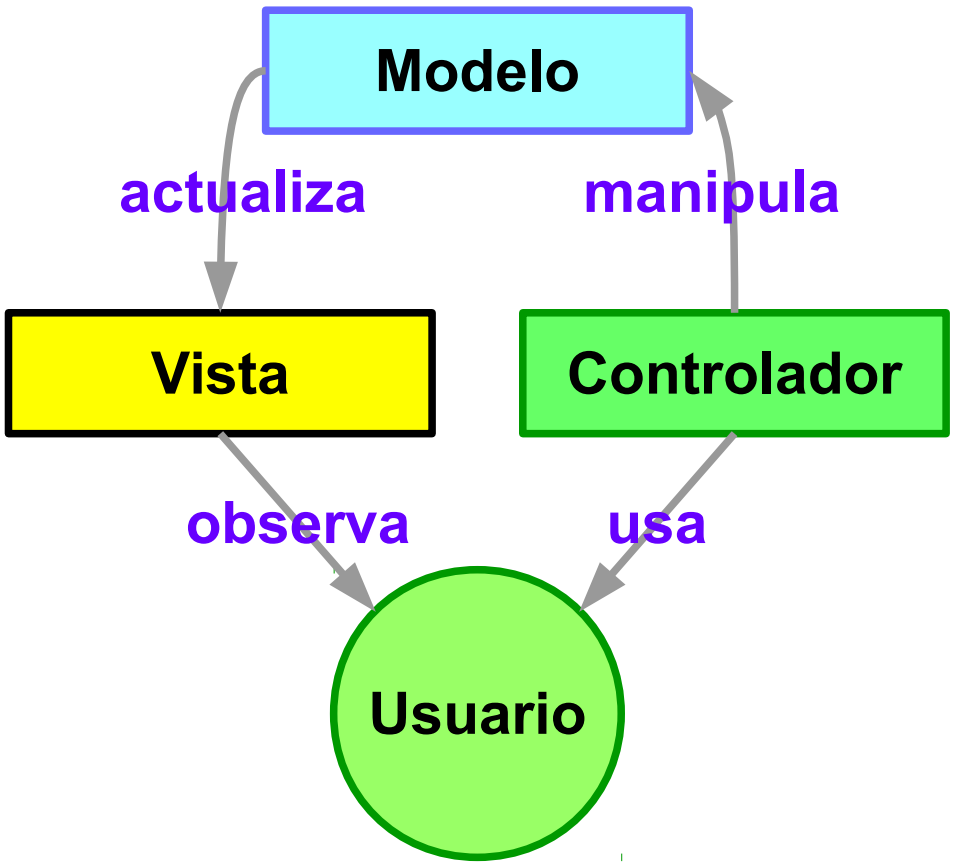
El Patrón MVC
(Modelo – Vista – Controlador)

El patrón MVC: Modelo – Vista - Controlador



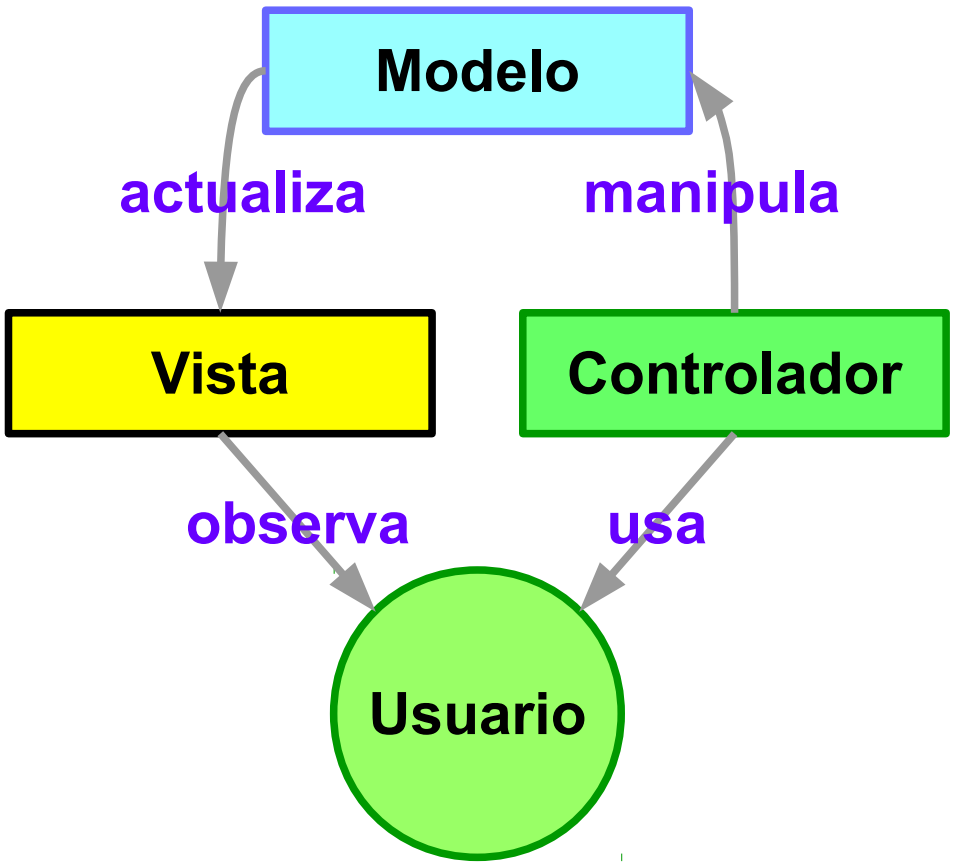
Es un **patrón de arquitectura** de las aplicaciones software

Separa la lógica de negocio de la interfaz de usuario

- Facilita la evolución por separado de ambos aspectos
- Incrementa reutilización y flexibilidad

Creado por el profesor Trygve Reenskaug

MVC: Historia



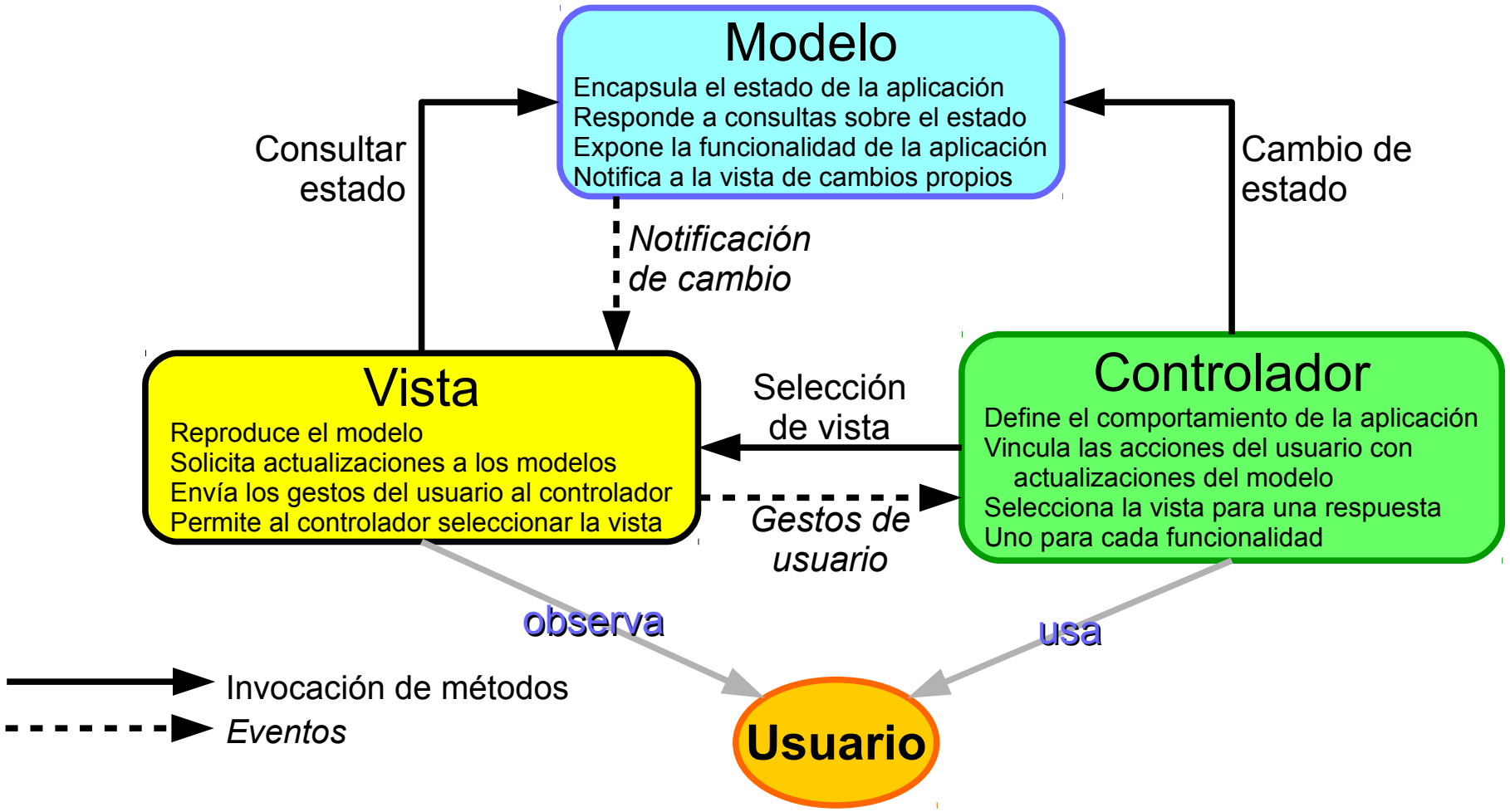
Descrito por primera vez en 1979 para Smalltalk

- http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html

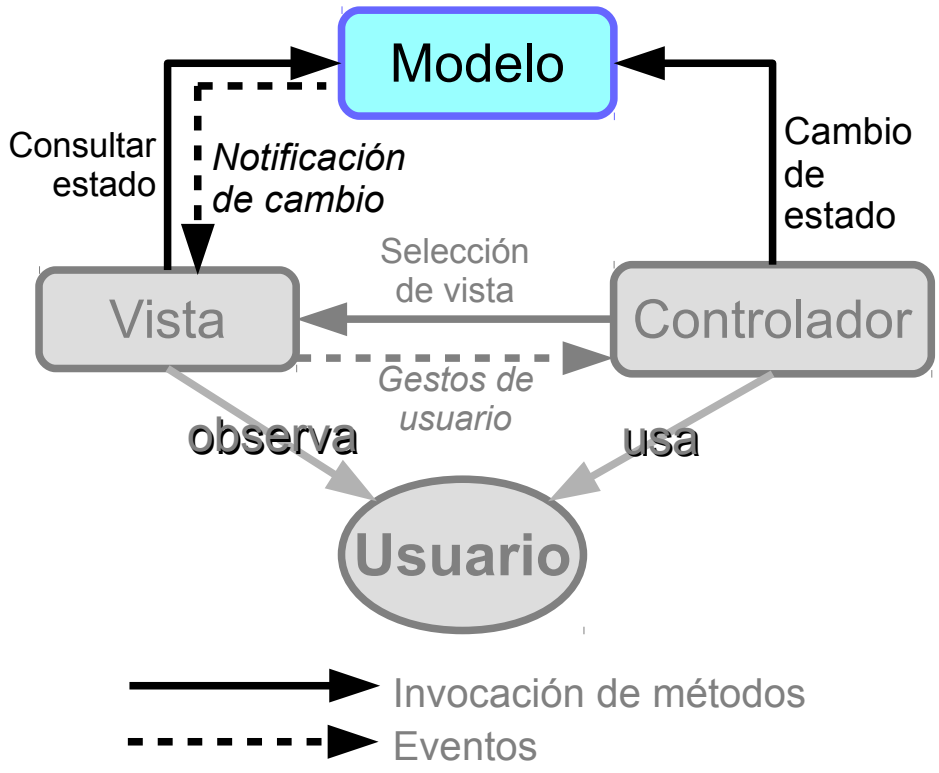
Utilizado en múltiples frameworks

- Java Swing
- Java Enterprise Edition (J2EE)
- XForms (Formato XML estándar del W3C para la especificación de un modelo de proceso de datos XML e interfaces de usuario como formularios web)
- GTK+ (escrito en C, toolkit creado por Gnome para construir aplicaciones gráficas, inicialmente para el sistema X Window)
- ASP.NET MVC Framework (Microsoft)
- Google Web Toolkit (GWT, para crear aplicaciones Ajax con Java)
- Apache Struts (framework para aplicaciones web J2EE)
- Ruby on Rails (framework para aplicaciones web con Ruby)
- Etc., etc., etc.

MVC

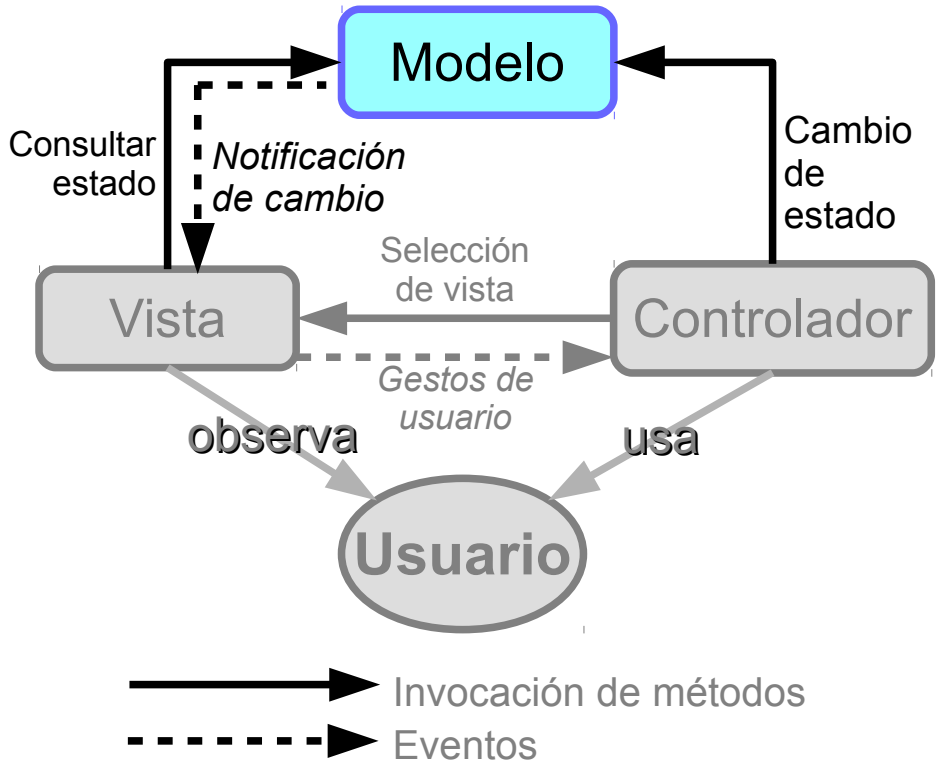

Modelo
Encapsula el estado de la aplicación
Responde a consultas sobre el estado
Expone la funcionalidad de la aplicación
Notifica a la vista de cambios propios
Consultar estado
Notificación de cambio
Cambio de estado
Vista
Reproduce el modelo
Solicita actualizaciones a los modelos
Envía los gestos del usuario al controlador
Permite al controlador seleccionar la vista
Selección de vista
Gestos de usuario
Controlador
Define el comportamiento de la aplicación
Vincula las acciones del usuario con actualizaciones del modelo
Selecciona la vista para una respuesta
Uno para cada funcionalidad
observa
usa
Usuario
Invocación de métodos
Eventos

El modelo



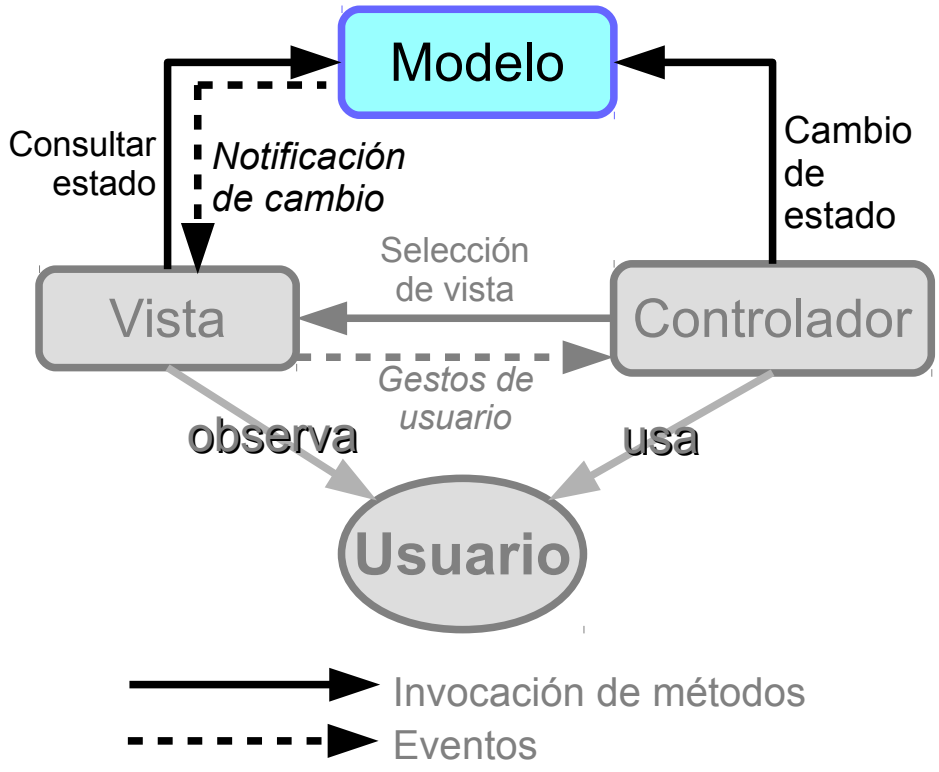
El modelo es un conjunto de clases que representan la información del mundo real que el sistema debe procesar, sin tomar en cuenta ni la forma en la que esa información va a ser mostrada ni los mecanismos que hacen que esos datos estén dentro del modelo, es decir, sin tener relación con ninguna otra entidad dentro de la aplicación.

El modelo



"El modelo desconoce la existencia de las vistas y del controlador". Ese enfoque suena interesante, pero en la práctica no es aplicable pues deben existir interfaces que permitan a los módulos comunicarse entre sí, por lo que SmallTalk (base de la implementación de la librería Swing) sugiere que el modelo en realidad esté formado por dos submódulos: El **modelo del dominio** y el **modelo de la aplicación**.

El modelo



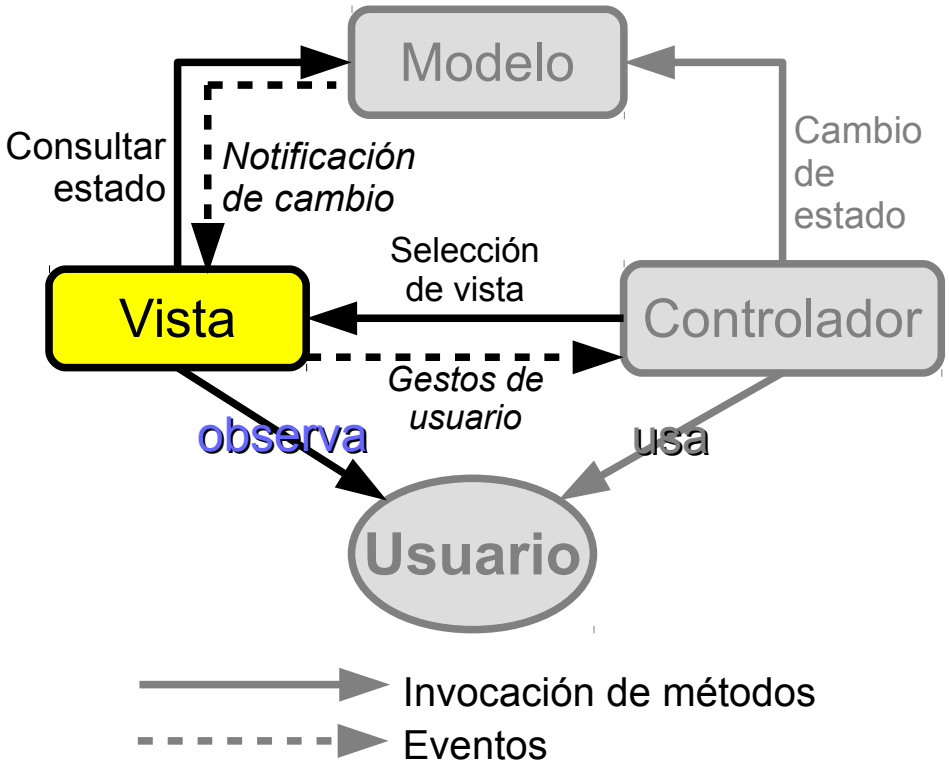
Modelo del dominio

Dominio propiamente dicho. Es el conjunto de clases que un ingeniero de software modela al analizar el problema que desea resolver; por ejemplo: el cliente, la factura, la temperatura, la hora, etc. No debería tener relación con nada externo a la información que contiene.

Modelo de la aplicación

Es un conjunto de clases que se relacionan con el modelo del dominio, que tienen conocimiento de las vistas y que implementan los mecanismos necesarios para notificar a éstas últimas sobre los cambios que se pudieren dar en el modelo del dominio. El modelo de la aplicación es llamado también coordinador de la aplicación.

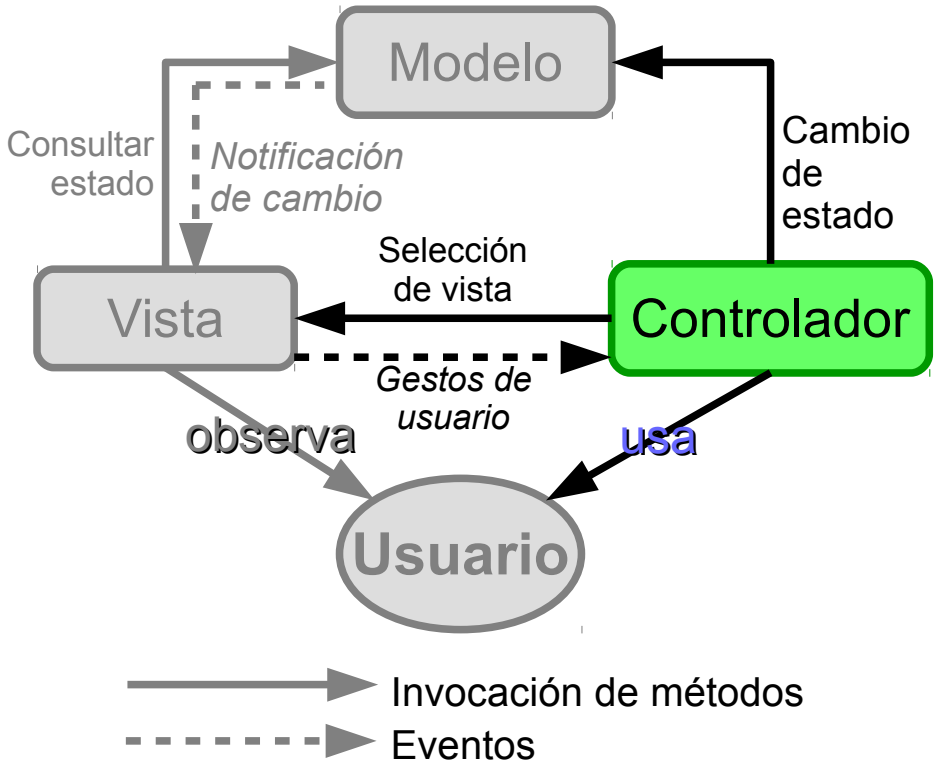
La Vista



Las vistas son el conjunto de clases que se encargan de **mostrar al usuario** la información contenida en el modelo. Una vista está asociada a un modelo, pudiendo existir **varias vistas** asociadas al mismo modelo; así por ejemplo, se puede tener una vista mostrando la hora del sistema como un reloj analógico y otra vista mostrando la misma información como un reloj digital.

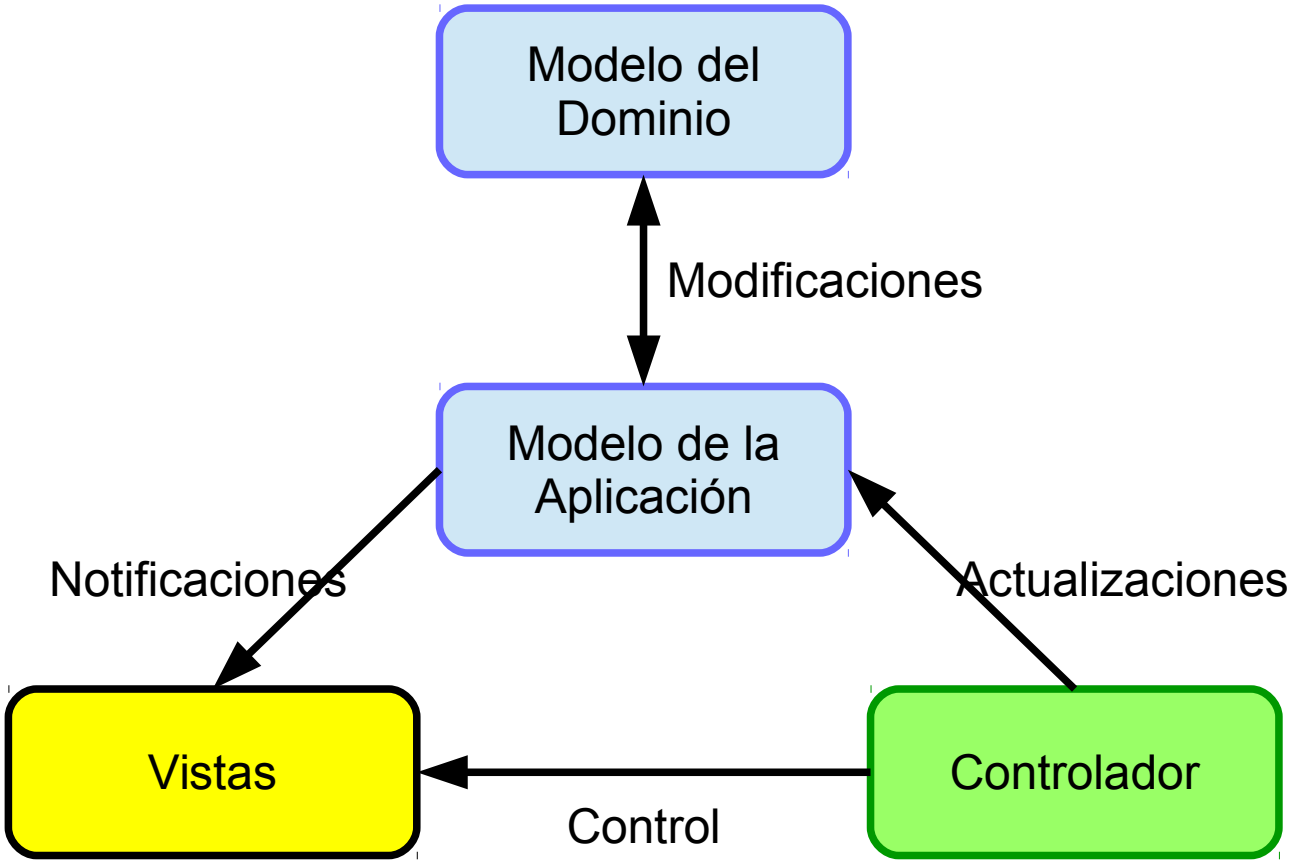
Una vista obtiene del modelo solamente la información que necesita para desplegar y **se actualiza cada vez que el modelo del dominio cambia** por medio de notificaciones generadas por el modelo de la aplicación.

El Controlador

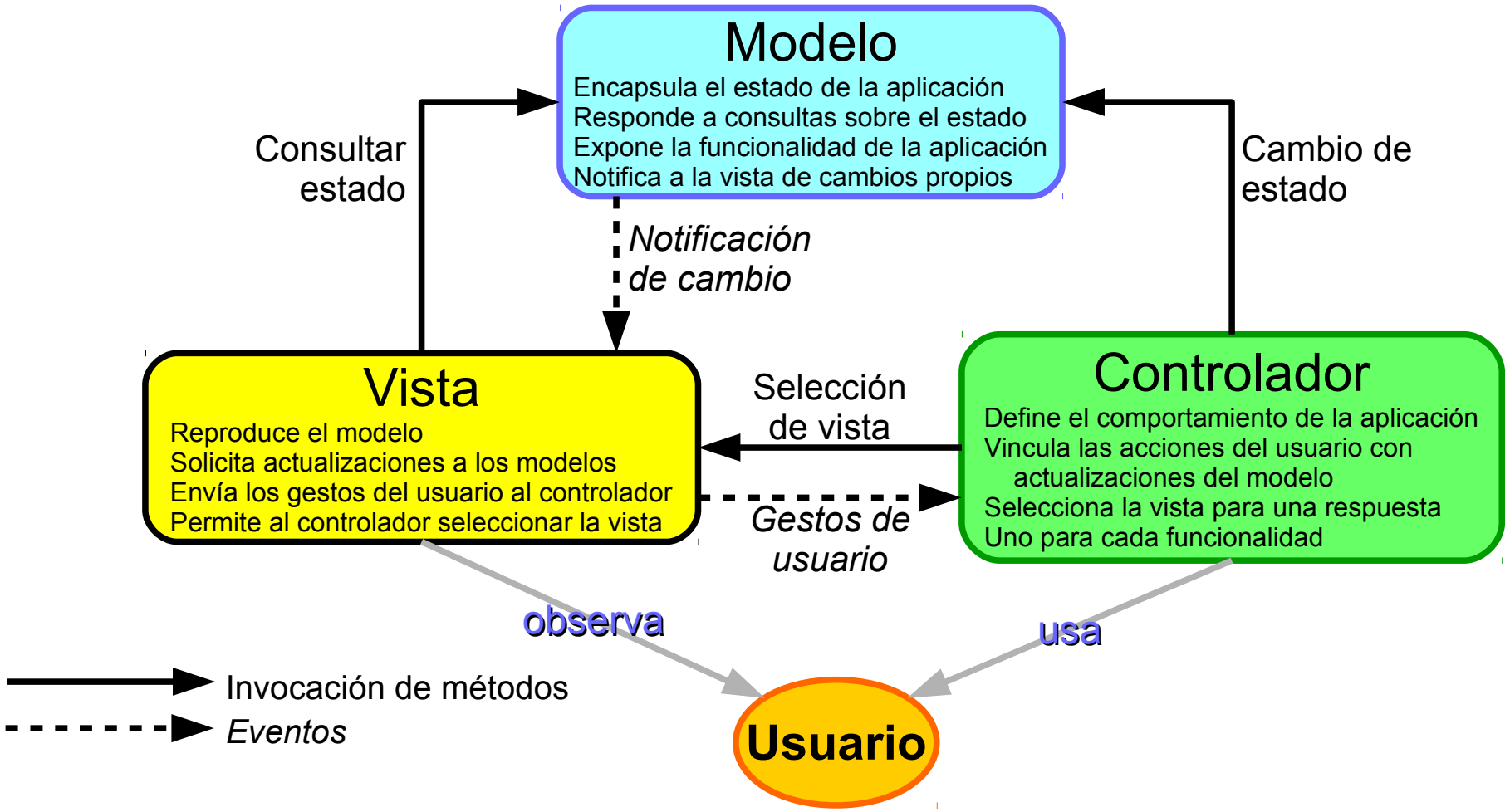


El controlador es un objeto que se encarga de **dirigir el flujo del control de la aplicación** debido a mensajes externos, como datos introducidos por el usuario u opciones del menú seleccionadas por él. A partir de estos mensajes, el controlador se encarga de **modificar el modelo o de abrir y cerrar vistas**. El controlador tiene acceso al modelo y a las vistas, pero las vistas y el modelo no conocen de la existencia del controlador.

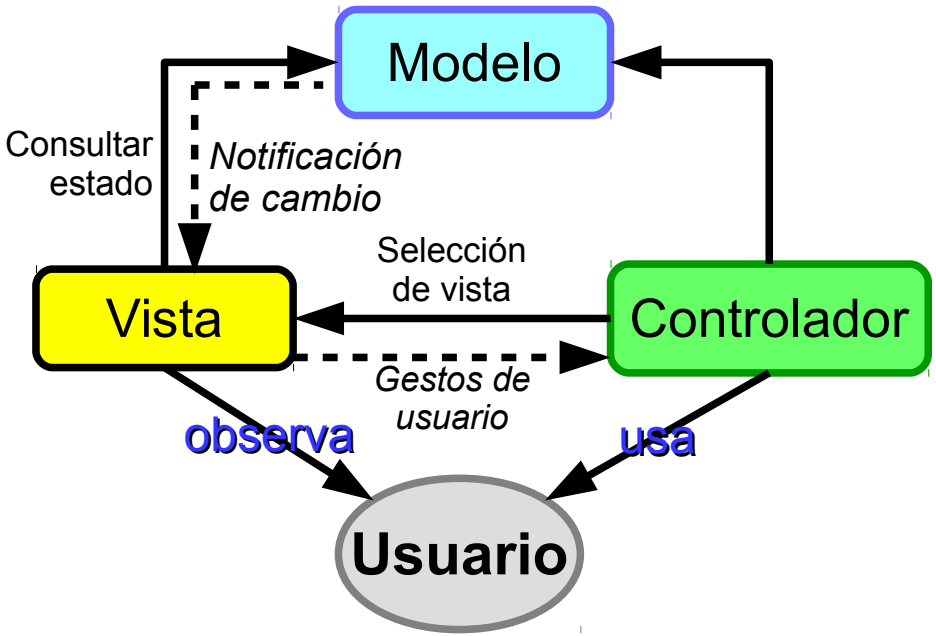
MVC


Modelo del Dominio
Modificaciones
Modelo de la Aplicación
Notificaciones
Actualizaciones
Vistas
Controlador
Control

MVC


Modelo
Encapsula el estado de la aplicación
Responde a consultas sobre el estado
Expone la funcionalidad de la aplicación
Notifica a la vista de cambios propios
Consultar estado
Notificación de cambio
Cambio de estado
Vista
Reproduce el modelo
Solicita actualizaciones a los modelos
Envía los gestos del usuario al controlador
Permite al controlador seleccionar la vista
Selección de vista
Gestos de usuario
Controlador
Define el comportamiento de la aplicación
Vincula las acciones del usuario con actualizaciones del modelo
Selecciona la vista para una respuesta
Uno para cada funcionalidad
observa
usa
Usuario
Invocación de métodos
Eventos

MVC: interacción de los componentes

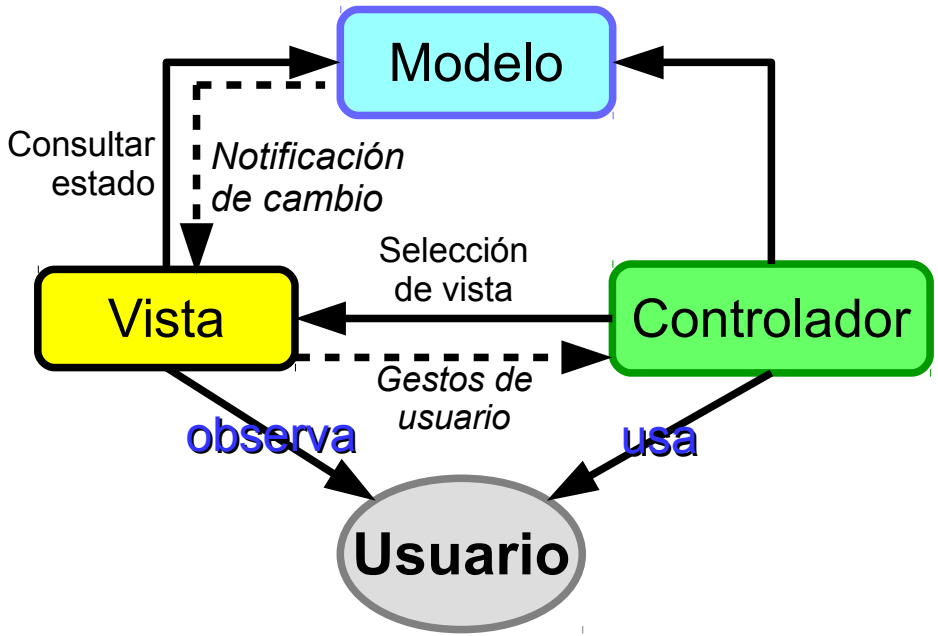


Aunque se pueden encontrar diferentes implementaciones de MVC, el flujo de control que se sigue generalmente es el siguiente:

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

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.

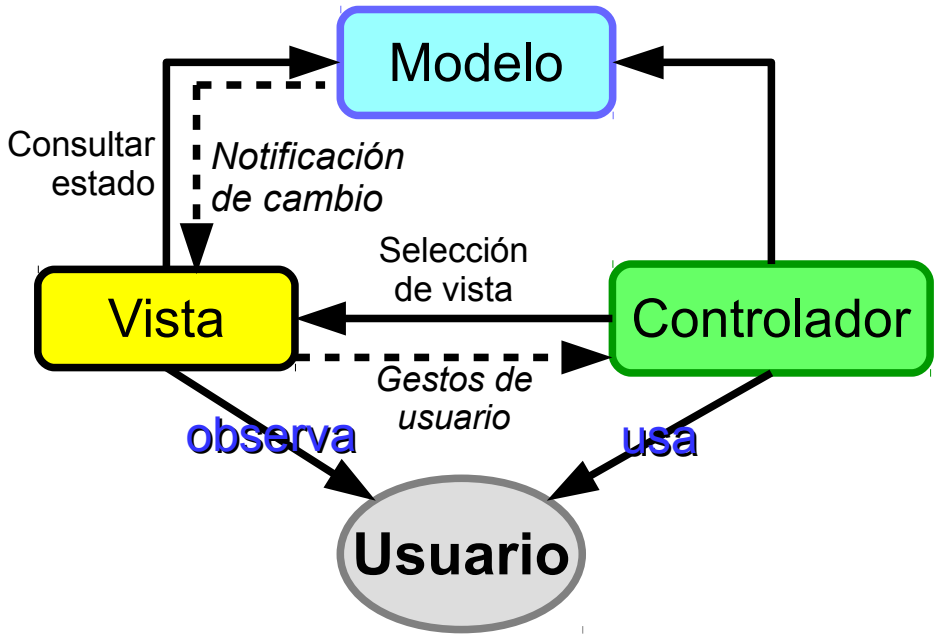
MVC: interacción de los componentes



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). Los controladores complejos están a menudo estructurados usando un *patrón de comando* que encapsula las acciones y simplifica su extensión.

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 reflejan los cambios en el modelo (por ejemplo, produce un listado del contenido del carro de la compra). El modelo no debe tener conocimiento directo sobre la vista.

MVC: interacción de los componentes

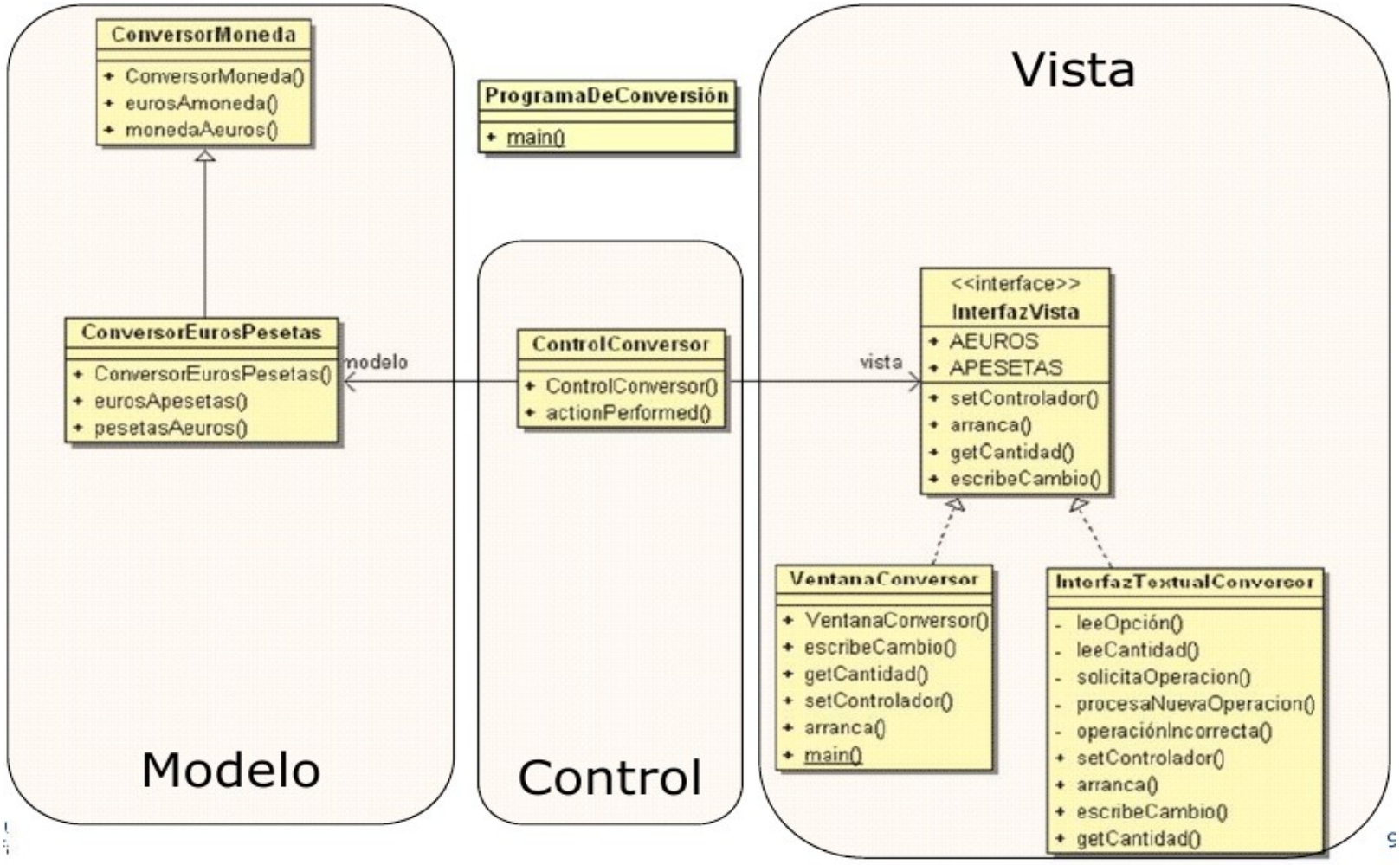


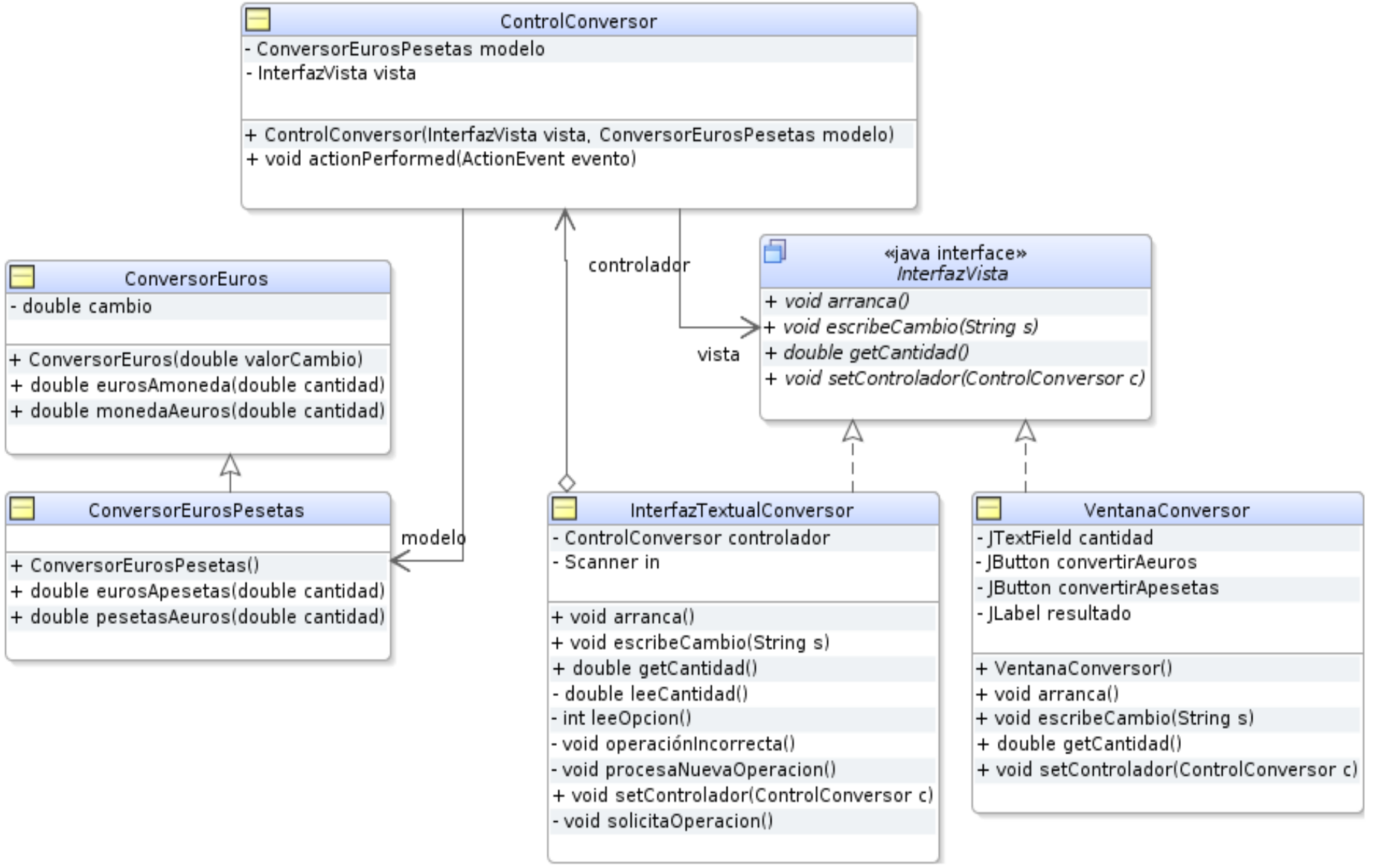
Sin embargo, se podría utilizar el *patrón Observador* para proveer cierta indirección entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aun así el modelo en sí mismo sigue sin saber nada de la vista.

Este uso del *patrón Observador* no es posible en las aplicaciones Web puesto que las clases de la vista están desconectadas del modelo y del controlador. En general 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. Por ejemplo en el MVC usado por Apple en su framework Cocoa.
Suele citarse como Modelo-Interface-Control, una variación del MVC más puro.
La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente....

EJEMPLO MVC: Conversor Euros a Pesetas




ControlConversor
- ConversorEurosPesetas modelo
- InterfazVista vista
+ ControlConversor(InterfazVista vista, ConversorEurosPesetas modelo)
+ void actionPerformed(ActionEvent evento)
controlador
vista
«java interface»
InterfazVista
+ void arranca()
+ void escribeCambio(String s)
+ double getCantidad()
+ void setControlador(ControlConversor c)
ConversorEuros
- double cambio
+ ConversorEuros(double valorCambio)
+ double eurosAmoneda(double cantidad)
+ double monedaAeuros(double cantidad)
ConversorEurosPesetas
+ ConversorEurosPesetas()
+ double eurosApesetas(double cantidad)
+ double pesetasAeuros(double cantidad)
modelo
InterfazTextualConversor
- ControlConversor controlador
- Scanner in
+ void arranca()
+ void escribeCambio(String s)
+ double getCantidad()
- double leeCantidad()
- int leeOpcion()
- void operaciónIncorrecta()
- void procesaNuevaOperacion()
+ void setControlador(ControlConversor c)
- void solicitaOperacion()
VentanaConversor
- JTextField cantidad
- JButton convertirAeuros
- JButton convertirApesetas
- JLabel resultado
+ VentanaConversor()
+ void arranca()
+ void escribeCambio(String s)
+ double getCantidad()
+ void setControlador(ControlConversor c)


```java
public interface InterfazVista
{
    public void setControlador(ControlConversor c);
    public void arranca();
    // comienza la visualización
    public double getCantidad();
    // cantidad a convertir
    public void escribeCambio(String s); //texto con la conversión
    // Constantes que definen las posibles operaciones:
    static final String AEUROS = "Pesetas a Euros";
    static final String APESETAS = "Euros a Pesetas";
}
```

Vista

```java
public class ConversorEuros
{
    private double cambio;

    public ConversorEuros(double valorCambio)
    {
        // valor en la moneda de 1 euro
        cambio = valorCambio;
    }

    public double eurosAmoneda(double cantidad)
    {
        return cantidad * cambio;
    }

    public double monedaAeuros(double cantidad)
    {
        return (cantidad / cambio);
    }
}
```

Modelo

```java
public class ConversorEurosPesetas extends ConversorEuros
{
    // Adaptador de clase
    public ConversorEurosPesetas()
    {
        super(166.386D);
    }

    public double eurosApesetas(double cantidad)
    {
        return eurosAmoneda(cantidad);
    }

    public double pesetasAeuros(double cantidad)
    {
        return monedaAeuros(cantidad);
    }
}
```

Modelo


```
public class ControlConversor implements ActionListener
{
    private InterfazVista vista;
    private ConversorEurosPesetas modelo;

    public ControlConversor(InterfazVista vista,
                            ConversorEurosPesetas modelo)
    {
        this.vista = vista;
        this.modelo = modelo;
    }
```

Controlador

```java
@Override
public void actionPerformed(ActionEvent evento)
{
    double cantidad = vista.getCantidad();

    if (evento.getActionCommand().equals(InterfazVista.AEUROS))
    {
        vista.escribeCambio(cantidad + " pesetas son: " +
                            modelo.pesetasAeuros(cantidad) + " euros");
    }
    else if (evento.getActionCommand().equals(InterfazVista.APESETAS))
    {
        vista.escribeCambio(cantidad + " euros son: " +
                            modelo.eurosApesetas(cantidad) + " pesetas");
    }
    else
        vista.escribeCambio("ERROR");
}
```

Controlador


```
public class VentanaConversor extends JFrame implements InterfazVista
{
    private JButton convertirApesetas;
    private JButton convertirAeuros;
    private JTextField cantidad;
    private JLabel resultado;

    public VentanaConversor()
    {
        super("Conversor de Euros y Pesetas");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel panelPrincipal = new JPanel();
        panelPrincipal.setLayout(new BorderLayout(10, 10));
        cantidad = new JTextField(8);
        JPanel panelaux = new JPanel();
        panelaux.add(cantidad);
        panelPrincipal.add(panelaux, BorderLayout.NORTH);
```

Vista

```
public VentanaConversor()
{
    super("Conversor de Euros y Pesetas");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JPanel panelPrincipal = new JPanel();
    panelPrincipal.setLayout(new BorderLayout(10, 10));
    cantidad = new JTextField(8);
    JPanel panelaux = new JPanel();
    panelaux.add(cantidad);
    panelPrincipal.add(panelaux, BorderLayout.NORTH);
    resultado = new JLabel("Indique una cantidad y pulse uno de los botones");
    JPanel panelaux2 = new JPanel();
    panelaux2.add(resultado);
    panelPrincipal.add(panelaux2, BorderLayout.CENTER);
    convertirApesetas = new JButton("A pesetas");
    convertirApesetas.setActionCommand(APESETAS);
    convertirAeuros = new JButton("A euros");
    convertirAeuros.setActionCommand(AEUROS);
    JPanel botonera = new JPanel();
    botonera.add(convertirApesetas);
    botonera.add(convertirAeuros);
    panelPrincipal.add(botonera, BorderLayout.SOUTH);
    getContentPane().add(panelPrincipal);
}
```

Vista

```java
@Override
public void setControlador(ControlConversor c)
{
    convertirApesetas.addActionListener(c);
    convertirAeuros.addActionListener(c);
}
```

```java
@Override
public void arranca()
{
    pack(); // coloca los componentes
    setLocationRelativeTo(null); // centra la ventana en la pantalla
    setVisible(true); // visualiza la ventana
}
```

Vista

```java
@Override
public double getCantidad()
{
    try
    {
        return Double.parseDouble(cantidad.getText());
    }
    catch (NumberFormatException e)
    {
        return 0.0D;
    }
}
```


```java
@Override
public void escribeCambio(String s)
{
    resultado.setText(s);
}
```

Vista


```java
public class InterfazTextualConversor implements InterfazVista
{
    /**
     * @aggregation shared
     */
    private ControlConversor controlador;
    // Gestión de la entrada por teclado
    private Scanner in = new Scanner(System.in);

    private int leeOpcion()
    {
        String s = null;
        try
        {
```

Vista

```java
public class VentanaConversor extends JFrame implements InterfazVista
{
    private JButton convertirApesetas;
    private JButton convertirAeuros;
    private JTextField cantidad;
    private JLabel resultado;
```

```java
public class InterfazTextualConversor implements InterfazVista
{
    /**
     * @aggregation shared
     */
    private ControlConversor controlador;
    // Gestión de la entrada por teclado
    private Scanner in = new Scanner(System.in);
```

Vista

```java
public class ProgramaDeConversion
{
    public static void main(String[] args)
    {
        // el modelo:
        ConversorEurosPesetas modelo = new ConversorEurosPesetas();
        // la vista:
        InterfazVista vista = new VentanaConversor();
        // y el control:
        ControlConversor control = new ControlConversor(vista, modelo);
        // configura la vista
        vista.setControlador(control);
        // y arranca la interfaz (vista):
        vista.arranca();
    }
}
```

Vista