AOSD기반에서 Aspect의 동적결합을 위한 Connector

김 태 웅[†] · 김 태 공[††]

요 약

영역지향 소프트웨어 개발 방법론(Aspect-Oriented Software Development)은 소프트웨어의 성능을 향상시키고 유지보수에 많은 이점을 가지는 새로운 소프트웨어 개발 방법론이다. 또한 기존의 프로그래밍 언어가 제공하지 못하는 보안이나 결함 내성과 같은 부가기능에 대해 모듈화하는 방법을 제공하고 있다. 하지만 AOSD기반으로 소프트웨어를 개발하기 위해서는 Aspect를 지원하는 새로운 영역지향 프로그래밍 언어를 사용하거나 레거시 시스템에 Aspect를 적용할 경우에 소스코드의 재 컴파일등과 같은 문제점을 가지고 있다. 이에 본 논문에서는 레거시 시스템에 Aspect를 동적으로 결합할 수 있는 Connector를 제안하고 설계한다. 이를 위하여 Core와 Aspect가 가지는 오퍼레이션에 대한 정보와 XML로 기술된 pointcut정보를 이용한다. 또한 사례연구를 통하여 제안된 Connector를 사용하기 위해 새로운 영역지향 컴파일러가 필요 없고, 레거시 시스템의 수정도 발생하지 않음을 보여 그 유효성을 검증한다.

키워드 : 영역지향소프트웨어개발방법론, 레거시 시스템, 커넥터, 영역지향프로그래밍

Connector for Dynamic Composition of Aspects Based on AOSD

Taewoong Kim[†] · Taegong Kim[††]

ABSTRACT

Aspect-Oriented Software Development is new software development method. It has many advantages related to software performance, maintenance and repair. Also it offers modularization method to a existing programming language for secondary function such as security and fault tolerance. But the present problem is that we have to use new aspect-oriented programming language. Further more when we apply Aspect to legacy system, we have to recompile the source code in order to build software system based on AOSD. In this paper, we propose and design Connector that can be composed with Aspect in legacy system dynamically. To elaborate this work, we use the information of operations about Core and Aspect, and the information of pointcut described with XML. We validate that the proposed Connector has features such as no need of new compiler, no recompilation and no modification of legacy system through case study.

Key Words : Aspect-Oriented Software Development, Legacy System, Connector, Aspect, Aspect-Oriented Programming

1. 서 론

소프트웨어를 개발할 때 견고한 디자인을 바탕으로 개발된 소프트웨어는 모듈화가 잘 되어 있다. 잘 정의된 모듈화는 소프트웨어의 성능을 향상시키게 되고 차후 유지보수에도 많은 이점을 준다. 하지만 여러 가지 이유로 인하여 모든 소프트웨어들이 좋은 모듈화를 유지할 수 있는 것은 아니다.

객체지향 프로그래밍에서는 클래스 혹은 오퍼레이션(Operation)이라는 단 하나의 모듈화 방법만을 제시하고 있기 때문에 여러 클래스나 오퍼레이션에 걸쳐서 구현되어야 하는 보안이나 결함내성과 같은 부가기능들의 모듈화에는 실패하였다[1]. 클래스는 클래스 자체의 정의뿐만 아니라 함께 연동하는 다른 클래스들과 관련된 행위를 혼합된 형태(tangling)로 가지고 있다. 때문에 구성하는 클래스들이 다른 프로그램 구성요소로 사용될 경우나, 지속적인 요구사항의 변경으로 유지보수가 빈번하게 일어나게 될 때 소프트웨어는 복잡성과 같은 문제를 야기 시킬 수 있다. 특히, 스케줄링, 로깅, 문맥의존형 에러처리, 트랜잭션, 보안과 같은 기능들은 여러 다른 모듈들에 걸쳐져서(scattering) 구현이 되어야 하기 때문에 한 파일이나 한 클래스 등으로 모듈화 될 수 없으며, 이 기능들을 구현한 코드들은 흩어져서 존재할 수밖에 없다.

이처럼 여러 클래스에 유사한 기능이 중복되어 정의되어야 하거나 클래스 구조를 이용하여 모듈화를 제공할 수 없는 부가기능의 단위를 크로스커팅단위(cross-cutting concern)라고 하며, 객체지향프로그래밍의 이 같은 단점을 보완하고자 고안된 것이 영역지향프로그래밍(Aspect-Oriented

† 준 회 원 : 동의과학대학 겸임교수
†† 정 회 원 : 인제대학교 컴퓨터공학부 교수
논문접수 : 2005년 10월 6일, 심사완료 : 2006년 1월 5일

Programming)[2, 3]이다. 이는 소프트웨어를 개발하는데 있어서 필요한 부분만을 임의로 분할하고 이를 바탕으로 모듈화를 통하여 시스템을 개발하는 새로운 개념의 기술이다[4].

현재 AOSD(Aspect-Oriented Software Development) conference(http://aosd.net)를 중심으로 Aspect 관련 기술들에 대한 연구가 활발히 진행 중에 있다. 구현 중심의 AOP를 요구사항 분석에서부터 설계[5], 구현에 이르기까지 전반적인 소프트웨어 개발 공정에서 적용하고자 많은 연구들이 이루어지고 있다. 그러나 이러한 연구들은 구현중심에서 출발하여 발전하고 있기 때문에 아직 그 연구에 대한 발전 정도가 미비하다. AOSD를 기반으로 소프트웨어를 개발하거나 Aspect를 지원하지 않았던 레거시 시스템일 경우 Aspect를 지원하는 새로운 언어를 사용하여 기존의 소프트웨어 소스코드와 Aspect에 관련된 소스코드를 통합하여 재 컴파일하거나 기존 시스템의 수정, 새로운 컴파일러의 사용등과 같은 문제점을 가지고 있다. 이와 같이 개발된 Aspect는 그 Aspect를 재사용하지 못하는 단점과 기존의 소프트웨어와 결합되어 복잡성을 야기시키는 또 다른 단점을 가지게 된다.

본 논문에서는 레거시 시스템과 Aspect를 동적으로 결합하기 위한 Connector를 제안하고자 한다. 이를 위하여 Aspect의 결합정보를 나타내는 결합점(pointcut)과 행위(behavior)정보인 크로스커팅단위로 분리하고, 코어(core)정보, 크로스커팅단위정보, 결합점정보로부터 동적으로 결합되어질 수 있는 Connector를 제안하고 설계하고자 한다.

2장에서는 관련연구로서 AOSD환경에서 사용하는 AOP 언어의 특성과 그 종류에 대해 설명하고 3장에서 본 연구에서 제안한 동적결합을 위한 Connector를 제안한다. 4장에서는 적용사례를 통해 Connector자동생성의 예를 보이고 레거시 시스템과 Aspect의 동적결합의 장점을 보임으로써 그 유효성을 검증한다. 마지막 5장에서는 향후 연구과제를 포함한 결론을 맺는다.

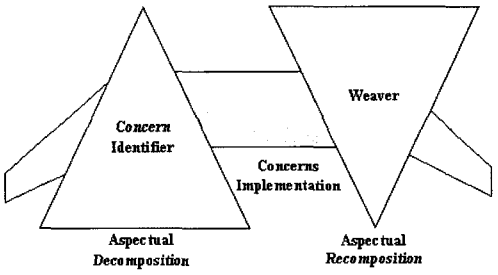
2. 관련 연구

2.1 AOP

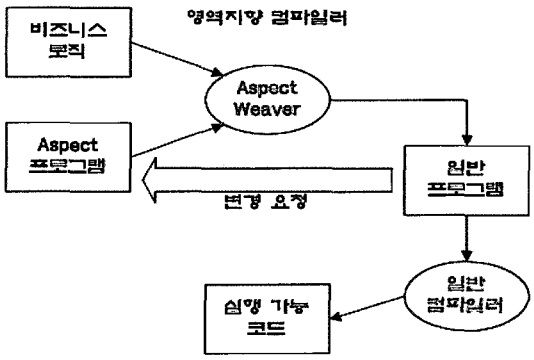
영역지향프로그래밍[3, 6]은 1997 제록스(Xerox)의 팔로알토(The Palo Alto Research Center) 연구소에 있는 Gregor Kiczales의 주도하에 연구가 진행되어 오고 있다[6].

Aspect는 기존의 프로그램 모듈이 추상화 및 인캡슐레이션 개념을 제공할 수 없는 이른바 여러 다른 기능들에 걸쳐져서 구현되는 기능으로 크로스커팅단위를 의미한다. 즉, 크로스커팅단위는 여러 클래스에 유사한 기능이 중복되어 정의되어야 하거나 전통적 클래스 구조를 이용하여 모듈화를 제공할 수 없는 부가기능이나 특성(feature)들을 의미하는 것으로 여러 모듈에서 필요한 부분만을 추출하여 새로운 모듈로 구성하게 된다[7].

(그림 1)처럼 영역지향프로그래밍의 개발 단계는 전체 요구사항에서 관심이 되는 사항을 기능별(핵심 비즈니스 로직, 보안, 트랜잭션 등)로 분리를 한 후 각 관심 사항별로 구현을



(그림 1) 영역지향 프로그래밍 개발 단계



(그림 2) 영역지향 프로그래밍의 기본 구조 및 절차

한다. 그 후 결합기(weaver)에 의해 다시 재결합함으로써 최종 시스템으로 완성되는 구조를 가지고 있다[8].

(그림 2)는 영역지향프로그래밍 언어의 처리 방법이 기본적으로 취하고 있는 구조 및 절차를 묘사하고 있다. 모듈은 기존의 기능성을 모듈화하는 핵심 비즈니스 로직인 코어와 크로스커팅단위를 모듈화하는 Aspect 모듈로 분리되어 구성된다. 이것은 다시 영역지향 언어 컴파일러에 의해 두 가지 행위가 결합된 형태의 일반 프로그램으로 번역 된다. 프로그램의 작성, 이해, 변경 작업이 모두 영역지향 컴파일러를 거치기 이전 단계에서 수행되므로 기존 비즈니스 로직인 코어 모듈과 크로스커팅단위를 정의하는 Aspect의 모듈화를 모두 지원해 줄 수 있다.

2.2 AOP Languages

영역지향 프로그래밍 방법론은 구현중심에서 출발하여 현재 요구사항 분석에서부터 설계, 구현에 이르기까지 전반적인 소프트웨어 개발 공정에 적용하고자 많은 연구들이 이루어지고 있다. 따라서 Aspect기반의 시스템 개발 및 기존의 레거시 시스템에 Aspect를 적용하기 위한 다양한 AOP를 지원하는 언어 및 방법들에 대한 연구가 활발하다[9-13]. 그러나 이러한 연구들은 동적결합을 가능하게 하기 위해 새로운 영역지향 언어를 제안[9, 12, 13]하거나, 자바 기반의 새로운 가상머신을 제안[11]하고 있어 레거시 시스템에 적용하기는 적합하지 못하다는 단점을 가지고 있다.

2.2.1 AspectJ[14]

현재 Aspect를 지원하는 프로그래밍 언어들 중 가장 많

이 사용하고 있다. 코어시스템과 Aspect가 결합되기 위해 결합점을 정의해야 하며 결합점은 상당히 복잡한 구조를 가지고 있어 AspectJ[12]를 기반으로 개발하기 위해서는 결합점과 advice에 대한 이해가 불가피 하다. 코어시스템의 소스코드와 크로스커팅단위에 해당하는 행위 코드, 여기에 결합점을 나타내는 소스코드, 이 세 가지 소스코드를 통합하여 컴파일 하여 새로운 애플리케이션을 생성하는 구조를 가진다. 따라서 이미 개발된 시스템의 유지보수보다는 Aspect를 기반으로 하는 새로운 시스템을 개발할 때 적합한 언어로 각광받고 있다. 또한 Aspect를 advice(행위, behavior)와 결합점을 하나의 클래스로 구현하고 있어 Aspect의 재사용은 거의 불가능하다.

2.2.2 JAsCo[15]

컴포넌트 기반의 개발방법론에 접근한 새로운 패러다임의 Aspect를 지원하는 언어이다. Aspect의 기능적인 측면을 담당하는 행위 부분을 hook라는 키워드로 설정하고 결합점에 해당하는 부분을 connect라는 키워드를 사용하여 표현한다. Aspect의 행위 부분과 결합점부분을 분리하여 결합점을 동적으로 로드할 수 있게 함으로써 Aspect의 재사용성은 이루어질 수 있으나 AspectJ과 마찬가지로 결합점을 정의하여 결합정보를 표현하고 있어 결합점과 advice에 대한 이해가 필수적이다. 또한 컴포넌트 기반으로 코어시스템과 Aspect의 결합을 제안하고 있어 컴포넌트 기반으로 개발된 시스템에 종속적이며 이미 개발된 레거시 시스템에 적용하고자 할 경우 코어시스템을 수정하여야 하는 단점을 가지고 있다.

2.2.3 JAC[16]

Apsect-Oriented Framework로서 동적으로 Aspect를 로드하고 크로스커팅단위를 표현하기 위해 새로운 언어의 사용을 하지 않는다는 장점을 가지고 있으며 기본적으로 JAsCO[13]의 추상적인 모델이다. 따라서 컴포넌트 기반으로 코어와 Aspect의 결합을 제안하고 있으며 이미 개발된 레거시 시스템에 Aspect를 지원하기 위해서는 기존의 컴포넌트를 변경해야 하는 작업을 가져야 한다.

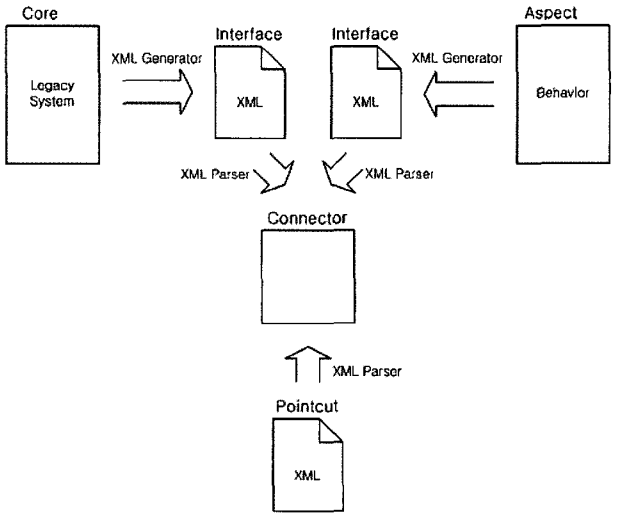
2.2.4 PROSE[17]

자바언어를 위한 라이브러리 형태로 Aspect를 지원하고 있으며 이벤트를 중간에서 가로채는 방식의 라이브러리로 구성되어 있다. 이러한 라이브러리를 사용하여 Aspect를 동적으로 결합(weaving)하고 분리(unweaving)하는 메커니즘을 제공한다. 그러나 기존의 자바 가상머신을 사용하지 않고 디버깅 모드에서 실행될 수 있는 자바 가상 머신을 사용하여야 하는 단점을 가지고 있다.

이러한 연구들은 Aspect를 지원하는 새로운 프로그래밍 언어의 사용, 새로운 컴파일러의 사용, 결합점정보를 위한 결합점과 행위 표현 방법에 대한 숙지, 레거시 시스템을 수정해야 하는 등 많은 단점을 가지고 있다. 또한 일부의 Aspect를 지원하는 언어들 중에는 Aspect의 기능을 나타내는 행위 부분에 결합정보를 나타내는 결합점부분이 포함되어 있어 Aspect자체에 대한 재사용과 유연성을 지원하지 못하고 있는 상황이다.

3. 동적 결합을 위한 Connector

본 연구에서는 레거시 시스템에 Aspect를 동적으로 결합하기위한 Connector를 제안하고 설계하고자 한다. 이를 위하여 Aspect를 Aspect의 행위인 크로스커팅단위와 결합정보인 결합점으로 분리한다. 크로스커팅단위는 비즈니스로직을 포함하고 있는 코어부분과 같은 일반적인 클래스 형태로 나타내고, 결합점 부분은 본 논문에서 제안한 XML스키마에 따라 표현한다. 결합점은 Aspect가 코어에 결합되는 위치정보를 가져야 하므로 코어에 있는 해당 객체의 오퍼레이션 정보가 필요하다. 따라서 코어와 Aspect 클래스로부터 공개 오퍼레이션을 가지는 인터페이스 정보를 추출하여 XML로 나타낸다. 이러한 XML로부터 코어와 Aspect를 동적으로 결합할 수 있는 Connector를 생성 한다. (그림 3)은 본 논문에서 제안하는 Connector의 생성과정을 표현하고 있다. 코어와 Aspect로부터 XML generator를 이용하여 오퍼레이션에 대한 정보를 XML문서로 나타내고 생성되어진 XML문서는 XML Parser기능을 가지는 generator에 의해 최종적인 Connector를 생성하게 된다.



(그림 3) Connector생성 구조

3.1 인터페이스 정보

기존의 코어와 Aspect가 동적으로 결합하기 위한 Connector를 생성하기 위해 코어와 Aspect가 가지고 있는 오퍼레이션에 대한 정보가 필요하다. 이는 Aspect의 기능이 코어 구조상에서 어떠한 위치, 시간, 상황에 의존하여 최종적으로 원하는 행위로 매핑시킬 것인가에 대하여 기술하는 결합점에 대한 정보가 필요하기 때문이다. 코어와 Aspect는 각각 클래스 형태로 표현되어 있으므로 각각의 클래스로부

Interface_schema

```
<?xml version='1.0' encoding="euc-kr"?>

<xsd:schema
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://selab.inja.ac.kr/interface"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified"
   xmlns="http://selab.inja.ac.kr/interface">

   <xsd:element name="category" type="ctCategory" />

   <xsd:complexType name="ctCategory">
      <xsd:choice>
         <xsd:element name="Core" type="ctInterface" />
         <xsd:element name="Aspect" type="ctInterface" />
      </xsd:choice>
   </xsd:complexType>

   <xsd:complexType name="ctInterface">
      <xsd:sequence>
         <xsd:element name="Operation" type="ctOperation" maxOccurs="unbounded" />
      </xsd:sequence>
      <xsd:attributeGroup ref="gBase" />
   </xsd:complexType>

   <xsd:complexType name="ctOperation">
      <xsd:sequence>
         <xsd:element name="Parameter" type="ctParameter" minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
      <xsd:attributeGroup ref="gBase" />
      <xsd:attribute name="returnType" type="xsd:string" />
      <xsd:attribute name="exception" type="xsd:string" />
   </xsd:complexType>

   <xsd:complexType name="ctParameter">
      <xsd:attributeGroup ref="gBase" />
      <xsd:attribute name="type" type="xsd:string" />
   </xsd:complexType>

   <xsd:attributeGroup name="gBase">
      <xsd:attribute name="id" type="xsd:ID" use="required" />
      <xsd:attribute name="name" type="xsd:string" use="required" />
   </xsd:attributeGroup>
</xsd:schema>
```

(그림 4) 인터페이스 정보에 대한 XML 스키마

터 얻어질 수 있으며 공개 오퍼레이션의 집합을 인터페이스로 묶어 XML정보로 관리한다. (그림 4)는 본 논문에서 제안하는 코어와 Aspect 클래스로부터 추출한 인터페이스 정보에 대한 XML 스키마 구조이다.

3.2 결합점(pointcut) 정보

Aspect의 결합정보를 나타내는 결합점은 Aspect의 기능이 코어 구조상에서 어떠한 위치, 시간, 상황에 의존하여 최종적으로 원하는 행위로 변형시킬 것인가에 대하여 기술하는 것으로 위치정보와 시간정보, 상황정보가 포함된다. 위치정보는 Aspect가 코어에 결합되는 위치로서 코어에 있는 해당 객체의 오퍼레이션 정보이다. 시간정보는 Aspect가 코어에 결합되는 시점에 관한 advice 정보로서 before, after, around, after returning, after throwing 정보가 포함된다. 따라서 (그림 5)와 같은 형태의 결합점정보에 대한 XML 스키마를 정의할 수 있다.

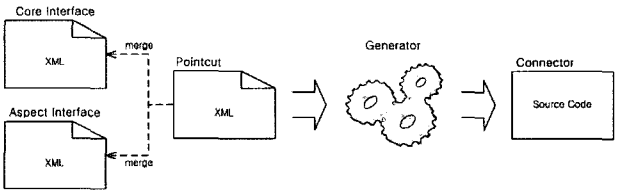
3.3 Connector의 생성

코어와 Aspect의 행위부분에서 추출된 인터페이스 정보와 결합점정보를 통하여 코어와 Aspect의 두 부분을 동적으로 결합하는 Connector를 생성한다. 즉, 3.1에서 생성된 인터페이스에 대한 XML문서와 3.2에서 생성된 결합점에 대한 XML문서를 통합하여 Connector을 생성하기 위한 최종적인 XML문서가 생성된다. 이러한 과정으로 생성되어진 XML문서를 해석하고 파싱하는 generator에 의해 코어와 Aspect를 동적으로 결합할 수 있는 Connector의 소스코드가 생성된다.

Pointcut_schema

```
<?xml version='1.0' encoding="euc-kr"?>

<xsd:schema
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://selab.inja.ac.kr/pointcut"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified"
   xmlns="http://selab.inja.ac.kr/pointcut">

   <xsd:element name="category" type="ctCategory"/>

   <xsd:complexType name="ctCategory">
      <xsd:sequence>
         <xsd:element name="Connector" type="ctConnector"/>
      </xsd:sequence>
   </xsd:complexType>

   ...

   <xsd:complexType name="ctPointcut">
      <xsd:sequence>
         <xsd:element name="Call" type="ctJoinpoint" minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element name="Execution" type="ctJoinpoint" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:string" use="required"/>
   </xsd:complexType>

   <xsd:complexType name="ctJoinpoint">
      <xsd:sequence>
         <xsd:element name="Advice" type="ctAdvice" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="idref" type="xsd:IDREF" use="required"/>
   </xsd:complexType>

   <xsd:complexType name="ctAdvice">
      <xsd:sequence>
         <xsd:element name="Args" type="ctArgs" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="idref" type="xsd:IDREF" use="required"/>
      <xsd:attribute name="type" type="stAdviceType"/>
   </xsd:complexType>

   <xsd:complexType name="ctArgs">
      <xsd:attribute name="idref" type="xsd:string" use="required"/>
      <xsd:attribute name="target" type="xsd:string" use="required"/>
      <xsd:attribute name="type" type="stArgsType"/>
   </xsd:complexType>

   <xsd:simpleType name="stAdviceType">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="before"/>
         <xsd:enumeration value="after"/>
         <xsd:enumeration value="around"/>
      </xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="stArgsType">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="parameter"/>
         <xsd:enumeration value="operation"/>
      </xsd:restriction>
   </xsd:simpleType>
</xsd:schema>
```

(그림 5) 결합점(pointcut) XML 스키마



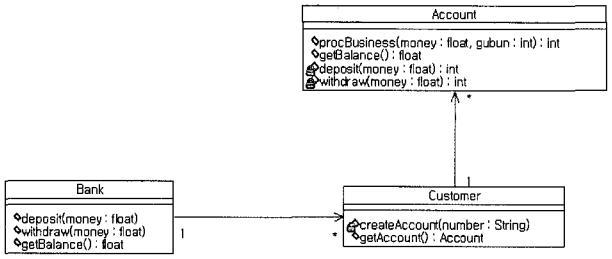
(그림 6) Generator에 의한 Connector의 생성

4. 적용 사례

이번 장에서는 은행 시스템을 레거시 시스템으로 하여 다음과 같은 3가지 방법으로 Aspect를 적용했을 경우 레거시 시스템의 구조적인 변화를 클래스 다이어그램을 통하여 살펴본다. 이를 통하여 본 논문에서 제안한 Connector를 사용하여 Aspect를 레거시 시스템에 적용할 경우 레거시 시스템의 수정없이 동적결합이 가능함을 보인다. 또한 본 논문에서 제안한 동적결합 가능한 Connector를 이클립스 기반의 플러그인을 통하여 생성한다. 레거시 시스템은 (그림 7)과 같은 은행시스템의 일부로 마이너스 계좌의 개설 및 사용이 가능한 시스템이며 적용할 Aspect는 마이너스 계좌의 개설

및 사용이 금지되는 BalanceCheck클래스를 모델로 한다.

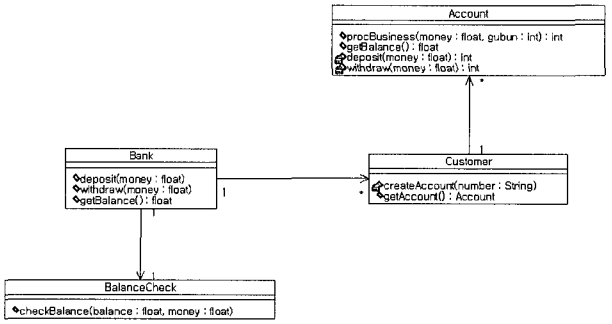
- Aspect를 지원하지 않는 전통적인 방법을 사용한 Aspect의 적용
- AspectJ를 이용한 Aspect의 적용
- 본 논문에서 제안한 Connector를 사용한 Aspect의 적용



(그림 7) 레거시 시스템

4.1 Aspect를 지원하지 않는 경우 레거시 시스템의 수정

Aspect를 지원하지 않는 시스템에서 기능적인 추가나 정책적인 변화에 따른 시스템의 유지 보수의 방법은 기존의 시스템을 분석하여 수정을 가하는 방식의 시스템의 전면적인 수정을 요구한다. 이는 개발시간과 비용의 측면에서 많은 문제점을 가지고 있다. (그림 8)은 (그림 7)과 같은 레거시 시스템을 Aspect를 지원하지 않는 방법으로 수정한 경우 시스템의 구조를 나타낸 클래스 다이어그램이다. Bank클래스가 BalanceCheck클래스의 존재를 인식할 수 있는 단방향 연관관계가 성립해야 하므로 레거시 시스템의 수정이 불가피하다.

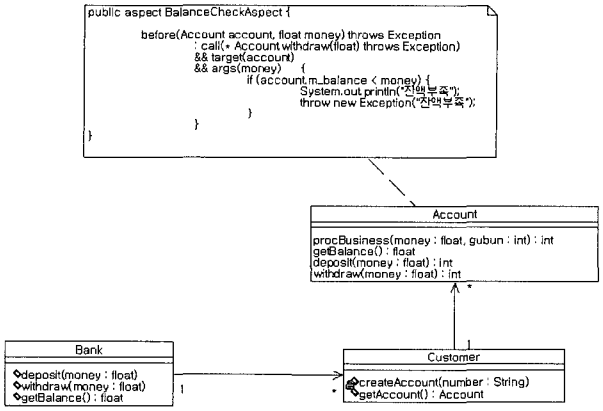


(그림 8) Aspect를 지원하지 않는 경우 BalanceCheck클래스의 추가

4.2 AspectJ를 사용한 레거시 시스템의 수정

4.1과 같은 문제점을 해결하기 위하여 AOP 언어를 사용하여 레거시 시스템을 수정하고자 할 경우 기존의 AOP관련 프로그래밍 언어들은 소스를 재 컴파일하거나 기존 레거시 시스템의 수정, 새로운 컴파일러의 사용등과 같은 단점을 가지고 있다.

(그림 9)는 AspectJ언어를 사용하여 (그림 7)의 은행 시스템을 수정할 경우 시스템의 구조를 나타내는 클래스 다이

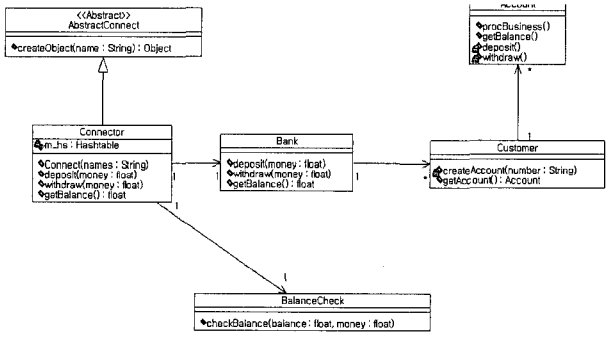


(그림 9) AspectJ 기반에서 BalanceCheck Aspect의 추가

어그램이다. 이미 개발된 레거시 시스템에 Aspect를 적용하기 위해 결합점정보를 메모 형태로 삽입한 모습이다. 결합점은 AspectJ에서 제공하는 문법을 이용하여 작성하였으며 기존의 레거시 시스템의 소스코드와 결합하여 재 컴파일해야 한다.

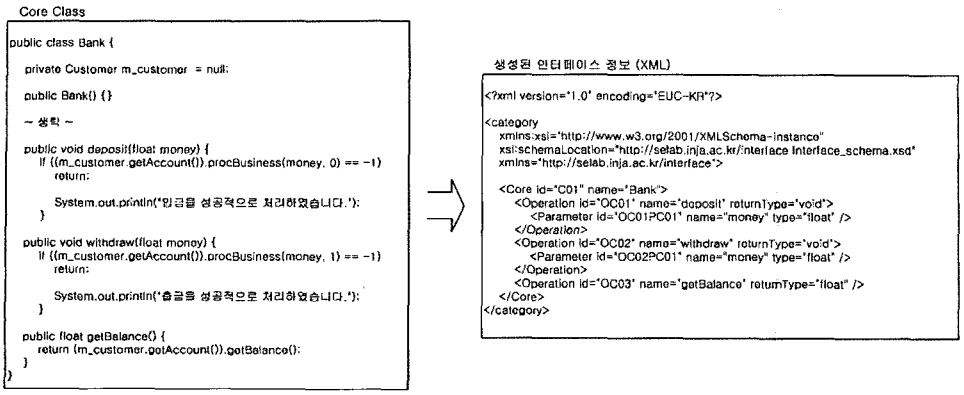
4.3 Connector를 이용한 Aspect의 동적 결합

(그림 7)과 같은 기존의 레거시 시스템에 수정을 가하는 방법으로 (그림 8)과 같이 기존의 레거시 시스템구조를 변경하는 작업과 (그림 9)와 같이 Aspect를 지원하는 프로그래밍 언어인 AspectJ를 사용하여 수정을 가하는 방법을 보여 주었다. Aspect를 지원하지 않는 방법으로 레거시 시스템의 구조를 변경하는 작업에 비해 AspectJ를 사용하여 수정을 가하는 방법이 많은 장점을 가지고 있으나 이 또한 기존의 소스코드와 같이 재 컴파일해야 하는 단점을 여전히 가지고 있다.

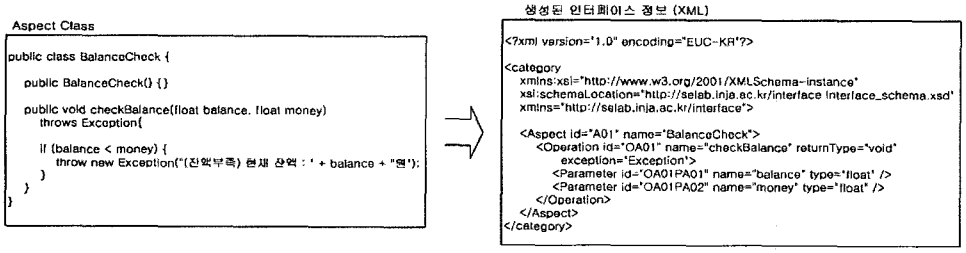


(그림 10) Connector를 적용한 후의 클래스 다이어그램

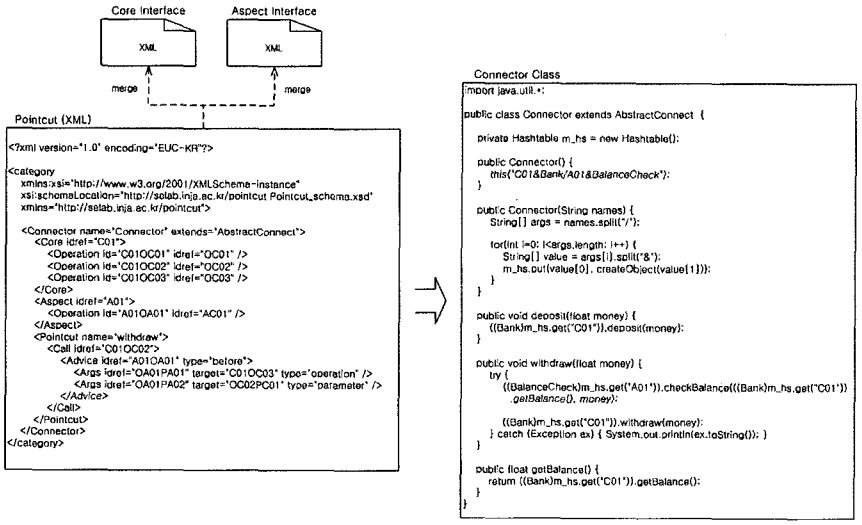
(그림 10)은 본 논문에서 제안한 Connector를 이용하여 레거시 시스템과 Aspect를 동적으로 결합하는 구조를 정적인 클래스 다이어그램을 사용하여 표현하고 있다. Connector 클래스는 코어인 Bank 클래스, Aspect의 행위인 Balance Check 클래스, 이 두 클래스를 동적으로 결합하기 위한 정보인 결합점정보로부터 생성된다. Connector클래스의 기능



(그림 11) Bank클래스로부터 추출된 인터페이스 정보



(그림 12) BalanceCheck클래스로부터 추출된 인터페이스 정보



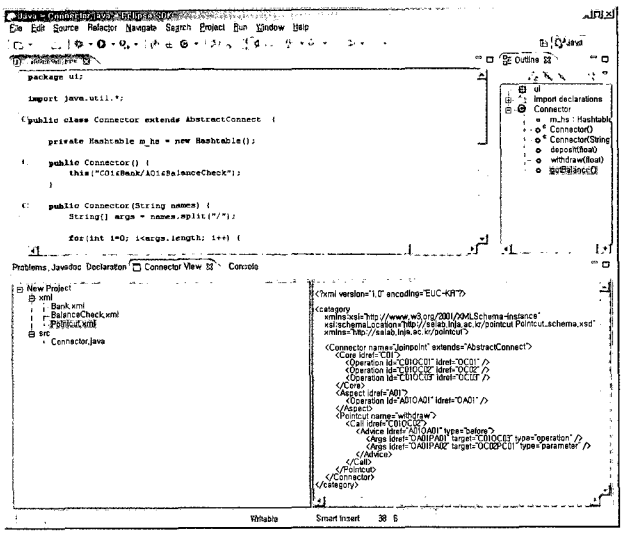
(그림 13) 최종 생성된 Connector클래스

은 Bank클래스와 BalanceCheck 클래스 사이에서 이 두 클래스에 존재하는 오퍼레이션의 호출을 중간에서 가로채어 그 순서와 시점을 제어한다. Connector 클래스는 Bank클래스와 BalanceCheck클래스의 단방향 연관을 위한 정보를 가지고 있는 AbstractConnector클래스를 상속하고 Bank클래스가 가지는 모든 오퍼레이션에 대한 정보를 가지고 있다. 또한 레거시 시스템의 Bank 클래스는 새로 추가된 Connector클래스와 관련된 어떠한 다른 클래스와도 연관관계를 가지지 않는다. 이는 레거시 시스템을 수정하지 않고도 Connector에 의해 동적으로 Aspect와 결합될 수 있음을 의미한다. 즉, Bank클래스는 Connector클래스의 존재를 알 필요 없고, Connector클래스만이 Bank클래스의 존재를 인식하는 단방향 연관으로 되어있어 레거시 시스템의 수정없이 Connector를 적용할 수 있다.

(그림 11)과 (그림 12)는 Bank클래스와 BalanceCheck클래스로부터 추출된 인터페이스 정보를 XML형태로 표현하고 있다.

레거시 시스템인 은행 시스템과 여기에 적용할 Aspect의 행위에 해당하는 클래스들은 자바언어로 작성되고 이미 컴파일되어 있는 바이트코드만 존재하는 경우를 가정하였으며

결합점은 본 논문에서 제안한 결합점을 나타내기 위한 XML스키마에 근거하여 직접 XML문서를 작성하여 적용하여 보았다. (그림 13)은 두 개의 바이트코드 형태의 클래스에서 생성된 XML문서와 결합점정보를 가지는 XML문서로부터 생성된 Connector클래스의 소스코드이며, (그림 14)는 이와 같은 과정을 이클립스 기반의 플러그인으로 개발하여 실행한 모습이다. 화면상단에 생성된 Connector의 소스코드가 있으며 왼쪽 하단에는 코어, Aspect,결합점에 대한 XML 문서가 있다.



(그림 14) 이클립스 기반의 플러그인

<표 1>에서 기존의 레거시 시스템에 Aspect를 적용하기 위한 방법으로 영역지향 프로그래밍 언어를 사용하지 않은 경우, AspectJ라는 새로운 영역지향 프로그래밍 언어를 사용한 경우와 본 논문에서 제안한 Connector를 사용한 경우를 비교하였다.

〈표 1〉 각 방법론에 대한 비교

특징	기존의 방법론	AspectJ 사용	동적결합 방법론(제안)
Recompile	예	예	아니오
Aspect 재사용성	낮음	낮음	높음
레거시 시스템의 수정	예	예	아니오
응답속도	빠름	빠름	다소느림
유지보수성	낮음	낮음	높음

5. 결론 및 향후연구

소프트웨어의 성능을 향상시키고 유지보수에 많은 이점을 가지는 새로운 소프트웨어 개발 방법론인 영역지향 소프트웨어 개발방법론(Aspect-Oriented Software Development)은 기존의 프로그래밍 언어가 제공하지 못하는 보안이나 결함 내성과 같은 부가기능에 대해 모듈화하는 방법을 제공하고 있다. 그러나 이러한 개발방법론을 사용하여 레거시 시스템의 수정과 같은 유지보수를 가능하게 하기 위해서는 AOP 언어를 사용하여야 하며 레거시 시스템 소스 코드와의 재컴파일등과 같은 문제점을 가지고 있다.

본 논문에서는 이러한 문제점을 해결하기 위하여 새로운 프로그래밍언어의 사용이나 기존 시스템 소스코드와의 재컴파일이 필요 없는 동적 결합 가능한 Connector를 제안하고 설계하였다. 레거시 시스템과 Aspect를 동적으로 결합하는 Connector를 생성하기 위하여 Aspect의 기능부분인 크로스커팅단위부분과 결합정보를 나타내는 결합점부분을 분리하고 크로스커팅단위부분은 일반적인 클래스 형태로 나타내며 결합점부분은 XML을 사용하여 정보를 표현하였다. 이 XML정보는 코어클래스와 Aspect클래스로부터 추출한 인터페이스 정보와 함께 Connector를 생성하는데 사용되었다.

본 연구를 통하여 AOSD기반으로 기존의 레거시 시스템에 Aspect를 추가하고자 할 경우 새로운 AOP 언어를 사용하지 않고도 Aspect를 적용할 수 있었다. 또한 적용사례를 통해 코어클래스나 Aspect의 행위클래스를 이미 컴파일되어진 바이트코드를 그 입력 모델로 함으로써 기존의 레거시 시스템 소스코드와의 재 컴파일등의 문제를 없앨 수 있었다. 뿐만 아니라 코어와 Aspect를 결합하는 결합점정보를 Aspect의 기능을 나타내는 크로스커팅단위와 분리하여 나타내고 결합점정보만을 수정함으로써 Aspect를 재사용 할 수 있는 장점도 가진다.

현재 본 논문에서 제안한 기존의 레거시 시스템과 Aspect와의 동적 결합을 위한 Connector는 다른 이클립스 기반의 모델관련 플러그인과 연계하여 실행 가능하도록 재개발 중에 있으며 동적 결합을 실시간으로 가능하게 하는 미들웨어에 대한 연구가 진행중에 있다. 또한 코어와 Aspect의 결합정보를 나타내는 결합점정보를 보다 쉽게 추출할 수 있도록 UML로 모델링하는 방법에 대한 연구가 진행중에 있다. 이러한 연구들은 향후 AOSD기반의 시스템개발을 지원하는 CASE 툴 개발에 유용하게 사용될 것으로 기대한다.

참 고 문 헌

[1] http://www-106.ibm.com/developerworks/java/library/j-aspectj/index.html

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

[3] 이준상, "미래소프트웨어 개발 기술 : Aspect-Orinted-Programming과 Subject-Orient-ed-Programming", 정보처리학회지, Vol.10 No.5 pp.94-101, 2003.

[4] http://www.aosd.net

[5] 김치수, 김태영 "영역지향프로그래밍 기술을 적용한 CBD방법론", 정보처리학회논문지D, 제11-D권 제7호, pp.1435-1442, 2004.

[6] G. Kiczales and et al, "An Overview of AspectJ(0.8)", in proceedings of Europe-an Conference for Objects-Oriented Programming, 2001.
[7] http://aspectj.org/doc/dist/progguide/index.html
[8] Ramnivas Laddad, "AspectJ in Action", Manning Publications Co., 2003.
[9] Andrei Popovici, Gustavo Alonso, Thomas Gross, "Just-In-Time Aspects: Effici-ent Dynamic Weaving for Java", AOSD Conference, pp.100-109, 2003.
[10] Carine Courbis, Ahthony Finkelstein, "Towards Aspect Weaving Application", IC-SE, pp.69-77, 2005.
[11] Christoph Bockisch, Michael Haupt, "Virtual Machine Support for Dynamic Join Points", AOSD Conference, pp.83-92, 2004.
[12] Yoshiki Sato, Shigeru Chiba, "A Selective, Just-In-Time Aspect Weaver", GPCE, LNCS Vol.2830, pp.189-208, 2003.
[13] Adnrei Popovici, Thomas Gross, Gustavo Alonso, "Dynamic Weaving for Aspect-Oriented Programming", AOSD Conference, pp.141-147, 2002.
[14] AspectJ Website, http://www.aspectj.org
[15] Davy Suvee, "JAsCO:an Aspect-Oriented approach tailored for Component Based Software Development", AOSD Conference, pp.21-29, 2003.
[16] R. Pawlak, L. Seinturier, L. Duchien, and G. Florin. Jac: A Flexible solution for Aspect-Oriented Programming in Java. In Proceedings of Reflection, LNCS, Vol.2192, pp.1-21, 2001.
[17] Popovici, A., Gross, T. and Alonso, G., "Dynamic Weaving for Aspect-Oriented Programming", In Proceedings ofthe 1st international conference on Aspect-Ori-ented Software Development, pp.141-147, 2002.

김 태 웅

e-mail : twkim@cs.inje.ac.kr
1994년 인제대학교 전산학과(이학사)
1998년 인제대학교 전산학과(이학석사)
2002년 인제대학교 전산학과(박사과정 수료)
1999년~현재 동의과학대학 겸임교수
관심분야 : 소프트웨어 공학, 영역지향 프로그래밍, 소프트웨어 개발 방법론

김 태 공

e-mail : ktg@cs.inje.ac.kr
1983년 서울대학교 계산통계학과(학사)
1985년 서울대학교대학원 계산통계학과 전산과학전공(이학석사)
1994년 서울대학교대학원 계산통계학과 전산과학전공(이학박사)
2003년~2004년 The University of Texas at Dallas 방문 교수
1990년~현재 인제대학교 컴퓨터공학부 교수
관심분야 : 소프트웨어 공학, Model Engineering, AOSD, Generative Programming