제18회 한국정보처리학회 추계학술발표대회 논문집 제9권 제2호 (2002.11)

Aspect-Oriented Programming을 이용한 Source Code 분석 시스템 설계

최민용*, 양해술*
*호서대학교 벤처전문대학원
e-mail: choiminyong@hotmail.com, hsyang@office.hoseo.ac.kr

A Design of Source Code Analysis System Using Aspect Oriented Programming

Min-Yong Choi*, Hae-Sool Yang*
*Graduate School of Venture, Hoseo University

요 약

소프트웨어를 설계, 구현하는데 있어서의 어려움이란 주변 환경이나 제반 기술 등과 같은 어려움도 있겠지만 실제 프로그래머가 작성하는 소스코드의 복잡성도 그 원인중의 하나라 할 수 있다. 이와 같은 문제의 해결을 위해서는 소스코드의 접근이나 이의 테스트를 위한 결합 등의 유연성이 높은 소프트웨어 기술이 필요한 실정이다. 이에 따라 AOP(Aspect-Oriented Programming) 기술을 이용하여 소스코드 자체의 접근 및 그의 처리를 용이하게 하여 복잡한 시스템의 개발이나 기존 시스템의 관리에 있어 효율을 높이고자 한다.

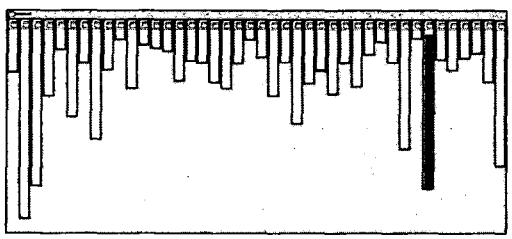
1. 서론

소프트웨어 기술의 흐름을 살펴보면 절차적인 프로그래밍 기술에서 객체 지향 기술로 그리고 더 나아가 현재 컴포넌트 기반의 소프트웨어 기술에 이르고 있다. 이는 소프트웨어의 생산적인 측면에서의 효율을 높이기 위한 개념이라 할 수 있으며, 실제 소프트웨어 개발에 있어서도 이전에 비해 그의 생산성이 많이 향상되었다. 현재 일반적으로 널리 사용되고 있는 컴포넌트 기반 소프트웨어 패러다임은 소프트웨어를 개발하는데 있어 마치 기계를 조립하듯 소프트웨어 부품들의 조립을 통하여 소프트웨어를 개발하는 것으로 이러한 부품들의 재사용으로 인하여 더 안전하고 성능 좋은 양질의 소프트웨어를 사용할 수 있게 되었다. 소프트웨어의 생산성의 향상으로 인하여 소프트웨어 기술의 발전을 가져온 것이다. 그러나 생산성 측면이 아닌 복잡성 측면에서 소프트웨어를 바라보면 기술적 패러다임의 변화를 통해 어느 정도 개선된 특징들은 있지만 아직까지는 이러한 문제 해결은 개발자들의 몫으로 돌아가고 있는 실정이다. 특히 객체지향 소프트웨어 기술의 보급 이후 프로그램 개발에 있어서의 복잡성은 더 증가하였다고 할 수 있다. 이것의 원인은 하나의 작업을 처리하는데 있어서 그것을 구현할 때 그것을 처리하는 소스코드들이 프로그램 내에 서로 다른 위치에 산재해 있기 때문이다. 객체지향 개념에서의 제대로 된 설계를 바탕으로 구현이 이루어 져야 하는데 그렇지 못한 상황에서 위와 같은 결과들이 나타나는 것이다. 그리고 또 이러한 경우 불필요하게 중복되고 명확하지 않은 소스코드들이 생기게 된다. 이러한 현 실정을 분석한 결과 소스코드 자체의 분석을 통한 소프트웨어 개발 기술의 향상이 필요하며, 또 이를 바탕으로 소프트웨어 평가의 기준을 마련하는 것이 필요하다. 이는 이미 개발된 시스템에 있어서도 소스코드 분석을 통하여 시스템의 관리 및 변경을 용이하게 할 수도 있다. 이에 따라 본 논문에서는 소스코드를 분석을 위한 시스템을 설계하고 분석의 패턴을 규정하여 소프트웨어의 질적 향상을 도모하고자 한다. 이를 위하여 새로운 프로그래밍

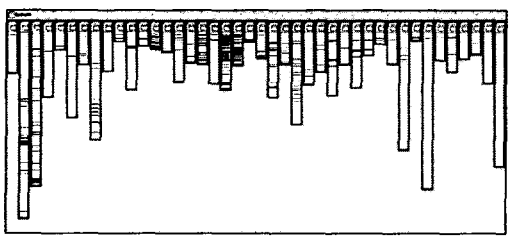
기술인 AOP(Aspect-Oriented Programming)의 개념과 기술을 응용할 것이며 실제 구현에 있어서는 AOP의 개념을 확장시킨 AspectJ를 이용할 것이다.

2. AOP(Aspect-Oriented Programming)

AOP(Aspect-oriented Programming)란 소프트웨어를 개발하는데 있어서 필요한 부분만을 임의로 잘라서 이를 바탕으로 모듈화를 통하여 시스템을 개발하는 새로운 개념의 기술이다[1]. 소프트웨어를 개발할 때 견고한 디자인을 바탕으로 개발된 소프트웨어인 경이 모듈화가 잘 되어 있다. 이처럼 잘 정의된 모듈화는 소프트웨어의 성능을 향상시키게 되고 차후 유지보수에도 많은 이점을 준다. 하지만 여러 가지 이유로 인하여 모든 소프트웨어들이 좋은 모듈화를 유지할 수는 없는 것이다. 아래의 (그림 1)과 (그림 2)는 모듈화의 예를 나타낸 것으로 (그림 1)은 톰캣 서버(Apache-Tomcat)에서 XML을 Parsing할 때의 모습이고, (그림 2)는 가 작동될 때 참조되는 소스코드를 나타낸 것이다. 그림을 비교해 보면 알 수 있듯이 (그림 2)의 경우 하나의 작업을 처리하는데 있어서 굉장히 복잡한 과정을 처리한 다는 것을 알 수 있다[5].



(그림 1) XML Parsing



(그림 2) Logging Handling

이와 같은 경우 AOP를 이용하여 (그림 2)의 톰캣서버의 Logging 처리를 (그림 1)과 같은 모듈구조로 바꿀 수 있다. 이러한 개념을 Crosscutting이라 하는데 이는 (그림 2)에서 Logging Handling에 사용되는 각 부분들을 잘라내어 하나로 모듈화 시키는 것이다. 즉 이것은 여러 모듈에서 필요한 부분만을 추출하여 새로운 모듈의 구성하는 것이다[2]. 이러한 Crosscutting의 개념은 프로그래밍을 할 때 모듈화를 쉽게 할 수 있고 전체 시스템이 자연스러운 구조를 띄는데 중요한 역할을 한다.

3. AspectJ

AspectJ는 AOP의 개념을 Java로 확장한 것으로 일반적으로 Java 언어의 기본적인 특성을 통하여 Aspect Programming을 할 수 있는 것이다. 이러한 호환은 다음과 같은 효과를 가져올 수 있다[3].

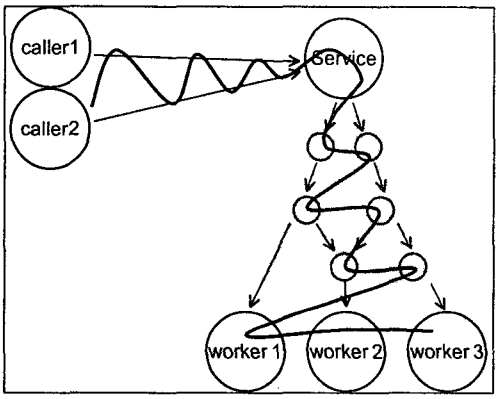
- *Upward Compatibility: 자바 언어로 작성된 모든 프로그램들은 AspectJ 프로그램과 서로 호환된다.*
- *Platform Compatibility: AspectJ 프로그램은 기본적으로 Java Virtual Machines에서 작동한다.*
- *Tool Compatibility: 현재 사용되고 있는 툴의 확장 개념으로 AspectJ를 지원하기가 용이하다.*
- *Programmer Compatibility: 프로그래머가 사용하기에 마치 자바프로그램을 작성하는 것과 같다.*

3.1. Semantics

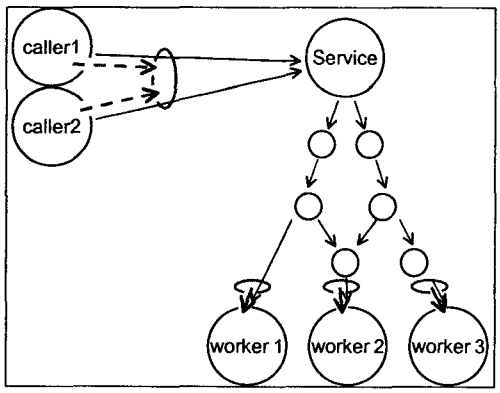
AspectJ는 Java Semantics에 Join Points라는 개념을 도입하여 Java의 확장 개념으로 사용하고 있는데, 이는 다음과 같은 4가지 요소로 구분할 수 있다.

- *Join Points: 프로그램이 실행될 위치를 정하는 것으로 이것은 메소드와 생성자 호출 등 여러 가지를 기술할 수 있다.*
- *Pointcuts: Join Points를 지정하기도 하고 Join Points의 실행코드 내의 값들을 지정하기도 한다.*
- *Advice: 각각의 Join Points 내의 Pointcuts의 실행코드를 나타낸는 것으로 Pointcuts에서 지정한 값들을 바탕으로 실행한다.*
- *Static Crosscutting: 프로그램의 구조의 형태를 변경할 수 있는 소스코드를 나타내는 말로 세부적으로 Introduction 과 Declaration으로 구분 지을 수 있다.*

그리고 위의 4가지 Pointcuts, Advice 그리고 Static Crosscutting 요소들을 전부 포함한 개념으로 Aspects 라고 한다. (그림 3)과 (그림 4)는 이와 같은 요소를 이용하였을 때의 전후 프로그램의 수행 처리 과정을 나타낸 것으로 (그림 4)의 경우가 월등히 간단해 진 것을 볼 수 있다[4].



(그림 3) 일반적인 프로그램 수행 처리 과정



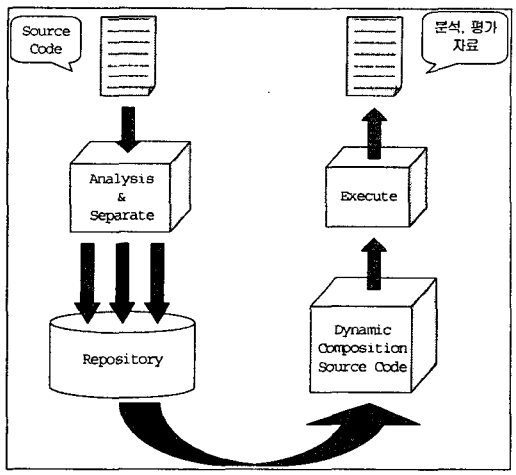
(그림 4) AspectJ기반 프로그램 수행 처리 과정

4. Source Code Analysis System Design

이 분석 시스템은 프로그램이 실행될 때 내부 흐름을 동적으로 검사하여 시스템의 내부수행에 따른 성능을 분석하고 평가하고자 함이 목적이다. 잘 짜여진 프로그램은 전체 시스템의 성능에 큰 영향을 미치는 것은 당연한 결과이므로 일차원 적으로 좋은 시스템의 개발을 위해서는 소스코드의 완성도가 높아야 한다. 특히 하나의 작업이 모듈화 되지 않았거나 중복되어있는 경우 프로그램의 성능뿐만 아니라 차후 유지보수에 있어서도 많은 어려움을 가져오게 된다. 이에 소스코드를 분석하는 도구가 필요하며 이를 바탕으로 시스템 개발단계에서나 관리 차원에서 활용 가능하며 더 나아가 시스템 변경에 있어서도 활용될 수도 있다.

4.1. Architecture

이 소스코드 분석 시스템은 프로그램의 실행 단계에서 작업이 이루어지므로 다음과 같은 흐름으로 설계를 하고자 한다. 첫째 분석하고자 하는 시스템이 작동하게 되면, 둘째 분석기에서 해당 소스코드의 분석이 이루어지며 분석된 소스코드들의 요소들을 리파지토리에 저장하게 되고, 셋째 저장된 요소들을 이용하여 성능측정을 위한 동적인 소스코드를 생성하고, 넷째 생성된 동적 소스코드를 실행하고, 다섯째 실행된 결과를 출력한다. 이와 같은 흐름은 (그림 5)와 같이 나타낼 수 있다.



(그림 5) 자료 흐름도

4.1.1 Analysis & Separate

본 시스템에서 소스코드를 분석하는 패턴은 Java 언어의 기본 문법에 따라 형태를 구분하고 각 각 구분되어진 요소들을 XML 형식의 Repository에 저장한다. 분석되는 범위는 하나의 작업이 이루어지는 전체로 하며 작업의 흐름에 따라 차례로 분석이 이루어진다. 중복 사용되는 소스코드에 대해서는 그에 따른 부가 정도를 기록해 둔다.

4.1.2 Repository

본 시스템에서는 Repository의 유형을 XML 형

식으로 하고자 한다. 일반적으로 Database를 사용하여 자료를 저장하지만 본 시스템에서는 저장된 자료 상태의 정보를 가공 추출할 수 있도록 하기 위하여 XML 형식으로 하고자 한다.

4.1.3 Dynamic Composition Source Code

Repository에 저장된 요소들을 추출하여 동적으로 Aspects Code를 생성하는 과정으로 하나의 완전한 작업이 이루어지는 전체 구조를 하나의 모듈로 구성하고자 한다. 그리고 구성되는 모듈 내에 분석과 평가에 필요한 출력 정보도 동적으로 생성된다.

4.1.4 Execute

생성된 Aspects Code를 실행하는 과정으로 실행을 통해 출력되는 정보를 바탕으로 시스템을 분석, 평가하게 된다.

5. 결론 및 향후 연구과제

본 논문의 취지는 소프트웨어 개발에 있어서 소스코드의 완성도를 높이는 과정을 통하여 전체 소프트웨어의 성능을 향상시키고자 하는 것이다. 그에 사용될 기술로는 소스코드 일부를 분할하여 새로운 모듈로 재구성이 가능한 기술인 Aspect-Oriented Programming의 확장개념인 AspectJ를 이용하여 소스코드의 분석을 통한 재구성의 방식으로 하고자 한다. 이를 위해 본 논문 2장에서는 Aspect-Oriented Programming의 개념에 대하여 알아보았고, 3장에서는 Aspect-Oriented Programming의 확장개념인 AspectJ에 대하여 알아보았고 4장에서는 실제 구현할 시스템의 전체 Architecture를 설계해 보았다. 향후 뒤따라야 할 연구과제로는 실제 동적 소스코드를 생성할 때의 생성 패턴 규칙과 관련된 연구가 필요하며 평가에 대한 규칙에 대한 규정도 필요할 것이다.

참고문헌

[1] http://www.aosd.net
[2] http://aspectj.org/doc/dist/progguide/index.html
[3] http://aspectj.org/documentation/papersAndSlides/ECOOP2001-Overview.pdf
[4] http://aspectj.org/documentation/papersAndSlides/ISPJ-2002-keynote-1.ppt
[5] http://www.parc.xerox.com/solutions/aspectj/