동적 형상조정 프레임워크의 모델링 및 구현

윤태웅* · 민덕기*

Modeling and Implementation of A Dynamic Reconfiguration Framework

Taewoong Yun · Dugki Min

Abstract

It requires a great deal of efforts to maintain a distributed system. What is important here is that we should be able to upgrade/maintain a distributed system without stopping the system in operation. For this, what we need is a dynamic reconfigurable framework for highly available distributed systems.

In this paper, we propose objects-hot-swapping methods as a solution to our problem. These methods permit us to dynamically upgrade/expand a system without stopping the system in operation. In addition, we analyze these methods and show that an approach based on proxy is the most efficient. Furthermore, we propose two proxy-based approaches: the first one based on the static proxy provides for a fast execution time but it is difficult to implement. The second one based on the dynamic proxy provides for a slow execution time but it is easier to implement. Finally, we propose a hot swapping framework for these static and dynamic proxy.

Key Words: Dynamic Reconfiguration, Object Hot Swapping

1. 서론

요즘 소프트웨어의 시대상황을 고려해 볼 때 대용량, 고성능의 서버들로 이루어진 슈퍼컴퓨터 프레임장비 보다는 개인 사용자들이 사용하는 성능의 값싼 PC들로 이루어진 클러스터 장비가 비용 면이나 성능적인 측면에서 더 선호 되고 있다. 뿐만 아니라 인터넷을 기반으로 한 각종 서비스들은 24시간 서비스를 계속 제공해야 하고 인터넷의 특성상 서비스의 가용성이 중요해지고 있다. 또한 클러스터 시스템을 포함하는 많은 분산 시스템도 서비스의 가용성과 시스템의 업그레이드가 중요시되고 많은 비용을 투자하고 있다.

이렇게 현재 존재하는 시스템의 일부 혹은 몇 부분의 모듈을 서비스의 중단 없이 새로운 시스템으로 변화하는 것을 Dynamic Reconfiguration 이라고 한다. 이러한 Dynamic Reconfiguration을 위해서는 특별히 고안된 프레임워크가 필요하며 프레임워크가 높은 가용성과 시스템 유지보수의 편리성을 보장해 준다. 인터넷을 기반으로 여러 가지 서비스 기반 시스템과 분산 시스템이 많이 쓰이고 있는 요즘 시대 상황을 고려해 볼 때 Dynamic Reconfiguration을 제공하는 프

* 건국대학교 컴퓨터 · 정보통신공학과

레임워크의 개발이 필요해지고 있다.

본 논문의 구성은 2장에서는 기존의 연구되었던 핫 스와핑 프레임워크에 대한 설계상의 이슈에 대해서 3장에서는 본 연구에서 제시하는 프레임워크를 4장에서는 본 프레임워크에서 사용하는 정적 프락시와 동적 프락시의 장단점을 분석하고 마지막으로 정적 프락시와 동적 프락시의 테스트를 통해서 적절한 정적, 동적 프락시 선택의 결론과 향후 연구 방향을 제시하는 것으로 끝을 맺는다.

2. 관련 연구

기존의 연구들은 객체 핫 스와핑을 다음의 4가지 방법으로 접근하고 있다[1][2][3].

2.1. Java JVM 변경 접근 방법

자바 객체는 JVM 내에서 힙 영역에 존재하게 되며 실행 시에 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 고치는 것이 쉽지 않으며 스와핑을 고려한 자바 쓰레기 수집기의 알고리즘을 변형하는 일은 어려운 일이다.

2.2. 관찰자 패턴을 통한 접근 방법

관찰자 패턴[7]을 사용해서 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 패턴을 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 패턴을 통한 접근 방법

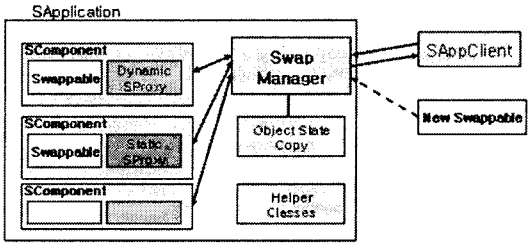
객체 참조를 제어하는 대리 객체 즉, 프락시(Proxy)객체를 두어서 이 프락시를 통해서만 접근 할 수 있게 하는 방법이다. 프락시 패턴에 의한 방법에서는 관찰자 패턴에서의 참조 전달문제는 발생하지 않는다. 또한 객체 함수 동적 호출도 가능하게 된다. 다른 방법에 보다 구현이나 성능 측면에 있어서 가장 현실적인 방법이다.

3. 핫 스와핑을 위한 프레임워크

그림 1은 본 연구에서 제시하는 핫 스와핑을 위한 프레임워크를 나타낸다. 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 Componnt의 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 리스

트가 존재하며, 내부 상태 복사를 위한 매핑 원칙이 존재한다.



〈그림 1〉 핫 스와핑 프레임워크

본 연구에서 핫 스와핑을 위한 프레임 워크의 설계 쟁점는 프락시의 설계에 따른 편의성, 재상용성 및 성능에 관계에 있다. 다음장에서는 프락시의 설계 및 모델링에 대해서 다룬다.

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.

4.1. 정적 프락시 (Static S-Proxy)

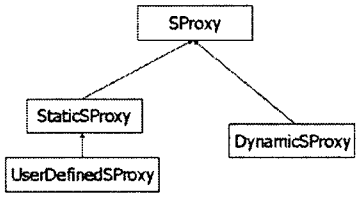
정적 프락시는 내부에서 위임(Delegation)패턴을 이용한 정적인 객체 함수 호출로 이루어짐으로 호출 속도가 빠른 장점이 있다. 또한 함수 호출을 같은 이름으로 위임시켜 사용하려는 측면에서는 수정되는 부분이 많지 않기 때문에 기존 코드의 재사용이라는 장점이 있다. 하지만 정적 프락시는 SModule의 개발자가 그 객체가 제공하는 인터페이스에 맞는 Static S-Proxy를 같이 제공해야 하고 SModule의 인터페이스가 변경되면 정적 프락시도 같이 변경해야 하며 SModule을 사용하는 다른 객체는 명시적인 함수를 사용해서 (addRef(),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

4.3. 프락시 클래스 상속 구조

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

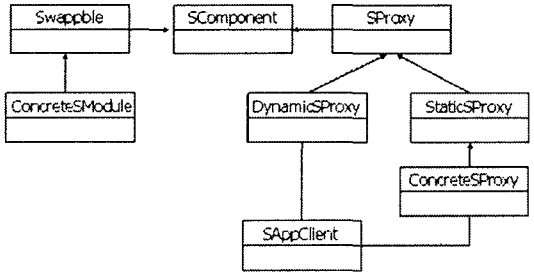


〈그림 2〉 동적, 정적 프락시의 상속 구조

4.4. SComponent 클래스

그림 3과 같이 하나의 SComponent는 하나의

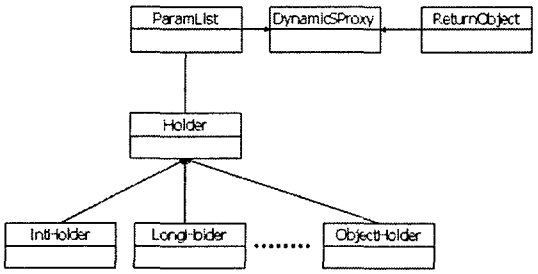
Swappable인터페이스를 구현한 스와핑 가능한 모듈과 하나의 SProxy인터페이스를 구현한 프락시로 구성된다. SComponent가 사용할 수 있는 프락시에는 프레임 워크 내에서 제공하는 동적 프락시인 DynamicSProxy가 있고 정적 프락시인 StaticSProxy가 있다.



〈그림 3〉 SComponent 관련 클래스

4.5. DynamicSProxy 클래스

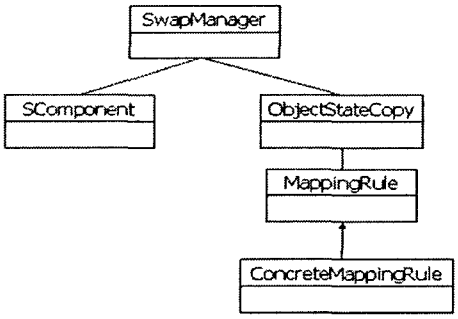
DynamicSProxy는 그림 4와 같이 객체 핫 스와핑 프레임워크 내부에 정의 되어 있다. 동적 프락시에 정의된 함수를 모두 구현되어 있으며 추가로 동적인 함수 호출을 Swappable에게 전달해 주는 Invoke함수가 리플랙션을 사용해서 구현되어 있다. 파라미터 전달을 위한 ParamList 클래스가 사용되고 ParamList를 만들기 위해서 자료형을 대표하는 여러 타입의 Holder클래스가 사용된다.



〈그림 4〉 DynamicSProxy 관련 클래스

4.6. SwapManager 클래스

SwapManager는 핫 스와핑 과정의 담당하며 이 클래스를 통해서 프레임워크 내에 있는 프락시를 찾거나 스와핑 모듈의 서비스를 호출하기 위한 작업을 처리한다. Proxy와 Swappable을 등록하는 함수와 제거하는 함수, 스와핑 모듈의 식별자를 가지고 프락시를 찾는 함수, 스와핑하는 함수로 구성되어 있다.

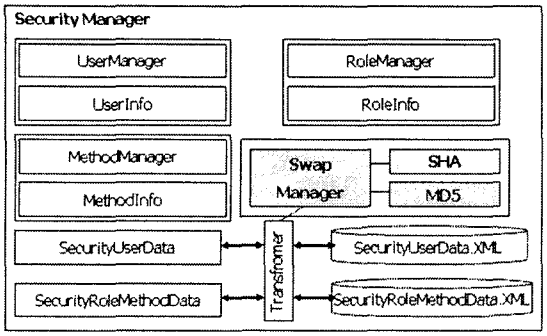


〈그림 5〉 SwapManager관련 클래스

그림 5는 SwapManager클래스와 관련된 다른 클래스들 간이 개념도 있다. SwapManager는 여러개의 SComponent를 관리하며 스와핑 과정에서 스와핑 모듈의 내부 상태 복사를 Object-StateCopy를 통해서 이루어지진다. 내부 상태 복사 매핑룰은 MappingRule인터페이스를 상속받아서 매핑 룰을 정의하게 된다.

5. 테스트 및 결과

본 논문에서는 구현된 핫 스와핑 프레임워크를 테스트하기 위해서 분산 시스템에서 사용되는 그림 6과 같은 보완관리자의 메시지 암호화 알고리즘 객체를 핫 스와핑 하는 경우로 테스트하였다. 분산 시스템의 관리영역 중 관리 콘솔 및 사용자들의 인증 및 보완 관리를 위해서 사용자들의 역할과 사용자 정보관리 그리고 그 사용자의 권한을 제어하기 위해서 보안관리자를 사



〈그림 6〉 보안관리자

용하게 된다. 팬티엄3-1G, 256M, Windows 2000 Server, JDK 1.3.2에서 테스트되었다.

함수 호출 시간 테스트는 암호화 객체를 핫 스와핑 프레임 워크를 사용하지 않고 바로 호출하는 경우는 평균 0.0739(ms), 정적 프락시는 평균 0.07692(ms), 동적 프락시는 평균 0.10546(ms)의 시간이 걸렸다. 또한 스와핑 시간 테스트에서는 보완관리자의 암호화 객체인 SHA를 MD5로 교체 하는 경우 핫 스와핑 시간은 평균 0.23263(ms)가 걸렸다.

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 요구조건에 따라 적당한 방법을 선택할 수 있도록 각 프락시 방식의 함수 호출 시간과 스와핑 시간을 테스트 하였다.

참고문헌

[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