Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

Rev : 2432

IIC1222 – Programación Avanzada

Tutorial 10 - Modelo Vista Controlador

Realizado por Hans Hanckes[1]

1. Introducción

En internet hay mucha información sobre el Modelo Vista Controlador, por lo que el siguiente tutorial tiene como propósito mostrar un resumen de la información que se puede encontrar y explicarla en términos claros y simples. Para los que quieran incursionar en el tema de los patrones de diseño, en internet hay mucha información y existe el ramo Ingeniería de Software en el cual, entre otras cosas, se enseñan varios patrones y aplicaciones de estos.

2. Patrones de Diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

Los patrones de diseño pretenden:

- Proporcionar catálogos de elementos reusables en el diseño de sistemas software.
- Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- Formalizar un vocabulario común entre diseñadores.
- Estandarizar el modo en que se realiza el diseño.
- Facilitar el aprendizaje de las nuevas generaciones de diseñadores condensando conocimiento ya existente.

No pretenden:

- Imponer ciertas alternativas de diseño frente a otras.
- Eliminar la creatividad inherente al proceso de diseño.

No es obligatorio utilizar los patrones, solo es aconsejable en el caso de tener el mismo problema o similar que soluciona el patrón, siempre teniendo en cuenta que en un caso particular puede no ser aplicable. Abusar o forzar el uso de los patrones puede ser un error.

Los patrones se pueden categorizar de la siguiente forma según la escala o nivel de abstracción que tengan:

- Patrones de arquitectura: Aquéllos que expresan un esquema organizativo estructural fundamental para sistemas software.
- Patrones de diseño: Aquéllos que expresan esquemas para definir estructuras de diseño (o sus relaciones) con las que construir sistemas software.

[1] y Juan Wikipedia

- Idiomas: Patrones de bajo nivel específicos para un lenguaje de programación o entorno concreto.

A continuación presentaremos el patrón "Modelo Vista Controlador", un patrón que se puede clasificar como de arquitectura.

3. Presentación v/s Lógica

Durante el curso hemos repetido incansablemente que la lógica del programa debe ser independiente del modelo, es decir el modelo y la vista deben estar lo menos acoplados que se pueda. Para asegurar que esto ocurrá, se creó un patrón de diseño: el Modelo Vista Controlador, que permite separa de la mejor manera posible la lógica de la presentación.

"El Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. 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"[2]

Es importante recalcar que ni Modelo, ni Vista, ni Controlador son una sola clase, sino que cada una puede ser un conjunto de clases (ya sea como proyectos diferentes o como conjuntos funcionales de clases dentro de un mismo proyecto).

3.1. Por qué usar MVC?

Al generar una aplicación nos encontramos con los siguientes problemas:[3]

- Propensión que tienen las interfaces de usuario para cambiar, al extenderla funcionalidad de una aplicación o al portarla a un entorno gráfico diferente
- Construir un sistema flexible es caro y propenso a los errores si la interfaz de usuario estáaltamente entremezclada con el núcleo funcional

Esto ocurre por las siguientes razones:

- La misma información es presentada de maneras diferentes en distintas ventanas.
- La presentación y el comportamiento de una aplicación deben representar los cambios en los datos inmediatamente.
- Los cambios en la interfaz de usuario deben ser sencillos e incluso factibles en tiempo de ejecución.
- Soportar diferentes estándares de interfaces gráficas o portar la interfaz de usuario no debe afectar al código del núcleo (modelo) de la aplicación.

4. Descripción de del MVC

El modelo general de la arquitectura MVC es el siguiente:

De esto se desprende que las restricciones impuestas por MVC son: Controlador conoce a la Vista y al Modelo, Vista conoce al Modelo, Modelo no conoce a nadie.

Las flechas negras continuas representan una asociación directa, y las discontinuas una indirecta (por ejemplo asociados mediante otro patrón o eventos). Para efectos prácticos y en el contexto del curso esto significa:

- El Controlador contiene al Modelo y a la Vista (relación de composición).
- La Vista notifica al Controlador de lo cambios que ocurren para que tome las acciones correspondientes. La notificación es transparente, es decir, la Vista no conoce al Controlador.
- El Modelo es actualizado por el Controlador y puede notificar a la Vista sobre los cambios producidos para que esta se actualice. El Modelo no conoce ni al Controlador ni a la Vista.

Modelo: Representa la lógica del programa, es la representación específica con cual el sistema opera.

[2] http://es.wikipedia.org/wiki/Modelo_Vista_Controlador

[3] http://zarza.fis.usal.es/ fgarcia/docencia/isoftware/05-06/Tema6.zip

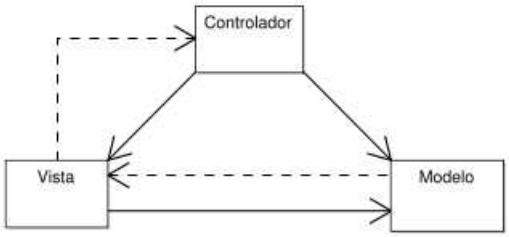

Figura 1: Patrón Modelo Vista Controlador

Vista: Este presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.

Controlador: Este responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista.

Como el MVC es un patrón de arquitectura, este define un esquema organizativo del sistema y no es muy específico en la forma de implementar esta organización, por esto se pueden encontrar diferentes implementaciones de MVC, pero generalmente el flujo que sigue es el siguiente:

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 (por ejemplo, el controlador actualiza el carro de la compra del usuario).
4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. El controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la vista para que se actualice. También el Modelo puede notificar a la vista sobre las actualizaciones que esta debe realizar (ya que el MVC define que el modelo puede notificar a la vista).
5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

En algunas implementaciones la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista, haciendo que el modelo y la vista queden aún más separados.

5. Usando el MVC en C#

Existen muchos frameworks basados en el MVC, en especial los que trabajan sobre plataforma web, por ejemplo: Ruby on Rails (ruby) y Symfony (php), el framework .Net no está diseñado exclusivamente para trabajar con el MVC, pero nosotros podemos generar una aplicación que tenga esa estructura simplemente aplicando el patrón.

Como el MVC es un patrón de arquitectura, define un esquema global del programa y no el detalle particular, por lo tanto existen muchas formas de implementar MVC correctamente, algunas que permiten una mayor separación de la vista y el modelo, son, por otro lado, más extensas y difíciles de implementar.

6. Ejemplo MVC

6.1. Problema Black Jack

Suponga que se tiene el juego Black Jack implementado en C#, se tienen 2 proyectos, la librería de clases que contiene la lógica (modelo) y el proyecto de windows forms que contiene la presentación visual y la lógica para que esto ocurra (vista). Se ha encomendado la tarea de llevar esta aplicación a una arquitectura Modelo Vista Controlador, y se nos ha entregado el diagrama del modelo y el de la vista, presentados a continuación:

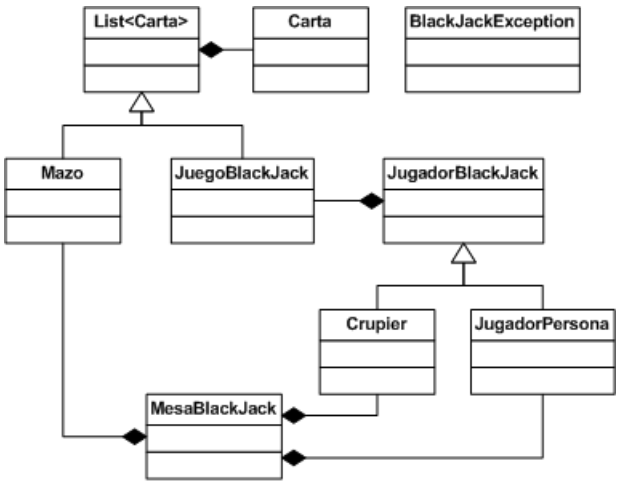

Figura 2: Diagrama Modelo

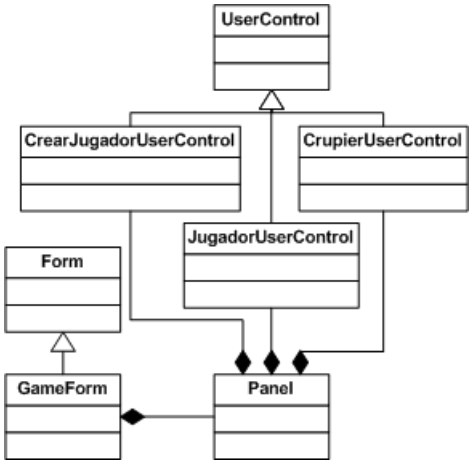

Figura 3: Diagrama Vista

6.2. Solución Black Jack

Como se mencionó anteriormente, para aplicar MVC existen varias posibles soluciones, a continuación detallaremos cómo serían 2 de ellas, pero antes de mostrarla veamos las cosas que serían independiente de la solución aplicada, esto sería:

- Crear un Controlador.
- El Controlador se comunica (contiene) con el Modelo y la Vista.
- La Vista y el Modelo notifican al controlador de los cambios que vayan ocurriendo.

Recordemos que cuando nos referimos al Modelo, Vista o Controlador, no hablamos de una sola clase, sino que cada uno puede ser un conjunto de clases.

6.2.1. Aplicando MVC Black Jack

Creación del Controlador Según MVC, el Controlador puede ser creado en cualquier lugar de la aplicación, en nuestro caso tenemos tres alternativas:

- En el proyecto "librería de clases" que contiene al Modelo.
- En el proyecto "windows foms" que contiene a la Vista.
- Crear un nuevo proyecto.

Comunicación Controlador - Modelo y Controlador - Vista Ahora necesitamos definir cómo será la comunicación del Controlador con el Modelo y del Controlador con la Vista.

Para nuestro ejemplo, el Controlador podría estar compuesto por simplemente una clase (llamada: "Control") y un par de interfaces de comunicación para el Modelo y la Vista.

Controlador - Modelo Si nos fijamos en el diagrama, en el Modelo, la clase MesaBlackJack es la que controla el flujo del juego y tiene el control de los jugadores, es decir es el "administrador" del juego, por lo tanto para dejar lo menos acoplado que se pueda al Controlador del Modelo, la clase "Control" va a tener la única instancia de MesaBlackJack, y de esta forma va a avisar al modelo de los cambios que ocurran (llamando a los métodos de MesaBlackJack) y el modelo va a notificar a Control de lo que ocurra en la lógica (por medio de eventos).

Controlador - Vista En el diagrama de la Vista notamos que tenemos la clase GameForm que es la encargada de controlar la lógica de dentro de la Vista, ya que contiene a los UserControls de los distintos participantes y opciones, por lo tanto para que el Controlador y la Vista queden lo menos acoplados basta con que la clase Control conozca únicamente a el GameForm y que la comunicación sea similar a la que tiene con MesaBlackJack. Además se debería definir una interfaz IVista dentro del Controlador que va a definir lo métodos y eventos que debe tener toda clase que se comunique con el controlador, así de esta forma cualquiera que se le ocurra hacer una Vista para el Modelo, lo único que tiene que hacer es implementar IVista y con eso se asegura la compatibilidad Controlador - Vista.

Ahora dependiendo de dónde creemos el Controlador va a ser el lugar y cómo se debe instanciar, además va a definir las referencias que tienen los proyectos hacia los otros. Por ejemplo si el controlador lo creo en el proyecto "librería de clases", voy a necesitar una referencia desde el proyecto que contiene la Vista hacia el que contiene el Modelo para poder instanciar en el "main" que está en el proyecto "windows forms" a la clase Control y entregarle una IVista, que en este caso sería implementado por GameForm.

Otra Solución I Otra solución podría ser generar 3 interfaces en el Controlador, una para manejar al "Crupier" (IVistaCrupier) otra para el "Jugador" (IVistaJugador) y otra para la Mesa (IVistaGeneral), de esta forma quedaría lo siguiente:

Controlador - Modelo El Controlador necesita conocer al Crupier, JugadorPersona y a la MesaBlackJack. La comunicación entre ellos es hacia un lado por llamas directas y hacia el otro por eventos.

Controlador - Vista El Controlador, definiendo estas tres interfaces, conoce a objetos del tipo IVistaCrupier que la implementa el CrupierUserControl, IVistaJugador que la implementa el JugadorUserControl e IVistaGeneral que la implementa el GameForm.

Otra Solución II La última solución que se va a exponer es una que acople al Modelo con la Vista, esta solución es más simple de programar que las anteriores pero entrega una menor flexibilidad para la generación de nuevas Vistas. La comunicación entre Controlador - Modelo y Controlador - Vista podría ser igual que la Solución I, pero adicionalmente se tiene:

Vista - Modelo La clase JugadorUserControl contiene una referencia a un JugadorPersona, CrupierUserControl contiene una referencia a Crupier y GameForm contiene una referencia a MesaBlackJack, la comunicación entre la Vista y Modelo es tal que el Modelo notifica indirectamente a la Vista (eventos) de los cambios que van ocurriendo en él.