제25회 한국정보처리학회 춘계학술발표대회 논문집 제13권 제1호 (2006. 5)

AOP 코드 이해를 지원하는 애스펙트 클래스 참조 테이블(ACRT)

박옥자*, 박종각*, 유철중*, 장옥배*, 신현철**
*전북대학교 컴퓨터정보학과
**백석문화대학 컴퓨터정보학부
e-mail:ojpark@bsc.ac.kr

The Aspect Class Reference Table on AOP Code Understanding

Oak-Cha Park*, Jong-Kack Park*, Cheol-Jung Yoo*, Ok-Bae Chang*, Hyeun-Chul Shin**
*Dept of Computer Information, Chonbuk National University
**Division of Computer Engineering, Baeksok College of Cultural Studies

요 약

AOP 기법의 가장 큰 장점은 관심사(concern)를 분리하여 모듈화하는데 있다. 모듈화는 클래스간의 결합도를 낮게 유지하면서 프로그램의 수정 및 확장을 용이하게 하므로 프로그램의 재사용 및 유지보수성을 높인다. 하지만, AOP에서 낮은 결합도를 유지하도록 작성된 클래스들은 서로간의 호출 및 연관 관계가 직접적으로 발생하지 않고 AOP에서 지원된 직조 과정에서 발생하게 되므로 일반 클래스와 에스펙트 클래스간의 참조 관계를 이해하기 어렵다. 따라서 시스템의 흐름을 파악할 수 있는 클래스 참조 방법론 제시가 필요하다. 본 논문에서는 AOP 클래스간의 참조 관계를 이해할 수 있는 애스펙트 클래스 참조 테이블(ACRT)을 템플릿으로 제시하였고, AOP 특징에 맞는 클래스를 유형별로 분류하여 클래스간의 참조 관계를 간단한 표기법으로 나타냈다.

1. 서론

AOP(Aspect-Oriented Programming)는 일반적으로 OOD(Object-Oriented Development)나 CBD(Component-Based Development)에서 해결하지 못한 횡단 관심사(crosscutting concern)의 문제점을 보완함으로써 관심사의 캡슐화라는 본래의 목적을 충분히 달성하도록 지원한다[1]. AOP는 소프트웨어 시스템을 상호 독립적인 관심사의 조합으로 간주하여 소프트웨어 시스템을 제작하는 새로운 방법론으로 대두되었다. 이는 객체지향 프로그래밍 기법의 개념을 기반으로 하고 횡단 관심사 처리를 위해 방법만 확장한 것이다. 비즈니스 로직을 구현하는 핵심 관심사(core concern)는 기존의 방법대로 클래스로 구현하고 횡단 관심사는 AOP로 구현하는 것이다. 핵심 관심사와 횡단 관심사는 서로 독립적으로 구현하지만 핵심 관심사는 횡단 관심사의 영향을 전혀 모르는 채 수행되므로 클래스간의 결합도는 줄어들면서 독립적으로 확장가능하다. 한편, AOP가 가진 독립적이면서 낮은 결합도를 유지하게 하는 고모듈화는 오히려 프로그램의 흐름을 파악하기 어렵게 하는 단점을 가져오기도 한다. AOP에서는 각각 모듈화하여 애스펙트로 구현한 관심사를 직조과정(weaving)을 통해 실행하기 때문에 코드만으로는 각 클래스 참조 관계를 정확히 이해하기 어렵다[2].

본 논문에서는 AOP 클래스간의 참조 관계를 이해할 수 있는 애스펙트 클래스 참조 테이블(Aspect Class Reference Table)을 템플릿으로 제시하였다. 테이블은 핵심 클래스 관점과 애스펙트로 구현한 관심사 관점으로 분류하여 작성하여 상호 보완할 수 있도록 제시하였다. 또한, AOP 특징에 적합한 클래스를 유형별로 분류하여 클래스간의 참조관계를 쉽게 이해하도록 간단한 표기법을 제시하였다.

논문의 구성은 다음과 같다. 2장에서는 AOP 기법과 AspectJ에 대해 살펴보고 3장에서는 ACRT에 대해 기술하였고 4장에서는 사례에 적용시켜 ACRT의 특징을 설명하였다. 마지막으로 5장에서는 결론 및 향후연구를 살펴보기로 한다.

2. AOP와 AspectJ

2.1 AOP

AOP 개념은 원래 Xerox PARC에서 Gregor Kiczales와 그의 팀이 도입한 방법으로 소프트웨어 설계 및 구현에서 관심사 분리 방법을 보다 향상시킨 기법이다[1]. AOP는 여러 모듈에 산재되어 있는 횡단 관심사를 추출하여 별도의 모듈로 구현한 후 그 안에 모듈 합성 규칙을 지정한다.

이 후 직조(Weaving) 단계에서 핵심 모듈과 결합하는 방법으로 최종 시스템을 구축하는 것이다. AOP를 실제로 수행하려면 관심사 분리, 관심사 구현, 직조의 세 단계 명세 방법이 필요하다.

2.1.1 관심사 분리(Separation of Concern)

관심사는 전체적인 소프트웨어 시스템의 목적을 처리해야 하는 구체적인 요구사항이나 고려사항을 의미한다. 관심사는 핵심 관심사(core concern)와 횡단 관심사(cross-cutting concern)로 분류한다. 핵심 관심사는 각 모듈에서 수행하는 기본적이고 대표적인 업무처리 기능을 취급하고 횡단 관심사는 여러 개의 모듈에 걸치는 시스템 전체적인 부가적인 요구사항을 다룬다.

2.1.2 관심사 구현

일정한 범주로 나눈 관심사를 독립적으로 구현한다. 핵심 관심사는 일반적인 프로그래밍 방법으로 구현하고 횡단 관심사는 AOP 기법을 이용하여 구현한다.

2.1.3 직조(weaving)

2.1에서 구현된 애스펙트 모듈에 이미 정의된 합성 규칙을 지정하여 직조 또는 통합하는 과정을 거쳐 최종 시스템으로 조립하는 것이다. 본 논문에서는 Java를 이용하여 핵심 로직을 구현하고 그 외 관심사는 AspectJ 구현하였다.

2.2 AspectJ

AspectJ는 Java 언어에 AOP 개념을 추가하여 확장한 범용의 언어이다. AspectJ는 Java의 확장이므로 모든 Java 프로그램은 AspectJ로 확장 가능하다. Java로 핵심 비즈니스 로직을 구현한 후 프로그램 확장시 추가적인 관심사는 애스펙트로 작성한다. 모듈화되어 애스펙트로 구현한 코드는 원시 프로그램을 수정하지 않고 직조과정을 거쳐 통합하게 되므로 프로그램 유지보수 및 확장이 용이하다[3].

3. AOP 클래스 참조 테이블(ACRT)

3.1 필요성

AOP의 큰 장점인 관심사의 분리는 높은 모듈화를 제공하면서 클래스간의 결합도가 낮게 한다. 클래스간의 관계가 메소드 호출관계가 아닌 코드에 어드바이스를 추가하여 직조 단계에서 애스펙트 코드가 삽입되기 때문이다. 따라서, 원시 코드만으로는 상호 클래스간의 참조가 직접적으로 명시되지 않기 때문에 프로그램 로직을 이해하기 어렵게 한다.

```
public class Home
{
    public static void main(String[] args) {
      HomeManagement ho = new HomeManagement();
      ho.homeExit();
      System.out.println();
      ho.homeEnter();
    }
}
```

[Code 1] Home.java

```
public class HomeManagement {
    public void homeEnter() {
            System.out.println("Home Enter");
            //Enter logic
    }

    public void homeExit() {
            System.out.println("Home Exit");
            //Exit logic
    }
}
```

[Code 2] HomeManagement.java

[Code 1]과 [Code 2]는 Java로 Home 원격 관리를 구현한 코드의 일부이다. Home 클래스와 HomeManagement 클래스는 상호 호출관계가 이루어지기 때문에 코드 상으로 충분히 프로그램 로직을 이해할 수 있다.

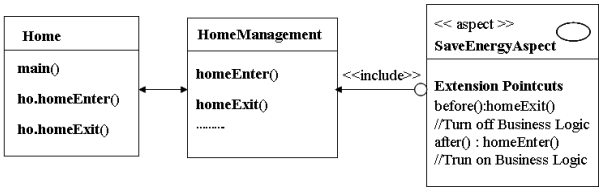
```
public aspect SaveEnergyAspect {
    before() : call(void ho.homeExit()) {
            System.out.println("Turn of the light");
            //Turn on logic
    }

    after() : call(void ho.homeEnter()) {
            System.out.println("Turn off the light");
            //Turn off logic
    }
}
```

[Code 3] SaveEnergyAspect.java

[Code 3]의 SaveEnergyAspect 클래스는 외출시 자동으로 전원을 차단해주는 프로그램을 애스펙트 클래스로 작성하여 Home.java, HomeManagement.java와 함께 ajc 컴파일 과정으로 직조하여 실행시킨다. 외출시 "Turn off the light" 로직을 실행하고 다시 집에 들어오게 되면 자동으로 "Turn on the light" 로직을 실행한다.

SaveEnergyAspect 클래스는 전원 제어를 담당하는 로직이 구현되어 있다고 볼 수 있지만 단독으로 실행 불가능할 뿐만 아니라 어떤 클래스가 호출하여 사용하는지 알 수 없다. 이 코드의 실행 결과는 AOP 프로그래밍에서 마지막 직조 단계에서만 알 수 있다[2].

이 세 개의 클래스 관계를 그림으로 표현하면 다음과 [그림 1]과 같다. 클래스간의 표기법은 [4][5]에서 제시한 방법을 기반으로 표기하였다.



[그림 1] 클래스 참조 관계도

위의 코드와 그림에서 보면 Home, HomeManagement 클래스는 호출자와 피호출자의 관계로 구현 코드에서 참조 관계를 이해할 수 있다. 반면, 어드바이스 코드를 삽입하여 액스팩트로 작성한 SaveEnergyAspect 클래스는 AOP 특성상 직조과정에서 Home, HomeManagement 클래스와 결합되기 때문에 코드만으로 어떤 클래스에 의해 참조되고 호출되는지 이해할 수 없다. 이와 같은 특징은 AOP에서 제시하고 모듈화가 잘 되어 있다는 장점을 나타내기도 하지만 단점으로서 프로그램의 흐름을 어렵게 하는 단점이 되기도 한다.

[4]에서는 이와 같은 이해 관점에서 AOP 클래스간의 참조관계를 나타내기 위해 Open, Class-Directional, Aspect-Directional 그리고 Closed로 분류하였다. 3.2절에서는 위 네 가지 유형별 클래스와 해당하는 표기법을 정리하여 [표 4]에서 보여준다. 네 가지 유형의 관계에서 Class-Directional과 Closed 관계는 AOP 관점에서 가장 지향하는 관계라고 볼 수 있다. 하지만 직조 과정을 거치기 전까지는 프로그램 호출 관계를 이해하기 어렵기 때문에 이를 보안할 수 있는 클래스 참조 테이블이 필요하다.

3.2 애스펙트 클래스 참조 테이블(ACRT)

본 논문에서는 AOP 프로그램의 유지보수 및 이해를 지원할 수 있는 클래스 참조 템플릿을 제시하였다. 클래스 참조 템플릿의 특징은 다음과 같다.

- 클래스를 일반 핵심 클래스(Core Class, CoC), 관심사 클래스(Concern Class, CnC) 두 가지 유형으로 분류하였다[2]. 핵심 클래스는 모든 레벨의 요구사항으로 각 모듈에서 수행해야 하는 기본적이고 대표적인 비즈니스 로직을 코딩한 부분이다. 애스펙트로 구현하는 관심사 클래스는 시스템 레벨에서 추가적으로 수행하게 되는 기능들을 코딩한 부분이다. 관심사 클래스는 [표 1]과 같이 동적 관심사 클래스(Dynamic Concern Class, DCC)와 정적 관심사 클래스(Static Concern Class, SCC)로 다시 분류하였다. 본 논문에서는 제시한 클래스 참조 관계도에서 [그림 1]과 같이 DCC는 일반 타원으로 SCC는 회색칠의 타원형으로 표기하였다.

Class Type		Description	Compile		Etc
핵심 클래스 (CoC)		· 프로그램의 비즈니스 로직만 구축 · 핵심 클래스와 관심사 클래스간의 중개자 역할을 수행	javac ajc	○ ○	일반 비즈니스 로직을 구현
관심사 클래스 (CnC)	동적 관심사 클래스 (DCC)	· 새로운 행위를 프로그램에 직조 · 핵심 프로그램이 실행될 때 여러 개의 모듈에 걸쳐 수행 · 추가되어 새로운 코드로 실행 또는 기존의 코드를 새로운 코드로 치환되어 실행	javac ajc	X ○	대부분의 애스펙트 코드에 해당
	정적 관심사 클래스 (SCC)	· 동적 관심사 클래스를 지원 · 시스템의 실제 행위는 변경하지 않음	javac ajc	X ○	우선순위 경고, 정책 오류 선언등

[표 1] ACRT에서의 클래스 유형

- 일반 비즈니스 로직을 구현한 핵심 클래스 관점에 핵심 클래스 참조 템플릿과 애스펙트 코드로 구현한 관심사 클래스 참조 템플릿 두 가지로 분류하였다[표 2][표 3]. 두 개의 템플릿은 상호 참조된다.

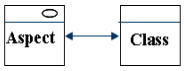
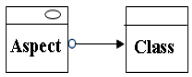
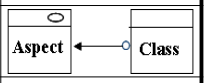
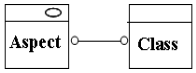
Core Class	Description of Class	Aspect	Aspect Class and Type
클래스명	핵심 클래스 기술	관심사 1	애스펙트 클래스(유형)
		관심사 2	애스펙트 클래스(유형)
		· · ·	· · ·

[표 2] ACRT Core Class 템플릿

Aspect	Description Aspect	Category	Associated Core Class	Static Aspect
관심사 1	관심사 기술	관심사 유형	클래스 이름 · · ·	관련된 SCC

[표 3] ACRT Aspect Class 템플릿

- AOP로 작성한 클래스간의 참조 관계를 간단한 표기법으로 명시하여 코드나 테이블 만으로도 충분한 클래스 관계를 이해할 수 있도록 제시하였다[표 4].

Relation	Description	Notation
Open	· 클래스와 애스펙트 간의 관계가 명시적으로 이루어짐	
Class-Directional	· 애스펙트 클래스는 핵심 클래스의 존재를 알고 있지만 핵심 클래스는 애스펙트 클래스를 인지하지 않음 · 코드 관점에서 전혀 상호 호출관계 없음 · 직조 과정에서 핵심 클래스에 결합됨 · AspectJ에서 일반적으로 구현되는 클래스 관계	
Aspect-Directional	· Class-Directional과 반대 · 바람직하지 않는 코드	
Open	· 핵심 클래스와 애스펙트 클래스가 서로 모르는 채 실행 · 가장 바람직한 관계이지만 AspectJ에서는 아직 지원되지 않음	

[표 4] ACRT에서의 클래스 관계 형태[4][5]

4. 적용 사례

사례 예제는 3장에서 제시한 예제에 참조하여 몇 가지 관심사를 애스펙트로 작성하여 ACRT 테이블 작성하였다. 실제 행위를 애스펙트로 작성하여 프로그램에 직조하는 동적 관심사 클래스(DCC) SaveEnergy외에 Security, AirCoordinating 애스펙트를 추가하였으며 각 애스펙트의 우선순위를 관리하는 HomeCoordinatingAspect는 정적 관심사 클래스(SCC)이다. 아래 코드 단편은 자동 보안 제어(Security)관심사를 AspectJ로 작성한 코드의 일부이다.

```
public aspect HomeSecurityAspect {
    before() : call(void ho.homeExit()) {
            //Security Lock Business Logic

    }

    after() : call(void ho.homeEnter()) {
            //Security unLock Business Logic
    }
```

[Code 4] HomeSecurityAspect.java

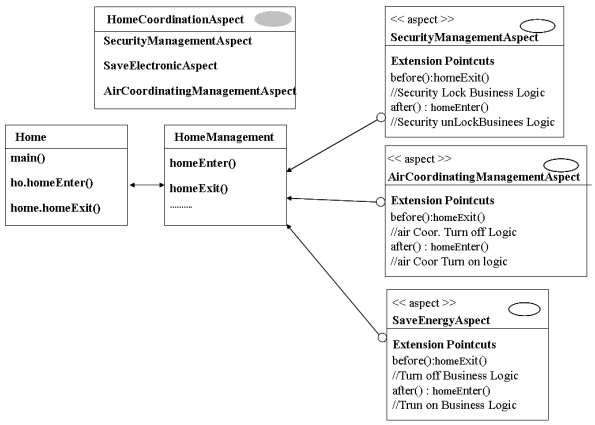
Core Class(.java)	Description of Class	Aspect	Aspect Class and Type
HomeManagement	Home 전체에 대한 원격 제어 관리를 로직	SaveEnergy	SaveElectronicAspect.java(DCC)
		Security	SecurityManagementAspect.java(DCC)
		AirCoordination	AirCoordinationAspect.java(DCC)
		Priority	HomeCoordiantionAspect.java(SCC, Priority)
		· · ·	· · ·
RoomManagement	각 Room에 해당하는 원격 제어 관리 로직	SaveEnergy	SaveElectronicAspect.java(DCC)
· · ·	· · ·	· · ·	· · ·

[표 5] 핵심 클래스 관점의 ACRT

Aspect	Description Aspect	Type	Associated Core Class(.java)	Static Aspect(.java)
Security	보안 경보 On, Off	DCC	HomeManagement	HomeCoordiantionAspect
			· · ·	
SaveEnergy	자동 전원 관리	DCC	HomeManagement	HomeCoordiantionAspect
			RoomManagement	
			· · ·	
AirCoordinating	냉난방 관리	DCC	HomeManagement	HomeCoordiantionAspect
			· · ·	
HomeCoordiantion Aspect	애스펙트 우선순위 관리	SCC	HomeManagement	HomeCoordiantionAspect
			· · ·	

[표 6] 관심사 클래스 관점의 ACRT

HomeSecurityAspect 코드는 결합점 before(), after() 어드바이스 명령을 거쳐 HomeManagement에 접근하도록 명시되어 있지만 HomeManagement 클래스 코드에는 애스펙트 클래스와의 연관 관계를 명시하는 명령이 기술되어 있지 않다. 독립적으로 구현한 관심사를 직조단계에서 결합하기 때문이다.



[그림 2] Home 원격관리의 클래스 참조 관계도

[표 5]와 [표 6]은 사례에서 제시한 코드를 기반으로 기술한 ACRT의 일부이고 [그림 2]는 ACRT 이해를 돕는 클래스 참조 관계도이다. 클래스 관계도에서 Home, HomeManagement 클래스는 일반 Java로 작성한 핵심 클래스이다. <<aspect>>와 타원으로 표시한 세 개의 클래스는 모두 관심사 클래스로서 AspectJ 문법에 따라 작성하였다. 모두 ajc 컴파일 과정에서 직조되어 프로그램이 실행된다. 각 클래스 관계를 표기한 표기법은 [표 4]에서 제시한 방법으로 표기하였다. HomeManagement.java와 세 개의 애스펙트 코드는 Class-Directional 관계로서 HomeManagement는 애스펙트 클래스의 참조관계를 인식하지 못하지만 애스펙트 클래스는 HomeManagement.java 클래스의 참조관계를 인식하고 있다. [그림 2]에서 같은 관심사 클래스이지만 DCC는 일반 타원으로 표기하였고 DCC의 외에 HomeCoordinationAspect와 같은 SCC는 회색 타원으로 표기하여 CoC, DCC, SCC를 명확히 분리하여 이해하도록 표기하였다.

5. 결론 및 향후 연구

AOP의 코드는 모듈화가 잘 되어있어 프로그램 유지보수 및 확장이 용이하지만 유지보수 관점에서 클래스간의 관계를 쉽게 이해할 수 있는 방법이 필요하였다. 본 논문에서는 AOP로 작성한 프로그램의 이해를 돕기 위한 클래스 참조 테이블을 작성하였다. 테이블은 핵심 클래스, 관심사 클래스 두 개의 관점으로 작성하였고 이들 테이블 관계를 보다 쉽게 나타내는 클래스 참조 관계도 표기법을 제시하였다. 향후 연구에서는 프로그램 영역을 보다 확장하여 횡단 관심사와 핵심 관심사의 참조 관계를 나타낼 수 있도록 제시하고자 한다.

참고문헌

[1] G. Kiczales, G., et al. "Aspect Oriented Programming," Lecture Notes in Computer Science. Vol. 1241, 1997. pp. 220-242

[2] Ramnivas Laddad, "AspectJ in Action," Manning, 2005

[3] G. Kiczales et al., "An Overview of AspectJ," ECOOP 2001-Object Oriented Programming, LNCS 2072, Springer-Verlag 2001, pp. 327-353

[4] Mik A. Kersten et al, "Atals : A Case Study in Building a Web-Based Learning Environment using Aspect-oriented Programming," Technical Report Number TR-99-04, 1999

[5] Ivar Jacobson, Pan-Weing, "Aspect-Oriented Software Development with Use Case", Addison Wesley, 2005