엔터프라이즈 솔루션 아키텍처 디자인 패턴을 활용한 LMS 프레젠테이션 컴포넌트의 설계 및 구현

김상훈* · 김정인**

1. 서 론

국내 e-Learning 시장 규모는 2002년 1조 7천억에서 2003년 2조 5천억으로 연 평균 32.5%의 고도 성장세를 지속하면서 게임 또는 영화산업 등의 여타 디지털 콘텐츠 산업보다 2~3배 큰 산업 규모를 형성하고 있다. 1990년대 중반부터 기업에 도입되기 시작한 e-Learning은 대학을 비롯한 고등 교육기관으로 확산되고 있으며, 삼성은 1996년, LG는 1998년, SK는 1999년에 각각 사이버 연수원을 설립하여 오프라인 교육을 대체해 가고 있고, 국내 1,000개 기업들 가운데 2002년 기준으로 이미 37%가 자체 직무 교육을 e-Learning으로 수행하고 있다.[1] 대학의 경우는 1999년 제정된 평생 교육법에 근거하여 설립된 15개 사이버 대학을 포함, 이미 전국 376개 대학의 41%에 해당하는 151개 대학이 e-Learning을 도입하여 수행하고 있다.[1]

e-Learning 수요가 급격히 증가함에 따라 관리 시스템들이 다수 등장하였지만, 국내의 상용 e-Learning 관리 시스템(LMS: Learning Management System)은 거의가 Java 플랫폼을 기반으로 동작하고 있다. 2000년 마이크로소프트가 닷넷 플랫폼을 발표하여 널리 보급되었지만 몇몇 제품들을 제외하고는 아직 닷넷 플랫폼을 기반으로 동작하는 LMS는 찾아보기가 힘들다. 앞으로의 IT 시장이 SMB(Small and Medium Business)에 초점이 맞추어 질 것이라는 예측 하에서, 여타 플랫폼에 비하여 상대적으로 가격대 성능비가 뛰어난 닷넷 플랫폼의 시장 점유율이 크게 향상될 수 밖에 없고, 닷넷 플랫폼 기반에서 동작하는 LMS는 가격대 성능비가 뛰어난 제품을 선호하는 IT 시장에서 경쟁력을 갖춘 제품이 될 것이다.

LMS에 접근하는 사용자는 각각의 사용자 권한에 따른 Role을 가지고 접근하게 된다. 일반적인 닷넷의 3-Tier 아키텍처에 따라 설계된 LMS는 Presentation Layer, Business Layer, Data Layer로 구분된 개별 상태를 가지는 컴포넌트들의 유기적인 조합으로 구성되는데[2], 3단계 구조로 이루어진 시스템에서 Presentation Layer를 구성하는 ASP.NET 기반의 Page객체들은 접근하는 사용자의 각각의 Role에 따라 개인화되어 렌더링 된 HTML 페이지를 보여줄 필요를 가지게 된다. 일반적으로 LMS에서는 일반 사용자, 학생, 교수, 교수지원자, 관리자등 다섯 등급의 사용자가 시스템에 접근하여 각자의 역할에 따른 동작을 수행하게 되는데, 시스템은 접근한 사용자의 Role과 상태에 따라 다르게 구성되어 보여지는 각각의 페이지를 생성할 수 있어야 한다.

지금까지 소개된 대부분의 상용 LMS에서는

* 동명정보대학교 부설 정보기술원 선임연구원
** 동명정보대학교 컴퓨터공학과 부교수

각 Role과 사용자의 상태에 따른 수백 개의 페이지를 따로 구성하여 완전히 다른 페이지를 보여주도록 구성되었다. 하지만 SCORM 1.2 표준에 따라 구현된 LMS에서는 각 교수자, 학습자 등으로 구분된 각 사용자의 권한과 상태에 따라 다르게 보이는 페이지가 몇 개의 틀에서 파생될 수 있는 페이지로 정형화 된다.

정형화 된 몇 개의 틀에 따라 작성되는 페이지들은 객체지향적인 관점에서 재 사용가능하게 구현되는 것이 마땅하고[4], 그런 이유로 같은 페이지가 Role이나 접근한 사용자의 상태에 따라 다르게 보이는 다형적 구성이 제공되는 것이 시스템의 설계와 구현에서 좋은 성능을 가져올 수 있다.

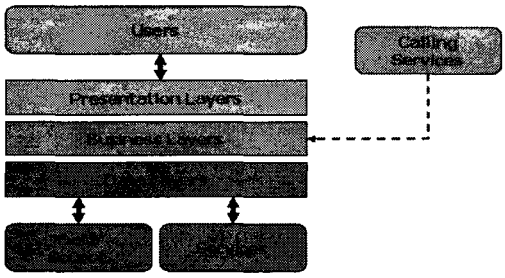
마이크로소프트는 .NET Framework를 사용하여 분산 컴포넌트 기반 솔루션을 구축함에 있어서 아키텍처 수준의 가이드와 디자인 수준의 가이드를 제시해 줄 수 있는 .NET Application Architecture를 공개했다. 이 아키텍처에는 분산 애플리케이션을 설계할 때 그 애플리케이션의 논리적, 물리적 아키텍처뿐만 아니라 기능 구현에 적용할 수 있는 인프라, 애플리케이션의 기능적 수행사항과 관리적 요구사항들을 기반으로 서비스 및 그 서비스를 통합할 수 있는 필요조건에 대한 가이드를 제시하고 있다. Enterprise Solution Pattern이라는 이름으로 불리는 이들 디자인 패턴은 주어진 환경에서 발생하는 문제를 기술하고 이 문제에 대한 솔루션을 제시한다.

본 논문에서는 .NET Application Architecture와 Enterprise Solution Pattern이 LMS의 구현에 어떻게 활용될 수 있을지를 제시한다. 여러 상태를 가지는 사용자가 접근하여 동작을 수행하는 LMS에서 이러한 동작을 중앙 집중화하여 로직을 단순화하고, 전체 애플리케이션이 공통적인 외형과 네비게이션 형태를 가짐으로써 사용성과 브랜드의 인식을 향상시킬 수 있는 방법에 관해 논한다.

본 논문은 다음과 같이 구성되었다. 2장에서 .NET Application Architecture에서 프레젠테이션 계층을 구성하는 컴포넌트의 설계 방법에 대해 알아보고 3장에서는 닷넷 애플리케이션 아키텍처에서 프레젠테이션 계층 컴포넌트를 구성하는 방법에 대해서 알아본다. 4장에서는 프레젠테이션 컴포넌트 설계 방법을 기반으로 하여 뷰와 모델, 컨트롤러를 명확히 구분하여 설계할 수 있는 MVC(Model-View-Controller)패턴의 닷넷 구현에 대해서 설명하고, 5장에서는 닷넷에서 구현 가능한 MVC 패턴을 기반으로 구성된 Page Controller 패턴을 ASP.NET 환경에서 구현한다. 6장에서 Page Controller 패턴을 사용하여 사용자의 Role에 따라 다형적으로 동작하는 LMS 프레젠테이션 컴포넌트를 구현하고, 끝으로 7장에서 결론을 맺는다.

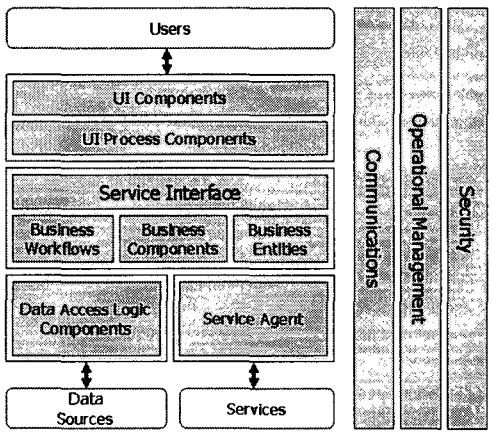
2. .NET Application Architecture

애플리케이션은 프레젠테이션, 비즈니스, 그리고 데이터 서비스를 제공하는 컴포넌트로 구분하여 설계해야 한다는 견해는 널리 알려져 있다. 유사한 형태의 기능을 수행하는 컴포넌트를 같은 레이어로 그룹화하고 그 레이어들을 하나의 스택 형식으로 체계화 할 수 있다. 그리하여, 특정 레이어의 상위에 존재하는 컴포넌트가 그 레이어에서 제공하는 서비스를 이용할 수 있고, 특정 컴포넌트는 자신의 레이어나 다른 레이어에 존재하는 컴포넌트의 기능을 이용할 수 있다. <그림 1>은 레이어가 포함된 애플리케이션의 약식 구조도를 나타낸다. 레이어들은 애플리케이션 혹은 서비스를 구성하는 소프트웨어 컴포넌트의 논리적인 그룹에 지나지 않는다. 레이어는 컴포넌트가 수행하는 서로 다른 종류의 작업을 구분하고 재 사용성이 높은 솔루션 설계에 도움이 되는 개념이다. 닷넷의 애플리케이션 아키텍처는 <그림 1>과 같은 전형적인 3-Tier 구조를 가지게 된다.



〈그림 1〉 컴포넌트의 역할에 따른 계층분리

각각의 논리적 레이어는 몇 개의 컴포넌트 유형들로 그룹화된 하위 레이어를 포함할 수 있다. 하위 레이어는 유사한 작업을 수행하는 그룹으로 볼 수 있고, 일반적으로 솔루션에 존재하는 여러 컴포넌트의 종류를 식별하게 되면, 애플리케이션 혹은 서비스의 의미 있는 구조도를 작성하여 설계 청사진으로 사용할 수 있게 된다. <그림 2>는 대부분의 분산 솔루션에서 발견되는 일반적인 컴포넌트의 유형을 나타낸다.



〈그림 2〉 일반적인 컴포넌트의 유형

이런 3-Tier Architecture는 응용 프로그램의 구성에 있어 많은 탄력성을 제공해 줄 수 있게 되는데 페이지의 변경을 감시하는 Observer나 UI 컴포넌트들의 상호작용을 관리해주는 Mediator를 구성한다거나 하는 등의 설계 개선을 꾀할 수 있는 유연한 구조의 재사용성과 교체 가능성, 유지 보수성이 뛰어난 응용 프로그램을 만들 수 있게 된다. 따라서, 응용 프로그램은 다음 사항들을 만족하도록 설계되어야 한다.

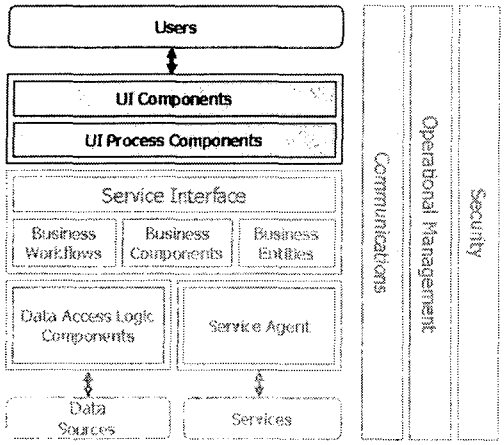
- 수직적 관계의 모든 컴포넌트들은 느슨한 결합관계를 가져야 한다.
- 비즈니스 컴포넌트는 자신을 보여줄 수 있는 공통된 방법을 가지고 있어야 한다.
- 데이터베이스 연결은 일관되며 공통된 규칙을 지킬 수 있도록 집중화 되어야 한다.

본 연구에서는, Presentation Tier 계층에 존재하는 컴포넌트를 Page Controller 패턴을 활용하여 구성함으로써 사용자의 Role이나 상태에 따라 다형적으로 동작하는 페이지의 구현을 위하여 프레젠테이션 계층에 존재하는 두 컴포넌트 계층인 UI Components 계층과 UI Process 컴포넌트 계층을 위주로 시스템을 설계하고 구현하였다.

3. 프레젠테이션 계층 컴포넌트 설계

프레젠테이션 계층에는 사용자 인터페이스 컴포넌트 (UI 컴포넌트)와 사용자 프로세스 컴포넌트, 두 종류의 컴포넌트가 존재한다.

사용자 인터페이스 컴포넌트는 사용자와 상호 작용하는 객체들이 모인 물리적 개체 집합이다. .NET Framework 기반 개발에서는 이 계층에 System.Windows.Forms 네임스페이스의 개체들과 System.Web.UI 네임스페이스의 개체들이 존재한다. 웹 기반 응용 프로그램이라면 TextBox 컨트롤, Button 컨트롤, DropDownList 컨트롤, DataGrid 컨트롤등 ASP.NET 페이지에 직접 붙여넣는 컨트롤들이 모두 이 범주에 속한다. <그림 2>에서 프리젠테이션 계층 컴포넌트를 구성하는 부분은 <그림 3>과 같다.



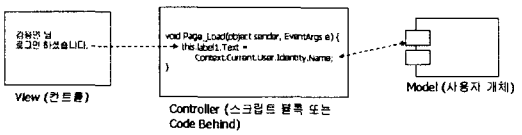
〈그림 3〉 프리젠테이션 계층 컴포넌트

사용자 인터페이스는 크게 세 가지 역할을 수행하여야 한다.

첫 번째 역할은 사용자에게 데이터를 보여주는 것이다. 예를 들면, 현재 로그인한 사용자의 이름을 보여주기 위하여 Label 컨트롤을 사용하거나 상품의 목록을 보여주기 위해서 DataGrid 컨트롤을 사용하는 등이다. 여기서 주의하여야 할 점은 Label 컨트롤에 로그인한 사용자의 이름을 보여주기 위해서 Label 컨트롤을 수정하여 Label 컨트롤이 직접 HttpContext 개체나 Cookie등에 액세스 하지 않는다는 것이다. Label 컨트롤이나 DataGrid 컨트롤은 단지 어떤 데이터를 받아들여서 표시할 수 있는 역할만을 하여야 한다.[3]

두 번째 역할은 사용자의 입력을 받아들이고 입력된 데이터를 하위 컴포넌트에 전달하는 역할을 한다. 입력된 데이터를 하위 컴포넌트에 전달하기 위해서는 함수를 호출하여야 하는데, 사용자의 이벤트를 받아들이는데 가장 많이 사용되는 버튼 컨트롤이 이러한 함수를 직접 호출하는 형태여서는 안된다. 사용자의 입력을 받아들이는 사용자 인터페이스 컴포넌트는 단지 속성과 이벤트만을 가지고 있을 뿐, 직접적인 함수의 호출 로직을 가지는 것은 금물이다.[3]

세 번째 역할은, 사용자가 입력된 데이터를 검증하는 역할이다. 사용자가 입력한 데이터가 정수형인지, 날짜인지 등의 애플리케이션 내에서 유효한 값인지를 검증할 수 있는 클라이언트 수준에서의 로직이 필요하다. 종합해 보면, 로그인한 사용자의 이름을 웹 페이지에 보여주는 사용자 인터페이스는 <그림 4>와 같은 형태로 작성되어야 한다.[3] 이는 MVC 패턴[5]의 닷넷 구현이라 할 수 있다.



〈그림 4〉 사용자 인터페이스 설계

본 연구에서는 사용자 인터페이스 컴포넌트에서 View와 Controller의 구성에 따라 구현할 수 있는 Page Controller 패턴을 사용하여 사용자의 Role 혹은 상태에 따라 다르게 렌더링 되는 페이지를 구현하였다.

4. MVC 패턴의 닷넷 구현

MVC(Model-View-Controller) 패턴은 사용자의 입력을 토대로 도메인, 프레젠테이션, 액션을 각각 다음과 같이 3개의 클래스로 분류하는 기법이다.[5]

• Model

애플리케이션 도메인의 행위와 데이터를 관리하고 일반적으로 뷰의 상태에 대한 정보 요청에 응답하며 일반적으로 컨트롤러의 상태를 변경하기 위한 명령에 응답한다.

• View

정보의 표시를 관리한다.

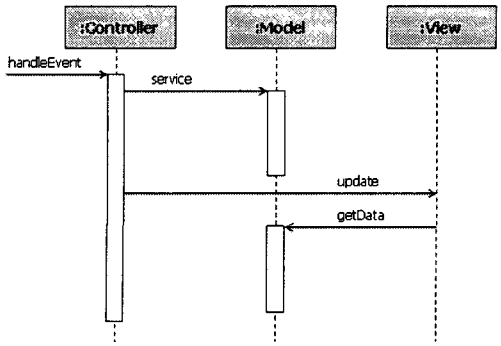
• Controller

사용자가 입력한 마우스와 키보드의 입력을 해석하여 적당한 시기에 모델 또는 뷰에 변경할 것을 알린다.

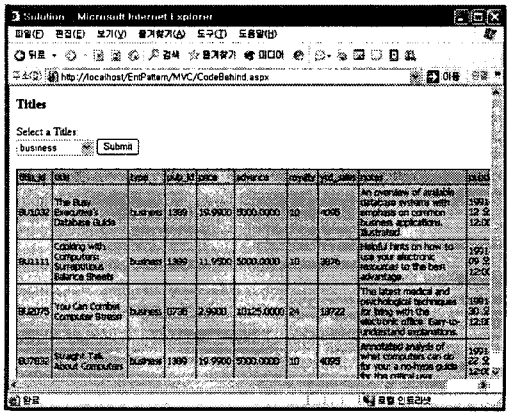
MVC 패턴은 사용자 인터페이스 로직을 비즈니스 로직과 구분하기 위한 가장 기초적인 디자인 패턴이다. MVC 패턴에서는 뷰와 컨트롤러 모두 모델에 의존하지만 모델은 뷰와 컨트롤러 모두에 의존하지 않는다는 것이 가장 중요하다. 이러한 구분을 사용하여 모델은 시각적인 프레젠테이션과 별도로 작성되고 테스트 할 수 있다. 웹 애플리케이션에서는 뷰(브라우저)와 컨트롤러(HTTP 요청을 처리하는 클라이언트 사이트 컴포넌트)간의 구분을 매우 잘 정의할 수 있다.

Steave Burbeck는 수동적인 모델과 능동적인 모델이라는 MVC의 두 가지 변형을 기술하였다[6]. 수동적인 모델은 하나의 컨트롤러가 모델을 배타적으로 다룰 때 사용하게 된다. 컨트롤러는 모델을 수정하고 뷰에게 모델이 변경되어서 반드시 새로 고쳐야 한다는 사실을 알린다. 수동적인 모델인 경우 모델은 뷰, 컨트롤러와 완전히 독립되어 HTTP 프로토콜처럼 모델이 자신의 상태가 변경되었음을 보고할 수 있는 방법이 없다. 웹 애플리케이션에서는 오직 사용자가 명시적으로 새로 고침을 요청했을 때에만 서버가 변경된 데이터에 대해서 응답하게 된다. <그림 5>는 수동적인 MVC 모델의 행위를 시퀀스 다이어그램으로 나타낸 것이다.

수동적인 MVC 모델의 닷넷 구현은 ASP.NET 프레임워크가 제공하는 ASP.NET 페이지의 코드 비하인드 기능을 사용할 수 있다. <그림 6>는 MVC 모델의 닷넷 구현 예제에서 볼 수 있는 뷰(View)이다.



〈그림 5〉 수동적인 MVC 모델



〈그림 6〉 뷰(View)

<그림 7>은 뷰를 표현하기 <그림 6>에서 보여지는 뷰를 구성하는 위한 소스코드를 나타낸 것이다.

```
1 <%@ Page language="c#" Src="CodeBehind.aspx.cs"
2     AutoEventWireup="false"
3     Inherits="CodeBehind1.CodeBehind" %>
4 <HTML>
5     <HEAD>
6         <title>Solution</title>
7     </HEAD>
8     <body>
9         <form id="Solution" method="post" Runat="Server">
10            <h3>Titles</h3>
11            Select a Title:<br>
12            <asp:DropDownList id="titlesSelect" runat="Server" />
13            <asp:Button Runat="Server" Text="Submit" id="Button1" name="Button1" />
14            <p>
15                <asp:DataGrid id="myDataGrid" Runat="Server" width="700"
16                    backcolor="#cccff" bordercolor="black"
17                    showfooter="false" cellpadding="3" cellspacing="0"
18                    font-name="tahoma" font-size="9pt"
19                    headerstyle-backcolor="#aaaadd"
20                    enableviewstate="false" />
21        </form>
22        </P>
23    </body>
24 </HTML>
25
```

〈그림 7〉 뷰 소스 코드

<그림 8>은 모델을 기술하고 데이터베이스에만 의존하는 모델 코드이다. <그림 8>에서 보여지는 파일은 전체 구조에서 데이터베이스에 의존하는 유일한 파일이다. 이 파일은 데이터베이스에 액세스하여 단일 테이블이나 뷰를 액세스 하기 위한 모든 SQL 코드를 가진다. 코드들은 데이터베이스와 상호 작용하기 위한 메소드를 호출한다.

```
using System;
using System.Data;
using System.Data.SqlClient;

public class Model {
    public static DataSet GetTypes() {
        string sqlQuery = "select distinct type from titles";

        SqlConnection myConnection =
            new SqlConnection("server=(local);database=pubs;UID=sa;PWD=");

        SqlDataAdapter myCommand =
            new SqlDataAdapter(sqlQuery, myConnection);

        DataSet ds = new DataSet();
        myCommand.Fill(ds, "titles");

        return ds;
    }

    public static DataSet GetTitles(string typeID) {
        string sqlQuery =
            String.Format("Select * from Titles where type = '{0}' order by title_id", typeID);

        SqlConnection myConnection =
            new SqlConnection("server=(local);database=pubs;UID=sa;PWD=");

        SqlDataAdapter myCommand =
            new SqlDataAdapter(sqlQuery, myConnection);

        DataSet ds = new DataSet();
        myCommand.Fill(ds, "titles");

        return ds;
    }
}
```

〈그림 8〉 모델 소스코드

<그림 9>는 모델 코드가 페이지에 존재하는 데이터 컨트롤러를 채용하고 컨트롤러가 뷰에서 받은 데이터를 전달하는 이벤트를 특정한 행위 메소드에 전달하기 위하여 코드 비하인드 기능을 사용한다. 컨트롤러 코드는 뷰 코드와 마찬가지로 모델에 영향을 받지 않는다. <그림 9>의 코드는 데이터베이스로부터 데이터를 얻는 방법에는 의존하지 않는다. 단지, 뷰에 존재하는 사용자가 입력한 데이터와 사용자의 이벤트에 대한 컨트롤만을 위한 코드가 존재한다.

5. MVC 패턴 기반의 Page Controller 패턴 구현

MVC 패턴은 모델과 뷰 간의 구분에 중점을

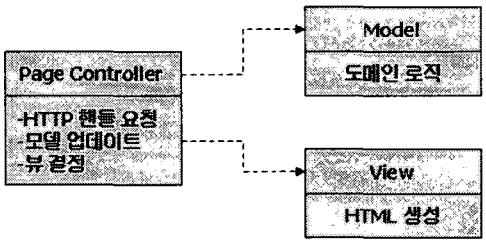
```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace CodeBehind1
{
    public class CodeBehind : System.Web.UI.Page
    {
        protected System.Web.UI.WebControls.DropDownList titlesSelect;
        protected System.Web.UI.WebControls.Button Button1;
        protected System.Web.UI.WebControls.DataGrid myDataGrid;

        private void Page_Load(object sender, System.EventArgs e)
        {
            if (!Page.IsPostBack) {
                DataSet ds = Model.GetTypes();

                titlesSelect.DataSource = ds;
                titlesSelect.DataTextField = "type";
                titlesSelect.DataValueField = "type";
                titlesSelect.DataBind();
            }
        }

        override protected void OnInit(EventArgs e)
        {
            InitializeComponent();
            base.OnInit(e);
        }

        private void InitializeComponent()
        {
            this.Button1.Click += new System.EventHandler(this.SubmitBtn_Click);
            this.Load += new System.EventHandler(this.Page_Load);

        }

        public void SubmitBtn_Click(object sender, System.EventArgs e)
        {
            DataSet ds = Model.GetTitles((string)titlesSelect.SelectedItem.Value);

            myDataGrid.DataSource = ds;
            myDataGrid.DataBind();
        }
    }
}
```

〈그림 9〉 컨트롤러 소스코드

두게 된다. 상대적으로 컨트롤러는 단순한 이벤트와 메소드의 매핑이외의 다른 관심을 두지 않게 된다. 하지만 대부분의 씬 클라이언트 애플리케이션에서는 컨트롤러가 서버 사이드에 존재하게 되므로 뷰와 컨트롤러는 본질적으로 구분된다. ASP.NET에서 동적인 웹 페이지를 렌더링 하는 과정은 사용자 인증을 검증하고, 쿼리 문자열이나 폼 필드로부터 페이지 매개 변수를 추출하고 세션 정보를 수집하는 등의 매우 유사한 과정을 보인다. 이러한 작업은 엄청난 양의 코드 복사를 필요로 한다[6]. ASP와 ASP.NET같은 스크립트 형식의 서버페이지는 생성하기 쉽지만, LMS와 같이 많은 페이지를 가지는 애플리케이션에서는 여러 가지 단점이 생긴다.

사용자 인터페이스 코드를 작성하고 테스트 하는 작업은 시간이 많이 걸리며 지루한 작업이다.

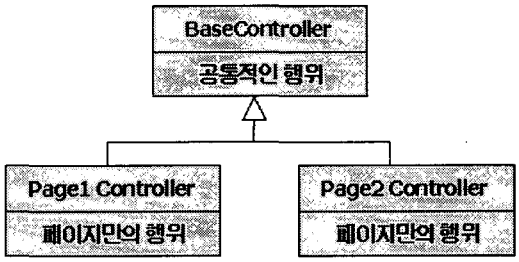
Page Controller 패턴은 이러한 지루한 작업을 줄여줄 뿐만 아니라, 사용자에 대한 Role 단위 또는 상태 단위로 보이는 페이지의 동작을 중앙 집중화 함으로써 많은 페이지를 가지는 대규모의 애플리케이션 작성에 있어 개발 작업에서의 부하를 줄여주며, 여러 페이지에 걸쳐 프레젠테이션 코드의 재사용성을 향상시켜줄 수 있는 컴포넌트의 작성을 가능하게 할 수 있다. <그림 10>은 페이지 컨트롤러의 구조이다.



〈그림 10〉 페이지 컨트롤러의 구조

페이지 컨트롤러는 페이지 요청을 받고 관련된 데이터를 추출하며, 모델을 업데이트 하고 뷰에 요청을 전달하는 역할을 한다. 이러한 기본적인 형식에서 웹 애플리케이션에 있는 모든 링크에 대한 컨트롤러를 생성할 수 있다. 이 방법은 한 번에 하나의 행위에 참여하게 되므로, 컨트롤러를 간단하게 유지할 수 있게 된다. 각 웹 페이지에 대한 별도의 컨트롤러를 생성하면 같은 동작을 하는 코드를 복사하게 되고 개발에 오류를 일으킬 수 있는 여지가 남게 된다.

이러한 오류를 줄여주고 개발 사이클을 줄여줄 수 있는 Page Controller를 닷넷 기반에서 구현하기 위해서는, 입력 변수의 검증과 같은 공통적인 코드나 모든 페이지가 공통적으로 유지해야 하는 공통적인 외형을 포함하는 BaseController 클래스를 생성해야 한다. Page Controller는 이러한 공통적인 기능 이외에도 컨트롤러가 공통적인 기능을 수행하기 위해서 사용할 수 있는 여러 가지 헬퍼 클래스의 집합을 정의할 수도 있다. 이렇게 생성된 BaseController 클래스를 각각의 개별적인 페이지 컨트롤러가 상속하여 공통적인 기능을 상속받거나 헬퍼 클래스를 사용할 수 있게 된다. <그림 11>은 이렇게 설계된 Page Controller의 클래스 다이어그램이다.



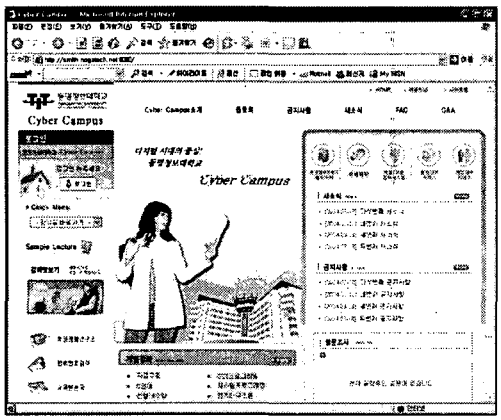
〈그림 11〉 상속성을 이용한 코드 재사용

웹 애플리케이션에서 페이지 컨트롤러를 사용하는 것은 대부분의 웹 애플리케이션 프레임워크가 페이지 컨트롤러에 대한 기본적인 구현을 제공하는 것과 같이 공통적으로 필요하다. ASP 프레임워크는 서버 페이지의 폼에 컨트롤러를 포함한다. ASP와 JSP,. PHP같은 프레임워크는 뷰 관련 코드와 컨트롤러 관련 코드가 혼합되기 쉽고, 컨트롤러 로직을 적절하게 구분하기가 어려워진다. 그런 이유로, Page Controller 방법이 좋지 않은 방법이라는 여러 가지 활용 결과가 있으나, ASP.NET 프레임워크에서는 Page Controller를 구성할 수 있는 훌륭한 메커니즘을 제공한다. ASP.NET 프레임워크는 코드 비하인드라고 불리는 컨트롤러 클래스를 제공하므로 이를 이용하면 완벽하게 실행 가능한 설계적인 선택사항이 될 수 있다.

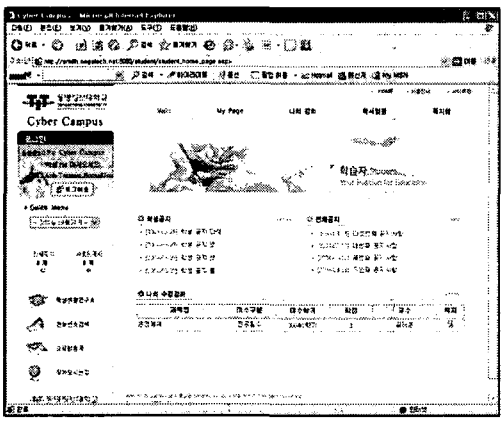
6. LMS에서의 구현

LMS에 접근하는 사용자는 그 등급에 따른 여

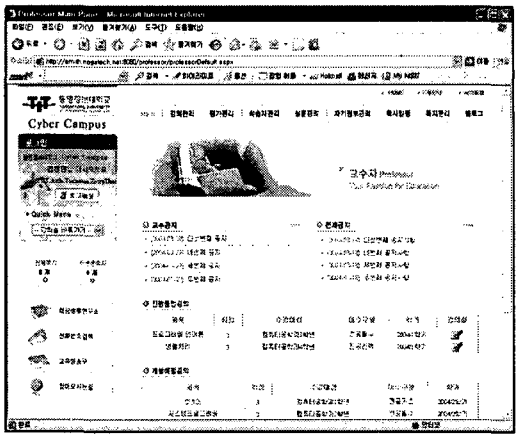
러 가지의 분류와 사용자의 상태에 따른 여러 가지 분류로 나눌 수 있다. 사용자의 등급은 크게 교수자와 학습자, 관리자 세 등급으로 나뉠 수 있는데, 시스템은 이들 사용자들이 로그인 하였을때 로그인한 사용자의 등급에 따라 각자 다른 페이지를 보이도록 하여야 할 필요가 있다. <그림 12>와 <그림 13>, <그림 14>는 본 연구에서 구현한 LMS의 일반 사용자, 학습자, 교수자 페이지이다. 세 페이지에서 알 수 있듯이 모든 페이지들은 비슷한 레이아웃을 보여주지만 접근한 사용자의 Role에 따라 다른 동작을 하고 있음을 알 수 있다. 시스템에 존재하는 모든 페이지들은 같은 레이아



〈그림 12〉 일반 사용자 페이지



〈그림 13〉 학습자 페이지



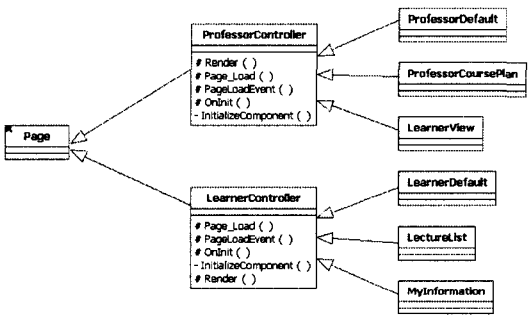
〈그림 14〉 교수자 페이지

웃을 보여줌으로써 솔루션의 브랜드 성을 높일 수 있을 뿐만 아니라 모든 사용자에 대한 일관적인 접근을 구성하게 할 수 있다. 연구에서 구현된 학습관리 시스템은 122가지의 요구사항에 따라 80여 페이지가 작성되었고, 학습자가 사용하는 학습 객체는 사용자의 마우스 또는 키보드의 이벤트에 따라 상태가 변화하며, 상태의 변화를 뷰에 보여주게 되므로 MVC 패턴에 따라 프레테이션 컴포넌트 들을 구성하는 것이 바람직하다.

개발한 시스템의 기능 중, 학습 관리 시스템에서 구현된 80여 페이지들은 페이지의 오른쪽 하단에 보이는 학습 객체의 뷰를 제외하고는 거의 동일한 형태를 보여준다. 학습자뿐만 아니라, 교수자와 관리자 등 시스템에 접근하는 모든 페이지는 이러한 형태를 가지게 된다.

MVC 패턴에서는 사용자에게 보여 지는 뷰와 뷰를 구성하는 데이터가 완전히 독립적으로 동작하게 되고, 같은 뷰를 보여주는 사용자 인터페이스를 80여개의 페이지에 모두 코딩한다는 것은 지루한 작업이 될 뿐만 아니라 간단한 HTML 태그 사용의 실수 등으로 인해서 일그러진 페이지의 레이아웃을 수정해야하는 등의 작업으로 인한 개발 지연을 가져올 수 있다. 이런 경우, Page

Controller 패턴을 사용하여 학습자 또는 교수자의 모든 페이지에 적용되는 공통적인 뷰를 처리할 수 있는 BaseController 클래스를 작성하고, BaseController 클래스를 상속하는 각각의 페이지들이 개별적인 고유한 동작을 구현하도록 함으로써 이러한 단점을 극복할 수 있게 된다. <그림 15>는 Page Controller를 기반으로 구현한 LMS의 클래스 다이어그램이다.



〈그림 15〉 클래스 다이어그램

<그림 15>의 클래스 다이어그램에서와 같이, 기본 컨트롤러는 닷넷 프레임워크의 System.Web.UI 네임 스페이스의 Page 클래스에서 상속된다. 이 클래스들은 교수자 또는 학습자의 페이지에서 공통적으로 보여 지는 페이지들을 렌더링하게 작성하고, 각각의 페이지들은 기본 컨트롤러 클래스를 상속하여 개별적인 동작들을 구현하게 된다.

본 연구에서는 Page Controller 패턴을 두 가지 방법으로 사용하였는데, 하나는 기본 컨트롤러 클래스에서 작성된 Page_Load 이벤트 핸들러 메소드를 GOF(Gang of Four)의 디자인 패턴 중 Template Method 패턴[4]의 Template Method로 동작하도록 설계 한 것이고, 두 번째 방법은 Page 클래스의 CreateChildControls 메소드를 오버라이드 하고 Render 메소드를 Template Method로 사용하여 공통적인 컨트롤들을 페이지에 포함할 수 있도록 하였다.

Page_Load 이벤트 핸들러 메소드를 Template Method로 사용하기 위해, <그림 16>과 같은 기본 컨트롤러 클래스의 코드를 작성하였다.

```
using System.Web;
using System.Web.UI;

namespace LMS.ProfressorSupport
{
    public class ProfessorController : Page
    {
        protected void Page_Load(object sender, EventArgs e) {
            // 기본 컨트롤러에서 기본으로 구현되는 동작
            this.PageLoadEvent();
            // 기본 컨트롤러에서 기본으로 구현되는 동작
        }

        protected virtual void PageLoadEvent(object sender, EventArgs e) { }

        protected override void OnInit(System.EventArgs e)
        private void InitializeComponent()
        protected override void Render(HtmlTextWriter writer)
    }
}
```

〈그림 16〉 기본 컨트롤러 클래스의 구현 코드

기본 컨트롤러 클래스를 상속하는 클래스들은 PageLoadEvent 메소드를 오버라이eld하여 각각의 페이지에 필요한 동작을 구현한다. 기본 컨트롤러 클래스의 PageLoadEvent 메소드는 구현이 없고, 이를 Page_Load 이벤트 핸들러 메소드에서 호출하므로 기본 컨트롤러 클래스를 상속하는 클래스에서는 Page_Load 이벤트 핸들러 메소드를 구현하지 않고 PageLoadEvent 메소드를 오버라이딩하여 자신에게 필요한 동작을 호출한다.

```
using LMS.BLL.ProfressorSupport;

namespace ClassLibrary6
{
    public class ProfessorCoursePlan : LMS.ProfressorSupport.ProfessorController
    {
        private DataGrid coursePlanGrid;

        protected override void PageLoadEvent(object sender, EventArgs e)
        {
            CourseList list = new CourseList();
            this.coursePlanGrid.DataSource = list.Courses;
            this.coursePlanGrid.DataBind();
        }
    }
}
```

〈그림 17〉 페이지 컨트롤러의 구현

첫 번째 구현 방법은 기본 컨트롤러에서 특정 사용자 이벤트가 발생하지 않는 공통된 구성을 가지는 페이지들의 구현들에서 사용하였다. 이런 페이지에서는 각 사용자의 Role 별로 공통적으로

보여 지는 뷰 컴포넌트에서 SSI (Server Side Include)를 포함하고, 페이지에 종속되는 기능들만을 Template Method로 구현하는데 적당하다. 하지만 공통적으로 보여야 되는 뷰 컴포넌트에서 단순 링크가 아닌 서버 사이드 사용자 이벤트가 발생되는 경우에는 사용하기 곤란하다. 이러한 경우에는 Render 메소드를 오버라이딩하여 Template 메소드로 사용할 수 있다.

<그림 12>에서 보이는 것과 같이, 모든 페이지는 로그인 / 로그 아웃을 담당하는 버튼 컨트롤을 가지고 있다. 이 버튼 컨트롤은 UI Process Component와 통신하여 사용자의 상태를 얻어내어 페이지에 출력하고, 사용자의 마우스 입력에 따라 애플리케이션에 인증을 증명하고 해제하는 역할을 담당한다. 본 논문에서는 공통적인 이벤트를 포함하고, 이벤트를 담당하는 서버 측 컨트롤을 공통적으로 사용하기 위해 기본 컨트롤러 클래스에서 컨트롤을 생성하고 이벤트를 등록하여 기본 클래스를 상속하는 하위 클래스가 코드를 복사하지 않아도 되도록 구성하였다.

```
namespace LMS.LearnerSupport
{
    public class LearnerController : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e) {}
        protected virtual void PageLoadEvent(object sender, EventArgs e)
        protected Button loginButton;

        protected override void OnInit(System.EventArgs e)
        private void InitializeComponent()

        public void OnLoginClick(object sender, EventArgs e)
        {
            LMSLogin login = new LMSLogin();
            login.Login();
        }

        protected void CreateHeader(HtmlTextWriter writer)
        protected void CreateButton(HtmlTextWriter writer)

        protected override void CreateChildControls()
        {
            this.Controls.Add(this.loginButton);
            this.loginButton.Click += new EventHandler(this.OnLoginClick);
            base.CreateChildControls ();
        }

        protected override void Render(HtmlTextWriter writer)
        {
            this.CreateHeader(writer);
            base.Render (writer);
            this.CreateButton();
        }
    }
}
```

〈그림 18〉 Render 메소드 구현

이러한 방법은, 기본 컨트롤러가 페이지의 렌더링을 담당함으로써 개발자가 뷰 컴포넌트의 작성 시에 페이지에 공통적인 서버 사이드 이벤트를 일일이 코딩하는 수고를 덜어주고, 일관된 페이지의 레이아웃을 유지하도록 해준다. 특히 LMS와 같이 접근하는 사용자의 역할이 명확히 구분되는 경우, 빠른 페이지의 작성이 원활히 이루어지고 시스템의 업그레이드로 인한 다른 페이지의 추가 시에도 코드의 수정이나 복사 없이 기본 컨트롤러의 상속만으로 구현할 수 있다는 장점을 가진다.

LMS에서의 Page Controller 구현은 닷넷 프레임워크에서 기본적으로 제공하며 애플리케이션에 특화된 행동을 컨트롤러가 노출하는 이벤트에 연결함으로써 쉽게 확장할 수 있고, 코드 비하인드 기능을 사용함으로써 모델과 뷰 코드로부터 컨트롤러에 특화된 코드를 쉽게 분리할 수 있다는 장점을 가지게 된다.

7. 결론 및 향후 연구과제

본 논문에서는 LMS를 구현함에 있어 확연하게 구분되는 접근 사용자의 역할에 따라 다르게 보여지는 페이지를, .NET Enterprise Solution Patterns에서 권장하는 Page Controller 패턴을 사용하여, 기본 컨트롤러를 구성하고 각 페이지들이 기본 컨트롤러를 상속하게 함으로써 뷰와 컨트롤러, 모델의 코드를 쉽게 분리해내고, 이를 이용하여 다형적으로 동작하는 컨트롤러를 작성하여 애플리케이션의 확장과 구현의 편리성을 도모하였다. LMS의 구현에 본 기법을 사용함으로써 다음과 같은 효과들을 기대할 수 있다.

1) ASP.NET에서 기본 제공하는 컨트롤러 기법을 사용함으로써 닷넷 프레임워크의 성능과 높은 생산성의 장점들을을 기대할 수 있다.

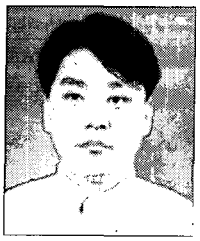
2) ASP.NET의 페이지 라이프 사이클에 대한 공통적인 단계를 포함함으로써 구현되는 모든 페이지가 공통적인 단계로 동작하도록 구현하여 개발의 생산성을 향상할 수 있다.
3) LMS의 모든 페이지가 공통적인 외형과 네비게이션을 사용함으로써 웹 애플리케이션의 사용성을 향상할 수 있다.
4) 애플리케이션 아키텍처의 구현을 강제화함으로써 프로젝트의 시작시 정의한 애플리케이션의 구조 변경이 일어나지 않도록 한다.
5) 기본 컨르롤러를 작성하고, 각 페이지들이 작성된 컨트롤러를 상속하게 하는 기법을 사용하여 객체지향적인 확장이 용이하다.
6) 코드의 재사용이 아닌 바이너리의 재사용을 기대할 수 있어 코드를 수정하지 않고 객체의 추가만으로도 확장 및 변경이 가능하도록 구성할 수 있다.

하지만 객체지향의 상속을 사용하여 구현한 관계로 LMS에서 좀 더 자세한 사용자별 구현을 필요로 할 때 깊은 상속트리를 가지게 된다. 깊은 상속성은 프로그램의 확장과 변경에 있어 융통성의 저하를 가져올 수 있다. 또한, 이러한 구현 방식은 웹 애플리케이션 프레임워크에 의존하게 되어 독립적으로 테스트하기가 어려워진다. 이는 중앙 집중적인 컨트롤러의 제어를 위한 애플리케이션 구성에서 활성화 되는 페이지의 개발적인 접근 가능성 측면에서 항상 나타나는 어려움이라 할 수 있다. 이러한 어려움은 본 논문에서 수행한 애플리케이션 아키텍처의 구조를 활용한 LMS 전용 프레임워크의 구성으로 해결할 수 있으리라 생각한다.

향후, 모든 페이지의 컨트롤을 중앙 집중제어하기 위한 Front Controller의 활용 및 그러한 패턴들을 구현하여 LMS에 특화된 전용 프레임워크의 구성과, 작성된 LMS를 오류 가능성을 줄이면서 쉽게 배포할 수 있는 전용 배포 도구 및 배포 패턴의 연구가 보완되어야 할 것으로 본다.

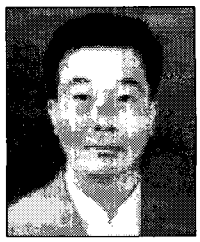
참 고 문 헌

[1] 이석용외, e-Learning 전개에 있어서 기업과 대학의 차이점에 관한연구, 한국경영정보학회 2004춘계학술대회, pp.30-3.
[2] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpatterns/html/espp atternsforbuildingenterprisesolutions.asp
[3] 김상훈, "엔터프라이즈 환경을 위한 닷넷 아키텍처의 변화", 월간마이크로소프트웨어, 2005. 2.
[4] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides "Design Patterns : Elements of Reusable Object-Oriented Software", Addison Wesley, 1885
[5] Burbeck, Steve "Application Programming in Smalltalk-80 : How to use Model-View-Controller (MVC)", University of Illinois in Urbana-Cham-paign (UIUC) Smalltalk Archive
[6] 마이크로소프트, 엔터프라이즈 솔루션 아키텍처 디자인 패턴, 정보문화사, 2003.



김 상 훈

- 2001년 동명정보대학교 컴퓨터공학과(공학사)
- 2003년 동명정보대학원 컴퓨터공학과(공학석사)
- 2001년~현재 동명정보대학교 정보기술원 선임연구원
- 관심분야 : Database, Generic, Application Architecture
- E-mail : deaperado@tit.ac.kr



김 정 인

- 1986년 계명대학교 통계학과(이학사)
- 1993년 게이오대학 컴퓨터과학전공(공학사)
- 1996년 게이오대학 컴퓨터과학전공(공학박사)
- 1996년~1998년 포항공과대학교 정보통신연구소 연구원
- 1998년~동명정보대학교 공과대학 컴퓨터공학과 부교수
- 관심분야 : e-Learning, 지식공학, 자연어처리, 정보검색
- E-mail : jikim@tit.ac.kr