Capítulo II.

Arquitectura del Software

Después de un cuidadoso análisis de los objetivos del proyecto, se determinó que la mejor manera de estructurar el sistema era haciendo uso del muy famoso "patrón de diseño"[2]: *Model-View-Controller*. A continuación se explicará más detalladamente en qué consiste esta arquitectura.

2.1 Model-View-Controller

MVC (por sus siglas en inglés) es un patrón de diseño de arquitectura de software usado principalmente en aplicaciones que manejan gran cantidad de datos y transacciones complejas donde se requiere una mejor separación de conceptos para que el desarrollo esté estructurado de una mejor manera, facilitando la programación en diferentes capas de manera paralela e independiente. *MVC* sugiere la separación del software en 3 estratos: Modelo, Vista y Controlador, los cuales serán explicados en breve:

Modelo: Es la representación de la información que maneja la aplicación. El modelo en sí son los datos puros que puestos en contexto del sistema proveen de información al usuario o a la aplicación misma.

Vista: Es la representación del modelo en forma gráfica disponible para la interacción con el usuario. En el caso de una aplicación Web, la "Vista" es una página HTML con contenido dinámico sobre el cuál el usuario puede realizar operaciones.

[2] Un *"patrón de diseño"* es una solución probada para un problema en un contexto

Controlador: Es la capa encargada de manejar y responder las solicitudes del usuario, procesando la información necesaria y modificando el Modelo en caso de ser necesario.

2.2 Ciclo de vida de *MVC*

El ciclo de vida de *MVC* es normalmente representado por las 3 capas presentadas anteriormente y el cliente (también conocido como usuario). El siguiente diagrama representa el ciclo de vida de manera sencilla:

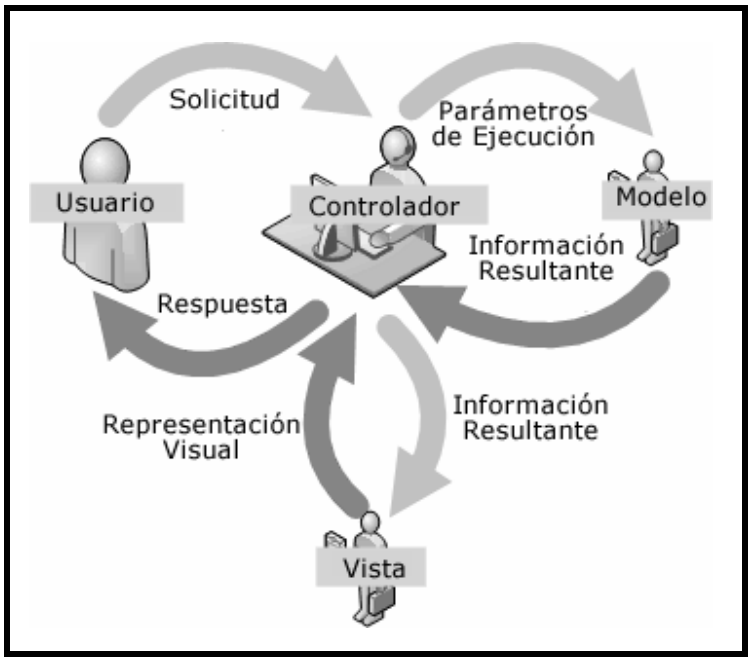

Diagrama 1 Ciclo de vida MVC

El primer paso en el ciclo de vida empieza cuando el usuario hace una solicitud al controlador con información sobre lo que el usuario desea realizar. Entonces el Controlador decide a quién debe delegar la tarea y es aquí donde el Modelo empieza su trabajo. En esta etapa, el Modelo se encarga de realizar operaciones sobre la información que maneja para cumplir con lo que le solicita el Controlador. Una vez que termina su labor, le regresa al Controlador la información resultante de sus operaciones, el cuál a su vez redirige a la

Vista. La Vista se encarga de transformar los datos en información visualmente entendible para el usuario. Finalmente, la representación gráfica es transmitida de regreso al Controlador y éste se encarga de transmitírsela al usuario. El ciclo entero puede empezar nuevamente si el usuario así lo requiere.

2.3 Ventajas y Desventajas de *MVC*

Las principales ventajas de hacer uso del patrón *MVC* son:

- La separación del Modelo de la Vista, es decir, separar los datos de la representación visual de los mismos.
- Es mucho más sencillo agregar múltiples representaciones de los mismos datos o información.
- Facilita agregar nuevos tipos de datos según sea requerido por la aplicación ya que son independientes del funcionamiento de las otras capas.
- Crea independencia de funcionamiento.
- Facilita el mantenimiento en caso de errores.
- Ofrece maneras más sencillas para probar el correcto funcionamiento del sistema.
- Permite el escalamiento de la aplicación en caso de ser requerido.

Las desventajas de seguir el planteamiento de *MVC* son:

- La separación de conceptos en capas agrega complejidad al sistema.
- La cantidad de archivos a mantener y desarrollar se incrementa considerablemente.
- La curva de aprendizaje del patrón de diseño es más alta que usando otros modelos más sencillos.

Cabe mencionar que la comparación de ventajas y desventajas de *MVC* puede ser un tema muy subjetivo y se puede prestar como tema de debate, sin embargo se tomó la

decisión usando principalmente los puntos mencionados anteriormente ya que en términos generales la balanza se inclina a favor del *MVC* en vez de en su contra.

2.4 MVC en uso

Actualmente existen muchos *frameworks*[3] disponibles para el desarrollo de aplicaciones MVC; sin embargo su uso requiere de una comprensión completa de su funcionamiento haciéndolas merecedoras de temas de tesis completas. En este caso se deseaba obtener resultados rápidos sin involucrar más complejidad que la que el tema propio requiere, así que se decidió seguir el esquema que plantea MVC sin hacer uso de *frameworks* externos.

Como anteriormente fue mencionado, el "Sistema Asistente para la Generación de Horarios de Cursos" es una aplicación Web y la plataforma de desarrollo elegida fue Java (de Sun Microsystems) ya que es un lenguaje de programación sumamente maduro y robusto, no solo para aplicaciones de escritorio, sino para aplicaciones que corren sobre Internet usando estándares mundiales.

Para desarrollar aplicaciones Java para Internet es necesario hacer uso de la tecnología *Java Server Pages*, las cuales permiten incrustar código que generen páginas con contenido dinámico cuando cualquier cliente Web solicite información del servidor. Los *JSPs* pueden ser entendidos como una abstracción de alto nivel de un *Servlet* ya que para poder ser entendidas por el servidor Web es necesario que sean compiladas en forma

[3] Un *framework* es términos de Software es un diseño re-usable de un sistema para facilitar la resolución de problemas complejos que presentan comúnmente proyectos o aplicaciones de Software. Algunos ejemplos de Frameworks para Java son: Spring, Struts, Tapestry y WebWork (entre otros)

de un *Servlet*[4] y posteriormente sean interpretadas por un compilador de Java[5]. (Wikipedia, 2007)

Con el propósito de seguir el patrón *MVC* se determinó usar la tecnología *Java Servlets* para cumplir las funciones del Controlador y la tecnología *JSP* para las funciones de la capa de la Presentación. Para satisfacer las necesidades del Modelo se hará uso de una plataforma llamada *Hibernate*, la cual será explicada con más detenimiento en el siguiente capítulo.

[4] Un *Servlet* es, en términos generales, un objeto de Java que recibe las peticiones de un cliente Web y responde en base a sus solicitudes.
[5] Los archivos *JSP* también pueden ser compilados directamente en *byte code* (dejando de lado la necesidad de una segunda compilación de Servlet a *byte code*) o pueden ser compiladas *al vuelo* si así se configura el servidor.