AspectJ를 지원하는 AOP 개발 프레임워크

박옥자°, 김정옥, 김재웅*, 유철중, 장옥배

전북대학교 컴퓨터정보학과°, TTA SW시험인증센터*
{ojpark@bsc.ac.kr}

The Aspect Development Architecture for AspectJ

Oak-Cha Park°, Jeong-Oak Kim, Jae-Woong Kim*, Cheol-Jung Yoo, Ok-Bae Chan
Dept of Computer Information, Chonbuk National University°
Telecommunication Technology Association*

요 약

관심사 분리는 소프트웨어 공학에서 핵심 문제로 다루어왔다. 기존의 OOD나 CBD등은 관심사를 분리하여 모듈화함으로써 프로그램 개발 및 유지보수를 용이하도록 발전해왔다. 하지만, 에러 처리나 로깅과 같이 여러 모듈에 산재되어 실행되는 횡단 관심사는 기존의 방법으로 해결하기 어려웠다. AOP는 이와 같은 횡단 관심사를 처리하려는 데 목적을 두고 제안된 방법으로 기존의 OOD나 CBD의 단점을 보완하면서 병행적으로 발전해왔다[1].
AOP가 나타난 가장 큰 특징은 기존의 개발 방법론을 기반으로 핵심 관심사를 개발하고 해결하기 어려운 횡단 관심사는 AOP로 개발하려는데 초기 목적을 두고 있다. 하지만, 대부분의 연구가 초기 요구사항 분석 단계에서 관심사를 명시하는데 초점을 두고 있을 뿐 구현 단계에서 효율적인 접근 방법은 아직 부족한 편이다. 본 논문에서는 Java와 AspectJ를 이용하여 구현한 간단한 사례 연구를 적용한 AOP 개발 프레임워크를 제안한다. AOP 개발 프레임워크에서는 관심사 분리, 구현, 평가의 세 단계를 기술한다. 이 중 구현단계에서는 핵심 관심사와 횡단 관심사 구현에 초점을 두고 AOP 기법에 쉽게 접근할 수 있는 방법을 기술하고 있다. 프레임워크는 프로그램 개발을 보다 용이하게 하고 확장 및 유지보수시 많은 시간을 단축시키려는데 있다.

1. 서론

소프트웨어 설계에서 복잡한 시스템을 단순화하는 최선의 방법은 관심사를 구별하여 모듈화하는 것이다. 기존의 OOD(Object-Oriented Development)나 CBD(Component-Based Development)를 포함한 이전 방법들은 단일 엔티티로 관심사를 분리하고 캡슐화하는데 초점을 맞추었다. 예를 들어 절차, 패키지, 클래스, 그리고 메소드 모두는 프로그래머가 단일 엔티티로 관심사를 캡슐화하도록 도움을 주었다. 이 방법들은 많은 소프트웨어 관심사들을 성공적으로 캡슐화하였지만 일부 관심사는 캡슐화하는데 어려움이 있다. 이는 한 모듈에 여러 관심사를 동시에 구현할 때 발생하는 코드 혼합(code tangling)과 한 개의 관심사가 여러 개의 모듈에 구현될 때 발생하는 코드 산재(code scattering) 때문이다. 이와 같은 비모듈화 현상은 코드를 복잡하게 만들고 유지보수를 어렵게 하였다. 이와 같은 관심사를 [1]에서는 횡단 관심사라 정의하였다. 횡단 관심사의 대표적인 예로는 로깅, 예외처리, 보안, 성능, 메모리 관리, 사용자 인증등이 된다[2].

AOP는 OOD나 CBD에서 해결하지 못한 횡단 관심사(crosscutting concern)의 문제점을 보완함으로써 관심사의 캡슐화라는 본래의 목적을 충분히 달성하려는데 있다[1]. 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 구현하는데 목적을 두고 있다. 이 방법은 OOD, CBD의 문제점에 남아있는 횡단적 관심사까지 모듈화함으로써 높은 품질의 프로그램 구축을 마련하게 하였다.

하지만, AOP의 가장 핵심 문제가 관심사 분리에 초점을 두고 있게 되어 구현 방법에는 다양한 방법이 제시되고 있지 않다. 본 논문에서는 Java와 AspectJ를 이용하여 구현한 간단한 사례 연구를 적용한 AOP 개발 프레임워크를 제안한다. AOP 개발 프레임워크에서는 관심사 분리, 구현, 평가의 세 단계를 기술한다. 이 중 구현단계에서는 핵심 관심사와 횡단 관심사 구현에 초점을 두고 AOP 기법에 쉽게 접근할 수 있는 방법을 기술하고 있다. 개발 프레임워크는 요구사항 단계부터 최종 테스팅 단계까지 이루어지는 절차들을 기술하며 각 단계별 가이드라인을 제시함으로써 프로그램 개발 및 확장성을 높여주려는데 있다.

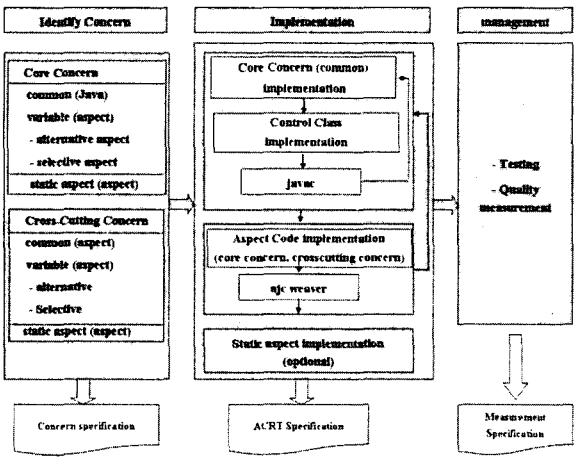
논문의 구성은 다음과 같다. 2장에서는 관련연구를 살펴보고 3장에서는 AOP 개발 아케텍쳐를 제시하여 단계별 가이드라인을 제시하고 4장에서는 적용 사례를 통해 아키텍쳐의 특징을 보여주고 마지막 5장에서는 결론 및 향후연구를 살펴보기로 한다.

2. AOP와 AspectJ

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.

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[2][3].

3. AOP 개발 아키텍쳐



[그림 1] AOP 개발 아키텍쳐

본 논문에서 제안한 개발 아키텍쳐는 [그림 1]과 같다. 프로그램 개발 단계는 관심사 명시, 구현, 프로그램 평가 및 테스팅이다. 본 논문에서는 관심사를 명시와 구현부분에 초점을 두고 기술하고자 한다.

3.1 Identify Concern

3.1.1 Core Concern

관심사 분리는 AOP 프로그래밍 과정에서 핵심적 역할을 수행한다. 관심사 분리 기법은 이미 여러 연구 결과에서 발표되어 왔는데 일반적으로 요구사항 분석 기법을 기본 방법으로 유지하고 횡단적 관심사 추출에 초점을 유지하고 있다. 관심사는 핵심 관심사(core concern)과 횡단 관심사(crosscutting concern) 분류하고 다시 common concern와 variable concern으로 분류하였다. 관심사 분류기법은 이미 제시한 [4][5]기법을 이용하였다.

핵심 관심사는 비즈니스 로직을 구현하는 모듈이다. 프로그램의 가장 기본적인 비즈니스 로직을 구현하며 공통적(common) 기능을 가진 관심사와 가변적(variable) 특성을 가진 관심사로 구별된다. 공통적 기능을 가진 관심사는 프로그램 전체의 로직을 이끌어 가는 모듈로서 Java에서 핵심 역할을 다룬다고 볼 수 있다. 이 모듈은 Java로 구현하고 차후에 분류된 관심사들을 Aspect코드로 구현하여 확장 지원한다. 가변적 관심사는 selective와 alternative로 분류한다. selective는 모듈 중 선택적으로 추가하거나 삭제 가능하다. 반면에 alternative는 여러 개의 유사한 모듈 중 반드시 추가해야 한다.

3.1.2 Cross-cutting concern

횡단 관심사는 여러 개의 모듈에 산재되어 프로그램을 실행시키는 관심사로서 로깅, 보안, 성능, 사용자 인증등을 예로 들 수 있다. 핵심 관심사와 마찬가지로 common, variable로 분류할 수 있는데 common 관심사로는 로깅관리, 프로세스 관리등을 예로 들 수 있다. common 관심사는 가장 우선적으로 프로그램을 구현하여 처리하고 나머지 variable 관심사는 핵심 관심사에서 제시한 방법과 유사하다.

3.2. Implementation

3.2.1 Core Concern implementation

AOP 아키텍쳐의 두 번째 단계로서 명시한 관심사를 프로그램 구현하는 것이다. 이 때 핵심 관심사의 core concen의 common 관심사를 우선적으로 Java로 구현한다. 이 때 Coordinate class는 Java로 작성한 common class와 Aspect class와의 프로그램 로직을 다루는 클래스로서 각각의 클래스와 독립적으로 작성해야 한다. 이 방법을 반복적으로 수행하면서 javac로 컴파일한 프로그램은 핵심 비즈니스 로직을 완성하게 된다.

3.2.2. Aspect Code implementatin

이 단계에서는 이전 단계에서 명시한 관심사 중에서 우선순위가 높은 관심사부터 프로그램을 구현하게 된다. 이 중 core concern와 crosscutting concern은 각각의 우선순위를 가진 관심사가 있는데 이 우선순위는 관심사 명시단계에서 명시해줌으로써 시간을 줄일 수 있다.

3.3 Management

이 단계는 구현된 프로그램을 평가 및 테스팅하는 단계이다. 이 단계는 일반적인 java 프로그램의 품질 평가 방법론이

적용된다. 하지만, AOP 기법을 적용함으로써 발생하는 몇 가지 문제가 있는데 먼저, AspectJ로 작성한 코드는 Java외에 추가되는 어드바이스가 존재한다. 이 어드바이스에 코드 규칙이 달라지므로 코드 품질평가 분야가 추가되어야 한다. 두 번째로 AspectJ로 작성한 코드를 직조과정을 거치게 되면 각각의 파일 크기가 두 배 이상으로 증가함으로써 성능저하를 유발시킨다. 이 성능부분을 평가할 방법이 필요하며 용량을 최소화할 수 있는 코드 작성 방법 또한 필요하다. 이 부분의 연구는 차후 연구 과제에서 다루고자 한다.

4. 적용사례

본 논문에서 제시한 아키텍처에 HomeManagement 시스템을 간단히 적용하였다. HomeManagement 시스템은 일반 가정에서 외출시 전원, 냉난방, 보안 시스템을 자동으로 제어해 주는 프로그램이다. 사례 연구에서는 핵심 관심사 구현 절차만 기술하였다. 구현한 클래스 테이블은 [표 1]과 같다.

[표 1] Core Concern 클래스 테이블

Core concern	class name	type	implementation
Common Concern	Home.java	common	java
	HomeManagement.java	common	java
Variable Concern	SaveEnergy.java	alternative	aspectJ
	HomeSecurityAspect.java	selectvie	aspectJ
	AirCoordination.java	selective	aspectJ
	Priority.java	static aspect	aspectJ

[표 1]에서 보면 핵심 관심사 프로그램 시작과 종료를 담당하는 Home.java와 HomeManagmenet.java를 Java로 구현한다. 간단한 코드 예는 아래와 같다.

```
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

variable concern은 전원관리, 보안, 냉난방 시스템으로 구분되고 SaveEnergy는 selecvite로 나머지는 alternatvie로 명시하였다. variable concern은 모두 AspectJ로 구현하여 직조함으로써 프로그램을 통합한다. 아래 [code 3]은 첫 번째 Aspect 코드를 추가하는 예제로서 전원을 담당하는 관심사 SaveEnergyAspect.java를 구현한 예제이다.

```
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

아래 [code 4]는 보안을 담당하는 관심사로서 Aspect 코드 추가를 반복적으로 수행하면서 프로그램을 확장하는 것이다.

```
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

이와 같은 과정을 반복하면서 프로그램은 확장된다. AOP의 가장 큰 특징은 프로그램 개발이 용이하고 프로그램 유지보수 및 확장이 쉽다. 논문에서 제시한 사례 연구처럼 각각 명시된 관심사는 프로그램 확장이 용이하고 차후 불필요한 모듈을 삭제할 경우에도 핵심 프로그램의 수정을 필요치 않고 지속적으로 확장 가능하다. 프레임워크에서 가장 중요한 특징은 관심사 명시 단계에서 Java로 구현할 핵심 코드는 무엇이고 차후에 추가될 관심사의 우선순위를 어떻게 명시할 것이가이다.

5. 결론 및 향후 연구

본 논문에서는 Java를 이용한 AOP 프로그램 구축시 필요한 AOP 개발 프레임워크를 제안하였다. 프레임워크는 AsepctJ의 특징을 각 단계별 가이드라인과 구현 절차를 기술함으로써 프로그램 개발 및 확장을 용이하게 하였다. 향후 연구에서는 프레임워크를 보다 상세화함으로써 개발 가이드라인을 제시하고 평가 및 테스팅 단계를 보완할 것이다.

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] Isabel Britio, Ana Moreira, "Towards a composition process for aspect-oriented requirements", AOSE 2003

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