관점지향 소프트웨어 개발 패러다임의 소개

성결대학교 금영욱

1. 서 론

기계언어에서 절차 프로그래밍과 객체지향 프로그래밍에 이르기까지 소프트웨어 공학은 여러 과정을 거치면서 발전해 왔다. 지금은 수십 년 전보다 훨씬 고차원적으로 문제를 해결한다. 그러나 현재 소프트웨어가 매우 복잡해져서 소프트웨어 산업은 심각한 위기를 겪고 있다[1]. 가속화되는 하드웨어 능력의 증가에 힘입어 개발자에게 향상된 기능과 새로운 개념을 제공하기 위해 소프트웨어는 복잡한 구조를 갖게 되었다. 이와 함께 소프트웨어 기술의 빠른 혁신(객체 지향, Java, J2EE,.NET, Web Services, SDO[2], SOAP, Grid 등)으로 인해 우리가 통합할 수 있는 능력을 벗어나는 중요한 레거시 애플리케이션, 자료와 프로토콜 등이 생성되었다. 또한 경쟁의 심화로 인해 단기간에 시장에 제품을 출시해야 하는 압력이 커지고 있다.

객체지향 프로그래밍은 대부분의 소프트웨어 개발 프로젝트에서 선택하는 방법론이다. 객체지향 프로그래밍의 강점은 공통된 동작을 모델링하는 능력에 있다. 그러나 많은 소프트웨어 개발자들이 경험한 것처럼, 객체지향 프로그래밍은 많은 모듈들에 걸쳐 있는 동작(횡단 관심사라고 칭함)을 모듈화하지 못하기 때문에 이런 공백을 채워질 수 있는 새로운 방법론 또는 언어의 필요성이 대두되었다.

여러 방법론 - 발생 프로그래밍(generative programming), 메타 프로그래밍(meta-programming), 반사 프로그래밍(reflective programming), 합성 필터링(compositional filtering), 적응 프로그래밍(adaptive programming), 주체 지향 프로그래밍(subject-oriented programming), 관점 지향 프로그래밍(aspect-oriented programming), 의도 프로그래밍(intentional programming) 등-이 횡단 관심사를 모듈화하는 가능한 방법으로 제안되었다.

관점지향 기술[3, 4]은 이 중에서 가장 주목을 받는 기술이며 프로그래밍 개발 방법론 발전사에서 객체지향 다음 단계의 방법론으로 부각되었다. 관점지향 프로그래밍은 횡단 관심사를 독립적으로 모듈화하여 나중에 합성(직조) 과정을 거쳐 한 개의 시스템으로 조립하는 새로운 방법론이다. 이런 방법을 사용하면 앞에서 지적한 소프트웨어의 복잡성과 시장 출시 시간을 줄이는 데 모두 도움이 된다. 또한 관점지향 개발 방법론은 규모가 작은 소프트웨어 개발에도 적용하여 작은 투자와 작은 위험 부담으로 큰 혜택을 얻을 수 있다.

관점 지향 프로그래밍 개념 형성기에 제록스의 팔로 알토 연구소의 연구원이었던 그레거 킥젤즈(Gregor Kiczales)와 노스이스턴 대학의 크리스티나 로페즈(Cristina Lopes)가 많은 기여를 하였다. **그레거가 1996년도에 Aspect-oriented Programming(AOP)이란 용어를 만들었고 제록스와 노스이스턴 대학의 연구원들과 미국의 국방고등연구소(Defense Advanced Research Projects Agency)의 자금 지원을 받아 1990년대 말에 관점 지향 프로그래밍을 최초로 구현한 AspectJ[5]를 제작하였다. 현재 AspectJ 프로젝트는 공개 소스 커뮤니티인 eclips.org에서 개발과 관리를 하고 있다.**

이 논문은 관점지향 프로그래밍과 이를 지원하는 가장 강력한 언어인 AspectJ 및 대표적인 응용 사례를 소개하여 관점지향 프로그래밍이 현재 이용 가능한 기술이며 많은 혜택이 있음을 보임으로 국내에서도 이 기술의 보급 및 확산을 목표로 한다. 이 논문의 구성은 다음과 같다. 2장에서 횡단 관심사와 이를 처리하는 현재의 방법론의 문제점을 기술한다. 3장에서 관점지향 프로그래밍의 개념을 소개한다. 4장에서 관점지향 프로그래밍을 구현한 AspectJ의 기능을 기술하고 간단한 애플리케이션을 보여준다. 5장에서 관점지향 프로그래밍의 적용 분야, IBM 사의 적용 사례, 지원 도구들 및 관점지향 모델링을 기술한다. 6장에서 결론을 맺는다.

2. 횡단 관심사의 문제점

관심사(concern)는 전체적인 소프트웨어 시스템의

목적을 달성하기 위해 처리해야 하는 구체적인 요구사항이나 고려사항을 의미한다. 소프트웨어 시스템은 요구되는 관심사들을 구현한 것이다.

관심사는 핵심(core) 관심사와 횡단(cross-cutting) 관심사로 분류할 수 있다. 핵심 관심사는 각 모듈에서 수행해야 하는 기본적이고 대표적인 업무 처리 기능을 취급하며 횡단 관심사는 여러 개의 모듈에 걸치는 시스템 전체적인 부가적인 요구사항을 다룬다. 기업 업무에서 대표적인 횡단 관심사의 예를 든다면 사용자 인증, 로깅, 자원 공유, 관리, 성능, 메모리 관리, 자료의 영속성, 보안, 다중 트레드 처리, 트랜잭션 무결성, 오류 검사, 정책 시행 등이다. 예를 들어, 로깅 관심사는 로깅을 필요로 하는 거의 중요한 모듈에 다 적용되며, 사용자 인증 관심사는 참조 제어(access control)를 하는 모든 모듈과 관련이 있다.

어떤 관심사가 모듈화가 잘 안되면 이는 코드 혼합(tangling)과 코드 산재(散在, scattering)의 2가지 형태로 나타난다. 어떤 시스템에 이런 두 가지 현상이 나타난다면 이는 대부분의 경우 횡단 관심사를 기존의 방식으로 구현하였기 때문이다.

2.1 코드 혼합(code tangling)

코드 혼합은 모듈에 여러 관심사를 동시에 구현하고자 할 때 발생한다. 개발자는 모듈을 구현할 때 비즈니스 로직, 로깅, 영속성, 성능, 보안과 동기화 등의 여러 관심사를 함께 고려한다. 따라서 각 관심사가 구현된 부분이 그림 1에서처럼 동시에 모듈에 나타나게 되며 결과적으로 코드 혼합을 초래한다.

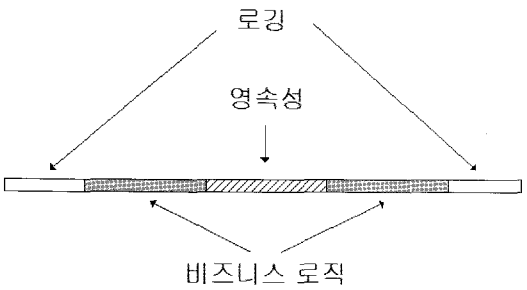

그림 1 코드 혼합

2.2 코드 산재(code scattering)

코드 산재(散在)는 한 개의 관심사가 여러 개의 모듈에 구현될 때 발생한다. 횡단 관심사는 그 정의대로 여러 개의 모듈들에 걸쳐있게 되므로 구현된 부분도 그 모듈들에 산재되게 된다. 그림 2에서처럼, 로깅을 사용하는 시스템에서 로깅 관심사는 로깅을 사용하는 모든 모듈에 영향을 미치게 된다.

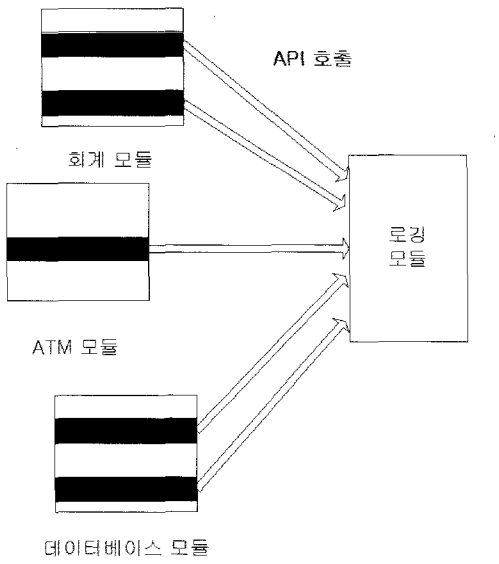

그림 2 코드 산재

3. 관점지향 프로그래밍

Aspect는 두 가지 의미를 가지고 있다. 요구 분석 또는 설계 시점의 aspect는 (Aspect Oriented Programming에서의 aspect도 같은 의미) 횡단 관심사(concerns that crosscut)의 의미가 있고 4.2.6절에서 소개되는 aspect는 프로그램 구성물(construct)이다. 전자의 의미로 사용된 aspect에 대해서는 이 논문에서 '관점'이란 용어를 사용하였다. 후자의 의미로 쓰였을 때는 '애스펙트'라는 용어를 그대로 사용하였는데 이는 객체지향의 class를 그대로 클래스로 사용하는 것과 유사하다.

관점 지향 기술[6]은 횡단 관심사를 모듈화하기 위하여 객체 지향 기술과 프로시저 프로그래밍의 기초 위에 이것들의 개념과 구성물을 확장한 것이다. 관점 지향 기술에서도 핵심 관심사는 기존의 선택된 기본 방법론으로 구현한다. 예를 들어, 객체 지향 기술이 기본 기술이라면 핵심 관심사는 클래스로 구현된다. 그러나 횡단 관심사는 관점 지향 기술로 구현된다.

횡단 관심사를 관리하는데 객체 지향 기술과 관점 지향 기술의 근본적인 차이점은 관점 지향 기술에서는 각 관심사를 구현하는 모듈이 자신에게 적용되는 횡단 관심사를 전혀 모른다는 사실이다. 예를 들어, 비즈니스 로직 모듈은 자신의 메소드가 로깅이 되고 있는지 또는 사용자 인증을 하고 있는지 전혀 모르게 된다. 따라서 각 관심사가 독립적으로 발전해 나갈 수 있다.

3.1 관점 지향 프로그래밍의 방법론

관점 지향 프로그래밍을 사용하여 시스템을 개발(관심사를 찾아냄, 관심사를 구현, 구현한 것을 통합하여

최종시스템을 제작)하는 것은 다른 방법론을 사용하여 시스템을 개발하는 것과 유사하다. 관점 지향 프로그래밍 연구 커뮤니티에서는 이런 과정을 다음과 같이 정의한다.

3.1.1 관점 분해(decomposition)

이 단계에서 요구 사항을 분해하여 핵심 관심사와 횡단 관심사를 찾아내고 핵심 관심사와 횡단 관심사를 분리해낸다. 예를 들어, 어떤 클래스에 핵심 비즈니스 로직, 로깅, 스레드 안전성 그리고 인증과 같은 관심사가 있다고 하자. 이 중에서 핵심 비즈니스 로직만이 핵심 관심사이다. 모든 다른 관심사는 시스템 전체에서 사용되는 관심사로 다른 모듈들에서도 필요하기 때문에 횡단 관심사로 분류된다.

핵심이란 용어는 상대적인 용어로 이해하여야 한다. 권한부여 모듈에서 핵심 관심사는 사용자를 신임장(credential)으로 변환하는 것과 이 신임장이 제한된 서비스를 참조하는데 자격이 있는지를 결정하는 것이 핵심 관심사가 될 것이다. 그러나 비즈니스 로직 모듈에서는 권한부여 관심사는 부수적인 관심사가 되고 관점 지향 프로그래밍을 사용하면 비즈니스 모듈 내에서 구현되지 않는다.

3.1.2 관심사 구현

이 단계에서 각 관심사(핵심 또는 횡단 관심사 모두 포함)를 독립적으로 구현한다. 그림 3에서 보듯이 개발자들은 핵심 관심사인 회계, ATM, 데이터베이스 모듈을 독립적으로 개발한다. 핵심 관심사를 개발하기 위해 평소 데로 프로시저 또는 객체 지향 프로그래밍을 사용할 수 있다.

기존의 방법으로는 각 핵심 관심사에 포함되었던 횡단 관심사인 로깅을 관점지향 프로그래밍에서는 애스펙트를 사용하여 독립적으로 개발한다. 로깅 모듈에 대한 API 호출은 프로그램 개발 당시에는 핵심 관심사에는 전혀 포함되지 않고 모두 로깅 애스펙트에서 처리된다. 애스펙트 모듈 안에 직조 규칙(weaving rule)이 포함되는데 이 규칙을 사용하여 다음 절에 설명하는 직조 과정을 거쳐 각 핵심 관심사에서 로깅이 호출되도록 변환된다.

3.1.3 관점 합성(recomposition)

이 단계에서 그림 4에서 보는 것처럼 핵심 관심사와 횡단 관심사가 합성 과정을 통해 통합된다. 합성하는 실제 과정은 직조(weaving) 또는 통합(integrating)이라고 하며 이 직조를 담당하는 소프트웨어를 직조기(weaver)라고 한다. 직조기는 애스펙트 내에 있는 직조 규칙을 사용하여 최종 시스템을 조립한다.

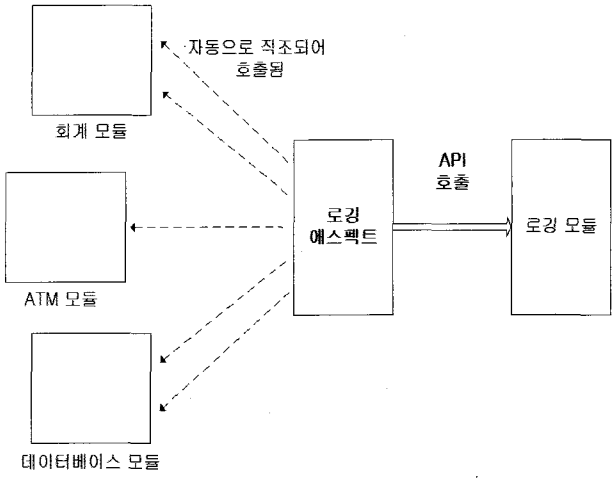

그림 3 AOP의 핵심 관심사와 횡단 관심사의 구현

직조는 소스 코드 수준에서 수행될 수도 있으며 바이트 코드 수준에서 수행될 수도 있는데 AspectJ의 경우는 바이트 코드 수준에서 직조가 된다.

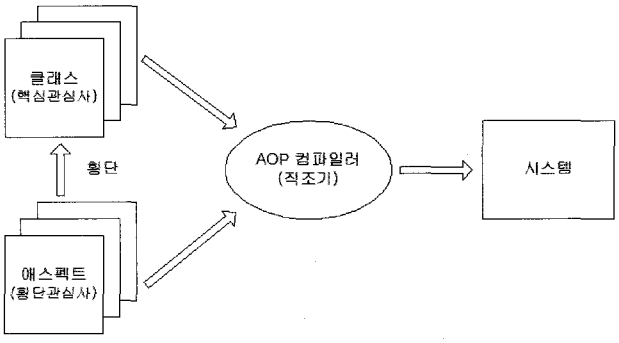

그림 5 애스펙트 직조(weaving)

4. AspectJ

AspectJ는 Java 언어에 관점 지향 개념을 추가하여 확장한 대표적인 범용의 언어이다[5,6]. AspectJ가 Java의 확장이므로 모든 유효한 Java 프로그램은 유효한 AspectJ 프로그램이 된다. AspectJ 컴파일러는 Java 바이트 코드 명세 규격을 따르기 때문에 Java 가상기계는 AspectJ가 생성하는 클래스 파일을 실행한다. Java를 기본 언어로 사용하기 때문에 AspectJ는 Java의 모든 장점을 이어 받으며 Java 개발자가 AspectJ 언어를 이해하기 쉽다.

AspectJ는 언어 명세와 언어 구현의 두 부분으로 구성된다. 언어 명세 부분은 코드를 작성하는 언어를 정의한다. 언어 구현 부분은 컴파일, 디버깅과 많이 사용하는 통합개발환경과 통합을 위한 도구를 제공한다.

4.1 AspectJ의 횡단(cross-cutting)

AspectJ에서 컴파일러가 직조 규칙을 구현하는 것을

횡단(cross-cutting)이라고 한다.횡단 관심사를 모듈화 하기 위하여 직조 규칙은 여러 모듈에 걸쳐 있게 된다. AspectJ의 횡단은 정적 횡단과 동적 횡단의 두 가지로 나누어진다.

4.1.1 동적 횡단

동적 횡단은 새로운 행위를 프로그램의 실행에 직조하는 것이다. AspectJ에서 발생하는 대부분의 횡단은 동적이다. 동적 횡단은 핵심 프로그램이 실행될 때 추가된 새로운 코드를 여러 개의 모듈에 걸쳐서 실행함으로, 또는 기존의 코드를 새로운 코드로 치환하여 실행함으로 시스템의 행위를 변경한다. 예를 들어, 특정한 메소드나 예외 처리 루틴이 실행되기 전에 어떤 행위가 실행되길 원한다면, 실행되어야 하는 지점과 그 지점에 이르렀을 때 취해야 될 행위를 별도의 모듈인 애스펙트에 지정하면 된다.

4.1.2 정적 횡단

정적 횡단은 클래스, 인터페이스, 또는 애스펙트와 같은 정적 구조에 변경을 직조하는 것이다. 이것은 시스템의 실행 행위를 변경하지 않는다. 정적 횡단은 동적 횡단의 구현을 지원하기 위해 많이 사용된다. 예를 들어, 동적 횡단에서 사용될 수 있는 클래스의 상태와 메소드를 정의하기 위하여 새로운 자료와 메소드를 클래스와 인터페이스에 추가할 수 있다. 정적 횡단의 다른 기능으로, 여러 모듈에 걸쳐 컴파일 시점의 경고나 오류를 선언하는 기능이 있다.

4.2 횡단 요소

AspectJ는 Java 언어를 확장하여 동적과 정적 횡단의 직조 규칙을 지정한다. AspectJ 확장은 다음에 나오는 구성물을 사용하여 직조 규칙을 프로그램 내에서 표시하게 한다. 이 구성물들이 횡단 관심사를 구현한 모듈을 만드는데 기초 요소로 쓰인다.

4.2.1 결합점(join point)

결합점은 프로그램 실행 과정에서 구별 가능한 지점이다. 결합점은 메소드 호출이나 객체의 멤버에 값 할당 등이 될 수 있다. 결합점은 횡단 코드가 들어오는 지점이기 때문에 AspectJ에서는 모든 것이 결합점을 중심으로 발생한다. 다음 Account 클래스에서 결합점은 credit() 메소드와 _balance 인스턴스 멤버의 값을 할당하는 명령어다.

가능한 모든 결합점이 AspectJ에서 허용되는 것은 아니고, AspectJ에서 제공되는 결합점은 메서드 결합점, 생성자 결합점, 필드 참조 결합점, 예외처리 실행 결합점, 클래스 초기화 결합점, 객체 초기화 결합점, 객체 초기화 이전, 충고 실행 결합점이다.

```
public class Account {
    ...
    void credit(float amount) {
        _balance += amount;
    }
}
```

4.2.2 교차점(pointcut)

교차점은 결합점들을 선택하고 결합점의 환경정보(context)를 수집하는 프로그램 구성물이다. 예를 들어, 교차점은 메소드 호출을 지정하고 메소드가 호출되었을 때의 대상 객체와 메소드의 매개변수 등과 같은 메소드의 환경정보를 얻을 수 있다. 앞에 있는 Account 클래스의 credit() 메소드의 실행을 선택하는 교차점은 아래와 같다.

```
execution(void Account.credit(float())
```

교차점에 개발자들이 횡단 모듈이 작동해야 할 프로그램의 위치를 알려주는 규칙을 서술하고 결합점은 이런 규칙을 만족시키는 프로그램 내에서의 위치이다.

AspectJ에서 제공하는 교차점의 종류로 결합점 유형에 따른 교차점, 제어 흐름 교차점, 어휘 구조에 근거한 교차점, 실행 객체, 매개변수, 조건 검사 교차점 등이 있다.

4.2.3 충고(advice)

충고는 교차점에서 지정한 결합점에서 실행되는 코드이다. 충고의 몸체는 메소드 몸체와 유사하여 결합점에 도달할 때 실행하는 로직을 포함하고 있다. 충고는 결합점의 실행 이전(before)에, 결합점의 실행 이후(after)에 또는 결합점을 대체하여(around) 실행될 수 있다. 이후 충고(after advice)에서 결합점 실행의 예외 발생 여부에 따라 충고를 다르게 적용할 수 있다. 대체 충고(around advice)에서 결합점에 원래 있던 기존 코드의 실행을 변경할 수 있는데, 즉 기존 코드를 다른 코드로 대체하거나, 또는 아예 기존 코드가 실행되지 않게 할 수도 있다.

앞에 있는 교차점을 사용하여 Account 클래스의 credit() 메소드 실행 이전에 메시지를 출력하는 충고를 다음과 같이 작성할 수 있다.

```
before() : execution(void
            Account.credit(float()) {
    System.out.println("About to perform
                        credit operation");
}
```

교차점과 충고는 함께 동적 횡단 규칙을 형성한다. 교차점은 필요한 결합점을 알려 주고 충고는 결합점에서

실행해야 할 행위를 제공하여 동적 횡단의 전체가 완성된다.

4.2.4 도입(introduction)

도입은 타입간의 선언(inter-type declaration)이라고도 한다. 도입은 시스템의 클래스, 인터페이스, 그리고 애스펙트에 변화를 도입하는 정적 횡단 명령이다. 도입은 모듈의 행위에 직접 영향을 미치지 않는 정적 변화를 만든다. 예를 들어 클래스에 메소드나 필드를 추가할 수 있다.

다음 도입은 Account 클래스가 BankingEntity 인터페이스를 구현하도록 선언한다.

```
declare parents: Account implements
                    BankingEntity;
```

4.2.5 컴파일 시점 선언

컴파일 시점 선언은 정적 횡단 명령으로, 컴파일 시점에 특정한 코드의 유형을 만나면 컴파일러가 경고나 오류 메시지를 발생할 수 있게 한다. 예를 들어, EJB에서 Abstract Window Toolkit(AWT)를 호출하면 오류라고 선언할 수 있다.

다음 선언은 시스템의 어떤 부분이 Persistence 클래스에 있는 save() 메소드를 호출하면 컴파일러가 경고 메시지를 내도록 한다. 메소드 호출을 걸러내기 위하여 call() 교차점을 사용해야 한다.

```
declare warning : call(void
                    Persistence.save(Object))
   : "Consider using
     Persistence.saveOptimized()";
```

4.2.6 애스펙트(aspect)

클래스가 Java의 중심 단위인 것처럼 애스펙트는 AspectJ의 중심 단위이다. 애스펙트는 동적 횡단과 정적 횡단을 위한 직조 규칙을 표현하는 코드를 포함한다. 앞에서 설명한 교차점, 충고, 도입과 선언이 애스펙트에 들어가게 된다. AspectJ의 요소 외에, 애스펙트는 클래스처럼 자료, 메소드와 중첩 클래스 멤버를 포함할 수 있다.

앞에 나온 모든 코드를 모아서 다음과 같은 애스펙트를 만들 수 있다.

```
public aspect SampleAspect {
   before() : execution(void
                Account.credit(float()) {
      System.out.println("About to perform
                credit operation");
   }
   declare parents: Account implements
                BankingEntity;
   declare warning : call(void
                Persistence.save(Object
                ))
      : "Consider using
        Persistence.saveOptimized()";
}
```

모든 기능들이 어떻게 동작하는지 살펴보자. 횡단 모듈을 개발할 때 처음으로 할 일은 행위를 확대하거나 수정하고자 하는 결합점을 찾는 일이고 다음으로 추가할 새로운 행위를 설계하는 일이다. 이 설계를 구현하기 위해 먼저 모든 구현을 포함하는 모듈인 애스펙트를 작성한다. 그리고 애스펙트 내에서 원하는 결합점을 표시하는 교차점을 작성한다. 마지막으로 각 교차점에 필요한 충고를 생성하고 결합점에 도달할 때 실행되어야 할 행위를 그 충고 내에 작성한다. 어떤 종류의 충고에 대하여는 구현을 지원하기 위해 정적 횡단을 사용할 수도 있다.

4.2.7 AspectJ의 고급 기능

지금까지 설명한 기능으로 관점지향 프로그래밍을 할 수 있지만 좀 더 복잡한 애플리케이션을 위해 AspectJ에서 다음과 같은 고급 기능을 제공한다. 결합점에 대한 정적 또는 동적 정보를 얻기 위해 반사(reflection) 기능을 사용할 수 있다. 여러 애스펙트에 있는 한 개 이상의 충고가 결합점에 적용될 때는 애스펙트 우선순위를 조정하는 기능을 사용할 수 있다. 애스펙트 인스턴스를 가상 기계별, 객체별, 제어 흐름별로 연관(association)시킬 수 있는 기능이 제공된다. 예외 처리 횡단 관심사를 모듈화하는데 예외 연화(軟化, softening) 기능을 사용하여 검사받는 예외(checked exception)를 검사받지 않는 예외(unchecked exception)로 변경할 수 있다. 클래스의 비공개 멤버를 접근하기 위해 애스펙트에 특권을 부여하는 기능이 제공된다.

4.3 간단한 애플리케이션 구현

이 절에서 간단한 애플리케이션을 구현하여 보자[6]. 우선 예제 1에서, 메시지를 프린트하는 두 개의 메서드가 포함된 클래스를 생성하자.

예제 1 MessageCommunicator.java

```
public class MessageCommunicator {
   public static void deliver(String mes-
   sage) {
      System.out.println(message);
   }
```

```
    public static void deliver(String person,
    String message) {
        System.out.print(person + ", " + mes-
        sage);
    }
}
```

MessageCommunicator 클래스는 메서드 두 개(일반인에게 메시지를 보내는 메서드와 특정인에게 메시지를 보내는 메서드)를 가지고 있다. 다음에 Message Communicator 클래스의 기능을 이용하는 Test 클래스를 예제 2에 작성하자.

예제 2 Test.java

```
public class Test {
    public static void main(String[] args) {
        MessageCommunicator.deliver
        ("AspectJ를 배우고 싶으세요?");
        MessageCommunicator.deliver
        ("길동", "재미있어요?");
    }
}
```

MessageCommunicator와 Test 클래스를 함께 컴파일하고 Test 프로그램을 실행하면 다음과 같은 출력을 얻는다.

```
>ajc MessageCommunicator.java Test.java
>java Test
AspectJ를 배우고 싶으세요?
길동, 재미있어요?
```

MessageCommunicator 클래스에 있는 코드를 전혀 변경하지 않고, 단지 애스펙트를 추가함으로 클래스의 기능을 향상시킬 수 있다. 예제 3에 있는 것처럼 예의범절 애스펙트(횡단 관심사)를 구현하자. 이 애스펙트는 "안녕하세요!"로 먼저 인사하고 그 후에 모든 메시지를 전달한다.

예제 3 MannersAspect.java

```
public aspect MannersAspect { // ① 애스펙트 선언
    pointcut deliverMessage() // ② 교차점(pointcut)
                                선언
        : call(* MessageCommunicator.
                deliver(..));
    before() : deliverMessage() { // ③ 충고(advice)
        System.out.print("안녕하세요! ");
    }
}
```

이제 클래스 파일을 애스펙트와 함께 컴파일하자. AspectJ 컴파일러인 ajc가 예제 3에 있는 애스펙트를 직조(weaving)한 클래스 파일을 생성하려면 모든 입력 파일을 함께 ajc로 컴파일해야 한다. 컴파일하고 실행하면 다음과 같은 출력을 보게 된다.

```
>ajc MessageCommunicator.java
MannersAspect.java Test.java
>java Test
안녕하세요! AspectJ를 배우고 싶으세요?
안녕하세요! 길동, 재미있어요?
```

이 애스펙트의 코드를 살펴보자.

① 애스펙트의 선언은 클래스 선언과 유사하다.

② 애스펙트에 정의된 교차점 deliverMessage()는 MessageCommunicator에 있는 메서드 deliver()에 대한 모든 호출을 포착한다. 교차점에 있는 와일드카드 *는 반환형에 관계없이, 또한 괄호 안에 있는 와일드카드 ..는 매개변수의 종류와 개수에 관계없이, 모든 deliver() 메서드를 포착한다.

③ 이제 충고를 정의한다. before()를 사용하면 MessageCommunicator.deliver()가 실행되기 전에 충고가 실행된다. 충고에서 메시지 "안녕하세요!"를 줄을 바꾸지 않고 출력한다.

이번에는 한국말 특성을 살려 인사하는 애스펙트를 만든다. 예제 4에서, 사람 이름 뒤에 "님"을 부쳐 인사하고 메시지를 전달한다.

예제 4 KoreanSalutationAspect.java

```
public aspect KoreanSalutationAspect {
    pointcut sayToPerson(String person)
//① deliver() 메서드를 포착하는 교차점
        : call(* MessageCommunicator.
                deliver(String, String))
          args(person, String);
    void around(String person) :
    sayToPerson(person) { //② 충고
        proceed(person + "님"); // ③ 충고 몸체
    }
}
```

클래스를 MannersAspect와 KoreanSalutationAspect와 함께 컴파일하고 Test 클래스를 실행하면 다음과 같이 출력이 된다.

```
>ajc MessageCommunicator.java
 MannersAspect.java
```

KoreanSalutationAspect Test.java

```
>java Test
안녕하세요! AspectJ를 배우고 싶으세요?
안녕하세요! 길동님, 재미있어요?
```

① 교차점 sayToPerson은 두 개의 매개변수를 취하는 MessageCommunicator.deliver()를 호출하는 결합점을 포착한다. 사람 이름 뒤에 "님"을 붙이려면 매개변수 person을 사용해야 한다. args()의 첫 번째 매개변수인 person은 메서드 deliver()의 첫 번째 매개변수가 person으로 들어오는 것을 지정한다. sayToPerson()의 매개변수 person은 deliver()의 첫 번째 매개변수와 일치하여야 한다.

② 출력에서 사람 이름 뒤에 "님"을 부치려면 deliver() 메서드의 매개변수를 고쳐서 실행해야 한다. before() 충고를 사용하면 deliver()가 실행되기 전에 충고의 코드가 실행되고, deliver()가 정작 실행될 때 매개변수가 변하지 않으므로 before() 충고를 사용할 수 없다. 충고를 받는 메서드의 매개변수를 수정하기 위해 around() 충고를 사용한다. around() 충고는 메서드의 기존 환경을 변경하여 실행할 수 있다.

③ AspectJ 키워드인 proceed()는 포착된 결합점을 실행하는 명령이다. 메서드 deliver()의 원래 매개변수 person을 sayToPerson(person)으로 포착하여 proceed() 내에서 "님"을 뒤에 추가하여 proceed()에 넘겨준다. 결과는 수정된 매개변수를 가지고 MessageCommunicator.deliver()를 실행하게 된다.

5. 관점지향 프로그래밍의 적용

개발자들이 AspectJ를 실제 프로젝트에 사용하여 미들웨어 플랫폼을 향상시키고[7,8], 성능을 증가시키며[9], 기존의 애플리케이션에 보안 기능을 추가하고 기업 애플리케이션 통합(Enterprise Application Integration, EAI)을 구현한다. 이런 프로젝트의 결과는 매우 고무적이며 코드의 양과 제품을 제작하는 시간을 줄인다.

5.1 적용 분야

관점지향 프로그래밍의 적용 범위는 매우 넓다.

5.1.1 기본 적용

이 분야는 누구나 쉽게 적용하여 혜택을 얻을 수 있으며 위험 부담이 적다. 제일 쉽게 적용할 수 있는 분야가 감시 기법(monitoring technique)으로 로깅, 트레이싱과 프로파일링 등에 적용할 수 있다. 이것을 사용하여 프로그램의 디버깅 또는 성능 분석 등에 매우 유용하게 사용할 수 있다.

두 번째로 정책 준수(policy enforcement)에 관점지향 프로그래밍을 적용하면 혜택이 크다. 정책 준수는 시스템 컴포넌트가 주어진 프로그래밍 관행(practices)을 따르고 지정된 규칙을 준수하며 주어진 가정을 만족하도록 시행하는 기법이다. 정책 준수는 말하기는 쉬워도 이것을 실제로 구성원들이 따르도록 하는데 어려움이 따른다. 관점지향 프로그래밍은 이를 아주 간단하게 해결한다. 관점지향적인 방법은 미리 작성된 몇 개의 애스펙트를 개발자가 작성한 코드와 함께 컴파일하면, 위반 사항을 찾아 낼 수 있게 한다. 또한 이런 애스펙트는 재사용이 가능하기 때문에 추가의 개발비 없이 다른 프로젝트에 적용할 수 있다.

세 번째로, 자원 풀링(pooling)도 대표적인 횡단 관심사인데 관점지향 프로그래밍에서는 풀링 로직을 애스펙트에 포함함으로써 이 횡단 관심사를 효율적으로 모듈화할 수 있다.

5.1.2 고급 적용

고급 적용은 관점지향 프로그래밍을 사용하여 복잡한 횡단 관심사를 모듈화하는 것으로 AspectJ의 고급 구성물을 사용한다. 기본 적용과 다르게, 고급 적용의 혜택을 얻으려면 배치된 시스템 즉 생산 시스템에서 관점지향 프로그래밍 기법이 사용되어야 한다.

이 분야로는 관점지향 프로그래밍을 이용한 설계 패턴과 이디엄, 스레드 안전, 인증, 권한부여, 트랜잭션 관리, 비즈니스 규칙, 미들웨어 리팩토링[7,8] 등에 적용될 수 있다.

또한 새로운 애플리케이션을 완전히 관점지향 개발 방법론에 기반하여 개발할 수 있으며 조만간 관점지향 소프트웨어 개발 방법론에 기초한 미들웨어가 출현할 것으로 기대된다.

5.2 기업에서 적용 사례

IBM은 관점지향 프로그래밍과 AspectJ를 다른 기업에 앞서 적용하여 관점지향 개발방법론이 약속이 아니고 현실이란 것을 입증하는 대표적인 사례를 보여주고 있다[1,8]. IBM은 관점지향 개발방법론을 사용하여 많은 혜택을 입었으며 기술 측면, 또한 비즈니스 측면에서 관점지향 개발 방법론을 매우 중요한 기술로 간주하여 현재 IBM의 Tivoli, WebSphere, DB2, Lotus, Rational 등 주요 제품군에 적용하고 있다. 애플리케이션 서버인 Websphere에 대한 IBM 초기의 적용 사례를 살펴본다.

5.2.1 동질적인 관심사에 적용

코드가 산재되어 구현되어 있으나 어느 지점에서도 실제로 하는 일이 동일하기 때문에 동질적인 관심사로 칭한다.

5.2.1.1 트레이싱과 로깅에 적용

기존의 정책 문서로 표시하였던 트레이싱과 로깅을 AspectJ의 애스펙트로 변환하여 구현하였으며 그 결과는 구현이 더 정확하고 완전하여졌다. 애스펙트를 사용하지 않았을 경우와 비교하여 성능에 있어서 1% 정도의 차이밖에 나지 않았다.

5.2.1.2 초기 장애 자료 수집(First Failure Data Capture)

초기 장애 자료 수집은 장애가 생긴 근원지에 가장 가까운 곳에서 자료를 수집함으로 장애 복구를 효율적으로 할 수 있다. AspectJ를 사용하여 기존에 110개의 소스 모듈의 241개 다른 장소에 흩어져 있던 코드를 1개의 애스펙트로 모듈화하였다. 또한 기존의 코드에 있던 355개의 잘 못된 부분을 애스펙트로 수정할 수 있었으며, 기존의 산재된 코드에서 제공하지 못하였던 새로운 기능들이 가능하게 되었다.

5.2.1.3 조직에의 영향

정책 준수(좋은 관행을 사용하게 하는 정책)를 위해 정책 문서 대신 애스펙트를 사용함에 따라 정책 준수의 정확하고 완전한 구현이 가능하였다. 사용자 교육이 필요 없어지고 정책의 구현과 변경이 매우 쉬워지고 경비가 절약된다.

5.2.2 이질적인 관심사에 적용

애플리케이션 서버 Websphere에서 대용량의 횡단 관심사인 EJB 컴포넌트를 성공적으로 리팩토링하였다[8]. 이 관심사도 코드 혼합과 코드 산재의 형태를 취하지만 5.2.1절과 다르게 각각의 장소에서 행위가 서로 다르므로 이질적인 관심사로 칭한다.

Websphere는 15,000개의 소스 파일과 수백만 라인의 코드로 이루어져 있고 내부적으로 250개의 컴포넌트로 구성되어 있으며 약 80개의 최상위 관심사로 구분될 수 있다. 이 관심사의 예를 들면 ORB, 웹 컨테이너, EJB 컨테이너, 웹 서비스 지원 등이다.

IBM에서 시도한 작업은 Websphere에서 AspectJ를 사용하여 EJB 지원을 완전히 분리해내고 Websphere 빌드 시점에 EJB 코드를 포함시키거나 제거할 수 있게 하였다. 결과는 성공적이었으며 비교적 짧은 시간에 이것이 가능하였다. EJB 지원을 포함하거나 또는 포함하지 않은 각각의 Websphere 버전이 필요한 테스트를 성공적으로 통과하였다. EJB 지원이 제거된 Websphere는 당연히 메모리를 덜 사용하며 성능도 좋아진 것을 확인할 수 있었다.

이 리팩토링을 통하여 EJB 지원 컴포넌트를 독립적으로 개발 및 유지보수 할 수 있어 소프트웨어 개발의 복잡성을 줄일 수 있다. 또한 고객에게 불필요한 기능은 빌드 시점에 제거하여 보냄으로 고객의 다양한 요구에 부응할 수 있다.

5.3 관점지향 프로그래밍 구현 제품

5.3.1 관점지향 프로그래밍을 지원하는 도구

Java 플랫폼의 도구들은 다른 플랫폼 도구보다 더 성숙되고 안정되어 있으며 사용자 커뮤니티도 크지만, C++와 .NET 진영의 AOP 사용자 커뮤니티도 성장하고 있다. AOSD.NET에 많은 도구들이 소개되어 있는데, 가장 많은 사용자가 채택한 도구들을 표 1에 정리하였다. 최근에 BEA와 Eclipse는 AspectWerkz[10]를 Eclipse의 AspectJ와 병합하는 프로젝트를 진행하고 있다. 이 도구들의 비교는 [11]을 참조하기 바란다.

Java 언어를 지원하는 다른 도구로는 abc(Aspect Bench Compiler for AspectJ)[15], JAC(Java Aspect Component)[16] 등이 있다.

Java 이외에 언어를 위한 도구로, C를 위한 AspectC[17], C++를 위한 AspectC++[18], C#을 위한 AspectC#[19], Python을 위한 Pythius[20] 등이 있는데 대부분 AspectJ에 있는 개념을 이들 언어에 적용한 것이다.

표 1 사용자가 많은 AOP 도구(모두 Java 언어용)

AOP 지원도구	출시년도	비고
AspectJ[12]	2001	eclipse.org, IBM 지원
AspectWertz[10]	2002	BOA 제품
JBoss AOP[13]	2004	JBoss의 프레임워크에 AOP 기능 추가
Spring AOP[14]	2004	Spring의 프레임워크에 AOP 기능 추가

5.3.2 AspectJ를 지원하는 통합개발환경

플러그인 AspectJ Development Tools(AJDT)[21]를 Eclipse에 설치하면 AspectJ 기능을 Eclipse에서 사용할 수 있다. 그 외 JDeveloper, NetBeans, JBuilder와 Emacs JDEE와 같이 널리 사용되는 통합개발환경에도 해당되는 플러그인을 설치하면 AspectJ가 지원되어 애스펙트 개발 과정이 쉬워진다.

5.3.3 AOP와 관련된 중요한 자원들

관점지향 학회 홈 페이지(http://aosd.net/) : 관점

지향 학회의 홈페이지이며 다양한 AOSD 자원을 제공한다.

관점지향 모델링 워크숍(http://dawis.informatik 2005/) : 모델링에 대한 논문, 발표 자료 등을 구할 수 있다.

AspectJ 홈 페이지(http://eclipse.org/aspectj/) : AspectJ를 다운받을 수 있다.

5.4 관점지향 모델링

개발 단계의 초기에 애스펙트를 찾아 모델링을 하면 설계 부품의 재사용이 가능하고 AOP 시스템을 위한 자동 코드 생성이 가능하고 요구분석, 설계, 코드 생성 등의 과정에서 일관성이 유지된다. 3가지 다른 관점지향 모델링 방법이 있다.

첫 번째로, 관점지향 분석과 설계를 위해 초기에 많은 연구자들이 관심을 가진 것은 UML에서 제공하는 Profile을 사용하여 기본 UML을 확장하는 방식[22]이었다. UML은 stereotype, tagged value와 constraints와 같은 확장 기능을 지원한다. UML에서 미리 정해진 확장 메커니즘을 Profile이라고 부른다. CORBA나 EAI 등에서도 이 Profile을 사용하여 그들의 영역에 UML을 적용하였다. AOSD의 경우에도 Profile에서 제공되는 방법을 사용하여 UML을 확장하였다. 이렇게 확장된 UML은 UML을 지원하는 기존의 케이스 도구에 쉽게 포함될 수 있어 코드 자동 생성이 가능하다. AOSD를 위한 Profile에 Stereotype으로 ⟨⟨aspect⟩⟩, ⟨⟨crosscut⟩⟩, tagged value로 {Synchronous}, operation으로 ⟨⟨Preactivation⟩⟩, ⟨⟨Postactivation⟩⟩이 추가 되었다.

두 번째로, Jacobson은 use-cases를 사용하여 요구분석 단계에서 횡단 관심사를 찾아내고 모델화하여 설계, 구현 단계에 이르기까지 일관되게 횡단 관심사를 독립적으로 관리할 수 있는 방법을 [23]에서 제공한다. 이것이 가능한 이유는 UML에는 AOP에서 제공하는 횡단 관심사를 표현할 수 있는 개념과 구성물이 이미 제공되고 있기 때문이다. 문제는 UML에서 제공하는 확장 use-case를 구현할 수 있는 프로그래밍 언어가 AOP가 나오기 이전에 존재하지 않았던 것이다. 표 2에서 확인하듯이 UML과 AspectJ 간에 서로 대응하는 구성물을 가지고 있음을 알 수 있다.

세 번째로 제안된 것으로 주제 접근 방식(theme approach)[24]이 있다. 여기에서 주제의 의미는 개발자가 고려하는 것으로 시스템에서 만족시켜주어야 하는 별개의 기능(functionality), 관점(aspect)과 관심사(concern) 등을 의미한다. 주제 접근 방식은 두 부분으로 구성되어 있는데, Theme/Doc 부분은 요구사항 분석 단계에서 관점을 포함하여 주제를 찾아내고, Theme/UML 부분은 주제를 설계 단계에서 UML로 표현하는 것이다.

표 2 UML과 AspctJ의 구성물 비교

UML	AspectJ
extension point	joinpoint/ pointcut
extension use-case flow	advice
extension use-case	aspect
base use-case	core concern

6. 결 론

우리는 이 논문에서 관점지향 프로그래밍(AOP)과 이를 구현하는 가장 강력한 언어인 AspectJ를 고찰하였다. 관점지향 프로그래밍은 기존의 프로그래밍 방법론이 해결하지 못하던 횡단 관심사를 모듈화하는 새로운 프로그래밍 패러다임이다. 횡단 관심사를 핵심 관심사와 독립적으로 개발하고 나중에 직조(weaving) 과정을 통하여 핵심 관심사와 횡단 관심사를 통합하여 시스템을 구현한다. 관점지향 소프트웨어 개발 방법론(AOSD)은 UML과 AOP를 사용하여 소프트웨어 개발 전 단계(요구분석, 설계, 구현 등)에 걸쳐 관심사(핵심 관심사와 횡단 관심사를 포함)들이 독립성을 유지하도록 지원한다.

관심지향 개발 방법론은 기술적으로 안정되고 성숙되어 있다. 현재 AspectJ를 사용하여 애플리케이션들이 개발되고 있으며 점차로 영역을 넓혀 가고 있다. 관점지향 프로그래밍의 방법론을 사용하면 고도의 모듈화를 꽤하며, 시스템 개선의 용이하고, 과잉 설계를 피하며, 코드 재사용을 확대할 수 있는 많은 장점을 얻을 수 있다. IBM에서는 Websphere 애플리케이션 서버에서 EJB 모듈을 분리하여 실제 Websphere를 빌드할 때 EJB의 포함 여부를 결정하는 리팩토링을 성공적으로 수행하였고, 이를 확대하여 기존의 주요 제품군과 신제품에도 관점지향 방법론을 적용할 계획을 가지고 있다. 조만간 업계에서 처음 설계 시점부터 관점지향 방법론을 사용하는 미들웨어 제품이 출시될 것으로 예견된다. 마지막으로 지적할 점은 이런 대형 프로젝트에서뿐만 아니라 중소형의 프로젝트에서도 관점지향 프로그래밍을 사용하면 많은 투자를 하지 않고 큰 혜택을 받을 수 있다는 것이다.

참고문헌

[1] Daniel Sabbah, "Aspects - from Promise to Reality," 2004 AOSD conference, 2004, pp.1-2.

[2] IBM and BEA Joint Specifications, Service Data Objects. http://www- 106.ibm.com/developerworks/library/j-cornmonj-sdowmt/
[3] Gregor Kiczales and et la, "Aspect-Oriented Programming," ACM Computing Surveys, Volume 28 , Issue 4es, Dec., 1996.
[4] Tizila Elrad and et la, "Aspect-Oriented Programming," Communications of the ACM, Vol.44, No.10, Oct., 2001, pp.29-32.
[5] Gregor Kiczales and et. la, "Getting Started with AspectJ," Communications of the ACM, Vol.44, No.10, Oct., 2001, pp.59-65.
[6] 금영욱 역, Ramnivas Laddad 저, AspectJ in Action, 도서출판그린, 2005.
[7] Charles Zhang and Hans-Arno Jacobsen, "Refactoring Middleware with Aspects," IEEE Transactions and Parallel and Distributed System, Vol. 14, No. 11, Nov. 2003, pp.1058-1073
[8] Adrian Colyer and Andrew Clement, "Large-scale AOSD for Middleware," 2004 AOSD conference, 2004, pp.56-65
[9] Erik Putrycz and Guy Bernard, "Using Aspect-Oriented Programming to build a portable load balancing service," Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops (ICDCSW'02), 2002
[10] AspectWerkz http://aspectwerkz.codehaus.org/index.html
[11] AOP tools comparison, http://www-128.ibm.com/developerworks/java/library/j-aopwork1/
[12] AspectJ, http://eclipse.org/aspectj/
[13] JBoss AOP, http://jboss.org/products/aop
[14] Spring AOP http://www.springframework.org/
[15] abc, http://abc.comlab.ox.ac.uk/introduction
[16] jac, http://jac.objectweb.org/
[17] AspectC Yvonne Coady, Gregor Kiczales, Mike Feeley and Greg Smolyn University of British Columbia "Using AspectC to Improve the Modularity of PathSpecific Customization in Operating System Code," http://www.cs.ubc.ca/labs/spl/papers/2001/coady-psc.pdf
[18] AspectC++, http://aspectc.org/
[19] Aspect#, http://www.castleproject.org/index.php/AspectSharp
[20] Pythius http://pythius.sourceforge.net/
[21] AJDT, http://www.eclipse.org/ajdt/
[22] Omar Aldawud, Tzilla Elrad, and Atef Bader, "UML Profile for Aspect-Oriented Software Development" Workshop on Aspect-Oriented Modeling with UML, 2003 (http://lglwww.epfl.ch/workshops/aosd2003/papers/AldawudAOSD_UML_Profile.pdf)
[23] Ivar Jacobson and Pan-Wei, Aspect-Oriented Software Development with Use Cases, Addison Wesley, 2004 Dec.
[24] Siobhan Clarke, Elisa Baniassad, Aspect-Oriented Analysis and Design The Theme Approach, Addison Wesley, 2005 March.

금 영 욱

1978 서울대학교 수학과(학사)
1992 서강대학교 정보처리학과(이학사)
1997 서강대학교 컴퓨터학과(공학박사)
1981~1993 한국 및 미국 IBM 시스템즈 엔지니어
2004~2005 호주 시드니대학교 방문교수
1997~현재 성결대학교 컴퓨터공학부 교수
관심분야 : 컴포넌트소프트웨어개발, CORBA, Web Services, AOP, AOSD
E-mail : ywkeum@sungkyul.edu