CURSO: HERRAMIENTAS DE PROGRAMACIÓN

Sesion7: Patron de Diseño MVC

Instructor: David Paúl Porras Córdova
@iscodem

Concepto de Patrón de Diseño

“Una arquitectura orientada a objetos bien estructurada está llena de patrones. La calidad de un sistema orientado a objetos se mide por la atención que los diseñadores han prestado a las colaboraciones entre sus objetos. Los patrones conducen a arquitecturas más pequeñas, más simples y más comprensibles”.

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.

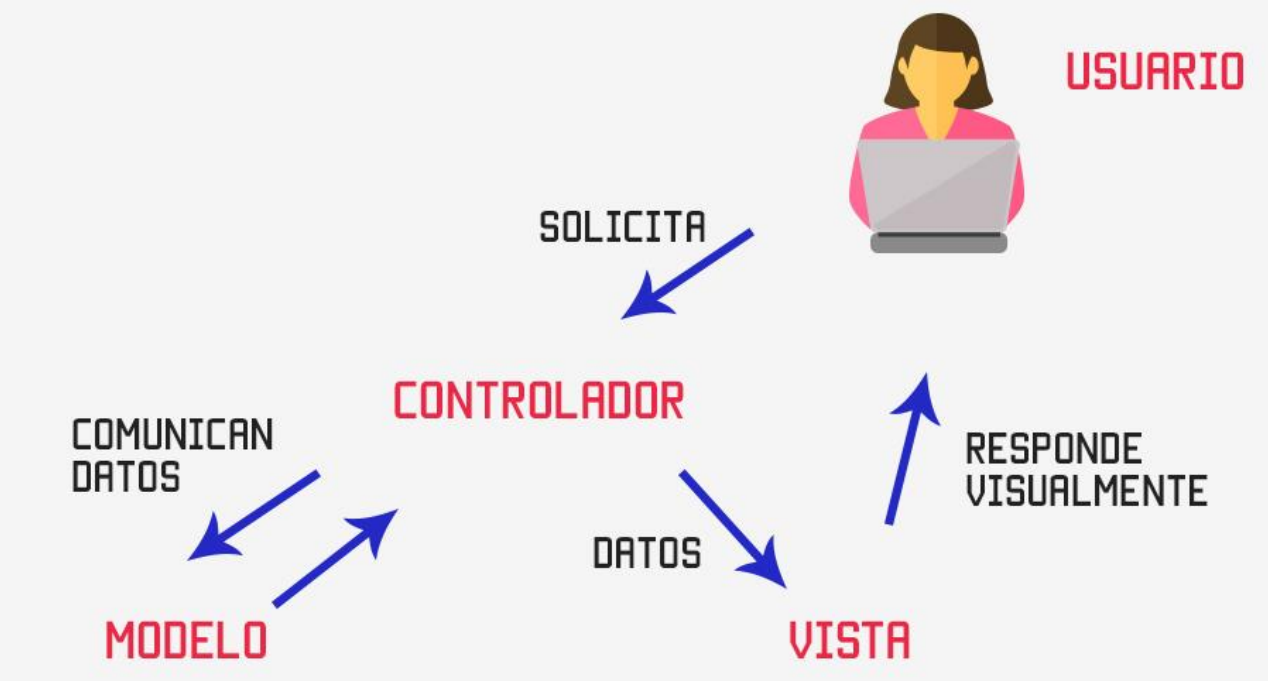
Surge de la necesidad de crear software más robusto con un ciclo de vida más adecuado, donde se potencie la facilidad de mantenimiento, reutilización del código y la separación de conceptos.

Qué es MVC

El MVC o Modelo-Vista-Controlador es un patrón de arquitectura de software que, utilizando 3 componentes (Vistas, Models y Controladores) separa la lógica de la aplicación de la lógica de la vista en una aplicación.

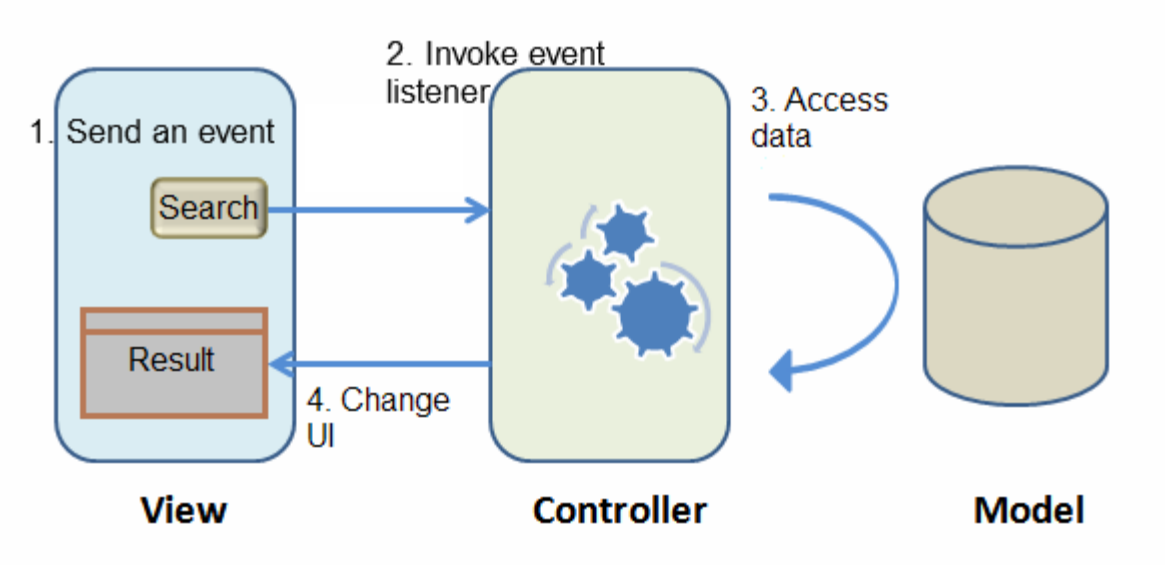
Es una arquitectura importante puesto que se utiliza tanto en componentes gráficos básicos hasta sistemas empresariales; la mayoría de los frameworks modernos utilizan MVC (o alguna adaptación del MVC) para la arquitectura, entre ellos podemos mencionar a Ruby on Rails, Django, AngularJS y muchos otros más. En este pequeño artículo intentamos introducirte a los conceptos del MVC.

Patrón MVC

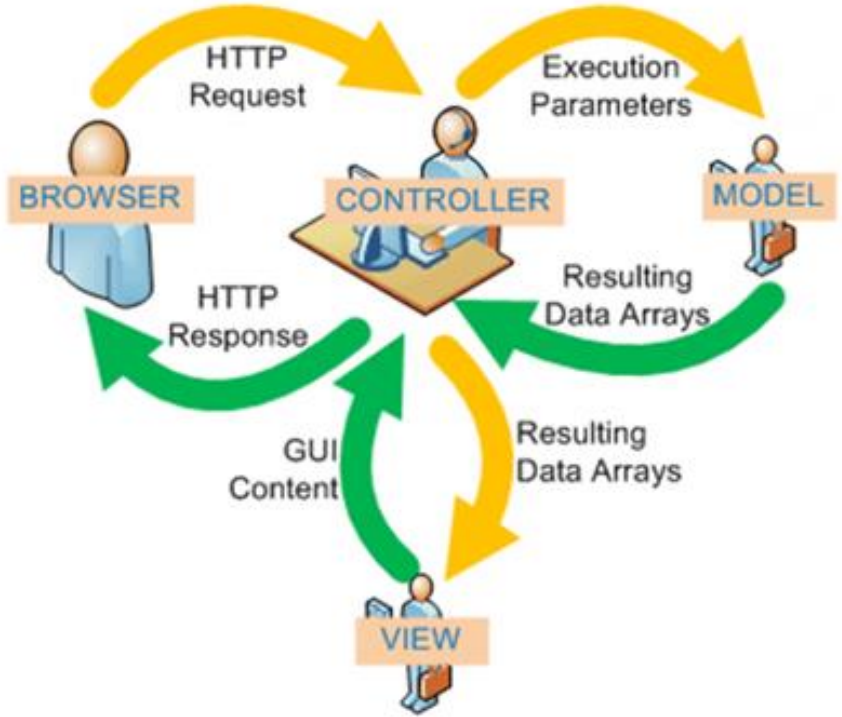

USUARIO
SOLICITA
CONTROLADOR
COMUNICAN
DATOS
RESPONDE
VISUALMENTE
DATOS
MODELO
VISTA

Patrón MVC

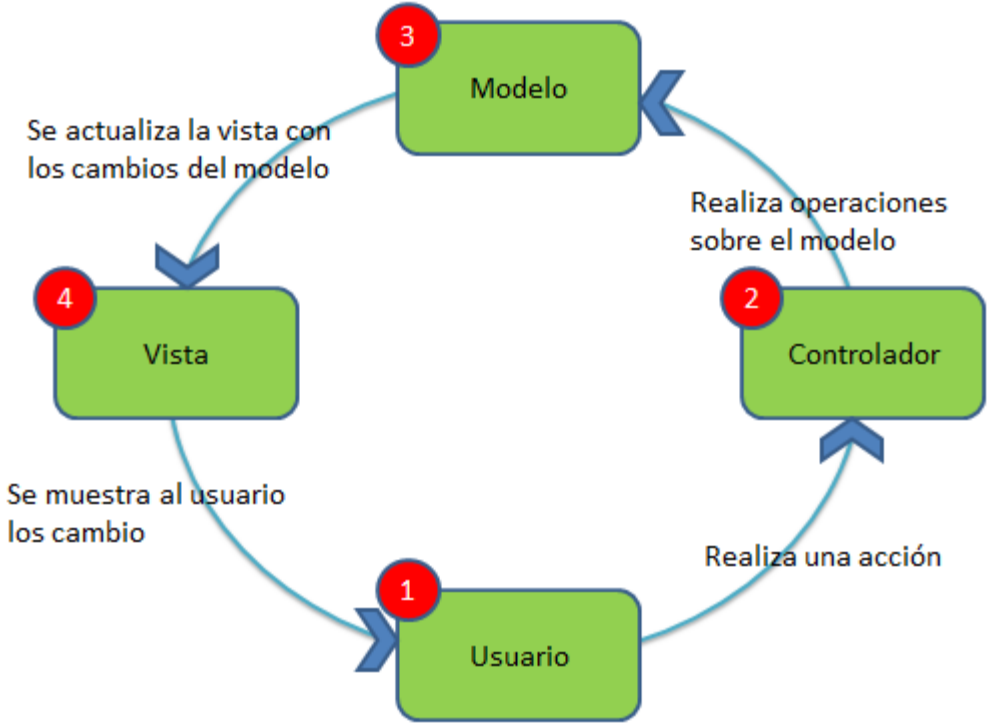
Para el diseño de aplicaciones con sofisticadas interfaces es muy recomendable usar el patrón de diseño **Modelo-Vista-Controlador.**



Qué es MVC


HTTP
Request
Execution
Parameters
BROWSER
CONTROLLER
MODEL
HTTP
Response
Resulting
Data Arrays
GUI
Content
Resulting
Data Arrays
VIEW

Qué es MVC


3
Modelo
Se actualiza la vista con
los cambios del modelo
Realiza operaciones
sobre el modelo
4
Vista
2
Controlador
Se muestra al usuario
los cambio
Realiza una acción
1
Usuario

Qué es MVC

Elementos del patrón MVC

- **Modelo:** Encapsula los datos y las funcionalidades. El modelo es independiente de cualquier representación de salida y/o comportamiento del controlador o la vista.

- **Vista:** Muestra la información a través de una interfaz de usuario. Pueden existir múltiples vistas del modelo. Cada vista tiene asociado un componente controlador con el que interactua.

- **Controlador**: Reciben las entradas de las vistas, usualmente como eventos que codifican los movimientos o pulsación de botones del ratón, pulsaciones de teclas, etc. Los eventos son traducidos a solicitudes de servicio ("**service requests**") para el modelo o la vista.

Una analogía

Una que me gusta mucho es la de la televisión. En tu televisión puedes ver distintos canales distribuidos por tu proveedor de cable o televisión (que representa al modelo), todos los canales que puedes ver son la vista, y tú cambiando de canal, controlando qué ves representas al controlador.

Modelo

Se encarga de los datos, generalmente (pero no obligatoriamente) consultando la base de datos. Actualizaciones, consultas, búsquedas, etc. todo eso va aquí, en el modelo.

Controlador

Se encarga de... controlar, recibe las órdenes del usuario y se encarga de solicitar los datos al modelo y de comunicárselos a la vista.

Vista

Son la representación visual de los datos, todo lo que tenga que ver con la interfaz gráfica va aquí. Ni el modelo ni el controlador se preocupan de cómo se verán los datos, esa responsabilidad es únicamente de la vista.

Ventajas y desventajas del uso del patrón

- La implementación se realiza de forma modular.
- Sus vistas muestran información actualizada siempre. El programador no debe preocuparse de solicitar que las vistas se actualicen, ya que este proceso es realizado automáticamente por el modelo de la aplicación.
- Cualquier modificación que afecte al dominio, como aumentar métodos o datos contenidos, implica una modificación sólo en el modelo y las interfaces del mismo con las vistas, no todo el mecanismo de comunicación y de actualización entre modelos.
- Las modificaciones a las vistas no afectan al modelo de dominio, simplemente se modifica la representación de la información, no su tratamiento.
- MVC esta demostrando ser un patrón de diseño bien elaborado pues las aplicaciones que lo implementan presentan una extensibilidad y una mantenibilidad únicas comparadas con otras aplicaciones basadas en otros patrones.

Desventajas del uso del patrón

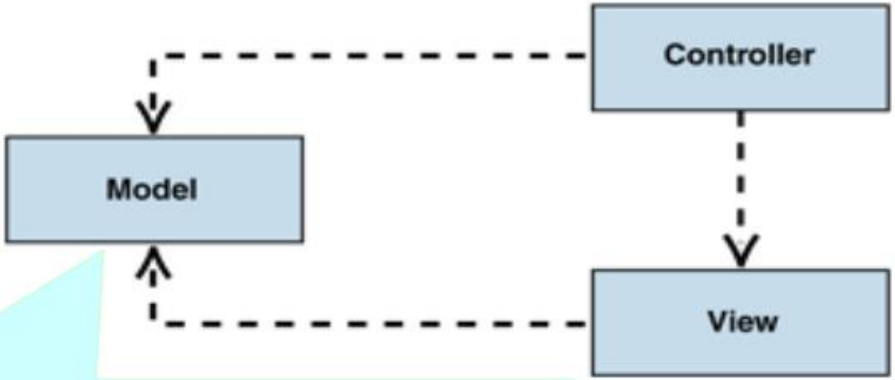
- Para desarrollar una aplicación bajo el patrón de diseño MVC es necesario una mayor dedicación en los tiempos iniciales del desarrollo. Normalmente el patrón exige al programador desarrollar un mayor número de clases que, en otros entornos de desarrollo, no son necesarias. Sin embargo, esta desventaja es muy relativa ya que posteriormente, en la etapa de mantenimiento de la aplicación, una aplicación MVC es mucho más mantenible, extensible y modificable que una aplicación que no lo implementa.
- MVC requiere la existencia de una arquitectura inicial sobre la que se deben construir clases e interfaces para modificar y comunicar los módulos de una aplicación. Esta arquitectura inicial debe incluir, por lo menos, un mecanismo de eventos para poder proporcionar las notificaciones que genera el modelo de aplicación; una clase Modelo, otra clase Vista y una clase Controlador genéricas que realicen todas las tareas de comunicación, notificación y actualización que serán luego transparentes para el desarrollo de la aplicación.
- MVC es un patrón de diseño orientado a objetos por lo que su implementación es sumamente costosa y difícil en lenguajes que no siguen este paradigma.

Resumen

Descripción del patrón

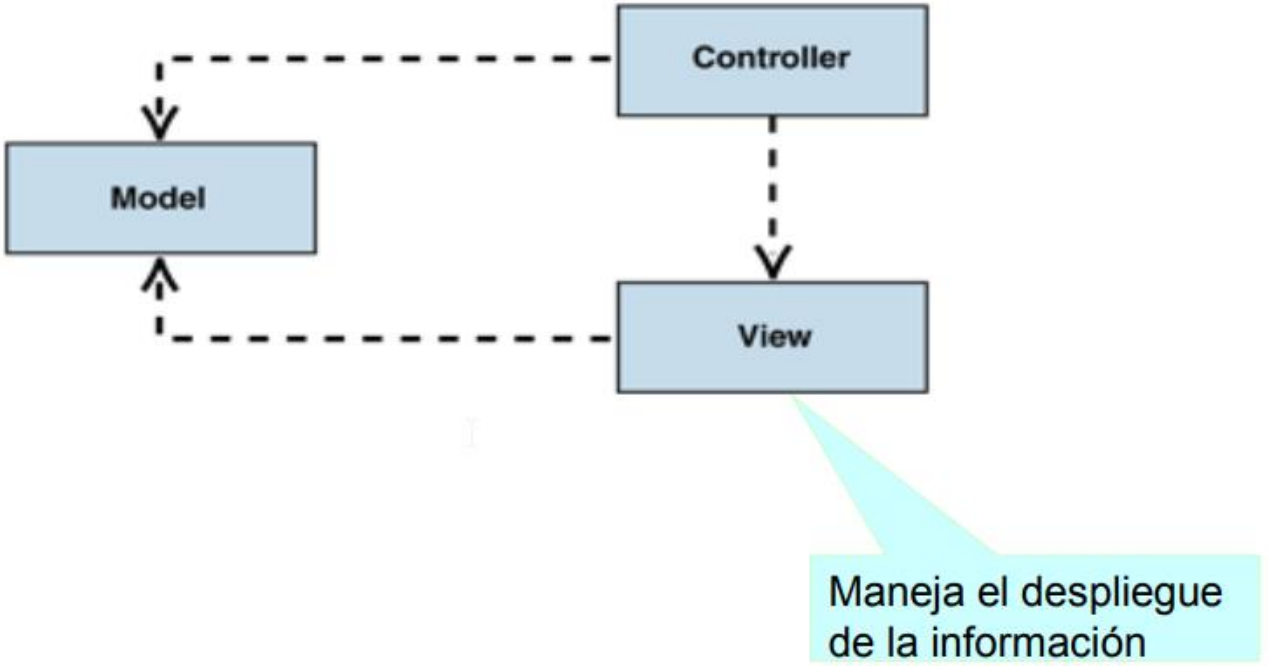
- Problema: Como modularizar la funcionalidad de la interfaz de usuario de una aplicación Web de tal forma que usted pueda modificar fácilmente sus partes individuales?
- Solución: El patrón MVC (Model-View-Controller) separa el modelado del dominio, la presentación y las acciones basados en las entradas del usuario en tres clases apartes. [Burbeck92]

Resumen



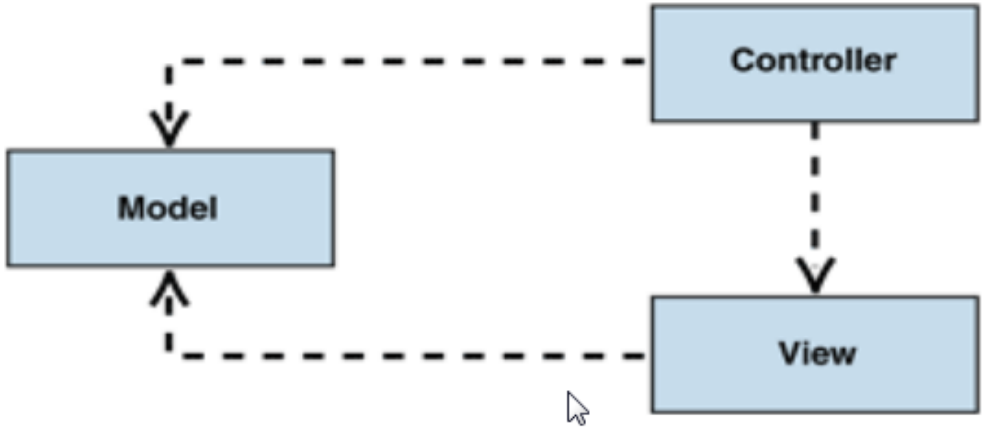
Maneja el comportamiento y los datos del dominio de la aplicación, responde a los requerimientos de información acerca de su estado (usualmente desde la vista) y responde a las instrucciones para cambiar de estado (usualmente desde el controlador)

Resumen


Controller
Model
View
Maneja el despliegue de la información

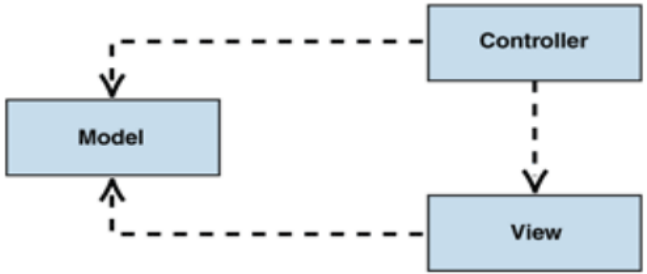
Resumen

Interpreta las acciones del usuario de teclado y ratón, informando al modelo y/o a la vista para cambiar apropiadamente sus estados.



Resumen

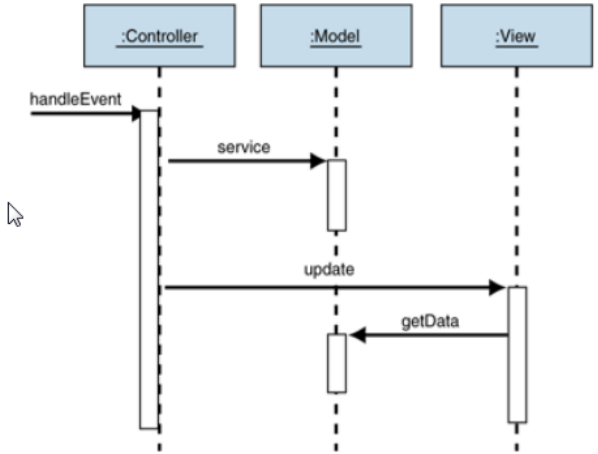
Dependencias



- Tanto la vista como el controlador dependen del modelo. Sin embargo, el modelo no depende ni de la vista ni del controlador.
- La separación permite que el modelo sea construido y probado independientemente de la presentación visual.
- La separación entre vista y controlador es secundaria en muchas aplicaciones, sin embargo en las aplicaciones Web la vista (el navegador) y el controlador (los componentes del lado servidor) están bien definidos.

Resumen

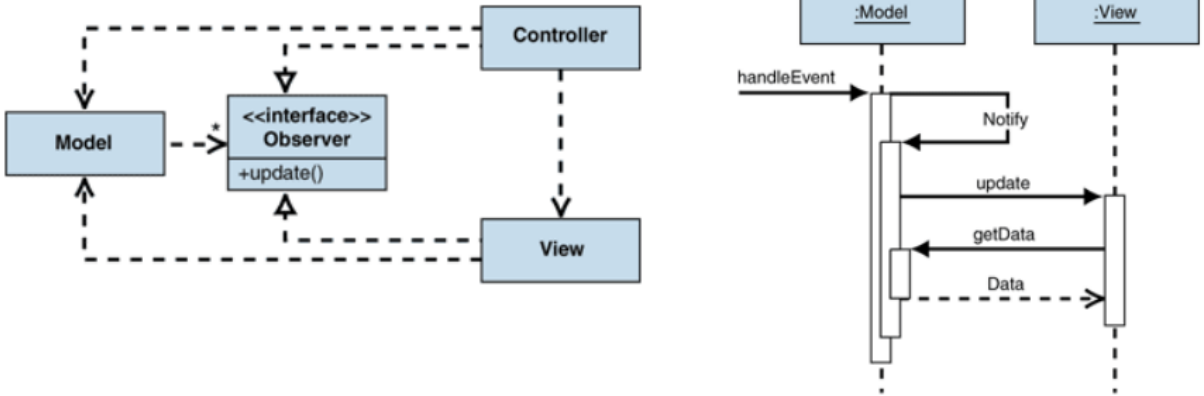
Comportamiento MVC (pasivo)



•Es utilizado cuando un controlador manipula el modelo exclusivamente
•El controlador modifica el modelo y le informa a la vista que este ha cambiado y debe ser refrescada.
•En este escenario el modelo es completamente independiente de la vista y del controlador

Resumen

Comportamiento MVC (activo)



•Es usado cuando el modelo cambia de estado sin la intervención del controlador, lo cual puede pasar cuando otras fuentes están cambiando los datos y los datos deben reflejarse en la vista.
•Debido a que solo el modelo detecta los cambios a su estado interno cuando estos ocurren, el modelo deberá notificar a la vista para refrescarla, pero esto crearía una dependencia entre el modelo y la vista, lo cual iría en contra de uno de los principios del patrón *MVC*.
•Como solución, se introduce el patrón *Observer*, el cual provee un mecanismo para alertar a otros objetos de cambios de estado sin introducir dependencias entre ellos.

Resumen

Ejemplo MVC: Vista Lógica de Arquitectura