J2EE Enterprise System의 웹 애플리케이션 프레임워크 설계

A Design on Web Application Framework of J2EE Enterprise System

이세환, 남택진, 김봉현
한밭대학교 컴퓨터공학과

Lee Se-Hwan, Nam Taek-Jin, Kim Bong-Hyun
Dept. of Computer Engineering, Hanbat National University

요약

웹 애플리케이션은 복잡한 비즈니스 소프트웨어 및 온라인 서비스의 바탕이 되는 플랫폼으로서 더욱 많이 사용되고 있다. 초기 JSP 분야에는 모델 1과 모델 2라는 두 가지 아키텍처 설계 용어가 등장한다. 이들 두 용어는 현재 JSP 분야 문서에는 언급되고 있지는 않지만, 현재 웹 어플리케이션의 아키텍처 설계에는 광범위하게 적용되고 있다. 본 논문에서는 웹 애플리케이션 프레임워크 정의 및 프레임워크 패턴, 설계시 고려사항등을 토대로 컴포넌트가 실행되는 기반을 구성하는 프레임워크를 설계한다.

Abstract

Web application is more plentifully used as the platform based on business software which is complicated and online services. Initially to JSP fields of appears two architecture design terms called model 1 and model 2. These two term are not referred currently to JSP fields of documents but, current be applicable to extensively in architecture design of the web application. In this paper, definition of web application framework and framework pattern, design then design framework the component which it spreads out is executed composition based on consideration facts.

I. 서 론

MVC는 디자인 패턴 중의 하나로 프로그래밍 세계에서는 전혀 새로울 것이 없다. Smalltalk 등과 같은 객체형 프로그래밍의 초기 시대부터 Java Swing 컴포넌트 집합의 기초를 제공하는 최근까지 디자인 패턴으로서의 MVC 사용과 관련된 많은 레퍼런스들이 존재하고 있다. 이와 같이 MVC가 다시 주목을 받게 된 이유는 그것의 패턴이 웹 기반 애플리케이션 구축 시 발생하는 기본 문제들 중의 대다수를 해결하는데 적합하다는 것을 깨달았기 때문이다. 일반적인 데이터베이스-중심 애플리케이션들과 특정한 웹 기반의 씬-클라이언트 애플리케이션들을 살펴보면 애플리케이션이 여러 가지 구분되는 작업들을 수행해야 한다는 것을 발견할 수 있다.

- 데이타 액세스
- 비즈니스 로직 구현
- 사용자 인터페이스 표시(데이터 프리젠테이션)
- 사용자 상호 작용
- 애플리케이션(페이지) 플로우

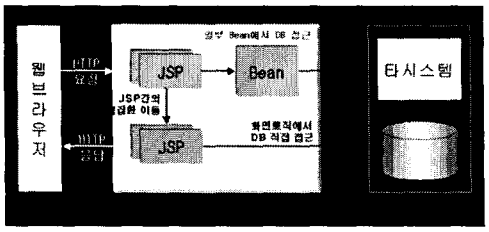
MVC 아키텍처 또는 패턴은 사용자가 사용자 인터페이스를 재작성하지 않고 다른 데이터 소스에서 애플리케이션으로 용이하게 플러그할 수 있도록 데이터 프리젠테이션 등과 같은 작업들을 데이터 액세스로부터 분리시켜야만 한다는 전제 조건을 갖고 이러한 작업들의 구분 방식을 제공하고 있다[1].

Ⅱ. MVC 패턴과 모델 1, 모델 2 방식

모델1 방식은 페이지 중심적인 애플리케이션을 구성한다. 즉, 클라이언트로부터 요청을 받아 응답하는 전 과정을 JSP 페이지가 담당한다. 그리고, 각 JSP페이지는 JavaBean을 사용하여 비즈니스 로직을 수행하게 할 수도 있지만, 이 방식에서 중요한 점은 각 JSP페이지가 자신의 입력을 처리한다는 것이다.

모델 1 방식의 가장 큰 이점은 개발이 쉽다는 것이다. 따라서 웹 애플리케이션이 단순한 경우나 개발 시간이 적은 경우에 모델 1 방식이 적당하다. 그러나 웹 애플리케이션의 규모가 커지면서 유지보수와 개발에 문제점을 갖는다. 즉, JSP 페이지에 프리젠테이션 로직과 비즈니스 로직이 같이 포함되어 있기 때문에 JSP 페이지가 복잡해질 수밖에 없으며, 따라서 개발자와 웹 디자이너 사이에 작업을 분리시키기 어려워진다.

이와 함께 복잡한 JSP 페이지는 사용자의 변경된 요구사항을 반영하기 어려워진다는 문제점을 노출시킨다.

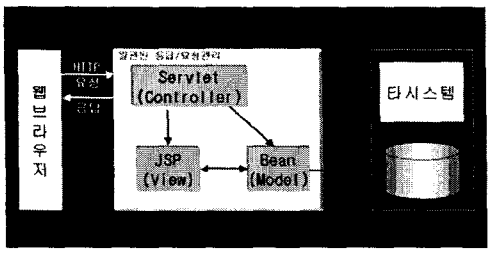


▶▶ 그림 1. 모델 1 방식 아키텍처

엔터프라이즈 웹 애플리케이션을 개발할 때는 모델 2 방식을 사용하는 것이 바람직하다. 모델 2 방식과 모델 1 방식 사이의 가장 중요한 차이점은 모델 2 방식에서 MVC 패턴을 기반으로 아키텍처를 구성하며, 클라이언트 요청의 초기 진입점(entry point)을 컨트롤러(controller)가 담당한다는 것이다. 모델 1 방식에서 초기 진입점은 뷰(view)인 JSP 페이지이지만, 모델 2 방식에서의 초기 진입점은 서블릿으로 구현한 컨트롤러가 된다. 모델 2 방식의 가장 큰 이점은 MVC 아키텍처 패턴을 사용하여 컨텐츠의 생성과 표현을 분리한다는 것이며, 이것은 다시 유연성이 있고, 관리 및 확장하기 쉬운 웹 애플리케이션을 손쉽게 만들 수 있게 한다[2].

MVC 패턴에서 서블릿이 담당하는 컨트롤러의 역할은 다음과 같다.

- 비즈니스 컴포넌트를 호출하고 요청에서 추출된 데이터를 전달한다.
- 비즈니스 컴포넌트를 호출한 결과에 따라 뷰가 표시할 모델을 생성한다.
- 세션 상태를 생성하고 조작한다.
- 뷰를 선택하고 뷰가 모델 데이터를 사용할 수 있게 한다.



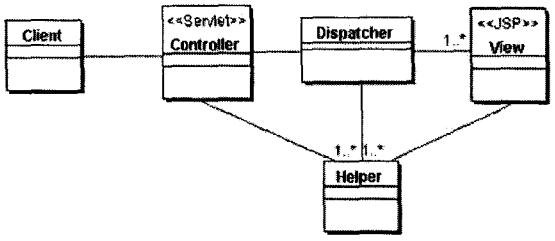
▶▶ 그림 2. 모델 2 방식 아키텍처

모델(model)은 뷰가 표시할 데이터를 포함하며, 일반적으로 J2EE 웹 애플리케이션에서 모델은 여러 자바빈(javabean)으로 구성된다. 일단 컨트롤러가 작업을 완료하고 컨텐츠를 생성할 뷰를 선택하면 모델은 뷰에 표시될 모든 데이터를 포함해야 한다. 모델 그 자체가 비즈니스 컴포넌트를 호출하여 데이터에 접근하지 않아야 한다. 그러나 모델 2 방식의 가장 큰 문제점은 개발하기 어렵다는 것이다. 따라서 모델 2 방식의 이점을 활용하면서 손쉽게 웹 애플리케이션을 개발하기 위해서는 웹 애플리케이션 프레임워크가 필수적으로 필요하게 된다.

Ⅲ. 웹 애플리케이션 프레임워크 아키텍처 패턴

1. Service to Worker 패턴

Service to Worker 패턴은 블루프린트에 소개된 Core J2EE Pattern의 하나로, Struts의 제어 과정을 이해하기 위해 반드시 학습해야 하는 기본 패턴이라고 할 수 있다. (그림 3)은 이 패턴의 클래스 다이어그램이다.



▶▶ 그림 3. Service to Worker

여기서 주목할 것은 클라이언트의 모든 요청이 예외 없이 Controller라고 이름 붙은 서블릿으로 전달되고 있다는 것이다. 이를 위해 web.xml에서 다음과 같이 *.do로 끝나는 모든 URL에 대해서 ActionServlet이 책임을 지도록 서블릿 맵핑을 설정해줘야 한다.

2. Dispatcher 패턴

Struts는 네비게이션과 뷰 관리를 담당할 Dispatcher 컴포넌트로써 Action 클래스를 사용하는데, 요청에 관련된 세부사항들을 Action에게 알리고 Action이 응답을 책임지도록 위임한다. 이것을 FrontController의 "Command and Controller" 전략이라고 부른다. 이 전략은 기존에 잘 알려진 Command 패턴에 Front Controller와 결합한 제어의 이동 역할을 더한 것이라고 볼 수 있다.

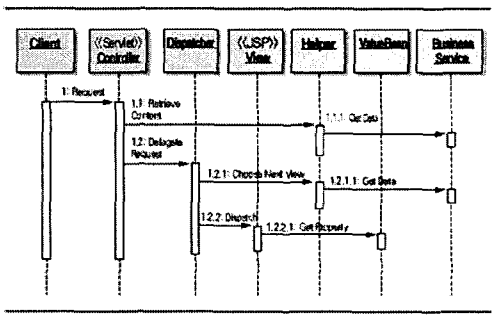
3. View Helper 패턴

Service to Worker 패턴에서 Dispatcher는 Helper를 사용하여 뷰 역할을 수행하는 JSP로 데이터를 보낸다. 비즈니스 데이터를 자바빈즈 형태의 뷰 컴포넌트로 만들어 JSP 로 넘기는 이러한 구조를 View Helper 패턴이라고 한다.

Struts에서는 이러한 Helper 역할을 ActionForm이 담당하

고 있다. 지금까지 설명한 Service to Worker 패턴의 동작 순서를 시퀀스 다이어그램으로 표시하면 (그림 4)와 같다.

Service to Worker 패턴의 동작 과정을 코드로 이해하고 싶다면, Command and Controller 전략이 적용된 Service to Worker 패턴을 구현한 서블릿 방식의 FrontController가 포함되어 있다. 또한 웹의 동기화로 인한 약점을 극복할 수 있도록 MD5 알고리즘이 적용된 Synchronizer Token 패턴도 구현되어 있다[3].



▶▶ 그림 4. Service to Worker 패턴의 시퀀스 다이어그램

Ⅳ. 프레임워크 설계 시 고려사항

프레임워크 설계 방법에는 크게 두 가지 접근 방식이 있다. 그 하나는 화이트박스 프레임워크(whitebox framework) 방식이고, 다른 하나는 블랙박스 프레임워크(blackbox framework) 방식이다.

1. 화이트박스 프레임워크

화이트박스 프레임워크는 주로 상속성(inheritance)이나 동적 바인딩(dynamic binding)과 같은 객체지향 언어의 특징에 의존한다. 이 방식에서는 프레임워크 안에 끼워 넣을 수 있는 컴포넌트의 인터페이스를 정의하는 방식을 사용한다. 이 방식에서는 프레임워크에서 정의한 인터페이스를 실현하는 컴포넌트를 구현하고, Strategy과 같은 디자인 패턴을 사용하여 이들 컴포넌트를 프레임워크 안에 통합시킴으로써 프레임워크의 기능을 확장하여 사용하게 된다. 또한, 프레임워크의 클래스 계층도의 세부사항과 밀접하게 결합되어있어 유연성이 결여된 시스템을 구축할 가능성이 많아진다는 단점을 갖는다.

2. 블랙박스 프레임워크

블랙박스 프레임워크는 상속성 보다는 객체 합성(object composition)이나 위임(delegation)을 사용하여 구조화한다. 따라서, 일반적으로 블랙박스 프레임워크가 화이트박스 프레임워크 보다 사용하거나 확장하기 쉽지만 설계하거나 구현하기는 더 어렵다.

3. Builtding Application Framework

이 프레임워크에서는 화이트박스 프레임워크와 블랙박스 프레임워크의 상황을 고려하여 도메인 지식으로부터 프레임워크를 도출하는데 다음과 같은 3개의 단계를 거쳐야 한다.

- 프레임워크를 개발하는데 기본이 되는 3개의 애플리케이션을 구현한다.
- 이들 애플리케이션을 일반화하는 화이트박스 프레임워크를 개발한다.
- 화이트박스 프레임워크를 블랙박스 프레임워크로 전환한다.
- 이를 테면 화이트박스 프레임워크가 프레임워크 개발 단계의 유아기라면, 블랙박스 프레임워크는 성숙기가 되는 셈이다.

4. 유연성과 확장성

유연성과 확장성을 갖는 프레임워크를 설계하기 위해서는 디자인 패턴을 활용하는 것이 중요하다. 애플리케이션의 가변성(variability) 영역에 디자인 패턴을 적용함으로써 유연성 있게 애플리케이션의 기능을 확장시키는 것이 바람직하다. 다음 표는 애플리케이션의 가변성 영역에 일반적으로 사용되는 GoF 디자인 패턴을 보여준다[4].

Ⅴ. 프레임워크 설계 과정

MVC 패턴을 사용하여 모델 2 방식의 웹 애플리케이션 아키텍처를 구성하며, 앞에서 언급한 일반적인 웹 애플리케이션의 요구사항을 해결하기 위해 다음과 같은 아키텍처 패턴을 적용한다.

- 프레임워크는 Front Controller 패턴을 적용하여 요청처리를 위한 집중식 접근 포인트를 제공
- 컨트롤러 중심적인 아키텍처를 제공
- Service to Worker 패턴을 사용하여 제어권이 뷰로 넘겨지기 전에 컨트롤러에 의해 비즈니스 로직이 호출되고 요청을 처리한다.
- View Helper 패턴을 적용함으로써 뷰에서 프로그램 로직을 분리
- 클라이언트 개발자와 웹 페이지 디자이너 사이의 역할을 분리시킨다.

각 아키텍처 패턴의 구성요소를 설계하는 작업을 수행할 때 우리는 디자인 패턴을 활용하여 디자인 패턴 기반으로 프레임워크를 설계할 수 있다.

- FrontController는 HelperFactory에게 request Helper의 생성을 요청
- helperFactory는 Factory Method 패턴을 사용하여 requestHelper를 생성하고 FrontController 에게 리턴한다.
- helperFactory는 Singleton 패턴을 적용된다.
- FrontController의 요청에 의해 requestHelper는 Template Method 패턴을 사용하여 Singleton 패턴이 적용된 CommandFactory에게 command의 생성을 요청
- FrontController에게 command가 반환되면 Command 패턴을 사용하여 command를 실행할 수 있게 된다.

위와 같은 때에 command Business Delegate 패턴이 적용된 BusinessDelegate에게 서비스를 요청할 수 있다.

각 프레임워크 설계 요소 즉, 디자인 패턴에 적용되는 클래스를 식별하기 위해 시퀀스 다이어그램을 작성한다. 이제 식별된 프레임워크 설계 요소를 각 디자인 패턴에 배분하여 디자인 패턴 기반의 클래스 다이어그램을 작성한다. 설계 요소 식별 후 각 프레임워크 설계 요소를 설계한다.

- FrontController 클래스 설계
- HelperFactor 클래스 설계
- IRequestHelper 인터페이스 설계
- AbstratRequestHelper 클래스 설계
- RequestHelper 클래스 설계
- ICommandFactory 인터페이스 설계
- CommandFactory 클래스 설계
- ICommand 인터페이스 설계

Ⅵ. 결 론

웹 애플리케이션의 복잡성과 다양한 플랫폼의 등장으로 아키텍처 설계에 많은 어려움이 따르고 있다. 이를 극복하기 위해 본 논문에서는 웹 애플리케이션 프레임워크 정의 및 프레임워크 패턴, 설계시 고려사항등을 토대로 컴포넌트가 실행되는 기반을 구성하는 프레임워크를 설계하고자 한다. 이를 위해 설계 단계에서 도출된 설계 요소들의 구현 모델을 구조화하여 구현 모델의 디렉터리 구조와 네임스페이스 구조를 정의하고 개발 팀에게 구현할 설계 요소를 배분한다. 또한, 설계 요소를 구조화하기 위해 구현 모델은 패키지 뷰와 디렉터리 뷰로 정의된다. 구현 모델의 패키지 뷰는 관련된 자바 클래스와 파일을 논리적, 물리적으로 구조화하여 표현한다. 패키지는 UML 패키지로 표현하며 패키지 안에는 해당 패키지에 속하는 클래스가 놓이게 된다. 구현 모델의 디렉터리 뷰는 웹 애플리케이션의 디렉터리 구조를 표현한다. 설계 요소 구현 활동에서는 설계된 요소들이 개발자들에게 분배되고, 개발자들은 자신에게 분배된 설계 요소를 구현한다. 개발 분야는 사용자 인터페이스 구현과 비즈니스 로직 구현, 데이터베이스 구현 등으로 나눌 수 있으며 아키텍처 단계에서 정의한 레이어 아키텍처 스타일과 일치한다.

▌참 고 문 헌▐

[1] 민현기, 김수동, "컴포넌트 프레임워크 설계를 위한 실용적인 커넥터 패턴", 한국정보과학회, 2004.
[2] 이우진, 김민정, 정양재, 윤석진, 최연준, "J2EE 플랫폼에서의 개념적 컴포넌트 모델링 및 컴포넌트 생성 지원 도구 개발", 한국정보처리학회, 2001.
[3] 김행곤, 한은주, "컴포넌트 기반 방법론을 사용한 프레임워크 개발에 관한 연구", 한국정보처리학회, 2000.
[4] 전태웅, "CBD 프레임워크", 한국정보처리학회지, 제10권 3호, pp.104-110, 2003.