UCH UNIVERSIDAD DE CIENCIAS Y HUMANIDADES

# PROGRAMACIÓN II


**SEMANA 10**

## PATRON MVC

Eric Gustavo Coronel Castillo
ecoronel@uch.edu.pe
gcoronelc.blogspot.com

# PATRON MVC

Uno de los patrones más utilizados en el desarrollo de software, es el patrón Model-View-Controller. Sobre todo en aplicaciones de escritorio se ha hecho muy famoso. En esta lección se verá los fundamentos de este patrón.

## Índice

# 1 INTRODUCCIÓN

## 1.1 ¿QUÉ ES UN PATRÓN DE DISEÑO?

Son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Son soluciones basadas en la experiencia y que se ha demostrado que funcionan.


**Figura 1: Introducción a patrones de diseño.**

## 1.2 ¿POR QUÉ DEBEMOS UTILIZAR PATRONES DE DISEÑO?


**Figura 2: ¿Por qué debemos usar patrones?**

Un patrón de diseño es el trabajo de una persona que ya se encontró con el problema anteriormente, intentó muchas soluciones posibles, escogió y describió una de las mejores. Y esto es algo que deberíamos aprovechar.

En otras palabras, brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares. Debemos tener presente los siguientes elementos de un patrón:

- Eu nombre.
- El problema (cuando aplicar un patrón).

- La solución (descripción abstracta del problema).
- Las consecuencias (costos y beneficios).

## 1.3 CLASIFICACIÓN DE LOS PATRONES

Los patrones se clasifican de la siguiente manera:

- **Patrones Creacionales:** Inicialización y configuración de objetos.
- **Patrones Estructurales:** Separan la interfaz de la implementación. Se ocupan de cómo las clases y objetos se agrupan, para formar estructuras más grandes.
- **Patrones de Comportamiento:** Más que describir objetos o clases, describen la comunicación entre ellos.

En este caso nos ocuparemos del patrón Model View Controller (MVC) que se encuentra dentro del grupo de patrones creacionales.

# 2 PATRÓN: MODEL VIEW CONTROLLER

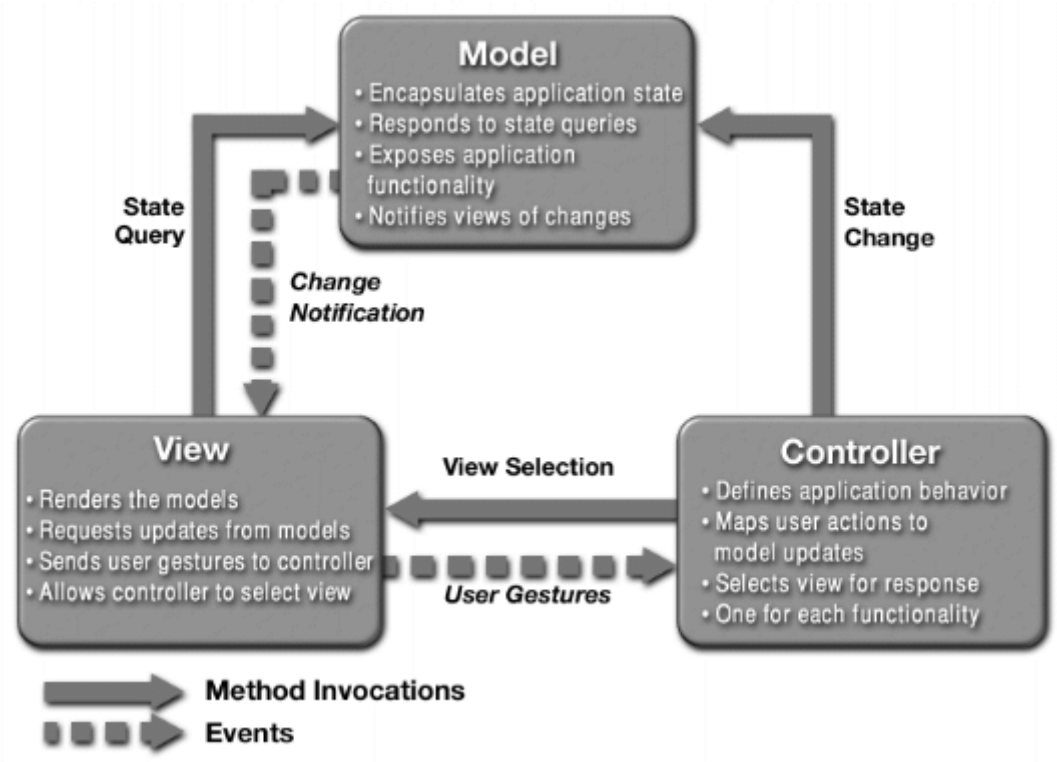


**Figura 3: Patrón MVC**

## 2.1 Introducción

En cada programa podemos encontrar tres responsabilidades principales:

- La lógica de la aplicación (qué hacer).
- Los datos sobre los que se opera (sobre qué datos se opera y cómo se organizan esto).
- La presentación de la aplicación (como se expone a sus usuarios, pueden ser otras aplicaciones).

Cada una de estas responsabilidades puede ser implementada por una o más clases. Normalmente más de una.

## 2.2 Orígenes del patrón MVC

Este patrón fue descrito por primera vez por Trygve Reenskaug en 1979, y la implementación original fue realizada en Smalltalk en los laboratorios Xerox.

El patrón MVC le da nombre a las 3 responsabilidades anteriores:

- Controlador: lógica.
- Modelo: datos.
- Vista: representación.
  - No decimos “interfaz” porque, si bien la GUI es una forma de vista (y probablemente la más común), no toda vista es una GUI.

Además nos dice como debiesen ser las relaciones entre las tres componentes que las implementan:

Se usa (él o alguna de sus variantes) en la gran mayoría de las interfaces de usuario.

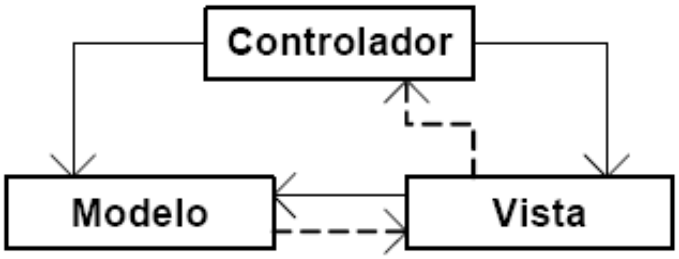


Figura 4: Relación entre los componentes del patrón MVC

## 2.3 El controlador

- El controlador es el que dirige la orquesta de la aplicaci'0n.
  - Este determina que se ha de hacer y cuando.
  - Como se ha de responder ante un evento de la interfaz gráfica (si es que la hay).
  - Como se ha de responder ante un mensaje por la red.
  - Como se han de producir los resultados.
  - Que vistas se han de usar.
  - Etc.

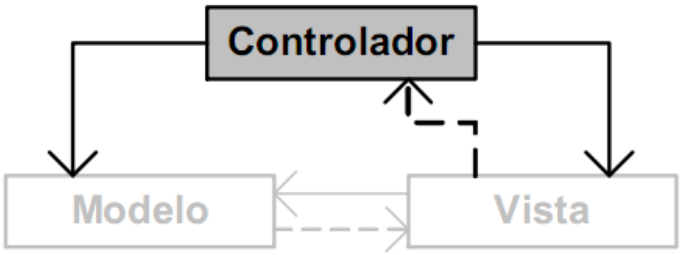


**Figura 5: MVC – El Controlador**

- El controlador conoce tanto al modelo de datos subyacente como a la vista que se está usando.
  - Necesita que así sea, dado que tiene que orquestarlos.
  - Es posible que no conozca directamente a la vista misma, sino que sólo a una interfaz de esta, de tal manera de pueda cambiarla sin problemas.
- Además recibe notificaciones por parte de la vista sobre cambios que han ocurrido.
  - Un usuario que hace click en un botón, por ejemplo.
  - La vista notifica (evento) al controlador, y es éste el que toma la acción que corresponda, potencialmente actualizando al modelo o a la vista.

## 2.4 El modelo

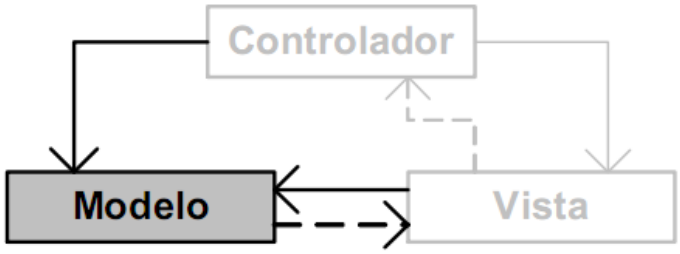


**Figura 6: MVC - El Modelo**

- El modelo es la parte de la aplicación que representa los datos sobre los cuales se va a operar.
  - Este puede ser tan complejo como se necesite para satisfacer las necesidades de representación de los datos.
  - Entrega valor agregado a los datos (por ejemplo, calcular la edad a partir de la fecha de nacimiento).
  - Encapsula cualquier forma de almacenamiento de los datos (por ejemplo, si salen de una base de datos o de un archivo).
- El modelo es actualizado/modificado por el controlador.
- Además es capaz de levantar notificaciones cuando sus datos cambian, de manera de que, quien sea que lo esté observando, se entere de los cambios.
- Nótese que el modelo no conoce ni al controlador ni a la vista.
  - Evitamos acoplamiento innecesario.
- El mismo modelo puede ser compartido por aplicaciones diferentes.

- Por ejemplo, en un juego, tanto el juego mismo como el editor de niveles tienen que poder operar sobre los mismos datos, por lo que tiene sentido que el modelo sea el mismo.
- Como no conoce a nadie, no está amarrado a nada.

## 2.5 La vista

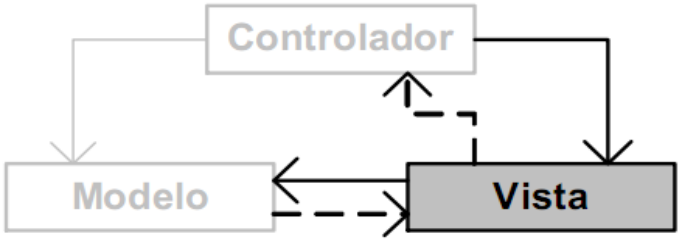


**Figura 7: MVC – La Vista**

- La vista es la frontera de comunicación con el mundo exterior.
  - Es quién se encarga de dar una representación al modelo.
- Normalmente su responsabilidad se confunde con la de la GUI.
  - Aunque no son estrictamente lo mismo, en adelante hablaremos indistintamente de la vista y de la GUI.
- La vista está expuesta al usuario.
  - Este interactúa con ella para lograr las metas que se propone.
  - La vista notifica al controlador.
  - El controlador toma las acciones que corresponda, potencialmente actualizando del modelo de datos.
  - El modelo, de cambiar, notifica (en este caso a la vista) de manera que actualice su estado (si corresponde).

## 2.6 Resumen

- Acoplamiento vista/lógica/datos es un problema.
- Son todas responsabilidades diferentes e independientes.
  - Hay que separarlas.
- El patrón MVC nos dice cómo separarlas.

- Y quienes tienen que conocer a quienes para que la solución sea extensible, escalable, etc.

- El controlador se encarga de la lógica.
  - Es el que define qué se tiene que hacer, cúando y cómo.
- El modelo se encarga de los datos.
  - Es capaz de avisar cuando cambia.
- La vista se encarga de la interacción con el usuario.
  - Muestra una representación del modelo.

# 3 VARIANTES DEL PATRÓN MVC

## 3.1 Variante inicial

Variante en la cual no existe ninguna comunicación entre el Modelo y la Vista y esta última recibe los datos a mostrar a través del Controlador.

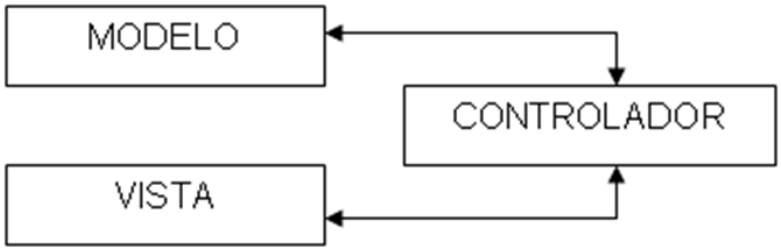


Figura 8: Variante inicial del patrón MVC.

## 3.2 Variante intermedia

Variante en la cual se desarrolla una comunicación entre el Modelo y la Vista, donde esta última para mostrar los datos los busca directamente en el Modelo, dada una indicación del Controlador, disminuyendo el conjunto de responsabilidades de este último.

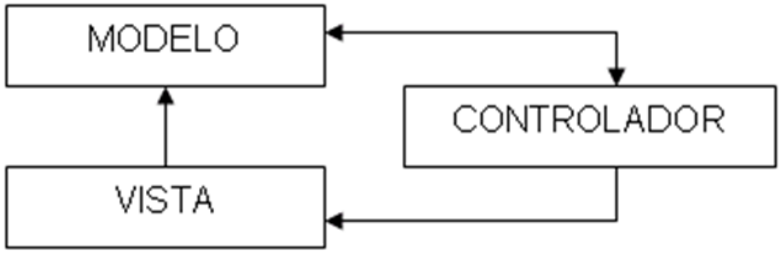


**Figura 9: Variante intermedia del patrón MVC.**