REVISTA DIGITAL DE LAS TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES
TELEM@TICA
Revista Telem@tica. Vol. 11. No. 1, enero-abril, 2012, p. 47-57 ISSN 1729-3804

Patrón Modelo-Vista-Controlador.

Yenisleidy Fernández Romero[1], Yanette Díaz González[2]

1 Delegación Provincial del MININT, La Habana. Ingeniero yenyfernandez@gmail.com

2 CUJAE, Ingeniero yanette.dg@electrica.cujae.edu.cu

RESUMEN

El patrón Modelo-Vista-Controlador (MVC) surge con el objetivo de reducir el esfuerzo de programación, necesario en la implementación de sistemas múltiples y sincronizados de los mismos datos, a partir de estandarizar el diseño de las aplicaciones. El patrón MVC es un paradigma que divide las partes que conforman una aplicación en el Modelo, las Vistas y los Controladores, permitiendo la implementación por separado de cada elemento, garantizando así la actualización y mantenimiento del software de forma sencilla y en un reducido espacio de tiempo. A partir del uso de *frameworks* basados en el patrón MVC se puede lograr una mejor organización del trabajo y mayor especialización de los desarrolladores y diseñadores.

Palabras claves: controlador, framework, modelo, vista.

Model-View-ControllerPattern

ABSTRACT

The Model-View-Controller (MVC) appears with the aim of reducing the programming effort necessary to implement multiple and synchronized systems of the same data, from standardizing the application design. The MVC structure is a paradigm that divides the parts that make up an application in the Model, Views and Controllers, allowing the implementation of each element separately, and thus the updating and maintenance of software easily and in a small space time. Effective use of frameworks based on MVC pattern can have a better organization of labor and specialization of developers and designers.

Palabras claves: controller, framework, model, view.

1. INTRODUCCIÓN

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, ycómo 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).

1.2 Orígenes del patrón Modelo-Vista-Controlador

Buscando un poco de información histórica, es posible afirmar que el patrón Modelo/Vista/Controlador o MVC (Model/View/Controller) fue descrito por primera vez en 1979 por TrygveReenskaug[1] e introducido como parte de la versión Smalltalk-80 del lenguaje de programación Smalltalk.

Fue diseñado para reducir el esfuerzo de programación necesario en la implementación de sistemas múltiples y sincronizados de los mismos datos. Sus características principales están dadas por el hecho de que, el Modelo, las Vistas y los Controladores se tratan como entidades separadas; esto hace que cualquier cambio producido en el Modelo se refleje automáticamente en cada una de las Vistas. Este modelo de arquitectura se puede emplear en sistemas de representación gráfica de datos, donde se presentan partes del diseño con diferente escala de aumento, en ventanas separadas.

Este modelo de arquitectura presenta varias ventajas [2]:
• Separación clara entre los componentes de un programa; lo cual permite su implementación por separado.
• Interfaz de Programación de Aplicaciones API (AplicationProgramming Interface) muy bien definida; cualquiera que use el API, podrá reemplazar el Modelo, la Vista o el Controlador, sin aparente dificultad.
• Conexión entre el Modelo y sus Vistas dinámica; se produce en tiempo de ejecución, no en tiempo de compilación.

Al incorporar el modelo de arquitectura MVC a un diseño, las piezas de un programa se pueden construir por separado y luego unirlas en tiempo de ejecución. Si uno de los componentes, posteriormente, se observa que funciona mal, puede reemplazarse sin que las otras piezas se vean afectadas. Este escenario contrasta con la aproximación monolítica típica de muchos programas de pequeña y mediana complejidad. Todos tienen un Frame que contiene todos los elementos, un controlador de eventos, un montón de cálculos y la presentación del resultado. Ante esta perspectiva, hacer un cambio aquí no es nada trivial.

2. Patrón Modelo-Vista-Controlador.

2.1 Definición de las partes.

El Modelo es el objeto que representa los datos del programa. Maneja los datos y controla todas sus transformaciones. El Modelo no tiene conocimiento específico de los Controladores o de las Vistas, ni siquiera contiene referencias a ellos. Es el propio sistema el que tiene encomendada la responsabilidad de mantener enlaces entre el Modelo y sus Vistas, y notificar a las Vistas cuando cambia el Modelo.

La Vista es el objeto que maneja la presentación visual de los datos representados por el Modelo. Genera una representación visual del Modelo y muestra los datos al usuario. Interactúa preferentemente con el Controlador, pero es posible que trate directamente con el Modelo a través de una referencia al propio Modelo.

El Controlador es el objeto que proporciona significado a las órdenes del usuario, actuando sobre los datos representados por el Modelo, centra toda la interacción entre la Vista y el Modelo. Cuando se realiza algún cambio, entra en acción, bien sea por cambios en la información del Modelo o por alteraciones de la Vista. Interactúa con el Modelo a través de una referencia al propio Modelo.

2.2 Elementos del patrón.

Modelo: datos y reglas de negocio.
Vista: muestra la información del modelo al usuario.
Controlador: gestiona las entradas del usuario.
La figura 2.1 ilustra los elementos del patrón y la interrelación entre estos.

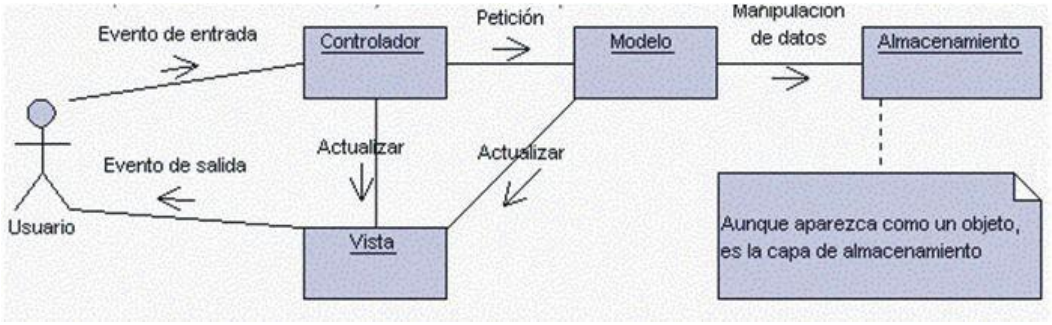

Figura 2.1 Interrelación entre los elementos del patrón MCV.

Un modelo puede tener diversas vistas, cada una con su correspondiente controlador, aunque es posible definir múltiples vistas para un único controlador. Un ejemplo clásico es el de la información de una base de datos, que se puede presentar de diversas formas: diagrama de tarta, de barras, tabular, etc. A continuación se detalla cada elemento [2]:

El modelo es el responsable de:
• Acceder a la capa de almacenamiento de datos. Lo ideal es que el modelo sea independiente del sistema de almacenamiento.

• Define 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". Es opcional, pues las reglas de negocio, pueden estar también en los controladores, directamente en las acciones.

• Notificará a las vistas los cambios que en los datos pueda producir un agente externo si se está ante un modelo activo (por ejemplo, un fichero bath que actualiza los datos, un temporizador que desencadena una inserción, etc.).

El controlador es responsable de:
• Recibir los eventos de entrada (un clic, un cambio en un campo de texto, etc.).

• 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:

• Recibir datos procesados por el controlador o del modelo y mostrarlos al usuario.

• Tienen un registro de su controlador asociado.

• 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.

Un ejemplo de MVC con un modelo pasivo (aquel que no notifica cambios en los datos) es la navegación web, que responde a las entradas del usuario, pero no detecta los cambios en datos del servidor. El Diagrama de Secuencia que se muestra en la figura 2.2 ilustra la interrelación de los elementos del patrón.

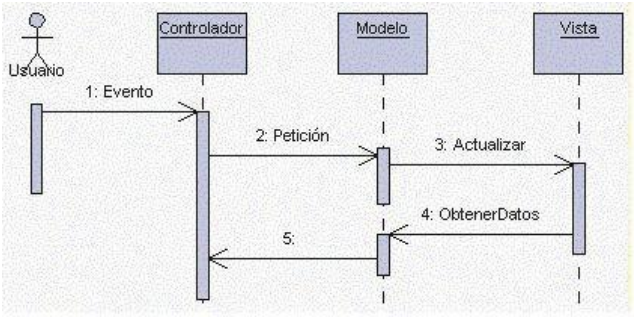

Figura 2.2 Diagrama de Secuencia.

Pasos:
1. El usuario introduce el evento.
2. El Controlador recibe el evento y lo traduce en una petición al Modelo (aunque también puede llamar directamente a la vista).
3. El modelo (si es necesario) llama a la vista para su actualización.

4. Para cumplir con la actualización la Vista puede solicitar datos al Modelo.
5. El Controlador recibe el control.

2.3 Modo de operación.

Aunque se pueden encontrar diferentes implementaciones de MVC, el flujo que sigue el control generalmente es el siguiente [3]:

1. El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace, etc.)

2. El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.

3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario. Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.

4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo. El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, el patrón de observador (controlador) puede ser utilizado para proveer cierta interacción entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aun así el modelo en sí mismo sigue sin saber nada de la vista. El controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la vista para que se actualice. Nota: En algunas implementaciones la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista.

5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

Para entender mejor de qué se está hablando veamos un ejemplo de implementación:
1. Un usuario entra a un sitio mediante la URL www.example.com/items/listar.

2. Se carga el Controlador Items para ejecutar la acción de Listar.

3. El controlador solicita al modelo que le entregue un arreglo con todos los items que hay almacenados en la base de datos.

4. Una vez que posee dicha información le indica a la vista que va a utilizar la plantilla correspondiente al listado de items y le provee el arreglo con todos los usuarios.

5. La vista, por su parte, toma el arreglo de items y los muestra uno a uno en la plantilla que le indicó el controlador.

6. Finalmente el usuario recibe el listado de items; lo observa un instante y decide que quiere agregar un nuevo item por lo que hace click en un enlace que lo lleva a la URL www.example.com/items/agregar.

7. Se repite el proceso desde el paso 1 pero con la nueva URL.

Analicemos como sería un ejemplo de código. Primeramente se muestra la programación estructurada en PHP, sin la utilización del patrón MVC y posteriormente con la utilización del patrón. El ejemplo no es más que un simple listado común presentado en una tabla HTML.

```
1
2  < ?php
3  require'conexion.php';
4  $db=newPDO('mysql:host='.$servidor.';dbname='.$bd,$usuario,$contrasenia
5  );
6  $consulta=$db->prepare('SELECT * FROM items WHERE id_item = ? OR
7  id_item= ?');
8  $consulta->execute(array(2,4));
9  $items=$consulta->fetchAll();
1  $db=null;
0  ?>
1  <!DOCTYPE html PUBLIC"-//W3C//DTD XHTML 1.0 Strict//EN"
1          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
1
2  <htmlxmlns="http://www.w3.org/1999/xhtml"xml:lang="en"lang="en">
1  <head>
3          <meta http-equiv="Content-Type"  content="text/html;
1  charset=utf-8"/>
4          <title>PDO -Journoly</title>
1  </head>
5  <body>
1  <table>
6          <tr>
1                  <th>ID
7                  </th><th>Item
1          </th></tr>
8          < ?php
1          foreach($itemsas$item)
9          {
2          ?>
0          <tr>
2                  <td>< ?phpecho$item['id_item']?></td>
1                  <td>< ?phpecho$item['item']?></td>
2          </tr>
2          < ?php
2          }
3          ?>
2  </table>
4  <a href="index.php">Men&uacute;</a>
2  </body>
5  </html>
```

Veamos como quedaría estructurado utilizando el Patrón MVC. Se separará dicho ejemplo, en 3 ficheros. Uno corresponderá al modelo, otro a la vista y el tercero será el controlador.

¿Cuál es el modelo en este ejemplo?

Como se mencionó anteriormente, el modelo es el que se ocupa, básicamente, de todo lo que tiene que ver con el acceso a la información. Sin dudarlo, en este ejemplo PDO es quien cumple el papel de Modelo.

Modelo.php

```
1 <?php
2 $db=newPDO('mysql:host='.$servidor.';dbname='.$bd,$usuario,$contrasenia);
3 $consulta=$db->prepare('SELECT * FROM items');
4 $consulta->execute();
5 $items=$consulta->fetchAll();
6 ?>
```

¿Y cuál es la vista?

La vista es quien representa la información para que el usuario la pueda entender, en este caso, el HTML, la tabla y todo lo usado para mostrar la información forma parte de la vista.

Vista.php

```
<!DOCTYPE html PUBLIC"-//W3C//DTD XHTML 1.0 Strict//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<htmlxmlns="http://www.w3.org/1999/xhtml"xml:lang="en"lang="en">
<head>
          <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
          <title>PDO -Jourmoly</title>
</head>
<body>
<table>
          <tr>
                    <th>ID
                    </th><th>Item
          </th></tr>
          <?php
          foreach($itemsas$item)
          {
          ?>
          <tr>
                    <td>< ?phpecho$item['id_item']?></td>
                    <td>< ?phpecho$item['item']?></td>
          </tr>
          < ?php
          }
          ?>
</table>
</body>
</html>
```

¿Y el controlador?

El controlador es el que permite que todo funcione.

Controlador.php

```
< ?php
//Se incluye el modelo
require'modelo.php';

//En $ítems tenemos un arreglo con todos los ítems gracias al modelo

//Ahora la vista recibe dicho arreglo para mostrarlo por pantalla
require'vista.php';
?>
```

Por último se tendrá un fichero más,index.php, que lo único que hará es incluir algunas variables de configuración y el controlador. Es decir, para ver el resultado del script entraremos por index.php.

Casi todas las peticiones a una aplicación seguirán este patrón básico.

2.4 Frameworks MVC.

Los patrones MVC cumplen perfectamente el fin particular de cualquier frameworks (proveer una estructura bien definida que de soporte a un proyecto web que ayude a que el proyecto sea organizado y bien desarrollado).

Ventajas de los Frameworks MVC [4].

El uso de los frameworks basados en este patrón permite tener una separación lógica y física de los componentes de la aplicación, ya que por un lado se tienen los modelos, por otro las vistas y por otro los controladores. De esta forma, los desarrolladores de la aplicación pueden centrarse en la parte que les toca, ya sea como diseñadores en las vistas, o como programadores de los modelos del negocio. Los frameworks ofrecen una elevada organización en el trabajo, ya que todo parece tener un sitio, aunque siempre existen cosas que son difíciles de acomodar, pero generalmente se obtiene mucha más organización que cuando se hace el layout de carpetas y la organización de los archivos manualmente. Generalmente estos frameworks poseen generadores que crean los archivos base de los modelos o vistas, para no tener que crear cada archivo relacionado a mano.

Los frameworks MVC tienen como desventaja o limitación que el manejo de flujos de tareas tiene que hacerse a mano, o sea, codificando los flujos directamente.

Algunos de los frameworks MVC más utilizados son:

Tabla1. Frameworks MVC

Lenguaje	Licencia	Nombre
Ruby	MIT	Ruby OnRails
Java / J2ee	Apache	Struts
.NET	Microsoft Patterns&Practices	User Interface Process (UIP) Application Block
AS3	Adobe Open Source	Cairngorm
Perl	GPL	Catalyst
PHP	LGPL	Agavi
Python	Varias	Turbogears
.NET	Microsoft Pre-Release Software	ASP.NET MVC

2.5 Utilidad del patrón.

El patrón de diseño Modelo-Vista-Controlador se utiliza para el diseño de aplicaciones con interfaces complejas. La lógica de una interfaz de usuario cambia con más frecuencia que los almacenes de datos y lalógica de negocio. Se trata de realizar un diseño que desacople la vista del modelo, con la finalidad de mejorar la reusabilidad de las partes.

De esta forma las modificaciones en las vistas impactan en menor medida en la lógica de negocio o de datos.

El patrón MVC se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página; el modelo es el Sistema de Gestión de Bases de Datos y la Lógica de negocio; y el controlador es el responsable de recibir los eventos de entrada desde la vista.
Una de las dificultades con las que debe lidiar la implementación del patrón es el hecho de que es posible incorporar en las clases de la vista gran parte o todo el procesamiento de eventos. Con lo que el controlador puede quedar semioculto dentro de la vista [5].

CONCLUSIONES

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.

REFERENCIAS

1. “Modelo Vista Controlador”, disponible en: http://www.neleste.com/modelo-vista-controlador/

2. Catalani, Exequiel.: “Arquitectura Modelo/Vista/Controlador”, disponible en: http://exequielc.wordpress.com/2007/08/20/arquitectura-modelovistacontrolador/.

3. “Patrón Modelo-Vista-Controlador“, disponible en: Http://www.proactiva-calidad.com/java/patrones/mvc.html

4. Figueroa, José.: “Frameworks MVC de desarrollo Web“, disponible en:http://blog.buhoz.net/blog1.php/2008/03/06/frameworks-mvc-de-desarrollo-web

5. “Modelo Vista Controlador“, disponible en: http://es.wikipedia.org/wiki/Modelo_Vista_Controlador

6. “Lógica de Negocio“, disponible en http://es.wikipedia.org/wiki/L%C3%B3gica_de_negocio

7. “Framework“, disponible en http://es.wikipedia.org/wiki/Framework“

8. Introducción a MVC con PHP, Primera Parte“¨, disponible en http://www.jourmoly.com.ar/introduccion-a-mvc-con-php-primera-parte/