論文
2-7-7

# 웹 기반 원격교육을 위한 서비스관리시스템 프레임워크 (Service Management System Framework for Web-based Remote Education)

배 제 민*
(Je-Min Bae)

## 요 약

소프트웨어 개발에 있어 코드와 분석 및 설계 정보에 대한 재사용이 가능한 객체지향 프레임워크는 개발자들의 생산성을 직접적으로 향상시킬 수 있다. 객체지향 프레임워크는 프로그래머가 사용하거나 확장 수정이 가능한 어느 특정 분야의 클래스들과 그들 간의 연결성들의 집합이다. 이는 어느 특정 분야에 전문적인 설계안과 프레임워크에서 재사용될 코드를 담고 있는 메타 해결안인 것이다. 본 논문에서는 현재 인터넷을 기반으로 한 원격 교육 시스템에서 기본적으로 사용하는 게시판, 채팅, 화이트보드, ftp 어플리케이션에 대한 공통으로 제공하는 서비스를 추출하여 이를 프레임워크로 구축하였다. 이러한 서비스는 이종의 어플리케이션에서도 대부분 사용되므로 부품화 하여 이용될 수 있다.

## ABSTRACT

In the process of software development, object-oriented framework enables directly improving the productivity of the developer through the reuse of code, analysis and design informations. object-oriented framework is a set of usable and expandable classes and their connectivity. It is a meta solution that contains the code to be reused in the framework and the expert design results on a specific area. This paper constructs the framework that extracts the common services of BBS, chatting, white board and ftp applications for internet-based remote education system. These services can be mostly reused within heterogeneous applications in the form of component.

## 1. 서론

객체지향성은 표준 인터페이스와 상속에 의해 코드의 재사용성을 증가시키는 방법을 제공해왔고 이를 상품화한 클래스 라이브러리들은 잘 정의되고 테스트된 재사용 부품들을 제공해 왔다. 이는 독립된 단위의 객체와 클래스를 통해 이해성의 증대, 추상화, 캡슐화를 통해 재사용성과 유지보수성의 개선에 크게 기여를 하였지만 코드의 재사용만으로는 최종 개발자 특히 프로그래머들의 생산성을 직

* 정회원 : 관동대학교 컴퓨터교육과 교수

논문접수 : 2001. 7. 13.
심사완료 : 2001. 7. 21.

접적으로 향상시키지는 못하였다. 즉 클래스 및 클래스 라이브러리가 개발자들의 실질적인 코드 양을 줄여 주지 못하였다는 것이다. 클래스를 재사용하기 위해서는 클래스 라이브러리의 구조를 잘 이해하고 각 인터페이스를 파악한 후 전체 프로그램을 만들기 위해서 각 클래스 부품을 조립해야 한다. 이 과정에서는 대형 라이브러리 전체를 이해해야 하고, 기초 클래스를 선정한 후 상호 연관성을 제어하는 코드를 개발해야만 되는 것이다. 이로 인해 라이브러리의 습득 속도와 라이브러리 재사용으로 인한 추가 부담이 생기게 된다. 이런 이유로 인해 클래스 라이브러리의 재사용이 실질적인 재사용성을 향상시키지 못하였으며, 상호운영성에도 많은 제약점을 갖고 있다. 이에 대한 해결책으로 객체지향 프레임워크의 재사용이 제안되었다[10].

객체지향 프레임워크에서 재사용 대상은 함수나 어플리케이션뿐만 아니라 어플리케이션 영역에 대한 정의도 해당된다. 프레임워크는 프로그래머가 사용하거나 확장 수정이 가능한 어느 특정 분야의 클래스들과 그들 간의 연결성들의 집합이다. 클래스들의 집합이라는 점에서는 클래스 라이브러리와 동일하지만 단순한 클래스들의 모임이 아니라 어플리케이션을 개발할 수 있는 아키텍처를 제공한다. 어느 특정 분야에 전문적인 설계안과 프레임워크에서 재사용될 코드를 담고 있는 메타 해결안인 것이다. 그러므로 객체지향 프레임워크는 몇 가지 유사한 어플리케이션들에 공통적인 기능을 추출하는데 목적이 있다. 프레임워크를 개발하고 재사용함으로써, 재사용은 분석과 설계 정보까지 확장될 수 있다. 객체지향 프레임워크의 개발은 해당 영역의 많은 어플리케이션에 대한 분석과 설계를 요구하므로, 그 타당성은 그 영역에 대한 폭넓은 지식을 가진 경험 많은 소프트웨어 설계자의 가담여부에 따라 크게 달라지게 된다. 이러한 특정 인물에 대한 의존도를 줄이기 위해서도 방법론과 개발공정의 필요성이 제기되고 있다[5,10].

현재 프레임워크 개발 분야에서 해결되어야 할 문제점은 프레임워크에 대한 체계적인 개발 방법과 프레임워크의 사용 용이성을 위한 문서화방법이다. 프레임워크 문서화를 위해서는 증명된 객체지향 지식에 대한 추상화 정보를 제공하는 설계패턴이 이용될 수 있다[5]. 전통적인 언어에서의 빌딩블록은 정수, 배열, 포인터이고, 객체지향 언어에서는 객체이며 전통적인 프로그래밍에서, 리스트, 트리, 큐와 같은 자료구조의 발명은 공통적인 문제를 해결하였고 포함된 빌딩블록들을 추상화시킨 것과 마찬가지로 설계 패턴도 비슷하게 공통적인 객체지향 해결책을 갖고있는 "객체구조(object structures)"로 볼 수 있고, 기본이 되는 빌딩블록 즉, 객체들을 추상화시킨다. 즉 설계 패턴은 객체지향 소프트웨어 개발에서의 자료구조 또는 알고리즘이라 할 수 있다. 이러한 설계 패턴은 프레임워크 설계와 문서화에 있어 유용하게 사용될 수 있다[2].

2장에서는 기초가 되는 관련연구들을 핵심적인 내용을 중심으로 간단히 기술하였고, 3장에서는 Hot spot과 설계 패턴을 기반으로 한 프레임워크의 체계적인 개발 방법에 대해 제안하였다. 4장에서는 제안한 방법에 따라 구축된 프레임워크를 통해 새로운 서비스 어플리케이션을 생성하는 과정을 기술하였고 5장에서는 결론을 기술하였다.

## 2. 프레임워크 개념 및 문서화 방법

클래스 라이브러리 접근방법의 문제점은 라이브러리 내에 영역전문지식을 포함하거나 디폴트 행위를 제공하는 것이 어렵다는 것이다. 부품 라이브러리에는 재사용자가 어플리케이션을 구축할 때 필요한 부품의 수가 적고, 적은 부품으로 어플리케이션을 개발할 때에는 다른 부품들 사이의 연결관계를 정의해야만 한다. 이러한 제약조건들은 프레임워크를 사용함으로써 해결될 수 있다. 어플리케이션 설계자는 각 함수를 호출하는 시기와 방법을 알 필요가 없다. 이는 프레임워크가 자동적으로 해주게 된다. 프레임워크내의 부품들 사이의 연결관계는 이미 정의되어있고 사용자는 이에 관여할 필요가 없는 것이다[8,10].

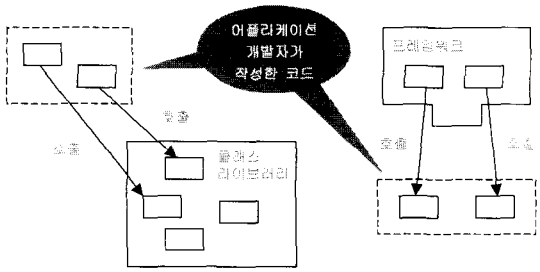
프레임워크 역시 클래스들의 collection을 포함하므로 일종의 클래스 라이브러리라 할 수 있지만, 클래스 라이브러리내의 모든 클래스들은 그 자체로만 유일하고 프레임워크내의 대부분의 클래스들은 서로 종속적이기 때문에 프레임워크 밖에서는 사용할 수 없는 경우가 많다. 재사용 라이브러리에서 가장 어려운 점은 재사용의 가치가 충분히 큰 영역 종속적인 아키텍처와 부품들을 포함해야만 한다는 점이다. 객체들은 거의 그 자체로 유용한 행위를 제공하지 않고 다른 객체들과 합성(상호작용)될 때 유용한 기능이 제공될

수 있기 때문에 이를 해결할 수 있는 방법이 필요한 것이다.

객체지향 프레임워크는 관계된 문제점들 집합의 해결책에 대한 추상설계를 포함하고 있는 클래스들의 집합으로 정의할 수 있다. 또한 프레임워크는 어플리케이션 서브시스템 영역에 대한 책임들을 수행하기 위해 상호·협력하는 객체들의 집합이라고 정의할 수 있다[6].

그러므로 객체지향 프레임워크는 그것이 추상적이던 구체적이던 서로 협력하는 클래스들의 집합이고, 이는 소프트웨어의 특정 클래스에 대해 재사용 가능한 설계를 구성한다. 프레임워크는 설계를 추상클래스들로 분리하고, 그들의 책임과 협동관계, 쓰레드 제어 관계를 정의함으로써, 프레임워크를 사용하여 구축된 어플리케이션의 아키텍처를 결정한다. 어플리케이션 영역에 공통된 설계결정들을 뽑아냄으로써 어플리케이션 설계자는 개발하려는 어플리케이션에서 새로 요구되는 곳에만 집중할 수 있게 된다. 그러므로 프레임워크 개발은 코드 재사용뿐만 아니라 설계의 재사용도 제공하는 것이다[5].

동적 바인딩으로 인하여 프레임워크는 객체를 그 구현사항에 관계없이 다루게 된다. 프레임워크를 재사용하여 새로운 클래스를 파생시키려는 어플리케이션 개발자는 추상 슈퍼클래스에 의해 제공되는 템플리트에 따라 코드를 작성하게 된다. 프레임워크는 그 메소드를 호출하게 될 것이고, 이러한 제어 관계로 인해 프레임워크는 [그림 1]과 같이 역 호출 라이브러리로 간주된다[10].




[그림 1] 프레임워크와 클래스 라이브러리의 차이점

[Fig. 1] Differences of Framework & Class Library

프레임워크 개발 시 투입되는 초과시간은 일종의 투자로 보아야 한다. 또한 부품 라이브러리의 설계보다는 프레임워크의 설계가 더 어렵지만, 재사용 관점에서 보면 그 잠재적 가치는 훨씬 높다. 프레임워크의 설계가 더 어려운 이유는 그 아키텍처를 설계해야 할 뿐만 아니라 내부 부품들 사이의 연결관계도 설계해야 하기 때문이다. 부품 라이브러리에 대한 부품을 설계할 때에는 그러한 결정들이 이루어지지 않는다. 성공적인 프레임워크 개발을 위해서는, 팀의 공정 및 조직에 의한 지원이 있어야만 한다. 프레임워크와 재사용으로 인한 이점은 프레임워크를 여러 번에 걸쳐 재사용함으로 인해 반복되어 얻어질 수 있다.

프레임워크를 이해하고 그 추상화를 사용하는데 드는 비용이 처음부터 프로그래머가 개발하는 비용보다 많다고 예상된다면, 그 프레임워크는 결코 재사용되지 않을 것이다[1]. 프레임워크에 대한 정보를 가진 설계자는 프레임워크에 대해 분명한 형태와 그 마이크로-아키텍처에 대해 알고 있지만 프레임워크의 사용자는 문서화가 제대로 이루어지지 않았을 경우 너무나도 많고 불분명한 클래스들을 어떻게 사용해야 할 지 모르게 된다[8]. 또한 프레임워크를 개발 시 이득은 프레임워크가 재사용되어 새로운 어플리케이션들이 짧은 시간에 개발될 수 있고 테스팅과 유지보수에 있어 제한된 노력만이 필요할 때 그 이점이 살아날 것이다. 따라서 프레임워크의 문서화는 그 재사용성에 있어 기본이다. 문서화에는 프레임워크의 목적, 프레임워크 사용방법, 프레임워크의 상세 설계 내용 등을 기술해야 한다. 이런 이유로 해서 모든 소프트웨어 프로젝트에는 공정과 결과물에 대한 문서화가 필요하지만, 특히 프레임워크의 개발에 있어서는 그것들이 더 중요한 것이다[9]. 상세 설계는 어플리케이션 개발자들이 사용할 수 있어야 할뿐만 아니라, 프레임워크의 사용방법 역시 기술하고 있어야 한다. 프레임워크 설계의 몇몇 특징들은 코드로 표현하기 어렵기 때문에, 이들에 대한 문서화방법으로는 설계패턴이 이용되고 있다[6].

소프트웨어 분야에 있어 설계 패턴이란 객체지향 설계를 하는 동안 자주 발생되는 문제점들에 대한 일반적인 설계 정보를 의미한다[3]. 설계 패턴들은 설계 경험을 효과적으로 사용할 수 있는 형태로 추출하여 성공적인 설계와 아키텍처를 좀더 쉽게 재사용할 수 있도록 하는데 그 목적이 있다 [4]. 설계 패턴은 설계

결정사항들을 동기화 시킬 수 있으므로 여러 개의 마이크로 아키텍처로 구성된 프레임워크를 문서화하는데 있어 적용될 수 있다[5,7,8]. 사실 여러 공통된 프레임워크 설계 문제점들이 다른 설계자에 의해 같은 방법에 의해 여러 번 해결되어왔다. 설계 패턴들은 목적에 따라 생성적(creational), 구조적(structural) 그리고 행위적(behavioral)으로 나누어진다. 생성적이란 객체 생성과 관련되었음을 나타내고, 구조적이란 클래스나 객체들의 합성을 처리함을 나타내며, 행위적이란 클래스 또는 객체들이 상호작용하고 책임을 분산하는 방법들을 특성화한 것이다.

그리고 목적에 따라 구분된 패턴들을 범위(scope)에 따라 다시 클래스와 객체 패턴으로 나누었다. 클래스 패턴은 클래스와 그들의 서브클래스간의 관련성을 다룬 패턴들로 이 관련성은 상속을 통해서 정해지고, 객체 패턴이란 객체 관련성을 의미하며 실행 시 변할 수 있고 좀더 동적인 특성을 갖는다.

## 3. 프레임워크 개발 방법

프레임워크 개발 과정은 기존의 소프트웨어 개발 과정에서 각 과정별로 프레임워크를 개발하기 위해 추가적인 노력이 필요하다. 다음은 각 과정에 대한 개략적인 설명이다.

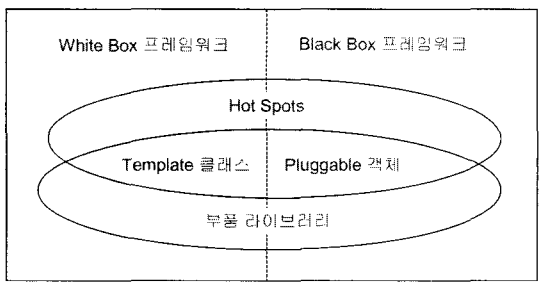
가. 계획 단계는 해당 영역에 있는 여러 어플리케이션에 공통적인 요구 사항을 파악하여 프레임워크의 요구명세를 작성하는 단계이다.

나. 구축 단계는 분석의 결과를 바탕으로 시스템화해 나가는 과정으로, 프레임워크에서는 설계 패턴을 사용하여 객체구조 즉 마이크로 아키텍처를 설계하고 이를 문서화시킨다. 또한 Hot spot의 식별과 명세를 통해 프레임워크의 기본 특성들을 이루는데 이용될 것이다.

다. 구현 단계는 시스템을 사용할 수 있는 형태로 개발하는 과정이다.

일반적으로 프레임워크 구축을 위해 개발한 어플리케이션의 수가 많으면 많을수록 좀더 일반적인 프레임워크를 만들 수 있지만 처음부터 프레임워크에서 지원할 모든 변경사항을 고려하기 위해 모든 어플리케이션을 개발하기는 불가능하다. 그러므로 몇 가지 어플리케이션의 개발을 통해 그 공통적인 지식을 갖는 프레임워크를 개발하고 이의 재사용을 통해 프레임워크는 점진적으로 진화시켜 나가야 할 것이다. 세부적으로는 더 많은 공정이 필요하지만 본 논문에서는 분석, 설계, 구현 단계에 있어서 프레임워크 개발에 필요한 사항 만을 설명하겠다.

제안하는 방법은 구축될 프레임워크의 구성 요소 및 그 개념에 대해 설명한다. 일반화를 통한 추상화 작업을 할 때 어떤 프레임워크는 상속에 너무 의지하고, 어떤 것들은 폴리모피즘에 의한 합성에 너무 의지한다. 상속은 부품들 사이에 강한 결합도 관계를 발생시키며 재사용하려는 부품들을 수정할 수 있도록 해줌으로써 원래의 설계자가 변경되리라고는 예상하지 못한 것들을 바꿀 수 있다. 또한 새로운 클래스를 만든다는 것은 프로그래밍 작업을 수반하게 된다. 반면 합성을 사용할 경우에는 설계자가 무엇이 변경될 것인지를 미리 알고 있어야 한다. 합성은 강력한 재사용 기법이지만, 정적인 프로그램 텍스트를 시험해봄으로써 이해하기는 어렵다. 또한 합성은 실행시간 시 변경될 수 있지만 상속은 정적이고 실행시간 시 쉽게 변경될 수 없다. 따라서 [그림 2]와 같이 프레임워크는 개발 범위에 따라 화이트박스 프레임워크와 블랙박스 프레임워크로 구분한다.




**[그림 2] 프레임워크 구성 요소**

[Fig. 2] Framework Components

각각의 구성요소에 대한 정의는 다음과 같다.

1) Hot spots : 프레임워크를 기반으로 어플리케이션을 개발하게 되면, 계속해서 반복되어 작성되는 유사코드가 발생될 것이다. 프레임워크내의 이러한

부분을 Hot spot이라고 한다. 이런 공통의 코드를 공유하기 위해서는 변화 가능한 코드 즉 Hot spot을 식별하고 이를 지원하는 환경이 개발돼야만 한다.

2) 화이트박스 프레임워크 : 화이트박스 프레임워크에서는 상속을 통하여 구성 요소를 변경하고 새로운 클래스를 생성시킴으로써 프로그래밍을 증진시킬 수 있다. 변화에 대한 처리는 일반적으로 폴리모피즘 기법을 적용한다.

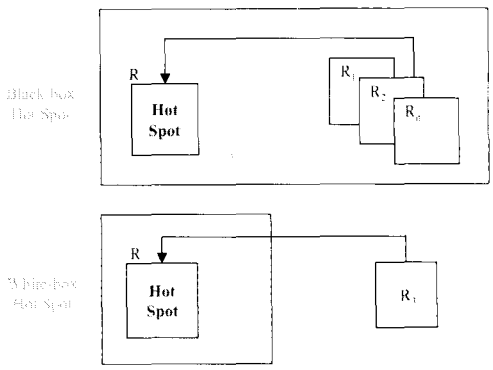
3) 블랙박스 프레임워크 : Hot spot들을 캡슐화시키고 플러그 객체들을 개발함으로써 블랙박스 프레임워크가 개발될 수 있다. 부품 라이브러리를 조직하기 위해서는 상속을 사용하고, 어플리케이션에 부품들을 결합시키기 위해서는 합성을 사용한다.

4) 부품 라이브러리 : 프레임워크의 인스탄스화에 필요한 유사 객체들의 작성을 피할 수 있는 방법이 제시되기 위해서는 부품 라이브러리가 필요하다. 일반적으로 추상클래스들은 프레임워크 내에 있게 되고 구체적인 클래스들은 어플리케이션에 있게 된다. 부품 라이브러리에는 프레임워크로부터 파생될 다양한 어플리케이션들의 생성에 필요한 모든 구체적 클래스들이 있게 된다.

5) Pluggable 객체 : [그림 3]과 같이 Hot spot은 블랙박스와 화이트박스 프레임워크에서 그 의미가 다르다. 블랙박스 프레임워크에서의 Hot spot은 이미 그 변경사항이 예측되어 프레임워크 개발 시 그 대안을 준비한 경우를 의미한다. 즉, [그림 3]에서와 같이 특정 Hot spot R에 플러깅될 수 있는 $R_1$, $R_2$, ... $R_n$의 부품이 프레임워크 내에 포함된다. [그림 2]에서 설명된 바와 같이 pluggable 객체는 블랙박스 프레임워크에서의 Hot spot으로 부품 라이브러리에 포함된다. 반면 화이트박스 프레임워크에서는 사용자가 임의의 코드를 작성하여 Hot spot R에 끼워 넣을 수 있고, 프레임워크에서는 이에 대해 준비된 Template 클래스를 사용자에게 제공할 수 있다. 이는 기존의 상속기반 재사용 라이브러리를 제공하는 접근방법과 역호출관계를 제외하고는 동일하다.

구현을 위해서는 작성할 필요가 있는 서브클래스들에서 변경부분은 무엇인 지를 결정하고 그 차이가 간단히 상수, 심벌 또는 클래스 참조에 있다면 그 참조를 포함하는 인스탄스 변수를 생성하고, 인스탄스 생성 메소드 내에 그 객체 안으로 인스탄스 변수를 전달하면 된다. 만약 변경이 코드 일부분에서 이루어진다면 인스탄스 생성 메소드에 그 코드를 표현하는 블록을 전달하고, 다시 하나의 인스탄스 변수에 그것을 저장하면 된다.

프레임워크를 사용하여 새로운 어플리케이션을 만들 때 기본적으로는 프레임워크에서 요구하는 Hot spot을 수정하고 설계 패턴으로 정의된 특정 설계안을 선택함으로써 이루어지게 된다. 이는 데이터 기반 즉 블랙박스 프레임워크를 재사용하여 새로운 어플리케이션을 생성하는 과정을 설명한 것이다. 하지만 프레임워크는 아키텍처 기반 즉 화이트박스 프레임워크에 대한 재사용 역시 허용한다. 이는 기존의 클래스 라이브러리 접근방법과 근본적으로 그 원리가 같다. 다만 프레임워크가 준비된 부품들 사이의 연결관계를 제어하고 프레임워크 사용자에게는 필요한 추가 코드를 요구한다는 점이 다르다.



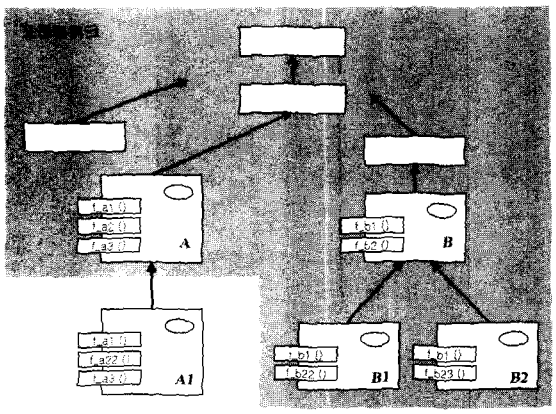

[그림 3] 블랙박스와 화이트박스 Hotspot
[Fig. 3] Hotspot of Blackbox and Whitebox

이러한 Hot spot이 요구 분석 단계에서 식별되어지면 영역에 필요한 구체적이고 필수적인 구성 패턴들과 어울려져 영역에 한정된 설계 패턴을 형성하게 된다. Hot spot을 이용한 프레임워크 설계는 보다 체계적인 소프트웨어 개발이 되게 한다. 그러므로 본 논문에서는 Hot spot을 기본으로한 객체지향 프레임워크의 설계방법을 제안한다.

블랙박스 프레임워크에서는 영역에 대한 완벽한 분석과 풍부한 영역 지식을 바탕으로 영역의 일반화된 특성과 범용적으로 사용 가능한 객체를 추상클래스로 정의한다. 그 후 어떤 문제에 대한 다양한 대안을 미리 정의한 추상클래스의 하위 클래스로 만든다. 이는 객체지향의 폴리모피즘이나 동적바인딩의 특성을 이용하는 것이다.

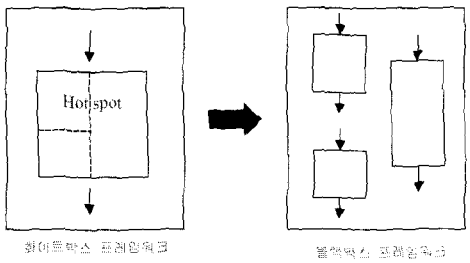
이와 같이 블랙박스 프레임워크는 수정을 위해 미리 준비된 부품을 갖고 있다. 수정은 프로그래밍을 통해서가 아니라 합성을 통해서 이루어지고 Hot spot은 재정의된 메소드에 대응된다.

*[그림 4]에서의 클래스 B는 두 개의 B1, B2 서브클래스를 갖고 있다.* 클래스 B의 추상 메소드인 f_b2()의 기본 구현은 하위 클래스의 f_b22()와 f_b23()에서 재공한다.




[그림 4] 클래스 라이브러리의 구조
[Fig. 4] Structure of Class Library

블랙박스 프레임워크의 주요 특성 중 하나는 새로운 어플리케이션이 재컴파일을 요구하지 않고 생성된다는 것이다. 이는 요구된 다양성을 제공하는 분리된 서브시스템에 의해 각 Hot spot을 구현함으로써 가능하다. [그림 4]와 같이 모든 Hot spot 서브시스템의 구조는 비슷한 방식으로 구성된다. Hot spot 서브시스템은 대안적인 클래스들의 집합을 포함하고, 그 각각은 변경사항이 발생할 때 합성되게된다. 프레임워크 사용자는 프레임워크 컴포넌트로부터 어플리케이션을 구성할 때 그 집합으로부터 하나 이상의 클래스를 선택한다. 이는 프레임워크 사용자가 선택한 경우에 대한 Hot spot의 다양성을 바인드한다는 것을 의미한다.




[그림 5] Hotspot의 특성별 분리
[Fig. 5] Hotspot separation by character

또한 [그림 5]와 같이 각 Hot spot은 서로에 독립적인 요소화 특성을 가진 부분으로 분리되어야한다. 각 요소화 특성들은 프레임워크의 Hot spot을 다시 구성한다. 이는 Hot spot에 대해 모든 대안적인 선택을 제공해야하기 때문이다. 예를 들어 세 개의 관련되어있지만 독립된 특성들이 있고 각각은 10, 10, 20의 선택이 가능하다고 하자. 세 개의 특성이 하나의 Hot spot으로 합쳐질 때, 프레임워크는 이 Hot spot에 대해 2,000개의 대안을 제공해야만 한다.

이는 실질적으로 불가능하다. 하지만 변화 가능한 특성들이 분리되었을 때 각 요소화 특성은 자체의 Hot spot에 포함되고 프레임워크는 단지 40개의 대안만을 제공하면 된다. 이는 이상적이다. 화이트박스 프레임워크에서는 Hot spot에 대한 실질적 대안을 제공하지 않고 단지 프레임워크 사용자에게 수정할 수 있도록 만 해주었기 때문에 문제점이 같은 형태로 존재하지 않는다.

블랙박스와 화이트박스 프레임워크 사이에 명확한 구분을 할 필요는 없다. 그들간의 스펙트럼은 지속적이기 때문이다. 화이트박스 프레임워크가 특성별로 분리되었을 때 자주 선택되는 대안들은 미리 계획될 수

있고, 블랙박스 모드의 완전한 서브클래스로 대치될 수 있다. 반면 잘 사용되지 않는 경우들은 새로운 서브클래스의 개발로 만들어질 수 있도록 화이트박스 모드에 두어야 한다.

객체지향 프레임워크를 개발 시 미래에 변경이 올 것이라고 예상되는 부분에 대해 예측하는 것은 매우 중요하다. 프레임워크 설계 단계에 있어서 식별된 변화 가능한 부분은 이와 같이 Hot spot으로 정의되어질 것이다. Hot spot을 추상화시키고 프레임워크의 마이크로 아키텍처로 정의하였을 때 이는 설계 패턴으로 문서화시킬 수 있고 이 공통된 설계 정보를 통해 프레임워크의 개발 및 재사용에 대한 이해를 증진시킬 수 있다. 다음 표는 프레임워크의 여러 부분들이 어플리케이션간에 변경이 올 때 사용할 수 있는 설계 패턴을 소개하고 있다. 이와 같이 설계 패턴들은 다양한 변경타입을 캡슐화시키고 이를 문서화하고 있다. 본 논문에서의 설계 패턴은 다음과 같이 4가지 기본 요소를 통해 문서화된다.

① 패턴이름 : 설계어휘를 증가시키고, 상위레벨에서 설계 추상화를 표현한 것이며 패턴에 대한 정의를 포함한다.
② 문제점 : 특정 설계 문제점에 대한 동기로서 패턴을 적용할 시기에 대해 설명한다.
③ 해결책 : 제시된 문제점에 대한 해결책으로 설계를 구성하는 요소들, 그들간의 관계, 책임, 협동관계 등을 클래스 다이아그램과 텍스트를 사용하여 기술한다.
④ 결과 : 패턴을 적용했을 때의 결과와 trade-off를 설명한다.

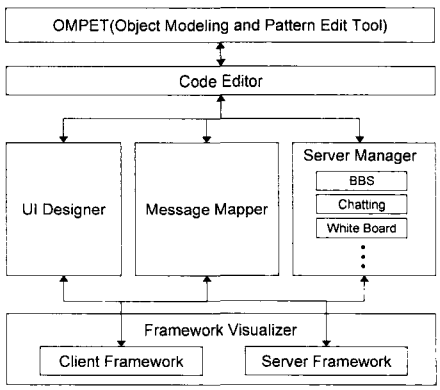
## 4. 서비스관리시스템

이번 장에서는 인터넷을 통한 원격교육에 있어 자주 사용되는 채팅, 게시판, 화이트보드, ftp 등에서 공통으로 사용되는 서비스 기능을 수행하는 어플리케이션을 구축된 프레임워크를 통해 개발하는 과정을 설명한다. 이를 위해 해당 영역의 공통 지식을 추출하여 지금까지 제안된 프레임워크 개발 방법에 따라 프레임워크로 구축하였다.

[그림 6]은 구축된 프레임워크에 대한 전체 구조도이다.

각 구성요소를 간단하게 설명하면 다음과 같다.

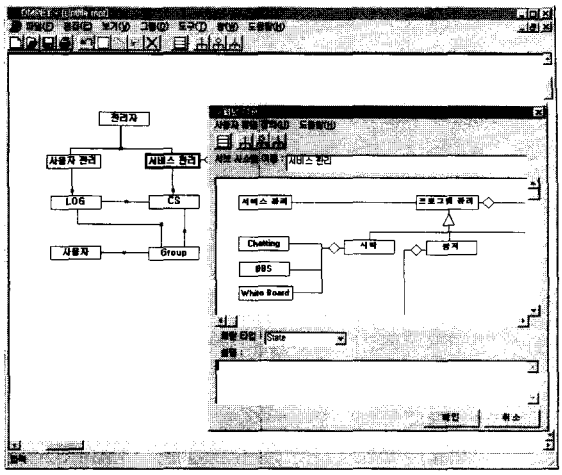
1) OMPET(Object Modeling and Pattern Edit Tool) : 클래스 다이아그램과 패턴에 대한 편집 프로그램.
2) 코드 편집기(Code Editor) : OMPET의 결과물인 클래스 다이어그램과 패턴정보에 따라 생성된 템플리트 소스 코드 편집기.
3) UI 설계기(UI Designer) : 클라언트 어플리케이션의 UI의 서식 형태를 결정하는 서브 시스템과 라이브러리.
4) 메시지 사상기(Message Mapper) : 클라이언트 어플리케이션의 UI Designer와 서버 어플리케이션의 Server Manager와의 사상관계를 관리하는 시스템.
5) 서버 관리기(Server Manager) : BBS 서버, White Board 서버, Chatting 서버를 관리하는 시스템.
6) 프레임워크 브라우저(Framework Browser) : 각 서브시스템에 대한 모델을 계층적으로 보여주고 이에 대한 브라우징도 할 수 있도록 해주는 프레임워크 UI 시스템.




[그림 6] 서비스관리시스템 프레임워크
[Fig. 6] Service Management System Framework

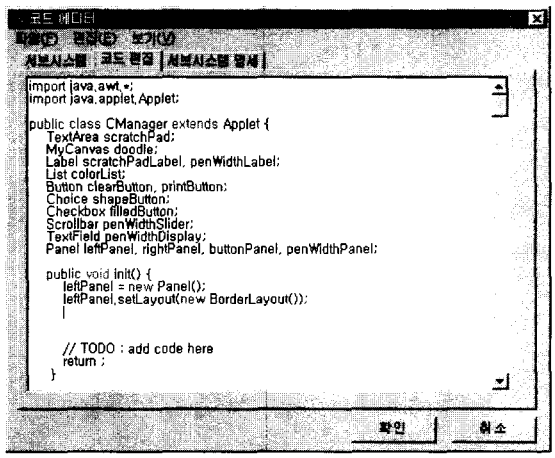
본 환경에서 어플리케이션을 생성하는 과정을 설명하면 다음과 같다. 먼저 OMPET를 이용하여 어플리케이션의 설계단계에서 기본 모형을 재사용할 수 있고 패턴을 적용하여 나중에 프레임워크 브라우저에서의 전체 구조를 보여줄 때 참조를 하게 된다. 이러한 설계 모형이 결정이 되면 코드 편집기를 통하여 화이트박스 프레임워크를 통한 재사용이 가능하다. UI 설계기는 블랙박스 프레임워크를 재사용할 수 있도록 해주는 서브시스템으로 본 사례는 웹 환경의 서비스 관리 시스템의 생성을 지원하므로 각 서비스 프로그램의 사용자 인터페이스에 대한 템플리트를 제공한다. 이는 HTML 태그로 작성된 문서들이 될 것이다. 이를 이용하여 어플리케이션의 UI를 결정하면 서버 관리기를 이용하여 시스템에 사용되는 여러 서버에 대한 서비스 지원 여부와 서비스 특성을 결정하게 된다. 그리고 UI 설계기와 서버 관리기를 사상시켜주는 메시지 사상기를 통해 두 프로그램이 통합되면 프레임워크 브라우저를 통해 생성된 어플리케이션을 브라우징하고 소스코드를 생성한다.

우선 OMPET를 사용하여 기본적으로 제공되는 기본 서비스 모델에서 원하는 모델로 변경을 할 수 있다. OMPET는 객체 모델링을 과정을 지원해주는 도구로서 프레임워크의 라이브러리에 저장된 특정 패턴을 선택하여 이를 재정의를 할 수 있는 기능을 가지고 있다. [그림 7]에서는 기본적인 제공되는 모델에서 서비스 관리가 선택이 되어 서비스 관리에 대해 이미 적용된 패턴을 보여주고 있다. OMPET를 통해 사용자는 객체 모델링을 할 수도 있고 여기에서 적용할 패턴을 결정하거나 패턴에 대한 새로운 정의도 할 수 있다. 그리고 사용자가 정의한 새로운 패턴을 적용하여 관련 모델을 다시 모델링할 수도 있다.



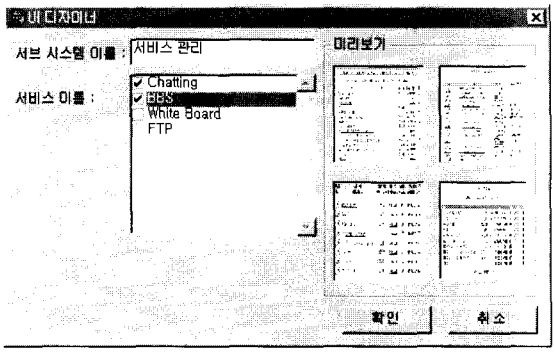

[그림 7] 모델링 과정
[Fig. 7] Modeling Process

이 시점에서 어플리케이션에 대한 설계는 두 가지 경우로 분리된다. 첫 번째는 프레임워크 라이브러리부터 가져온 모델을 그대로 사용하는 경우이고 두 번째는 사용자가 이를 재정의하여 모델링한 경우이다. 두 번째 경우에는 이미 작성된 모델에 대한 소스 코드를 재사용할 수 없을 것이다. 즉 화이트박스 프레임워크를 통한 재사용의 경우이므로 본 환경에서는 사용자는 이런 경우 [그림 8]과 같은 코드 편집기를 통해 새로운 코드를 작성할 수 있다.




[그림 8] 코드 편집기
[Fig. 8] Code Editor

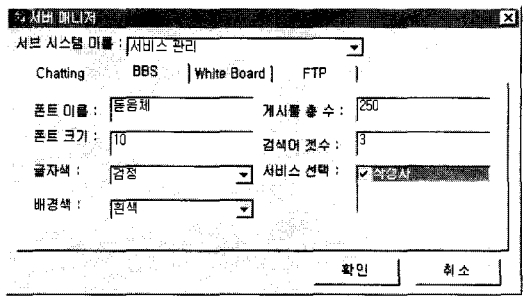
어플리케이션에 대한 모델이 작성되고 구체적인 코드가 만들어지면 UI를 결정해야 한다. [그림 9]에서는 OMPET에서 결정한 모델에 적용될 서브시스템의 UI의 서식을 결정하는 UI 디자이너의 예를 보여주다. 그림은 BBS를 선택한 경우이고, 이는 웹 브라우저에 나타날 BBS의 UI를 결정하게 된다.



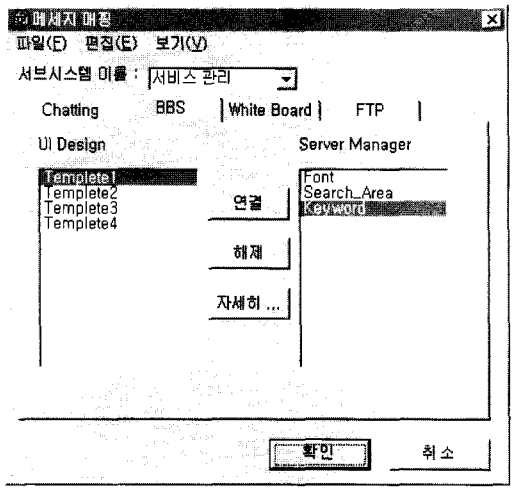

[그림 9] UI 설계기
[Fig. 9] UI Designer

[그림 10]은 서버 관리기의 예로 개발될 어플리케이션인 서비스 관리 서브시스템의 Chatting, BBS, White Board, FTP 서비스에 서버 쪽에서 결정되어야 할 내용들을 사용자가 기술할 수 있도록 해준다. 이는 블랙박스 프레임워크에 의한 재사용의 경우를 보여준 것으로 BBS의 경우 설계과정에서 폰트 타입, 폰트 크기, 글자색, 배경색, 게시물의 총 개수 그리고 검색어 개수가 Hot spot으로 추출되어 이에 대한 결정을 사용자로부터 입력받게 된다.

[그림 11]에서는 UI 설계기의 서식 파일과 서버 관리기에서 사용자로부터 입력받은 사항과의 연결 및 해제를 제어하는 메시지 사상기를 보여주고 있다. 이와 같이 프레임워크에서 지원하는 기본 기능만으로도 새로운 어플리케이션을 개발할 수 있지만 프레임워크 사용자가 항상 프레임워크에 준비된 부품들의 조합만으로 어플리케이션을 생성시키지는 않을 것이다. 실제로 프로그래밍 경험이 많은 개발자의 경우 프레임워크를 사용하여 어플리케이션을 개발하더라도 새로운 기능을 첨가하는 작업을 할 확률이 매우 높다. 이를 위해 프레임워크에서는 화이트박스 재사용을 지원하고 있다.




[그림 10] 서버관리기
[Fig. 10] Server Manager

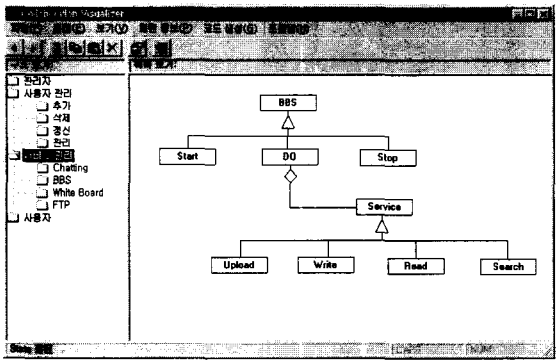



[그림 11] 메시지 사상기
[Fig. 11] Message Mapper

[그림 12]는 프레임워크 브라우저의 예로 오른쪽 TreeView부분에서는 서브시스템인 관리자, 사용자 관리, 서비스 관리, 사용자를 보여주고 각각의 서브시스템은 계층적으로 하부에 서브시스템이 존재함을 알 수가 있다. 브라우저는 각 시스템에 대한 모델을 계층적으로 보여주고 이에 대해 브라우즈도 할 수 있게 해준다. 또한 브라우저의 상태 표시줄에는 선택된 서비스에서 적용된 패턴을 보여주고 있다. 여기서는 State 패턴이 BBS 서비스에 사용되었음을 보여주고 있다.

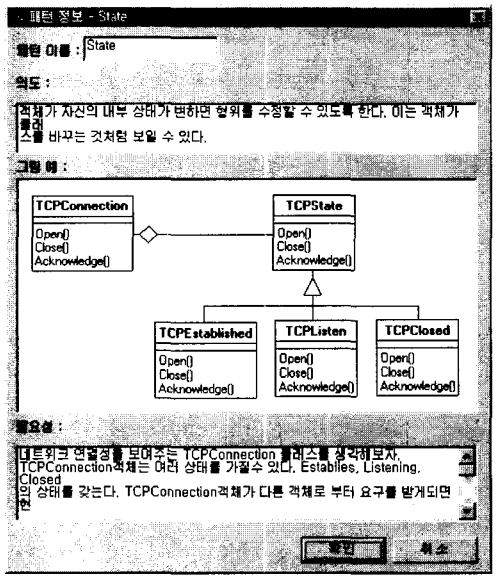
생성시킬 어플리케이션에 대한 모델이 완료되면 코드 생성 메뉴를 통해 OMPET, 코드 편집기, UI 설계기, 메시지 사상기, 서버 관리기를 통해서 받아들인 사용자 요구를 통합하여 이에 대한 서비스 관리 시스템의 소스 코드를 생성한다.

패턴정보 메뉴는 특정 서브시스템에 적용된 패턴을 자세히 보고자 할 경우 원하는 서비스를 선택하면 [그림 13]과 같은 패턴 정보 대화상자를 통해 해당 패턴에 대한 정보와 그 구조도를 사용자에게 보여준다.




[그림 12] 프레임워크 브라우저

[Fig. 12] Framework Browser




[그림 13] 패턴 정보 대화상자

[Fig. 13] Dialogbox of Pattern Information

## 5. 결론

본 논문에서는 프레임워크 개발에 있어 문제점으로 지적되고 있는 개발 방법과 이를 통해 원격교육에서 기본적으로 사용되는 서비스관리시스템 프레임워크를 구축하였다.

먼저 프레임워크에 대한 체계적인 설계 방법을 제안하였다. 특히 Hot spot과 설계 패턴을 적용하여 이를 실현하는 방법에 대해 정의하였다. 또한 프레임워크의 설계 및 재사용 대상이 되는 마이크로 아키텍처를 문서화시킬 방법을 제안하였다. 이는 설계 패턴의 형태로 제공된다.

제안한 방법을 지원할 수 있는 프레임워크는 6개의 서브시스템으로 구성되어있고 웹을 통한 원격 교육에 있어 필수적으로 필요한 사용자 관리시스템 프레임워크에 대한 정보를 갖고 있다. 제안된 방법에 따라 필요한 지식만을 저장하여 이를 지원하는 환경으로 구현한 것이다.

향후 연구과제로는 설계 패턴을 정의할 수 있는 스크립트 언어의 개발과 이를 지원할 수 있는 서브시스템이 필요하다.

## ※ 참고문헌


[1] Grady Booch, "Designing an Application Framework." Dr.Dobb's Journal 19, No.2, 1994.

[2] James Coplien et al., "Pattern Languages of Program Design", Addison-Wesley, 1995.

[3] James Coplien, "Idioms & Patterns as Architectural Literature", IEEE Software, 14(1), 1997.1

[4] Erich Gamma, et al., "Design Patterns : Abstraction and Reuse of Object-Oriented Design", ECOOP'93, 1993.7

[5] Erich Gamma et al. "Design Patterns - Elements of Reusable Object-Oriented Software", Addison-Wesley, Reading, MA, 1994.

[6] Ralph E. Johnson, Vincent F. Russo. "Reusing Object-Oriented Designs", University of Illinois tech. report UIUCDCS 91-1696, 1991.

[7] Ralph E. Johnson. "Documenting frameworks using patterns", OOPSLA '92 Proceedings, 1992.

[8] R. Lajoie, "Design & Reuse in Object-Oriented Frameworks: Patterns, Contracts & Motifs in Concert", Proceedings of 62nd Congress of the Association Canadienne Francaise pour l'Avancement des Sciences, Montreal, Canada, May 1994.

[9] Stephen J. Mellor, Ralph Johnson, "Why Explore Object Methods, Patterns, and Architectures?", IEEE Software, 14(1), 1997.1

[10] Building Object-Oriented frameworks, Taligent, Inc., 1994.

배 제 민

1991 중앙대학교 전자계산학과 (이학사)
1993 중앙대학교 컴퓨터공학과 (공학석사)
1998 중앙대학교 컴퓨터공학과 (공학박사)
1999-현재 관동대학교 컴퓨터 교육과 조교수
관심분야: 객체지향 멀티미디어, 컴퓨터교육, 컴포넌트 모델링
E-Mail : gemini@mail.kwandong.ac.kr