INGENIERÍA DE APLICACIONES

Modelo Vista Controlador

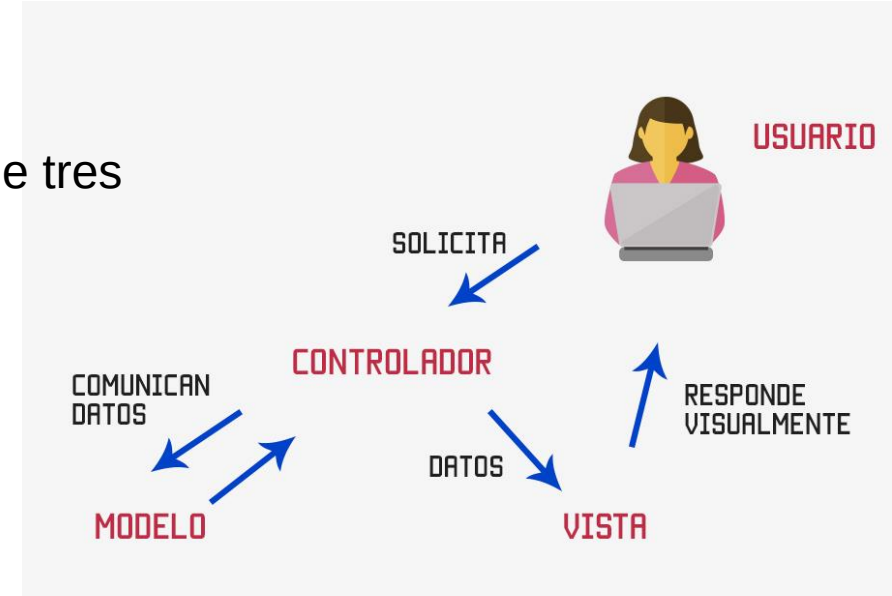
Dra. María Luján Ganuza
mlg@cs.uns.edu.ar
DCIC - Depto. de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur, Bahía Blanca
2019

MVC

Modelo-vista-controlador (MVC) es un patrón de arquitectura de software, que **separa los datos y la lógica** de una aplicación **de su representación y el módulo encargado de gestionar los eventos y las comunicaciones**.

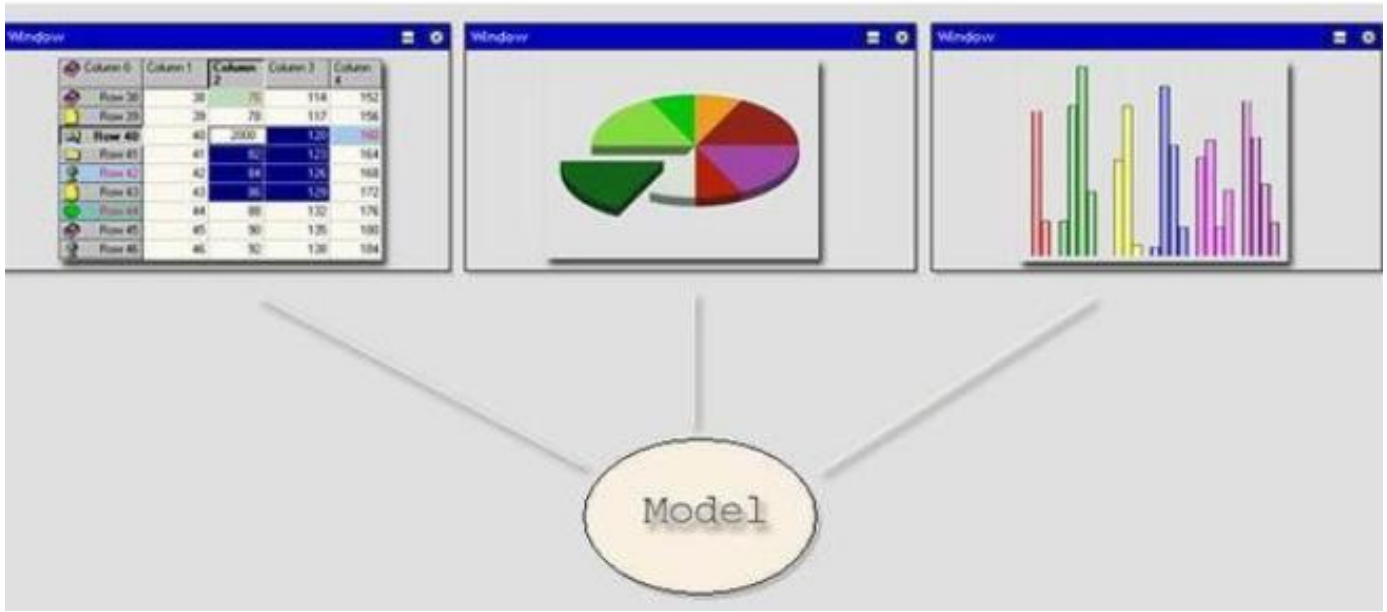
Para ello, MVC propone la construcción de tres componentes distintos:

- el **modelo**,
- la **vista** y
- el **controlador**



MVC

- Para un mismo modelo de datos se pueden tener **varias vistas diferentes**.



MVC

- Para un mismo modelo de datos se pueden tener **varias vistas diferentes**.
- Las vistas y comportamiento deben reflejar las manipulaciones de los datos de forma **inmediata**.
- Permitir diferentes estándares de interfaz de usuario o portarla a otros entornos no debería afectar al código de la aplicación.
- El MVC es utilizado con mayor frecuencia en aplicaciones web.

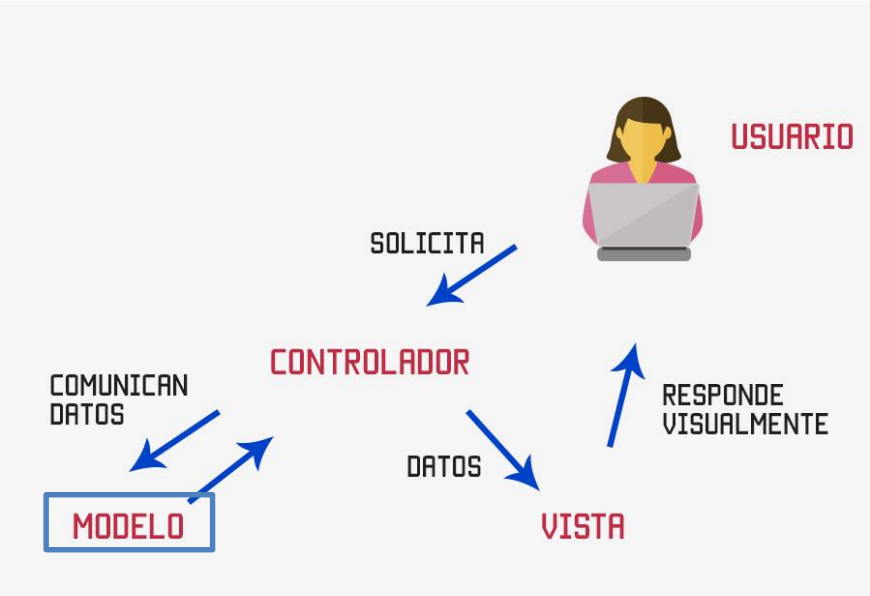
MVC – El Modelo

Es la **representación de la información** con la cual el sistema opera.

Gestiona todos los accesos a dicha información (consultas, actualizaciones, etc.)

Envía a la **vista** la información solicitada para ser mostrada.

Las peticiones de acceso o manipulación de información llegan al **modelo** a través del **controlador**.

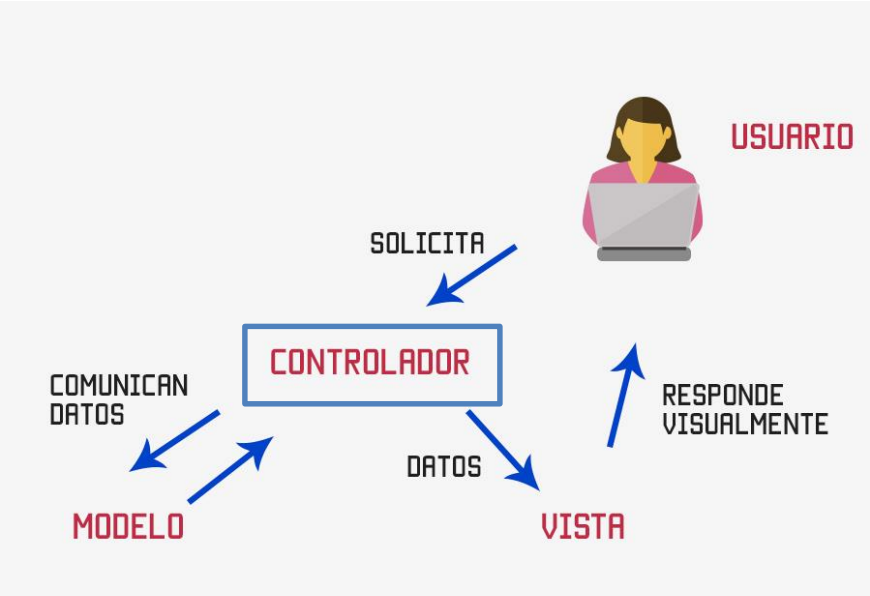


MVC – El Controlador

Responde a eventos (usualmente acciones del usuario) e invoca peticiones al **modelo** cuando se hace alguna solicitud sobre la información.

También puede enviar comandos a la **vista** si se solicita un cambio en la forma en que se presenta el **modelo**.

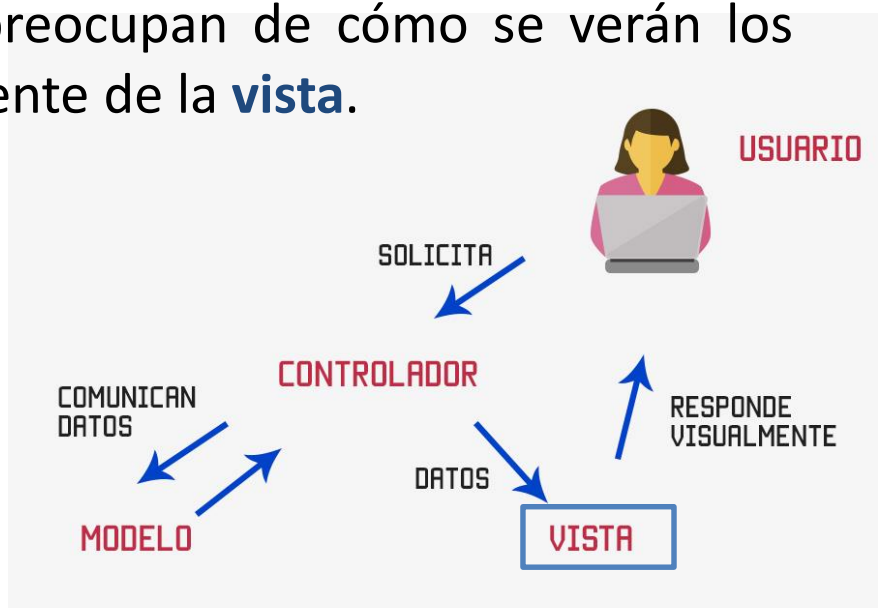
El **controlador** hace de intermediario entre la **vista** y el **modelo**.



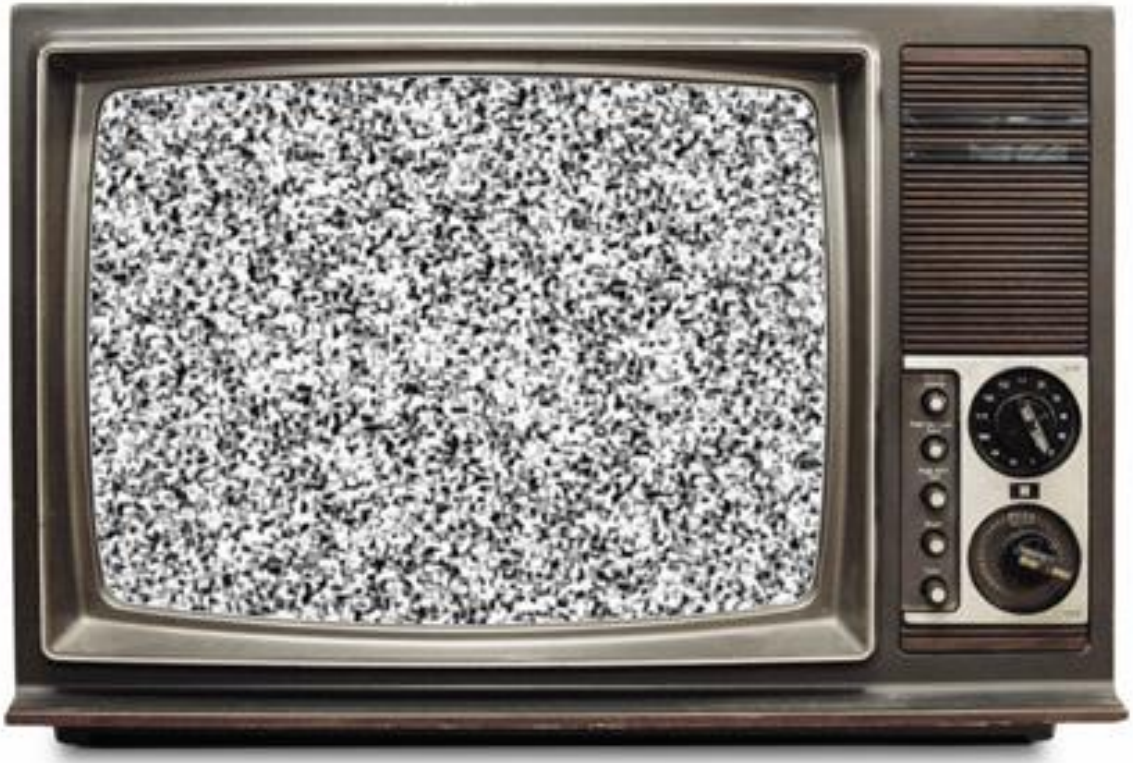
MVC – La Vista

Presenta el **modelo** (información y lógica de negocio) en un formato adecuado para **interactuar** (usualmente la interfaz de usuario).

Ni el **modelo** ni el **controlador** se preocupan de cómo se verán los datos, esa responsabilidad es únicamente de la **vista**.



MVC – Analogía



MVC

¿Por qué se utiliza MVC?

Para separar los componentes de nuestra aplicación dependiendo de la responsabilidad que tengan.

Por ejemplo, si modificamos nuestra Base de Datos, sólo deberíamos modificar el **modelo** que es quién se encarga de los datos y el resto de la aplicación debería permanecer intacta.

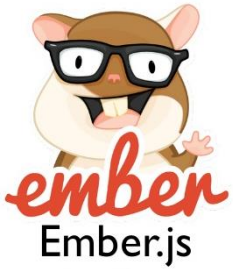
MVC

Flujo de Control

1. El usuario realiza una acción en la interfaz
2. El controlador trata el evento de entrada
3. El controlador notifica al modelo la acción del usuario, que puede implicar un cambio del estado del modelo.
4. Se genera una nueva vista. La vista toma los datos del modelo (el modelo no tiene conocimiento directo de la vista).
5. La interfaz de usuario espera otra interacción del usuario, que comenzará un nuevo ciclo.

MVC

Muchos Frameworks utilizan MVC

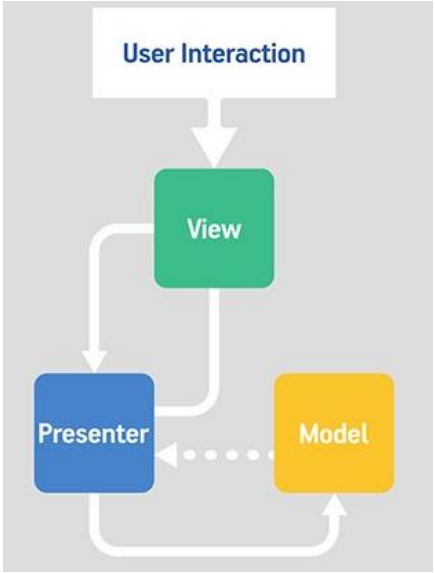


MVP

El **Modelo-Vista-Presentador (MVP)** surge para ayudar a realizar pruebas automáticas de la interfaz gráfica.

La idea es **codificar la interfaz de usuario lo más simple posible**, teniendo el menor código posible.

En su lugar, toda la lógica de la interfaz de usuario, se hace en una clase separada (que se conoce como **Presentador**), que no dependa en absoluto de los componentes de la interfaz gráfica y que, por tanto, es más fácil de testear.

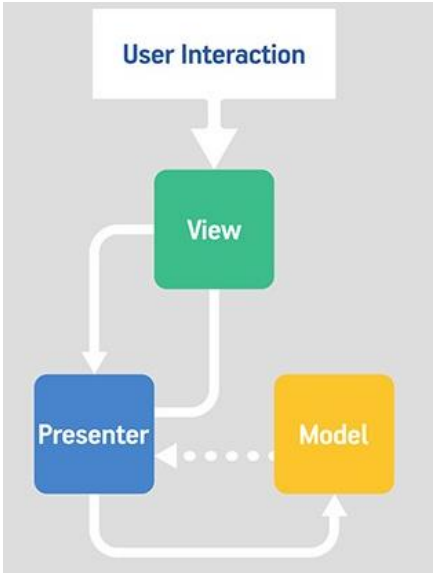


MVP

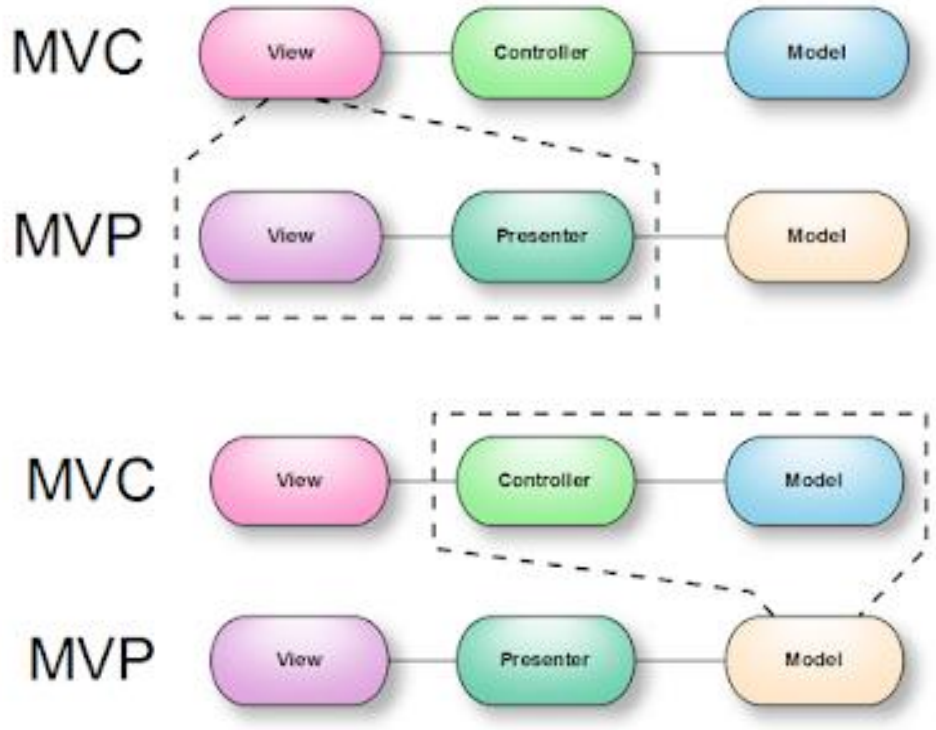
El **modelo** representa el modelo de negocio, tanto los datos como la lógica de negocio.

La clase **presentador** haga de intermediario entre la vista y el modelo de datos.

La **vista** recibe los datos ya “mascados” (una lista de cadenas por ejemplo, en vez del modelo…), y únicamente debe meter esos datos en los componentes gráficos (cajas de texto, checkbox, etc).



MVC y MPV


MVC
View
Controller
Model
MVP
View
Presenter
Model
MVC
View
Controller
Model
MVP
View
Presenter
Model

Material Bibliográfico

- Meyer, B., *Object Oriented Software Construction*. ISE, Inc. 2nd Ed., 1997.

- Abadi, M., & Cardelli, L. A theory of objects. Springer Science & Business Media, 2012.

- Szyperski, C., Component Software: Beyond Object Oriented Programming. AddisonWesley, 2nd Ed., 2011

- Zamir, S., Ed., Handbook of Object Oriented Technology. CRC Press, 2000.