동적 형상조정 프레임워크의 모델링 및 구현*

윤태웅**, 민덕기***

Modeling and Implementation of A Dynamic Reconfiguration Framework

Taewoong Yun, Dugki Min

Abstract

For 24 hours-7 days service on distributed systems, a great deal of efforts are investigated on high availability for seamless operation. In this paper, we propose a dynamic reconfiguration framework of distributed systems, called "hot-swapping" framework. This framework allows us to upgrade and exchange a number of modules of a distributed system without stopping running service as well as the system itself. In order to hide the state of service operation, the framework employes the "proxy" design pattern. Our framework provides two types of proxies: a static proxy and a dynamic proxy. Static proxies can achieve fast execution time, but they need to be changed whenever any minor change exists in the related swappable module. Meanwhile, dynamic proxies takes longer execution time, but do not need to be changed under minor changes of swappable modules. We compare performances of static and dynamic proxies and also apply the framework to a real situation with security management modules.

Key Words: Dynamic Reconfiguration Framework, Hot Swapping, Distributed Systems,

* 본 논문은 한국 시뮬레이션 학회 2003년 춘계학술대회에서 발표한 내용을 수정, 보완한 것임.

** 건국대학교 컴퓨터.정보통신공학과 박사과정

*** 건국대학교 컴퓨터.정보통신공학과 교수

1. 서론

요즘 소프트웨어의 시대상황을 고려해 볼 때 대용량, 고성능의 서버들로 이루어진 슈퍼 컴퓨터 프레임장비 보다는 저렴한 가격의 PC 들로 이루어진 클러스터 장비가 비용 면이나 성능적인 측면에서 더 선호되고 있다. 또한 인터넷을 기반으로 한 각종 서비스들은 24시간 일주일 동안 쉬지 않고 서비스를 제공해야 한다. 기존의 서비스들도 인터넷으로 통합이 되면서 불특정 다수가 불특정 시간에 접근하기 때문에 인터넷의 특성인 서비스의 가용성이 중요해 지고 있다.

서비스의 가용성 측면뿐만 아니라 유지보수 도 더욱 중요시되고 있다. 하나의 시스템이 개발되고 유지되면서 발생하는 여러 문제를 해결하기 위해서 새로운 시스템의 개발보다는 기존 시스템의 일부를 변경 혹은 새로운 모듈을 결합해서 새로운 버전의 시스템으로 업그레이드 하는 것이 효과적이다. 소프트웨어의 재배치 또는 업그레이드 문제에 있어서 기존의 서비스를 하고 있는 작동하는 소프트웨어를 정지시키지 않고 서비스의 가용성을 유지하면서 처리하는 것이 필요하다.

현재 존재하는 시스템의 일부 혹은 몇 부분의 모듈을 서비스의 중단 없이 새로운 시스템으로 변경하는 것을 동적 형상조정(Dynamic Reconfiguration)이라고 한다. 이러한 동적형상조정을 위해서는 시스템 아키텍처가 동적 변경을 필요로 하는 부분에 동적 형상조정 프레임워크을 포함하여야 한다. 또한 변경하려고 하는 모듈들은 그 프레임워크에 적당한 인터페이스를 구현하고 있어야 한다. 동적 형상조정 프레임워크은 특별히 디자인된 높은 가용성과 시스템 유지 보수의 편리성을 보장해주는 장점을 제공한다. 인터넷을 기반한 다양한 기반 서비스와 웹 서비스가 확산되고 있는 시대 상황에서 효율적인 동적 형상조정 프레임워크의 개발이 절실히 필요하다.

동적형상조정 프레임워크에 대한 기존연구는 크게 두 개의 관점으로 나누어진다. 하나는 불완전 동적형상조정 (Semi-Dynamic Reconfiguration) 프레임워크[14] 이다. 불완전 동적 형상조정은 서비스를 제공하는 시스템을 재 가동하지 않고 동적으로 변경하지만 변경시키려는 서비스는 멈추고 관련된 모듈은 변경하는 방식이다. 다른 하나는 완전 동적형상조정 프레임워크[1]이다. 완전 동적형상 조정 프레임워크은 핫 스와핑(Hot-Swapping) 프레임워크이라고도 불리 우는데 이 방법은 시스템 뿐 아니라 제공하는 서비스도 정지하지 않고 서비스를 제공하던 모듈을 변경시키는 프레임워크이다.

본 논문은 완전 동적형상조정 프레임워크인 객체 핫 스와핑 프레임워크를 제시하고 그에 대한 성능을 분석하였다. 본 프레임워크은 프락시 패턴을 사용하여 외부에 대하여 서비스 동작상태에 대한 투명성(Transparancy)를 제공한다. 프락시 객체의 구현은 서비스 호출 시간 단축과 개발의 편리성이라는 상호 배타적인 관점에서 프락시를 정적 프락시와 동적 프락시를 모두 제공하는 형태로 하였다. 정적 프락시는 서비스 호출 시간은 빠르나 스와핑 하려는 모듈에 따라서 정적 프락시를 일일이 개발해야 한다. 따라서 스와핑 하려는 모듈의 서비스가 변경되거나 추가되었을 때 프락시 또한 변경해야 한다는 단점이 있다. 동적 프락시는 정적 프락시와는 반대로 스와핑 하려는 모듈과 관계없이 프레임 워크에 내장된 동적 프락시를 그대로 사용한다. 그러나 핫 스와핑 모듈의 서비스의 호출을 위임하기 때문에 서비스 호출 시간이 느리다는 단점이 있다. 본 논문에서 제시한 객체 핫 스와핑 프레임워크의 한 활용 예로서 보완 관리자의 보완 알고리즘의 동적 변경에 적용해 본다.

본 논문의 구성은 다음과 같다. 2절에서 객

체 핫 스와핑을 위한 기존의 접근 방법들의 장담점을 제시한 후 선택된 프락시 패턴의 접근 방법의 장점을 분석한다. 3절에서는 기존의 연구되었던 핫 스와핑 프레임워크에 대한 설계상의 이슈를 알아본다. 본 연구에서 제시하는 정적 프락시와 동적 프락시를 지원하는 객체 핫 스워핑 프레임워크은 4절에 제시하고 장단점을 분석하고 마지막 절에서는 정적 프락시와 동적 프락시의 성능 분석을 통해서 상황에 맞는 프락시의 선택의 조건으로 알아보고 적절한 정적, 동적 프락시 선택 방법과 결론으로 끝을 맺는다.

2. 관련 연구

이번 절에서는 기존의 연구되었던 소프트웨어 핫 스와핑을 위한 스와핑 가능한 모듈의 설계에 대한 관련연구를 분석한다.

2.1 소프트웨어 핫 스와핑의 두 가지 타입

소프트웨어 핫 스와핑[5,6]에는 프로그램 내에 있는 하나의 모듈을 스와핑하는 모듈 레벨의 소프트웨어 핫 스와핑이고 다른 하나는 하나 혹은 여러 개의 스와핑 가능한 모듈을 포함한 프로그램 레벨의 소프트웨어 핫 스와핑이 있다. 기본적으로 핫 스와핑 가능한 모듈의 디자인을 더 확장한 것이 프로그램 레벨의 핫 스와핑이다. 본 논문에서 제시하는 핫 스와핑 프레임워크은 모듈에 대한 핫 스와핑이다.

2.2 핫 스와핑을 위한 접근 방법

본 절에서는 기존의 핫 스와핑을 위한 접근 방법[1,2,3]에 대해서 설명한다.

2.2.1 Java JVM 변경 접근 방법

자바 객체는 JVM 내에서 힙 영역에 존재하게 되며 실행 시에 간접적인 객체 참조를 통해서 사용된다[7]. 간접적인 객체 참조는 쓰레기 수집기를 프로그램으로부터 독립적으로 작동되게 할 수 있으며 하나의 객체를 실행시간에 스와핑 할 수 있는 기회를 제공한다. 따라서 간접적으로 사용되는 이 객체 참조를 직접 조작하여 새로 스와핑 된 객체를 참조하게 함으로써 객체 핫 스와핑이 가능하게 된다. 즉 참조의 실제 위치(메모리번지)를 변경하는 방법이다.

이 방법은 서로 다른 벤더들이 구현하여 놓은 JVM을 변경하여야 하며 특히 자바 쓰레기수집기의 알고리즘을 변경해야 한다. 따라서 구현이 힘든 방법이며 구현한다 해도 JVM의 장점인 플랫폼에 독립적이라는 장점을 일어버리게 된다.

2.2.2 관찰자 패턴(Observer Pattern)을 통한 접근 방법

관찰자 패턴은 객체들 간의 관계 중 일대다(1:n)의 객체 관계를 표현하기 위한 패턴이다[11].관찰자 패턴은 하나의 주제 객체(Subject Object)의 내부 상태가 변경되었을 때 관찰하고 있는 모든 관찰자 객체(Observer Object)들은 자동으로 변경된 객체 생태를 전달받기 위한 공지(Notify)를 받게 된다. 따라서 변경되려는 객체의 참조를 가지고 있는 주제 객체를 여러 관찰자 객체들이 관찰하고 있다가 객체가 교체되면 교체된 객체의 참조를 공지(Notify)를 통해서 수정하는 방법으로 스와핑이 가능하다.

관찰자 패턴을 사용해서 스와핑을 하는 접근 방법은 스와핑 하려는 객체와 그 객체를 참조하는 객체들 간의 추상적 커플링 관계를 만들어주는 장점이 있으나 참조 전달(Reference Propagation)[1] 문제가 있으며 구현이 복잡하다. 또한 관찰자 패턴만으로는 모든 참조 전달을 막을 수는 없다는 문제가 있다.

2.2.3 조정자 패턴(Mediator Pattern)을 통한 접근 방법

조정자 패턴은 객체 참조를 제어하는 대리 객체 즉, 조정자 객체 (Mediator Object)에게 위임함으로써 참조 전달 문제를 해결하는 방법이다.

조정자 패턴을 통합 접근 방법은 스와핑을 하려는 객체의 수가 많고 연관되는 객체들이 많을 때 그들 간의 복잡한 관계를 다루기 위한 정보를 제어하기 어려우며 따라서 조정자 객체를 구현하기도 힘들고 한번 작성된 복잡한 구조의 조정자를 유지 보수하기가 어렵다는 단점이 있다. 또한 자바의 리플렉션을 사용하기 때문에 성능이 중요시되는 어플리케이션에는 적합하지 않다.

2.2.4 프락시 패턴(Proxy Pattern)을 통한 접근 방법

프락시 패턴[11]은 객체 참조를 제어하는 대리 객체 즉, 프락시(Proxy)객체를 두어서 이 프락시를 통해서만 접근 할 수 있게 하는 방법이다.

프락시 패턴 접근 방법을 사용할 경우 관찰자 패턴과는 다르게 참조는 프락시 객체가 관리하게 됨에 따라서 참조 전달문제는 발생하지 않는다. 또한 객체 함수 동적 호출도 가능하게 된다. 동적 호출을 통해서 실시간에 어떠한 함수라도 처리 가능한 구조를 가지게 된다. 또 다른 장점은 스와핑 객체를 사용하려는 클라이언트 객체는 이러한 내부과정을 알지 않고 대리 객체만 가지고 처리 가능하게 된다. 즉, 스와핑에 따른 내부 구조를 모르더라도 일반 함수 호출처럼 스와핑 가능한 객체의 함수를 호출할 수 있다가 장점이 있다.

만약 시스템에 스와핑 하려는 객체가 많을 경우 스와핑 하려는 객체의 수만큼 프락시의 객체를 생성해야 하고 단점이 있다. 그러나 시스템을 동적으로 형상조정을 할 수 있다는 유연성을 생각해 볼 때 프락시 객체 생성에 따른 단점은 다소 감소 될 수 있다. 다른 접근 방법에 비해 구현이나 성능 측면에 있어서 가장 현실적인 방법이다. 본 논문에서 제시하는 프레임워크도 이 프락시 객체 방법을 사용하였다.

2.3 기존 연구 분석

기존의 연구에서는 핫 스와핑 가능한 모듈의 모든 함수 호출을 정적 호출이 아닌 동적 호출로 변경함에 따른 성능 저하의 문제를 해결하지 못했다. 또한 실제 핫 스와핑 가능한 모듈의 내부 상태에 따른 핫 스와핑 과정의 성능 분석 결과가 부족하다. 본 연구는 이 프락시 방법을 실시간 시스템 혹은 분산 시스템과 같은 다양한 요구사항을 갖는 현실 시스템들에 적용하기 위한 유연한 구조의 프레임워크을 제시한다. 또한 각 프락시 사용방법에 대한 성능 테스트 결과로부터 각 프락시의 활용 방법에 대한 제안을 한다.

3. 핫 스와핑 프레임웍 설계 이슈

본 절에서는 객체 핫 스와핑 프레임워크의 설계 시 고려하여야 할 이슈에 대하여 알아본다. 여기서 말하는 객체 핫 스와핑은 앞에서도 설명하였듯이 시스템을 멈추지 않고 하는 동적 형상조정 중에서도 서비스를 중단하지 않는 방법을 말한다.

3.1 기능적 요구사항

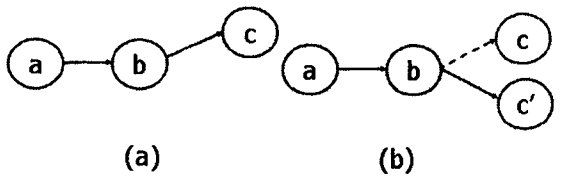
객체 핫 스와핑 프레임워크가 제공하는 기능적 요구사항은, 첫째 객체 단위로 스와핑 가능해야 하며, 둘째 객체 여러 개를 동적으로 스와핑 할 수 있는 기능을 제공해야 하며, 셋째 프레임워크를 사용해서 개발하고자 하는

개발자에서 제공되는 어플리케이션 프로그래밍 인터페이스(API)를 제공해야 한다. 또한 이러한 인터페이스는 사용하기에 쉽고 최대한 간단해야 한다. 기존의 시스템을 객체 핫 스와핑이 가능한 구조로 변경할 경우 API에 따라서 그 변경의 용이하거나 어렵게 된다. 즉 객체 핫 스와핑 프레임워크에서 제공하는 API는 기존의 객체 있는 메서드 호출하는 방법과 같거나 최대한 비슷해야 한다.

또한 클래스 로더에 대한 고려가 필요하다. 기존의 클래스 로더의 경우 클래스를 로딩한 뒤 객체를 생성하게 된다. 기존의 클래스 로더를 그대로 핫 스와핑을 위한 프레임워크에서 사용할 경우 로딩된 클래스를 같은 이름의 새로 버전 업된 클래스로 핫 스와핑한 뒤에는 이전 객체 혹은 클래스의 정보가 그대로 메모리에 남아 있게 된다. 핫 스와핑을 위한 클래스 로더에서는 이전의 클래스 혹은 객체에 대한 정보는 핫 스와핑 이후에는 메모리에서 제거되어야 한다. 핫 스와핑 후에는 핫 스와핑 이전의 클래스나 객체에 대한 정보가 메모리에 남아 있다면 메모리 누수를 일으키는 원인이 될 수 있다. 따라서 객체 핫 스와핑 프레임워크에서는 기존의 클래스 로더를 사용하지 않고 핫 스와핑을 위한 클래스 로더를 따로 설계, 구현해야 한다.

3.2 설계 상의 이슈

3.2.1 객체 참조의 투명성 문제



<그림 1> 객체 참조 투명성 문제

<그림 1>의 (a)는 핫 스와핑 이전의 상태이다. 객체 b가 객체 c를 참조하고 있고 객체 a는 객체 b로부터 객체c의 참조를 리턴 받아서 객체 c를 참조하는 경우이다. 즉 객체 a, b가 객체 c를 참조는 경우이다. 이러한 상황에서 객체 c를 c '로 변경한 경우 객체 b는 새로 변경된 c'를 참조하고 있지만 객체 a는 b로부터 넘겨받은 c의 핫 스와핑 이전의 객체를 참조하고 있게 된다. 객체 참조를 넘겨받는 이러한 구조의 경우 객체 참조의 투명성이 문제가 되게 된다. 핫 스와핑 프레임워크를 설계함에 있어서 이러한 객체 참조 투명성 문제를 해결해야 한다.

3.2.2 상태 전이 문제

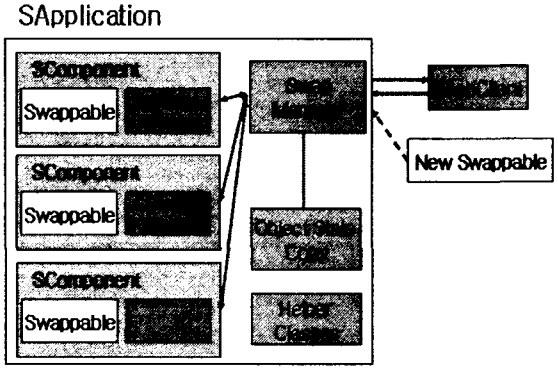
객체를 스와핑 함에 있어서 스와핑 이전의 객체의 상태를 스와핑 된 이후의 객체의 상태로 옮기는 문제가 있다. 스와핑 이후에 서비스는 동일하게 작동될 수 있으나 이전의 객체 상태는 보존하기 어렵다. 따라서 객체 핫 스와핑 프레임워크 내에서 객체 상태를 보존하기 위한 기능을 따로 제공해야 한다. 이는 자동으로 프레임워크에 의해서 처리되어야 하며 어떠한 객체를 스와핑 하더라도 객체의 상태는 보존되어야 한다.

3.2.3 트랜잭션

여러 개의 객체들이 서로 연관되어 있거나 의존성이 있는 경우 이들의 핫 스와핑을 하나의 트랜잭션 내에서 모두 처리되어야 하고 그 순서는 연관관계나 의존관계에 따라서 일정하게 스와핑 처리되어야 한다. 즉 하나의 이상의 객체를 스와핑 함에 있어서 전부 스와핑 되거나 전부 롤백하는 트랜잭션을 제공해야 한다. 또한 그 순서가 연관이나 의존관계에 따라서 처리되어야 한다.

4. 핫 스와핑을 위한 프레임워크의 설계

<그림 2>는 본 연구에서 제시하는 객체 핫 스와핑을 위한 프레임워크를 나타낸다.



<그림 2> 핫 스와핑 프레임워크

이 구조는 관련연구 [2]에서 제시한 프락시(Proxy)를 이용한 핫 스와핑 구조와 유사하다. 본 연구는 프락시 구조를 성능과 편리성의 기준에 따라서 정적 프락시(Static Proxy)와 동적 프락시(Dynamic Proxy)를 둘 다 제공하는 유연한 구조로 제시하고 이들을 사용하여 구현 방법을 제시한다.

<그림 2>의 프레임워크 내의 각 컴포넌트의 기능을 살펴보면 다음과 같다. Swap Manager는 새로 들어올 스와핑 가능한 모듈인 New Swappable과 기존의 스와핑 가능한 모듈인 Swappable을 교체하는 일을 담당한다. 이때 객체의 속성 매핑 관련은 Object State Copy를 통해서 한다. 본 프레임워크에서는 스와핑 가능한 모듈과(Swappable)과 또한 스와핑 가능한 모듈의 대리자인 프락시를 하나의 캡슐화된 모듈인 SComponent로 관리한다. SComponent의 함수호출 시에는 매개변수나 반환 값은 Helper Classes를 통해서 이루어진다.

4.1. Swappable Module

Swappable Module인 SModule은 핫 스와핑에 의해서 교체될 객체를 의미하며 일반 객체와는 구별하기 위해서 특정 인터페이스를 상속받아서 구현된다. Swap Manager에서는 이 인터페이스를 확인하고 그 객체가 핫 스와핑 가능한지 아닌지를 판별하게 된다. SModule이 가져야하는 정보는 다른 SModule과 구별되는 유일한 식별자 및 버전, 프락시를 통해서 외부의 클라이언트에게 호출하게 되는 서비스, 현재 SModule이 스와핑 처리 과정에서 어디에 속해 있는지는 알려주는 상태정보, 하나의 트랜잭션에서 다른 SModule과의 의존관계 정보, 핫 스와핑 이전 SModule과 이후 SModule의 상태 복사를 위한 매핑 규칙을 포함한다. 또한 핫 스와핑 하려는 SModule은 유한 상태(Finite State)를 가져야 한다. 어떤 특정한 시점에서 서비스를 제공하지 않는 상태(Idle State)를 가져야 하기 때문이다.

4.2. 동적 프락시와 정적 프락시

Dynamic SPorxy와 Static SProxy는 SModule에 대한 대리객체로써 참조문제, 스와핑 위한 작업 등을 처리한다. 각 각 동적, 정적인 방법으로 SModule의 함수호출을 대리한다. 정적 혹은 동적인 프락시를 사용하더라도 같은 SModule을 그대로 수정없이 사용 할 수 있다. 핫 스와핑을 위한 프레임워크의 설계 장점은 프락시의 설계에 따른 편의성, 재사용성 및 성능에 관계에 있다. 각 각의 프락시의 장단점에 대해서는 다음 절에서 자세히 다룬다.

4.3. Swap Manager

Swap Manager의 기능에는 새로운 SModule을 기다리고 기능, 스와핑 처리가 시작되기 전

에 SModule을 초기화하는 기능, 스와핑을 위해서 입력된 SModule이 올바른 SModule인지 보안에 문제가 없는 SModule인지 판별하는 보안 기능, 핫 스와핑 처리 동안 트랜잭션을 제공하는 트랜잭션 기능, 핫 스와핑의 완료가 지정된 특정 시간 안에 완료 할 수 있도록 하는 타임 기능, 핫 스와핑 과정을 모니터링 하거나 이벤트를 듣길 원하는 외부 컴포넌트에 이벤트를 발생, 전달하는 기능, 스와핑 가능한 모듈에 대한 정보 저장하는 저장소 기능을 가져야 한다.

4.4. 설계 이슈 해결방안

핫 스와핑을 위한 프레임워크를 설계함에 있어서 중요한 이슈는 객체 참조 투명성 문제와 객체 상태 전이 문제가 있다. 이 두 가지의 문제를 해결 할 수 없는 프레임워크는 서비스를 계속적으로 사용할 수 있는 동적 형상조정을 제공 할 수 없다. 본 프레임워크에서는 두 가지 문제를 해결하기 위한 방안은 제시한다.

4.4.1 객체 참조 투명성 문제 해결 방안

객체 참조 투명성 문제는 프락시 패턴을 사용한다. 프락시 패턴을 통해서 객체 참조를 프락시가 관리하게 되기 때문에 객체 참조 투명성 문제를 해결한다. 또한 객체 참조를 직접적으로 프락시를 거치지 않고 접근하는 경우를 대비해서 참조자 리스트를 보관하고 이 참조자 리스트에 SModule을 참조하던 객체의 리스를 저장한다. 핫 스와핑 이후 이 참조자 리스트의 특정 인터페이스를 통해서 변경된 참조를 갱신하게 된다.

4.4.2 객체 상태 전이 문제 해결 방안

객체 상태 전이 문제는 객체를 스와핑 하는 단계에서 객체 내부를 조사 할 수 있는 리플랙션 API를 사용해서 스와핑 되기 전 객체의 상태를 보관한 뒤 스와핑 이후 이전의 객체에서 얻어는 상태를 새로 스와핑된 객체에 매핑하는 과정을 한다. 본 연구에서는 Java Reflection API[10]를 사용했다. 이러한 매핑에는 일정한 법칙이 있을 수 있다. 이전 객체의 상태를 그대로 복원하는 경우가 있고 새로 스와핑 된 이후 초기화를 하는 경우가 있다. 이 밖에 다양한 매핑 과정을 위해서 스와핑 하는 단계에서 다양한 매핑 관계를 처리하기 위해서 객체 상태에 따른 매핑 리스트를 저장하게 된다.

5. 정적, 동적 프락시의 설계

본 논문에서는 프락시 변경에 따른 프락시 객체 참조의 투명성을 위한 두 가지 방법을 제안한다. 두 가지 방법은 프락시패턴을 이용하여 교체될 객체의 함수 호출 방법에 따라서 정적 프락시와 동적 프락시로 나눠진다. 여기서 말하는 정적, 동적의 의미는 3절에서의 정적, 동적의 의미와는 다르다. 스와핑 가능한 객체의 서비스 함수를 호출할 때 정적 함수호출로 이루어지면 정적 프락시, 서비스 함수를 동적 함수 호출[8, 9]로 처리하는 경우를 동적 프락시라고 한다.

5.1. 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 위임시켜 사용하기 때문에 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

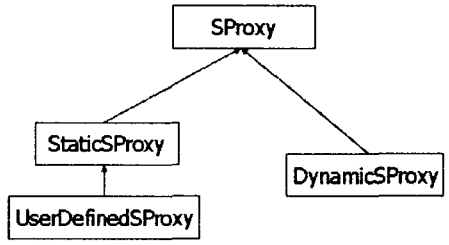
하며 SModule을 사용하는 다른 객체는 명시적인 함수를 사용해서 참조 리스트를 처리해야 하는 단점들이 있다. 명시적인 함수는 프락시 사용을 명시적으로 선언하는 addRef()와 프락시 사용을 마치고 해제하는 removeRef() 함수가 있다. 정적 프락시를 사용하는 경우 반드시 선언을 하고 나서 사용한 뒤 해제해야 한다. 또한 정적 프락시를 통해서 스와핑 가능한 모듈의 함수를 호출하는 과정에서 스와핑 가능한 상태에서는 함수 호출을 블록하는 과정이 필요하기 때문에 정적 프락시에서는 새로운 함수 호출을 블록할 수 있는 Locking 메카니즘이 필요하다.

5.2. 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 다른 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. 그리고 정적 프락시처럼 함수 호출 블록을 위한 Lock 메커니즘은 프레임워크 내부에서 구현하는 장점이 있다. 단점은 자바의 리플렉션(Reflection)을 이용한 동적인 객체 함수 호출을 하기 때문에 호출 속도가 느리다는 점이다.

5.3. 프락시 클래스 상속 구조

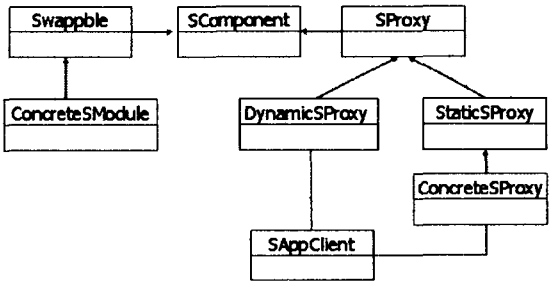
프레임워크 내부에서 정적 프락시와 동적 프락시를 동일시 취급하기 위해서는 <그림 3>과 같이 각각 동일한 프락시 인터페이스(SProxy interface)로부터 유도되어야 한다. 이러한 프락시 클래스 상속 구조는 시스템 내부에서 프락시와 스와핑 가능한 객체간의 의존성이 줄어든다. 즉, 같은 스와핑 객체를 동적, 정적 프락시에서 수정 없이 연결이 가능하게 되는 것이다.



<그림 3> 동적, 정적 프락시의 상속구조

5.4. SComponent 클래스

<그림 4>와 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.

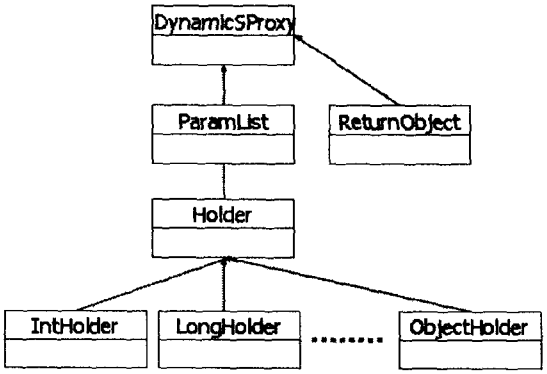


<그림 4> SComponent관련 클래스

5.5. DynamicSProxy 클래스

DynamicSProxy는 <그림 5>와 같이 객체

핫 스와핑 프레임워크 내부에 정의되어 있다. 동적 프락시에 정의된 함수를 모두 구현되어 있으며 추가로 동적인 함수 호출을 Swappable에게 전달해 주는 Invoke함수가 리플랙션을 사용해서 구현되어 있다. 파라미터 전달을 위한 ParamList클래스가 사용되고 ParamList를 만들기 위해서 자료형을 대표하는 여러 타입의 Holder클래스가 사용된다.

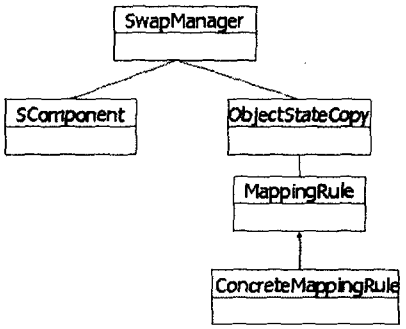


<그림 5> 동적 프락시 관련 클래스

5.6. 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.

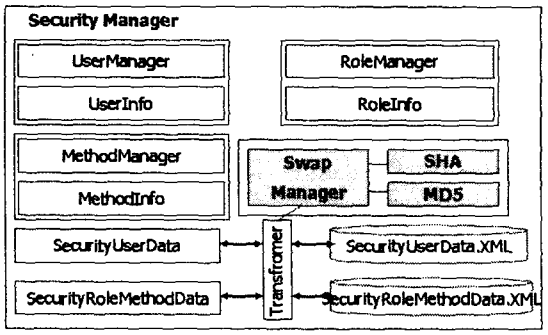
<그림 6>은 SwapManager클래스와 관련된 다른 클래스들 간이 개념도 있다. Swap Manager는 여러 개의 SComponent를 관리하며 스와핑 과정에서 스와핑 모듈의 내부 상태 복사를ObjectStateCopy를 통해서 이루어지진다. 내부 상태 복사 매핑 룰은 MappingRule인터페이스를 상속받아서 매핑 룰을 정의하게 된다.



<그림 6> SwapManager 관련 클래스

6. 테스트 및 결과

본 논문에서는 구현된 핫 스와핑 프레임워크의 성능을 평가하기 위하여 분산 시스템에서 많이 사용하는 <그림 7>과 같은 보완관리자의 메시지 암호화 알고리즘 객체를 핫 스와핑 하는 경우를 적용시켜 보았다. 분산 시스템의 관리영역 중 관리 콘솔 및 사용자들의 인증 및 보완 관리를 위해서 사용자들의 역할과



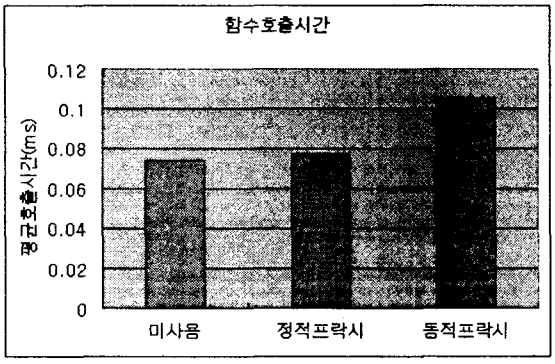
<그림 7> 보안 관리자

사용자 정보관리 그리고 그 사용자의 권한을 제어하기 위해서 보안관리자를 사용하게 된다. 이때 각 도메인의 보안 정책에 따라서 다른 보안 모듈을 사용할 수 있게 된다. 이러한 경우 변화해야 하는 암호화 모듈만 변경함으로써 전체 서비스에는 영향을 끼치지 않을 수 있다.

실험환경은 펜티엄3-1G, 256M, Windows 2000 Server, JDK 1.3.2이다.

6.1 함수 호출 시간

<그림 8>은 서비스 함수를 직접 호출하는 방법과 정적 프락시, 동적 프락시를 사용해서 서비스 호출을 하는 시간을 막대그래프로 비교한 것이다. 단위는 Millisecond 이다. 암호화 객체를 핫 스와핑 프레임워크를 사용하지 않고 바로 호출하는 경우는 평균 0.0739(ms), 정적 프락시는 평균 0.07692(ms), 동적 프락시는 평균 0.10546(ms)의 시간이 걸렸다. 정적 프락시의 경우 함수 호출을 프락시에서 스와핑 객체로 다시 호출하는 오버헤드 정도가 소요됨을 알 수 있다.

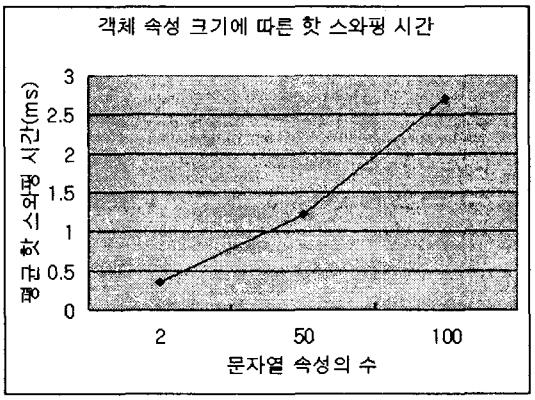


<그림 8> 함수 호출 시간 테스트 결과

정적 프락시의 경우 일반적이 함수 호출 시간 보다 많이 걸리지 않지만 동적 프락시의 경우 정적인 경우보다 약 1.4배의 시간이 걸리는 것을 나타났다. 그러므로 함수 호출이 빈번하지 않고 이전에 만든 스와핑 모듈에 대한 세부사항을 모를 경우 간단하게 사용할 수 있으나 함수 호출이 빈번하게 일어나고 실행 성능이 중요시되는 부분에서는 사용되기 어렵다는 결론을 얻을 수 있다.

6.2 핫 스와핑 시 상태 전이 속도

<그림 9>는 전이해야 하는 상태의 속성 하나 하나를 문자열이라 가정하고 속성 개수의 변경, 즉 전이 상태의 크기 변화에 따른 평균 핫 스와핑 시간의 변화를 비교한 것이다.



<그림 9> 객체 내부 속성크기에 따른 핫 스와핑 시간 테스트

문자열 속성을 2개 갖는 객체 스와핑의 경우는 그림 9와 같이 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. 테스트의 보안 관리자의 암호화 객체인 SHA를 MD5로 교체하는 경우의 핫 스와핑 시간은 평균 0.23263(ms)이 걸렸다.

7. 결론

분산 시스템에서의 시스템의 유지보수의 과정에서 시스템이 제공하는 서비스의 중단없이 업그레이드나 유지보수를 하는데 많은 어려움이 있다. 본 연구에서는 제공하는 서비스의 중단하지 않고, 즉 시스템의 고가용성을 보장하면서, 시스템을 형상 조정할 수 있는 핫 스와핑 프레임워크를 제시하였다. 우리는 핫 스와핑 프레임워크의 설계상의 이슈와 프락시

패턴을 이용한 객체 참조 문제 해결 방법을 소개하였다. 본 프레임워크는 함수호출 성능 높은 정적 프락시와 사용하기 편리한 관점에서의 동적 프락시를 모두 지원하고 있다. 본 논문에서는 정적, 동적 프락시 사용에 대한 성능 테스트 결과와 각 프락시의 활용방법에 대한 가이드라인을 제안하였다.

향후 과제로는 정적 프락시의 단점인 개발의 어려움을 정적 프락시 자바 코드를 자동으로 생성해주는 코드 생성기의 개발로 자동화한다면 해결 할 수 있다. 스와핑 할 수 있는 단위의 크기를 모듈 레벨에서 여러 모듈을 포함한 단위로 확장시켜 나갈 것이다.

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] L. Tan, B. Esfandiari, and B. Pagurek, "The Swap Box: A Test Container and a Framework for Hot-swappable Java Beans", Sixth International Workshop on Component-Oriented Programming, 2001.

[5] Ning, F., S-Module Design for Software Hot Swapping Technology, Technical Report SCE-99-04, Systems and Computer Engineering, Carleton University, May, 1999.

[6] Gang, A., Software Hot Swapping Techniques, Technical Report SCE-98-11, Systems and Computer Engineering, Carleton University, December, 1998

[7] Tim Lindholm ,Frank Yellin , "The JavaTM Virtual Machine Specification" available at URL : http://java.sun.com/docs/books/vmspec/2nd-edition/html/VMSpecTOC.doc.html

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

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

[10] 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

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

[12] Ousay H. Mahmoud , "Distributed Programming with JAVA", Manning computer publishing, 2001

[13] Douglas Lea, Concurrent Programming in Java, Second Edition: Design Principles and Patterns , Addison -Wesley, 1999

[14] Prashan Jain, Dougles C. Schmidt, "Service Configuratior Pattern for Dynamic Configuration of Service," Proceeding of the 3rd USENIX Conference on Object-Oriented Technologies and System, 1997

주 작 성 자 : 윤 태 웅
논문투고일 : 2003. 9. 20
논문심사일 : 2003. 10. 22
심사판정일 : 2003. 10. 22

● 저자소개 ●

윤태웅(e-mail : taewoong@konkuk.ac.kr)
2001 건국대학교 공과대학 컴퓨터공학과 학사
2003 건국대학교 정보통신대학 컴퓨터.정보통신공학과 석사
2003 ~ 현재 건국대학교 정보통신대학 컴퓨터.정보통신공학과 박사과정
관심분야: 분산 시스템, 소프트웨어 공학

민덕기(e-mail : dkmin@konkuk.ac.kr)
1986 고려대학교 공과대학 산업공학과 학사
1991 미시건 주립 대학 컴퓨터공학 석사
1995 미시건 주립 대학 컴퓨터공학 박사
1995 ~ 현재 건국대학교 정보통신대학 컴퓨터공학부 교수
2000 ~ 현재 소프트웨어 컴포넌트 포럼 기술분과 위원장
관심분야: 분산 및 병렬 시스템, 분산 객체 및 컴포넌트 기술, 미들웨어, 소프트웨어 시스템 아키텍처, 시스템 성능 분석, 웹 기반 분산 컴퓨팅, 웹 서비스