Tema: IV.
Interfaces gráficos de usuario

Herramientas Avanzadas para el Desarrollo de Aplicaciones

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante

lsi Department of Software and Computing Systems

Contenido

Preliminares (I)

- En este tema vamos a ver cómo dotar a nuestras aplicaciones de un *Interfaz Gráfico de Usuario* .
- Veremos cómo se apoya en los conceptos de *programación dirigida por eventos* y *ejecución diferida de código* vista en los temas previos.
- Daremos una breve introducción a la arquitectura MVC ya que para escribir el código de nuestras aplicaciones haremos uso de la misma.
- El interfaz gráfico de nuestras aplicaciones de escritorio empleará la biblioteca Gtk+ .

Preliminares (II)

- Veremos como crear gráficamente el interfaz de la aplicación con un programa dedicado al diseño de interfaces gráficos de usuario. Este se llama Glade .
- Los interfaces gráficos creados con `glade` serán *cargados* dinámicamente en tiempo de ejecución y mostrados al usuario para que pueda interactuar con los mismos.
- Glade genera archivos XML que contienen la descripción del interfaz de usuario diseñado. Estos archivos se pueden leer/cargar desde diversos lenguajes de programación: `C`, `C++`, `C#`, `Vala`, `Java`, etc. . .
- Todo esto lo haremos con el lenguaje de programación Vala empleado en las prácticas de la asignatura para la parte de 'escritorio'.

Preliminares (III)

- Vala emplea algunas tecnologías que componen los fundamentos de `Gtk+`.
- Estas son GLib y Gobject .
- Esto influye para que con Vala sea muy sencillo construir aplicaciones que tengan un interfaz gráfico de usuario basado en `Gtk+`.
- Vamos a ver un vídeo donde en menos de cinco minutos se nos muestra cómo crear una aplicación sencilla dotada de interfaz gráfico de usuario[1].

[1] En este caso sin hacer uso de Glade.

MVC (I)

- MVC surge junto con Smalltalk durante los años 70.
- Es aplicable al desarrollo de cualquier aplicación independientemente del lenguaje de programación elegido.
- No es necesario el uso de un lenguaje orientado a objetos para emplearlo, aunque esta metodología lo haga más sencillo.

MVC (II)

- La idea clave de MVC consiste en dividir el código de una aplicación en capas, concretamente 3:
  1. **Modelo**
  2. **Vista**
  3. **Controlador**
- Cada una de estas capas puede ser sustituida en cualquier momento sin afectar a las otras, p.e., tener diferentes vistas para un mismo modelo.
- Esta división del código garantiza mayor facilidad de portabilidad y de adaptación a los requerimientos del usuario.

MVC: Modelo. –Capa de la Aplicación–

- Es la representación 'software' del problema a resolver, sus datos, funciones, etc. . . – personas, coches, asientos contables. . . –
- Proporciona los métodos necesarios para que:
  - Se puedan consultar los datos del modelo ($\approx$ *getters*).
  - Se puedan modificar los datos del modelo ($\approx$ *setters*).
- Los modelos no se comunican con las vistas[2], de este modo conseguimos una mayor independencia entre el código que constituye cada capa.
- Un modelo puede tener asociadas varias vistas.

[2] Aunque en ocasiones determinadas puede resultar interesante.

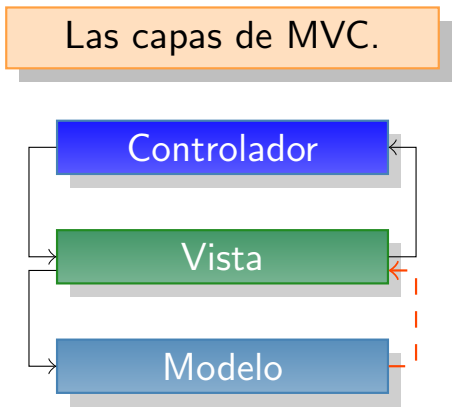
MVC: Vista. –Capa de Presentación–

- Sirve para mostrar al usuario los 'datos' del modelo que le interesan en cada caso –el nombre de una persona, la velocidad de un coche, el importe de un asiento...–
- Una vista no tiene porqué ser solamente en modo gráfico, puede ser en modo texto...
- Las vistas se comunican con los modelos de forma bi-direccional –solicitan información, pueden modificar información–
- En la arquitectura MVC original las vistas se pueden 'anidar' dando lugar a lo que se llama una vista principal –*top-view*– compuesta de subvistas. A efectos de la asignatura sólo usaremos vistas simples, no compuestas por otras.

MVC: Controlador. –Capa de Interacción–

- Contiene el código que hace de interfaz entre los dispositivos de entrada –teclado, ratón, etc. . .– y las capas de Vistas y Modelos.
- Este es el código que permite al usuario interactuar con las Vistas.
- Normalmente no tendremos que escribir código relacionado con esta capa ya que el código que iría aquí es el que proporciona la biblioteca gráfica utilizada (Gtk+ en nuestro caso).

MVC: Diagrama de interacción entre capas



La conexión entre el modelo y las vistas asociadas sólo tiene sentido si los datos del modelo se van a modificar internamente por algún cálculo y queremos que se actualicen las vistas asociadas a ese modelo.

Gtk+ (I)

- Gtk+ se desarrolla como toolkit libre para la aplicación gimp de tratamiento de imágenes. Hoy en día es una de las bases del escritorio gnome .
- Se distribuye con licencia LGPL.
- Disponemos de una extensa documentación para Gtk+ y para usarlo desde Vala en formato electrónico que se puede consultar en línea.
- También consta de un constructor gráfico del interfaz de la aplicación: glade .

Gtk+ (II)

- Gtk+ es actualizado sistemáticamente un par de veces al año, hoy día podemos encontrarnos con las versiones 2.x.y (en modo mantenimiento) y, la actualmente activa, 3.x.y. En el laboratorio de prácticas de la EPS tenemos instalado Gtk+ 3.x.y.
- Como veremos más adelante, esto influye a la hora de las opciones que hemos de dar al compilador de Vala: `--pkg gtk+-2.0` o `--pkg gtk+-3.0`.
- Lo que denominamos de forma general Gtk+ es un compendio de una serie de bibliotecas: Glib, GdkPixbuf, Gdk, Gtk, Atk y Pango.
- La estructura interna de Gtk+ es la de una jerarquía de clases formada por varios árboles (distintas raíces) con herencia simple.
- Estos árboles representan a cada una de las bibliotecas que hemos comentado antes (glib, gdk, gtk, etc...).

lsi Department of Software and Computing Systems

Gtk+ + Vala + signal/handler (I)

- El uso de Gtk+ desde Vala se basa en lo que hemos visto en los temas anteriores: eventos/señales y manejadores/callbacks.
- Los elementos de interfaz de usuario (widgets, controles) que proporciona Gtk+ definen una serie de señales que pueden emitir.
- Nosotros nos dedicamos a conectarles los métodos o funciones de nuestro código que hacen de manejador o callback, p.e. consultando la documentación de la clase Button encontramos un apartado dedicado a señales:

```
1     public virtual signal void activate ()

3     The activate signal on GtkButton is an action signal and
      emitting it causes the button to animate press then
5     release....

7     public virtual signal void clicked ()

9     Emitted when the button has been activated (pressed and
      released).
11
      ...
```

Gtk+ + Vala + signal/handler (II)

Veamos un ejemplo completo:

```
    // File: gtk-hello.vala
 2  using Gtk;

 4  int main (string[] args) {
      Gtk.init (ref args);
 6
      var window = new Window (); //Gtk.Window (using Gtk)
 8    window.title = "First GTK+ Program";
      window.border_width = 10;
10    window.window_position = WindowPosition.CENTER;
      window.set_default_size (350, 70);
12    window.destroy.connect (Gtk.main_quit);

14    var button = new Button.with_label ("Click me!");
      button.clicked.connect (() => {button.label = "Thank you";});
16
      window.add (button);
18    window.show_all ();

20    Gtk.main ();
      return 0;
22  }
```

Se compila así: `valac --pkg gtk+-3.0 gtk-hello.vala`.

Gtk+ + Vala + signal/handler (III)

Destacar del código anterior:

- using Gtk: equivalente a un `import` de Java o `#include` de C o C++ para tener acceso a declaraciones/definiciones de Gtk+.
- Gtk.init (ref args): Inicia la biblioteca Gtk+. Es indispensable hacerlo siempre al principio del programa principal.
- var window = new Window (): Crea un objeto de clase `Gtk.Window`, es decir, una ventana sobre la que poder añadir otros elementos de interfaz de usuario.
- var button = new Button.with_label("Click me!"): Crea un objeto de clase `Gtk.Button`.

Gtk+ + Vala + signal/handler (IV)

- button.clicked.connect(() => {button.label = "Thank you";}): La clase `Button` dispone de la señal `clicked`, aquí le conectamos un manejador. En este caso es una función-λ.
- window.add (button): Añadimos el botón creado a la ventana.
- window.show_all(): La ventana hace visibles todos los widgets que contenga.
- Gtk.main(): Es el bucle de espera de eventos, de él solo salimos para finalizar la aplicación.

Gtk+ Widgets (I)

- Gtk+ nos ofrece una colección importante de widgets predefinidos organizados como una jerarquía de clases con herencia simple. Podemos verla aquí
- La clase base de cualquier *elemento de interfaz de usuario* es la clase GtkWidget .
- En Vala el prefijo `Gtk` de cualquier identificador, p.e. 'GtkWidget' se interpreta como un *espacio de nombres*, por lo que el identificador en Vala sería: 'Gtk.Widget'.
- Disponemos de la documentación equivalente para la adaptación de Gtk+ a Vala aquí .

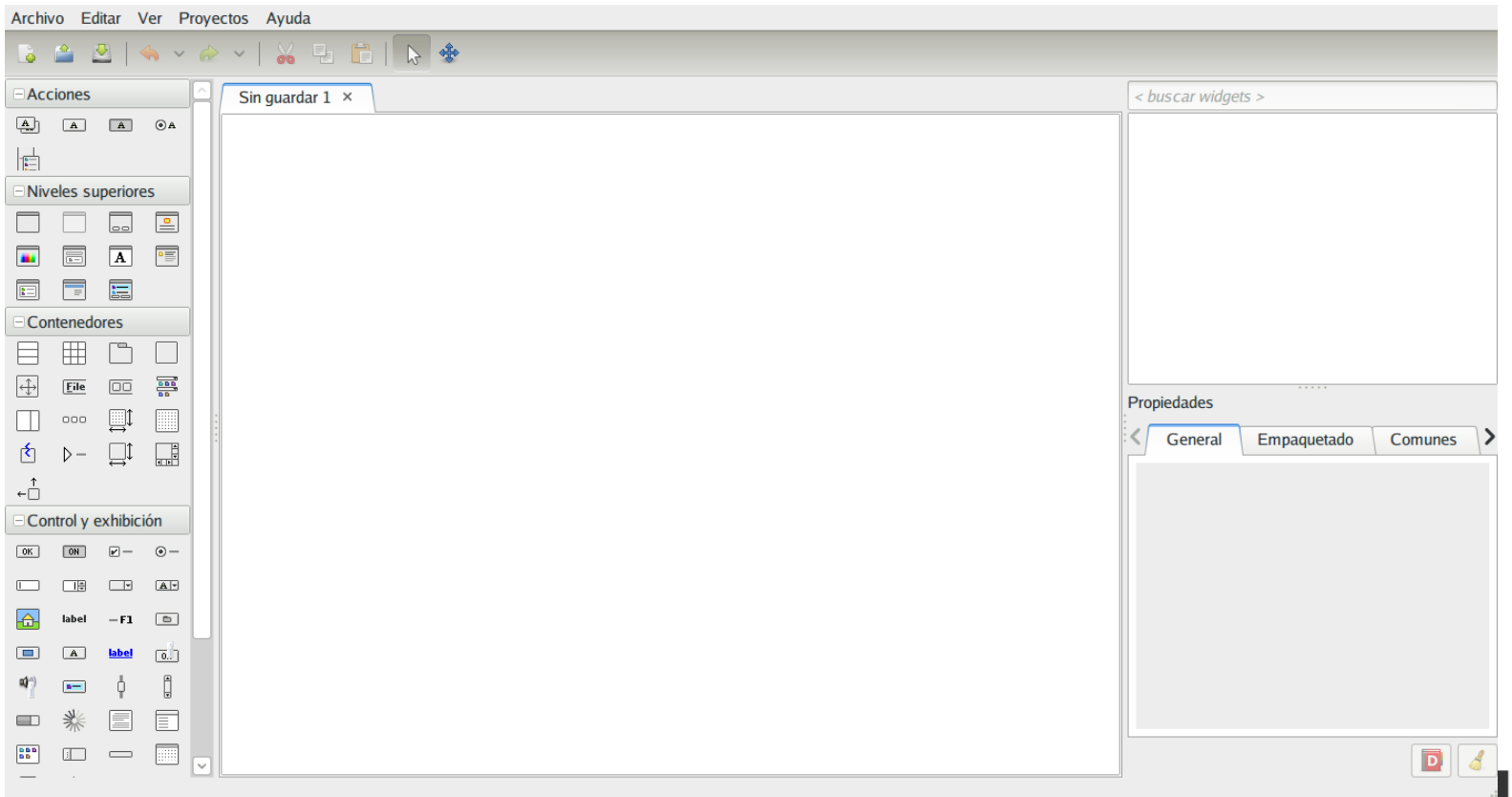
Gtk+ Widgets (II)

- En Gtk+ un widget normalmente solo puede contener a otro widget.
- Para crear interfaces de usuario funcionales necesitamos solventar esta limitación.
- Existe un tipo especial de widgets que son los *contenedores*, concretamente las clases derivadas de Gtk.Box . Puedes ver más información sobre sus clases derivadas aquí y en Gtk.HBox y Gtk.VBox .
- También es aconsejable que conozcas el *contenedor en forma de tabla*: Gtk.Table . Tienes más información sobre él en la documentación de Gtk.
- Visualmente no tienen ninguna apariencia pero tienen como característica que pueden contener más de un widget, así como gestionar el espacio que ocupan y qué ocurre cuando cambia el tamaño de este espacio.

lsi Department of Software and Computing Systems

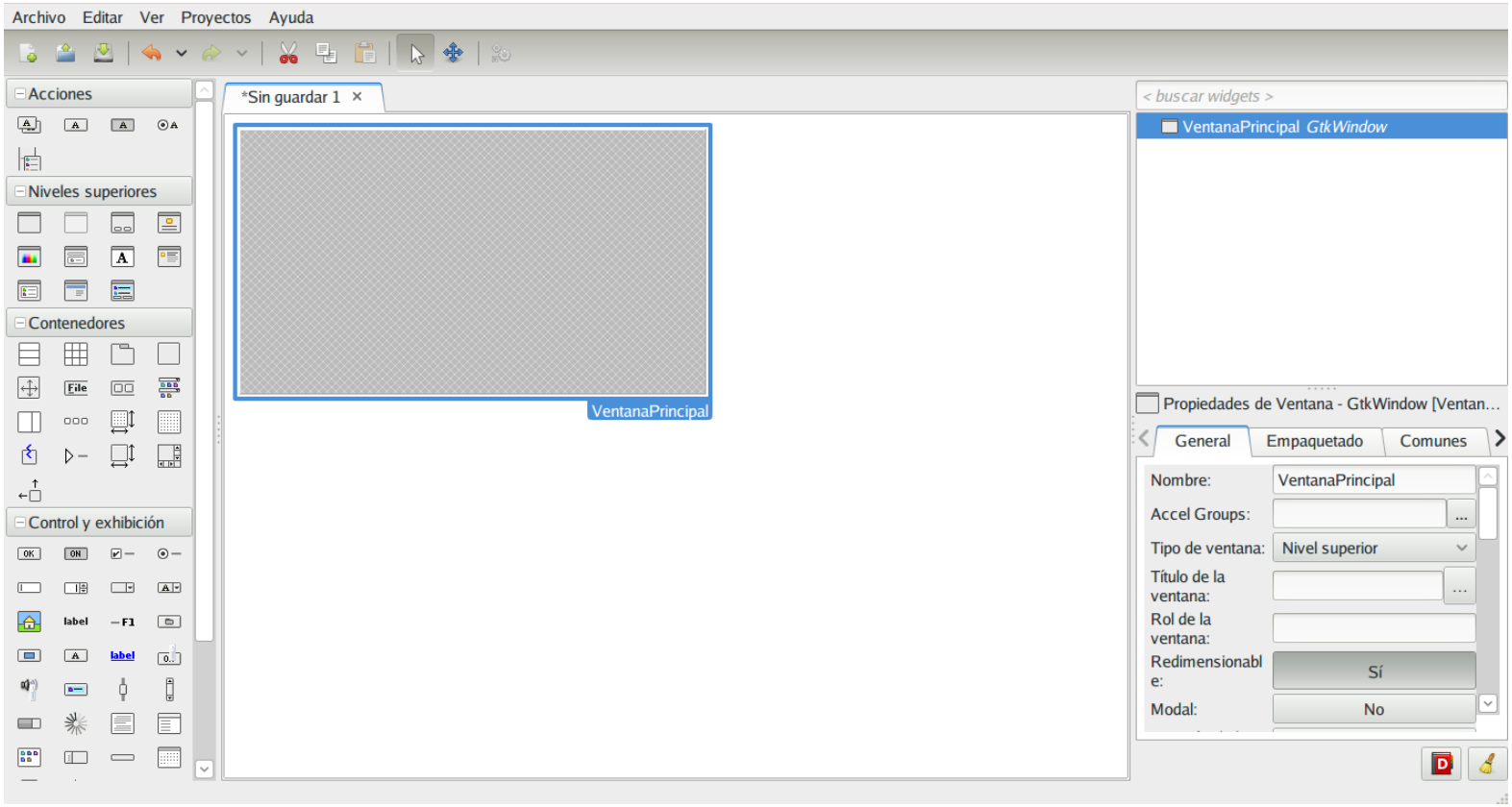
Glade (I)

- Glade es el constructor gráfico oficial de interfaces de usuario para Gtk+.
- El aspecto que presenta es así:



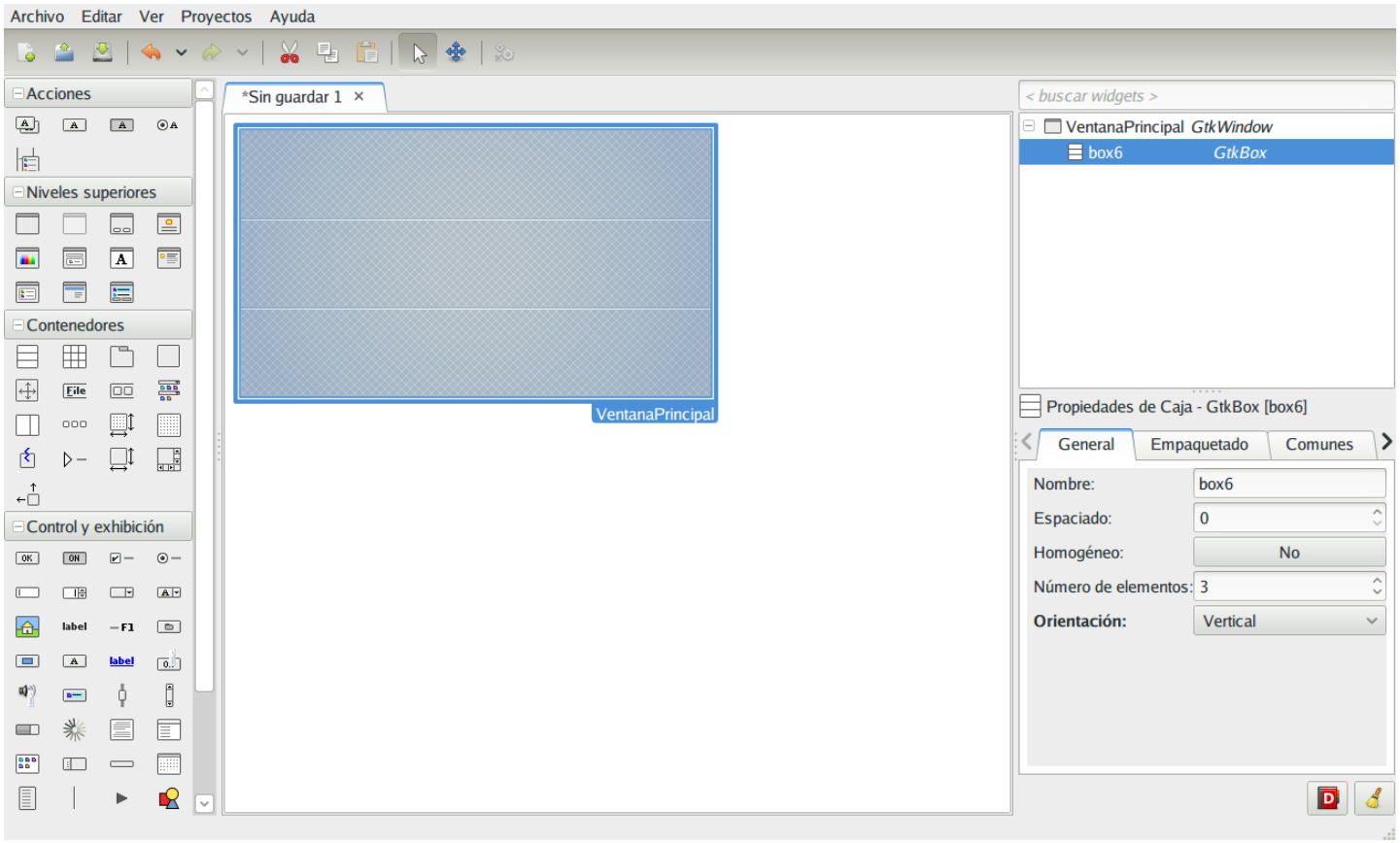
Glade (II)

Creamos las ventanas, diálogos, etc... iniciales eligiéndolos de entre la lista de 'Niveles superiores':



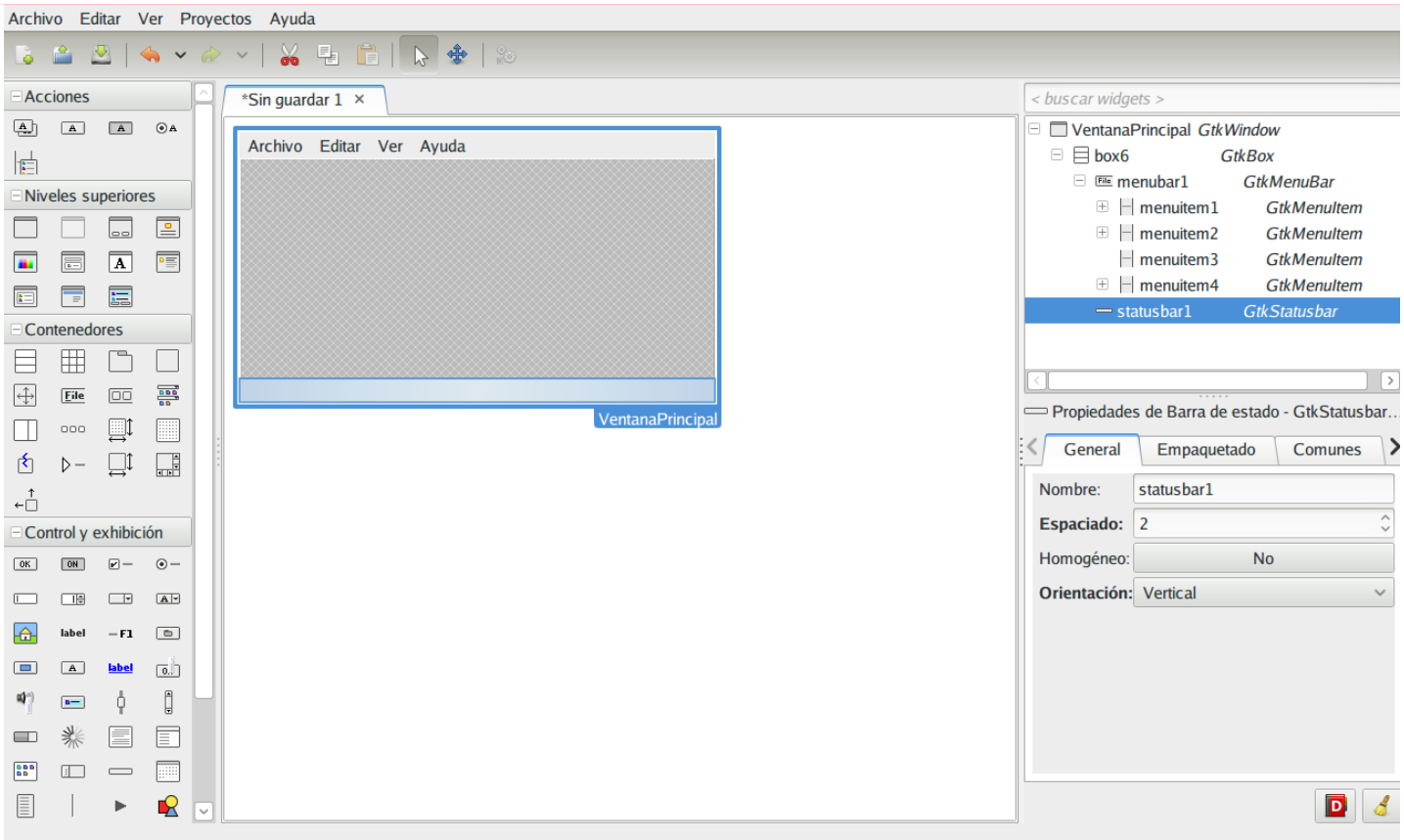
Glade (III)

Le añadimos los ‘contenedores’ necesarios para construir nuestro interfaz:



Glade (IV)

Insertamos en los 'huecos' de los 'contenedores' los widgets que necesitemos, p.e. botones, etiquetas de texto, campos de texto editable, etc. . . :



Glade (V)

- Una vez tenemos creado el interfaz lo guardamos desde el menú 'Archivo'.
- Son archivos de texto en formato `xml`.
- Suelen llevar la extensión '`.ui`'.
- Desde nuestra aplicación los cargamos dinámicamente con un objeto de la clase '`Gtk.Builder`'.
- Con el método '`add_from_file`' leemos el archivo '`.ui`'.
- Con el método '`get_object`' cargamos uno a uno por su nombre los widgets que nos interesan del archivo '`.ui`'

Glade + Gtk+ (I)

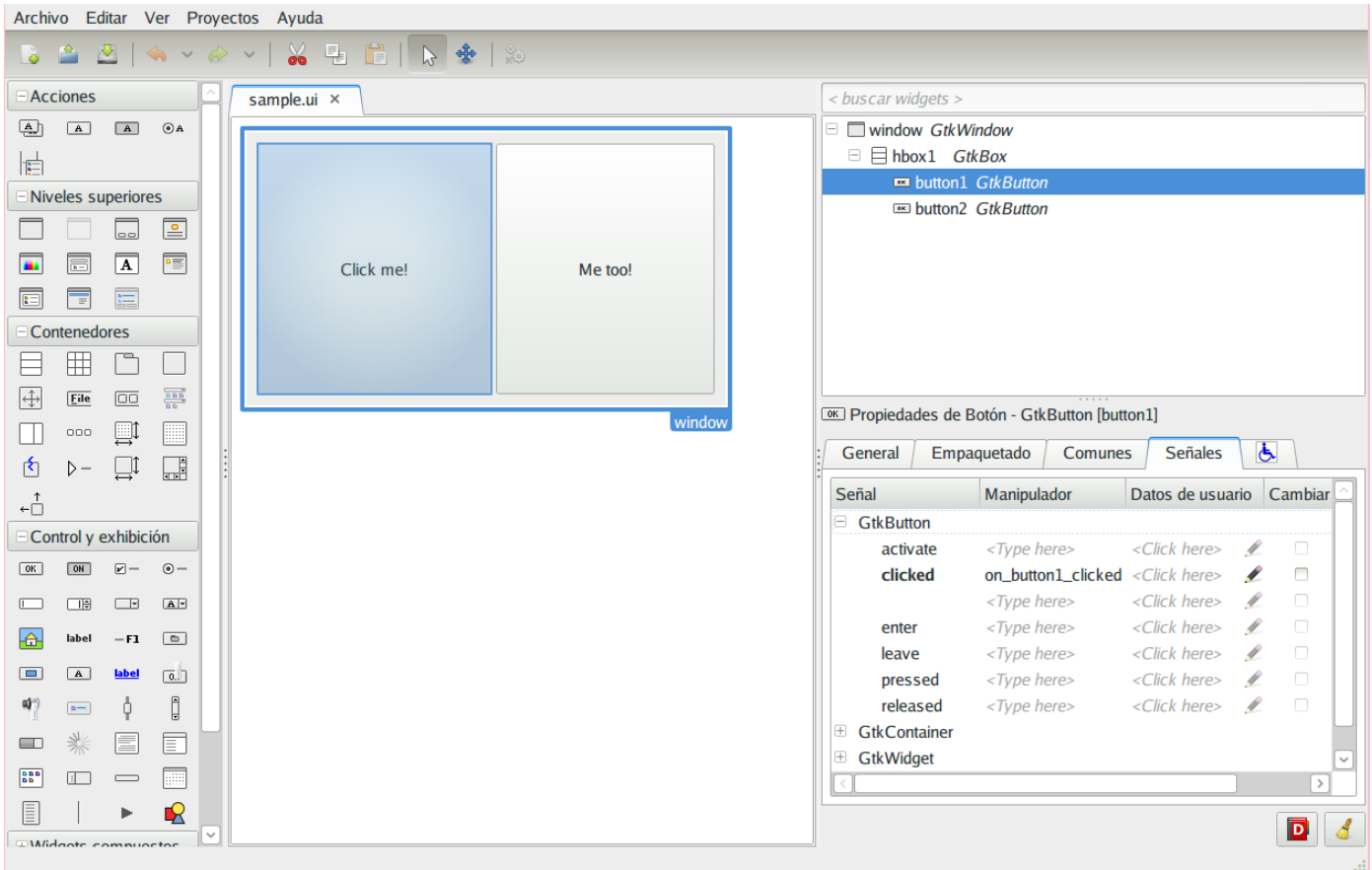
- Veamos cómo podría quedar un ejemplo de código similar al visto en la transparencia 15.

```
// valac --pkg gtk+-3.0 --pkg gmodule-2.0 gtk-builder-sample.vala
using Gtk;
public void on_button1_clicked (Button source) {
  source.label = "Thank you!";
}
public void on_button2_clicked (Button source) {
  source.label = "Thanks!";
}
int main (string[] args) {
  Gtk.init (ref args);
  try {
    var builder = new Builder ();   // cargador de archivos de Glade
    builder.add_from_file ("sample.ui");  // carga del interfaz
    builder.connect_signals (null);       // auto conexion de senyales
    var window = builder.get_object ("window") as Window;
    window.show_all ();
    Gtk.main ();
  } catch (Error e) {
    stderr.printf ("Could not load UI: %s\n", e.message);
    return 1;
  }
  return 0;
}
```

El interfaz de usuario lo podemos descargar de sample.ui

Glade + Gtk+ (II)

Visto `sample.ui` desde Glade tiene este aspecto:



Glade + Gtk+ (III)

- Podemos conectar métodos como manejadores de señales.
- En este caso hay que seguir unas normas para dar nombres a las señales en Glade.
- Veamoslo con un ejemplo:

```
1   using Gtk;

3   namespace Foo {
      public class MyBar {
5
        [CCode (instance_pos = -1)]
7       public void on_button1_clicked (Button source) {
          source.label = "Thank you!";
9       }

11      [CCode (instance_pos = -1)]
        public void on_button2_clicked (Button source) {
13        source.label = "Thanks!";
        }
15    }
    }
17
    // ...
19  var object = new Foo.MyBar ();
    builder.connect_signals (object);
21  // ...
```

Glade + Gtk+ (IV)

- Si declaramos los métodos que harán de callbacks dentro de una clase y/o dentro de un espacio de nombres...
- ... en Glade deberemos preceder el nombre del método que hará de callback con el nombre del espacio de nombres y/o la clase a la que pertenece, en minúsculas y separados por símbolos de subrayado.
- Por ejemplo: '`Foo.MyBar.on_button1_clicked`' en Glade sería: '`foo_my_bar_on_button1_clicked`', como podemos ver en esta imagen: