닷넷 프레임워크에서 클래스 최적화를 위한 추상구조트리 생성 및 크로스커팅 위빙 메커니즘

AST Creating and Crosscutting Concern Weaving Mechanism for Class Optimization in .NET Framework

이승형, 박제연, 송영재

경희대학교 컴퓨터공학과

Seung-Hyung Lee(shlee7@khu.ac.kr), Je-Yeon Park(jy_bak@khu.ac.kr), Young-Jae Song(yjsong@khu.ac.kr)

요약

엔터프라이즈 시스템은 점점 복잡해지고 대형화되고 있다. 시대적 흐름에 따라 재사용에 초점을 맞춘 객체지향 프로그래밍 방법으로 시스템을 개발하고 있다. 하지만, 객체지향 방법에서는 core class에 중복되는 코드가 삽입되기 때문에, 생산성 저하, 새로운 요구사항을 적용하기 어려운 문제가 발생한다. 이 단점을 해결하기 위하여, 메타데이터와 크로스커팅 개념을 적용하는 위빙 메카니즘을 제안한다. 클래스 최적화와 다른 언어사이의 통합을 위하여 다음의 방법을 사용한다. 리플렉션을 이용한 메타데이타 생성, 추상구조트리로의 변환, 그리고 XML로 명세된 크로스커팅 정보를 통한 매핑을 이용한다. 제안하는 방법을 이용하여, 기능의 분산과 코드의 혼란을 해결함으로서 클래스를 최적화 할 수 있다.

■ 중심어 : | 클래스 최적화 | 메타데이터 | 크로스커팅 개념 | 추상구조화트리 |

Abstract

The enterprise system is becoming more complex and larger. With the changes of the times, the system is developing to object-oriented programming method(OOP). However, the same code inserts to the core class repetitiously in the OOP, that causes a decrease in productivity and a trouble of application of another requirement. To solve this weak point, we propose a weaving mechanism what applies to metadata and crosscutting concern. For a class optimization and an integration between different languages, we take the following way. This paper uses three ways, those are, metadata generation using reflection, transformation to Abstract Syntax Tree, and mapping through crosscutting information specified XML. Through the proposed theory, class optimization can be accomplished by solving a functional decentralization and a confusion of codes.

■ keyword : | Class Optimization | Metadata | Crosscutting Concern | Abstract Syntax Tree |

I. 서 론

객체지향 프로그래밍의 가장 큰 특징은 시스템이 다루고자 하는 도메인 모델을 설계와 구현에 손쉽게 반영할 수 있다는데 있다. 하지만, 복잡한 시스템에서는 요구사항과 기능을 모두 충족시키면서 객체지향 설계를

접수번호 : #100121-002

접수일자 : 2010년 01월 21일

심사완료일 : 2010년 01월 28일

교신저자 : 이승형, e-mail : shlee7@khu.ac.kr

하는 것이 어려워진다. 객체지향 프로그래밍의 장점을 살리지 못하는 어려움은 크게 두 가지로 나타난다. 기능의 분산과 코드의 혼란이다. 두 가지 문제를 해결하기 위하여 크로스커팅 개념을 이용한 개발 방법이 주목받기 시작하였다[1].

기존의 크로스커팅 개념을 적용할 수 있는 방법은 자바 플랫폼에서 지원되는 AspectJ, HyperJ, AspectWerkz 등이 있다. AspectJ[2][3]는 자바 언어 키워드 외 다른 어스펙트 키워드를 이용해 구현되기 때문에, 일반적인 자바 컴파일러가 아닌 특별한 AspectJ 컴파일러를 필요로 한다. HyperJ[4]는 구현 메커니즘은 바이트 코드에 적용되기 때문에 재컴파일을 필요로 하지 않으나, 정적인 위빙 만을 지원한다. AspectWerkz[5]는 일반 클래스와 메소드를 이용하여 구현하는 어드바이스와 달리 복잡한 문법이 필요한 포인트컷은 별도의 XML 파일을 이용하여 기존의 문제점을 해결하였으나, 자바 환경에서만 사용할 수 있다는 단점을 가지고 있다. 현재 프로그래밍 개발 환경은 닷넷 환경으로 전향되고 있다. 위의 방법들은 공통적으로 자바언어를 기반으로 하는 문제점이 있다.

닷넷 환경에서 크로스커팅을 적용하는 방법은 CLAW, Aspect# 등이 있다. CLAW[7]는 C#을 위한 방법이다. 특징은 CLR 환경에서의 크로스커팅 개념을 적용할 수는 있으나, 실행 엔진을 이용한 런타임 환경에서만 위빙이 가능하고 C# 같은 제한된 언어만을 지원한다. Aspect#[8]은 크로스커팅 선언 을 기본적으로 제공하는 CLI를 위한 AOP 지원 호환 프레임워크이다. XML 커넥터를 이용하여, 크로스커팅을 MSIL에서 컴파일하고 어셈블리에서 합성된다. 실행은 메타데이터 API와 닷넷 프레임워크를 이용하여 얻을 수 있다. 이 접근 방법의 제한은 관리되는 코드만 사용할 수 있다. 그리고 프로파일링 AOP는 관리되는 코드로 조정할 수 없는 단점이 있다[8].

본 논문에서는 기존의 방법들에서 발생되는 문제를 분석, 해결하고자 한다. 객체지향프로그래밍 방법에서 발생하는 생산성 저하와 재사용의 문제, 변경된 코드에 의한 재컴파일 문제, 새로운 키워드를 사용함으로서 자체 컴파일러를 제작하여야 하는 문제, 닷넷 프레임워크 환경에서 개발자에게 클래스를 이용하여 크로스커팅 개념을 적용할 수 있도록 하는 것이다. 기존의 제한된 환경과 언어를 지원하는 방법을 개선하여 닷넷 프레임워크에서 지원하는 서로 다른 언어의 호환을 지원하고 통합을 가능하게 하는 것이 목적이다. 위의 문제를 해결하기 위하여 메타데이터를 사용하는 위빙 메카니즘을 제안한다.

논문의 구성은 다음과 같다. II장에서는 크로스커팅의 개념과 기존의 위빙 메카니즘을 기술한다. III장에서는 메타데이터를 이용하여 크로스커팅에 적용하는 방법에 대하여 기술한다. IV장에서는 메타데이터를 이용한 크로스커팅 위빙 프로세스에 대하여 기술한다. V장에서는 객체지향과 크로스커팅 설계를 비교 평가한다. 마지막으로 VI장에서는 결론 및 향후 연구에 대하여 기술한다.

II. 관련연구

1. 크로스커팅 개념

시스템에는 주관심사(Primary concern) 혹은 고려해야 할 사항이 있고, 이것에 의해 시스템이 모듈화된다. 이러한 방식에는 처리해야 할 문제가 있다. 주요 관심사에 의해 시스템을 모듈화하고 나면 나머지 부관심사(Secondary concern)에 해당하는 코드는 여기저기에 분산될 수밖에 없게 된다. 어스팩트는 기능의 분산과 코드의 혼란을 해결하기 위한 크로스커팅 개념의 구현 방법이다. 크로스커팅 개념은 어플리케이션을 가로질러 무분별하게 분산되는 것을 해결한다. 크로스커팅은 분산과 혼란을 해결하기 위한 목적과 상호작용 포인트를 갖는다. 어스팩트지향 프로그래밍을 활용한 구현을 위하여 [그림 1]과 같이 조인포인트, 포인트컷, 어드바이스, 위빙의 개념을 사용한다. 조인포인트는 메소드 실행이나 예외를 다루는 것처럼 모듈실행 동안의 특정 포인트로 나타낸다. 포인트컷은 분리된 기능을 결합시키기 위한 규칙이다. 이는 삽입 포인트 선택과 크로스커팅 위빙을 위한 패턴의 집합으로 나타난다. 어드바이스는 특정 조인포인트에서 어스팩트에 의하여 수행되는

행위(Behavior)이다. 어드바이스를 이용하여 언제 공통 기능을 핵심 클래스에 적용할지를 정의한다. 핵심 클래스와 분리된 추가기능을 결합시키는 것을 위빙이라 한다[1][8][9].

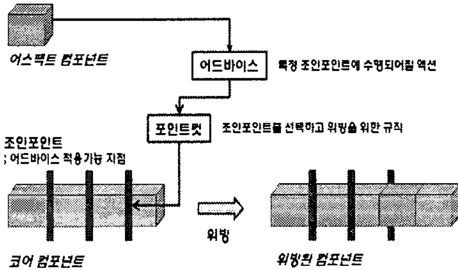

그림 1. 크로스커팅 개념을 이용하는 어스펙트 지향 프로그래밍

2. 기존의 크로스커팅 위빙 메카니즘

AsepctJ[2][3]는 자바 언어를 기반으로 크로스커팅 개념을 적용하기 위한 최초의 방법이다. 크로스커팅 하기위한 코드를 클래스 형태로, 「Aspect」란 키워드를 사용해 어스팩트나 포인트컷, 어드바이스를 생성한다. 자바 컴파일러로는 컴파일이 불가능하고 AspectJ를 위해 제작된 컴파일러를 사용해야 한다. 만들어진 바이너리는 표준 JVM에서 동작 가능한 구조로 되어있기 때문에 특별한 클래스 로더의 지원 없이도 실행 가능한 장점도 있지만, 위빙이 컴파일에서 이루어지기 때문에 포인트컷에 의해 선택된 모든 클래스는 어스팩트가 변경될 때 마다 재컴파일이 필요하다.

HyperJ[4]는 자바언어를 기반으로 한다. 크로스커팅 식별과 캡슐화 지원을 위한 구조를 포함한다. 크로스커팅 개념은 클래스 구조를 이용하여 합성된다. HyperJ의 구현 메커니즘은 바이트 코드에 적용된다. 소스코드의 합성은 컴파일 단계에서 정적인 방법으로 합성된다.

AspectWerkz[5]는 AspectJ와 달리 자바 언어를 확장하지 않고, 자바 표준클래스를 이용해서 AOP를 구현해 낼 수 있다. 일반 클래스와 메소드를 이용하여 구현하는 어드바이스와 달리 복잡한 문법이 필요한 포인트컷은 별도의 XML 파일을 이용하여 설정할 수 있다. 자바 클래스와 XML 설정 파일의 매치를 사용한 위빙은 특별한 클래스 로더를 이용한 로딩타임 바이트코드 생성을 이용한다.

JBose AOP[6]는 프록시를 이용한 인터셉터 체인(Interceptor Chain)을 활용하여 위빙을 처리하는 것이 특징이다. 필요에 따라 JavaAssist를 통한 바이트코드 조작을 이용하기도 한다. JBossAOP는 원래 JBoss 서버의 EJB를 위한 인터셉터 체인 기술을 통하여 발전되었다. JBoss는 최초로 디플로이 시점의 코드 생성이 아닌 인터셉터 체인을 이용한 방식으로 EJB 호출과 그 사이에 필요한 엔터프라이즈 서비스 기능의 삽입의 구현이 가능하다.

CLAW[7]는 .NET 환경에서 지원되는 방법이다. 런타임에서 기본 코드와 어스팩트 코드를 합성하는 실행 엔진을 포함한다. 위빙은 컴파일러에 의해 생성되는 위빙 명령어를 통하여 이루어진다. 합성 메커니즘은 언어 확장을 지원하지 않는다. 구현 메카니즘은 CLR 환경에서 MSIL에 적용된다.

AOP#[8]은 J2EE EJB 컨테이너 모델과 유사하다. XML 커넥터라 불리는 클래스를 이용하여, 핵심 클래스와 어스팩트 클래스는 MSIL에서 컴파일되고 어셈블리에서 합성된다. 핵심 클래스에 어스팩트 클래스를 삽입은 런타임에서 실행된다. 이 방법의 제한은 관리되는 코드에서만 사용될 수 있다.

3. 닷넷 프레임워크

닷넷 프레임워크의 특징은 코드관리 환경이다. 닷넷이 지원하는 다른 언어로 코딩한 것이 호환성을 유지하며, 상호 운영성을 보장하며 실행될 수 있도록 만드는 것이 CTS(Common Type System)와 CLS(Common Language Specification)이다. 닷넷 프레임워크에서는 CTS, CLS을 기반으로 하기 때문에, C#에서 작성한 클래스나, VB.NET 등에서 작성한 클래스나 동일한 환경에서 실행된다. CTS는 값 타입(value type), 레퍼런스 타입(reference type), 클래스, 인터페이스, 위임, 인터페이스, 포인터, 열거형과 같은 다양한 타입을 지원한다.

III. 크로스커팅 적용을 위한 메타데이터

시스템 개발에서 발생하는 생산성 저하와 재사용의 문제, 변경된 코드에 의한 재컴파일 문제, 새로운 키워드를 사용함으로서 자체 컴파일러를 제작하여야 하는 문제를 해결하기 위하여 클래스 메타데이터와 크로스커팅 개념을 결합하는 방법을 제안한다.

1. 소스코드에서의 메타데이터 추출

핵심 클래스에 크로스커팅하기 위하여, 각 클래스의 구조를 분석하여 메타데이터의 개념을 사용한다. 클래스는 일반적으로 네임스페이스, (생성자/멤버/호출) 메소드, 이벤트, 필드, 속성으로 구성된다. 각 클래스를 구성하는 요소(element)들이 바로 메타데이터가 된다. 핵심/어스팩트 클래스의 구조 정보를 분석하여 메타데이터로 활용한다. 클래스의 구조를 분석하여 메타데이터와 추출하여 다른 언어 사이의 구조와 실행의 호환성을 지원하여 필요로 하는 클래스 단위의 합성을 가능하게 할 수 있다. 메타데이터의 사용하는 이유는 닷넷 프레임워크에서 지원하는 다른 프로그래밍 언어와 컴파일러의 확장을 위함이다. 추출된 클래스의 구조는 [그림 2]와 같다. 그래프 엘리먼트의 루트는 「CodeCompileUnit」이 되고, 네임스페이스, 클래스 등의 추출된 메타데이터 요소는 계층적으로 표현한다.

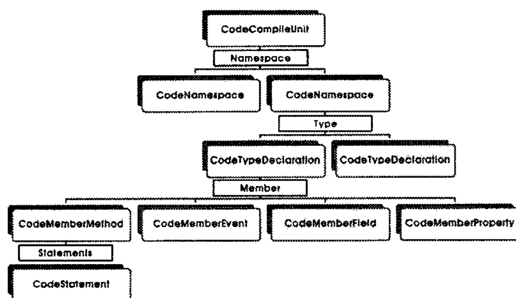

그림 2. 클래스 분석을 통하여 추출된 메타데이터 구조

2. 메타데이터를 활용한 크로스커팅의 적용

어스팩트 지향 프로그래밍에서는 핵심 클래스/ 어스팩트 클래스에 크로스커팅 개념을 적용하기 위하여 조인포인트, 포인트컷, 어드바이스 등의 합성정보를 필요로 한다[그림 1].

크로스커팅 위빙을 위하여, 우선적으로 조인포인트를 식별, 식별된 조인포인트에서 필요로 하는 기능을 명세하여야 한다. 명세를 통하여, 어스팩트 클래스가 삽입되는 위치인 조인포인트를 식별할 수 있다. [표 1]은 사용할 수 있는 조인포인트를 나열하였다.

표 1. 지원하는 조인포인트

조인포인트 구분	조인포인트 타입
Execution	method execution
	Constructor execution
	handler execution
Call	method call
	constructor call
Field Access	field reference
	field assignment

크로스커팅 기능을 구현하려면 포인트컷이라고 하는 프로그래밍 구조를 사용하여 필요한 조인포인트를 선택해야 한다. 포인트컷은 조인포인트를 선택하고 선택된 조인포인트에서 문맥(context)를 수집한다. 포인트컷은 프로그램의 기존 엘리먼트의 속성을 지정한다. 이 작업의 주요 부분과 필요로 하는 어스팩트를 적용하기 위하여 포인트컷을 작성한다. 이 포인트컷을 작성하는 것이 크로스커팅을 지원하는 핵심이다. 조인포인트를 선택하는 방식은 클래스 분석을 통하여 생성된 메타데이터(구성 엘리먼트)의 속성을 활용하는 것이다. [표 2]는 조인 포인트를 식별할 때 사용할수 있는 포인트컷를 나열하였다. 개개의 문맥 속성을 결합하면 좀 더 복잡한 포인트컷을 만들 수 있다. 크로스커팅 통합에 적용될 때 프로그램 엘리먼트 속성 기반 조인포인트 모델은 유용하게 사용할 수 있다.

어드바이스는 조인포인트에 수행될 기능이고 중요한 점은 기능이 언제 수행될지 하는 것이다. 수행되는 시점은 조인포인트와의 전후 관계에 따라서 around, before, after로 정의할 수 있다. Before 어드바이스는 조인포인트 실행 이전에 어드바이스가 동작(behavior)하게 된다.

표 2. 지원하는 포인트컷

포인트컷 구분	Primitive
Signature and type based	execution(signature)
	call(signature)
	get(signature)
	set(signature)
	handler(type pattern)
	initializer(type pattern)
	staticInitializer(signature)
	object initialization(signature)
	within(type pattern)
	withincode(signature)
Control Flow	cflow(pointcut)
	cflowbelow(pointcut)
Context	this(type pattern/variable)
	target(type pattern/variable)
	args(type pattern/variable)

Ⅳ. 닷넷 프레임워크에서 크로스커팅 위빙 프로세스

중복된 코드 제거, 최적화를 위한 크로스커팅 위빙 프로세스는 [그림 3]과 같이 3단계로 구성된다. 첫 번째 단계는 핵심/어스팩트 클래스의 메타데이터를 추출하는 단계, 두 번째 단계에서는 3.2절의 크로스커팅 요소를 반영하여 XML로 명세된 크로스커팅 합성정보를 기반으로 매핑하는 단계, 세 번째 단계에서는 codeDOM을 사용하여 합성된 추상구조화 트리를 이용하여 크로스커팅 된 소스 코드를 생성단계이다.

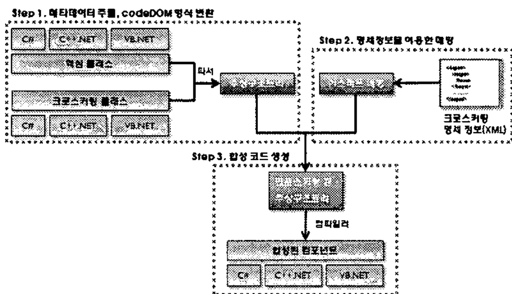

그림 3. 닷넷 프레임워크에서 메타데이터와 크로스커팅 개념을 적용한 위빙 프로세스

1. 리플렉션을 이용한 메타데이터생성

조인포인트를 식별하고, 식별된 조인포인트에서 필요한 클래스로 위빙이 가능하다. 크로스커팅 개념을 적용, 핵심 클래스와 크로스커팅 클래스를 위빙 하기 위하여 각 클래스의 구조분석을 한다. 구조분석을 통하여, 네임스페이스, (생성자/멤버/호출) 메소드, 파라미터 등의 정보(3.1절 기술)를 추출한다.

리플렉션을 적용하기 위하여 닷넷 프레임워크에서 제공하는 System.Type을 이용한다. 리플렉션 클래스는 컴포넌트의 모든 자료형 및 구조를 표현하고 리플렉션 API 사이를 상호 연결하는데 사용한다. Type 클래스의 멤버는 타입 선언에 대한 정보를 조회한다. 이 정보에는 코드의 어셈블리 정보가 포함된다.

클래스의 엘리먼트를 추출한 후, 객체 그래프 표현을 생성하기 위하여 추상구문트리(Abstract Syntax Tree)로 변환한다. 객체 그래프 포맷은 codeDOM을 사용한다. 이는 닷넷 프레임워크에서 지원하는 각 언어로 구성된 소스 코드를 독립적인 구조로 표현하기 위한 클래스와 인터페이스를 제공한다. [그림 4]는 C#언어로 작성된 계좌이체 관련 핵심 클래스이다. 「Transfer_Svc」 클래스 안에 「Transfer」 메소드가 있다. 「Transfer」 메소드 안에는 데이터베이스 접근과 관련된 생성자 메소드 「DB_Transaction *DB_cn = new DB_Transaction()」, 파라미터 접근과 관련된 「Packet_Error_Log」, 「Packet_Data_Log」 등이 있다. (그림 5)는 C++.NET으로 작성된 데이터베이스 트랜젝션 관련 어스팩트 클래스이다. 「DB_Transaction」 클래스 안에 「SQL_Fetch」, 「SQL_Commit」, 「SQL_Command」 메소드가 있다. 「SQL_Fetch」 메소드 안에는 데이터베이스 실행과 관련된 「PQexec」, 「PQresultStatus」, 「SysLog」 등의 파라미터들이 있다. 각 클래스의 오른쪽에 표현한 추상구조트리는 리플렉션을 이용하여 분석, 추출된 클래스 메타데이터를 [그림 2]의 형식을 이용하여 계층적으로 표현하였다. 핵심 클래스[그림 4]와 크로스커팅 클래스[그림 5]는 서로 다른 언어를 사용하고 있지만, codeDOM 형식의 공통적인 형식의 구조를 갖는다. 이 공통된 형식은 .NET 프레임워크의 다른 언어로 생성된 클래스 사이의

합성하기 위한 기반이 된다.

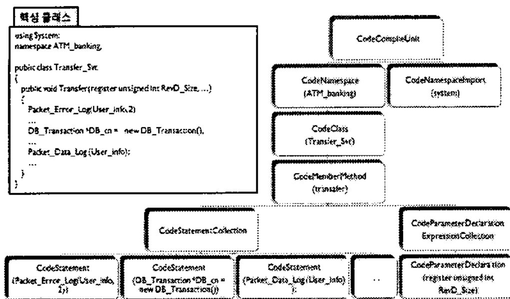

그림 4. 핵심 클래스의 코드와 추상구조트리

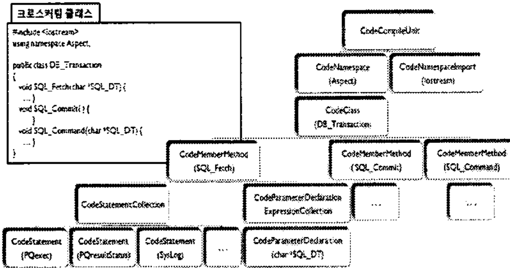

그림 5. 크로스커팅 클래스의 코드와 추상구조트리

2. 크로스커팅 합성정보를 이용한 매핑

메타데이터 기반의 크로스커팅을 지원하기 위하여 크로스커팅 정보(XML)를 통하여 매핑하는 방법을 사용한다. 핵심 클래스와 크로스커팅 클래스의 위빙을 위하여 합성정보를 명세한다. 위빙 시스템에서는 명세된 정보를 기반으로 조인포인트(3.2절 기술)를 선택한다. 명세된 매핑 정보를 사용하여, 핵심 클래스에서 식별된 조인포인트(메소드, 호출, 실행 등)에 크로스커팅 클래스를 매핑하여 위빙 한다. 합성정보는 어스팩트 위빙 타입 지정을 위한 어드바이스 선언과 선택된 조인포인트에 위빙할 것인지의 포인트컷 선언이 필요하다. 위빙 정보를 XML로 명세를 하는 이유중 다른 하나는 위빙 컴포넌트 생성시 새로운 컴파일러를 사용하지 않고, 범용적인 컴파일러를 사용하기 위함이다. 닷넷 프레임워크에서 크로스커팅 위빙을 위하여 MSIL 레벨에서 코어 클래스와 어스팩트 클래스를 위빙하고, JIT 컴파일러를 호출한다.

크로스커팅하기 위하여, 매핑 정보를 [그림 6]과 같이 명세한다. 매핑 정보는 3단계로 구성한다. 첫 번째 부분은 어스팩트 선언부분이다. 선언부분에서는 어스팩트 클래스를 식별하기 위한 명세이다. 어스팩트 명세를 선언하고, 어스팩트 클래스의 이름 「Aspect.DB_Transaction」을 지정한다. 두 번째 부분은 어드바이스 선언부분이다. 어스팩트 클래스에서 어드바이스로 사용되는 「SQL_Fetch」 메소드를 식별하고, XML 테그를 사용하여 <after> 어드바이스 타입을 선언한다. 세 번째 부분은 포인트컷 선언부분이다. 코어 클래스에서 선언된 「SQL_Fetch」 메소드를 식별한다. 어드바이스가 하나 이상일 경우, 첫번째 어드바이스는 <first> 태그(Tag)를 사용하고, 두번째 어드바이스는 <second> 태그를 사용하여 정의 하였다. 실행 포인트컷과 코어 컴포넌트의 「void」 반환 타입을 연결하고, 조인포인트가 크로스컷 되는 「Transfer_Svc」 메소드를 선언한다. 포인트컷에서 명세된 「unsigned int」 파라미터 연결을 선언한다. 어스팩트 합성정보를 이용하여, 코어 클래스와 어스팩트 클래스의 CodeCompileUnit이 참조 된다. 조인포인트를 식별하고, 어드바이스로 사용될 엘리먼트를 추출하고, 포인트컷 정보를 통하여 클래스 사이의 위빙이 이루어진다.

언어에 따라 파라미터 표현 방법이 다르기 때문에, 표현 형식이 다른 언어들 사이의 위빙에 문제가 발생한다. 지원하는 언어들 사이에서 위빙을 가능하게 하기 위하여 파라미터에 관련된 부분까지 상세하게 명세 하였다.

```
<!--어스팩트 선언-->
<name> Aspect </name>
<type> Aspect. DB_Transaction </type>

<!--어드바이스 선언-->
<advice>
  <pointcut> <first> <after>
    <behavior>
      <name> SQL_Fetch </name>
    </behavior>
  </after> </first> </pointcut>
  <pointcut> <second> <after>
    ...
</advice>

<!--포인트컷 선언-->
<named_pointcut>
  <modifier> <public/> </modifier>
    <name> Transfer
      <pointcut> <first> <execution>
        <method_signature>
          <return_type>
            <type_name> void </type_name>
          </return_type>
        <join_point_type>
          <type_name> Transfer_Svc </type_name>
        </join_point_type>
        <methodname> Transfer </methodname>
          <parameter>
            <type_name> register unsigned int </type_name>
          </parameter>
        <method_signature>
        ...
</named_pointcut>
```

그림 6. 크로스커팅 위빙을 위한 합성정보 명세

3. codeDOM을 통한 위빙 모듈 생성

제안하는 메커니즘은 메타데이타를 이용한 크로스커팅 구현을 위하여 정적/동적의 두 가지 타입을 지원한다. 존재하는 타입에서 변수와 메소드 추가를 가능하게 하는 정적 타입과 프로그램에서 정의된 포인트에서 실행을 위한 추가적인 구현을 가능하게 하는 동적 타입을 지원한다. [그림 7]은 동적타입의 크로스커팅 위빙 흐름을 시퀀스 다이어그램으로 표현하였다. 「CodeBuilder」는 어드바이스 명세정보를 통하여, 크로스커팅 대상의 메소드 이름과 파라미터를 호출하고, 생성자 메소드를 통하여 어드바이스 타입을 확인한다. 어드바이스 정보를 호출한 후, 코어 클래스에 메소드와 클래스를 추가하는 흐름으로 생성된다.

크로스커팅 개념의 목적은 포인트컷 정의뿐만 아니라 포인트컷에 대한 어드바이스를 포함하고 있는 시스템 스팩에서 공통된 부분을 분리하는 것이다. 또한, 기존 어플리케이션의 리팩토링에 의하여 존재하는 클래스(컴포넌트)를 활용, 새로운 프로젝트 개발에 적용할 수 있다. 컴파일 된 결과물을 컴포넌트(.exe, .dll)로 생성하고 활용이 가능하다.

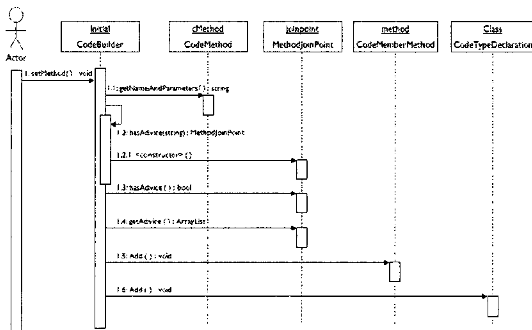

그림 7. 동적 크로스커팅을 위한 위빙 시퀀스 다이어그램

V. 객체지향 설계와 크로스커팅 설계 비교

객체지향 설계와 크로스커팅 설계를 비교하기 위하여 은행계좌 시스템을 도입하였다. 예제에서 사용된 다이어그램은 3가지의 클래스로 구성하였다. 상위클래스(Account)와 계좌 타입에 따른 두 하위클래스(SavingAccount, CheckingAccount)로 구성하였다.

1. 객체지향설계

Account 클래스는 계좌가 가지는 일반적인 속성과 오퍼레이션을 표현하였다. Account 클래스의 상속을 이용하여 적금계좌(SavingAccount)와 예금계좌(CheckingAccount)를 구현하였다.

CheckingAccount 소스코드는 예금을 위한 프로시저를 표현한다. 트랜젝션은 데이터베이스 연결과 트랜젝션을 위하여 BeginTransaction으로 시작한다. 오류 여부에 따라서 Commit 메소드와 Rollback 메소드를 통하여 실행된다. 클래스 소스코드(SavingAccount, CheckingAccount)에서는 트랜젝션 조정을 위한 어플리케이션을 위하여 각각 필요하다. 트랜젝션에 관련된 코드는 각 하위 클래스에서 중복적으로 발생하기 때문에, 최적화된 소스코드를 제공할 수 없게 된다.

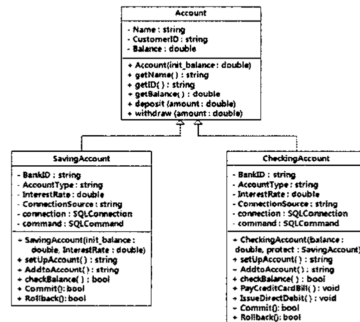

그림 8. 객체지향으로 구현된 은행계좌 클래스

```
// DB 연결, 트랜젝션 시작
SQLConnection connection = new SQLConnection(
                              Connection_Source);
connection.Open();
SQLTransaction tx = connection.BeginTransaction();
string message = "";

try {
      string CommandText = "업데이트 계좌 " +
      " 잔액 = " + amount +
      " 고객 ID = " + base.getID() +
      " 계좌 타입 = " + AccountType;
      SQLCommand command = new SQLCommand();
      command.CommandText = CommandText;
      command.Connection = connection;
      command.Transaction = tx;
      command.ExecuteNonQuery();

      tx.Commit();
      message = "업데이트 성공!";
   }  // 업데이트를 위한 테스트와 성공시 트랜젝션 명령
   catch(Exception e) {
      tx.Rollback();
      message = "업데이트 실패!";
   }  // 업데이트 실패시 트랜젝션 명령
   finally {
      connection.Close();
   }  //DB 연결 종료
```

그림 9. CheckingAccount에서 트랜젝션 관련 코드

2. 크로스커팅 설계

크로스커팅 설계는 객체지향에서 중복적으로 발생하는 트랜젝션 관련 소스코드를 각 클래스로부터 분리[그림 10]하고, AspectTransaction의 어스팩트 클래스로 분리하였다. 분리된 어스팩트 클래스에서 데이터베이스 연결과 트랜젝션 관리를 조정할 수 있다. AspectTransaction는 크로스커팅 메소드로 구성된다. beforeTransaction 메소드는 데이터베이스 연결과 트랜젝션 셋업을 조정하고, afterTransaction 메소드는 트랜젝션의 commit/rollback 조정하고 데이터베이스 연결을 종료한다. [그림 11]의 소스코드는 AspectTransaction에서 두 메소드 BeforeTransaction과 AfterTransaction을 보여준다. [그림 12]의 소스코드는 어떻게 핵심 클래스의 메소드가 어스팩트 클래스에 의하여 영향을 받는지를 보여준다. 필요한 SQL이 메소드에 포함될 수 있기 때문에 만들어진 코드는 이해하기 쉽고, 클래스로 만들기 쉬워진다. 그러나, 핵심 클래스의 메소드로 부터 연결변수를 제거하고 어스팩트 클래스를 생성해야 한다.

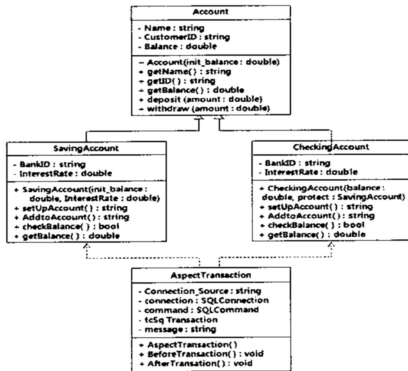

그림 10. 크로스커팅 개념을 이용 구현된 은행계좌 클래스

```
public void BeforeTransaction(){
    connection = new SQLConnection(Connection_Source);
    connection.Open();
    tx = connection.BeginTransaction();
    command.Transaction = tx;
}
public void AfterTransaction(){
    if(message.Equals("에러")) {
        tx.Rollback();
    }
    else {
    tx.Commit();
    }
    connection.Close();
}
```

그림 11. AspectTransaction 클래스에서 추출된 크로스커팅 메소드

```
Try {
    string CommandText = "업데이트 계좌 " +
        " 잔액 = " + amount +
        " 고객 ID = " + base.getID() +
        " 계좌 타입 = " + AccountType;

    command.CommandText = CommandText;
    command.Connection = connection;
    command.ExecuteNonQuery();
    message = "잔액 업데이트 성공";
}
catch(Exception e)
{
    Console.WriteLine(e.StackTrace);
    message = "에러";
}
return message;
```

그림 12. AspectTransaction 클래스를 이용한 구현

VI. 결론 및 향후연구

객체지향언어는 개발하고자 하는 시스템의 도메인 모델 설계와 구현을 쉽게 적용할 수 있는 장점에도 불구하고, 기능의 분산으로 인하여 소스코드의 혼란이 발생한다. 분산은 하나의 기능을 클래로스 캡슐화하기 불가능하거나 힘든 경우에 발생한다. 도메인의 핵심기능을 객체지향 원리에 따라 설계하여도, 여러 기능을 단일 클래스로 독립시키지 못하고 여러 클래스로 존재하게 된다. 혼란은 여러 개의 클래스에 분산되고 중복되어 있어 컴포넌트화가 불가능한 경우가 발생하게 된다. 클래스의 복잡도와 응집도에 따라 설계하였더라도 엉켜 있는 지저분한 코드가 된다.

본 논문에서는 기능의 분산과 코드의 혼란을 해결함으로써 클래스를 최적화 시키고, 리팩토링을 통하여 추출된 클래스의 재사용성을 높일 수 있는 메타데이터와 크로스커팅 개념을 적용한 프로세스를 제안하였다. 메타데이터는 클래스 엘리먼트 정보를 활용하는 방식이다. 리플렉션을 통하여 코어 컴포넌트와 어스팩트 클래스에서 메타데이터 정보를 추출하여 추상구조트리로 변환하였다. 변환된 트리를 합성하기 위하여 3장 2절에서 기술한 조인포인트를 지원한다. 또한 재사용과 유연성을 위하여 XML 명세를 지원한다.

추출된 컴포넌트의 엘리먼트를 codeDOM 형식의 추상구조트리를 사용하여 이유는 .NET 프레임워크에서 지원되는 서로 다른 언어들도 2장 3절의 특징을 이용하여 크로스 언어를 지원하는 위빙을 지원하기 위함이다. 개발자는 다양한 환경에서 위빙 메커니즘을 통하여 크로스커팅 개념을 이용 클래스, 더 나아가 컴포넌트를 이용한 개발에 적용할 수 있다. 이는 기존 어플리케이

션의 리팩토링에 의하여 추출되는 어플리케이션의 컴포넌트를 활용, 새로운 프로젝트에 쉽게 이식, 개발이 가능해 진다. 또한, 평가에서 보는 결과와 같이 크로스커팅 개념을 이용하여 중복된 코드를 최소화 할수 있고, 최적화된 시스템 구성이 가능하게 된다.

제안하는 방법은 codeDOM에 의존하기 때문에 닷넷 언어에서 codeDOM 파서가 없다면 위빙될 수 없다. 또한, codeDOM 표현의 제약도 있다. C# 언어 구조는 codeDOM과 자연스러운 매치가 되지만, 닷넷 환경의 다른 언어에서는 구조적인 매치 문제가 발생할 수도 있다. 이러한 문제를 개선할 수 있는 추가 연구가 필요하다.

참 고 문 헌

[1] R. Awais, G. Alessandro, and M. Ana, "Aspect-oriented software development beyond programming," Proceedings of the 28th international conference on Software engineering, pp.1061-1062, 2006.

[2] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, and W. G. Griswold, "Getting started with aspectJ," Commun. ACM, Vol.44, No.10, pp.59-65, 2001.

[3] H. Erik and H. Jim, "Advice weaving in AspectJ," Proceedings of the 3rd international conference on Aspect-oriented software development, pp.26-35, 2004.

[4] H. Youssef and Constantinos A. Constantinides, "The development of generic definitions of hyperslice packages in Hyper/J," ETAPS 2003 Workshop on Software Composition, pp.127-134, 2003.

[5] B. Jonas, "What are the key issues for commercial AOP use," How does AspectWerkz address them&quest, Proceedings of the 3rd international conference on Aspect-oriented software development, pp.22-24, 2004(3).

[6] J. Nico, T. Eddy, S. Frans, and J. Wouter, "Adding dynamic reconfiguration support to JBoss AOP," Proceedings of the 1st workshop on Middleware-application interaction in conjunction with Euro-Sys 2007, pp.1-8, 2007.

[7] L. John, "Cross-Language Load-Time Aspect Weaving on Microsoft's Common Language Runtime," Demonstration at the 1st International Conference on Aspect-Oriented Software Development, 2002(4).

[8] V. Safonov, "Aspect.NET- a Framework for Aspect-Oriented Programming for .Net platform and C# language," .NET Developer's Journal, pp.28-33, 2005(7).

[9] W. Schult and A. Polze, "Aspect-Oriented Programming with C# and .NET," In 5th IEEE International Symposium on Object-oriented Real-time Distributed Computing, pp.241-248, 2002.

저 자 소 개

이 승 형(Seung-Hyung Lee) 정회원

- 1999년 8월 : 용인대학교 전산통계학과(이학사)
- 2002년 2월 : 경희대학교 전자계산공학과(공학석사)
- 2004년 2월 : 경희대학교 전자계산공학과(박사과정수료)

<관심분야> : 관점지향 프로그래밍(AOP), 소프트웨어 재사용, 리팩토링

박 제 연(Je-Yeon Park) 정회원

- 2000년 2월 : 한신대학교 정보통신학과(이학사)
- 2003년 7월 : 경희대학교 전자계산공학과(공학석사)
- 2005년 7월 : 경희대학교 전자계산공학과(박사과정수료)

<관심분야> : 관점지향 프로그래밍(AOP), 역공학, 소프트웨어 재사용

송 영 재(Young-Jae Song) 정회원

- 1969년 2월 : 인하대학교 전기공학과(공학사)
- 1976년 2월 : 일본 Keio University 전산학과(공학석사)
- 1979년 2월 : 명지대학교 전산학과(공학박사)
- 1971년 ~ 1973년 : 일본 Toyo Seiko 연구원
- 1982년 ~ 1983년 : 미국 Maryland University 전산학과 연구교수
- 1989년 ~ 1990년 : 일본 Keio University 전산학과 객원교수
- 1976년 ~ 현재 : 경희대학교 전자정보대학 교수

<관심분야> : 컴포넌트 기반 소프트웨어 엔지니어링, 역공학, 소프트웨어 재사용