관점 지향 개발 방법론에서 횡단 관심사 구현 명세 기법

박옥자° 박종각 최유순°° 유철중 장옥배
전북대학교, 원광대학교°°
cjpark@bsc.ac.kr° {canego, cjyoo, okjang}@chonbuk.ac.kr

On Specification of Crosscutting Concerns in AOSD

Oak-Cha Park°, Jong-Kock Park, Yoo-Sun Choi, Cheol-Jung Yoo, Ok-Bae Jang

Department of Computer Science, Chonbuk National University

요 약

프로그램 구현 방법에 편중되어온 기존의 AOSD 방법론이 최근에는 전체 개발 라이프사이클에서 초기 요구사항의 중요성이 강조되면서 요구사항 모델링, 분석, 설계분야에서 많은 연구가 진행되고 있다. 하지만, 요구사항 분석에서 추출된 관심사를 설계하여 구현 단계로 변환하는 과정에서 아직까지 상세화된 프로세스가 부족하다. 본 논문에서는 횡단 관심사 구현 명세 기법을 제시한다. 이 기법은 추출된 관심사를 구현 코드로 변환하기 위한 4단계의 프로세스로 구성되어 있다. 상세화된 명세 기법과 가이드라인은 AOSD에서 해결하기 어려운 설계에서 구현간의 갭을 줄여줌으로써 이해 및 유지보수성을 높여준다.

1. 서 론

관점지향 소프트웨어 개발 방법론(Aspect-Oriented Software Development, AOSD)는 횡단 관심사를 모듈화하기 위하여 객체지향 기술과 프로시저 프로그래밍의 기초 위에 이것들의 개념과 구조물을 확장한 것이다. 요구분석 단계에서 관심사를 추출하여 횡단 관심사(crosscutting concerns)는 에스팩트(aspect) 클래스로 작성하고 핵심 관심사(core concerns)는 Java와 같은 일반 객체지향 프로그래밍 언어로 구현한다[1].

AOP의 가장 큰 장점은 코드 구현에서 핵심 관심사를 구현한 객체지향 클래스와의 의존 관계가 없이 결합점(join points), 교차점(point cut), 충고(advice)라는 추가적인 구조를 이용하여 자체 에스팩트 클래스 내에서 구현하는 것이다. 이와 같은 특징은 객체지향 프로그래밍의 단점으로 지적되어온 횡단 관심사 문제를 해결하였다. 또한 관심사 분리를 통해 높은 모듈화와 재사용 및 확장성을 높여준다. 하지만, 아직 해결해야 할 몇 가지 문제점을 가지고 있다. 첫째, 초기 프로그램 구현이 어렵다. AOSD는 초기 요구분석 단계에서부터 횡단 관심사와 핵심 관심사로 분리하여 추출한다. 이 때 핵심 관심사는 일반 프로그래밍 언어로 구현한다. 반면, 횡단 관심사는 AOP 기법을 이용하여 프로그래밍 구현하는데 이 때 AOP에서 부가적으로 필요한 정보를 핵심 관심사에서 추출하여 연결할 수 있는 방법이 어렵다[2][3]. 둘째, 횡단 관심사를 구현한 에스팩트 클래스만으로 프로그램 흐름을 파악하기 어렵다. AOP에서만 제공하는 교차점, 결합점, 충고와 같은 구조는 핵심 클래스와의 낮은 결합도를 유지하게 하는 장치가 될 수 있지만 반면에 직조(weaving) 후 프로그램 실행 결과를 보기 전까지는 프로그램 흐름이 어떻게 이루어지고 어떤 클래스들과 연관되어 있는지 파악하기 어렵다. 셋째, 다른 클래스와의 참조 관계를 이해하기 어렵다. AOP로 구현한 클래스 관계에서 에스팩트 클래스는 핵심 클래스의 관계를 인지하지만 핵심 클래스는 애스펙트 클래스의 존재를 알 수 없도록 구현된다. 이 장점은 프로그램 모듈성을 높여주는 장점이 되면서 반면에 핵심 클래스만으로는 시스템 전체의 구조를 이해하기 어렵게 하는 단점이 되기도 한다. 마지막으로, 위의 세 항목이 명확하게 해결되지 않았을 때 프로그램 유지보수(maintenance) 및 추적(traceability)이 어렵다. AOP의 장점은 프로그램 유지보수 및 확장성을 높여준다고 제시되어 왔지만 초기 프로그램 개발 시 관심사 분리 및 구현 명세가 명확히 이루어지지 않는다면 오히려 프로그램의 복잡성만을 높여주는 결과를 가져올 수 있다.

본 논문에서는 이와 같은 문제점을 해결하기 위하여 횡단 관심사 구현 명세 기법(Specification of Crosscutting Concerns, SCC)을 제시한다. 횡단 관심사 구현 명세 기법은 요구분석 단계에서 추출한 관심사를 코드로 변환하는 과정에서 부족한 명세 및 가이드라인을 제시한다. 이 기법은 추출한 횡단 관심사를 하나의 클래스로 매핑하는 과정까지의 갭을 줄여줌으로써 프로그램 개발 및 이해성을 높여준다. 또한, 다른 핵심 클래스와의 연관성을 이해할 수 있고 추적성을 높여줌으로써 프로그램 유지보수 및 확장성을 높여준다.

본 논문에서는 초기 요구공학 단계에서 이루어지는 관심사 추출 방법은 다루지 않는다. 관심사 명시 방법은 기존의 개발 방법론을 이용하고 적용 사례에 사용되는 언어는 Java와 AspectJ를 이용하였다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문에서 제시한 SCC의 필요성 및 방법론을 설명한다. 3장에서는 ATM에 기법을 적용함으로써 특징을 살펴보고 4장에서는 결론 및 향후 연구를 제시한다.

2. 횡단 관심사 구현 명세 기법

초기 AOSD는 구현 방법에 편중되어 개발자는 오직 코드 중심의 관심사 명시 및 추출에 치중하였다. 최근에는 초기 요구공학의 중요성이 대두되면서 요구사항 모델

링, 분석 및 설계 분야에서 많은 연구가 진행되고 있다[4][5]. 하지만, 설계 단계에서 구현 단계로 변환하는 과정에서 몇 가지 어려운 문제점이 있다. 문제점을 해결하기 위하여 본 논문에서는 그림 1과 같은 SCC를 제시하고 표 1에 해결방안을 기술하였다.

그림 1에서 보면 왼쪽의 패키지 다이어그램은 추출한 횡단 관심사를 보여주고 있다. 각 횡단 관심사는 일반적으로 하나의 클래스로 매핑하여 구현한다. 이 때 클래스마다 정형화된 명세 기법을 제공함으로써 코드 구현까지 가이드라인을 제시한다. SCC Mgmt는 애스팩트로 구현한 클래스가 같은 이름, 타입, 초기값 등으로 인해 충돌이 발생할 경우 우선순위와 같은 에스팩트 관리가 필요하다. 이와 같은 에스팩트의 충돌(conflicts), 우선순위(priority), 정책(policy)등을 관리하기 위한 상세화 내용을 기술한다.

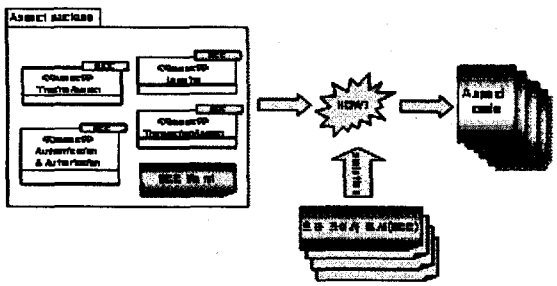

그림 1 SCC의 필요성

표 1은 본 논문에서 제시한 문제점과 해결방안을 제시하였다. 이 중 두 가지 항목은 SCC 가이드라인과 명세서를 제시함으로써 해결하고 ACRT(Aspect Class Reference Table)는 [6]에서 제시한 방법론으로 본 논문에서는 다루지 않는다.

문제점	해결방안
프로그램 구현	일정한 가이드라인 필요
	SCC Step 4
프로그램 흐름 파악	이해를 위한 명세서 필요
	SCC 명세
다른 클래스간의 관계	참조 테이블 필요
	ACRT
유지보수 및 추적	명세 및 map 필요
	ACRT

표 1 AOSD의 문제점 및 해결방안

그림 2는 SCC 프로세스이다. 각 프로세스는 4단계로 이루어져 있고 각 단계는 상세 가이드라인을 제시한다.

SCC 1 : 관심사 명시(identify Concerns)

요구분석 단계에서 추출한 관심사를 명시한다. 관심사 추출 기법은 기존에 많은 논문을 통해서 제시되었기 때문에 본 논문에서는 다루지 않았다. 단, 적용사례에서 일반적으로 사용되고 있는 유스 케이스 기반 관심사 추출 기법을 적용하였다[7][8][9][10].

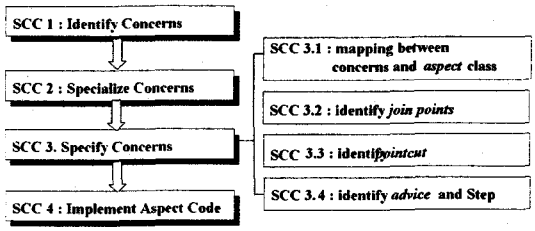

그림 2 SCC 프로세스

SCC 2 : 관심사 명세(Specialize Concerns)

SC 1에서 명시한 관심사를 가이드라인으로 제시한다. 이 단계에서는 관심사들 간의 우선순위, 정책 등을 다루고 구현 단계에서는 코드로 표현된다.

SCC 3 : 관심사 상세화(Specify Concerns)

본 논문의 핵심 분야로서 추출된 관심사를 코드로 매핑하는 단계이다. 이 단계는 다시 4 단계로 세분화된다.

SCC 3.1: mapping between concern and aspect class name

에스팩트는 횡단 관심사 구현 단위이다. 명시된 관심사는 일반적으로 하나의 클래스 형태로 매핑된다. 그 방법은 ConcernName_Aspect.java와 같은 형식의 명명규칙을 따른다.

SCC 3.2: Identify join points

SC 3.1에서 명시한 에스팩트에 구현될 결합점을 추출한다. AspectJ에서 결합점은 메서드나 생성자 호출, 메서드 또는 생성자 실행, 클래스 생성, 필드 참조, 핸들러 실행 등을 들 수 있다. 결합점은 직조시 에스팩트 클래스와 핵심 클래스의 실행 지점을 연결되므로 프로그램 이해 과정에서 가장 어려운 부분이기도 하다[11][12].

SCC 3.3 : identify pointcut

교차점은 결합점들을 선택하고 결합점의 환경정보를 수집하는 프로그램 구조물이다. 교차점은 에스팩트 클래스와 핵심 클래스간의 직조 규칙을 정하는 것으로 SCC 3.1에서 명시한 클래스에 교차점을 선언한다[13].

SCC 3.4 : Identify advice and execution Step

충고를 명시하고 프로그램 이해를 돕기 위해 실행 단계를 명시한다. 충고는 교차점에서 지정한 결합점에서 실행되어야 할 코드이다. 이미 전 단계에서 정해진 교차점에 충고를 추가함으로써 에스팩트 클래스의 골격 구조는 정해진다. 하지만, 에스팩트 코드의 단점은 여러 개의 교차점이 존재하면서 클래스내의 프로그램 로직을 이해하기 어려우므로 미리 실행 단계를 명시해주면 이해성을 높여준다.

SCC 4 : Implement Code

SCC 3에서 네 단계의 상세 명세 단계를 수행하면 간단한 골격 코드를 얻게 된다. 이 코드를 기반으로 하여 좀 더 상세화하여 프로그램 개발을 용이하게 할 수 있다.

SCC 구현 단계는 이와 같이 4단계로 구성된다. 각 단계에 해당하는 명세 템플릿과 이 있지만 이 장에서는 기술하지 않고 적용 사례에서 살펴보기로 한다.

3. 적용 사례

본 논문에서는 제시한 명세 방법론을 ATM에 적용하였다. ATM 시스템은 그림 3과 같은 클래스 다이어그램을 형성하는데 이 중 화살표 하단 부분이 추출된 관심사를 명시하였다.

관심사 추출 방법은 Jacobson의 유스 케이스 기반 방법을 이용하여 추출하였다[7].

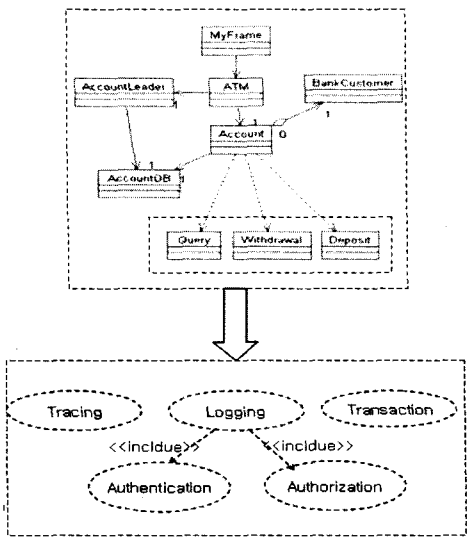

그림 3 ATM으로부터 추출한 횡단 관심사

SCC 1 : identify Concerns

요구분석 단계에서 추출한 관심사를 명시한다. 그림 3에서 추출된 관심사 명세서는 표 3과 같다. 관심사 이름은 네 가지로 분류하였고 우선순위는 여러 개의 관심사를 구현하였을 경우 실행 시 충돌이 발생할 수 있다. 이 문제점을 미리 방지하기 위하여 관심사 우선순위를 명시해 주는 것이다. 우선순위 명시는 프로그램 특성에 따라 다를 수 있고 삭제될 수도 있다. 이 결과는 SC 2의 상세화 단계에 적용된다.

SCC 2 : Specialize Concerns

SCC 1에서 추출한 관심사를 상세화한다. 에스팩트가 같은 이름, 타입, 초기값 등으로 인해 충돌이 발생할 경우 우선순위와 같은 에스팩트 관리가 필요하다. 이와 같은 에스팩트의 충돌, 우선순위, 정책 등을 관리하기 위한 상세화 내용을 기술한다. 이 명세 결과는 그림 1에서 SCC Mgmt 산출물로 사용되기도 한다. ATM 예제에서는 로깅(logging) 관심사 구현시 인증(authorization)과 권한 인증(authentication)을 함께 구현한다는 내용을 가이드라인 1에서 제시하고 있다[표 4 참조]. 또한, 세 개의 관심사가 충돌하였을 경우 어떤 관심사의 메소드나 초기값이 우선권을 갖는지 명시한다[12]. 그 결과는 표 4에 나타난다.

표 3 Identified concerns from ATM

Concern Name	Description	priority
Logging	Logging message on the current operation	MAX
Tracing	Tracing the current operation	LOW
Transaction	The fundamental unit of the work for which consistency of the system before and after the execution in maintained	HIGH
Authentication Authorization	Validation the user Validation how much sufficiently the authorized user is qualified for a resource	MAX

SCC 3 : Specify Concerns

관심사를 명시하는 것으로 4단계로 이루어져 있다. 그림 4는 SCC 3.2에서 필요한 결합점 추출에 사용된 순차 다이어그램이다.

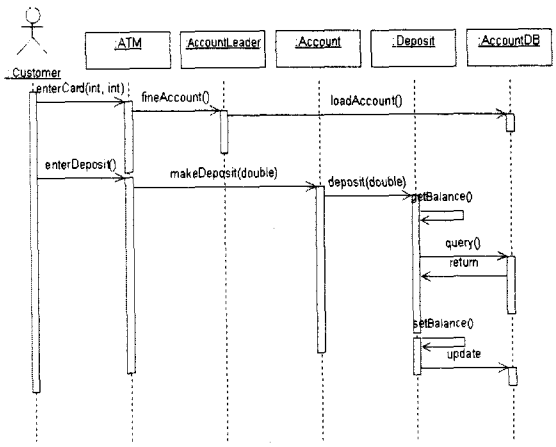

그림 4 Deposit에 해당하는 순차 다이어그램

그 외 SCC 3.1에서 SCC 3.4까지의 명세 결과는 표 4에 기술되어 있다.

표 4는 단계 절차에 따라 산출된 로깅 에스팩트 SCC이다. 로깅 관심사 클래스 이름은 AuthLogging_Aspect.java이고 SCC 3.2에서는 관련된 클래스와 결합점을 보여준다. SCC 3.3과 SCC 3.4는 결합점이 실행될 교차점을

표 4 로깅, 인증, 권한인증에 관한 SCC

<table>
<tr><th colspan="5">Concern name : logging, Authorization, Authentication</th></tr>
<tr><th colspan="2">Phase</th><th colspan="2">Description</th><th>Guideline</th></tr>
<tr><td colspan="2">SCC 1</td><td colspan="2">로깅, 인증, 권한인증, 트랜잭션, 추적</td><td>두 개의 횡단 관심사를 하나의 클래스에 구현 명시</td></tr>
<tr><td colspan="2">SCC 2</td><td colspan="2">1. 프로그램 효율성을 높이기 위한 인증, 권한인증은 로깅 관심사 구현 시 같은 클래스에 명시한다.
2. 로깅, 인증 두 개의 관심사가 충돌이 발생했을 때 로깅을 가장 우선적으로 설정한다.</td><td>제약사항 명시</td></tr>
<tr><td rowspan="11">SCC 3</td><td>SCC 3.1</td><td colspan="2">AutoLogging_Aspect.java</td><td>에스팩트 클래스 이름 명시</td></tr>
<tr><td>SCC 3.2</td><td>Account.makeDeposit()
Deposit.deposit()
Deposit.getBalance()
...</td><td>Account.java
Deposit.java
AccountDB.java</td><td>핵심 클래스의 결합점을 추출하고 관련된 클래스를 명시한다.</td></tr>
<tr><td rowspan="4">SCC 3.3</td><td>accountProcedure</td><td>계좌 정보 접근</td><td rowspan="4">결합점의 동작을 구현할 프로시저별 교차점 선언</td></tr>
<tr><td>authenticationOperation</td><td>권한 접근 관련 프로시저</td></tr>
<tr><td>authorizationOperation</td><td>인증 관련 프로시저</td></tr>
<tr><td>loggedOperation</td><td>로깅 프로시저</td></tr>
<tr><td rowspan="5">SCC 3.4</td><td>accountProcedure</td><td>Step 2-3</td><td rowspan="5">충고까지 추가하고 프로그램 흐름을 이해할 수 있도록 단계를 명시한다.</td></tr>
<tr><td>authenticationOperation</td><td>Step 2-2</td></tr>
<tr><td>authoorizationOperation</td><td>Step 2-1</td></tr>
<tr><td>loggedOperation</td><td>Step 2(2-1, 2-2, 2-3)</td></tr>
<tr><td>loggedOeration:BEFORE</td><td>Step 1</td></tr>
<tr><td>SC 4</td><td colspan="3">Implement Code</td></tr>
</table>

기술하였고 교차점과 충고간의 실행 흐름을 순차적으로 명시하였다. 따라서, 이 명세서를 통해서 로깅 클래스가 어떤 절차에 의해 구현되고 실행되는지 예측할 수 있으며 다른 클래스간의 참조 관계도 테이블을 통해 이해할 수 있다.

SCC 4 : Implement Code

표 5는 표 4의 명세서를 통해 얻어진 골격코드의 단편이다. 오른쪽 주석으로 표시된 부분은 표 4 SCC와 코드간의 매핑 결과를 보여준다.

그림 5는 각 단계별 추적 흐름도를 보여준다. 각 단계는 하위 단계로 내려갈수록 상세화되면서 추적 가능하므로 프로그램 수정 및 확장이 필요한 경우 추적 흐름도와 SCC를 참조함으로써 프로그램의 수정이 가능하다.

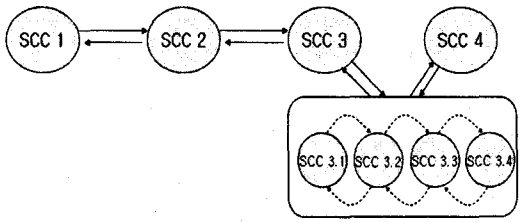

그림 5 추적 흐름도

표 5 AuthLogging_Aspect.class 코드

```
public Aspect AuthLogging_Aspect {                    // SCC 3.1
        declare precedence : AuthLogging_Aspect.*; // SCC 3.2

    public pointcut accountProcedure ()          // SCC 3.3
      :call(void Account.makeDeposit())              // SCC 3.2
      || call(void Deposit.query())               // SCC 3.2
          ...
      // implementation code
    public pointcut authenticationOperation()  // SCC 3.3
       //implementation code

    public pointcut authorizationOperation()   // SCC 3.3
       //implementation code

    public pointcut loggingOperation() {         // SCC 3.3
    : accountProcedure()
      || authenticationOperation()
      || authorizationOperaton();

    before() : loggingOperation() {             // SCC 3.4
    //implementation code                          // SCC 3.4
}
```

4. 결론 및 향후 연구

본 논문에서는 AOSD에서 횡단 관심사 구현 명세 기법을 제시하였다. 이 기법은 추출된 관심사를 구현 코드로 변환하기 위한 4단계의 프로세스로 구성되어 있다. 상세화된 명세 기법과 가이드라인은 AOSD에서 해결하기 어려운 설계에서 구현간의 갭을 줄여줌으로써 다음과 같은 장점을 가져온다. 첫째, 요구분석에서 얻어진 관심사를 에스팩트 코드로 구현하기까지 일관성 있는 가이드라인을 제공한다. 둘째, 에스팩트 클래스와 핵심 클래스간의 추적성을 제공하여 프로그램 수정 및 유지보수를 가능하게 하고 설계 가공물과 코드간의 이해성 증가시킨다. 셋째, 프로그램 수정 및 확장이 필요한 경우 SCC 명세서를 기반으로 참조함으로써 보다 일관성 있는 수정이 가능하다. 차후 연구에서는 명세 기법이 코드 및 요구 분석과 같은 단계별 개발 방법과 상호 보완하면서 동시에 진행되도록 하는 자동화된 명세 툴을 개발하고자 한다.

참고 문헌

1. G.Kiczales and et al." *Aspect-Oriented Programming.*", In Proceeding of European Conference for Object-Oriented Programming, Lecture Notes in Computer Science, Vol. 1241. Springer-Verlag, (1997) 220243
2. Timo Aaltonen, Joni Helin, Mika Katara, Pertti Kellomaki., "*Coordinating Aspects and Objects*", Electronic Notes in Theoretical Computer Science 68 No. 3(2003)
3 Davy Suvee, Wim Vanderperren, Viviane Jonckers, "*JAsCo: An Aspect-Oriented Approach tailored for Component Based Software Development.*" AOSD Conference(2003) 21-29
4. Elisa Baniassad, Paul C. Clements, Joao Araujo and Ana Moreira, Awais Rashid, Bedire Tekinerdogan, "*Descovering Eearly Aspect.*" IEEE Software(2006) 61-70
5. Georgia, S., Sergio, S., Paulo, B., Jaelson, C, "*Separation of Crosscutting Concerns from Requirements to Design: Adapting and Use Case Driven Approach.*" Aspect-Oriented Requirements Engineering and Architecture Design Workshop(2004) 93-102
6. 박옥자 외, "AOP 코드 이해를 지원하는 애스펙트 클래스 참조 데이블", 한국정보처리학회 춘계학술발표대회 논문집 제31권 제1호, 2006. 5
7. Ivar, Jacobson., Pan-Wei, Ng, "*Aspect-Oriented Software Development with Use Case.*" Addison Wesley(2005)
8. Isabel Brito, Ana Moreira, "*Integrating the NFR framework in a RE model.*" Aspect-Oriented Requirements Engineering and Architecture Design Workshop(2004) 28-33
9. Chethana, K., Armin, E., "*Aspect-Oriented Requirements Engineering for Software Product Lines.*" ECBS(2003). Proceeding of the 10thIEEE International Conference(2003) 673-676
10. Elisa, B., Siobhan, C., "*Finding Aspects in Requirements with Theme/Doc.*" Aspect-Oriented Requirements Engineering and Architecture Design Workshop(2004) 15-22
11. The AspectJ Team, "The AspectJ Programming Guide", 2001
12. Ramnivas, Laddad, "AspectJ in Action." Manning(2005)
13. G. Kiczales et al., "*An Overview of AspectJ*" ECOOP(2001). Object Oriented Programming. Lecture Notes in Computer Science, Vol. 2072. Springer-Verlag, (2001) 327353