AOP를 적용한 프로덕트 라인 가변기능의 구현

허 승 현[†] · 최 은 만[††]

요 약

소프트웨어 개발 방법론은 자원의 재사용을 통하여 생산성을 향상시키고, 제품을 만들어 시장에 배포하기까지 소요되는 시간인 time-to-market을 감소시킬 목적으로 발전되어왔다. 이러한 방법 중의 하나인 프로덕트 라인을 구현하는 기존의 방법은 중심 자원에 대한 간섭이 심하여 가변적 기능과의 조합 절차에서 많은 비용을 요구하므로 기대만큼의 효과를 얻기 힘든 상황이다. 본 논문에서는 소프트웨어 프로덕트 라인의 조합 프로세스를 개선한 방법으로 관점 지향 프로그래밍(Aspect-Oriented Programming)을 도입한다. AOP의 문법 요소인 결합점(join point)과 교차점(pointcut), 충고(advice)를 이용하여 중심 자원과 가변적 기능을 코드 변경 없이 조합하는 방법을 소개하고, 간단한 시스템을 사례로 들어 관점 지향 개념을 적용하여 요구를 분석 하고 UML로 설계한다. 설계 단계에서 도출된 가변 기능은 구현 단계에서 관점 지향 언어인 AspectJ를 이용하여 중심 기능과 결합한다. 이 실험을 통하여 효율적인 프로덕트 라인의 구현을 보이고, 유용성과 실용성을 입증한다.

키워드 : 재사용, 소프트웨어 프로덕트 라인, 관점 지향 프로그래밍, 결합점, 교차점, 충고, AspectJ

Implementation of Software Product-Line Variabiliy Applying Aspect-Oriented Programming

Seung-hyun Heo[†] · Eun Man Choi[††]

ABSTRACT

Software development methodology has been developed for satisfying goals of improvement in productivity and reduction in time-to-market through the reuse of software assets. The current methods that implement software product-line, one of software development methodologies, interfere massively with the core assets, which require high cost in assembly level reducing the effectiveness. In this paper, we introduce Aspect-Oriented Programming (AOP) as a method for improving assembly process in software product-line. The method that assembles core assets and variabilities is described by grammar elements such as Join point, pointcut and advice without code-change. We analyze requirements of a mini-system as an example adapting AOP and design using UML. Our study implements the variabilities, which are from design stage, using an Aspect Oriented Programming Language, AspectJ and prove usability and practicality by implementing the proposed idea using an Aspect-Oriented Programming Language, AspectJ.

Key Words : Resue, Software Product-Line, AOP, Join Point, Pointcut, Advice, AspectJ

1. 서 론

소프트웨어 개발에서 가장 중요하고 결정적인 문제인 생산성의 향상과 time-to-market 감소를 해결해 줄만한 기술이 재사용과 버전 관리이며 이를 통합한 패러다임이 소프트웨어 프로덕트 라인이다. 소프트웨어 프로덕트 라인을 제품 개발에 적용해 효과를 얻기 위해서는 제품 개발 초기 단계부터 요구사항을 제품 계열의 공통적 요구사항과 가변적 요구사항으로 구분하고 설계, 구현하여야 한다.

프로덕트 라인에 대한 연구는 프로덕트 패밀리를 설계하고 개발하는 방법으로부터 프로덕트 라인 개발과 관리에 사용되는 도구에 대한 연구, 가변적인 부분의 표현과 이를 구현하는 문제 등을 다루고 있다. 이 중에서 가변적인 부분에 대한 설계 표현과 구현이 프로덕트 라인에서는 중요한 이슈이며 프로덕트 라인의 요구 분석을 위해 관점 지향 프로그래밍 개념의 적용을 시도한 연구와[3] UML의 확장에 의한 설계 표현 방법이 제안되었다[9, 11].

구현 단계에서는 프로덕트 라인의 가변적인 부분을 구현하기 위해 사용할 수 있는 방법들의 비교에 관한 연구[2]가 있었으나 동적 연결과 상속, 조건부 컴파일, AOP 등 비교되는 기법 들의 분류 기준이 불분명하여 실질적 비교가 불가능하였으며, AOP의 코드 수준에서의 용이성을 증명하지 못하였다. 다음으로 컴포넌트 라이브러리의 자원(asset)을 관리하고 이를 이용하여 같은 계열의 제품을 개발하려는 시도[4, 10]가 있었으나 XML 등 원시 코드 이외의 별개의 언어

※ 본 연구는 동국대학교 논문 게재 장려금 지원으로 이루어졌음.
† 정 회 원 : 동국대학교 대학원 컴퓨터공학과 석사과정
†† 정 회 원 : 동국대학교 컴퓨터공학과 교수(교신저자)
논문접수 : 2006년 2월 21일, 심사완료 : 2006년 6월 7일

가 자원의 관리 및 생성을 위해 요구되었다.

프로덕트 라인에서는 공통적 기능을 구현하는 범용 컴포넌트와 가변적 기능을 구현하는 컴포넌트의 조합이 얼마나 효율적인가에 따라 생산성이 좌우된다. 따라서 자원 관리를 위한 별도의 비용을 필요로 하고, 자원의 개발자와 이용자가 다를 경우 자원 코드의 이해를 요구하는 기존의 자원 관리 및 조합 방법은 프로덕트 라인을 도입함으로써 얻게 되는 이익을 감소시킨다.

본 논문은 기존에 개발 된 범용적 자원과 가변적 기능을 구현한 컴포넌트 간의 조합 과정에 있어 이들 부품 컴포넌트에 대한 개입을 최소화 하고자, 관점 지향 프로그래밍을 도입하였으며, 아교(glue) 역할로써 AOP의 애스펙트(aspect)를 이용하였다. 변경 가능 기능(variants)의 구현은 애스펙트의 문법 요소인 충고(advice)를 통해 공통 기능을 구현한 중심 자원(core asset)과 결합된다. 가변 컴포넌트를 선택적(optional) 기능과 대안적(alternative) 기능으로 구분하여 AOP의 이전(before) 충고와 이후(after) 충고, 대체(around) 충고를 이용할 경우 프로덕트 패밀리들은 중심자원의 표현과 이들의 어떠한 코드 변경도 없이 결합한다. 자원의 코드 변경 없이 다음 버전의 제품을 완성할 수 있다면, 자원의 관리와 이용에 있어 매우 효율적일 것이다.

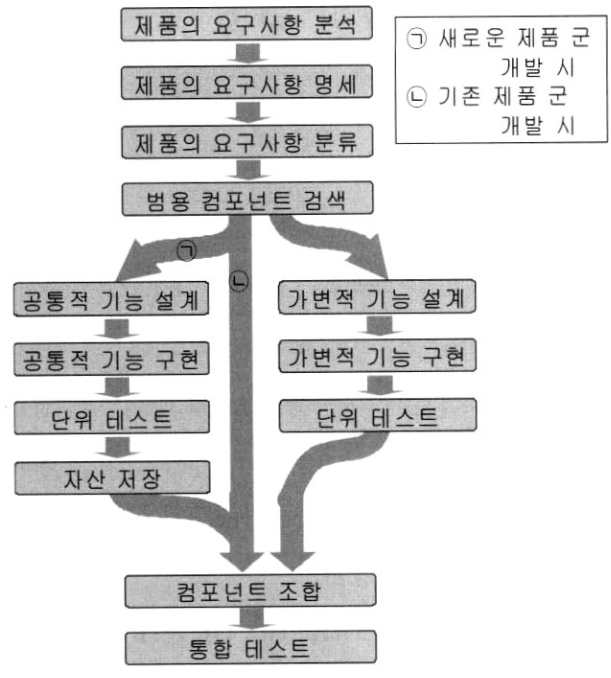
본격적인 연구결과 소개에 앞서 2장과 3장에서는 소프트웨어 프로덕트 라인과 관점 지향 프로그래밍을 소개하고 그들을 적용시킨 성공사례를 기술한다. 4장에서는 주요 아이디어를 설명하며, 5장에서는 작은 규모의 프로젝트를 예로 들어 요구 분석 단계부터 구현 단계까지를 보였다.

2. 프로덕트 라인

2.1 프로덕트 라인 개발 프로세스

소프트웨어 프로덕트 라인이란 같은 제품군에 대해 범용적으로 사용하기 위해 만든 공통적인 자산을 일컫는다. 단일 어플리케이션이나 임베디드 소프트웨어의 다변화와 함께 기본적인 기능에 변경을 가하여 새로운 버전의 제품을 출시하는 경우가 많아졌다. 이에 따라 공통적인 기능이나 특징을 갖는 제품에 대하여 범용적 컴포넌트를 제작하고 변경 가능한 기능의 컴포넌트들을 선택 조합하여 하나의 제품을 완성하게 된다. 재사용을 지원하는 핵심 자산을 구현하기 위해서는 개발 초기 단계에서부터 범용적 컴포넌트를 완성하는데 초점을 두어야 하며, 범용적인 부분과 가변적인 부분에 대한 구분이 뚜렷해야 한다. 프로덕트 라인을 적용한 소프트웨어의 개발 프로세스는 다음과 같다.

먼저 제품 계열의 요구사항을 분석하고 산출된 요구사항에 대해서 공통적인 기능과 가변적인 기능을 구분한다. 공통적인 기능에 대하여 기존에 개발한 제품의 자산 중에 이용 가능한 자산이 있을 경우 ㉡ 개발 주기를 적용하여 가변적 기능과 조합하고, 그렇지 않을 경우는 ㉠ 개발 주기를 적용하여 컴포넌트의 조합 단계 전에 범용 컴포넌트를 구현하고 테스트하고 자산관리 시스템에 저장하여 관리하게 된다. (그림 1)과 같은 프로덕트 라인을 적용한 개발 프로세스를 따를 경우 기존에 개발한 제품군에 대해 이미 만들어진 범용 자산을 이용함으로써 개발 기간이 단축되고 개발 비용이 절감되며, 신뢰성 있는 제품을 얻게 된다.



(그림 1) 프로덕트 라인 적용 소프트웨어 개발 프로세스

스웨덴의 CelsiusTech Systems는 백만에서 백오십만 줄의 Ada 코드로 작성된 군함의 통합 시스템을 개발하는데 프로덕트 라인을 적용함으로써 7건의 프로젝트에 대해 70~80%의 컴포넌트를 재사용하였고 이로 인해 8배에 달하는 생산성 이익을 달성하였다는 사례를 보고한바 있다[6].

2.2 프로덕트라인에서 변경 가능 부분의 표현 방법

프로덕트 라인의 아키텍처로부터 제품의 아키텍처를 쉽게 얻기 위해서는 제품의 공통적 기능과 가변적 기능이 정확하게 표현되어야 한다[11]. 가변 기능을 표현 하는 수단으로 여러 가지가 연구되었으나 본 장에서는 UML을 이용하는 방법과 XML을 이용하는 방법에 대해 소개한다.

가변 기능의 표현 방법 중 첫 번째로 UML을 이용하여 표현하는 방법이 있다. UML을 이용한 표현법은 소프트웨어 개발 프로세스 중 설계 단계에서 활용할 수 있는 방법이다. UML을 포함한 기존의 모델링 언어는 프로덕트 라인의 설계를 지원하지 않는다. 그러나 UML 2.0이 발표되면서 UML 2.0을 확장하여 가변 기능을 표현하려는 연구가 진행되었다[11]. 이 연구에서는 UML 2.0에서 새롭게 컴포넌트와 커넥터(connector) 등의 요소를 제공하며 이러한 요소들을 이용하여 아키텍처의 가시성을 높이고, 동적인 관점과 정적인 관점으로 설계함으로써 아키텍처의 이해를 쉽게 하려는 노력이 있었다.

또 다른 가변 기능의 표현 방법으로 XML을 이용하는 방법이 있었다. XML을 이용한 방법은 소프트웨어의 구현 프

로세스와 관계가 있다[4]. 각각의 기능들을 x-frame 이라는 단위로 표현하고 컴포넌트 별로 분류하여 x-framework을 구성한다. 이 설계를 바탕으로 코드에 XML의 확장성을 이용한 XVCL(XML-based Configuration Language) 코드를 삽입함으로써 자원 관리에 이점을 기대할 수 있다.

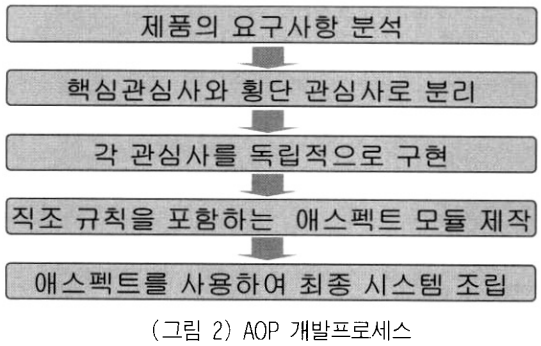
3. 관점지향 프로그래밍 (Aspect-Oriented Programming)

3.1 관점 지향 프로그래밍의 개발 프로세스

관점 지향 프로그래밍(Aspect-Oriented Programming)이란 1996년 Gregor Kiczales가 처음 만든 용어로, 시스템의 기본적이고 대표적인 업무처리 기능을 나타내는 핵심 관심사와 여러 개의 모듈에 걸치는 사용자 인증, 로깅과 같은 횡단(cross-cutting) 관심사를 구분함으로써 시스템의 설계와 구현의 복잡성을 감소시키고 모듈화를 이루려는 패러다임이다[5].

(그림 2)는 AOP의 개발 프로세스를 보여준다. 제품의 요구사항 분석이 끝나면 요구사항을 핵심(core) 관심사와 횡단(cross cutting) 관심사로 분리한다. 관심사 중에서 핵심 관심사는 기존의 구현 기술로, 횡단 관심사는 AOP기술을 이용하여 구현한다. 마지막으로 직조 규칙을 포함하는 애스펙트 모듈을 작성, 조립하면 프로젝트가 완성된다.

AOP는 IBM의 애플리케이션 서버 Websphere에 적용 되었다. 대표적인 횡단 관심사인 로깅, 트레이싱, FFDC(First Failure Data Capture), 성책 시행에 적용하여 품질과 서비스의 향상을 보였으며, Websphere에서 Enterprise Java Bean(EJB) 컴포넌트를 분리해내는 리팩토링을 성공적으로 수행하였다[5].



(그림 2) AOP 개발프로세스

3.2 AspectJ 와 AspectC++

AspectJ는 Java 언어에 관점지향 개념을 추가하여 확장한 언어이다. AspectJ의 횡단 관심사를 표현하기 위한 문법 요소로는 실행 시간에 프로그램의 행위를 추가하거나 변경시키는 동적 횡단 요소와 클래스나 인터페이스와 같이 프로그램의 정적 구조의 변경에 관여하는 정적 횡단 요소가 있다. 그 중 본 논문을 전개하는데 핵심적인 동적 횡단 요소를 소개한다[5].

1) 결합점(join point) - 프로그램의 실행 과정에서 구별 가능한 프로그램의 한 지점. 메서드 호출부나 객체의 멤버에 값을 할당하는 부분이 될 수 있다.
2) 교차점(pointcut) - 결합점에 대한 직조 규칙을 형성하기 위해 애스펙트내에 포함시키는 구조물.
3) 충고(advice) - 교차점에서 지정한 결합점에서 실행되어야 할 코드. 결합점 이전에 실행되는 이전(before) 충고와 결합점 이후에 실행되는 이후(after) 충고, 결합점을 대체하여 실행되는 대체(around) 충고가 있다.

다음은 프로그램 내에서의 결합점과 교차점, 충고의 예를 보인 것이다.

Hello 클래스 내에서 sayHello() 메서드를 호출하고 있으며, 이 부분은 메서드 호출에 의한 결합점이 된다. (그림 4)에서는 애스펙트 내에서 결합점을 명시하고 있으며 이것이 교차점이다. 이후(after) 충고로써 결합점의 연산 뒤에 처리를 추가하고 있다. 위의 클래스(class)와 애스펙트를 AspecJ 컴파일러로 동시에 컴파일 하는 경우 컴파일과 함께 클래스와 애스펙트 간의 직조(weaving)작업이 이루어진다.

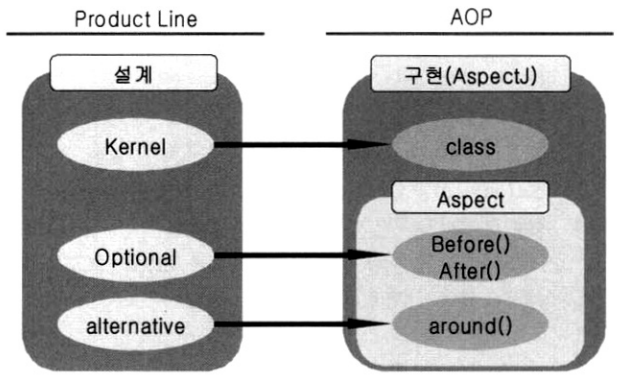
(그림 5)는 애스펙트의 역할을 도식화 한 것이다. 제품의 공통 기능과 가변 기능은 모두 클래스로 구현되어 자원을 형성한다. 이후 애스펙트는 이전 충고(before())와 이후 충고(after())라는 문법 요소를 이용하여 가변 기능 중에서도 선택적(optional) 기능을 호출하게 된다. 그리고 대안적(alternative) 기능은 대체 충고(around())를 이용하여 기존의 기능을 대체하도록 되어있다. 이는 가변점에 대한 상호 작용의 모듈화를 극대화하는 효과를 제공함으로써 공통기능과 가변 기능의 조합 프로세스를 간단하고 명백하게 한다. 이러한 조합 프로세스의 변경은 프로그래머에게 기존 자원에 대한 이해의 부담을 감소시킨다.

```
public class Hello{
        void sayHello(){
                System.out.println("Hello!");
        }
        public static void main(String[] args){
                Hello h = new Hello();
                h.sayHello();                    //결합점
        }
}
```

(그림 3) 결합점

```
public aspect HelloAspect{
pointcut message():call(void Hello.sayHello(..));//교차점
        after():message(){                       //충고
                System.out.println("How are you?");
        }
 }
```

(그림 4) 교차점과 충고



(그림 5) 제품라인의 관점 지향적 구현

AspectJ는 위에서 소개한 요소들 이외에 정적 횡단 요소를 가지고 있으며, C++ 언어를 확장한 AspectC++ 역시 유사한 구조물들로 이루어져 있다. 현재 AspectWertz, JBossAOP 등 여러 AOP 지원 도구들이 개발되어 있다.

4. 관점 지향 프로그래밍을 적용한 중심 자원과 가변 기능의 조합 방안

프로덕트 라인은 개발 초기부터 최대한 범용성을 고려하여 설계되고, 구현되어야 한다. 이는 범용적으로 개발된 프로덕트 라인의 자산을 재사용하는 경우, 가변적인 기능과의 조립이 최소한의 노력과 자산에 대한 최소한의 간섭으로 이루어지게 하기 위해서이다. 또한 핵심 자산 관리가 깨끗하게 이루어지도록 할 뿐 아니라 소프트웨어 프로덕트 라인이 갖는 궁극적인 목적이라 할 수 있는 time-to-market의 감소와 생산성의 향상에 직결되는 문제이기도 하다.

본 논문에서는 이러한 목적을 위한 직접적인 방법으로써 자산에 대한 코드의 변경 없이 가변적인 기능과의 조합을 이룰 수 있는 방안에 대하여 연구하였으며, 관점 지향 프로그래밍이 프로덕트 라인 개발 방법론을 구체화 할 수 있는 수단임을 설명한다.

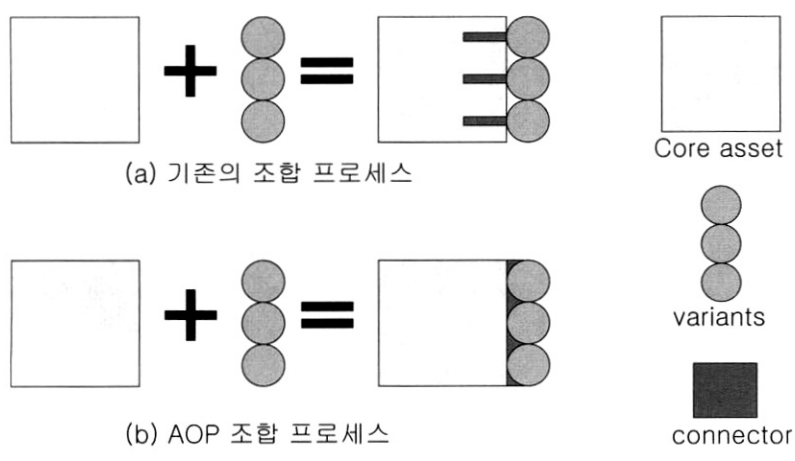
(그림 6)은 프로덕트 라인을 이용한 개발 프로세스 중 중심 자원과 가변적 기능을 조합하는 방법에 대해 기존의 방법과 논문에서 설명하는 방법의 차이점을 도식화하여 보여준 것이다.

첫 번째 그림은 기존의 조합 프로세스이다. 중심 자원과 가변적 기능을 조합하기 위해 연결부가 중심 기능 안에 삽입되어 있는 것을 볼 수 있다. 일반적으로 중심 자원에서 가변적 기능이 구현된 다른 컴포넌트의 연산을 호출하는 부분이 이에 해당한다. 두 번째 프로세스는 중심 자원에 어떠한 침가나 변경도 가해지지 않는다. 가변적 기능의 컴포넌트에도 마찬가지이다. 중심 자원과 가변적 기능을 연결하는 제 3의 연결부가 별도로 존재하는 것이다. 이것이 애스펙트이다. 애스펙트는 AOP의 구조물로써 클래스와 유사한 모양을 하고 있지만 그 안에는 새로운 문법 구조, 즉 교차점, 충고, 도입 등이 존재한다[8]. 이들 문법 구조로서 연결부를 구현하고 기존의 중심자원과 가변적 기능의 컴포넌트를 함께 컴파일 하면 전체 프로젝트가 완성되는 것이다.

(그림 7)의 코드는 간단한 연산을 AOP를 적용하지 않고 기존의 방법으로 중심 자원인 microwave의 인자와 가변적 기능인 utitlities의 인자와의 결합을 구현한 코드 사례다. operate() 연산은 microwave 컴포넌트 내의 연산으로 makeWarm()을 수행할 시간을 설정하고 makeWarm()을 수행하는 중심 자원의 구현이다. 여기에 microwave가 운전하는 동안 utilities의 조명 기능을 작동시키기 위한 코드를 삽입하였다. 프로그래머는 조명 기능을 추가하기 위해 코드를 이해하고 makeWarm() 연산의 위치를 알아야만 한다. 이후 기능을 다시 제거하는 경우 같은 노력을 반복해야 한다.

```
void operate(){
        Timer.setTime(10);
        utilities.Light.turnLightOn();
        makeWarm(timer);
        utilities.Light.turnLightOff();
}
```

(그림 7) AOP 미적용 조합 구현 코드



(그림 6) 조합 프로세스의 비교

```
public aspect Weaver
{
        //optional
        pointcut connectComponents():call( void microwave.kernel.Operator.makeWarm());
        before():connectComponents(){
                microwave.utilities.Light.turnLightOn()
        }
        after():connectComponents(){
                microwave.utilities.Light.turnLightOn()
        }
        //alternaive
        pointcut setLanguage():call(void microwave.kernel.Attribute.setLanguage(*));
        void around() :setLanguage(){          microwave.kernel.Attribute.setLanguage(KOREAN)
        }
}
```

(그림 8) AOP 적용 조합 구현 코드

(그림 8)은 연결부를 애스펙트로 구현한 모습이다. 그림에서 보이는 것처럼 직조 규칙을 가진 애스펙트를 구현함으로써 기존의 operate() 연산에는 어떠한 첨가나 변경도 하지 않게 된다. 이전 충고와 이후 충고를 사용하여 선택적 변경사항인 조명 기능을 추가하였고, 대체 충고를 사용하여 대안적 기능인 언어 선택을 한국어로 변경하였다. 이러한 애스펙트를 작성하기 위해 프로그래머가 알아야 할 것은 처음 개발된 제품의 코드에 대한 명세뿐이다. 변경하기를 원하는 메소드의 위치나 코드의 분석 등에 시간을 투자하지 않을 수 있다. 또한 자원 관리 측면에서도 심리적 부담감을 줄일 수 있다. 이 후 같은 계열의 다른 제품을 생산할 때에도 같은 방식으로 원하는 기능의 추가 및 변경을 위한 애스펙트를 작성하여 주면된다.

5. 관점 지향 프로그래밍을 적용한 프로덕트 라인 개발 실험

이번 장에서는 먼저 간략화 한 전자레인지를 모델로 프로덕트 라인 개념을 적용하여 요구사항을 정의한다. 또한 AOP를 도입하여 프로덕트 라인의 요구를 분석하고, UML을 이용하여 설계를 표현한다. 다음으로 본 논문에서 제안한 방법을 적용하여 설계를 구현으로 옮김으로써 모든 개발 단계에 걸쳐 AOP를 프로덕트 라인에 적용하여 생산하고 관리하는 실험을 보인다.

5.1 프로덕트 라인의 관점 지향 요구분석

실험할 제품의 요구사항은 <표 1>과 같다. 기본적인 조리 기능 외에 조리 상태와 설정 상태를 사용자에게 보이는 디스플레이 기능, 조리 온도를 설정하기 위한 온도 조절 기능, 조리가 진행 되는 동안 조리대가 회전하는 기능, 조리의 완료를 사용자에게 알리는 알람기능이 있으며, 시간 설정 등의 기능은 공통 기능인 조리 기능에 포함되어 있다고 가정한다.

위의 기능을 관점 지향적 개념을 적용하여 프로덕트 라인의 공통 기능(commonalities)과 가변적 기능(variabilities)으로 분리한다. 결과는 <표 2>와 같으며 설계와 구현을 위해 가변적 기능을 선택적 기능과 대안적 기능으로 구분한다.

조리 기능은 전자레인지 제품의 공통적 기능이며, 조명 기능과 턴테이블 기능, 알람 기능은 제품에 추가 가능한 선택적 기능이다. 온도 조절 기능은 온도 선택 기능이 없는 one-level 시스템과 단계별로 온도 조절이 가능한 multi-level 시스템이 있다. 또한, 디스플레이 기능은 국가별로 언어가 다르고, 한 줄로 표시되는 one-line display unit과 여러 줄로 표시되는 multi-line display unit이 있다고 가정한다. 따라서 앞 선 두 가지 요소는 대안적 기능으로 포함된다.

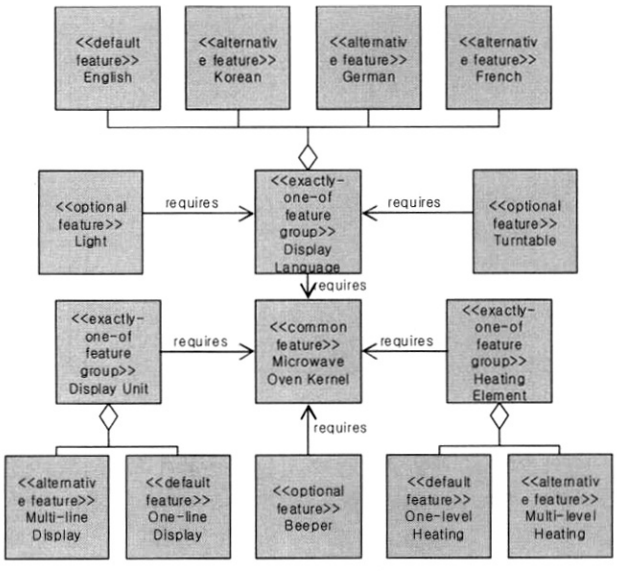
〈표 1〉 전자레인지 요구 사항

요 구 사 항	설 명
조리 기능	조리 칸 안의 음식을 조리 한다.
조명 기능	문이 열려있는 동안 조명이 켜진 상태를 유지한다.
디스플레이 기능	상태를 사용자에게 알린다.
온도조절 기능	조리 온도를 설정한다.
턴테이블 기능	조리되는 동안 조리대가 회전한다.
알람 기능	조리 완료 시 사용자에게 소리로써 알린다.

〈표 2〉 요구 사항 분리 결과

요소	기능 구분	비고
조리	공통적 기능	
조명	선택적 기능	
디스플레이	대안적 기능	디스플레이 언어 선택, 디스플레이 패널 선택
온도	대안적 기능	One-level 가열 시스템/multi-level 시스템
턴테이블	선택적 기능	
알람	선택적 기능	

5.2 UML을 이용한 프로덕트 라인의 설계



(그림 9) UML을 이용한 전자레인지 기능 설계

이전 단계에서 작성된 요구 사항 분석 결과를 UML로 설계한다. 본 논문에서 사용하는 프로덕트 라인의 설계는 Gomma가 제시한 방법을 따른다. (그림 9)에 보이는 바와 같이 전자레인지 프로덕트 라인의 공통 기능인 Microwave Oven Kernel이 <<common feature>>로 표현되었고, Light와 Turntable, Beeper가 선택적 기능인 <<optional feature>>로 표현되었다. Display Language와 Display Unit, Heating Element는 대안적 기능으로 <<default feature>>와 <<alternative feature>>로 표현된 요소들 중 한 가지를 선택하게 설계 하였다.

다음 장에서는 위에서 묘사된<<common feature>>와 <<optional feature>>, <<exactly-one-of feature group>>이 구현 단계에서 AOP 문법 요소로써 어떻게 구현되는지 보인다.

5.3 AOP를 적용한 프로덕트 라인의 구현

5.3.1 클래스 구조

구현 단계 에서는 JAVA 언어를 확장한 AspectJ를 사용하였으며, AspectJ는 자바 개발 툴, 이클립스(www.eclipse.org)에서 플러그인(plug-in)으로써 제공하고 있으므로 이클립스를 구현 도구로 이용하였다.

(그림 10) 클래스 구조

(그림 10)은 클래스의 구조를 보이고 있다. 공통적인 기능의 구현은 StandardMicrowaveOven 클래스에 모두 구현되어 있고, 이는 중심 자원을 묶는 commonalities 패키지에 포함된다. 가변 기능은 variabilities 패키지에 포함된다. 이 중 선택적인 기능들은 optional 패키지에, 그리고 대안적 기능들은 alternative 패키지에 분리되어 포함된다.

optional 패키지와 alternative 패키지에 구현된 가변 기능들과 commonalities에 구현된 공통 기능을 조합하기 위해서는 직조 규칙의 구현이 필요하다. AspectJ 파일인 weaver 패키지의 Weaver 애스펙트가 이들 간의 직조 규칙을 구현하고 조합을 돕는다.

5.3.2 테스트 메소드 구현 및 실행

(그림 11)은 테스트 메소드의 모습이다. 프로그램의 초기화를 담당하는 init()에서 대안적(alternaive) 기능인 Language와 DisplayUnit, HeatingElement가 초기화된다. 선택적(optional) 기능은 현재 추가되지 않은 상태이다. 다음으로 문을 열고 닫는 사용자의 이벤트와 함께 음식을 넣고 조리하고, 조리 완료 후 음식을 꺼내는 시나리오를 구성하였다. openTheDoor()와 closeTheDoor()는 문을 열고 닫는 이벤트 발생 시 호출되는 메소드이다. operate() 에는 운전 시간을 설정하고, 음식을 데우고, 종료를 알리는 메소드 등이 포함되어 있다.

```
StandardMicrowaveOven smo = new StandardMicrowaveOven();
        smo.init();//initialization
        smo.openTheDoor();
        System.out.println("just put the food in.");
        smo.closeTheDoor()
        smo.operate();
        smo.openTheDoor();
        System.out.println("just take the food out.");
        smo.closeTheDoor();
```

(그림 11) 테스트 메소드

```
set ENGLISH
set the variabilities.alternative.OneLineDisplayUnit@1a46e30
set the variabilities.alternative.OneLevelHeatingElement@3e25a5
just opened the door.
just put the food in.
just closed the door.
just set the operating time.
is warming for 10 in 120 degrees.
have just made warm.
just opened the door.
just take the food out.
just closed the door.
```

(그림 12) 전자레인지 기본 모델의 실행 결과

테스트 메소드의 실행 결과는 (그림 12)와 같다. 언어는 기본 언어인 "ENGLISH"로 설정 되었고, DisplayUnit과 HeatingElement는 "OnlineDisplayUnit"과 "OnLevelHeating Element"로 설정되었다. 초기화 이후 문을 열고, 음식을 넣고, 문을 닫고, 음식을 데우는 등의 시나리오가 테스트 되었다. 조리가 시작되기 전에 조리 시간과 온도가 설정된다.

5.3.3 직조의 구현 및 실행

가) 선택적 가변 기능의 구현

(그림 13)은 공통 기능과 가변 기능의 조합을 위한 직조 규칙이 구현되어 있는 Weaver 애스펙트 코드 중 선택적 (optional) 가변 기능의 구현 부분이다. 전자레인지의 문을 열고 닫을 때, 그리고 음식의 조리를 교차점으로 선언하고, 교차점 전 후에 조명이 켜지고 꺼지는 기능을 구현하였다. 다음으로, 조리의 완료시점을 교차점으로 선언, 이를 알리는 경고음 발생을 구현하였으며, 음식의 조리 중에 테이블이 회전하는 기능을 추가하였다.

나) 대안적 가변기능의 구현

(그림 14)는 공통 기능과 가변 기능의 조합을 위한 직조 규칙이 구현되어 있는 Weaver 애스펙트 코드 중 대안적 (alternative) 가변 기능의 구현 부분이다. 디스플레이 유닛을 설정하는 부분을 교차점으로 선언하여, 기존의 One-Line 디스플레이 유닛을 Multi-Line 디스플레이 유닛으로 교체하였다. 또한 언어를 설정하는 부분을 교차점으로 선언하여, 디스플레이 언어를 영어에서 한국어로 변경하였다. 조리의 온도를 담당 Heating Element를 온도 조절이 안 되는 요소에서 온도 조절을 할 수 있는 MultiLevelHeatingElement로 교체하였다.

```
public aspect Weaver{
        //********optional variants********//
        //adding the Light
        Light l = new Light();
        pointcut openingDoor():call(public void
        StandardMicrowaveOven.openTheDoor(..));
        after(): openingDoor(){l.turnOn();}
        pointcut closingDoor():call(public void
        StandardMicrowaveOven.closeTheDoor(..));
        after(): closingDoor(){l.turnOff();}
        pointcut makingWarm():call(public void
        StandardMicrowaveOven.makeWarm(..));
        before(): makingWarm(){l.turnOn();}
        after(): makingWarm(){l.turnOff();}
        //adding the Beeper
        Beeper b = new Beeper();
        pointcut endOfOperation():call(void
        StandardMicrowaveOven.exit());
        after(): endOfOperation(){b.beep(3);}//beep for 3 seconds
        //adding the Turnable function
        Turntable t = new Turntable();
        pointcut turningTheTable():call(public void
StandardMicrowaveOven.makeWarm(..));
        before(): turningTheTable(){t.turnAround();}
        after(): turningTheTable(){t.stopOff();}
    ......
}
```

(그림 13) 선택적 가변 기능의 구현

```
public aspect Weaver{
        . . .
        //*******alternative variants**********/
        //resetting DisplayUnit
        pointcut settingDisplayUnit(DisplayUnit displayUnit)
        :call(public void
                StandardMicrowaveOven.setDisplayUnit(DisplayUnit))
                        &&args(displayUnit);
        void around(DisplayUnit displayUnit):
        settingDisplayUnit(displayUnit){
                        proceed(new MultiLineDisplayUnit());}
        //resetting Language
        pointcut settingLanguage(String language)
        :call(public void
                StandardMicrowaveOven.setLanguage(String))
                        &&args(language);
        void around(String language):
        settingLanguage(language){proceed("KOREAN");}
        //resetting HeatingElement
        pointcut settingHeatingElement(HeatingElement
        heatingElement):call(public void
        StandardMicrowaveOven.setHeatingElement(HeatingElement))
                &&args(heatingElement);
        void around(HeatingElement heatingElement)
                :settingHeatingElement(heatingElement){
                        proceed(new MultiLevelHeatingElement());}
}
```

(그림 14) 대안적 가변 기능의 구현

```
set KOREAN
set the variabilities.alternative.MultiLineDisplayUnit@1fb8ee3
set the variabilities.alternative.MultiLevelHeatingElement@61de33
just opened the door.
just turn the light on
just put the food in.
just closed the door.
just turn the light off
just set the operating time.
just turn the table Around
is warming for 10 in 120 degrees.
just stop the table off
have just made warm.
Beep!!Beep!!Beep!!
just opened the door.
just turn the light on
just take the food out.
just closed the door.
just turn the light off
```

(그림 15) 개선된 전자레인지의 실행 결과

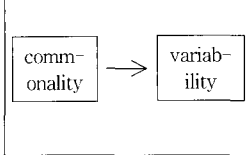
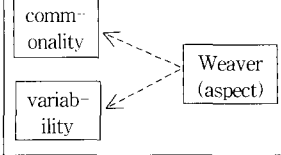
선택적 기능과 대안적 기능 모두를 적용하여 개선된 전자레인지의 실행 모습이 (그림 15)이다. 문을 열고 닫을 때 램프가 켜지고 꺼진다. 운전 중에는 선반이 회전하며, 조리가 완료되었을 때는 경보음이 발생한다. 또한 기존의 선택적 기능의 객체들이 상이한 객체로 대체되어 초기화 된 것을 확인할 수 있다.

5.4 실험 평가

본 실험에서는 AOP를 적용하여 전자레인지의 프로덕트 라인을 구현하였다. 공통 기능과 가변 기능을 조합하는 과정에 있어 AOP를 적용하지 않은 경우와 AOP를 적용한 경우는 <표 3>과 같은 차이를 보인다.

소프트웨어 프로덕트 라인을 구현함에 있어 AOP를 적용할 경우 가변 기능이 삽입될 위치의 파악이 불필요할뿐더러 자원에 대한 코드 간섭이 전혀 없고, 모듈간의 의존 정도 역시 <표 3>에서처럼 약해진다. 또한, 각 기능을 구현한 모듈간의 결합이 제어 결합에서 자료 결합으로 느슨해짐으로써 더 나은 모듈화를 이룰 수 있다.

〈표 3〉 공통 기능과 가변 기능의 조합을 위한 AOP 적용과 미적용 효과 비교

	AOP 미적용	AOP 적용
코드이해	가변기능이 삽입될 위치를 파악하기 위한 코드의 이해가 필요함.	가변 기능이 삽입될 위치의 파악이 불필요함.
코드간섭	공통 기능에 가변 기능의 생성 및 호출을 위한 코드가 삽입됨.	공통기능과 가변 기능을 조합하기 위한 코드가 별개의 애스펙트에 구현 되므로 각 자원들에 대한 코드 변경이 없음.
모듈간의 의존 정도		
모듈간의 결합도	제어결합(control coupling)	자료결합(data coupling)

6. 결 론

소프트웨어 개발 방법론은 자원의 재사용을 통한 생산성 향상과 time-to-market의 감소를 목적으로 발전되어 왔다. 소프트웨어 프로덕트 라인은 같은 제품 군에 대해 공통적으로 사용되는 기능과 부가적이고 변경이 예상되는 기능을 분리하여 개발하고, 제품 계열의 공통 기능은 범용 컴포넌트로 개발하여 다음 버전의 제품이나 비슷한 제품을 개발하는 경우 공통 기능인 범용 컴포넌트를 재사용함으로써 소프트웨어의 생산성을 극대화 하는 개발 방법론이다. 이러한 개발 방법론은 소프트웨어 개발 주기의 어느 한 단계에만 적용함으로써 효과를 기대할 수 없다. 요구 분석 단계와 설계 단계, 구현 단계에 이르기까지 일괄적이고 총체적인 관리와 적용이 요구된다.

프로덕트 라인의 일괄적이고 총체적인 생산 및 관리를 위해 관점 지향 프로그래밍의 개념을 요구 분석에 적용하고, 요구 분석 결과로 분류된 공통 기능과 가변 기능을 설계에 적용하려는 연구가 선행되었다[3, 9, 11]. 또한 구현 단계에 분리된 요구를 구현하여 자원을 관리하고 다음 제품의 개발에 이용하는 방법에 대한 연구가 진행되어 왔다[1, 4]. 이들 연구에서는 자원의 관리를 위해 XML 등 자원 코드와 다른 성격의 마크업 언어를 동원하거나 자원을 이용한 제품의 생성에 제 2의 언어를 도입하였다. 도입한 이유는 소프트웨어 프로덕트 라인의 조립 공정에 도움이 되기 위해서였지만 이러한 노력은 자원의 재사용으로 인해 얻어지는 생산성의 이익에 비해 큰 노력이 요구된다.

본 논문의 연구는 위에서 설명한 노력을 감소시킴으로써 재사용으로 얻어지는 이익을 높이고자 하였으며, 소프트웨어 프로덕트 라인의 개발 방법론에 따라 공통 기능과 가변 기능 각각의 구현을 조립하는 조합 프로세스의 개선에 초점을 맞추었다. 공통 기능을 구현한 범용 컴포넌트와 가변 기능을 구현한 컴포넌트의 아픈 역할을 애스펙트라는 AOP의 구조물에게 위임함으로써 프로그래머는 중심 자원과 가변 기능의 코드 변경 없이 새로운 구조물을 작성한다. 이 구조물은 중심 자원과 가변 기능의 직조 규칙을 가지고 있으며, 각 기능간의 상호작용을 담당한다.

본 논문에서는 모듈화를 도와주는 AOP 패러다임을 소프트웨어 프로덕트 라인의 구현에 적용하였으며, 작은 제품에 대해 요구 분석, 설계, 구현 단계를 실험하였다. 소프트웨어 개발 방법론으로써의 장점은 인정되지만 아직 실무에서의 적용이 활발하지 못한 소프트웨어 프로덕트 라인 방법론에 구체적인 활용 방안을 제시하였다는데 본 논문의 의의가 있다.

소프트웨어 프로덕트 라인은 하드웨어의 생산 패턴에서 유래한 만큼 임베디드 소프트웨어에 적용하는 경우 더욱 유리한 방법론이다. 임베디드 제품은 짧은 시간에 유사한 버전의 많은 제품이 출시되므로 time to market에 매우 민감하며, 이를 충족시키기 위해서는 제품에 대한 자원 활용이 필수적이다. 차후 연구로써 임베디드 소프트웨어의 개발 패턴을 분석하고 소프트웨어 프로덕트 라인 개발 방법론을 적용하는 실험적 연구가 필요하겠다.

참 고 문 헌

[1] Walker, R.J., Baniassad, E.L.A., Murphy, G.C., "An initial assessment of aspect-oriented programming", Proceedings of the 1999 International Conference on Software Engineering, pp.120-130, 1999.

[2] Michalis A., Cristina G., "Implementing Product Line Variabilities," ACM SIGSOFT Software Engineering Notes, Proceedings of the 2001 Symposium on Software Reusability, 2000.

[3] Kuloor, C., Eberlein, A., "Aspect-Oriented Requirements Engineering for Software Product Line," Proceedings of 10th IEEE International Conference and Workshop Engineering of Computer-Based Systems, pp.98-107, 2003.

[4] Zhang, H., Jarzabek, S., "An XVCL approach to handling variants: a KWIC product line example," IEEE Tenth Asia-Pacific Software Engineering Conference, pp.116-125, 2003.

[5] Ramnivas Laddad, AspectJ in Action: Practical Aspect-Oriented Programming, Manning Publications, 2003.

[6] Len Bass, Paul Clements, Rick Kazman, Software Architecture in Practice(second edition), Addison-Wesley, 2003.

[7] 민진우, 이인선, 이클립스 기반 프로젝트 필수 유틸리티: CVS, Ant, JUnit, 한빛 미디어, 2004.[11] Youhee Choi, Gyusang Shin, Youngjong Yang, Changsoon Park, "An Approach to Extension of UML 2.0 for Representing Variabilities", Fourth Annual ACIS International Conference, pp.258-261, 2005.

[8] Yoshida, M., Iwane, N., Matsubara, Y., "The integrated software product line model," IEEE Conference Cybernetics and Intelligent Systems, pp.538-543, 2004.

[9] Hassan Gomaa, Designing Software Product Line with UML, Addison-Wesley, 2004.[12] Hideaki S., Tetsuo T., "Impact Analysis of Weaving in Aspect-Oriented Programming," Software Maintenance ICSM'05. Proceedings of the 21st IEEE International Conference, pp.657-660, 2005.

[10] 정주미, 최승훈, "ACAB:소프트웨어 프로덕트 라인 지원을 위한 컴포넌트 개발 도구," 제32회 한국정보과학회 추계학술발표회 논문집 Vol.32, No.2(II), pp.427-429, 2005.

[11] Youhee Choi, Gyusang Shin, Youngjong Yang, Changsoon Park, "An Approach to Extension of UML 2.0 for Representing Variabilities", Fourth Annual ACIS International Conference, pp.258-261, 2005.

[12] Hideaki S., Tetsuo T., "Impact Analysis of Weaving in Aspect-Oriented Programming," Software Maintenance ICSM'05. Proceedings of the 21st IEEE International Conference, pp.657-660, 2005.

[13] Yokoyama, T., "An aspect-oriented development method for embedded control systems with time-triggered and event-triggered processing," Real Time and Embedded

Technology and Applications Symposium RTAS 2005. 11th IEEE, pp.302-311, 2005.

[14] Schmidt, P., Milstein, J., Alvarado, S., "Architectural assessment of embedded systems using aspect-oriented programming principles," Object-Oriented Real-Time Distributed Computing ISORC 2005. Eighth IEEE International Symposium, pp.90-93, 2005.

허 승 현

e-mail : seknj@hanmail.net

2004년 동국대학교 컴퓨터공학과(공학사)

2006년 동국대학교 컴퓨터공학과 (공학석사)

관심분야 : 소프트웨어 프로덕트 라인, 관점 지향 프로그래밍, 기능점수

최 은 만

e-mail : emchoi@dgu.ac.kr

1982년 동국대학교 전산학과(학사)

1985년 한국과학기술원 전산학과 (공학석사)

1993년 일리노이 공대 전산학과 (공학박사)

1985년~1988년 한국표준연구소 연구원

1988년~1989년 데이콤 주임연구원

1998년~2004년 한국정보과학회 소프트웨어공학회 운영위원

2000년~2001년 콜로라도 주립대 전산학과 방문교수

2001년~2003년 한국정보처리학회 학회지 편집위원

2002년 카네기멜론대 소프트웨어공학 단기과정 연수

1993년~현재 동국대학교 컴퓨터공학과 교수

관심분야 : 소프트웨어 설계, 소프트웨어 테스트, 품질측정, Program Comprehension