

Universidad Interamericana de Panamá

Faculta de Ingeniería

Profesor

Abdel Martínez

Curso

Programación 2

Tema

El Modelo Vista Controlador

Estudiantes

Melvin Domínguez 08-0858- 0868

Año

2016

**Introducción**

MVC es un patrón de diseño orientado a objetos. Fue desarrollado en el Centro de Investigaciones Xerox Palo Alto a finales de los años setenta.

Es una solución que va muy bien con los sitios web.

Cada hit en una página es la interacción de un usuario (entrada) con el sistema (el servidor web procesando los scripts PHP). Asumiendo que se necesita mantener persistencia y que se necesita presentar la información en una variedad de maneras, el patrón MVC es una buena solución.

Hoy día en cualquier lugar del mundo los que construyen aplicaciones informáticas centran su atención en dos aspectos fundamentales: cómo lograr construir mejores aplicaciones en menos tiempo, y cómo utilizar mayor cantidad de estándares en el diseño de las aplicaciones que permitan mayor reutilización del código y mejores mantenimientos de los sistemas desarrollados.

Teniendo en cuenta el creciente uso de la programación orientada a objeto en la concepción e implementación de este tipo de aplicaciones, y la gran actualidad que tiene el uso de patrones internacionalmente aceptados, se propone en este artículo un análisis somero del patrón Modelo- Vista Controlador (MVC).

**El Modelo Vista Controlador**

**Qué es MVC**

En líneas generales, MVC es una propuesta de diseño de software utilizada para implementar sistemas donde se requiere el uso de interfaces de usuario. Surge de la necesidad de crear software más robusto con un ciclo de vida más adecuado, donde se potencie la facilidad de mantenimiento, reutilización del código y la separación de conceptos.

Su fundamento es la separación del código en tres capas diferentes, acotadas por su responsabilidad, en lo que se llaman Modelos, Vistas y Controladores, o lo que es lo mismo, Model, Views & Controllers, si lo prefieres en inglés. En este artículo estudiaremos con detalle estos conceptos, así como las ventajas de ponerlos en marcha cuando desarrollamos.

MVC es un "invento" que ya tiene varias décadas y fue presentado incluso antes de la aparición de la Web. No obstante, en los últimos años ha ganado mucha fuerza y seguidores gracias a la aparición de numerosos frameworks de desarrollo web que utilizan el patrón MVC como modelo para la arquitectura de las aplicaciones web.

**Por qué MVC**

La rama de la ingeniería del software se preocupa por crear procesos que aseguren calidad en los programas que se realizan y esa calidad atiende a diversos parámetros que son deseables para todo desarrollo, como la estructuración de los programas o reutilización del código, lo que debe influir positivamente en la facilidad de desarrollo y el mantenimiento.

Los ingenieros del software se dedican a estudiar de qué manera se pueden mejorar los procesos de creación de software y una de las soluciones a las que han llegado es la arquitectura basada en capas que separan el código en función de sus responsabilidades o conceptos. Por tanto, cuando estudiamos MVC lo primero que tenemos que saber es que está ahí para ayudarnos a crear aplicaciones con mayor calidad.

Quizás, para que a todos nos queden claras las ventajas del MVC podamos echar mano de unos cuantos ejemplos:

1. Aunque no tenga nada que ver, comencemos con algo tan sencillo como son el HTML y las CSS. Al principio, en el HTML se mezclaba tanto el contenido como la presentación. Es decir, en el propio HTML tenemos etiquetas como "font" que sirven para definir las características de una fuente, o atributos como "bgcolor" que definen el color de un fondo. El resultado es que tanto el contenido como la presentación estaban juntos y si algún día pretendíamos cambiar la forma con la que se mostraba una página, estábamos obligados a cambiar cada uno de los archivos HTML que componen una web, tocando todas y cada una de las etiquetas que hay en el documento. Con el tiempo se observó que eso no era práctico y se creó el lenguaje CSS, en el que se separó la responsabilidad de aplicar el formato de una web.
2. Al escribir programas en lenguajes como PHP, cualquiera de nosotros comienza mezclando tanto el código PHP como el código HTML (e incluso el Javascript) en el mismo archivo. Esto produce lo que se denomina el "Código Espagueti". Si algún día pretendemos cambiar el modo en cómo queremos que se muestre el contenido, estamos obligados a repasar todas y cada una de las páginas que tiene nuestro proyecto. Sería mucho más útil que el HTML estuviera separado del PHP.
3. Si queremos que en un equipo intervengan perfiles distintos de profesionales y trabajen de manera autónoma, como diseñadores o programadores, ambos tienen que tocar los mismos archivos y el diseñador se tiene necesariamente que relacionar con mucho código en un lenguaje de programación que puede no serle familiar, siendo que a éste quizás solo le interesan los bloques donde hay HTML. De nuevo, sería mucho más fácil la separación del código.
4. Durante la manipulación de datos en una aplicación es posible que estemos accediendo a los mismos datos en lugares distintos. Por ejemplo, podemos acceder a los datos de un artículo desde la página donde se muestra éste, la página donde se listan los artículos de un manual o la página de backend donde se administran los artículos de un sitio web. Si un día cambiamos los datos de los artículos (alteramos la tabla para añadir nuevos campos o cambiar los existentes porque las necesidades de nuestros artículos varían), estamos obligados a cambiar, página a página, todos los lugares donde se consumían datos de los artículos. Además, si tenemos el código de acceso a datos disperso por decenas de lugares, es posible que estemos repitiendo las mismas sentencias de acceso a esos datos y por tanto no estamos reutilizando código.

Quizás te hayas visto en alguna de esas situaciones en el pasado. Son solo son simples ejemplos, habiendo decenas de casos similares en los que resultaría útil aplicar una arquitectura como el MVC, con la que nos obliguemos a separar nuestro código atendiendo a sus responsabilidades.

Ahora que ya podemos tener una idea de las ventajas que nos puede aportar el MVC, analicemos las diversas partes o conceptos en los que debemos separar el código de nuestras aplicaciones.

**Modelos**

Es la capa donde se trabaja con los datos, por tanto contendrá mecanismos para acceder a la información y también para actualizar su estado. Los datos los tendremos habitualmente en una base de datos, por lo que en los modelos tendremos todas las funciones que accederán a las tablas y harán los correspondientes selects, updates, inserts, etc.

No obstante, cabe mencionar que cuando se trabaja con MCV lo habitual también es utilizar otras librerías como PDO o algún ORM como Doctrine, que nos permiten trabajar con abstracción de bases de datos y persistencia en objetos. Por ello, en vez de usar directamente sentencias SQL, que suelen depender del motor de base de datos con el que se esté trabajando, se utiliza un dialecto de acceso a datos basado en clases y objetos.

**Vistas**

Las vistas, como su nombre nos hace entender, contienen el código de nuestra aplicación que va a producir la visualización de las interfaces de usuario, o sea, el código que nos permitirá renderizar los estados de nuestra aplicación en HTML. En las vistas nada más tenemos los códigos HTML y PHP que nos permite mostrar la salida.

En la vista generalmente trabajamos con los datos, sin embargo, no se realiza un acceso directo a éstos. Las vistas requerirán los datos a los modelos y ellas se generará la salida, tal como nuestra aplicación requiera.

**Controladores**

Contiene el código necesario para responder a las acciones que se solicitan en la aplicación, como visualizar un elemento, realizar una compra, una búsqueda de información, etc.

En realidad es una capa que sirve de enlace entre las vistas y los modelos, respondiendo a los mecanismos que puedan requerirse para implementar las necesidades de nuestra aplicación. Sin embargo, su responsabilidad no es manipular directamente datos, ni mostrar ningún tipo de salida, sino servir de enlace entre los modelos y las vistas para implementar las diversas necesidades del desarrollo.

**Arquitectura de aplicaciones MVC**

A continuación encontrarás un diagrama que te servirá para entender un poco mejor cómo colaboran las distintas capas que componen la arquitectura de desarrollo de software en el patrón MVC.

Veamos paso a paso cómo sería el flujo de trabajo característico en un esquema MVC.

1. El usuario realiza una solicitud a nuestro sitio web. Generalmente estará desencadenada por acceder a una página de nuestro sitio. Esa solicitud le llega al controlador.
2. El controlador comunica tanto con modelos como con vistas. A los modelos les solicita datos o les manda realizar actualizaciones de los datos. A las vistas les solicita la salida correspondiente, una vez se hayan realizado las operaciones pertinentes según la lógica del negocio.
3. Para producir la salida, en ocasiones las vistas pueden solicitar más información a los modelos. En ocasiones, el controlador será el responsable de solicitar todos los datos a los modelos y de enviarlos a las vistas, haciendo de puente entre unos y otros. Sería corriente tanto una cosa como la otra, todo depende de nuestra implementación; por eso esa flecha la hemos coloreado de otro color.
4. Las vistas envían al usuario la salida. Aunque en ocasiones esa salida puede ir de vuelta al controlador y sería éste el que hace el envío al cliente, por eso he puesto la flecha en otro color.

Es decir

El modelo es el responsable de:

1. Acceder a la capa de almacenamiento de datos. Lo ideal es que el modelo sea independiente del sistema de almacenamiento.
2. Define las reglas de negocio (la funcionalidad del sistema). Un ejemplo de regla puede ser: "Si la mercancía pedida no está en el almacén, consultar el tiempo de entrega estándar del proveedor".
3. Lleva un registro de las vistas y controladores del sistema.
4. Si estamos ante un modelo activo, notificará a las vistas los cambios que en los datos pueda producir un agente externo (por ejemplo, un fichero por lotes que actualiza los datos, un temporizador que desencadena una inserción, etc.).

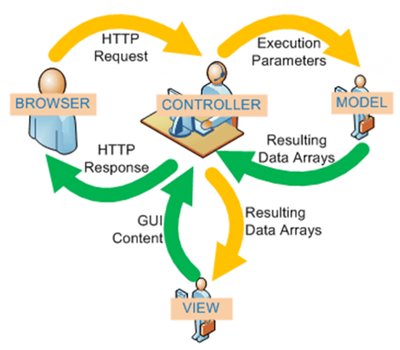
El controlador es responsable de:

1. Recibe los eventos de entrada (un clic, un cambio en un campo de texto, etc.).
2. Contiene reglas de gestión de eventos, del tipo "SI Evento Z, entonces Acción W". Estas acciones pueden suponer peticiones al modelo o a las vistas. Una de estas peticiones a las vistas puede ser una llamada al método "Actualizar ()". Una petición al modelo puede ser "Obtener\_tiempo\_de\_entrega ( nueva\_orden\_de\_venta )".

Las vistas son responsables de:

1. Recibir datos del modelo y la muestra al usuario.
2. Tienen un registro de su controlador asociado (normalmente porque además lo instancia).
3. Pueden dar el servicio de "Actualización ()", para que sea invocado por el controlador o por el modelo (cuando es un modelo activo que informa de los cambios en los datos producidos por otros agentes).

**Mapa conceptual**



**Conclusión**

A manera de conclusión decimos que conforme se incrementan las necesidades de cualquier aplicación, la modificación al código existente se hace inminente, y si no existe una clara división de uso, el código no sólo se torna indescifrable, sino en ocasiones impredecible, debido a la mezcla de funcionalidades que pueden surgir.

La estructura MVC ("Model-View-Controller") es un paradigma utilizado en el desarrollo de diversos software, a través de este patrón se logra una división de las diferentes partes que conforman una aplicación, permitiendo la actualización y mantenimiento del software de una forma sencilla y en un reducido espacio de tiempo.

El uso de los frameworks basados en el patrón MVC permite tener una separación lógica y física de los componentes de la aplicación, permitiendo a su vez, una mayor especialización de los desarrolladores y diseñadores de la aplicación, además de contribuir a una elevada organización en el trabajo.