Widgets, Contenedores y Señales: Creando Aplicaciones con GTK+

Claudio Saavedra

csaavedra@alumnos.utalca.cl,
http://www.gnome.org/~csaavedr/

Resumen Se presentan los elementos básicos necesarios para el desarrollo de interfaces de usuario en el lenguaje C mediante el uso de la biblioteca GTK+. Se introducen los conceptos de Widgets, Contenedores, Señales, Callbacks.

1. Introducción

GNOME, como un entorno de escritorio moderno, basa la construcción de sus aplicaciones en una completa y sofisticada biblioteca, GTK+, que permite el fácil desarrollo de aplicaciones de usuario completas y profesionales.

En este artículo, introduciremos los elementos básicos para el desarrollo de aplicaciones GNOME mediante GTK+, utilizando el lenguaje C.

No se pretende explicar como GTK+ está fundamentado ni cual es su arquitectura y metodología de funcionamiento. Simplemente, mostraremos los componentes más comunes para el desarrollo de aplicaciones (ventanas, botones, entradas de texto, etc.), cómo agruparlas dentro de una aplicación (mediante el uso de distintos contenedores) y cómo darles vida (mediante el uso de sus señales). Mayores detalles sobre el funcionamiento interno de GTK+, además de su relación con otras bibliotecas dentro de GNOME, y el cómo GTK+ permite la extensión de sus componentes mediante un sistema de objetos, son temas que no serán discutidos, pero si brindaremos referencias para quienes estén interesados en profundizar en ellos.

2. Ejemplo introductorio

Para romper el hielo, presentaremos el siguiente ejemplo básico de un programa desarrollado en GTK+, que no es más que una ventana con un botón y una etiqueta de texto en su interior, como en la figura 1.

```
#include <gtk/gtk.h>

void
on_button_clicked (GtkButton *button,
                   gpointer data)
```

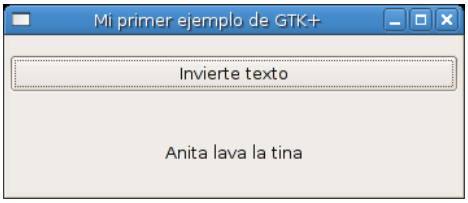

Figura 1. Simple ejemplo de una aplicación escrita con GTK+.

```
{
    GtkLabel *label = GTK_LABEL (data);
    const gchar* text = gtk_label_get_text (label);
    gint i;
    gint t_size = g_utf8_strlen (text, -1);

    gchar* inverted_text = g_new0 (gchar, t_size);

    for (i = 0; i < t_size; i++) {
        inverted_text[i] = text [t_size - i - 1];
    }

    gtk_label_set_text (label, inverted_text);

    g_free (inverted_text);
}

GtkWidget *
construct_window (void)
{
    GtkWidget *window;
    GtkWidget *vbox;
    GtkWidget *button;
    GtkWidget *label;
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window),
                          "Mi primer ejemplo de GTK+");

    vbox = gtk_vbox_new (TRUE, 0);

    button = gtk_button_new_with_label ("Invierte texto");

    label = gtk_label_new ("Anita lava la tina");

    gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);
    gtk_box_pack_start (GTK_BOX (vbox), label, FALSE, FALSE, 0);
```

```
    gtk_container_add (GTK_CONTAINER (window), vbox);

    g_signal_connect (G_OBJECT (button),
                      "clicked",
                      G_CALLBACK (on_button_clicked),
                      label);

    return window;
}

int
main (gint argc, gchar **argv)
{
    gtk_init (&argc, &argv);
    GtkWidget *window = construct_window ();

    gtk_window_set_default_size (GTK_WINDOW (window), 400, 300);
    gtk_widget_show_all (window);
    g_signal_connect (G_OBJECT (window), "delete-event",
                      gtk_main_quit, NULL);

    gtk_main ();

    return 0;
}
```

En el ejemplo vemos todos los elementos que se discutirán en el artículo.

Para poder compilar software desarrollado con GTK+, utilizaremos la herramienta `pkg-config`. Ésta nos permite simplificar los parámetros que debemos pasar a nuestro compilador al momento de compilar.

Sin `pkg-config`, nuestra linea de compilación sería aproximadamente así:

```
$ gcc -o ejemplo ejemplo.c -I/opt/gnome-2.16/include/gtk-2.0 \
-I/opt/gnome-2.16/lib/gtk-2.0/include -I/opt/gnome-2.16/include/atk-1.0 \
-I/opt/gnome-2.16/include/cairo -I/opt/gnome-2.16/include/pango-1.0 \
-I/opt/gnome-2.16/include/glib-2.0 -I/opt/gnome-2.16/lib/glib-2.0/include \
-L/opt/gnome-2.16/lib -lgtk-x11-2.0 -lgdk-x11-2.0 -latk-1.0 -lgdk_pixbuf-2.0 \
-lm -lpangocairo-1.0 -lpango-1.0 -lcairo -lgobject-2.0 -lgmodule-2.0 -ldl \
-lglib-2.0
```

Como podemos apreciar, es bastante engorroso. Más aún, la ubicación de las bibliotecas puede variar de sistema en sistema, lo cual complicaría aún más el proceso. Usando `pkg-config`, todo se reduce a:

```
$ gcc -o ejemplo ejemplo.c `pkg-config --cflags --libs gtk+-2.0`
```

Debe notarse que el carácter ‘ es un acento grave, como en la palabra *Viquipèdia*.

2.1. Inicializando GTK+: Función main ().

El código de la función `main ()` nos permite entre otras cosas, inicializar GTK+. Para esto es imprescindible llamar a la función `gtk_init ()` antes de llamar a cualquier otro método del toolkit. Esta función se encargará de inicializar todo lo que sea necesario para que GTK+ pueda operar.

Una vez que tenemos inicializado el toolkit, podemos crear nuestra ventana. No entraremos todavía en detalles y simplemente asumiremos que la llamada

```
GtkWidget *window = construct_window ();
```

almacena una ventana que ha sido creada mediante la función `construct_window ()` en el puntero `window`.

¿Pero qué es GtkWidget? Pese a estar desarrollado en el lenguaje C, GTK+ cuenta con una jerarquía de objetos para organizar los controles y elementos forman parte de él. El sistema de objetos que se utiliza es `GObject`, un potente componente de la arquitectura de desarrollo de GNOME. Todas las clases en dicho sistema utilizan como clase base `GObject`.

`GtkWidget` es la clase base para todos los controles que están disponibles en GTK+. Toda ventana, botón, caja de texto, o menú emergente que sea parte de una aplicación, será una subclase de `GtkWidget`, que a la vez es una subclase de `GObject`.

A continuación, vemos el primer ejemplo de una llamada a un método parte de una clase de GTK+. `gtk_window_set_default_size ()` es un ejemplo perfecto de como están organizados los métodos en GTK+. Comencemos por discutir el porqué de su nombre. La convención es tal que, para organizar los cientos de funciones en GTK+ (y en general en todas las bibliotecas de GNOME), la primera parte del nombre siempre hará referencia a la biblioteca a la que es parte, en este caso, GTK+. La segunda parte del nombre, menciona la clase base de la cual la función es método. Finalmente, el nombre especificará de manera clara cual es la función del método sobre dicha clase. De esta manera, es fácil de entender que: `gtk_window_set_default_size ()` es parte del toolkit GTK+, actúa sobre objetos de la clase `GtkWindow`, y la acción que realiza es definir el tamaño por defecto de la ventana.

Con respecto a la utilización de las funciones, el primer parámetro representa el objeto sobre el cual se ejecuta el método, y debe tener siempre el tipo que el nombre de la función especifica. Como en nuestro código de ejemplo, `window` ha sido declarado `GtkWidget` y no `GtkWindow`, tenemos un problema.

Sabemos que, gracias a `GObject`, todo objeto `GtkWindow` es al mismo tiempo un objeto `GtkWidget`. Las malas costumbres de C nos sugerirían realizar simplemente una conversión explícita de tipo, la cual en nuestro ejemplo, no es

riesgosa en absoluto, dada la simplicidad de éste. Sin embargo, `GObject` provee una metodología segura para realizar dichas conversiones, la cual debe ser utilizada *siempre* al escribir software basado en `GObject`.

Cada clase basada en `GObject`, brinda un conjunto de macros que nos permiten tanto verificar el tipo de un objeto y realizar una conversión de tipo segura, entre otras funcionalidades. Al utilizar `GTK_WINDOW ()` para convertir el tipo en vez de simplemente `(GtkWindow *)`, podemos confiar en que `GObject` emitirá una advertencia en tiempo de ejecución si el objeto a convertir no es realmente un objeto de la clase `GtkWindow`.

2.2. Construyendo la interfaz: construct_window ()

La función `construct_window ()` nos permite ejemplificar uno de los aspectos fundamentales para la creación de una interfaz de GTK+: el concepto de contenedores.

Un *contenedor* en GTK+ es un control que admite la inclusión de otros controles en él. Naturalmente, una ventana sólo podría ser útil si pudiésemos insertar otros controles en ella, por lo que podemos suponer desde ya que `GtkWindow` es una subclase de `GtkContainer`, la clase base de todos los contenedores.

En particular, existe una clase de contenedores llamados `GtkBin`, que pueden tener sólo un control hijo. Ejemplos de estos son `GtkWindow` y `GtkButton`.

¿Pero cómo podemos incluir más controles en una ventana si éstas sólo admiten un hijo?

Como la idea de `GtkWindow` es simplemente ser una ventana, y no un complejo control donde se puedan insertar muchos controles hijos, existe una serie de contenedores *invisibles* que son usados para esta tarea. En nuestro ejemplo, usaremos una `GtkVBox`, un contenedor que admite una cantidad arbitraria de controles hijos ordenados de manera vertical. Entonces, primero lo creamos, mediante la llamada a

```
vbox = gtk_vbox_new (TRUE, 0);
```

y luego lo insertamos en la ventana usando el método `gtk_container_add ()`.

```
gtk_container_add (GTK_CONTAINER (window), vbox);
```

De acuerdo a la naturaleza de cada contenedor, éstos proveen convenientes funciones que permiten añadirles controles. `GtkVBox` y su contenedor hermano para ordenamientos horizontales, `GtkHBox`, son subclases de la clase abstracta `GtkBox`. Esta clase, nos brinda una serie de métodos para añadir y manipular controles hijos ordenados ya sea vertical u horizontalmente. Una vez que hayamos creado un botón y una etiqueta de texto simples, los agregaremos al ordenamiento vertical usando

```
gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);
gtk_box_pack_start (GTK_BOX (vbox), label, FALSE, FALSE, 0);
```

Los distintos parámetros que pueden darse a los métodos de las clases contenedores, permiten definir la apariencia que tendrán los controles hijos dentro del contenedor. Para poder comprenderlos de mejor manera, recomendamos leer la documentación de GTK+, y por sobre todo, practicar combinando distintas configuraciones hasta lograr entender como se comportan.

2.3. Dándole vida a los controles: on_button_clicked ()

Una interfaz con controles no es útil si dichos controles no nos permiten activar acciones y generar cambios de estado en la aplicación. Por ejemplo, cuando se cuenta con un botón en una aplicación, la necesidad de que ésta realice alguna acción cuando el botón.

Para ésto, GTK+ utiliza los conceptos de *señal* y *callback*[1]. Una señal es un evento que se activa cada vez que se realiza un cambio de estado en algún objeto, ya sea por interacción del usuario (como en el caso de un botón que es presionado) o por interacción con otras partes del programa.

Cada objeto en GObject puede tener diversas señales. Ejemplos de señales para `GtkButton` son las señales `"clicked"`, `"pressed"`, y `"released"`.

A cada señal se puede conectar uno o más callbacks. Estas son funciones que serán llamadas cada vez que la señal sea emitida. Dependiendo de la utilidad que tenga cada señal, será necesario que el callback cuente con distinta información del ambiente para saber como reaccionar. Es por esto que distintas señales exigen distintos prototipos para sus callbacks. El prototipo correspondiente a cada señal está especificado en su propia documentación.

Para poder conectar un callback a una señal, podemos utilizar la función `g_signal_connect ()`. En nuestro ejemplo, hemos definido un callback que queremos ejecutar cada vez que se presiona el botón en nuestra interfaz. Para conectarla, usamos la llamada

```
g_signal_connect (G_OBJECT (button), "clicked",
                  G_CALLBACK (on_button_clicked), label);
```

De esta manera, conectamos a la señal `"clicked"` del objeto `button` el callback `on_button_clicked ()`. El cuarto parámetro de `g_signal_connect ()` representa datos que serán pasados como parámetro a la función `on_button_clicked ()`, de modo tal que esta pueda usarlos.

Dicha función tiene un prototipo como corresponde a la señal `"clicked"`, es decir:

```
void on_button_clicked (GtkButton *button, gpointer data);
```

El primer parámetro de nuestro callback es el botón que recibe la señal, y el segundo representa un puntero genérico que recibe el dato que es entregado como cuarto parámetro a la función `g_signal_connect ()`. Para poder utilizarlo, simplemente lo convertimos (de manera segura, por supuesto) al tipo de dato que corresponde, en este caso, un `GtkLabel`

[1] En Español, callback significa *retrollamada*, pero este término es muy poco utilizado, por lo que preferimos utilizar el vocablo Inglés.

```
GtkLabel *label = GTK_LABEL (data);
```

3. Ventanas y Contenedores

En esta sección mostraremos la secuencia y código necesario para generar la estructura de contenedores necesaria para generar una ventana como la de la figura 2.

Figura 2. Ventana a crear.

Comenzamos por crear la ventana y definir su título. Además destinaremos 10 píxeles de borde entre la ventana y los elementos que la compongan:

```
GtkWidget *window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
gtk_container_set_border_width (GTK_CONTAINER (window), 10);
gtk_window_set_title (GTK_WINDOW (window), "ejemplo");
```

De esta manera, obtendremos una ventana vacía, que puede imaginarse tal como la de la figura 3.

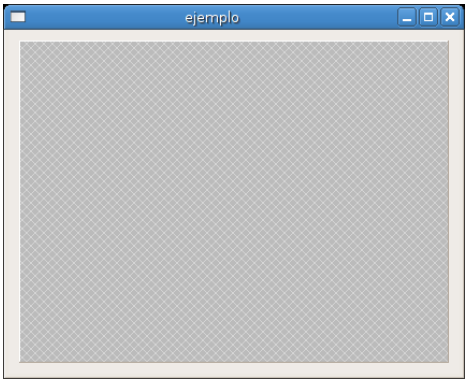

Figura 3. Ventana en blanco.

Para continuar, dividimos la ventana mediante un ordenamiento vertical, como lo muestra la figura 4. Para esto, creamos un `GtkVBox` y lo insertamos en la ventana:

```
GtkWidget *vbox = gtk_vbox_new (FALSE, 0);
gtk_container_add (GTK_CONTAINER (window), vbox);
```

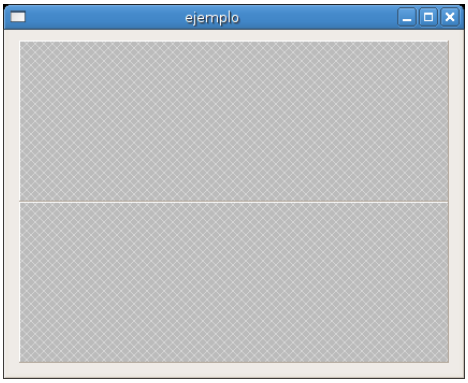

Figura 4. Ventana con tan sólo la división vertical.

Lo primero que insertamos en la división vertical es una división horizontal, para poder posicionar ahí la entrada de texto y el botón. Esta vez, necesitamos definir un espacio de 10 píxeles entre el botón y la entrada de texto, lo cual haremos mediante la propiedad `"spacing"` de `GtkHBox`. Además, al insertar la `GtkHBox` en la `GtkVBox`, definiremos un espacio de 10 píxeles entre el ordenamiento horizontal y su alrededor. La apariencia en este momento la podemos apreciar en la figura 5.

```
GtkWidget *hbox = gtk_hbox_new (FALSE, 10);
gtk_box_pack_start (GTK_BOX (vbox), hbox, FALSE, FALSE, 10);
```

Una vez que tenemos la ventana completamente organizada, creamos los controles y los insertamos. Sin entrar en el detalle de la creación de los controles (ya que analizaremos esto en la próxima sección), los insertaremos uno por uno en la ventana.

Para el caso de la entrada de texto, nos interesa que ésta se expanda cuando el usuario expanda la ventana, de modo tal que se facilite la inserción de textos largos. Por lo tanto, al empacar el botón en el ordenamiento horizontal, definimos las propiedades `"expand"` y `"fill"` como *verdadero*.

Para el botón, no nos interesa que se expanda. Los botones que se expanden se ven generalmente horribles. Por lo tanto, dichas propiedades las dejaremos en *falso*.

Figura 5. Ventana con tan sólo los ordenamientos horizontal y vertical, siendo el primero necesario para el botón y la entrada de texto.

```
gtk_box_pack_start (GTK_BOX (hbox), text_entry, TRUE, TRUE, 0);
gtk_box_pack_start (GTK_BOX (hbox), button, FALSE, FALSE, 0);
```

Hasta ahora, ya tenemos una ventana tal como la de la figura 6.

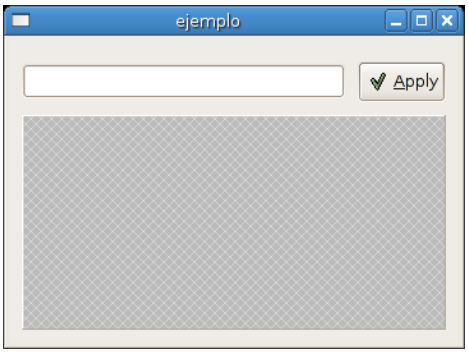

Figura 6. Ventana casi finalizada, sólo falta la etiqueta de texto.

Sólo nos queda insertar la etiqueta en el ordenamiento vertical:

```
gtk_box_pack_start (GTK_BOX (vbox), label, FALSE, FALSE, 10);
```

Y con esto ya tenemos la ventana tal como la queríamos.

4. Controles elementales

En la sección anterior hemos omitido la creación de los controles que insertamos. En esta sección nos referiremos a esa sección de código.

4.1. Botones

Los botones son uno de los elementos más comunes en una interfaz gráfica. En GTK+ contamos con varias posibilidades para crear botones.

Lo recomendado es utilizar los botones con texto e icono genérico, que son parte de `GtkStock`. Este cuenta con los típicos iconos y textos que encontraremos en una aplicación: *Aceptar*, *Cancelar*, *Sí*, *No*, *Aplicar*, etc. Una de las ventajas, además de su calidad de elementos genéricos, es que serán automáticamente presentados en el idioma en que esté configurado el escritorio donde se ejecute la aplicación.

Para nuestro ejemplo, utilizaremos el stock *Aplicar*. Para crearlo, simplemente llamamos:

```
GtkWidget *button = gtk_button_new_from_stock (GTK_STOCK_APPLY);
```

4.2. Etiquetas

Una etiqueta de texto nos permite desplegar un texto estático en una aplicación. La función que GTK+ provee para crearlas, nos permite definir inmediatamente el texto que debe desplegar. En nuestro programa, esto sería:

```
GtkWidget *label = gtk_label_new ("ejemplo");
```

Convenientemente, el texto puede contener etiquetas para darle formato. Para utilizar esto, debemos definir la propiedad `"use_markup"` como verdadero, y al momento de definir el texto, insertar las etiquetas. Por ejemplo,

```
gtk_label_set_use_markup (GTK_LABEL (label), TRUE);
gtk_label_set_text (GTK_LABEL (label), "<i>texto en cursiva</i>");
```

presentará el texto *texto en cursiva* en la etiqueta utilizando cursivas.

4.3. Entradas de texto

Una entrada de texto es un control que permite al usuario ingresar texto en una linea. Definamos a continuación una entrada de texto que admita un máximo de 20 caracteres, y que cada vez que su texto sea modificado, se ejecute una función `on_text_edited_callback ()`:

```
GtkWidget *text_entry = gtk_entry_new ();
gtk_entry_set_max_length (GTK_ENTRY (entry), 20);
g_signal_connect (G_OBJECT (text), "changed",
                  G_CALLBACK (on_text_edited_callback), NULL);
```

5. Otros recursos para la programación con GTK+

El presente artículo presenta tan sólo una breve introducción en el desarrollo de aplicaciones con GTK+. No hemos examinado exhaustivamente la API, pero esta puede ser encontrada en [1]. Un completo manual de GTK+ que explica detalladamente cada uno de los controles, con ejemplos mucho más extensivos, se encuentra en [2].

Un excelente recurso en español para la programación en GTK+, y en general, para el desarrollo de aplicaciones para GNOME es [3]. Una guía para el uso de la biblioteca libglade para facilitar la creación de la interfaz en conjunto con autotools es [4].

En el año 2004, se publicó [5], que puede ser un buen recurso para quienes prefieren un buen libro a un manual en linea. Un poco más reciente e igualmente completo es [6], que además introduce el desarrollo de aplicaciones en GTK+ utilizando otros lenguajes, como C++ y Python.

Referencias

1. GTK+ developers: *GTK+ Reference Manual*. `http://developer.gnome.org/doc/API/2.0/gtk/`.
2. Gale, T., Main, I., and the GTK+ team: *GTK+ 2.0 Tutorial*. `http://www.gtk.org/tutorial/`.
3. Moya, R., et al.: *Programación en el entorno GNOME*. `http://libros.es.gnome.org/librognome/librognome/librognome/book1.html`.
4. Saavedra, C.: *Iniciando un proyecto GNOME con libglade y autotools*. `http://www.gnome.org/~csaavedr/documents/usinglibglade/index.html`.
5. Warkus, M.: *The Official GNOME 2 Developer's Guide*.
6. Newren, E.: *Developing with Gnome In C, C++, Perl, and Python*. `http://www.gnome.org/~newren/tutorials/developing-with-gnome/html/`.