관점지향프로그래밍(AOP)의 소개와 응용

숭실대학교 최재영* · 곽동규**
한국IBM 이명철 · 장현기 · 백영상

1. 관점지향프로그래밍(Aspect Oriented Programming)에 대하여

프로그래밍개발 방법론은 일괄처리프로그래밍에서 구조적프로그래밍, 객체지향프로그래밍으로 발전해왔다. 프로그래밍개발 방법론의 발전은 프로그램의 높은 모듈화를 이루기 위한 연구의 성과이다. 구조적프로그래밍과 객체지향프로그래밍은 함수를 이용하여 요구사항을 모듈화 할 수 있는 방법을 제공한다. 관점지향프로그래밍은 구조적프로그래밍과 객체지향프로그래밍에서 모듈화하지 못한 여러 요구사항에 걸쳐져있는 부가적인 요구사항을 모듈화하기 위한 방법으로 연구되었다. 본 장에서는 관점지향프로그래밍의 기본 개념 및 주 용어, 그리고 장점에 대하여 알아본다.

1.1 관점지향프로그래밍의 기본 개념

관점지향프로그래밍(AOP) 기법은 1997년 Gregor Kiczales가 "Aspect-Oriented Programming"[1]에서 제안한 프로그램 개발 방법론이다. 관점지향프로그래밍에서 프로그램의 요구사항이나 처리사항을 관심사(concern)라고 한다. 관심사는 핵심관심사(core concern)와 횡단관심사(cross-cutting concern)로 분류할 수 있다. 핵심관심사는 단일 모듈이 가지는 주된 요구사항이고, 횡단관심사는 여러 개의 모듈에 공통적으로 적용되는 부가적인 요구사항이다. 관점지향프로그래밍은 객체지향프로그래밍(Object Oriented Programming)에서 원래 모듈화 되지 못한 횡단관심사를 해결하기 위한 방법으로 연구되었다. 그림 1은 은행 시스템을 예로 하는 관심사의 분석 방법이다.

그림 1과 같이 은행 시스템은 계좌이체와 입출금, 이자계산과 같은 여러 관심사가 존재한다. 각 관심사는 객체(Object)의 메소드로 모듈화하여 해결한다. 각 모듈은 로깅과 보안, 트랜잭션과 같이 공통적인 처리사항을 가진다. 그림 1과 같은 프로그램을 설계하는 방법으로 각 모듈의 공통적인 처리사항을 횡단관심사로 처리하여, 핵심관심사와 다른 방법으로 재사용성을 높이는 방법을 생각할 수 있으나, 객체지향프로그래밍의 상속만으로는 그림 1과 같은 예에서의 횡단관심사를 구조적으로 구현하기 어렵다. 하지만 관점지향프로그래밍에서는 횡단관심사를 표현할 수 있는 구조적인 방법을 제공하고 있어 공통적인 모듈의 변경이나 재사용성을 높일 수 있는 장점을 가진다.

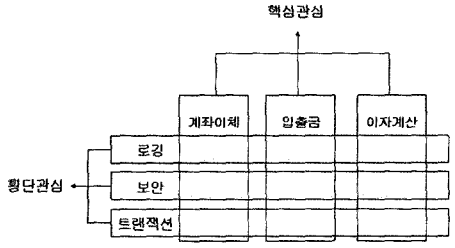


그림 1 관점지향프로그래밍의 관점 분석 방법의 예

효과적으로 관점지향프로그래밍을 이용한 프로그램을 작성하기 위해서는 프로그램의 관심사를 분석하여 공통적인 요구사항, 즉 횡단관심사를 찾아 모듈화한다. 그림 2는 관심사 분석(concern decomposition)을 프리즘에 예를 들어 설명한 그림이다[2].

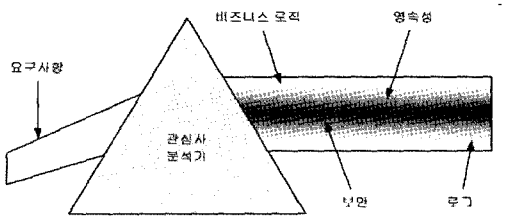


그림 2 관심사 분석

프로그램은 요구사항을 달성하기 위해 다수의 관심사를 갖는다. 프리즘이 빛을 분해하여 서로 다른 색으로 분해하듯이, 프로그램의 요구사항을 분석하면 다수의 다른 관심사로 분해할 수 있다. 그림 2와 같이 관

* 종신회원
** 학생회원

심사 분해기를 통해 분석된 관심사는 다시 직조기(weaver)를 통해 합성되어 횡단관심사와 핵심관심사가 모두 적용된 프로그램 결과를 얻는다. 그림 3은 관심사 분석과 직조의 과정을 직관적으로 보인다.

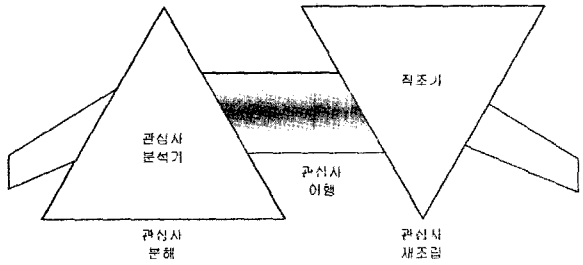


그림 3 관심사 분석과 직조

관점지향프로그래밍에서의 직조는 횡단관심사와 핵심관심사를 옷감을 짜듯이 교차시켜 두 관심사를 충족시키는 프로그램을 생성하는 것을 의미한다. 관점지향프로그래밍으로 작성된 프로그램은 핵심관심사로 작성된 모듈에서 교차지점을 통해 횡단관심사로 분기한다.

1.2 관점지향프로그래밍의 용어

관점지향프로그램은 기존의 개발방법론과 다른 개념을 포함하고 있어 새로운 용어가 등장한다. 관점지향프로그래밍의 개념을 구체화하기 위해 아래와 같은 용어를 사용한다.

1.2.1 결합점(join point)

프로그램 실행에서 구별 가능한 프로그램의 지점이다. 즉 모듈의 호출이나 멤버 변수에 대한 값 할당 등이 결합점이 될 수 있다.

1.2.2 교차점(pointcut)

결합점들 중에서 선택된 핵심관심사와 횡단관심사의 직조 지점이다. 관점지향 프로그래밍으로 작성된 프로그램은 핵심관심사로 작성된 모듈에서 교차점을 통해 횡단관심사로 분기한다.

1.2.3 어드바이스(advice)

각 결합점에 삽입되어 실행될 수 있는 프로그램 코드를 의미한다. 주로 메소드 단위로 작성되는 어드바이스는 교차점에 의해 결정된 결합점에서 실행된다.

1.2.4 직조(weaving)

교차점을 통해 결합점에 어드바이스가 삽입되는 과정을 의미한다. 직조의 방법은 관점지향프로그래밍 기법을 지원하는 컴파일러를 사용하거나 실시간으로 목적코드(object code)를 변환하는 방법 등 여러 가지가 있다.

1.2.5 애스팩트(aspect)

교차점과 어드바이스를 작성하는 단위이다. 많은 관점지향프로그래밍 언어에서 애스팩트를 클래스와 같은 프로그램의 단위로 사용한다.

1.3 관점지향프로그래밍의 장점

관점지향프로그래밍은 높은 모듈화를 제공하여 모듈간의 책임소재, 모듈간의 결합도, 코드 재사용성 등에 대해 다음과 같은 장점을 가진다. 첫째로 관점지향프로그래밍은 모듈에 대한 책임 소재가 명확하다. 개발자는 핵심관심사를 작성할 때에는 핵심관심사에만 집중하고 횡단관심사에 대해서는 관여할 필요가 없다. 두 번째 장점은 느슨한 결합도로 모듈화를 보다 높일 수 있는 방법을 제공한다. 관점지향프로그래밍은 횡단관심사를 통해 여러 모듈에 걸쳐있는 횡단관심사를 모듈화 하여 코드의 중복성을 제거할 수 있다. 또한 느슨한 결합도는 코드의 재사용성을 증가시킬 수 있다. 세 번째 장점은 시스템 수정의 용이함이다. 개발된 프로그램에 새로운 횡단관심사의 추가/직조를 통해 새로운 기능을 추가할 수 있는 장점이 있다.

2. 관점지향프로그래밍의 특징 및 작동방식

앞서본 관점지향프로그래밍의 개념을 실제 프로그래밍언어에 적용한 언어는 직조가 일어나는 시점에 따라 정적 직조(static weaving)와 동적 직조(dynamic weaving)로 나누어진다. 정적 직조는 소스를 컴파일할 때 직조가 일어나는 방식이고 동적 직조는 프로그램 로딩이나 실행 시 직조가 일어나는 방식이다. 본 장에서는 정적 직조와 동적 직조를 모두 제공하는 AspectJ[3]를 소개한다.

2.1 AspectJ

AspectJ는 대표적 객체지향언어인 Java에 관점지향프로그래밍을 도입한 최초의 관점지향프로그래밍 적용 언어이다. 제록스 연구원들에 의해 구현된 관점지향언어인 AspectJ는 현재 이클립스의 서브프로젝트로 발전하고 있다. AsepctJ는 Java 언어에 관점지향 개념을 추가하여 확장한 범용의 언어이다. 그러므로 모든 유효한 Java 프로그램은 유효한 AspectJ 프로그램이 된다. AsepctJ 컴파일러는 Java 바이트 코드 명세 규칙을 따르기 때문에 Java 가상기계(virtual machine)에서 실행된다.

AspectJ를 이용한 프로그램에서 핵심관심사는 Java 언어로 작성되고 횡단 관심사와 직조는 AspectJ만의 문법인 애스팩트 단위로 작성한다. AspectJ는 컴파일과 디버깅을 위한 도구인 통합개발환경(integrated development environment)을 제공한다.

2.2 예제를 통한 관점지향프로그래밍 동작방식

AspectJ는 관점지향프로그래밍을 Java에 적용할 수 있도록 한다. 그림 4는 간단한 AspectJ 프로그램과 그 실행 결과이다.

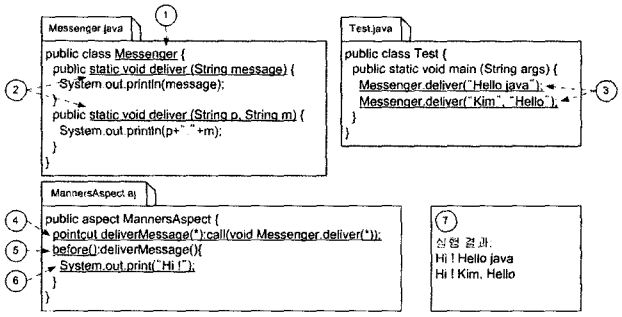


그림 4 간단한 AspectJ 예제와 실행 결과

그림 4의 예제는 간단한 문자열을 출력하는 프로그램이다. ①은 문자열을 하나 또는 둘을 인자로 받아 다른 형식으로 출력하는 클래스이다. Messenger 클래스는 ②와 같이 두 개의 함수를 중첩하였다. 이 프로그램의 main 함수는 ③과 같이 Messenger 클래스의 두 함수를 호출한다. MannersAspect.aj의 ④는 횡단관심사가 삽입될 함수와 횡단관심사를 구현한 함수를 기술한다. ⑤는 "Advice"가 적용되는 지점이다. 이 예제에서는 함수가 시작될 때 "Advice"가 실행된다. ⑥은 횡단관심사를 실제로 구현한 "Advice"이다. ⑦의 실행결과와 같이 각 문자열을 출력하기 전에 "Hi!"를 출력한다.

표 1은 기존의 프로그램에 로그를 발생시키는 요구사항을 추가하기 위해 수작업으로 추가한 예와 AspectJ를 이용한 예이다.

표 1은 수작업으로 작성해야 하는 횡단관심사를 AspectJ를 이용하여 모듈화 시킨 예이다. 표 1에서 ①은 관점지향프로그램을 사용하지 않고 모든 메서드에 로그를 삽입하는 예제이다. 일반적으로 로그를 삽입하기 위해서는 ①과 같은 처리루틴을 모든 함수에 작성하여야 한다. 하지만 ②와 같은 관점지향프로그램을 사용하면 핵심관심사 메서드를 전혀 수정하지 않고 메소드에 로그 루틴을 적용할 수 있다. AspectJ는 객체지향프로그래밍에서 class와 유사하게 aspect를 단위로 프로그램을 작성한다. 그리고 join point들은 다른 join point들을 가지고 있을 수 있다. 예를 들어 하나의 메소드 호출은 결과 값을 반환하기 전에 몇 개의 다른 메소드를 호출할 수 있다. pointcut은 정의된 기준에 따라 일련의 joinpoint들을 가져오는 언어 구조를 가리킨다. 예제에서 "publicMethods"라는 첫 번째 pointcut은 rg.apache.cactus 패키지에서 모든 public 메소드 실행을 선택한다.

3. 관점지향프로그래밍의 응용사례

관점지향프로그래밍은 기존에 모듈화하기 어려운 횡단관심사를 모듈화할 수 있는 방법을 제공한다. 횡단관심사를 바라보는 관점에 따라 새로운 시스템을 제안할 수 있다. 본 장에서는 지금까지 연구된 응용사례를 3가지로 - 테스팅, 기능추가, 코드생성 시 응용 - 나누어 살펴본다.

3.1 프로그램 테스팅을 위한 관점지향프로그래밍

프로그램에 로그나 디버깅 정보를 추가하는 등의 프로그램 테스팅 분야는 Aspect가 가장 많이 응용되는 분야중 하나다. 일례로, 앞서 소개한 Eclipse의 서브 프로젝트인 AspectJ는 예제 프로그램의 하나로 함수 트레이스(Function Trace) 프로그램을 소개하고 있다. 함수 트레이스 프로그램은 함수가 호출 경로를 분

표 1 AspectJ 코드의 예

구분	코드

① 모든 모듈에 수작업으로 삽입된 로그	public void doGet(JspImplicitObjects theObjects)
	throws ServletException{
	logger.entry("doGet(...)");
	JspTestController controller = new JspTestController();
	controller.handleRequest(theObjects);
	logger.exit("doGet");}

② 모든 메소드에 자동으로 적용되는 로그	public aspect AutoLog{
	pointcut publicMethods():execution(public * org.apache.cactus..*(..));
	pointcut logObjectCalls():execution(* Logger.*(..));
	pointcut loggableCalls():publicMethods() && ! logObjectCalls();
	before() : loggableCalls(){
	Logger.entry(thisJoinPoint.getSignature().toString()); }
	after() : loggableCalls(){
	Logger.exit(thisJoinPoint.getSignature().toString()); }}

석하는 테스트 프로그램이다. 예제로 보인 테스트 프로그램은 테스트 대상이 되는 프로그램을 핵심관심사로 보고 테스팅을 위한 코드를 횡단관심사 작성하여 프로그램을 테스트하는 방법을 사용하였다.

이에 더 나아가 Rajan[4]는 AOP를 이용한 테스팅 코드 생성방법에 대해 연구하였으며, 한 예로 코드 커버리지 분석 도구(Code Coverage Analysis)인 AspectCov 테스팅 도구를 소개하였다. 커버리지 분석 도구는 대상 코드의 실행 영역과 실행되지 않는 영역을 분석하는 도구이다. 그림 5는 테스팅 시스템의 프레임워크를 보인다.

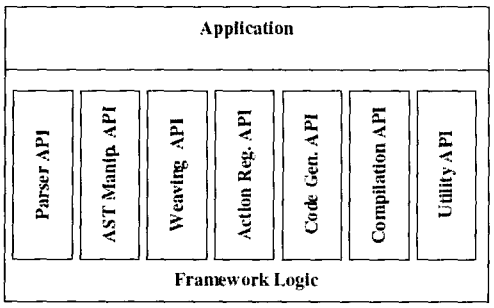


그림 5 Rajan가 제안한 테스팅 프로그램의 프레임워크

AspectCov는 테스트 대상 프로그램을 파싱 및 분석하여 테스트 대상 프로그램의 커버리지를 확인할 수 있는 로그를 발생하는 AspectJ 코드(concern coverage)를 생성한다. 프로그램의 커버리지는 생성된 AspectJ 코드와 대상 프로그램을 컴파일 및 실행한 결과인 로그를 분석함으로써 확인할 수 있다. 기존의 커버리지를 확인하는 시스템은 기존 코드에 로그를 발생시키는 코드를 자동으로 삽입하여 커버리지를 확인하였다. 하지만 코드를 삽입하는 방법은 테스트 대상 코드와 로그 발생 코드를 구분하기 어렵다. Rajan는 AspectCov를 통해 명료하고 가독성 높은 테스트 코드를 생성할 수 있음을 보였다.

3.2 기존 프로그램의 기능추가를 위한 관점지향 프로그래밍

기존에 작성된 프로그램의 새로운 기능을 추가하기 위한 방법으로 Lopes는 관점지향프로그래밍을 이용한 AML (Aspect Markup Language)[5]을 제안하였고, 이를 Java에 적용하여 JAML(Java Markup Language)을 만들었다. 그는 자바의 목적 코드(.class 파일이나 .jar)만 존재하는 프로그램의 기능을 추가하기 위해서 인터셉터(interceptor)를 정의한 XML 기반의 AML 문서를 제안하였다. 그림 6은 JAML의 직조 프로세스 구조를 보인다.

사용자는 필요에 따라 기존 모듈의 인터셉트할 위치를 AML 문서로 기술하고 인터셉트 시 호출할 함수를 작성하는 방법으로 기존 모듈에 새로운 기능을 추가할 수 있다. AML 문서와 호출할 함수는 AST converter를 통해 AML AST로 생성되고 AST를 분석하여 AspectJ 소스를 생성한다. 이를 기존의 모듈과 AspectJ 컴파일러로 새로운 프로그램을 생성할 수 있다. 그림 7은 제안한 시스템의 AML과 코드 생성 과정을 보인다.

그림 7은 Lopes가 제안한 시스템의 한 예제이다. 이 예제에서 JAML을 이용해서 이미 작성되어 컴파일된 응용프로그램에 트레이스 로그를 생성하는 코드를 추가할 수 있음을 보였다. 그림 6에서 ①은 AML 문서이고 ②는 추가할 실행 코드이다. AML 문서에는 대상 프로그램의 어느 위치에서 실행할 코드인지를

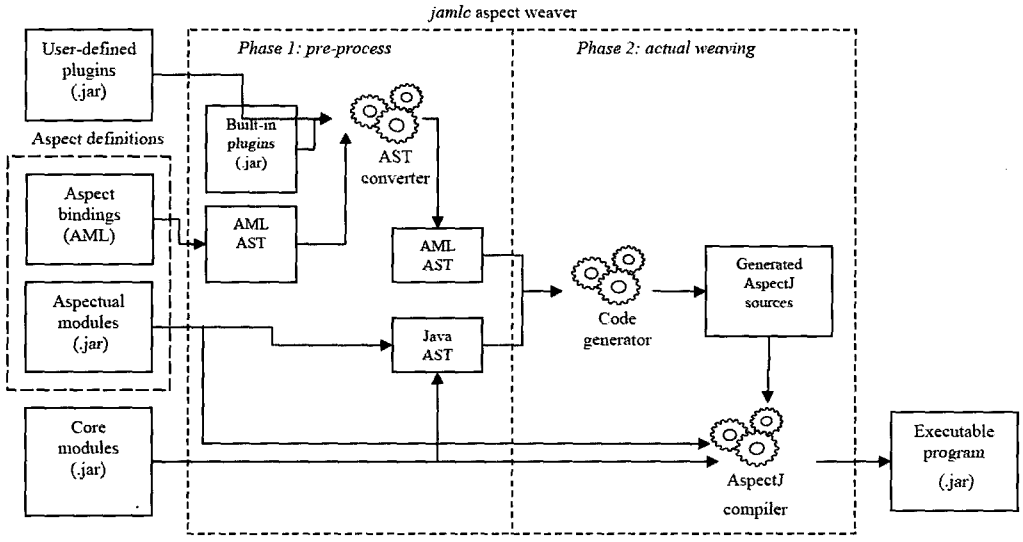


그림 6 Lopes가 제안한 JAML 시스템의 직조 프로세스

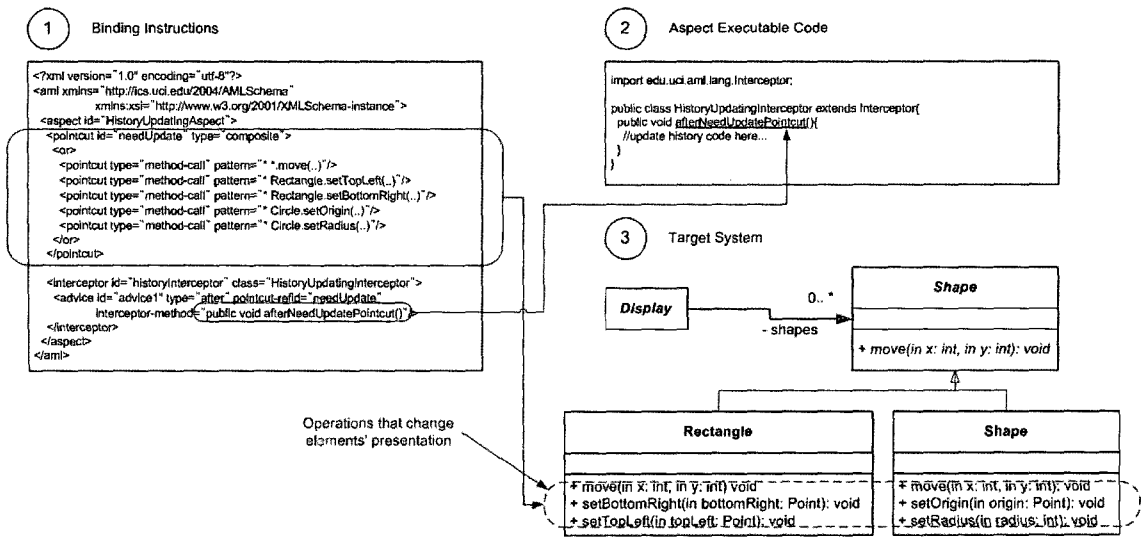


그림 7 Lopes가 제안한 JAML의 코드 생성

pointcut으로 기술한다. Lopes는 AspectJ의 목적 프로그램에 관점지향프로그램을 적용할 수 있는 특성을 이용하여 프로그램에 기능을 추가하는 방법을 소개하였다. 또한 AML이란 마크 업 언어를 사용하여 AspectJ 문법에 대한 학습 부담을 감소시켰다.

3.3 코드생성 프로그램에서의 관점지향프로그래밍

코드를 생성하여 생성된 코드를 이용하는 시스템은 성능이 뛰어나고 이식성이 높아 다양한 분야에서 사용중이다. 코드생성 프로그램의 요구사항은 코드를 생성하는 루틴에 포함되어 있다. 그러므로 코드생성 프로그램의 요구사항을 추가하기 위해서는 코드를 생성하는 루틴을 분석하여 수정해야 한다. 하지만 코드 생성 루틴은 코드를 생성하는 요구사항을 가지고 있어 생성된 프로그램의 요구사항을 분석하기 어렵다. 관점지향프로그램을 이용하여 코드를 수정할 경우 컴파일 루틴의 수정과 생성된 프로그램의 횡단관심사로 새로운 요구사항을 추가할 수 있다. 본 장에서는 (주)한국IBM 유비쿼터스컴퓨팅연구소와 숭실대학교 시스템소프트웨어 연구실과 함께 연구한 코드 생성 엔진에 AOP를 이용하여 새로운 기능을 추가한 사례를 보인다.

코드를 생성하여 생성된 프로그램을 사용하는 프로그램 중 BPEL(Business Process Execution Language)[6]를 지원하는 B2J(BPEL to Java)[7]가 있다. BPEL은 웹 서비스 환경에서 비즈니스 프로세스를 정의하고 실행하기 위한 표준 언어로써 현재 많은 응용분야를 가지고 있다. 또한, B2J는 BPEL4WS를 지원하는 서비스 엔진으로서 다른 BPEL 엔진과 다르게 BPEL 문서를 자바 소스로 변환하여 서비스를 제공한다. 하지만 BPEL은 컨텍스트 정보(Context Information)를 분석하기 위한 직관적인 조건을 제공하지 않아 편재형 컴퓨팅(Pervasive Computing) 환경에서 사용하기 어렵다. 규칙(Rule)[8]은 컨텍스트 정보를 분석하기 위한 높은 수준의 조건을 제공한다. 그러므로 BPEL에 규칙을 처리하는 언어를 추가하면 BPEL의 프로세스 기능과 웹 서비스 기능을 그대로 사용하면서 컨텍스트 정보를 처리할 수 있는 편재형 컴퓨팅 환경에 적합한 언어로 사용할 수 있다.

B2J 엔진이 생성한 자바 소스에 새로운 요구사항을 추가하기 위해 AspectJ를 이용한다. 사용자는 BPEL 문서와 인터셉터(Interceptor) 문서를 작성하고 본 시스템은 이 두 문서를 입력으로 한다. 인터셉터 문서는 BPEL 실행 루틴 중 특정 부분에 규칙을 처리하는 루틴을 삽입하기 위한 정보를 가지고 있다. BPEL 문서는 B2J Coordinator를 통해 자바 소스를 생성하고 인터셉터 문서는 Interceptor2AspectJ Engine을 통해 AspectJ 소스로 변환된다. 이렇게 생성된 두 프로그램 소스는 AspectJ 컴파일러를 통해 컴파일 되어 실행 가능한 자바 클래스 파일을 생성하고 B2J Worker Host에서 실행된다. 그림 8은 인터셉터의 예제를 보여준다.

그림 8에서 ①은 Interceptor를 적용할 BPEL 문서를 나타내고 ②의 "Interceptor" 엘리먼트는 BPEL 문서에 적용할 단위이다. "when" 속성은 XPath를 이용하여 적용할 BPEL 엘리먼트 위치를 기술한다. ③은 인터셉터 문서에서 사용할 변수를 표현하는 방법이고 ④에 "executeRule" 엘리먼트는 등록되어 있는 규칙을 실행하게 하는 표현 방법이다.

이 연구에서는 인터셉터와 BPEL 문서를 독립적으로 작성하여 BPEL 문서에 새로운 요구사항을 추가하

```
<?xml version="1.0" encoding="UTF-8"?>
<interceptors xmlns:addr="http://test.com/address" xmlns:ex="http://test.com/ex">
      <bpelDocs>                                                          1
            <bpelDoc location="simple_variable_test.bpel" />
      </bpelDocs>
      <interceptor when="/:process/:scope">                               2
            <variables>
                  <variable id="sex" type="String" />                     3
                  <variable id="age" type="int" />
                  <variable id="returnValue" type="String" />
            </variables>
            <before>
                  <assign>
                        <copy>
                              <from>20</from>
                              <to variable="age" />
                        </copy>
                  </assign>
                  <assign>
                        <copy>
                              <from>"female"</from>
                              <to variable="sex" />
                        </copy>
                  </assign>
                  <executeRule ruleId="CustomerLevel">                    4
                        <inputs>
                              <input id="sex" type="text"/>
                              <input id="age" type="int"/>
                        </inputs>
                        <return id="returnValue" />
                  </executeRule>
            </before>
            <after>
            </after>
      </interceptor>
</interceptors>
```

그림 8 인터셉터 예제

여 사용할 수 있는 장점을 가진다. 또한, B2J 엔진의 수정을 최소한으로 하여 규칙을 처리하도록 설계하였다.

4. 결 론

관점지향프로그래밍은 90년대 말에 소개되어 최근 더욱 주목받고 있고 다양한 분야에 다양한 방법으로 사용되고 있다. 관점지향프로그래밍은 프로그램을 핵심관심사 프로그램과 핵심관심사에 영향을 주지 않는 횡단관심사 프로그램으로 나눌 수 있는 방법을 제공한다. 여기서 주목할 사실은 핵심관심사에 전혀 영향을 주지 않는 프로그램을 작성한다는 것이다. 이런 특성을 이용하여 (주)한국IBM의 유비쿼터스 컴퓨팅연구소와 숭실대학교 시스템소프트웨어연구실에서는 BPEL 엔진 중 BPEL 문서를 자바 소스를 생성하여 실행하는 B2J를 분석하여 B2J가 생성하는 자바 소스에 규칙을 AspectJ를 이용하여 추가하는 연구를 진행하였다. 이 연구로 B2J 엔진의 수정을 최소로 하여 B2J 엔진이 생성한 자바 소스에 새로운 기능을 추가할 수 있음을 보였다.

관점지향프로그래밍은 개발자에게 더욱 강력한 모듈화 방법을 제공한다. 또한 우리에게 새로운 프로그램 기법에 대한 제시한다. 앞에서 소개한 간단한 관점지향 프로그래밍 사용 예제 외에도 다른 많은 곳에서도 다양한 방법으로 연구되고 있으리라고 생각되며, 관심사를 바라보는 새로운 시각이 또 다른 이슈를 만들 것이라고 기대한다.

참고문헌

[1] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin, "Aspect-Oriented Programming," ECOOP, pp220-242, 1997.

[2] R. Laddad, *AspectJ in Action: Practical Aspect-Oriented Programming*, Manning, 2003.

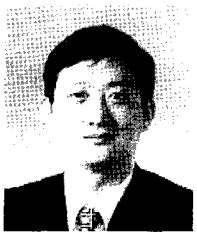
[3] eclipse AspectJ, "http://www.eclipse.org/aspectj"

[4] Hridesh Rajan, Kevin Sullivan, "Generalizing AOP for Aspect-Oriented Testing," In the proceedings of the Fourth International Conference on Aspect-Oriented Software Development (AOSD 2005), 14-18 March, 2005, Chicago, IL, USA.

[5] C. V. Lopes, T. C. Ngo. "The Aspect Markup

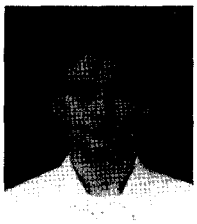
Language and its support of Aspect Plugins." ISR Technical Report UCI-ISR-04-8. University of California, Irvine. 2004.
〔6〕 http://www-128.ibm.com/developerworks/library/specification/ws-bpel/
〔7〕 B2J, "http://www.eclispe.org/stp/b2j"
〔8〕 witmate, "http://witmate.com/default.html"

최 재 영



1984 서울대학교 제어계측공학과(공학사)
1986 미국 남가주대학교 컴퓨터공학과 (석사)
1991 미국 코넬대학교 컴퓨터공학과 (박사)
1992.1~1994.2 미국 국립 오크리지 연구소 연구원
1994.3~1995.2 미국 테네시 주립대학교 연구교수
1995.3~현재 숭실대학교 컴퓨터학부 부교수
관심분야 : 유비쿼터스 컴퓨팅, HPC 컴퓨팅, 시스템 소프트웨어
E-mail : choi@ssu.ac.kr

곽 동 규



2002 서경대학교 응용수학과(학사)
2004 숭실대학교 컴퓨터학과(석사)
2004~현재 숭실대학교 컴퓨터학과 박사과정
관심분야 : 프로그래밍언어, XML 스크립트, 시스템 소프트웨어
E-mail : coolman@ss.ssu.ac.kr

이 명 철

1987 경북대학교 전자공학과 전산전공 (공학사)
1991 경북대학교 전자공학과 전산전공 (석사)
1991~현재 한국 IBM 유비쿼터스 연구소 책임연구원
관심분야 : 유비쿼터스 컴퓨팅, 보안, 임베디드 소프트웨어
E-mail : mclee@kr.ibm.com

장 현 기

1995 서울대학교(이학사)
1997 서울대학교 물리학(석사)
2002 서울대학교 물리학(박사)
1998~2000 일본 KEK(고에너지 물리) 연구소 파견연구원
2004~2005 삼성전자 S/W 센터 책임연구원
2005~현재 IBM 유비쿼터스 컴퓨팅 연구소 연구원
관심분야 : Context-awareness, Personalization, Semantic 기술 in Mobile domain
E-mail : hkjang@kr.ibm.com

백 영 상

1998 연세대학교(공학사)
1998~2000 (주)LAS21 지식관리 시스템 개발자
2000~현재 IBM 유비쿼터스 컴퓨팅 연구소 연구원
관심분야 : 유비쿼터스 컴퓨팅, 분산컴퓨팅, 미들웨어 기술, 차세대 웹 기술
E-mail : yspaik@kr.ibm.com