훅과 Decorator 패턴을 이용한 Aspect 동적 위버 아키텍처 설계

김진향° 송영재
{hesperjh°, yjsong}@khu.ac.kr

Aspect Dynamic Weaver Architecture Design using Hook and Decorator Pattern

Jinhyung Kim° Youngjae Song
Dept of Computer Engineering, The Kyunghee University of Korea

요 약

동적 Aspect 프로그래밍(Aspect-Oriented Programming : AOP)은 로깅과 같은 비 기능적인 부분을 모듈화할 수 있도록 해주는 새로운 프로그래밍 기법이다. AOP는 여러 클래스에 영향을 미치는 행위들을 재사용 가능하도록 Aspect로 캡슐화 시켜준다.

기존의 Aspect 위버(weaver)에서는 Aspect를 위빙 할 경우, 새로운 서비스나 메소드를 추가하고자 한다면 서브클래스를 별도로 상속을 해주어야하며, 이로 인해 코드의 라인수가 증가되며, 수행 시간도 느려지게 된다. 이러한 문제점을 해결하기 위하여 동적 Aspect 위버 상에 Decorator 패턴을 적용하여 별도의 서브클래스를 상속하지 않고, 필요한 메소드만을 추가하여 위빙시키고, Aspect의 join point에 훅(hook)을 첨가하여 불필요한 메소드의 실행을 제거해준다. 이로 인해 수행속도와 재사용성을 증가시킬 수 있게 된다.

1. 서 론

기존의 객체지향 프로그래밍은 로깅, 보안과 같은 비 기능적인 속성을 모듈화하는 것이 어려웠다. 이를 보완할 수 있는 방법으로 AOP(Aspect-Oriented Programming)가 대두되었으며, AOP는 여러 클래스에 영향을 미치는 행위들을 재사용 가능하도록 Aspect로 캡슐화 함으로써 재사용성을 높여준다.

AOP는 여러 모듈과 횡단하는 로깅, 동시성, 보안과 같은 횡단 업무(crosscutting concern)의 분리를 제공하는데, 이러한 업무는 모든 어플리케이션 시스템을 제공하기 위해 위빙 되어진다. 이러한 위빙(weaving)을 제공하는 엔진을 위버라고 부른다.

기존의 Aspect 위버에서는 Aspect를 위빙 시키고자 할 경우, 새로운 서비스나 메소드를 추가하고자 한다면 서브클래스를 별도로 상속을 해주어야하며, 이로 인해 코드의 라인수가 증가되며, 수행 시간도 느려지게 된다.

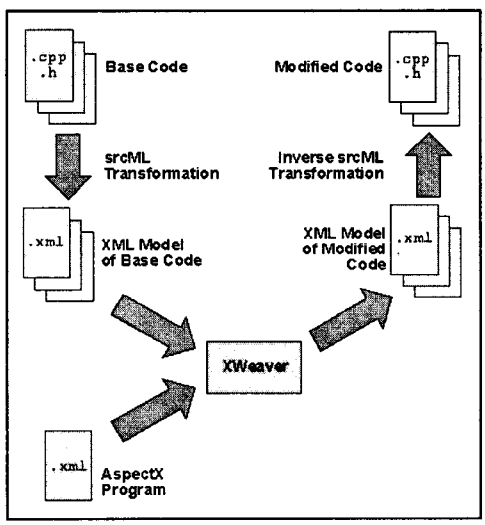
본 논문에서는 이러한 문제점을 해결하기 위하여 Decorator 디자인 패턴을 적용하여 동적 Aspect 위버의 아키텍처를 설계하였다.

위버에서 정적 Aspect를 위빙할 때, Decorator 디자인 패턴은 슈퍼클래스의 원형을 그대로 유지하면서, 슈퍼클래스와 동일한 클래스를 Dacorator로 생성한다. 다시 생성된 Dacorator 슈퍼클래스를 이용하여 새로 추가된 기능을 추가하게 되며, 별도의 서브 클래스를 만들지 않는다. 별도의 서브 클래스를 만들지 않고, Dacorator 클래스로부터 상속받는 클래스에 추가되는 기능을 속성으로 정의함으로써, 소스 코드의 라인 수를 줄일 수 있게 되며, 수행 시간을 보다 빠르게 만들어 준다[3].

그리고, 각 Aspect는 훅을 삽입함으로써, 프로그래머가 많은 메소드 중에서 가장 적합한 메소드만을 선택하여 실행할 수 있도록 도와줌으로써, 시스템의 수행 속도를 보다 빠르게 만들어줌으로써, 성능을 향상시킨다.

2. 관련연구

2.1 XWeaver



[그림 1] XWeaver의 구조

XWeaver는 C/C++을 위한 Aspect 위버의 명칭이다. Xweaver는 AspectX 언어에서 작성된 Aspect 프로그램

을 처리하기 위해 사용되는 Aspect 위버이다.

XWeaver는 특정 모듈에서 실행되고, 다른것과 독립적인 규칙의 집합에 의해 제어되는 Aspect 위빙 프로세스의 의미로 확장된다. 위버는 Aspect의 추가적인 타입을 다루기 위해 새로운 규칙으로 작성되어진 새로운 모듈을 추가함으로써 확정되어질 수 있다.

Xweaver는 기본 코드의 srcML 표현과 AspectX 프로그램에서 상속되는 XSL 프로그램의 순서로서 만들어진다. Xweaver의 출력물은 변경된 코드의 srcML 표현이다. 그림1은 Xweaver의 구조를 보여준다[1].

2.2 Aspect 위빙 전략

Aspect 위빙 전략은 영역 지향 시스템을 실행하는 모델을 제안한다. 이러한 모델은, 객체지향 언어에도 적용되고, 선택되어질 수 있다.

Aspect는 프로그램 추상화의 변경이나 새로운 클래스와 메소드의 추가에 의해 생성된다. 이러한 Aspect를 위빙하는 전략은 크게 네 가지로 나누어질 수 있다[5] .

- 상속을 통한 위빙
  : 상속 메커니즘은 Aspect에서 명세되는 행위를 실행하기 위해 사용된다.
- 연관관계를 통한 위빙
  : 연관관계 메커니즘은 Aspect에서 설명되는 행위를 실행하기 위해 사용되는 합성 메커니즘이다.
- Decorator 디자인 패턴을 통한 위빙
  : Decorator의 순서가 실시간으로 변경될 때, 패턴의 사용으로 인해 유연성이 크다는 장점이 있다.
- reflection을 통한 위빙
  : Aspect에 의해 참조되는 각 클래스는 위버에 의해 생성되고, 파싱 트리에서 Aspect와 연관된 구문을 추가한다.

2.3 Wool

Wool은 동적 AOP 기능성을 제공하는 Java 라이브러리로 실행되고, 프로그램과 함께 Aspect를 구성하기 위한 위버와 Aspect를 작성하기 위한 API와 실행되는 프로그램으로부터 위빙을 위한 요청을 받아들이는 서브시스템으로 구성된다.

Wool은 Java에서 요구한 것처럼 프로그램에 훅을 삽입한다. 훅은 모든 차단되어진 join point를 지정한 후 삽입되고, 불필요한 훅은 삽입하지 않는다.

Wool은 동일한 JVM에서 실행되는 어플리케이션에서 지역적으로 위빙되기 위한 Aspect나 Wool의 서브시스템에 자동적으로 보내어지는 Aspect를 다룬다. 아래의 코드는 Wool에 의해 Aspect가 어떻게 위빙되는지 간단히 보여주고 있다.

```
WIAspect azpect = WIAspect.forName("ProfileAspect")_;
Wool wool = Wool.connect("localhost", 5432);
wool.weave(azpect);
```

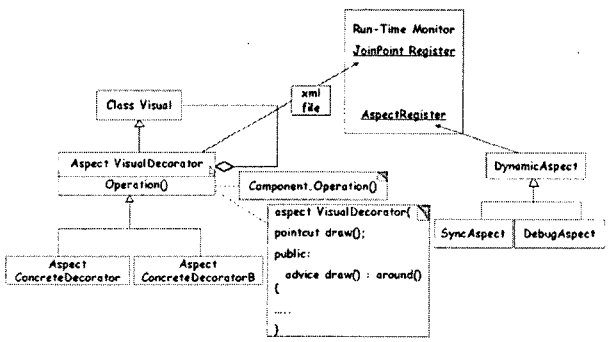
지역적으로 위빙되는 경우, Aspect 인스턴스인 "azpect"는 실행되는 프로그램에서 생성된다. 위버 인스턴스인 "wool"은 wool의 서브시스템에 연결된다.

3. Aspect 동적 위버 아키텍처 설계

그림3은 Aspect를 동적으로 위빙시키기 위한 Aspect 동적 위버 아키텍처를 보여주고 있다.

위버는 join point마다 다중 Aspect를 지원할 수 있고, 동적 Aspect를 보다 잘 지원할 수 있는 방법으로 구성된다. 동적 위버는 아래와 같이 독립적인 세 가지 주요 모듈로 구성된다[4].

- 런 타임 모니터 (Run-time monitor)
  : 동적 위버에서 중심적인 역할을 수행한다. 주요 역할은 join point와 Aspect를 기록하는 것이다. Aspect와 기능성 클래스 사이에서 상호작용한다. join point를 기억시키기 위한 run-time monitor에 대한 소스 파일은 정적 위버에 의해 생성된 XML 파일이다.
- Aspect 바인딩 (동적 Aspect)
  : 실행시 동적으로 호출되어지는 Aspect이다. 동적 위버는 성능에 의존하는 컴파일 시간에 정적으로 위빙되는 Aspect를 포함하며, 실시간으로 위빙되기를 요구하는 Aspect를 결정할 수 있다.
- 위버 바인딩 (정적 위버)
  : 정적 Aspect는 AspectJ로 명세된다. 정적 위버는 기능성 클래스를 가지는 상호작용의 결과로서 XML 파일을 생산한다. XML 파일은 기능성 클래스에서 포함하고 있는 join point의 모든 정보로 구성된다.



[그림 3] Decorator 디자인 패턴을 적용한 Aspect 동적 위버 아키텍처

정적인 Aspect가 위빙되는 경우, Decorator 디자인 패턴을 사용하여 위빙하고 있다. Decorator 디자인 패턴은 객체에 동적으로 새로운 서비스를 추가할 수 있게 하고, 기능의 추가를 위해서 서브클래스를 생성하는 것보다 융통성 있는 방법을 제공한다. 이러한 Decorator 패턴을 적용하는 방법은 실세 상속에 의해 서브클래스를 계속해서 만드는 것이 유용하지 않을 때 필요한 방법이다[3].

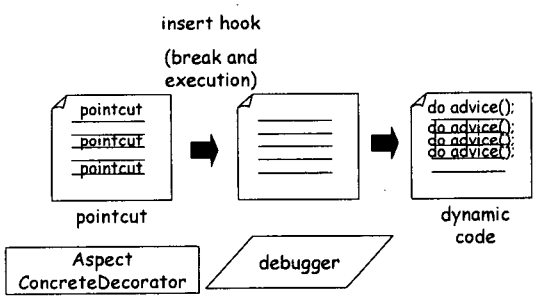
위빙 프로세스는 Visual 클래스는 Visual 클래스로부터 상속받는 서브클래스인 Aspect VisualDecorator를 생성하고, 원시 클래스의 정보는 변경되지 않고 보존된다.

Decorator 클래스는 단순히 Draw()에 대한 요청을 자신이 갖고 있는 요소에 전달하는 기능만을 가지며, Decorator 클래스의 서브클래스들은 Draw() 오퍼레이션을 확장하여 필요한 기능을 구현한다.

여러 가지 패턴 중에서 Decorator 패턴을 사용함으로써 Aspect에서 수행하고자 하는 기능을 Decorator를 이용하여 구체적으로 생성할 수 있다.

이러한 Decorator 패턴을 적용하여 생성된 Aspect 정보는 최종적으로 정적 위버의 결과물인 XML 파일을 생성한다. 각 Aspect에 대한 기능을 추가하고자 할 경우, 새로운 서브클래스를 생성하지 않고, 기존에 파생된 Dacorator 패턴에 오퍼레이션과 상태를 추가시킴으로써 코드의 라인수를 줄일 수 있으며, 위빙되는 시간을 단축시킬 수 있다.

동적 위버에서 Aspect는 실행시간을 단축시키기 위하여 훅을 적용하여 실행된다. 그림3은 훅 삽입을 정의한 시스템인 Wool을 바탕으로 Aspect ConcreteDecorator의 훅 삽입을 위하여 어떠한 과정이 필요한지 간단히 보여주고 있다.



[그림 4] 훅 삽입을 위한 두 가지 과정[2]

Wool은 적재된 모든 클래스를 조사하고, 조사된 클래스에 정지 지점(breakpoint)로서 훅을 삽입한다. 프로그래머는 클래스 내의 메소드 중에서 가장 적합한 메소드를 선택하고, 메소드를 선택한 후, 디버거를 사용하여 실행한다. 디버거는 어드바이스(advice)를 실행시키고, 실행된 어드바이스를 호출한다. 마지막으로 가장 적합한 훅만을 소스에 삽입한다. 이러한 훅을 사용함으로써, 불필요한 메소드 호출을 감소시킬 수 있으며, Aspect의 수행 시간을 단축시킬 수 있다.

4. 결론

전통적인 프로그래밍 기법은 로깅, 보안과 같은 비 기능적인 속성을 모듈화하지 못한다. 이러한 비 기능적인 요소를 모듈화하기 위해 AOP를 적용하였다.

본 논문에서는 동적 Aspect 위버의 아키텍처 모델을 설계하였다. Aspect 위버는 정적 위버와 동적 Aspect로 구성되며, 정적 위버는 여러 가지 패턴 중에서 Decorator 패턴을 이용하여 join point의 모든 정보로 구성된 XML 파일을 생성하게 된다. 이러한 Decorator 패턴을 사용함으로써, 별도의 서브클래스를 생성하지 않고 동적으로 객체에 새로운 메소드를 추가하고, 다른 객체에 영향을 주지 않음으로써, 수행시간을 보다 빠르게 할 수 있다.

그리고, Aspect의 소스 코드에 훅을 삽입하여, 프로그래머가 가장 적합한 메소드를 선택하여 실행할 수 있게 함으로써, 수행 시간을 증진시켜주며, 재사용성을 높여준다.

향후 연구과제는, 본 논문에서 Decorator 디자인 패턴을 이용하여 설계한 위버를 구현하여 Aspect 지향 프레임워크에 적용하는 것이다.

5. 참고문헌

[1] http://www.pnp-software.com/XWeaver/

[2] Yoshiki Sato, Shigeru Chiba, Michiaki Tatsubori, A selective, Just-in-time aspect weaver, In Generative Programming and Component Engineering 2003.

[3] Gamma, E., R. Helm, R. Johnson, and J. Vlissides, Design Patterns. Addison-Wesley, 1994.

[4] Wasif Gilani, Olaf Spinczyk, A Family of Aspect Dynamic Weavers, Proceedings of the 2003 Dynamic Aspect Workshop (DAW04 2003), RIACS Technical Report 04.01, March, 2004, Lancaster, UK.

[5] Eduardo Piveta, Luiz Calos Zancanella, Aspect Weaving Strategies, Journal of Universal Computer Science, 9(8):970-983, 2003.