 유사한 기능이 중복되어 정의되어야 하거나 클래스 구조를 이용하여 모듈화를 제공할 수 없는 부가기능의 단위를 크로스커팅 단위(cross-cutting concern)라고 하며, 객체지향프로그래밍의 이 같은 단점을 보완하고자 고안된 것이 영역지향프로그래밍(Aspect-Oriented Programming) [2,3]이다. 이는 소프트웨어를 개발하는데 있어서 필요한 부분만을 임의로 분할하고 이를 바탕으로 모듈화를 통하여 시스템을 개발하는 새로운 개념의 기술이다[4]. 그러나 AOP를 기반으로 소프트웨어를 개발하거나 Aspect를 지원하지 않았던 레거시 시스템일 경우 Aspect를 지원하는 새로운 언어를 사용하여 기존의 소프트웨어 소스코드와 Aspect에 관련된 소스코드를 통합하여 재 컴파일하거나 기존 시스템의 수정, 새로운 컴파일러의 사용등과 같은 문제점을 가지고 있다. 이와 같이 개발된 Aspect는 그 Aspect를 재사용하지 못하는 단점과 기존의 소프트웨어와 결합되어 복잡성을 야기시키는 또 다른 단점을 가지게 된다.

본 논문에서는 레거시 시스템과 Aspect를 동적으로 결합하기 위한 동적결합 메커니즘을 제안하고자 한다. 이를 위하여 Aspect의 결합정보를 나타내는 결합점(pointcut)과 행위(behavior)정보인 크로스커팅단위로 분리하고, 코어(core)정보, 크로스커팅단위정보, 결합점정보로부터 동적으로 결합되어질 수 있는 Connector를 설계하고 이를 동적으로 결합가능하게 하는 메커니즘을 제안하고자 한다.

2장에서는 관련연구로서 AOP언어의 특성과 그 종류에 대해 설명하고 3장에서 본 연구에서 제안한 동적결합을 위한 Connector와 동적결합 메커니즘에 대해 제안한다. 마지막 4장에서는 향후 연구과제를 포함한 결론을 맺는다.

## 2. 관련연구

### 2.1 AOP

Aspect는 기존의 프로그램 모듈이 추상화 및 인캡슐레이션 개념을 제공할 수 없는 이른바 여러 다른 기능들에 걸쳐져서 구현되는 기능으로 크로스커팅단위를 의미한다. 즉, 크로스커팅단위는 여러 클래스에 유사한 기능이 중복되어 정의되어야 하거나 전통적 클래스 구조를 이용하여 모듈화를 제공할 수 없는 부가기능이나 특성(feature)들을 의미하는 것으로 여러 모듈에서 필요한 부분만을 추출하여 새로운 모듈로 구성하게 된다[5].

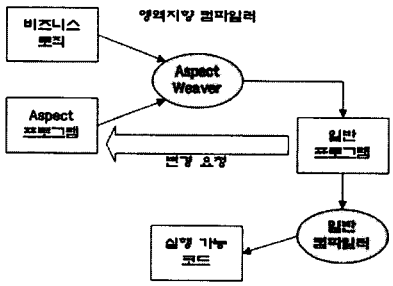


그림 1 영역지향 프로그래밍의 기본 구조 및 절차

그림 1은 영역지향프로그래밍 언어의 처리 방법이 기본적으로 취하고 있는 구조 및 절차를 나타내고 있다. 모듈은 기존의 기능성을 모듈화하는 핵심 비즈니스 로직인 코어와 크로스커팅 단위를 모듈화하는 Aspect 모듈로 분리되어 구성된다. 이것은 다시 영역지향 언어 컴파일러에 의해 두 가지 행위가 결합된 형태의 일반 프로그램으로 번역 된다.

### 2.2 AOP Languages

AspectJ[6]

현재 Aspect를 지원하는 프로그래밍 언어들 중 가장 많이 사용하고 있다. 코어시스템과 Aspect가 결합되기 위해 결합점을 정의해야 하며 결합점은 상당히 복잡한 구조를 가지고 있어 AspectJ[7]를 기반으로 개발하기 위해서는 결합점과 advice에 대한 이해가 불가피 하다. 코어시스템의 소스코드와 크로스커팅단위에 해당하는 행위 코드, 여기에 결합점을 나타내는 소스코드, 이 세 가지 소스코드를 통합하여 컴파일 하여 새로운 애플리케이션을 생성하는 구조를 가진다. 따라서 이미 개발된 시스템의 유지보수보다는 Aspect를 기반으로 하는 새로운 시스템을 개발할 때 적합한 언어로 각광받고 있다. 또한 Aspect를 advice(행위, behavior)와 결합점을 하나의 클래스로 구현하고 있어 Aspect의 재사용은 거의 불가능하다.

JAsCo[8]

컴포넌트 기반의 개발방법론에 접근한 새로운 패러다임의 Aspect를 지원하는 언어이다. Aspect의 기능적인 측면을 담당하는 행위 부분을 hook라는 키워드로 설정하고 결합점에 해당하는 부분을 connect라는 키워드를 사용하여 표현한다. Aspect의 행위 부분과 결합점부분을 분리하여 결합점을 동적으로 로드할 수 있게 함으로써 Aspect의 재사용성은 이루어질 수 있으나 AspectJ과 마찬가지로 결합점을 정의하여 결합정보를 표현하고 있어 결합점과 advice에 대한 이해가 필수적이다. 또한 컴포넌트 기반으로 코어시스템과 Aspect의 결합을 제안하고 있어 컴포넌트 기반으로 개발된 시스템에 종속적이며 이미 개발된 레거시 시스템에 적용하고자 할 경우 코어시스템을 수정하여야 하는 단점을 가지고 있다.

JAC[9]

Apsect-Oriented Framework로서 동적으로 Aspect를 로드하고 크로스커팅단위를 표현하기 위해 새로운 언어의 사용을 하지 않는다는 장점을 가지고 있으며 기본적으로 JAsCO[10]의 추상적인 모델이다. 따라서 컴포넌트 기반으로 코어와 Aspect의 결합을 제안하고 있으며 이미 개발된 레거시 시스템에 Aspect를 지원하기 위해서는 기존의 컴포넌트를 변경해야 하는 작업을 가져야 한다.

PROSE[11]

자바언어를 위한 라이브러리 형태로 Aspect를 지원하고 있으며 이벤트를 중간에서 가로채는 방식의 라이브러리로 구성되어 있다. 이러한 라이브러리를 사용하여 Aspect를 동적으로 결합(weaving)하고 분리(unweaving)하는 메커니즘을 제공한다. 그러나 기존의 자바 가상머신을 사용하지 않고 디버깅 모드에서 실행될 수 있는 자바 가상 머신을 사용하여야 하는 단점을 가지고 있다.

## 3. 동적결합 메커니즘

본 연구에서는 레거시 시스템에 Aspect를 동적으로 결합하기위한 Connector를 제안하고 이를 동적으로 결합가능하게 하는 메커니즘에 대해 제안하고자 한다. Connector의 생성은 Aspect를 Aspect의 행위인 크로스커팅단위와 결합정보인 결합점으로 분리한다. 크로스커팅단위는 비즈니스로직을 포함하고 있는 코어부분과 같은 일반적인 클래스 형태로 나타내고, 결합점 부분은 본 논문에서 제안한 XML스키마에 따라 표현한다. 결합점은 Aspect가 코어에 결합되는 위치정보를 가져야 하므로 코어에 있는 해당 객체의 오퍼레이션 정보가 필요하다. 따라서 코어와 Aspect 클래스로부터 공개 오퍼레이션을 가지는 인터페이스 정보를 추출하여 XML로 나타낸다. 이러한 XML로부터 코어와 Aspect를 동적으로 결합할 수 있는 Connector를 생성 한다.

### 3.1 Connector의 생성

Core와 Aspect를 결합가능하게 하는 Connector를 그림 2와 같은 방법으로 생성한다. 이를 위하여 Aspect를 Aspect의 행위인 크로스커팅단위와 결합정보인 결합점으로 분리한다. 크로스커팅단위는 비즈니스로직을 포함하고 있는 코어부분과 같은 일반적인 클래스 형태로 나타내고, 결합점 부분은 본 논문에서 제안한 XML스키마에 따라 표현한다. 결합점은 Aspect가 코어에 결합되는 위치정보를 가져야 하므로 코어에 있는 해당 객체의 오퍼레이션 정보가 필요하다.

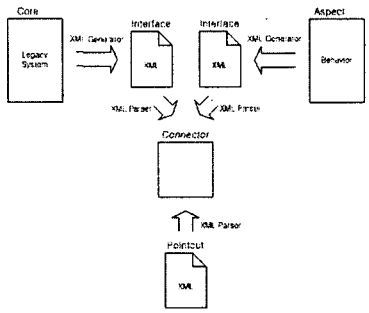


그림 2 Connector 생성구조

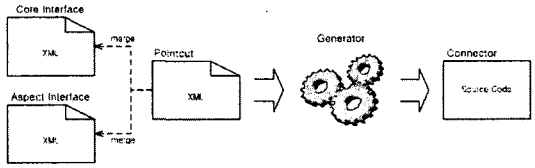


그림 3 Generator에 의한 Connector의 생성

### 3.2 Connector의 동적결합

그림 3에서 생성된 Connector클래스는 코어클래스의 오퍼레이션 정보와 Aspect가 가지는 오퍼레이션 정보 그리고 두 클래스간의 결합점 정보를 포함하고 있다. 이는 기존의 레거시 시스템에서 사용하던 코어클래스에 대한 위임(delegation)역할을 수행할 수 있다는 의미로 해석할 수 있다. 따라서 기존의 레거시 시스템에서 사용하는 코어클래스에 대한 참조가 *Connector*클래스로 이동되며, *Connector*클래스는 코어클래스와 Aspect의 오퍼레이션과 결합점을 이용하여 위임자로서의 역할을 수행하게 된다.

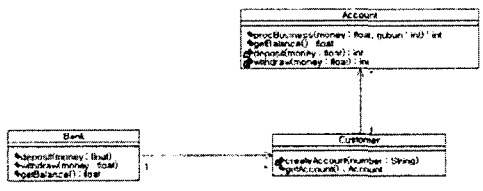


그림 4 레거시 시스템을 표현한 클래스 다이어그램

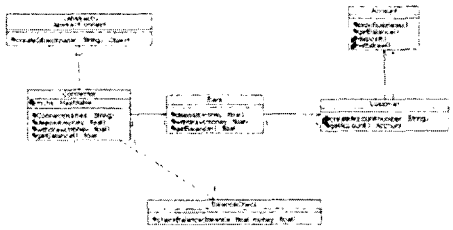

그림 5 Connector를 적용한 후의 클래스 다이어그램

그림 5는 그림 4와 같은 은행계좌 시스템(레거시 시스템)을 본 논문에서 제안한 Connector를 사용하여 변경된 시스템을 나타내고 있다. 레거시 시스템에서 사용하던 Bank, Account, Customer클래스는 수정사항이 발생하지 않고 Connector클래스가 추가되어 있다. Connector클래스의 기능은 Bank클래스와 BalanceCheck클래스 사이에서 이 두 클래스에 존재하는 오퍼레이션의 호출을 중간에서 가로채어 그 순서와 시점을 제어한다. Connector 클래스는 Bank클래스와 BalanceCheck클래스의 단방향연관을 위한 정보를 가지고 있는 AbstractConnector 클래스를 상속하고 Bank클래스가 가지는 모든 오퍼레이션에 대한 정보를 가지고 있다. 이는 레거시 시스템을 수정하지 않고도 Connector에 의해 동적으로 Aspect와 결합될 수 있음을 의미한다. 즉, Bank클래스는 Connector클래스의 존재를 알 필요 없고, Connector클래스만이 Bank클래스의 존재를 인식하는 단방향 연관으로 되어있어 레거시 시스템의 수정없이 Connector를 적용할 수 있다.

표 1에서 기존의 방법론과 본 논문에서 제안한 Connector를 사용한 경우를 항목별로 비교하였다.

표 1 각 방법론에 대한 비교

| 특징 | 기존의 방법론 | 동적결합 방법론(제안) |
|---|---|---|
| Recompile | 예 | 아니오 |
| Aspect 재사용성 | 낮음 | 높음 |
| 레거시 시스템의 수정 | 예 | 아니오 |
| 응답속도 | 빠름 | 다소느림 |
| 유지보수성 | 낮음 | 높음 |

## 4. 결론 및 향후연구

소프트웨어의 성능을 향상시키고 유지보수에 많은 이점을 가지는 새로운 소프트웨어 개발 방법론인 영역지향 소프트웨어 개발방법론(Aspect-Oriented Software Development)은 기존의 프로그래밍 언어가 제공하지 못하는 보안이나 결함 내성과 같은 부가기능에 대해 모듈화하는 방법을 제공하고 있다. 그러나 이러한 개발방법론을 사용하여 레거시 시스템의 수정과 같은 유지보수를 가능하게 하기 위해서는 AOP 언어를 사용하여야 하며 레거시 시스템 소스 코드와의 재컴파일등과 같은 문제점을 가지고 있다.

본 논문에서는 이러한 문제점을 해결하기 위하여 새로운 프로그래밍언어의 사용이나 기존 시스템 소스코드와의 재 컴파일이 필요 없는 동적 결합 가능한 Connector를 제안하고 이를 동적으로 결합가능하게 하는 메커니즘에 대해 제안하였다. 레거시 시스템과 Aspect를 동적으로 결합하는 Connector를 생성하기 위하여 Aspect의 기능부분인 크로스커팅단위부분과 결합정보를 나타내는 결합점부분을 분리하고 크로스커팅단위부분은 일반적인 클래스 형태로 나타내며 결합점부분은 XML을 사용하여 정보를 표현하였다. 이 XML정보는 코어클래스와 Aspect클래스로부터 추출한 인터페이스 정보와 함께 Connector를 생성하는데 사용되었다. 생성된 Connector클래스는 코어클래스의 위임자로서 그 역할을 수행하게 된다.

본 연구를 통하여 AOP를 기반으로 기존의 레거시 시스템에 Aspect를 추가하고자 할 경우 새로운 AOP 언어를 사용하지 않고도 Aspect를 적용할 수 있었다. 또한 코어와 Aspect를 결합하는 결합점정보를 Aspect의 기능을 나타내는 크로스커팅단위와 분리하여 나타내고 결합점정보만을 수정함으로써 Aspect를 재사용 할 수 있는 장점도 가진다.

현재 본 논문에서 제안한 위임자 역할을 수행하는 Connector클래스로의 실시간 참조이동을 위한 미들웨어에 대한 연구가 진행중에 있으며, 코어와 Aspect의 결합정보를 나타내는 결합점정보를 보다 쉽게 추출할 수 있도록 UML의 순차도 다이어그램으로 모델링하는 방법에 대한 연구가 진행중에 있다. 이러한 연구들은 향후 AOP기반의 시스템개발을 지원하는 CASE 툴 개발에 유용하게 사용될 것으로 기대한다.

## 참고문헌


1. http://www-106.ibm.com/developerworks/java/library/j-aspectj/index.html
2. G. Kiczales and et al, "Aspect-oriented programming", in proceedings of European Conference for Object-Oriented Programming, LNCS Vol. 1241, pp.220-243, 1997
3. 이준상, "미래소프트웨어 개발 기술:Aspect-Orinted-Programming과 Subject-Orient-ed-Programming", 정보처리학회지, VOL.10 No.5 pp.94-101, 2003
4. http://www.aosd.net
5. http://aspectj.org/doc/dist/progguide/index.html
6. AspectJ Website, http://www.aspectj.org
7. Yoshiki Sato, Shigeru Chiba, "A Selective, Just-In-Time Aspect Weaver", GPCE, LNCS Vol.2830, pp.189-208, 2003
8. Davy Suvee, "JAsCO:an Aspect-Oriented approach tailored for Component Based Software Development", AOSD Conference, pp.21-29, 2003
9. R. Pawlak, L. Seinturier, L. Duchien, and G. Florin. Jac: A Flexible solution for Aspect-Oriented Programming in Java. In Proceedings of Reflection, LNCS, Vol. 2192, pp.1-21, 2001
10. Adnrei Popovici, Thomas Gross, Gustavo Alonso, "Dynamic Weaving for Aspect-Oriented Programming", AOSD Conference, pp.141-147, 2002
11. Popovici, A., Gross, T. and Alonso, G., "Dynamic Weaving for Aspect-Oriented Programming", In Proceedings ofthe 1st international conference on Aspect-Ori-ented Software Development, pp.141-147, 2002