Aplicaciones RIA con ZK

- Sesión 7: El patrón MVC

Índice

- Descripción del patrón MVC
- Ejecutando código Java
- Ejemplo de controlador sencillo
- Controlador completo
- ToDo: un gestor de tareas con data binding y MVC

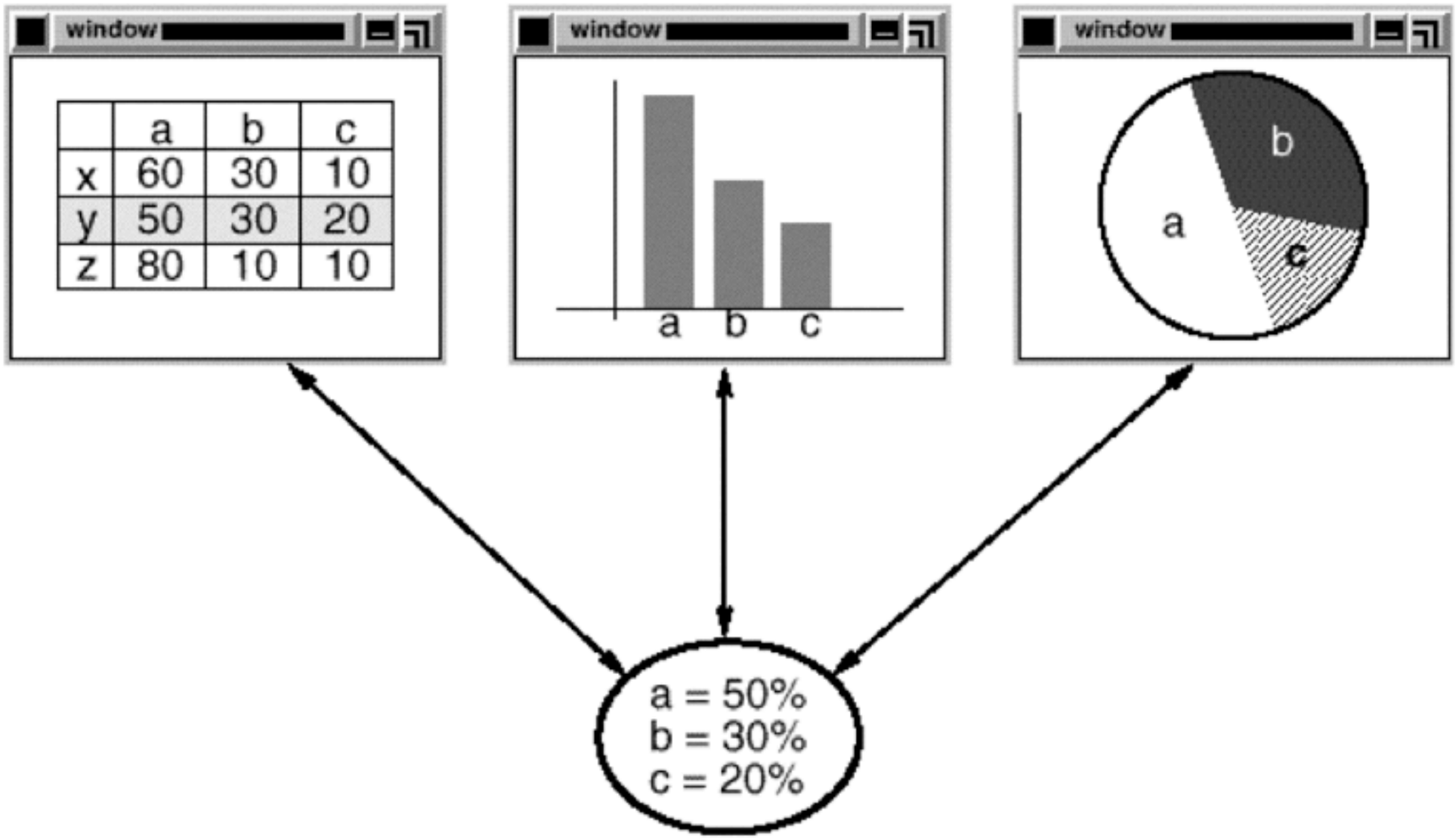
Patrón MVC

- Patrón de arquitectura de software para separar responsabilidades y mejorar el desarrollo, prueba y mantenimiento de la aplicación
- **Modelo** (M) = los datos de la aplicación y la lógica de negocio
- **Vista** (V) = el aspecto visual de la aplicación; su interfaz de usuario
- **Controlador** (C) = código que gestiona los eventos de la IU, comunica los datos al modelo y modifica la vista

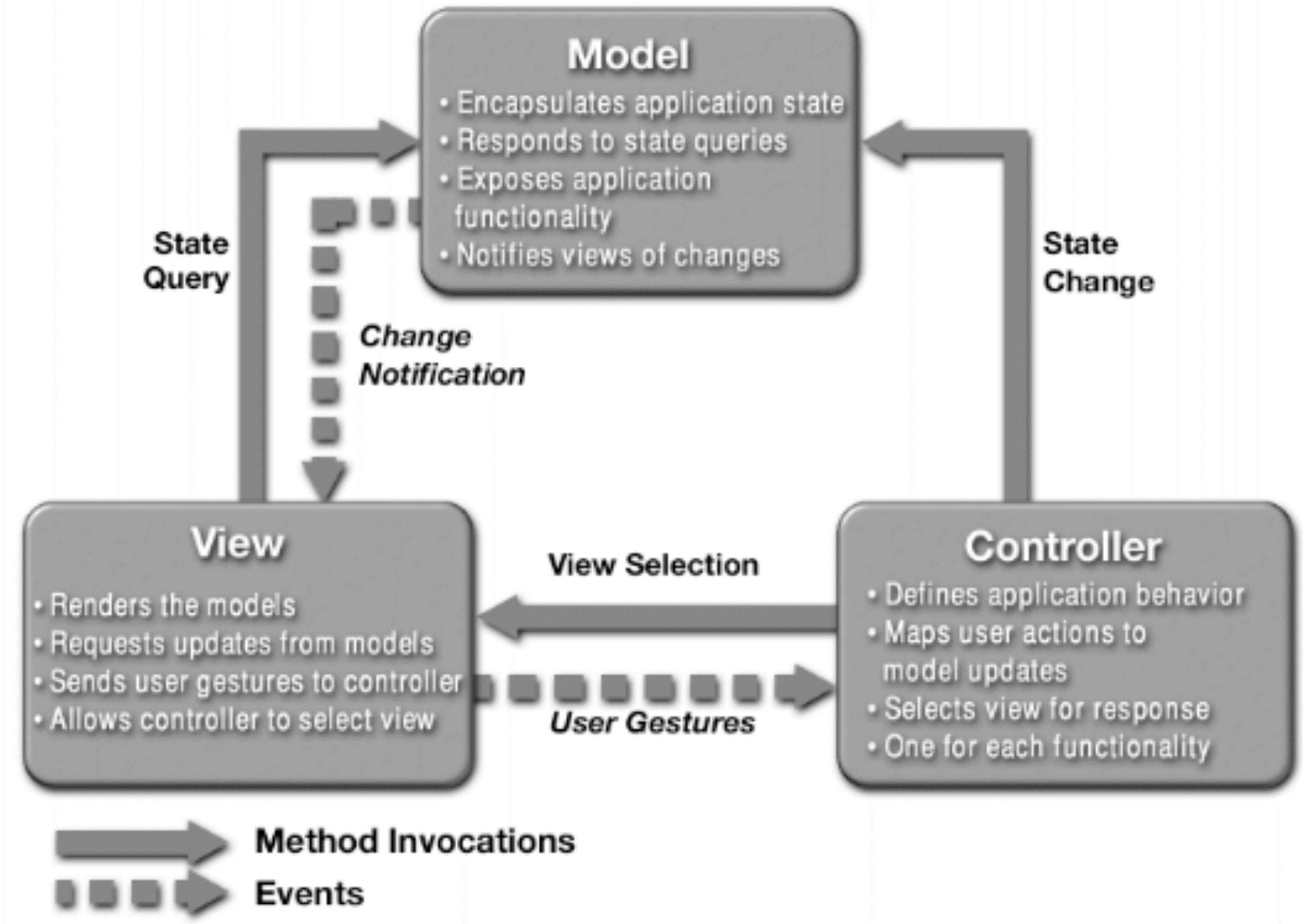
Ejemplo

- Aplicación de gestión de clientes
- **Modelo**: clases Cliente, Dirección, ... y métodos de negocio: añade cliente, consulta clientes que no han hecho compras, ...
- **Vista**: listas, formularios, gráficas y pantallas que presentan la información recogida del modelo
- **Controlador**: código que envía las peticiones al modelo y que cambia las pantallas

Separación entre el modelo y las vistas



Comunicación entre las capas



Ventajas de la separación MVC

- Independencia del modelo y el resto de la aplicación: es posible desarrollar y probar el modelo y la lógica de negocio sin depender demasiado de la IU
- Multiplataforma: el mismo modelo y la lógica de negocio sirve para distintas versiones de la aplicación
- Aislamos los cambios: cambios en la presentación no afectan al modelo
- Reutilización: podemos generalizar componentes reusando el código del controlador

MVC en ZK

- **Modelo**: Clases Java con los objetos que se muestran en los componentes ZK
- **Vista**: Ficheros .zul
- **Controlador**: Clases Java que gestionan los eventos, invocan al modelo y modifican la vista
- **Data binding**: mecanismo por el que se conecta el modelo y la vista y los mantiene sincronizados

Cómo ejecutar código Java desde ZK

- En ZK hay diversas formas de invocar a las funciones del modelo
- Desde zscript invocado en los eventos: bueno para prototipado
- Utilizando los atributos **use** y **apply** que asocian una clase controlador a un componente

Atributo use

- Permite definir una versión especializada de un componente
- La clase Java debe extender el componente y podemos definir métodos adicionales
- Los métodos se pueden invocar desde los eventos

Atributo use

```
<window id="win_1" use="MyWindow" title="list">
   <listbox id="lb_1">
       <listitem label="${each}" foreach="${win_1.contacts}"
   </listbox>
   <button label="Hello" onClick="win_1.onTest()"/>
</window>
```

```
public class MyWindow extends Window {
   String[] contacts = new String[] { "Monday", "Tuesday", "Wednesday" };

   public String[] getContacts(){
      return contacts;
   }

   public void onTest() {
      Listbox lb = (Listbox) getFellow("lb_1");
      for (int i = 0; i < contacts.length; i++) {
         Listitem li = new Listitem();
         li.setLabel(contacts[i]);
         lb.appendChild(li);
      }
   }
}
```

Atributo apply

- Para definir controladores aplicables a distintos componentes
- La clase Java debe implementar la interfaz Composer, con el método doAfterCompose para configurar el componente

Ejemplo básico

```
<window apply="MyController">
   <button id="btn_1" label="test" forward="onClick=onTest" />
</window>
```

```
public class MyController implements Composer {
   private Button btn;

   public void doAfterCompose(Component win) throws Exception {
      btn = (Button) win.getFellow("btn_1");
      // define and register event listeners
      win.addEventListener("onTest", new EventListener() {
         public void onEvent(Event event) throws Exception {
         btn.setLabel("event handled");
         }
      });
   }
}
```

Versión automática

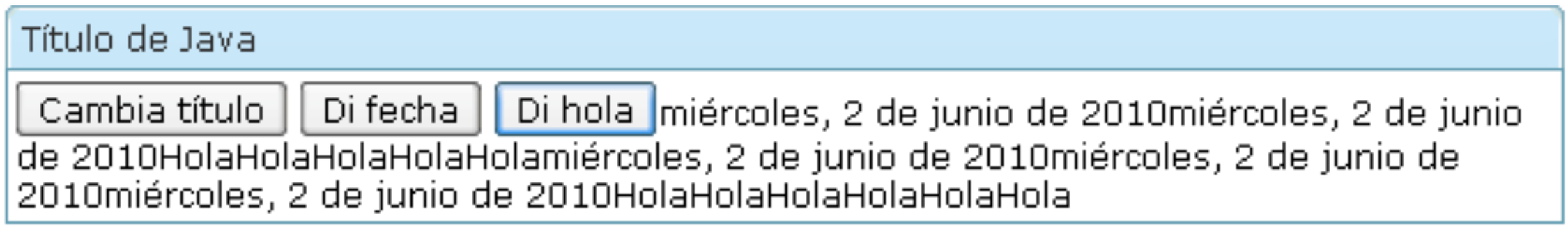
- Se simplifica mucho la declaración de eventos y la obtención de componentes utilizando el GenericForwardComposer

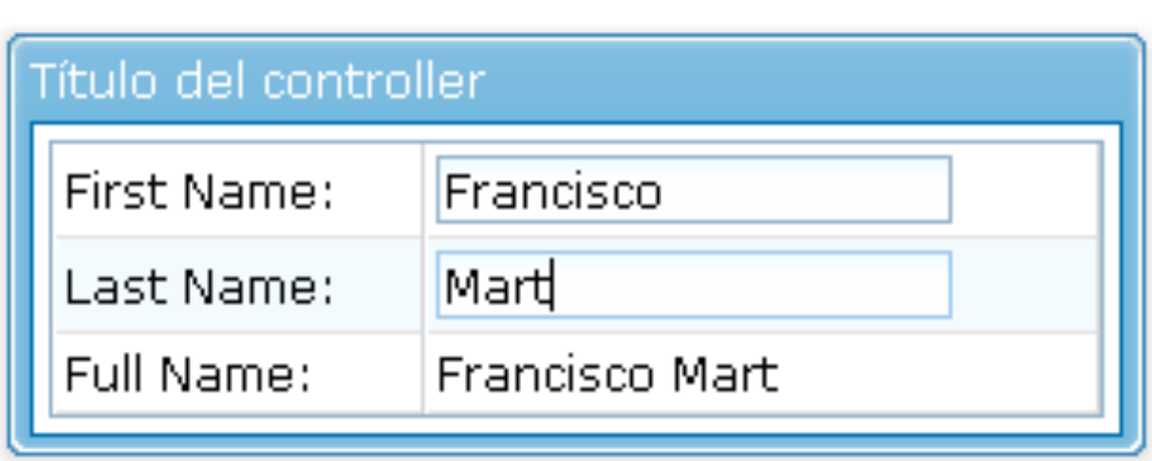
```
<window apply="MyController">
   <button id="btn_1" label="test"/>
</window>
```

```
public class MyController extends
GenericForwardComposer {
   private Button btn_1;

   //onClick event from btn_1 component
   public void onClick$btn_1(Event event) {
      btn_1.setLabel("event handled");
   }
}
```

Probamos varios ejemplos





Una aplicación completa

- Un ejemplo de aplicación completa MVC que combina clases Java como modelo, un controlador y data binding