Fractal 컴포넌트 환경에서 Decorator 패턴을 이용한 동적 위버 설계

Dynamic Weaver Design using Decorator Pattern on Fractal component Environment

박제연, 김진향, 송영재
경희대학교 컴퓨터공학과

Park Jae-Youn, Kim Jin-Hyang, Song Young-Jae
Dept of Computer Engineering,
Kyung-Hee University

요약

컴포넌트를 다양한 영역에 사용하기 위해 코드를 추가적으로 요구할 경우, 추가되는 코드가 뒤엉켜져서 컴포넌트 재사용을 방해하게 된다. AOP는 이러한 컴포넌트의 다양성 문제를 해결하기 위해 나온 방법이다. 본 논문에서는 Fractal 컴포넌트 모델에 제어 객체와 인터셉터 객체를 합성하기위해 혼합클래스(mixin class)를 사용하여 joinpoint controller interface를 생성하였으며, 이러한 인터페이스에 합성된 aspect을 적용하였다. 그리고, Aspect의 재사용성을 높이기 위하여, Decorator 디자인 패턴을 사용하여 동적으로 위빙하는 위버를 설계하였다.

Abstract

In order to use component for various domain, it need to add tangling code. Component reuse suffers from it. AOP was resolved this problem. We present in this paper a component model, called fractal, that added aspect interface using mixin class named joinController interface. Mixin class are used to combine controller object and interceptor object. we design the dynamic weaver that use the decorator pattern, for increase the reuse of aspect.

I. 서 론

컴포넌트 기반 프로그래밍은 객체지향 프로그래밍이 갖는 바이너리 표준 문제와 버전 관리, 개발 언어의 비호환성 문제를 해결한 반면에 컴포넌트를 다양한 영역에 사용하기 위해 코드를 추가적으로 요구할 경우, 추가되는 코드가 흩어지거나 뒤엉켜져서 컴포넌트 재사용을 방해하게 된다. AOP(Aspect Oriented Programming)는 이러한 컴포넌트의 다양성 문제를 해결하기 위해 나온 방법이다.

기존의 컴포넌트 모델에서 비기능적인 속성을 동적으로 추가하는 데는 많은 어려움이 있으며, 따라서 어플리케이션 개발자가 제약사항(처리시간과 메모리)간의 트레이드 오프를 고려할 수 없었다. 제어 역행화(Inversion of Control), 즉 프레임워크에서 컴포넌트를 합성할 때는 컴포넌트의 설정을 사용에서 분리하는 방법으로 위의 문제를 해결할 수 있다[1]. FCM은 제어능력을 가진 open set을 컴포넌트에 추가하여 제어 역행화를 지원한다.

본 논문에서는 첫째, FCM에 aspect 개념을 추가하기 위한 joinpoint라는 제어 인터페이스(control interface)를 정의하고, 둘째, 런타임시 비기능적인

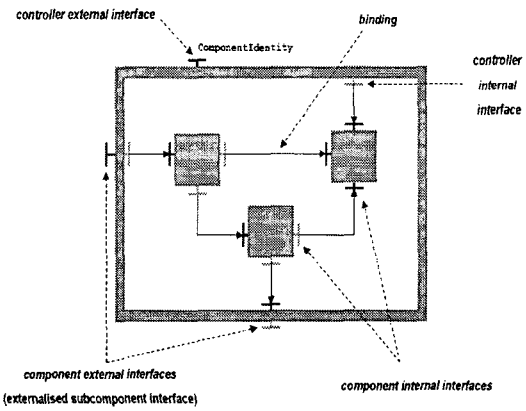
속성을 효율적으로 재구성하는 동적 위빙(dynamic weaving)을 지원하기 위해서, fractal 컴포넌트의 제어기능을 담당하는 membrane에 있는 구성요소 중 제어 객체(control object)와 인터셉터 객체(interceptor object)를 혼합클래스(mixin class)를 사용하여 간단하게 선택하고 구성할 수 있도록 하였다.

그리고, Aspect의 재사용성을 높이기 위하여, Decorator 디자인 패턴을 사용하여 동적으로 위빙하는 위버를 설계하였다[9]. Decorator 패턴은 aspect의 추가되는 기능을 동적으로 추가할 수 있으며, aspect의 불필요한 메소드 실행을 줄이기 위하여 aspect 정보에 훅(hook)[8]을 삽입하였다.

Ⅱ. 관련 연구

2.1 FCM(fractal component model)

fractal component[2]는 복잡한 소프트웨어 시스템을 관리(모니터하고 동적으로 재조정)하고 배치하고 구현하기 위한 open set을 가진 컴포넌트 모델이다.



▶▶ 그림 1. concrete fractal component model

컴포넌트의 내부특성에 대한 제어(intercession)와 내부·외부특성에 대한 인트로스펙션(introspection) 여부에 따라 타입이 분류된다.

fractal 컴포넌트 구조는 크게 두 부분으로 구성된다.

- content-실제 비지니스 로직에 해당하며, 서브 컴포넌트와 속성, 메소드들이 들어있다.
- membrane-컴포넌트 인터페이스가 들어 있으며, 오퍼레이션 호출을 중간에서 가로채어 컴포넌트content를 모니터한다.

membrane에 크게 두 가지 컴포넌트 인터페이스가 있다. server interface, client interface로 구성된 기능적 인터페이스와, 컴포넌트를 재구성하고 인트로스펙션(introspection)하기 위해 비기능적인 특성을 담당하는 제어 인터페이스가 있다.

미리 정의된 제어 인터페이스로 life cycle 제어 인터페이스, attribute 제어 인터페이스, content 제어 인터페이스, binding 제어 인터페이스 등이 있으며 이러한 제어 인터페이스는 컴포넌트의 용도에 따라 임의대로 추가할 수 있다.

2.2 CBD-AOP 통합 및 위빙에 관한 연구

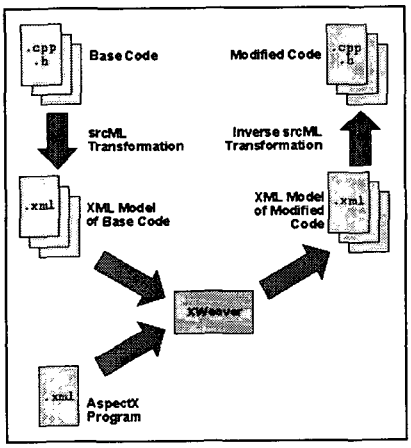
aspect and software component[3]에서는 aspect을 컴포넌트화하여 aspect에 대한 핵심 비즈니스는 advice component에 정의하고 point-cut과 위빙 규칙은 weaving component에 정의하였다. 그러나 런타임시 동적위빙을 위해 클래스를 다 중상속하므로 코드의 길이가 길어진다.

JAsCO[4]에서는 자바 빈즈 컴포넌트 모델에 AOP를 적용하였다. aspect bean에 pointcut의 종류와 처리시점(when-after, before), advice(what-확장 behavior)를 훅을 사용하여 명세하고, 커넥터에는 특정 컨텍스트 내 어느 곳에 aspect bean을 배치할 것인가를 명세하였다. 그러나, aspect bean과 커넥터를 분리하여 명세되어 있어 수정이 복잡한 문제점이 있다.

Jboss-Aop[5]에서는 jBoss(jeee based application server)를 사용하여 인터셉터를 통해 advice를 생성하고 poincut은 xml로 정의하였으나 aspect은 컴포넌트가 아닌 클래스여서 모듈화가 용이하지 않다.

Julia 프레임워크[6]에서는 FCM을 기반으로 컴포넌트 정적 구성과 동적 재구성을 지원하는 확장가능한 소프트웨어 프레임워크로, 분산시스템을 효율적으로 지원해준다. fractal 컴포넌트의 membrane을 프로그램하기 위한 프레임워크이며, 특히 제어 객체와 인터셉터 객체를 사용자가 쉽게 선택, 조립할 수 있도록 제어 객체에 대한 확장 open set을 제공한다. 반면에 모든 aspect을 재사용하지 않고 새로이 생성하므로 처리시간의 오버헤드가 발생할 수 있다.

2.3 XWeaver



▶▶ 그림 2. XWeaver의 구조

XWeaver는 C/C++을 위한 Aspect 위버의 명칭이다. Xweaver는 AspectX 언어에서 작성된 Aspect 프로그램을 처리하기 위해 사용되는 Aspect 위버이다[7].

XWeaver는 특정 모듈에서 실행되고, 다른 것과 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. 그림2는 Xweaver의 구조를 보여준다.

2.4 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[11].

- 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.

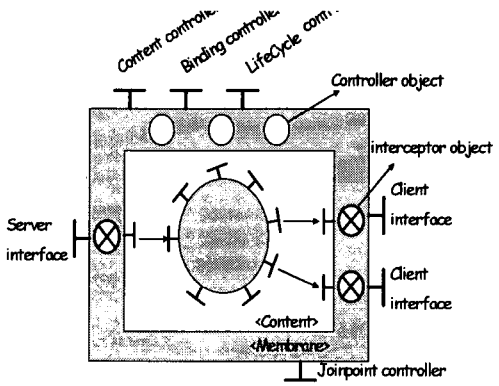
- reflection을 통한 위빙

: Aspect에 의해 참조되는 각 클래스는 위버에 의해 생성되고, 파싱 트리에서 Aspect와 연관된 구문을 추가한다.

Ⅲ. 동적 위빙 FCM 프레임워크

본 논문에서는 프레임워크에서 컴포넌트를 합성할 때 컴포넌트의 설정을 사용에서 분리하기 위해 개방형 컴포넌트 모델(open component model)인 FCM에 cross-cuting되는 aspect을 관리하기 위해 제어 능력을 가진 open set을 추가하였으며, 바이트코드 타입의 제어 객체와 인터셉터 객체를 사용자가 직접 선택하고 수정하여 컴포넌트간의 합성, 컴포넌트와

aspect간의 위빙(weaving)을 동적으로 지원하도록 설계하였다.



▶▶ 그림 3. Fractal-AOP functional 컴포넌트 모델 구조

3.1 FCM-AOP 적용

Fractal-AOP의 기능적 컴포넌트 구조와 aspect에 해당하는 advice 컴포넌트 구조는 그림 3과 같으며, 두 joinpoint controller가 위빙(weaving)시 point cut 후보중 하나가 된다.

content controller, Binding controller, LifeCycle controller, attribute controller등은 fractal component에서 미리 정의한 controller로서, attribute를 read/write, 컴포넌트의 라이프사이클 상태를 변경, 컴포넌트간의 연결/해제, 서브-컴포넌트의 추가/삭제를 결정할 수 있다.

join-point controller에는 미리 정해진 join point를 기술하고 동적 위빙을 지원하기 위한 인터페이스로 AspectDataBase에 저장될 수 있는 point cut에 대한 join point 후보들이다.

fractal-AOP 컴포넌트는 자바 객체로 표현하며, 크게 3그룹으로 나뉘어진다.

- 컴포넌트 인터페이스를 구현한 object로 [그림 2]에서 ㅜ로 표현한 부분이며, 컴포넌트 인터페이스당 하나의 object가 있으며, 각 object에 대한 레퍼런스를 할당받아 object의 모든 메소드 호출을 위임받으며 client interface에 대한 레퍼런스는 null이다.
- component membrane을 구현한 object로 [그림 2]에서 ○⊗ 로 표현한 부분으로 비기능적인 특성을 제어하기 위한 controller interface를 구현한 controller object와, server/client interface에서 호출되는 메소드를 가로채는 interceptor object로 구성된다. 각 제어 객체와 인터셉터 객체는 다른 객체들과 통신하기 위한 해당 레퍼런스를 가질 수 있다.
- 컴포넌트 content를 구현한 객체로 [그림 2]에서 C에 해당하는 부분이며 실제 비니지스 로직이 들어있는 부분이다.

Ⅳ. Aspect 동적 위버 아키텍처 설계

그림4는 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[10].

■ 런 타임 모니터 (Run-time monitor)

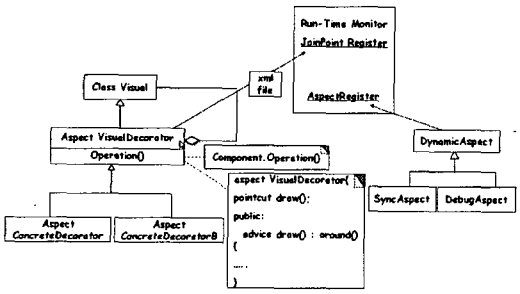
: 동적 위버에서 중심적인 역할을 수행한다. 주요 역할은 join point와 Aspect를 기록하는 것이다. Aspect와 기능성 클래스 사이에서 상호작용한다. join point를 기억시키기 위한 런 타임 모니터(run-time monitor)에 대한 소스 파일은 정적 위버에 의해 생성된 XML 파일이다.

■ 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. 정적위버는 기능성 클래스를 가지는 상호작용의 결과로서 XML 파일을 생산한다. XML 파일은 기능성 클래스에서 포함하고 있는 join point의 모든 정보로 구성된다.



▶▶ 그림 4. 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].

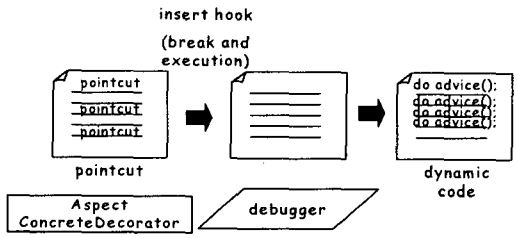
위빙 프로세스는 Visual 클래스는 Visual 클래스로부터 상속받는 서브클래스인 Aspect VisualDecorator를 생성하고, 원시 클래스의 정보는 변경되지 않고 보존된다.

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. 그림5는 훅 삽입을 정의한 시스템인 Wool을 바탕으로 Aspect ConcreteDecorator의 훅 삽입을 위하여 어떠한 과정이 필요한지 간단히 보여주고 있다.



▶▶ 그림 5. 훅 삽입을 위한 두 가지 과정[8]

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.

V. 결론 및 향후 연구방향

본 논문에서는 제안하는 프레임워크는 컴포넌트의

설정과 기능을 분리하기 위해 fractal component model에 joinpoint controller를 추가하여 functional 컴포넌트와 advice 컴포넌트를 생성하였으며, 혼합 클래스를 이용하여 각각의 제어 객체와 인터셉터 객체를 선택적으로 합성하여 동적 위빙(Dynamic weaving)이 가능하도록 설계하였다. 또한 AspectDataBase에 미리 정의해둔 pointcut에 관련된 정보를 저장함으로 재사용성을 높였다.

그리고, aspect를 동적으로 위빙하는 위버에 Decorator 디자인 패턴을 적용하여 aspect 정보에 기능을 동적으로 추가할 수 있으며, 불필요한 메소드의 호출을 줄이기 위하여 aspect 정보에 훅을 삽입하였다. 이로 인해 aspect의 코드 라인 수는 줄어들며, 실행 속도는 향상된다.

향후 연구과제로는 Aspect을 효율적으로 생성하고 관리하기 위해 본 논문에서 제시한 동적 위버를 aspect 동적 프레임워크에 적용하여 구현하고자 한다.

▌참 고 문 헌▐

[1] Michael Mattsson, Jan Bosch, Mohamed E. Fayad, October 1999 Communications of the ACM

[2] http://fractal.objectweb.org

[3] Houssam Fakih, Noury Bouraquadi, International Workshop onAspect-Oriented Software Development (WAOSD 2004)

[4] D Suv, W Vanderperren, V Jonckers. JAsCo: an aspectoriented approach tailored for component based software development. In Proceedings of the 2nd international conference on Aspect-oriented software development, pp 21-29. ACM Press, 2003.

[5] Bill Burke, Austin Chau, Marc Fleury, Adrian Brock, Andy Godwin, and Harald Gliebe. JBoss aspect oriented programming. http://www.jboss.org/

[6] E Bruneton, T Coupaye,... An Open Component Model and Its Support in Java. In Proceedings of the International Symposium on Component-based Software Engineering, Scotland, May 2004.

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

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

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

[10] 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.

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