프로그램 개발 및 유지보수를 지원하는 횡단관심사 명세 기법

(Specification of Crosscutting Concerns to Support Program Development and Maintenance)

박 옥 자 † 유 철 중 †† 장 옥 배 ††
(Oak-Cha Park) (Cheol-Jung Yoo) (Ok-Bae Jang)

요 약 관점지향 프로그래밍은 기존의 소프트웨어 개발 방법론으로 해결하기 어려운 횡단관심사를 모듈화 하는데 목적을 두고 있다. 초기 관점지향 프로그래밍은 프로그램 개발 단계에 초점을 맞추어 발전하다가 최근에는 요구사항 분석 및 설계부터 코드 구현 단계까지 전체 프로세스에 관점지향 프로그래밍 방법론을 적용하고자 많은 연구가 이루어지고 있다. 하지만, 관점지향 프로그래밍에서 필요한 교차점, 결합점, 충고 정의를 위한 표준화된 지침 및 명세가 없어 프로그램 개발 및 유지보수를 어렵게 하고 있다.

본 논문에서는 이와 같은 문제점을 해결하기 위하여 관점지향 프로그래밍 개발 및 유지보수를 지원하는 횡단관심사 명세 기법을 제시한다. 관심사 명시, 관심사 설계, 그리고 코드 구현 세 단계로 이루어지는 명세 단계는 요구사항 분석 단계에서 얻어지는 횡단관심사 정보를 코드로 개발할 수 있도록 단계별 지침을 제공한다. 명세 과정의 각 단계에서는 전체 횡단 관심사 목록, 횡단관심사 명세서, 애스펙트 클래스 참조 테이블, 핵심 클래스 참조 테이블을 구축하여 코드 개발 및 유지보수시 프로세스간의 추적 자료로 사용된다. 본 논문의 명세 기법은 관점지향 프로그래밍에서 해결하기 어려운 구현의 문제점, 프로그램 이해의 어려움, 재사용의 문제점을 해결함으로써 시스템 개발을 지원한다.

키워드 : 관점지향 프로그래밍, 횡단관심사, AspectJ, 명세, 유지보수, 추적성

Abstract Aspect-Oriented Programming(AOP) has focused on improving the modularity of the crosscutting concerns. The existing AOP methodology has been mainly focused on the implementation method of programs and thus developer-oriented concern extraction and development were performed. Recently, many studies for applying AOP to the entire software development processes ranging from requirement analysis to design and implementation are being conducted. But specification methods having consistency from the initial phase of concern extraction to implementation phase are not sufficient.

In this paper, we have presented the specification of crosscutting concerns so as to solve these problems. The specification of crosscutting concerns provides guidelines and specification from the requirement analysis phase to the process of converting extracted crosscutting concerns to codes. This method reduces the gap to the process of mapping extracted crosscutting concerns into a single class and thus enhances program development and understandability. In addition, it raises program reusability, maintenance and extensibility by enhancing traceability.

Key words : Aspect-Oriented Programming, crosscutting concerns, AspectJ, specification, maintenance, traceability

1. 서 론

소프트웨어 설계에서 복잡한 시스템을 단순화하는 최선의 방법은 관심사 분리(Separation of Concerns)이다. 관심사 분리는 컴퓨터 프로그래밍의 가장 기초가 되는 원리중의 하나이다. 관심사 분리 과정은 문제 영역들을 독립적으로 모듈화 함으로써 소프트웨어의 성능을 향상

† 학생회원 : 전북대학교 컴퓨터학부
ojpark@bsc.ac.kr
†† 종신회원 : 전북대학교 컴퓨터학부 교수
cjyoo@chonbuk.ac.kr
okjang@chonbuk.ac.kr
논문접수 : 2007년 3월 23일
심사완료 : 2007년 6월 9일

시키게 되고 차후 유지보수에도 많은 이점을 준다. 기존의 절차지향 프로그래밍, OOSD(Object-Oriented Software Development), CBSD(Component-Based Software Development)는 관심사 분리 과정을 통해 문제 영역들을 독립적으로 분해하는데 목적을 두고 발전하였다. 하지만, OOP(Object-Oriented Programming)와 같이 모듈화가 뛰어난 방법을 사용하더라도 해결되지 않는 기능들이 있다. 예를 들어, 객체지향 프로그래밍은 클래스 혹은 오퍼레이션(operation)을 이용하여 관심사를 모듈화 하지만 보안(security), 메모리 관리(memory management), 로깅(logging)과 같은 부가기능들은 다른 클래스에 긴밀하게 결합되어 있거나 여러 클래스나 오퍼레이션에 횡단(cross-cutting)하여 결합되어 있기 때문에 모듈화하기 어렵다[1]. 이와 같이 여러 클래스에 중복 정의되거나 여러 클래스에 횡단하여 구현되는 기능을 횡단 관심사(crosscutting concerns)라 하고 해당 시스템의 일반적인 비즈니스 로직과 목적이 그대로 드러난 관심 영역을 핵심 관심사(core concerns)라 한다. 핵심 관심사는 기존의 OOP를 통해 쉽게 모듈화 및 추상화가 가능하지만 횡단 관심사는 모듈화가 어렵다. 모듈화 되지 않은 횡단 관심사는 다른 핵심 관심사를 구현한 모듈과 매우 긴밀하게 결합되어 있어 요구사항 변경으로 유지보수가 빈번하게 일어날 경우 소프트웨어 복잡성(software complexity)과 같은 문제를 유발할 수 있다.

관점지향 프로그래밍(Aspect-Oriented Programming, 이하 AOP)은 기존의 프로그램 개발 방법론으로 해결하기 어려운 횡단 관심사를 모듈화 하는데 좋은 기법으로 제시되고 있다. AOP는 횡단 관심사를 모듈화하기 위하여 객체지향 기술과 절차지향 프로그래밍의 기초 위에 이것들의 개념과 구조물을 확장한 것이다. AOP를 이용하여 개발자들은 횡단 관심 모듈을 각각 독립된 모듈로 중복 없이 작성하고 이를 결합(weaving)을 통해 핵심 관심 모듈과 결합시키기 때문에 서로 독립성을 가진 다차원(multi dimension)의 모듈을 작성할 수 있게 되었다[2-4].

초기 AOP는 프로그램 개발 단계에 초점을 맞추어 발전하다가 최근에는 요구사항 분석 및 설계부터 코드 구현 단계까지 전체 프로세스에 AOP 방법론을 적용하고자 많은 연구가 이루어지고 있다. 하지만, 해결해야 할 몇 가지 문제점을 가지고 있다. 첫째, 프로그램 개발을 위한 표준화된 방법이 없다. 초기 AOP는 코드 구현에 초점을 둔 개발자 중심이었기 때문에 AOP 코드 구현을 위한 용어, 문법, 구현 방법 등이 일정한 기준이 없이 그 내용과 형식이 다양했다. 최근에는 요구사항 분석에서부터 설계, 구현에 이르기까지 전반적인 소프트웨어 개발 공정에 적용하고자 많은 연구가 제시되고 있지만 AOP에서 필요한 교차점(pointcut), 결합점(join point), 충고(advice) 정의를 위한 표준화된 지침 및 명세가 없어 프로그램 개발 및 유지보수를 어렵게 하고 있다. 둘째, AOP로 만든 프로그램은 흐름을 이해하고 추적하기가 어렵다. AspectJ의 경우 교차점, 결합점, 충고를 이용하여 개발한 애스펙트 클래스(aspect code)는 핵심 클래스(core class)와 독립적으로 구현되어 모듈화가 잘 되지만 결합 후 프로그램 실행 결과를 보기 전까지는 프로그램 흐름이 어떻게 이루어지고 어떤 클래스들과 연관되어 있는지 코드만으로 파악하기 어렵다[5][6]. 즉, AOP로 구현한 애스펙트 클래스는 핵심 클래스의 참조(reference) 관계를 인지하지만 핵심 클래스는 애스펙트 클래스의 존재를 알 수 없기 때문이다. AOP에서 나타나는 이와 같은 문제점이 명확하게 해결되지 않았을 때 프로그램 유지보수(maintenance) 및 추적(traceability)을 어렵게 한다[7].

본 논문에서는 이와 같은 문제점을 해결하기 위하여 횡단관심사 명세 기법(Specification of Crosscutting Concerns, SCC)을 제시였으며 그림 1은 이의 필요성을 도식화한 것이다. 그림 1에서와 같이 초기 요구분석 단계에서 추출한 관심사는 핵심 관심사와 횡단 관심사로 분류하여 추출한다. 구현 단계에서 핵심 관심사는 OOD나 CBD와 같은 기존의 프로그램 개발 방법론을 적용하여 개발한다. 횡단 관심사는 AOP 기법을 이용하여 프로그램을 개발하는데 본 논문에서는 그림 1과 같이 횡단 관심사를 애스펙트 클래스로 변환하기 위한 횡단관심사 명세(SCC) 및 지침(guideline)을 제공한다. SCC는 요구분석 단계에서 추출한 횡단 관심사를 코드로 변환하는 과정에서 필요한 표준화되는 명세를 제공함으로써 프로그램 개발 및 추적을 쉽게 지원하고자 한다. 명세 정보에는 하나의 횡단 관심사에 대한 특징, 결합점, 교차점 및 충고 정보, 프로그램 이해를 위한 다른 클래스와의 참조 관계를 보여준다. 명세는 세 단계의 프로세스로 이루어지며 각 프로세스마다 일정한 지침이 제공된다. 각 단계에서 얻어지는 산출물은 다음 단계에 필요한 입력 정보가 되어 단계간의 갭(gap)을 줄여준다. 또한, SCC는 AOP에서 해결하기 어려운 코드 구현의 문제점을 해결하고 프로그램 흐름을 용이하게 하는 추적 자료 제공됨으로써 재사용성을 높여준다.

논문의 구성은 다음과 같다. 2장은 관련 연구로서 AOP 개발 기법 및 문제점을 살펴보고 3장에서 본 연구에서 제안한 명세 기법을 기술한다. 4장에서는 적용사례를 통해 모바일 뱅킹 시스템의 명세 과정을 살펴보고 5장에서 토의 및 평가를 수행한다. 6장에서는 향후 연구과제를 포함한 결론을 맺는다.

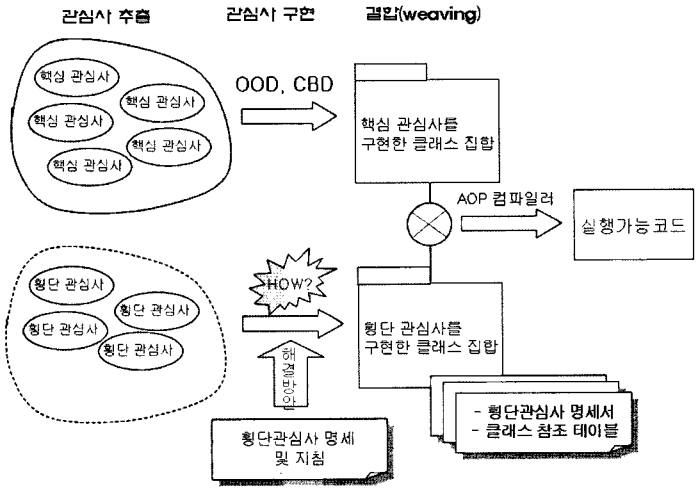

그림 1 횡단관심사 명세기법의 필요성

2. 관련 연구

Java를 이용한 객체지향 개발 프로그램이 다양하게 사용되면서 Java와 AOP 결합을 위한 다양한 개발 방법론과 프로그래밍 언어들이 제시되었다. Java와 호환이 되는 AOP 언어로는 AspectJ, JAsCO[8], Hyper/J[9], JAC[10]등이 있다. JAsCo는 CBD와 Java Beans 컴포넌트 모델을 수정한 방법으로 애스펙트 빈(aspect beans)과 연결자(connector) 기법을 도입하였다. 애스펙트 교차점과 충고 정보를 *hook*이라는 키워드로 설정하고 결합점은 *connector*를 이용하여 표현하였다. 하지만, 실제 핵심 클래스에 애스펙트 클래스가 결합하고 있어 프로그램 수정 및 재사용시 핵심 클래스를 수정해야 하는 문제점이 발생한다. AspectJ는 Java 언어와 유사한 구조를 가지면서 AOP에 필요한 횡단요소를 추가하여 행위(behavior)를 표현하므로 가장 이상적인 AOP 툴로 각광받고 있다. 하지만, 결합점, 교차점 정보 추출의 어려움이 있고 Java 언어로 구현한 핵심 관심사와 AspectJ를 이용한 횡단관심사 코드를 완전히 독립적으로 개발하기 때문에 결합에 의한 컴파일러를 수행하기 전까지는 실행결과를 예측하기 어렵게 한다. 또한, 프로그램 수정 및 확장시 클래스간의 참조 관계가 명시되지 않기 때문에 코드 흐름을 이해하기 어렵게 한다[7]. 다음의 방법론은 AspectJ를 이용한 기존의 개발 기법들이다.

2.1 AML을 이용한 AspectJ 코드 생성

Gropher[11]는 애스펙트 구조 설계 모델링을 위한 AML(Aspect Modeling Language)을 제시하였다. 초기 추출된 횡단관심사를 AML로 표기하여 이를 UML로 매핑하는 방법이다. AML은 UML로 표현하기 힘든 횡단관심사를 클래스 다이어그램과 같은 형태로 표현하였고 이를 기반으로 코드 정보를 추출하도록 지원하였다. 관심사를 기본 패키지(base package), 애스펙트 패키지(aspect package), 그리고 이를 연결하는 커넥터 패키지(connector)로 표현하였다. 또한, AspectJ에 AML을 적용한 AspectJ 커넥션 모델(AspectJ connection model)을 제시하였다. 하지만, 결합점, 교차점, 충고 정보를 표현하는 데는 한계가 있고 실제 명세된 정보를 개발자가 이해하여 코드로 변환하기에는 복잡한 구조를 가지고 있다. 또한 커넥터 패키지를 명세하기 위한 명확한 지침이 없어 초기 개발자가 설계하기에는 어려움이 있다.

2.2 Composition 패턴을 이용한 AspectJ 코드 생성

Clarke[12]는 Composition 패턴을 이용한 AOP 개발 기법을 제시하여 이를 AspectJ에 적용하였다. 이 패턴은 목적 기반 디자인(Subject-Oriented Design)과 UML 템플릿을 통합을 기반으로 한 구조이다. 패턴 명세(Pattern Specification), 패턴 바인딩(Pattern Binding), 결합(Composition Output)의 세 단계로 이루어진 Composition 패턴은 디자인 단계에서 애스펙트 클래스에 필요한 모든 정보를 추출 및 명세하여 AspectJ와 매핑하는 AOP 개발 기법이다. 이 기법은 디자인 단계에서 횡단관심사와 핵심관심사의 관계를 명확히 알 수 있고 이 패턴을 구현 단계에 직접적으로 적용함으로써 코드의 분리를 지원한다. 개발 방법은 Composition 패턴의 이름을 애스펙트 클래스로 매핑하고 패턴 내에서 사용한 오퍼레이션 템플릿을 클래스내의 교차점으로 선언하였다. 이후 Composition 패턴과 다른 클래스간의 관계를 충고로 명시하여 AspectJ를 이용한 코드 개발을 하였다. 이 패턴은 디자인 단계에서 직접적으로 코드 변

환을 할 수 있다는 장점을 가져오지만 대부분의 횡단관심사 설계 정보를 패턴 설계 단계에서 얻어야 하며 대부분의 패턴 정보가 하나의 횡단 관심사와 하나의 핵심 클래스와 연결되도록 되어 있으므로 다양한 클래스에 횡단되는 클래스일 경우 여러 개의 패턴을 산출하게 되므로 프로그램 이해를 어렵게 할 수 있다.

2.3 *AspectU*, *AspectSD*를 이용한 코드 생성

논문 [13]에서는 *AspectU*, *AspectSD*를 이용한 개발 기법을 제시하였다. 유스 케이스 기반 교차점 선언을 제안하였고 이를 *AspectU*로 표현하였다. 또한, 결합점 추출을 위하여 *AspectSD* 기법을 제안함으로써 설계 정보로부터 코드 개발을 가능하게 하였다. 시스템 설계 및 개발 동안 순차다이어그램이 유스케이스와 소스코드간의 다리 역할을 수행하듯이 *AspectSD*는 *AspectU*와 *AspectJ*간의 다리 역할을 수행한다. 관심사 추출, *AspectU*, *AspectSD*, *AspectJ* 단계로 이루어지는 이 기법은 기존의 다른 방법론에 비하여 단계간의 갭을 줄여줌으로써 프로그램 개발을 지원하였다. 하지만, 코드 구현을 위하여 *AspectU* 명세 기법과 *AspectSD* 언어를 이해해야 하므로 오히려 *AspectJ* 외에 추가적인 개발 기법의 이해를 필요로 한다.

이러한 기법들은 애스펙트 코드를 핵심 클래스 내에 표기하여야 하거나 대부분 설계 단계에서 애스펙트 추출 정보를 표현하고자 하지만 이와 같은 기법은 초기 개발자가 설계 정보로부터 코드 구현을 어렵게 하고 유지보수성을 떨어뜨린다. 따라서 기존의 방법론을 보완할 수 있도록 개발에 필요한 상세 지침을 제공하고 유지보수 및 재사용성을 증가시킬 수 있는 횡단관심사 명세 기법(Specification of Crosscutting Concerns, SCC)을 제시하였다.

3. 횡단관심사 명세 기법

3.1 SCC 프로세스 및 제약사항

그림 2는 SCC 명세를 위한 단계별 절차 및 산출물을 기술한 프로세스이다. SCC 프로세스는 세 단계로 이루어져 있으며 각 단계마다 개발에 필요한 지침을 제공하고 있다. 전체횡단관심사테이블(CCT)는 초기 요구분석 단계에서 얻어진 횡단관심사를 테이블로 구축한 결과이며 이를 기반으로 SCC 명세가 이루어진다. Phase 2에서는 핵심 클래스 참조 테이블(Core Class Reference Table, CCRT), 애스펙트 클래스 참조 테이블(Aspect Class Reference Table, ACRT)과 같은 산출물을 얻게 되는데 이 산출물은 프로그램 이해 및 유지보수를 위한 추적 자료로 제공된다.

본 논문에서 제안한 SCC는 이미 추출된 횡단관심사를 명세하는데 초점을 두고 있으므로 요구사항으로부터

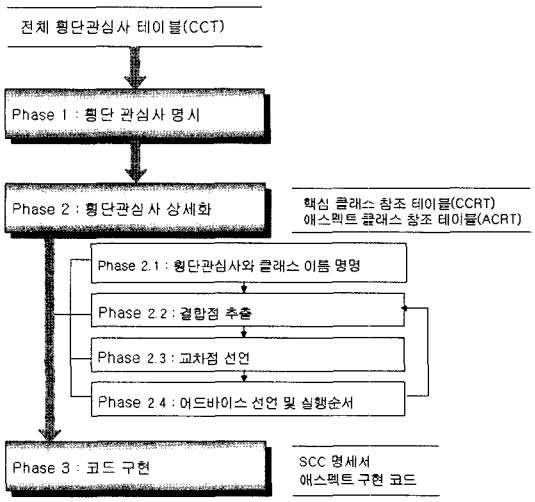

그림 2 SCC 프로세스 및 산출물

얻게 되는 횡단관심사 추출은 기존의 유스케이스 기반 방법론을 적용한다[14-16]. 또한, 결합점 정보는 [13]에서 제안한 순차다이어그램(sequence diagram)을 이용한 결합점 추출 방법을 기반으로 하고 교차점 선언은 본 논문에서 제시한 실행기반 교차점 선언 방법을 적용한다.

3.2 횡단관심사 명세

이 절에서는 그림 2에서 보여준 SCC 프로세스의 상세 과정을 기술한다. 각 단계에서 사용되는 지침은 표 1을 기반으로 하고 있다.

Phase 1 : 관심사 명시

요구분석 단계에서 추출한 각각의 횡단관심사에 대한 구체적 정보를 기술하는 단계이다. 이 단계에서는 횡단관심사의 특징, 구현 정책(policy), 다른 애스펙트와의 충돌에 대비한 우선순위(priority)정보를 보여준다. 구현 정책은 프로그램 구현시 필요한 제약사항(constraints)이나 특징을 기술한다. 일반적으로 하나의 횡단관심사는 하나의 애스펙트 클래스로 구현한다. 하지만, 인증(authentication)과 같은 횡단관심사는 실행과정에서 로깅(logging)과 권한인증(authorization) 기능이 동시에 필요하다. 이 경우 하나의 애스펙트 클래스에 로깅, 인증, 권한인증을 같이 구현하는 것이 바람직할 수 있다. 이와 같은 상세 정보를 관심사 명시 단계에서 기술함으로써 차후 프로그램 이해 및 유지보수에 참조하게 한다. 우선순위 정보는 횡단 관심사간의 충돌을 방지하게 위한 장치이다. 시스템에 여러 개의 애스펙트가 존재할 경우 각각의 애스펙트에 있는 한 개 이상의 충고가 핵심 클래스의 결합점에 적용될 수 있다. 이 때 애스펙트 클래스에 우선순위를 명시함으로써 같은 결합점에서 발생할 수 있는 충돌을 방지한다. 충돌의 우선순위는 프로그

표 1 횡단관심 명세를 위한 지침

단계	특징 및 지침		산출물
Phase 1	요구분석 단계에서 추출한 횡단관심에 대한 특징 및 제약사항을 기술하는 단계이다.		전체횡단관심사테이블(CCT)
	지침 1.1	횡단관심사의 이름과 특징을 기술한다.	
	지침 1.2	구현 정책에 대해 기술한다.	
	지침 1.3	다른 횡단 관심과의 충돌에 대비한 우선순위를 명시한다.	
Phase 2	관심사를 코드로 변환하기 위한 설계 단계이다. 결합점을 추출하고 교차점, 충고를 선언한다. 결합점에 관련된 클래스 정보를 기술한다.		애스펙트클래스 참조테이블(ACRT)과 핵심클래스참조테이블(CCRT)을 작성
	지침 2.1	하나의 횡단 관심을 하나의 횡단관심 유스케이스로 명시한다.	
	지침 2.2	횡단관심 유스케이스는 하나의 애스펙트 클래스 이름으로 매핑된다.	
	지침 2.3	코드 이해를 위하여 *관심사이름_Apsect.java*와 같은 형식의 명명 규칙을 따른다.	
	지침 2.4	결합점을 추출한다.	
	지침 2.5	교차점을 선언한다. [교차점 선언 지침 참조]	
	지침 2.6	애스펙트 클래스 테이블, 핵심 클래스 테이블을 구축한다.	
	지침 2.7	충고를 선언한다.	
Phase 3	Phase 2에서 명시한 클래스에 대한 상세 코드를 구현한다.		코드 및 SCC

램 특성에 따라 달라질 수 있으니 일반적인 횡단 관심사의 우선순위 명시는 [17,18]의 방법에 따른다.

Phase 2 : 관심사 상세화

Phase 1에서 명시한 횡단 관심사를 코드로 변환하기 위한 상세화 단계이다. 횡단 관심사는 애스펙트 클래스로 선언하고 애스펙트 클래스에서 구현한 각각의 기능이 결합되어 동작할 결합점을 추출하고 교차점과 충고를 선언한다. 관심사 상세화 단계는 다음과 같이 세 단계로 세분화된다.

Phase 2.1: 횡단 관심사와 클래스 이름 명명

하나의 횡단관심사를 하나의 클래스 이름으로 매핑(mapping)한다. 횡단 관심 유스케이스를 하나의 애스펙트 클래스로 변환할 때 프로그램 이해를 돕기 위하여 클래스 이름은 *관심사 이름_Aspect.java* 명명 규칙을 따른다.

Phase 2.2: 결합점 추출

결합점은 횡단 관심 모듈의 기능이 삽입되어 동작할 수 있는 실행 가능한 특정 위치이다. AOP에서 가장 핵심적인 개념이며 교차점과 충고의 결합 규칙에 의해 횡단 행위가 동작하게 되는 위치이기도 하다. 일반적으로 교차점에서 선택할 수 있는 노출된 결합점 만이 사용 가능한데 메소드 호출, 필드 액세스 위치, 생성자 호출, 메서드 또는 생성자 실행, 클래스 생성, 필드 참조, 예외 핸들러(exception handler)가 동작하는 위치 등이 있다[17]. 이와 같이 시스템에서 노출된 결합점 만이 프로그램의 실행을 확대하거나 변화할 수 있는 유일한 코드이다.

Phase 2.3 : 교차점 선언

교차점 선언은 애스펙트 클래스에서 구현한 충고가 어느 결합점에서 동작할지 위치를 지정하는 단계이다. 교차점은 결합점을 선택하고 결합점의 환경정보를 수집하는 프로그램 구조물로서 횡단 관심사 구현 및 코드 이해를 위한 골격을 형성하는 부분인데 대부분 명확한 지침과 표준화된 방법이 없이 개발자의 일로 치부해왔다[19]. Jacobson만이 extension 메커니즘을 이용한 *before/after* 교차점 방법을 사용하였지만 Renaud Pawlak[19]은 이 방법 또한 재사용의 문제점이 있다고 반론하였다. 간단하고 명확한 교차점 선언 방법과 명세가 없는 상태에서의 개발자 중심의 구현은 여전히 프로그램 개발을 어렵게 하고 코드 흐름을 파악하기 어렵게 한다. 나아가 이미 개발된 AOP 코드에 대한 재사용성이 어렵게 되기도 한다.

본 논문에서는 실행기반 교차점 선언 방법을 제시하였다. 실행 기반 교차점은 Phase 2.2에서 추출한 결합점 위치 중 공통된 횡단관심 모듈이 수행될 결합점 위치를 모듈화 하여 하나의 교차점으로 선언한다. 이를 위한 지침은 다음과 같다.

지침 2.3.1. 하나의 횡단관심 유스케이스를 세부적인 실행 순서로 분류한다. 각 실행순서는 같은 충고가 동작할 결합점 위치 집합이 될 수 있다.

지침 2.3.2 각 실행순서를 교차점으로 선언한다.

지침 2.3.3 추출한 전체 결합점 중 교차점에서 실제 충고가 동작할 결합점을 교차점 단위로 명시한다.

지침 2.3.4 교차점과 충고를 이용하여 횡단 규칙을 형성한다.

이 단계에서는 또한 프로그램 이해를 지원하는 클래스 참조 테이블을 구축한다. AOP는 횡단 관심 모듈이 핵심 클래스의 특정 위치에 직접적으로 삽입되는 것이 아니고 결합 단계에서 참조되기 때문에 핵심 클래스와의 명시적인 코드 참조관계는 이루어지지 않는다. 논문[7]에서는 AOP 기법에 의한 클래스 간의 참조 관계를 개방형(Open), 명시적 클래스(Class-Directional), 명시적 애스펙트(Aspect-Directional) 그리고 폐쇄형(Closed)

으로 분류하였다. 논문 [20]에서는 이와 같은 클래스 관계를 도식화하여 그림으로 표현하였다. 이 중 애스펙트 클래스와 핵심 클래스간의 가장 바람직한 참조 관계를 명시적 클래스 관계로 제시하였으며 그림 3은 명시적 클래스 관계를 보여준다. 그림 3의 (a)는 애스펙트 클래스 관점을 보여준다. 애스펙트 클래스에는 선언된 결합점과 이와 연결된 핵심 클래스 정보를 알고 있으며 하나의 애스펙트 클래스는 여러 개의 핵심 클래스와 연결된다. 그림 3의 (b)는 핵심 클래스 관점에서 보여준 그림으로 핵심 클래스는 애스펙트 클래스의 참조 관계를 인지하지 못하고 있다. 또한, 핵심 클래스는 여러 개의 애스펙트 클래스에 의해 참조될 수 있다. 그림 3과 같이 명시적 애스펙트 관계로 유지되는 AOP 코드에서 애스펙트 클래스와 핵심 클래스는 코드만으로는 두 클래스 간의 결합관계를 파악할 수 없게 한다. 이는 두 클래스 간의 소프트웨어 결합도를 낮게 함으로써 모듈화를 높여주지만 프로그램 수정 및 확장시 코드 이해와 추적을 어렵게 하는 단점이 될 수 있다. 따라서 교차점 선언 단계에서는 횡단 관심 모듈이 삽입되어 동작할 결합점 선언을 하고 더불어 결합점에 관련된 클래스 정보도 기술하여야 한다. 이 클래스 정보는 프로그램 이해에서 필요한 애스펙트 클래스와 핵심 클래스간의 유일한 추적 정보가 될 수 있다. 클래스 참조 테이블은 다음과 같이 두 가지 유형으로 구축할 수 있다.

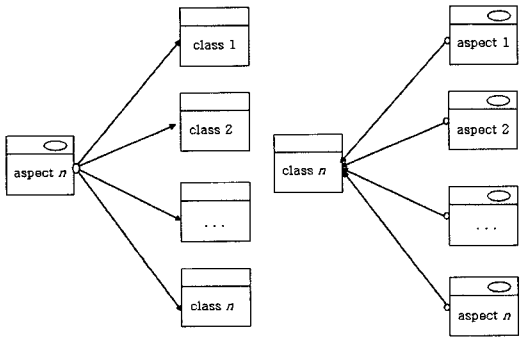

그림 3 명시적 클래스 관계

aspect n: 애스펙트 클래스

class n: 핵심 클래스 (*n: number*)

표 3 핵심 클래스 참조 테이블(CCRT) 템플릿

CCRT Number : 핵심 클래스 참조 테이블 번호		
핵심 클래스 이름	애스펙트 클래스 이름 (횡단관심사 명세서 번호)	횡단관심사 특징
클래스 이름	애스펙트 클래스 이름과 횡단관심사 명세서 번호	...
	...	...

표 2는 애스펙트 클래스 관점에서 관련된 핵심 클래스 정보를 보여주는 애스펙트 클래스참조 테이블(Aspect Class Reference Table, ACRT) 템플릿으로 그림 3의 (a) 관계를 표로 보여준 것이다. 하나의 횡단관심사의 기능이 변경될 경우 이를 구현한 애스펙트 클래스뿐만 아니라 관련된 핵심 클래스도 동시에 변경되어야 한다. ACRT에는 관련된 클래스 이름, 결합점 위치, 결합점 수와 같은 추적정보를 제공한다. 하나의 횡단관심사에 관련된 핵심 클래스의 개수와 결합점 수는 모듈화 정도를 평가할 수 있는 기준이 되어 응집도 및 재사용성과 같은 코드 품질평가에 적용할 수 있다. 표 3은 핵심 클래스와 관련된 횡단관심사 정보를 보여주는 핵심 클래스 참조 테이블(Core Class Reference Table, CCRT) 템플릿으로 그림 3에서 (b)의 관계를 보여준다. 핵심 관심사의 기능이 변경될 경우 이를 구현한 핵심 클래스의 코드를 수정하게 되면 애스펙트 클래스의 결합점, 교차점, 충고를 구현한 코드도 동시에 변경되어야 한다. 이 두 유형의 클래스 참조 테이블은 횡단관심사 명세 과정에서 상호보완적으로 구축되며 프로그램 이해 및 수정을 위한 핵심 자료가 된다.

Phase 2.4 : 충고 선언 및 실행순서 명시

AspectJ에서 하나의 애스펙트는 하나 이상의 충고를 포함한다. 애스펙트 클래스 내에 선언된 충고는 교차점에서 선언된 결합점 집합에 삽입되어 실제 동작할 행위를 구현한다. Phase2.2에서 선언된 교차점과 *before*, *after*, *around*와 같은 충고를 이용하여 프로그램 실행 순서를 제어한다. 교차점과 충고에 의해 횡단규칙이 선언되므로 횡단관심사 명세에서 교차점과 충고의 실행 순서를 명시해주면 애스펙트 클래스 내의 흐름을 보다 쉽게 파악할 수 있도록 한다.

Phase 3 : 코드 구현

Phase 1에서 Phase 3까지의 프로세스 과정은 애스펙

표 2 애스펙트 클래스 참조 테이블(ACRT) 템플릿(n : number)

ACRT Number : 애스펙트 클래스 참조 테이블 번호			
애스펙트 클래스 이름 (횡단관심사 명세서 번호)	핵심 클래스 이름	결합점 정보	결합점 수
애스펙트 클래스 이름과 횡단관심사 명세서 번호	관련된 클래스 이름	결합점 위치	n
	〃	〃	〃
	...	...	...

트 클래스에 전체 설계 구성요소를 추출하고 설계한다. 이 Phase 3 단계에서는 이전 단계에서 얻어진 정보를 기반으로 실제 상세 코드를 구현하는 단계이다. 이미 추출된 명세를 기반으로 코드를 구현하게 되므로 간단하면서 모듈화가 잘된 코드를 얻을 수 있다. 이 장에서는 기술하지 않고 적용 사례에서 살펴보기로 한다.

4. 적용 사례

본 논문에서는 제시한 명세 방법론을 모바일뱅킹 시스템(Mobile Banking System, 이하 MBS)에 적용하였다. 모바일뱅킹이란 고객이 언제 어디서나 모든 기기를 통해 은행 잔고를 확인하고 돈을 이체할수록 하는 서비스이다. 모바일뱅킹은 이용하고자 하는 고객이 개인 휴대폰에 모바일 칩을 탑재하고 PIN(Personal Identity Number)라는 고유 번호를 할당받음으로서 사용 권한을 가질 수 있다. PIN은 모바일 뱅킹을 접속할 때 입력하는 인증 패스워드이다. 모바일뱅킹의 기본 기능은 조회, 계좌이체, 그리고 현금 인출이다. 조회 기능은 PIN 인증만으로 가능하고 계좌이체는 PIN 인증 외에 개인에게 할당된 보안카드 번호를 입력하는 추가 인증절차를 거쳐야 한다. 현금인출 기능은 단말기에 들어있는 모바일 칩(chip)을 적외선 센서가 부착된 ATM(Automated Teller Machine)에 가까이 함으로써 인증 기능을 수행하므로 현금카드 기능을 대신할 수 있다. 이와 같은 시스템은 ATM기와 마찬가지로 보안, 인증, 트랜잭션과 같은 여러 컴포넌트들로 구성된다. 따라서 시스템을 수행하기 위해서는 철저한 로깅, 트랜잭션, 인증과 같은 부가적인 기능들을 구현하여야 한다. 모바일 뱅킹 시스템은 휴대폰 사용자가 지속적으로 증가하면서 모바일 뱅킹이 지닌 편리성, 저렴성, 즉시성 등으로 은행은 물론 고객에게도 큰 이점을 주기 때문에 지속적으로 성장할 가능성이 높다[21]. 따라서, 보안 및 성능향상에 있어서 지속적인 유지보수 및 변경을 필요로 하므로 AOP 기법을 적용함으로써 생산성을 증가시킬 것으로 보고 본 논문에서 제시한 횡단관심사 명세 기법을 적용하였다.

본 논문에서 제안한 SCC는 이미 추출된 횡단관심사를 명세하는데 초점을 두고 있으므로 요구사항으로부터 얻게 되는 횡단관심사 추출은 기존의 유스케이스 기반 방법론을 적용한다. 또한, 결합점 정보는 제안한 순차다이어그램(sequence diagram)을 이용한 결합점 추출 방법을 기반으로 하고 교차점 선언은 본 논문에서 제시한 실행기반 교차점 선언 방법을 적용한다.

4.1 관심사 추출

그림 4와 그림 5는 MBS로부터 유스케이스 기반으로 추출한 횡단 관심 유스케이스의 일부를 도식화한 것이다. 인증은 시스템 관점에서 사용자가 맞는지 확인하는 과정이다. 권한인증은 사용자가 시스템 내에서 어떤 기능을 접근하는데 충분한 자격이 있는지를 결정한다. 이중 인증은 권한인증의 선결조건이 될 수 있으므로 두 개의 모듈은 항상 같이 병행적으로 구현하게 된다. 따라서, 이와 같은 횡단관심사를 애스펙트 코드로 구현하기 위해서는 하나 또는 그 이상의 횡단관심 유스케이스를 이와 대응되는 애스펙트 클래스로 코드 변환을 함으로써 모듈화를 수행한다. 본 논문에서는 모바일뱅킹의 여러 가지 횡단관심사 중에서 로깅, 인증, 권한인증을 명세 기법에 적용하였다.

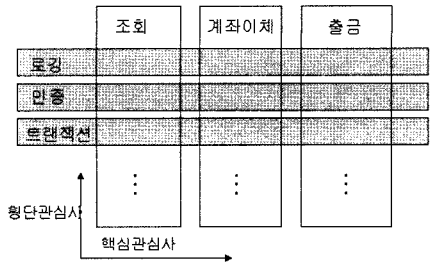

그림 4 유스케이스 기반 관심사 추출

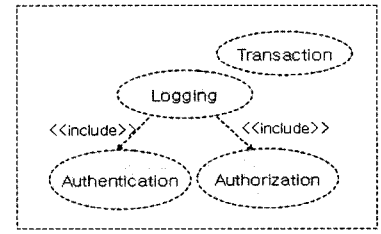

그림 5 MBS로부터의 추출한 횡단관심사 유스케이스 일부

표 4는 MBS로부터 추출한 전체 횡단관심사 목록 테이블(CCT)이다. CCT에는 요구사항으로부터 추출한 전체 횡단관심사의 이름, 특징, 관련된 클래스 이름, 그리고 우선순위 정보를 가지고 있다. 횡단관심사 명세 번호는 다음 단계에서 명세할 각각의 횡단관심사 명세서의 식별번호이다. 식별번호는 프로그램 이해 및 유지보수를 지원하는 핵심 자료의 역할을 하게 된다. 관련된 클래스 이름은 하나의 횡단관심사와 연관된 핵심 클래스 이름을 주며 초기 CCT 작성단계에서는 입력정보가 없지만 각 횡단관심사의 횡단관심사 명세 과정 Phase 2.2로부터 얻어진 정보를 기반으로 입력한다. CCT는 또한 애스펙트 코드간의 충돌을 방지하기 위하여 우선순위를 명시하였으며 이 우선순위는 4.2 횡단관심사 명세 과정에서 적용된다.

4.2 횡단관심사 명세 과정

표 4 전체 횡단관심사 테이블(CCT)

횡단관심사 이름	특징	관련 클래스 이름	횡단관심사 명세서 번호	우선순위
Authentication	프로그램 인증에 관한 횡단관심사	…	SCC1	1
Transaction	트랜잭션 기능 수행	…	SCC2	2
Security	보안 기능 수행	…	SCC3	-
…	…	…	…	-

표 5 로깅, 인증, 권한인증에 관한 SCC 1

<table>
<tr><td colspan="6">횡단관심사 명세서 번호 : SCC1</td></tr>
<tr><td colspan="6">횡단 관심사 이름 : 인증 (Authentication : Authorization, Logging 포함)</td></tr>
<tr><td colspan="2">단계</td><td colspan="3">추출 정보</td><td>상세 지침 및 산출물</td></tr>
<tr><td colspan="2">Phase 1</td><td colspan="3">1.1. 프로그램 로깅, 인증, 권한인증에 관한 횡단관심사이다.
1.2. 권한인증과 로깅을 같은 애스펙트 클래스에서 구현한다.
1.3. 다른 횡단관심사의 충돌에 대비해 가장 높은 우선순위를 갖는다.</td><td>CCT의 우선순위 참조</td></tr>
<tr><td rowspan="7">Phase 2</td><td>Phase 2.1
클래스 이름</td><td colspan="3">AuthenticationLog_Aspect.java</td><td>-</td></tr>
<tr><td>Phase 2.2
결합점 정보</td><td colspan="3">Account.makeDeposit()
Deposit.deposit()
Deposit.getBalance()
…</td><td>CCT의 관련 클래스 정보 입력</td></tr>
<tr><td rowspan="5">Phase 2.3
교차점 정보</td><td>accountProcedure</td><td>계좌정보 접근</td><td>Step 3</td><td rowspan="5">SCC1_ACRT
Account_CCRT
Deposit_CCRT
AccountDB_
CCRT
작성</td></tr>
<tr><td>authenticationOperation</td><td>인증관련 프로시저</td><td>Step 2</td></tr>
<tr><td>authorizationOperation</td><td>권한인증 프로시저</td><td>Step 2</td></tr>
<tr><td>loggedOperation</td><td>로깅 프로시저</td><td>Step 2</td></tr>
<tr><td>loggedOeration:BEFORE</td><td>로깅 충고</td><td>Step 1</td></tr>
<tr><td>Phase 3</td><td colspan="3">AuthenticationLog_Aspect.java 코드</td><td>애스펙트 코드</td></tr>
</table>

표 5는 MBS에서 추출한 CCT중 횡단관심사 번호 SCC1에 관한 명세 결과이다. SCC1은 인증 횡단관심사를 코드로 구현하기 위한 명세 결과로서 표 1에서 제시한 지침에 따라 작성된다.

Phase 1: 횡단관심사 명시

Phase 1에서는 횡단관심사의 특징을 기술한 것으로 인증은 사용자가 모바일뱅킹에 접속하여 인증을 받는 순간부터의 현재 상태를 저장하므로 로깅, 권한인증 기능을 추가하여 구현하도록 명시 하였다. 즉, 하나의 인증 애스펙트 클래스에 세 가지의 횡단관심사가 같이 구현하게 된다. 이 단계에서는 또한 CCT에 명시한 우선순위 정보를 참조하여 애스펙트 클래스 코드에 명시한다. 예를 들어, 인증, 트랜잭션, 보완 등 서로 다른 에스펙트 코드가 핵심 클래스의 같은 이름, 타입의 결합점에 접근하려 충돌이 발생할 경우를 대비하여 우선순위를 명시하여야 하는데 이 경우 인증에 우선순위를 할당한다. 인증을 구현한 애스펙트 코드는 권한인증, 로깅을 포함하고 있는데 로깅은 사용자가 모바일 뱅킹에 접속하여 PIN을 입력하는 순간 즉시 시작하게 되므로 가장 우선적으로 수행되어야 한다. 따라서, 로깅을 포함하고 있는 인증 애스펙트 코드에 가장 높은 우선순위를 할당한 것이다.

Phase 2: 횡단관심사 상세화

이 단계에서는 이전 단계에서 선언한 지침에 따라 횡단관심사를 코드 구현으로 변환하기 위한 설계 단계이다. 표 5의 Phase 2.1는 실제 구현코드를 위한 명세 단계로서 애스펙트 클래스 이름을 *AuthenticationLog_Aspect.java*로 명명하였다. Phase 2.2는 결합점 추출 단계로서 그림 6은 핵심관심사의 계좌조회 유스케이스를 순차다이어그램으로 표현하여 나타나는 결합점을 보여준다. 결합점은 실제 애스펙트 코드에서 구현한 충고가 실행되는 위치가 될 수 있으므로 명확한 결합점 추출을 수행하여야 한다. Phase 2.3은 Phase 2.2에서 추출한 결합점 중 인증, 로깅 횡단관심사 모듈이 수행될 결합점을 분류하여 교차점에서 선언한 결과이다. Phase 2.2에서 추출한 모든 결합점 중에서 충고가 동작하는 위치정보에 따라 교차점을 선언하는데 인증 횡단관심사일 경우 계좌조회, 이체를 위한 로그인이 시작된 순간부터 로깅, 인증 프로그램이 동작하게 된다. 따라서, 그림 6에서 실제 메소드 호출이 일어나는 Account 클래스의 경우

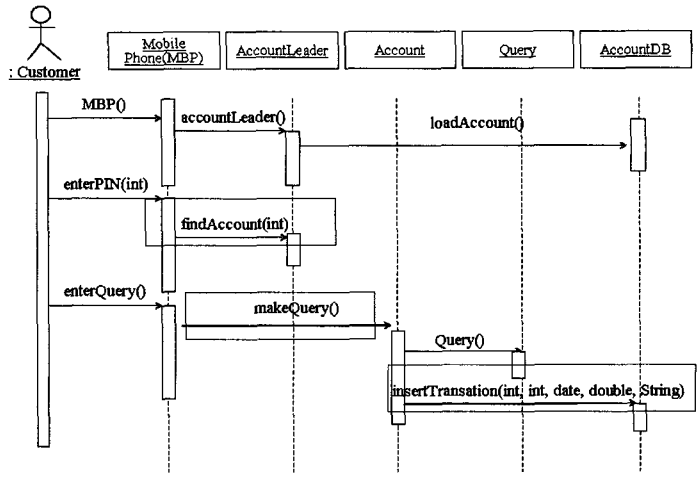

그림 6 MBS에서의 계좌조회 순차다이어그램

makeQuery(), Query(), insertTransaction()에서 횡단관심사 모듈이 수행된다고 볼 수 있다. 이 단계에서는 교차점에 선언된 결합점 정보 외에 결합점을 포함하고 있는 클래스 참조 테이블을 작성한다. 표 6과 표 7은 인증과 관련된 클래스 참조 테이블이다. 각각의 테이블 이름은 표 5의 명세서에 명시된 이름으로 구축되고 유지보수 및 재사용시 명세서를 기반으로 상호 참조된다. Phase 2.3은 Phase 2의 마지막 단계로 충고 설계 단계이다. 충고는 교차점이 지정한 결합점에서 수행할 행위 코드를 구현한다. 충고를 설계하고 이 단계에서는 교차점과 충고의 코드 이해를 위한 실행 절차를 간단히 명시한다.

Phase 3: 코드 구현

Phase 3은 실제 코드 구현단계로서 결과는 그림 7과 같다. Phase 1에서 Phase 2까지의 과정에서 얻어진 설계 정보를 기반으로 구체적인 애스펙트 코드를 구현한다. 애스펙트 클래스 이름은 AuthenticationLog_Aspect.java로 명명하였으며 declare precedence 명령은 phase 1에서 명시된 우선순위 정보를 기반으로 선언하였다.

그림 8은 횡단관심사 명세 기반의 프로그램 추적과정을 도식화한 것이다. 모든 추적을 위한 기본 산출물은 횡단관심사 명세 단계에서 만들어지고 요구사항 변경에 의한 프로그램 수정이 필요할 경우 횡단관심사 명세뿐만 아니라 관련된 코드 및 테이블 정보가 동시에 수정되어야 한다. 추적 방향은 양방향으로 가능하므로 프로그램 개발 단계 중 어느 단계에서 수정이 요구되더라고

표 6 인증에 관련된 클래스(SCC1_ACRT : Authentication)

ACRT Number : SCC1_ACRT			
애스펙트 클래스 이름 (횡단관심사 명세서 번호)	핵심 클래스 이름	결합점 정보	결합점 수
AuthenticationLog_Aspect (SCC 1)	Account.java	makeQuery() makeDeposit() makeTransfer() ...	3
	Query.java	Query()	1
	AccountDB.java	insertTransaction()	1

표 7 Account.java와 관련된 횡단 관심사(Account_CCRT)

CCRT Class Name : Account_CCRT		
핵심 클래스 이름	애스펙트 클래스 이름 (횡단관심사 명세서 번호)	횡단관심사 특징
Account.java	AuthenticationLog_Aspect.java (SCC 1)	인증
	Transaction_Aspect.java (SCC 2)	트랜잭션
	...	...

일관성 있는 프로그램 수정을 가능하게 한다.

```
public aspect AuthenticationLog_Aspect {
      declare precedence : AuthenticationLog_Aspect.*; //우선순위 설정

    public pointcut accountProcedure ()
      :call(void Account.makeQuery())
      :call(void Account.makeQuery())
      || call(void Account.makeDeposit())
      || call(void Account.makeTransfer())
       ........
    public pointcut authenticationOperation()
      :call(void Account.makeQuery())
       //인증이 수행되어야 할 결합점 위치 정보

    public pointcut authorizationOperation()
      :call(void Account.makeQuery())
       // 권한인증이 수행되어야 할 결합점 위치 정보

    public pointcut loggingOperation()
    : accountProcedure()
      || authenticationOperation()
      || authorizationOperation();

    before() : loggingOperation() {
      Signature ss = thisJoinPointStaticPart.getSignature();
       System.out.println("--" + ss.getName() + "--");
}
}
```

그림 7 AuthenticationLog_Aspect.java코드

5. 토의

본 장에서는 제시한 명세 기법을 비교 및 평가하기 위하여 기존에 제시된 개발 기법과 비교하고자 한다. 비교항목은 프로그램 개발 관점과 유지보수 관점에서 상세 분류하여 프로그램 개발 용이성, 코드 이해, 추적성, 확장가능성, 그리고 유지보수성으로 분류하였다. 평가 기준은 '높다', '낮다', '보통' 이렇게 세 가지로 나누었으며 개발기법에서 지원되는 기능이 있으면 '높다', 그렇지 않으면 '낮다', 그리고 지원되는 기능이 있지만 약하면 '보통'으로 명시하였다. JAsCo는 개발이 간단하고 코드 이해는 쉽지만 핵심 클래스와 애스펙트 클래스가 결합되어 있어 코드 변경시 핵심 클래스의 코드를 수정하여야 하는 단점을 갖고 있다. 반면, AML과 Composition 패턴을 이용한 개발 기법은 디자인 단계에서 애스펙트 구현에 필요한 정보를 추출하여 도식화하였지만 설계 단계에서 구현 단계로 변환하기 위한 일정한 지침이 없어 설계 정보만으로는 코드 개발이 어렵고 복잡하다. 이를 보완하여 본 논문에서는 개발 지침 및 횡단관심사 명세를 제공하여 설계 정보로부터 구현 단계 변환에 필요한 갭을 줄였다. 또한, 코드 이해 관점에서 지원되는 방법론이 미약하여 본 논문에서는 애스펙트 클래스와 핵심 클래스간의 참조 및 코드 이해를 위하여 CCRT와 ACRT를 구축함으로써 코드 이해성을 높여주었으며 나아가 이를 기반으로 추적 테이블을 구축하여 추적성 및 확장가능성을 높여줌으로써 유지보수성을 높여주었다.

6. 결론 및 향후 연구

AOP는 기존의 프로그래밍 기법에서 해결하기 보안, 트랜잭션, 인증, 로깅과 같은 부가기능에 대한 모듈화방

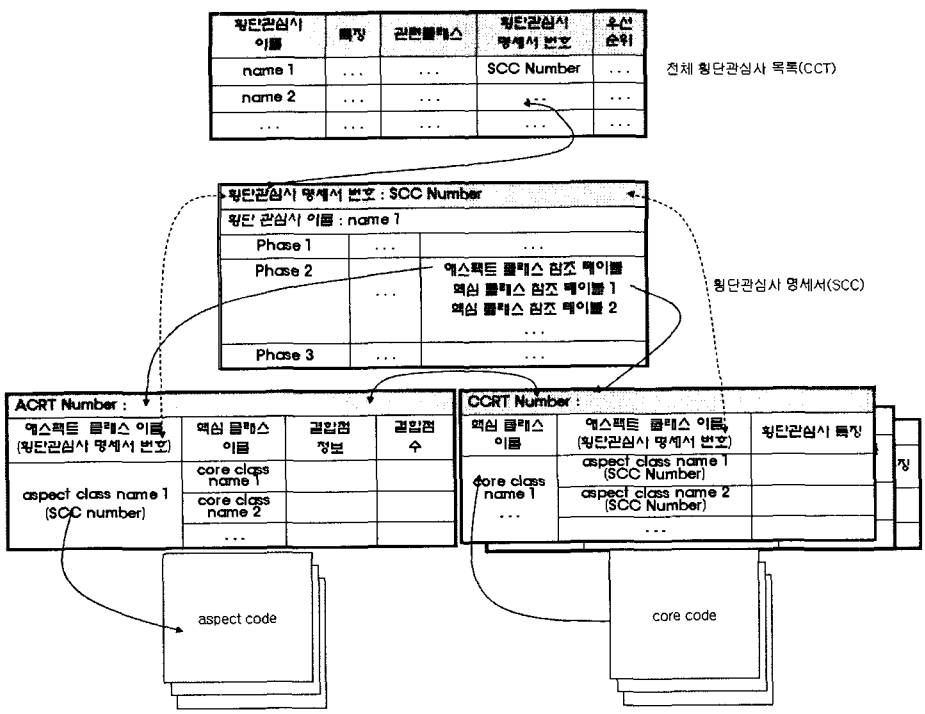

그림 8 횡단관심사 명세를 이용한 프로그램 추적도

표 8 타 개발기법과의 비교 및 평가

타 개발 기법 / 평 가	JAsCo	AML	Composition Patterns	AspectU	제안 기법
프로그램 개발 용이성	보통	보통	보통	낮음	높음
코드 이해	낮음	보통	보통	높음	높음
추적성	높음	낮음	보통	높음	높음
확장 가능성	낮음	보통	높음	낮음	보통
유지 보수성	보통	보통	높음	낮음	높음

법을 제공하였다. 또한, 기존의 개발 방법론과 AOP 기법을 상호보완적으로 수행함으로써 소프트웨어 성능을 향상시키고 유지보수에 많은 이점을 가져왔다. 초기 AOP는 프로그램 코드 구현에 초점을 둔 개발자 중심의 개발 방법이었다. 최근에는 요구사항 분석 단계에서부터 코드 구현 단계까지 소프트웨어 개발 공정을 적용하고자 다양한 방법에 대두되면서 AOP에 초점을 맞추고 있다. 하지만, 아직 명확한 개발 지침이나 표준이 마련되어 있지 않기 때문에 코드 구현의 어려움, 프로그램 흐름 파악의 어려움, 이로 인한 재사용성의 어려움이 제시되고 있다.

본 논문에서는 이와 같은 문제를 해결하기 위하여 횡단관심사 명세 기법을 제시하였다. 명세 기법은 관심사 명시, 관심사 설계, 그리고 코드 구현인 세 단계로 이루어져 있고 각 단계마다 상세 지침을 제공함으로써 요구사항 분석 단계에서 코드 구현단계까지 일관성 있는 프로세스를 유지하도록 지원하였다. 실행 기반 교차점 선언 방법을 제시하여 애스펙트 코드 구현이 쉽게 이루어지도록 지원하였으며 각 단계에서 얻어진 횡단관심사 명세서, 애스펙트 클래스 참조 테이블, 핵심 클래스 참조 테이블은 요구사항 변경으로 인한 프로그램 수정이 발생할 때 전체 프로세스간의 추적 자료로 제공된다. 이와 같이 횡단관심사 구현 지침 및 명세서는 AOP에서 해결하기 어려운 설계 단계에서 구현 단계까지의 갭을 줄여줌으로써 구현을 용이하게 지원하였고, 프로그램 이해의 어려움, 재사용의 문제점을 해결하였다.

AOP의 가장 큰 장점은 핵심 코드를 거의 수정하지 않고 시스템 전반에 영향을 주는 횡단관심 모듈을 독립성 있게 개발하고 자유롭게 삽입 및 제거가 가능한 방법이다. 따라서 AOP는 XP(eXtreme Programming) 방식의 개발자들에게 적합한 개발 기법에 될 수 있으므로 차후 연구에서는 XP 프로그래밍 기법에 필요한 AOP 개발 방법론을 제시하고자 한다.

참 고 문 헌

[1] Shari Lawrence Pfleeger, Software Engineering Theory and Practice, Prentice Hall, 2005.

[2] G. Kiczales and et al., "Aspect-Oriented Programming," In Proceeding of European Conference for Object-Oriented Programming, LNCS, Vol. 1241, pp. 220-243, 1997.

[3] G. Kiczales and et al., "An Overview of AspectJ," in Proceeding of European Conference for Object-Oriented Programming, LNCS, Vol.2072. pp. 327-352, 2001.

[4] 이준상, "미래소프트웨어 개발 기술: Apsect-Oriented Programming과 Subject-Oriented Programming," 정보처리학회지, Vol.10, No.5, pp. 94-101, 2003.

[5] Timo Aaltomen, Joni Helin, Mika Katara, Pertti Kellomaki., "Coordinating Aspects and Objects," Electronic Notes in Theoretical Computer Science 68, No.3, 2003.

[6] Georgia, S., Sergio, S., Paulo, B., and Jaelson, C., "Separation of Crosscutting Concerns from Requirements to Design: Adapting and Use Case Driven Approach," Aspect-Oriented Requirements Engineering and Architecture Design Workshop, pp. 93-102, 2004.

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

[8] Davy Suvee, Wim Vanderperren, and Viviane Jonckers, "JAsCo: An Aspect-Oriented Approach tailored for Component Based Software Development," AOSD Conference, pp. 21-29, 2003.

[9] Ossher, H, and P. Tarr, "Multi-Dimentional Separation of Concerns and The Hyper Approach," Software Architecture and Component Technology, Kluwer, 2001.

[10] R. Pawlak, L. Seinturier, L. Duchein, and G. Florin, "JAC: A Flexible Solution for Aspect-Oriented Programming in Java," In Proceeding of Reflection, LNCS, Vol.2192, pp. 1-21, 2001.

[11] Iris Gropher, Thomas Baumgarth, "Aspect-Orietned from Design to Code," in Proceedings of the Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design; AOSD, March, 2004.

[12] Siobhan Clarke, Robert J. Walker, "Composition Patterns: An Approach to Designing Reusable Aspects," AOSD, 2002.

[13] Jonathan Sillito, Christopher Dutchyn, Andrew David Eisenberg, and Kris De Volder, "Use Case

Level Pointcuts," European Conference on Object-Oriented Programming(ECOOP), 2004.
[14] Ivar, Jacobson., Pan-Weing, Aspect-Oriented Software Development with Use Case, Addison Wesley, 2005.
[15] Elisa Baniassad, Paul C. Clements, Joao Araujo and Ana Moreira, Awais Rashid, Bedire Tekinerdogan, "Discovering Early Aspect," IEEE Software, pp. 61-70, 2006.
[16] Ivar Jacobson, "Use Cases and Aspects-Working Seamlessly Together," Journal of Object Technology, Vol.2, No.4, 2003.
[17] Ramnivas, Laddad, AspectJ in Action. Manning, pp. 2005.
[18] The AspectJ Team, "The AspectJ Programming Guide," http://www.eclipse.org/aspectj/doc/next/progguide/index.html
[19] Renaud Pawlak, Houman Younessi, "On Getting Use Cases and Aspects to Work Together," Journal of Object Technology, Vol.3, No.1, January-February, 2004.
[20] 박옥자, 박종각, 유철중, 장옥배, "AOP 코드 이해를 지원하는 애스펙트 클래스 참조 테이블", 한국정보처리학회 춘계학술발표대회 논문집 제31권 제1호, 2006.
[21] 유 일, 신선진, 소순후, "모바일뱅킹서비스 수용요인에 관한 실증연구", Journal of Information Technology Application and Management, 제13권, 제2호, pp. 68-82, 2006.

박 옥 자

1997년 전북대학교 대학원 전산통계학과 졸업(이학석사). 1999년 8월~현재 전북대학교 대학원 컴퓨터 통계정보학과 박사수료. 관심분야는 컴포넌트 기반 소프트웨어 개발 방법론, 관점 지향 프로그래밍, 소프트웨어 재사용, 소프트웨어 유지보수 및 추적기법

유 철 중

1982년 전북대학교 전산통계학과 졸업(이학사). 1985년 전남대학교 대학원 계산통계학과 졸업(이학석사). 1994년 전북대학교 대학원 전산통계학과 졸업(이학박사). 1982년~1985년 전북대학교 전자계산소 조교. 1985년~1996년 전주기전여자대학 전자계산과 전임강사~부교수. 1997년~현재 전북대학교 자연과학대학 컴퓨터과학과 전임강사~부교수.관심분야는 소프트웨어 개발 프로세스, 소프트웨어 품질, 컴포넌트 소프트웨어, 소프트우어 메트릭스, 소프트웨어 에이전트, GNSS, GIS, 교육공학, 인지과학 등

장 옥 배

1973년 고려대학교 졸업(학사, 석사). 1988년 산타바바라대 대학원(Ph. D.). 1974년~1980년 죠지아 주립대. 오하이오 주립대 박사과정 수료. 1980년~현재 전북대학교 자연과학대학 전자정보공학부 교수. 관심분야는 소프트웨어 공학, 전산교육, 수치해석, 인공지능 등