객체 핫 스와핑을 위한 정적, 동적 프락시 설계 및 구현

윤태웅*, 최은미†, 민덕기*
*건국대학교 컴퓨터·정보통신공학과, †한동대학교 전산전자공학부
*{taewoong, dkmin}@konkuk.ac.kr, †emchoi@handong.edu

Object Hot Swapping Framework Using Static and Dynamic Proxy

Taewoong Yun*, Eunmi Choi†, Dugki Min*
*Dept. of Computer Science and Engineering, Konkuk University
†Dept. of Computer and Electrical Engineering, Handong University

요 약

서비스의 중단 없이 소프트웨어를 동적으로 업그레이드하고 확장하기 위해서는 객체지향 소프트웨어의 작은 단위인 모듈 즉, 객체를 동적으로 교체할 수 있는 구조가 필요하다. 특히 컴퓨터통신 네트워크 분야에서는 서로 영역에 넓게 퍼져 있을 경우 특별히 복잡한 구조를 가지고 된다. 본 논문에서는 이러한 객체 핫 스와핑을 가능하게 방법들을 비교분석 하고, 가장 효율적인 방법을 사용해서 객체 핫 스와핑이 가능하게 하는 프레임워크를 설계 및 구현하였다.

* 외곽의 빨간 선은 메뉴의 보기/안내선을 선택하면 보이며, 출력되지는 않습니다.

1. 서 론

소프트웨어를 개발함에 있어서 소프트웨어 개발 방법론의 각 단계에 따른 비용을 비교해 볼 때 이미 만들어진 소프트웨어의 유지 보수에 해당하는 비용이 많이 소모되고 있다. 유지 보수에 의한 소프트웨어의 재배치의 문제에 있어서 기존의 서비스를 하고 있는(작동하는) 소프트웨어를 정지 시켰다가 다시 가동하기 위해서는 또 다른 비용의 문제가 발생한다. 객체 핫 스와핑을 위한 프레임워크는 이러한 부가적인 비용의 발생을 효과적으로 없앴을 수 있고 시스템을 운영함에 있어서 시간이 중요한 부분에 자리하는 실시간 시스템의 유지 보수가 쉬워진다 객체 핫 스와핑은 객체 지향 소프트웨어를 이루는 한 부분인 하나의 객체 혹은 하나 이상의 객체를 교체 함에 있어서 그 소프트웨어가 서비스의 중단 없이 이루어짐을 의미한다. (?? 이 논문의 구성은 국내외 연구 동향을 살펴본 후 문제점 해결을 위한 해결 방안을 제시하고 성능 분석으로 하는 것으로 한다.)

2. 관련 연구

기존의 연구들은 객체 핫 스와핑을 다음의 5가지 방법으로 접근하고 있다[1].

2.1. Java JVM 변경 접근 방법

자바 객체는 JVM 내에서 힙 영역에 존재하게 되며 실행시에 간접적인 객체 참조를 통해서 사용된다. 따라서 이 객체 참조를 직접 조작하여 새로 스와핑 된 객체를 참조하게 함으로써 객체 핫 스와핑이 가능하게 된다. 단점은 여러 기업들에 의해서 각각 구현된 모든 자바 가상 머신을 고친 다는 건 쉬운 일이 아니며, 스와핑을 고려한 자바 쓰레기 수집기의 알고리즘을 변형하는 일은 또한 어려운 일이다.

2.2. 관찰자 패턴(Observer Pattern)을 통한 접근 방법

관찰자 패턴은 일대 다의 객체 관계를 표현하기 위한 패턴이다. 관찰자 패턴을 사용해서 한 주제 객체 (Subject Object) 의 내부 상태가 변경되었을 때 관찰하고 있는 모든 관찰자 객체(Observer Object)들은 자동으로 변경된 객체 생태를 전달받게 된다. 따라서 변경되려는 객체의 참조를 가지고 있는 주제 객체를 여러 관찰자 객체들이 관찰하고 있다가 객체가 교체되면 교체된 객체의 참조를 수정하는 방법으로 스와핑이 가능하다. 스와핑 하려는 객체와 그 객체를 참조하는 객체들간의 추상적 커플링 관계를 만들어 주는 장점이 있으나 참조 전달(Reference Propagation)[1] 문제와 구현상의 복잡함으로 인해서 많은 문제가 있다.

2.3 조정자 패턴(Mediator Pattern)을 통한 접근 방법
스와핑 하려는 객체와 이 객체를 사용하려는 객체들간의 참조 제어를 조정자 객체 (Mediator Object)에게 위임함으로써 참조 전달 문제를 해결하는 방법이다. 단점은 조정자 객체를 구현하고 유지보수하기 어려워지며 실행 성능이 느린 단점이 있다.

2.4 프락시 패턴(Proxy Pattern)을 통한 접근 방법
객체 참조를 제어하는 대리 객체 즉, 프락시(Proxy)객체를 두어서 이 프락시를 통해서만 접근 할 수 있게 하는 방법이다. 프락시 패턴에 의한 방법에서는 관찰자 패턴에서의 참조 전달문제는 발생하지 않는다. 또한 객체 함수 동적 호출도 가능하게 된다. 스와핑 객체를 사용하려는 객체는 이러한 내부과정을 알지 않아도 대리 객체만 가지고 처리 할 수 있다. 다른 방법에 보다 구현이나 성능 측면에 있어서 가장 현실적인 방법이다.

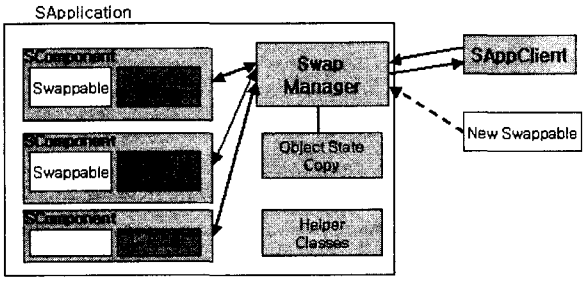

그림 1 핫 스와핑 프레임 워크

3. 핫 스와핑을 위한 프레임워크
그림 1은 본 연구에서 제시하는 핫 스와핑을 위한 프레임워크를 나타낸다. 이 구조는 [2]에서 제시한 Proxy를 이용한 핫 스와핑 구조와 유사하다. 본 연구는 이 프레임워크의 Proxy 구조를 성능과 편리성에 따라서 Static Proxy와 Dynamic Proxy로 구현하는 방법을 제시하고 있다.

SwapManager는 새로 들어올 New Swappable과 기존의 Swappable을 교체하는 일을 한다. 이때 객체의 속성 복사는 Object State Copy를 통해서 한다. 또한 SComponent의 함수호출 시에는 매개변수나 반환값을 Helper Classes를 통해서 한다. Swappable Module(SModule)은 스와핑에 의해서 교체될 객체를 의미하며, S-Module과 구별되는 ID, 버전이 있고, 서비스를 하는 함수가 존재하며, 정해진 상태(스와핑 가능상태, 블록상태 등)를 갖고, 다른 S-Module과 의존관계 리스트가 존재하며, 내부 상태 복사를 위한 매핑 원칙이 존재한다.

Dynamic SPorxy와 Static SProxy는 SModule에 대한 대리 객체로써 참조 문제, 스와핑 위한 작업 등을 처리한다. 각 각 동적, 정적인 방법으로 SModule의 함수 호출을 대리한다. 정적 혹은 동적인 프락시를 사용하더라도 같은 SModule을 그대로 수정 없이 사용 할 수 있다. 핫 스와핑을 위한 프레임 워크의 설계 쟁점는 프락시의 설계에 따른 편의성, 재상용성 및 성능에 관계에 있다.

4. 정적, 동적 프락시의 설계
본 논문에서는 객체 참조의 투명성을 위한 두 가지 방법을 제안한다. 두 가지 방법은 프락시 패턴을 이용하여 교체될 객체의 함수 호출 방법에 따라서 정적 프락시와 동적 프락시로 나눠진다. 정적 프락시는 객체 참조의 투명성 문제를 해결하기 위해서 프락시 내에 그 프락시를 참조하는 다른 객체의 참조 리스트를 유지한다. 핫 스와핑 후 이 리스트에 저장된 객체내의 프락시를 새로운 프락시로 교체함으로써 객체 참조의 투명성 문제를 해결한다. 동적 프락시는 객체 참조의 투명성 문제가 자체가 발생하지 않으므로 이 문제 해결을 위한 부가적인 비용이 필요 없다.

4.1. 정적 프락시 (Static S-Proxy)
정적 프락시는 위임(Delegation)패턴을 이용한 정적인 객체 함수 호출로 이루어짐으로 호출 속도가 빠른 장점이 있다. 또한 함수 호출을 같은 이름으로 위임(Delegation) 시켜 사용하려는 측면에서는 수정되는 부분이 많지 않기 때문에 기존 코드의 재사용이라는 장점이 있다. 하지만 정적 프락시는 SModule의 개발자가 그 객체가 제공하는 인터페이스에 맞는 Static S-Proxy를 같이 제공해야 하고 SModule의 인터페이스가 변경되면 정적 프락시도 같이 변경해야 하며 SModule을 사용하는 다른 객체는 명시적인 함수를 사용해서 (addRef(), removeRef())참조 리스트를 처리해야 하는 단점들이 있다.

4.2. 동적 프락시 (Dynamic S-Proxy)
동적 프락시의 장점은 내부에서 미리 정의된 Dynamic S-Proxy를 사용하기 때문에 정적인 프락시의 경우와 같이 SModule 개발자가 프락시를 개발하지 않아도 된다는 점이다. 그러므로 SModule의 인터페이스가 변경되더라도 프락시를 변경 할 필요가 없게 된다. 또 하나의 장점은 정적인 프락시의 경우에서와 같은 레퍼런스 처리 즉, 스와핑 가능한 객체를 사용함에 있어서 참조(Reference) 문제를 고려하지 않아도 된다. 그러므로 동적 프락시에 있는 참조 리스트가 필요 없다. 단점은 자바의 리플렉션(Reflection)을 이용한 동적인 객체 함수호출을 하기 때문에 호출 속도가 느린 단점이 있다

4.3. 프락시 클래스 상속 구조

프레임워크 내부에서 정적 프락시와 동적 프락시를 동일시 취급하기 위해서는 각각 동일한 프락시 인터페이스(SProxy interface)로부터 유도되어야 한다 SModule 개발자는 그림 2와 같이 UserDefinedSProxy를 개발해야 한다. 이러한 프락시 클래스 상속 구조는 클라이언트에서 각각 즉, 동적 프락시 ,정적 프락시 에 관계없이 사용이 가능하게 한다. 또한 시스템 내부에서도 프락시와 스와핑 가능한 객체간의 의존성이 줄어든다 즉, 같은 스와핑 객체를 동적, 정적 프락시에서 수정 없이 연결이 가능하게 되는 것이다.

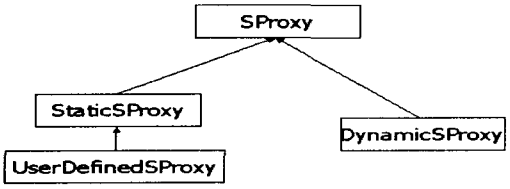

그림 2 동적, 정적 프락시의 상속 관계

4.4. SProxy Interface

SProxy 인터페이스는 스와핑 가능한 상태인지 알아내는 isSwappableSate(), SModule을 설정하거나 반환하는 함수, 매핑룰을 설정하거나 반환하는 함수, 그리고 스와핑 트랜재션을 실행하는 동안 새로운 함수 호출을 기다리거나 해제하는 함수가 있다. 동적프락시의 경우는 위의 함수 외에 서비스 함수를 대리 호출하는 invoke() 함수를 부가적으로 구현해야 하며, 정적 프락시의 경우는 SModule을 사용하려는 SAppClinet의 레퍼런스를 유지해야 하기 때문에 레퍼런스 유지를 위한 dependencyList(), addRef(),removeRef() 함수의 부가적인 구현이 필요하다.

```
public interface SProxy {
        public boolean      isSwappableState();
        public void         setSwappable(Swappable sw);
        public Swappable    getSwappable();
        public void         setMappingRule(MappingRule mRule);
        public MappingRule  getMappingRule();
        public void         blockNewMethodCall();
        public void         unBlockNewMethodCall();
```

그림 3 SProxy 인터페이스

5. 테스트 및 결과

성능 분석은 테스트에 의한 방법을 통해서 스와핑 가능한 객체의 함수 호출 시간 테스트과 스와핑 시간 테스트으로 나누어서 실험했다. 테스트 환경은 팬티엄3-1G, 256M, Windows 2000 Server, JDK 1.3.2이다.

5.1. 함수 호출 시간 테스트

정수 2개를 입력받아서 그 합을 반환하는 객체의 함수 호출 시간은 객체 함수 직접 호출(스와핑 구조를 사용하지 않은 일반적 객체 함수 로출)은 평균 0.000024(ms), 정적 프락시는 평균 0.000049(ms), 동적 프락시는 평균 0.018655(ms)의 시간이 걸렸다. 정적 프락시의 경우 함수 호출을 프락시에서 스와핑 객체로 다시 호출하는 오버헤드정도가 소요됨을 알 수 있다.

5.2. 스와핑 시간 테스트

문자열 속성을 2개 갖는 객체 스와핑의 경우 평균 0.3497(ms), 50개 갖는 경우 1.2199(ms), 100개 갖는 경우 2.7067(ms)가 걸렸다. 스와핑 시간은 객체 속성을 매핑하는 시간에 크게 의존됨을 알 수 있다.

6. 결론

본 논문은 Proxy를 사용한 객체 핫 스와핑 프레임워크를 제시하였다. 프락시 설계에 있어서 빠른 실행시간 지원하는 정적 프락시와 사용상의 편의성 및 재사용성의 잇점이 있는 동적 프락시의 구조를 모두 제시하였다. 개발시 성능 요구 조건에 따라 적당한 방법을 선택할 수 있도록 각 프락시 방식의 함수 호출 시간과 스와핑 시간을 실험 성능을 비교하였다. 정적 프락시의 단점인 개발의 어려움은 프락시 자바 코드를 자동으로 생성해주는 코드 생성기의 개발를 개발한다면 많은 부분은 자동화 할 수 있을 것이다.

7. 참고 문헌

[1] Ning EFNG, S-Module Design for Software Hot-Swapping. M.Eng.,1999.

[2] Feng, N., Gang, A., White, T., and Pagurek, B. , Dynamic Evolution of Network Management Software by Software Hot-Swapping. In Proc. of the Seventh IFIP/IEEE International Symposium on Integrated Network Management (IM 2001), Seattle, May 14-18, , pp. 63-76. 2001

[3] Ao, G., Software Hot-swapping Techniques for Upgrading Mission Critical Applications on the Fly. M.Eng., May 2000.

[4] Alex Blewitt, "Use Dynamic messaging in java" available at URL
:http://www.javaworld.com/javaworld/javatips/jw-javatip71.html

[5] Jeremy Blosser, "Explore the Dynamic Proxy API " available at URL
:http://www.javaworld.com/javaworld/jw-11-2000/jw-1110-proxy_p.html

[6] Chuck McManis, "Take an in-depth look at the Java Reflection API " available at URL :
http://www.javaworld.com/javaworld/jw-09-1997/jw-09-indepth.html

[7] Mark Grand, "Patterns in Java Volume 1 : A Catalog of Reuseable Design Patterns Illustrated with UML", wiley computer publishing, 1998