# Capítulo 2 Marco Teórico

## *2.1 La Plataforma J2EE*

Antes que surgiera la plataforma J2EE (Java 2 Enterprise Editon) existían diferentes APIs para la programación en Java del lado del servidor, esto provocó la división de los programadores en grupos aislados, haciendo difícil el desarrollo de tecnologías compatibles.

Con la introducción de esta plataforma se logró la estandarización de los APIs, mediante la especificación formal de las normas que regirían las aplicaciones del lado del servidor.

Las aplicaciones desarrolladas corren sobre un Servidor de Aplicaciones. Éste esta basado en la especificación J2EE por lo que las aplicaciones desarrolladas con esta plataforma corren en servidores de diferentes vendedores [SHENOY, 2004]. Los Servidores de Aplicaciones proveen al desarrollador de una infraestructura de servicios de bajo nivel para que solo se preocupe de la lógica de la aplicación y de la interfaz con el usuario.

Dentro de esta especificación se definen también los llamados contenedores que puede soportar un Servidor de Aplicaciones. Un Contenedor se encarga de la administración del ciclo de vida de los componentes del lado del servidor. Existen 2 tipos de contenedores [INDERJEET, 2002]:

- Contenedor Web. Administra el ciclo de vida de los Servlets y JSP.
- Contenedor EJB. Se encarga del manejo de los Enterprise Java Beans.

Es importante aclarar que los Enteprise Java Beans no son lo mismo que los Java Beans ya que se pueden confundir en los siguientes capítulos.

### 2.1.1 Servlets

Un Servlet es una clase de Java que extiende la clase **javax.servlet.http.HttpServlet** e implementa los métodos **doPost** y **doGet**, mediante los cuales responde de manera dinámica a las peticiones de los clientes. Informalmente podemos decir que un Servlet es código Java con HTML incrustado.

La desventaja del uso de Servlets es la dificultad para mantener y programar el código HTML que puede contener y por lo regular los diseñadores de páginas Web no saben programar en Java.

### 2.1.2 Java Server Pages

Los JSP de sus siglas en ingles **Java Server Pages** son usados para la parte de la presentación y como su nombre lo indica son archivos con código HTML, código Java, e identificadores diseñados especialmente para generar contenido dinámico. Aunque tienen la extensión ".jsp" cuando un cliente hace una petición por un JSP por primera vez o el desarrollador lo precompila, el contenedor Web lo convierte en un Servlet automáticamente.

## *2.2 Patrones de Diseño*

Cuando se desarrollan aplicaciones para resolver problemas de la vida real, se crean modelos que probablemente se ajustan a dicha solución, pero muy pocas veces el código que se genera es fácil de mantener y reutilizar. Cuando se aprende a programar se enfoca más a cuestiones de la sintaxis de un lenguaje en específico, así como la solución de problemas de manera intuitiva, es decir, no se usan estándares prediseñados que permitan solucionar los problemas de manera elegante y eficiente.

Estos estándares que los expertos utilizan para implementar sistemas, mediante una guía o plantilla de solución, son llamados comúnmente Patrones de Diseño, que son soluciones bien documentadas que los expertos aplican para solucionar nuevos problemas ya que han sido utilizados con éxito en el pasado [STELTING, 2002].

Cuando se adentra en los patrones de diseño, se debe estar conciente que no siempre se programan siguiendo al pie de la letra los principios de estos modelos, ya que los patrones de diseño no deben ir directamente al código, sino a un análisis mental [FREEMAN, 2004], de ahí que para poder utilizarlos se debe conocer perfectamente el problema para poder ubicar un patrón o una variación de este, que se ajuste perfectamente a la solución.

### 2.2.3 MVC: El “Rey” de los Patrones

Dentro de la gran cantidad de patrones, existe un patrón muy utilizado por los lenguajes orientados a objetos llamado Model-View-Controller. Ha sido utilizado desde la aparición de Smalltalk en los 80s [KRASNER, 1988], algunos autores consideran este patrón como el rey de los patrones debido a que ha tenido un gran impacto en las interfaces gráficas de usuario (GUI por sus siglas en ingles) y en la WEB [FREEMAN, 2004].

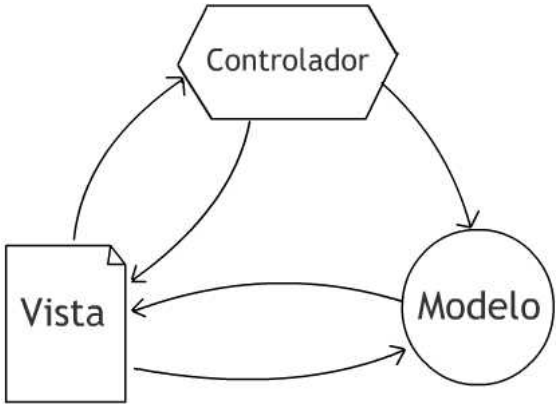


**Figura 2.1 Model-View-Controller**

El Model-View-Controller, como su nombre lo dice, divide nuestro sistema en 3 partes con actividades específicas, Modelo, Vista y Controlador. En la Figura 2.1 se muestra cada componente de esta arquitectura según la definición original de este patrón. El modelo se encarga de la lógica aplicativa de nuestro sistema, la vista es la interfaz que mostramos al usuario y el controlador es el mediador entre la vista y el modelo.

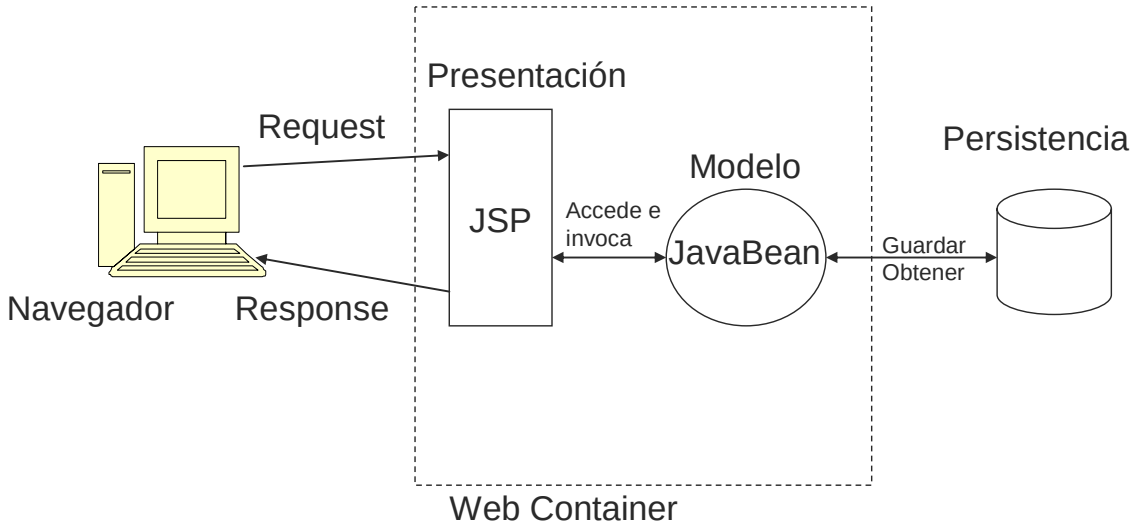


**Figura 2.2 Arquitectura del Modelo 1**

## *2.3 Arquitectura de las Aplicaciones Web*

Dentro de los diseños arquitecturales para el desarrollo de aplicaciones Web, se usan comúnmente 2 arquitecturas dependiendo del tamaño de nuestra aplicación y de las necesidades de la misma.

### 2.3.1 Arquitectura del Modelo 1

Este modelo es la forma más fácil de desarrollar aplicaciones Web, esta centrado en los JSP, los cuales reciben directamente las peticiones de los clientes y al mismo tiempo administran la lógica de la navegación, es decir, dependiendo de la petición del usuario, eligen cuál será el próximo JSP a mostrar. Están acompañados de modelos representados por los **JavaBeans** que contienen la lógica aplicativa de nuestra aplicación (Figura 2.2).

La desventaja de este modelo es que mezcla la presentación (vistas de los usuarios) con la lógica de la navegación, haciendo difícil el mantenimiento y el desarrollo de aplicaciones grandes.

### 2.3.2 Arquitectura del Modelo 2

En este modelo, la vista es responsable de la presentación y obtiene el estado del modelo desde el controlador, ya que no tiene comunicación directa con el modelo, dicho modelo representa la lógica aplicativa y el estado de nuestro sistema, es el único que debe tener comunicación con un DBMS. El controlador por su parte recibe las peticiones de los usuarios y se comunica con el modelo, hace que el modelo se actualice y lo pone a disposición de la vista [BASHMAN, 2004].

A diferencia del Modelo 1, aquí contamos con un controlador que se encarga de las peticiones del usuario y de la lógica de la navegación en vez de que sea un JSP, esto hace aún más modular nuestra aplicación, permitiendo separar presentación, navegación y la lógica aplicativa (Figura 2.3).

Cuando se construyen aplicaciones de tamaño considerable, típicamente se tienen 2 opciones, tener varios controladores uno por cada vista-modelo que utilicemos o utilizar un solo controlador con varios modelos y vistas. Utilizar varios controladores tiene la desventaja que se generan demasiados Servlets con duplicación de código y se tiene que implementar la seguridad sobre cada uno de los controladores [BASHMAN, 2004]. Por eso se tiende a usar la segunda opción ya que centraliza todas las peticiones del usuario en una sola parte.

Por otro lado, si se analiza la estructura de una aplicación Web con un solo controlador, encontramos que la ventaja de centralizar todas las tareas en un solo lugar puede causar que el controlador este muy saturado y dependiendo de la petición, además de tener que validar los parámetros, el controlador debe buscar el modelo que debe actualizar

y la vista que va a presentar, por ejemplo, si se tiene que desarrollar una aplicación con 15 JSP, y cada uno tiene 5 ligas, el número total de peticiones que el controlador debe manejar suma un total de 75, es decir, 75 bloques de IF en una sola clase, volviéndose un **Fat Controller** [SHENOY, 2004].

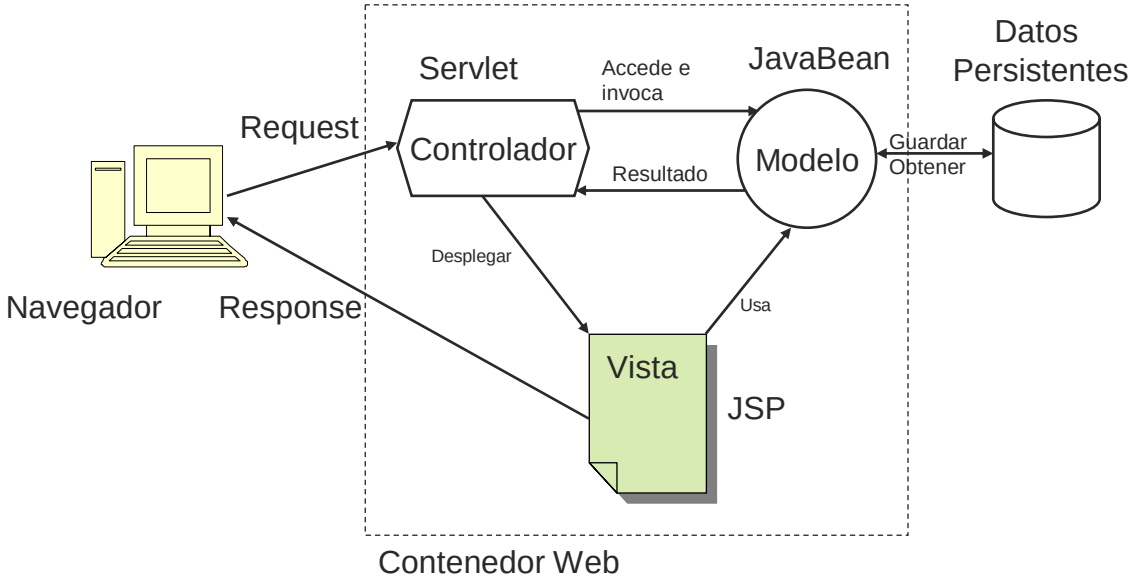


**Figura 2.3 Arquitectura del Modelo 2**

La solución por la que se puede optar es hacer el controlador configurable y utilizar clases externas que se encarguen de las peticiones, es decir, seguir usando un solo controlador que puede ser configurado desde un archivo de texto, donde se especifican las clases que se encargan de las peticiones de las vistas, es decir, dividir el controlador en 3 partes, un solo Servlet que recibe todos las peticiones, varias clases (Manejadores de Peticiones) que ayuden al Servlet con las peticiones y la comunicación con los modelos, y un archivo de configuración donde se especifiquen las relaciones entre peticiones y clases para que el Servlet principal pueda delegar responsabilidades a las clases especificadas como lo muestra la Figura 2.4 [SHENOY, 2004].

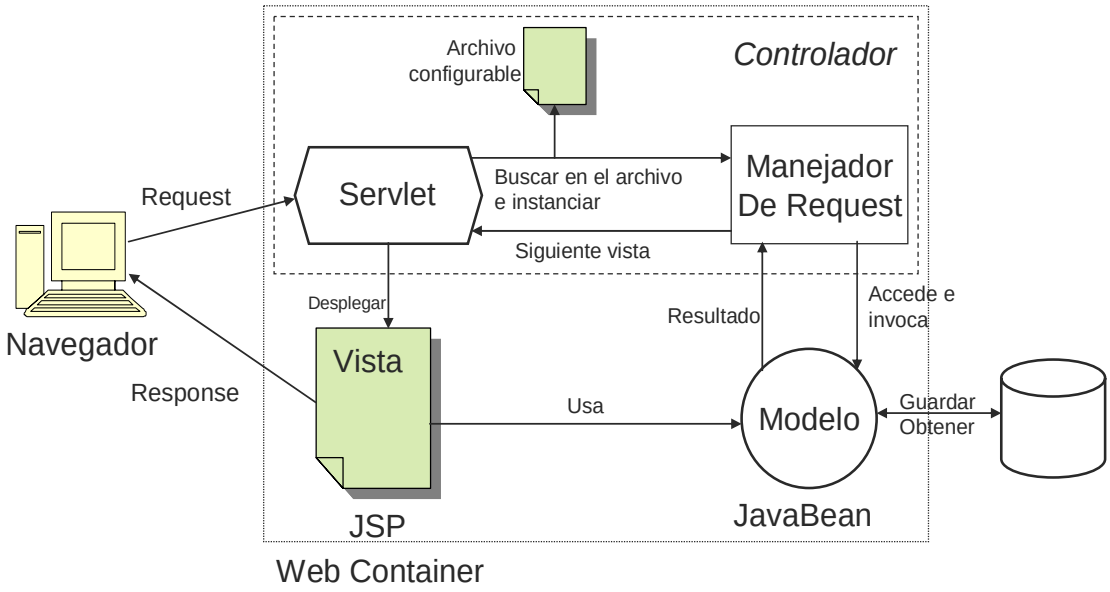


**Figura 2.4 MVC Configurable**

## *2.4 Introducción a Struts*

A finales de Mayo del 2000, Craig McClanaham desarrollo la primera versión de un framework que utiliza la idea del controlador configurable, al cual llamó Struts, este sistema además incluía algunas otras características que lo hicieron popular rápidamente entre la comunidad de desarrolladores Java [MCCLANAHAN, 2004].

Struts facilita el desarrollo de aplicaciones Web encargándose de las desventajas mostradas sobre el uso del modelo 2, ya que como se dijo anteriormente, utiliza un solo controlador, el cuál por medio de un archivo declarativo sabe a que modelo, vista o acción acudir dependiendo de la petición del usuario.

Struts esta construido sobre varias tecnologías entre ellas, Java Servlets, Java Server Pages, Java Beans y XML básicamente, puede interactuar con otras herramientas como Enterprise Java Beans y JDBC, si se requiere manejar datos persistentes [MCCLANAHAN, 2004]. Las partes principales de Struts son iguales a las de la Figura 2.4 lo único que cambia son los nombres, en general un cliente hace una petición al Servlet principal llamado **ActionServlet**, que ya viene implementado en el framework, este servlet busca

dentro del archivo de configuración llamado **strus-config.xml** (nombre dado por convención pero que no es obligatorio) el **Action** que corresponde a esa petición, el cual puede acceder a un modelo y por consiguiente a datos persistentes, esta **Action** regresa como resultado la siguiente vista que se debe mostrar al cliente. El ciclo se repite hasta que el usuario sale del sistema como lo muestra la Figura 2.5.

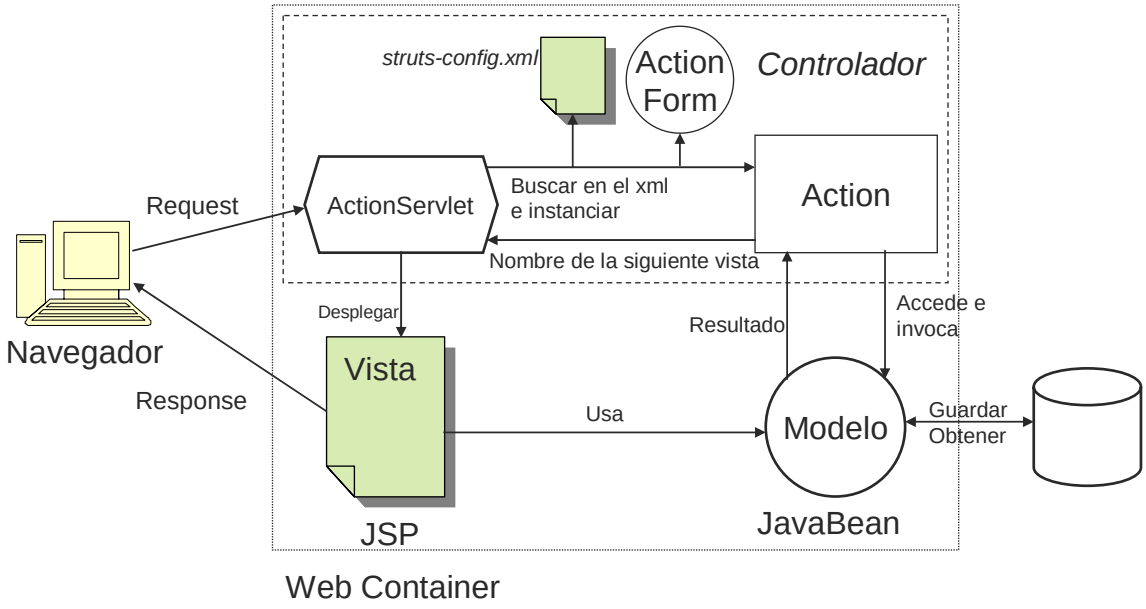


**Figura 2.5 Arquitectura general de Struts**

## 2.4.1 Frameworks

A medida que se gana experiencia en el desarrollo de software, se descubren ciertas partes de un sistema que pueden implementarse para ser reutilizables, al juntar dichas partes se genera una colección de piezas genéricas que nos ahorran tiempo de programación. Un framework es un conjunto de clases cooperativas que implementan los mecanismos esenciales para el dominio de un problema en particular [HORSTMANN, 2004]. Por lo regular los programadores extienden la funcionalidad del framework para resolver el problema.

Existe un patrón de diseño dentro de los frameworks llamado “Inversión de Control” o “Principio de Hollywood”, por el dicho “No nos llames, nosotros te llamamos”, este principio describe la analogía de los productores y los artistas cuando se hace un casting para una película. En términos de programación, dice que los componentes de alto nivel deben llamar a los componentes de bajo nivel y no al revés [FREEMAN, 2004]. En otras palabras significa que las clases del framework controlan el flujo de la aplicación.

Como se ha dicho anteriormente Struts es un framework que cumple con el principio de Inversión de Control, ya que el llamado a las **Actions** que el usuario extiende, se hace mediante el **ActionServlet**, clase que viene implementada en el framework.