Desarrollo de aplicaciones científicas con Glade

Francisco Domínguez-Adame (adame@material.fis.ucm.es)

http://valbuena.fis.ucm.es/~adame/programacion

17 de abril de 2003

Sobre este documento

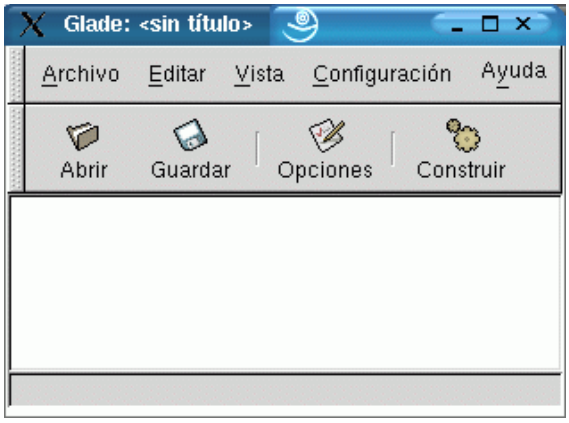
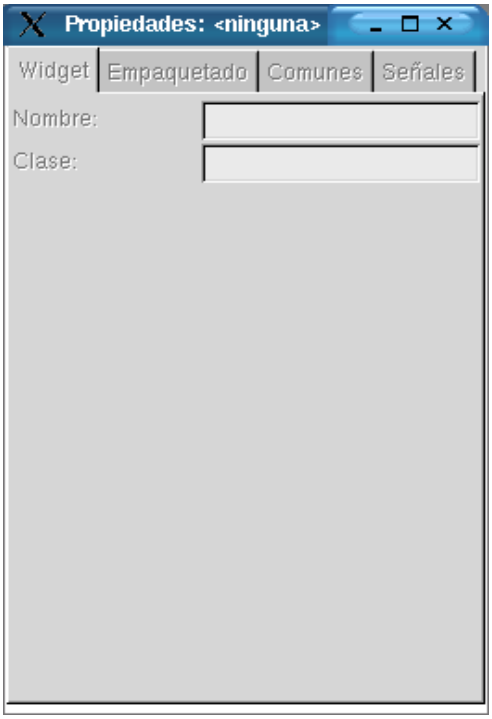
El origen de este documento se encuentra en la necesidad que encontré en mi actividad docente e investigadora de desarrollar rápidamente aplicaciones gráficas. Comencé acudiendo a las librerías Xlib, pero indudablemente es una ardua tarea que no permite un desarrollo rápido de aplicaciones a una persona con escasos conocimientos de programación (mi primer y único programa desarrollado con XLib es **xeigenfunction.c** y se puede descargar desde mi página dedicada a la programación http://valbuena.fis.ucm.es/~adame/programacion/). Por otra parte, al no ser usuario de Windows, desconozco el uso de herramientas como Visual Basic y Visual C++, por no mencionar que se trata de programas propietarios. En esta tesitura encontré las librerias GTK (http://www.gtk.org), que me proporcionaron la flexibilidad y potencia que necesitaba trabajando con Linux. La pregunta que me surgió entonces es si habría alguna aplicación gráfica para crear una interfaz de manera simple, y poder dedicar los esfuerzos al desarrollo del código inherente a la aplicación..

Sobre Glade

Y la respuesta es Glade (http://glade.gnome.org), que se encuentra disponible en todas las distribuciones Linux que conozco. Además, Glade se puede complementar perfectamente con Anjuta (http://anjuta.sourceforge.net), una plataforma integrada de desarrollo (IDE). Glade es un programa gratuito, con amplia difusión entre los desarrolladores, que permite la creación de interfaces gráficas de usuario de manera visual. Tiene la enorme ventaja de estar muy bien documentado y de continuar en proceso de mejora, además de listas de correo activas (http://lists.ximian.com/archives/public/glade-users/). Si aún no lo ha hecho, instale Glade antes de proseguir con este manual siguiendo las instruccions del gestor de sofware de su distribución de Linux (Yast2 para Suse o linuxconf para RedHat, por ejemplo).

Iniciando Glade

Para iniciar el programa desde un terminal introducimos **glade** en la línea de comandos, y veremos que aparecen tres ventanas: **Ventana principal**, **Paleta de Widgets** y **Editor de propiedades**., según se muestra en la siguientes imágenes.

Ventana principal	Paleta de Widgets	Editor de propiedades
		

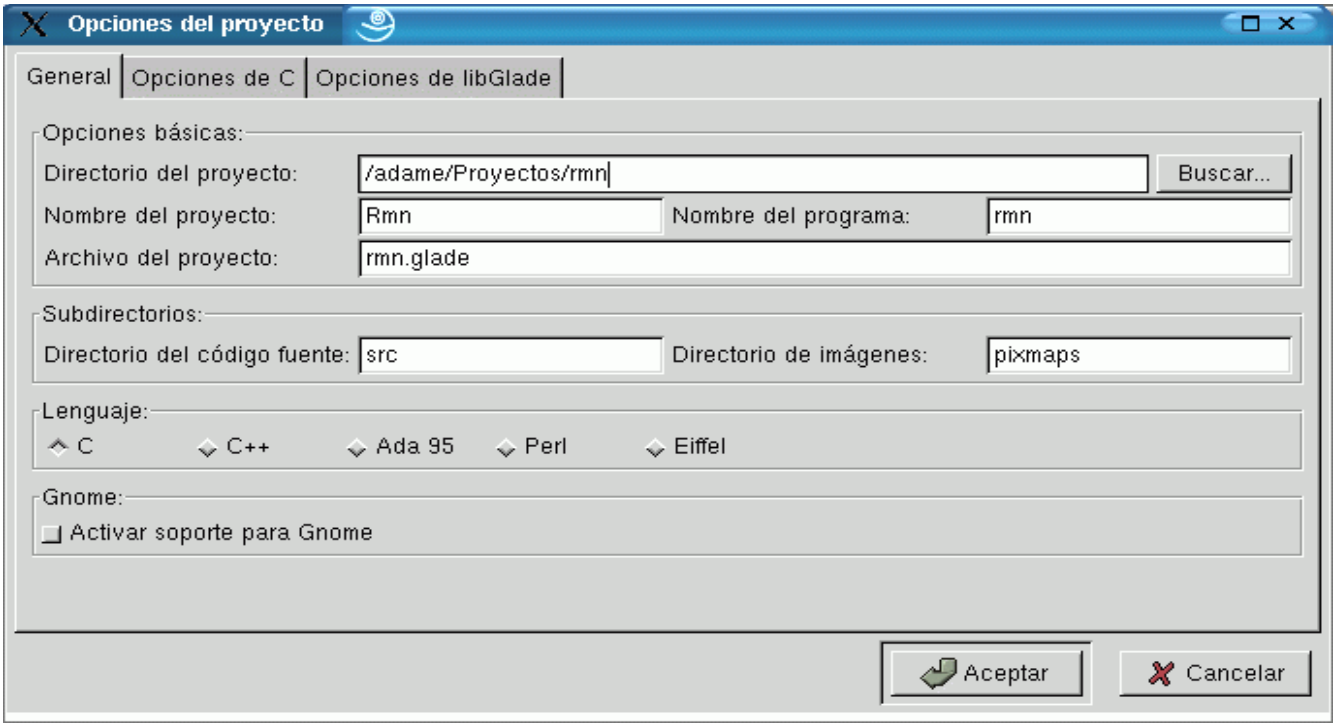
La **Ventana principa**l muestra los widgets que vayamos añadiendo a nuestra aplicación. (en la imagen superior no se ha añadido ninguno todavía). Por *widget* entenderemos cualquier componente que queramos incluir en la interfaz, por ejemplo Botones, Barra de menú, Barras de progreso, Cuadro de diálogo, etc. A lo largo de este manual mantendremos dicho término y no será traducido al castellano.

La **Paleta de Widgets** permite seleccionar qué elementos agregaremos a nuestra aplicación. En el ejemplo que presentaremos más adelante comenzaremos con una ventana. Para ello pulsamos en primer lugar sobre el icono de la esquina superior izquierda de la **Paleta de Widgets Básicos**. y sobre ella iremos añadiendo el resto de elementos, como indicaremos en el ejemplo.

El **Editor de propiedades** se activa cuando un widget esta seleccionado y permite modificar sus atributos (en la imagen superior permanece inactivo porque aún no hemos incluido ningún widget en nuestra interfaz).

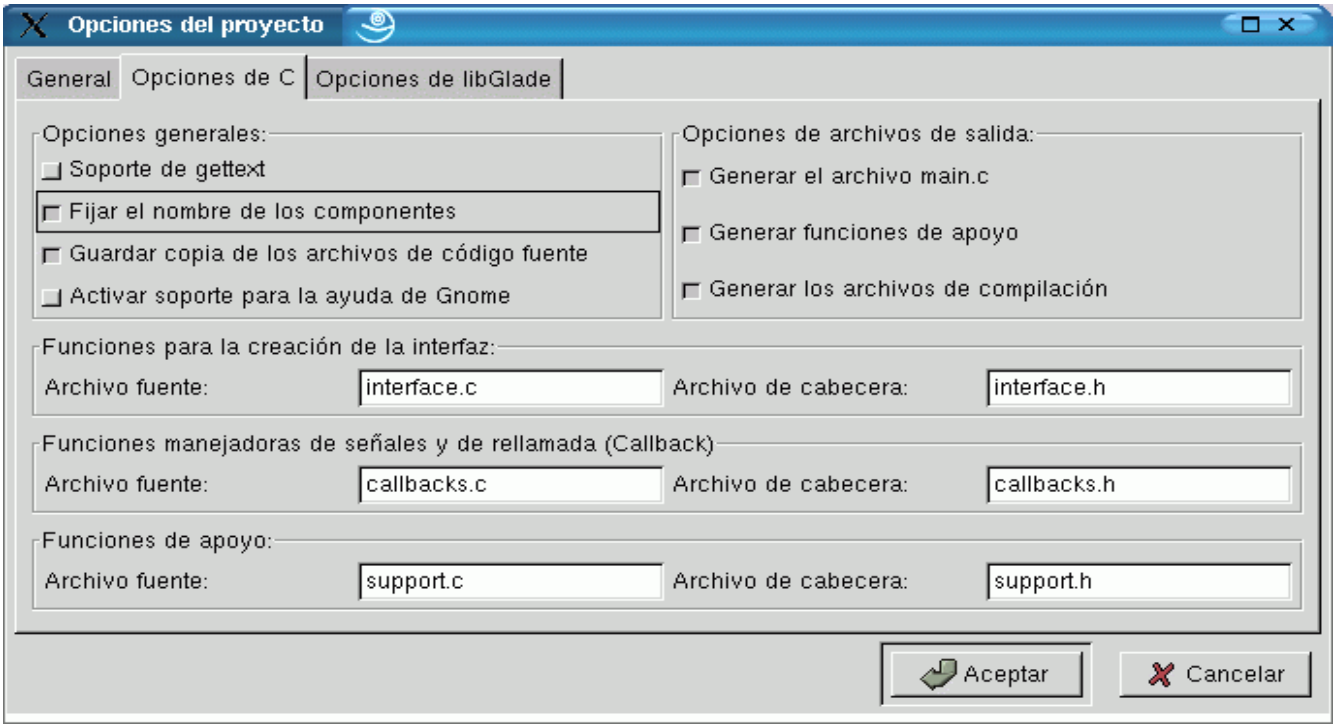
Iniciando un nuevo proyecto

En la opción *Archivo* de la **Ventana principal** elegimos *Nuevo Proyecto* y, lógicamente, respondemos afirmativamente a la pregunta de Glade sobre si realmente deseamos crearlo. Posteriormente pulsaremos sobre las *Opciones del Proyecto* y se desplegará una ventana como la que mostramos a continuación.



En esta primera pantalla podemos escoger en que directorio se guardará nuestro proyecto, con que nombre y en que directorio se guardarán los archivos con los códigos y los mapas de pixels (en nuestro ejemplo no vamos a incluir imágnes prediseñadas en la interfaz). Si no vamos a desarrollar una aplicación para Gnome, como será nuestro caso, debemos **desactivar** el correspondiente botón de la parte inferior izquierda. En nuestro caso el *Nuevo Proyecto* se llamará Rmn (resonancia magnética nuclear) y dejamos activada la casilla que indica que el lenguaje de programación es C (viene marcada por defecto).

Podemos acceder a la segunda ventana pulsando sobre la pestaña denominada *Opciones de C*. En esta segunda ventana podemos configurar los archivos donde se declararán las funciones que controlan los eventos, aunque normalmente estos se dejan como están. Si no va a trabajar con soporte multilingüe, recomiendo **desactivar** la casilla denominada *Soporte de gettext* (véase la imagen inferior). Finalmente, para el ejemplo que desarollaremos en este manual hay que **activar** la casillas *Fijar el nombre de los componentes*, como indica la imagen inferior. En nuestro ejmplo, no necesitamos modificar los datos de la tercera ventana, a la que accederíamos pulsando sobre la pestaña denominada *Opciones de libGlade*, por lo cual pulsamos Aceptar para finalizar la selección de opciones..



Comenzando el desarrollo de la interfaz

Antes de comenzar el desarrollo, es conveniente detenerse unos instantes para pensar cómo queremos distribuir los distintos widgets de la interfaz, cúales nos harán falta (por ejemplo, una Entrada de texto para indicar el valor de una variable necesaria para la ejecución de la aplicación, un Área de Dibujo, una Barra de menú o una Barra de herramientas). Las aplicaciones pueden tener una o más Ventanas, Cuadros de diálogo, etc. Nuestro sencillo ejemplo sólo tendrá una Ventana principal sobre la que iremos añadiendo los distintos widgets. Por tanto, antes de comenzar debemos hacer una breve presentación de la **Paleta de Widgets** . En dicha paleta, los iconos de los widgets se agrupan en tres clases: GTK+ Básico, GTK+ Adicional y Gnome. Para acceder a ellos pulsamos sobre la correspondiente pestaña en la parte superior de la **Paleta de Widgets.**. Para saber el nombre de cada uno de ellos basta dejar unos breves instantes el puntero del ratón sobre el icono, hasta que se despliegue una pequeña ventana indicando el nombre del mismo..

		
Widgets GTK+ Básicos	**Widgets GTK+ Adicionales**	**Widgets GTK+ Gnome**
Widget más habituales en el desarrollo de la interfaz. Son los únicos que utilizaremos en este manual.	Aquí se presentan widgets algo más especiíficos y no serán discutidos en este manual.	Presenta los widgets propios del desarrollo de aplicaciones Gnome. Tampoco los discutiremos en este manual.

En la aplicación que vamos a desarrollar como ejemplo a lo largo de este manual utilizaremos muy pocos widgets (al fin y al cabo es sólo un sencillo ejemplo para mostrar las capacidades de Glade y GTK). Por ello, conviene familiarizarse con sus correspondientes iconos desde ahora., como indica la siguiente Tabla.

Icono	Nombre	Icono	Nombre	Icono	Nombre
	*Ventana*		*Caja vertical*		*Barra de menú*
	*Marco*		*Área de Dibujo*		

Conviene recordar que, a diferencia de lo que sucede en muchos lenguajes visuales, la mayoría de los Widgets deben estar insertados en un *Contenedor*. Hay muchos tipos de *Contenedores* (por ejemplo, los Widgets que aparecen en las tres últimas filas de la **Paleta de Widgets** *GTK+ Básicos* son *Contenedores*), y en nuestro ejemplo utilizaremos tres, a saber, la *Ventana*, el *Marco* y la *Caja Vertical*. Sobre ellos podemos colocar otros Widgets, pero Glade se quejará si intentamos colocar un Widget sobre otro que no es un contenedor (por ejemplo, en nuestra aplicación no podemos insertar un Widget sobre el *Área de Dibujo*, ya que no se trata de un *Contenedor*).

Eventos

Cada uno de estos Widgets está asociado al menos a un evento, así que debemos dedicar unas líneas a este concepto. Siempre que pulsamos una tecla o presionamos algún botón del ratón se produce un evento y se emite una señal. Los Widgets de la interfaz deben respoder a uno o más eventos. Por ejemplo, podemos diseñar un elemento de un menú que se llame *Salir*, y precisamente queramos que la aplicación responda de una determinada manera al pulsar sobre dicho elemento (terminando la ejecución de la aplicación en este caso). Hay muchísimos eventos y señales, y la mayoría de ellos

van a ser ignorados por la aplicación (por ejemplo, el teclado no va a jugar ningún papel en la aplicación que vamos a desarrollar aquí, así que las pulsaciones de las teclas serán simplemente ignoradas). Si queremos que un elemento de la interfaz responda a un determinado evento, debemos escribir una función que indique al programa la respuesta que queremos que ofrezca a tal acción. A estas funciones se les *retrollamadas* (callbacks en inglés), y son declaradas como prototipos en el archivo *callbacks.h* y llamadas en *callbacks.c*. Ambos archivos son generados automáticamente por Glade cuando pulsamos **Archivo** → **Escribir código fuente** de la **Ventana principal**. En el ejemplo del elemento *Salir*, encontraremos la siguiente función en el *callbacks.c:*

```
on_boton_salir_activate()
{
  Líneas de código;
}
```

y ahí podremos escribir el código que corresponda a lo que queremos que el programa haga cuando presionemos dicho botón. ¿Cómo sabe Glade las señales a las que debe responder un determinado elemento y, por tanto, qué funciones de retrollamada debe incluir en los archivos *callbacks.c* y *callbacks.h* ? Pues bien, si nos detenemos de nuevo en el **Editor de Propiedades** (recuerde, una de las tres ventanas que aparecen al iniciar Glade), veremos que tiene cuatro pestañas (Widget, Empaquetado, Comunes y Señales) y será en la última de ellas dónde indicaremos a Glade cual o cuales serán las señales a las que deba responder un determinado elemento de la interfaz. En el desarrollo del ejemplo explicaremos este aspecto con algo más de detalle.

Resonancia magnética nuclear

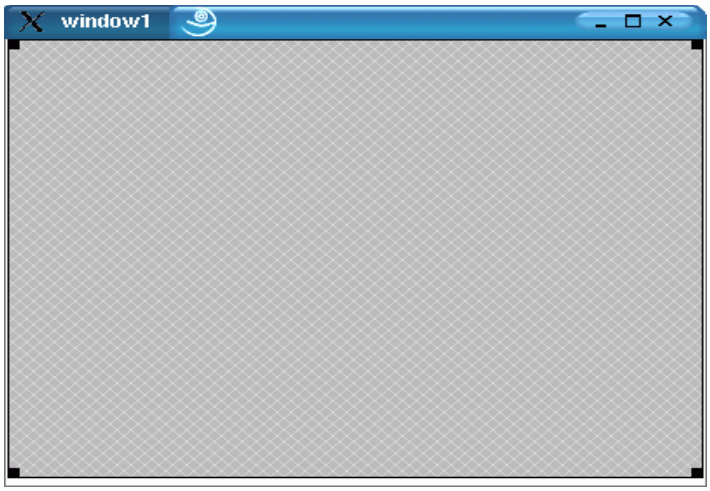
Llegados a este punto, hagamos un alto en el camino para presentar el problema que queremos resolver mediante una aplicación basada en GTK. La resonancia magnética nuclear en los sólidos está asociada con los efectos dinámicos del espín nuclear sometido a la acción de campos magnéticos intensos. Las ecuaciones de Bloch para las componentes transversales de la magnetización **M** se escriben como (véase **Física del Estado Sólido: Teoría y Métodos Numéricos**, por Francisco Domínguez-Adame, publicado por la Editorial Paraninfo en 2000, http://valbuena.fis.ucm.es/fes)

$$\frac{dm_x}{d\tau} = \frac{2\pi}{\alpha} m_y - m_x \qquad \frac{dm_y}{d\tau} = -\frac{2\pi}{\alpha} m_x - m_y$$

donde $\mathbf{m}=\mathbf{M}_\perp/|\mathbf{M}_\perp(t=0)|$ es la magnetización transversal reducida, $\tau=t/T$ es el tiempo expresado en unidades del tiempo de relajación transversal T, y $2\pi/\alpha=\gamma BT$, siendo γ la razón giromagnética y B el campo magnético. Las condiciones iniciales son $m_x(0)=1$ y $m_y(0)=0$. Es claro que este sencillo sistema de ecuaciones tiene solución analítica, pero en nuestro caso vamos a integralarlas numéricamente mediante el método de Runge-Kutta de cuarto orden y a representarla gráficamente. Todo ello lo haremos a través de la aplicación que presentamos a continuación.

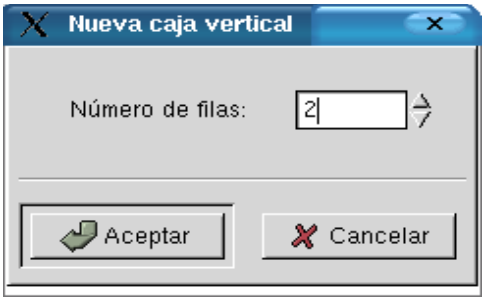
Sin más dilación, pues, comencemos el desarrollo del ejemplo, lo que nos servirá para repasar conceptos ya explicados y profundizar en otros.Primero ejecutamos **glade** y en la **Paleta de widgets** presionamos el icono de ventana:

con lo que se creará una ventana como la que se muestra a la derecha. Esta ventana es el widget que contendrá los demás widgets de la aplicación.

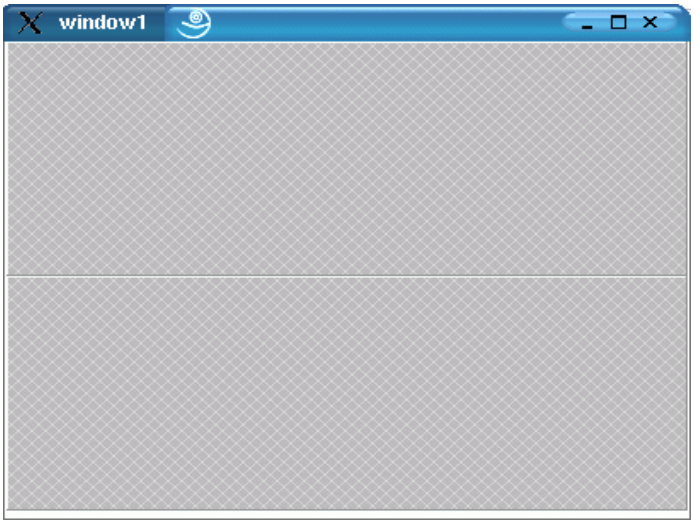


A continuación presionaremos el icono de **Caja vertical** en la **Paleta de Widgets**

y llevamos el puntero del ratón sobre la ventana principal que acabamos de crear. Obervaremos que el puntero cambia al símbolo + y presionamos sobre la ventana. Entonces aparecerá una diálogo donde indicaremos cuantas cajas queremos (en nuestro caso solamente dos)

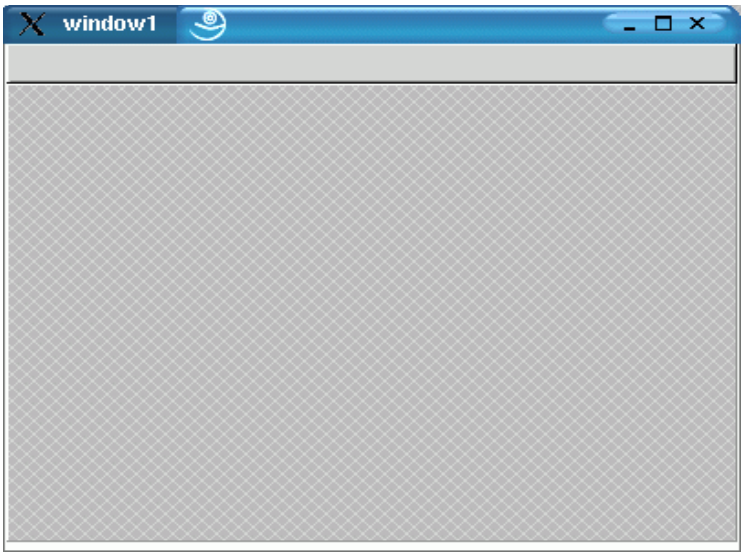


La ventana principal aparece entonces dividida en dos cajas colocadas verticalmente como aparece en la siguiente figura

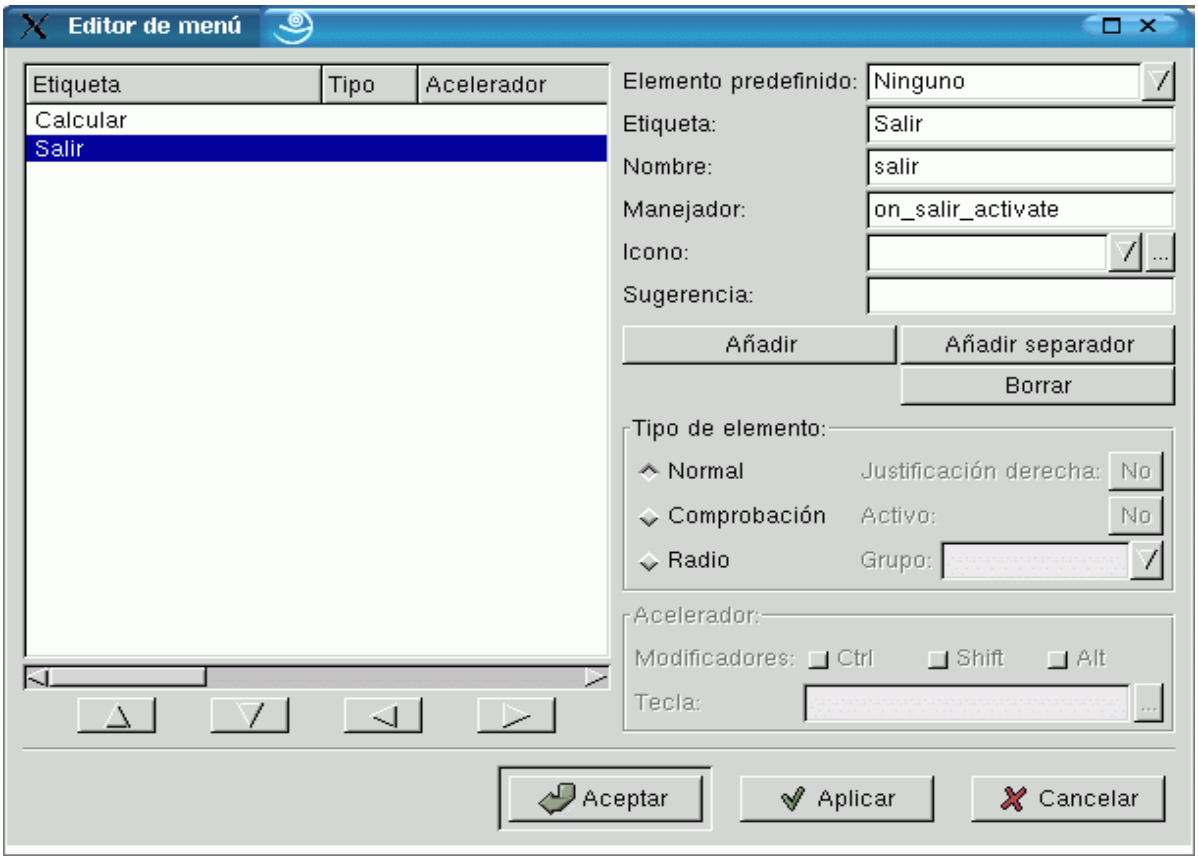


A continuación presionaremos el icono de barra de menús:

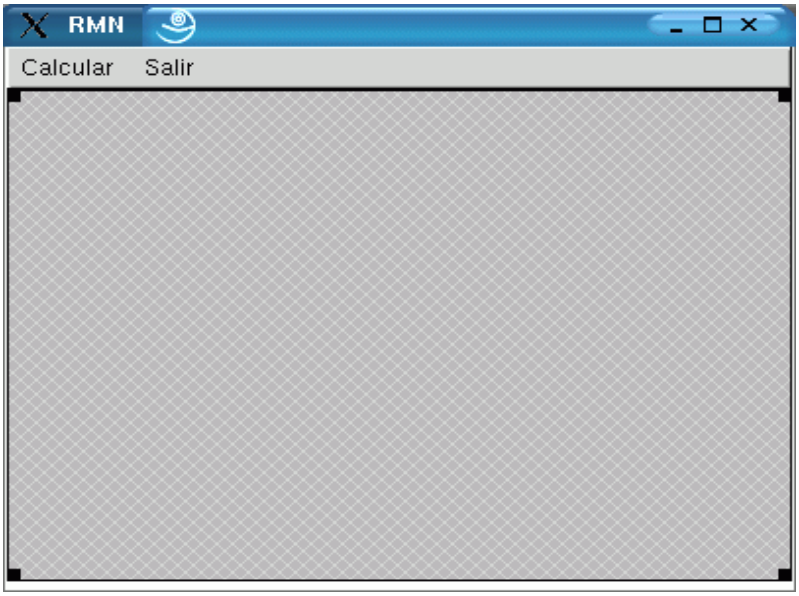
y pulsamos sobre la caja superior, con lo que la ventana tomará el siguiente aspecto:



Al **Editor de Menú** se accede pulsando sobre la **Barra de Menú** con el botón derecho del ratón, con lo que se mostrará la ventana que aparece a la derecha en la imagen de arriba, donde seleccionaremos **Editar Menú**. Con el botón **Añadir** se crean los elementos de menú. Añadiremos las entradas **Calcular** y **Salir** en las correspondientes etiquetas, y los widgets se llamarán *calcular* y *salir*, respectivamente. Así, Glade llamará **on_calcular_activate** y **on_salir_activate** a las retrollamadas asociadas a los elementos del menú.

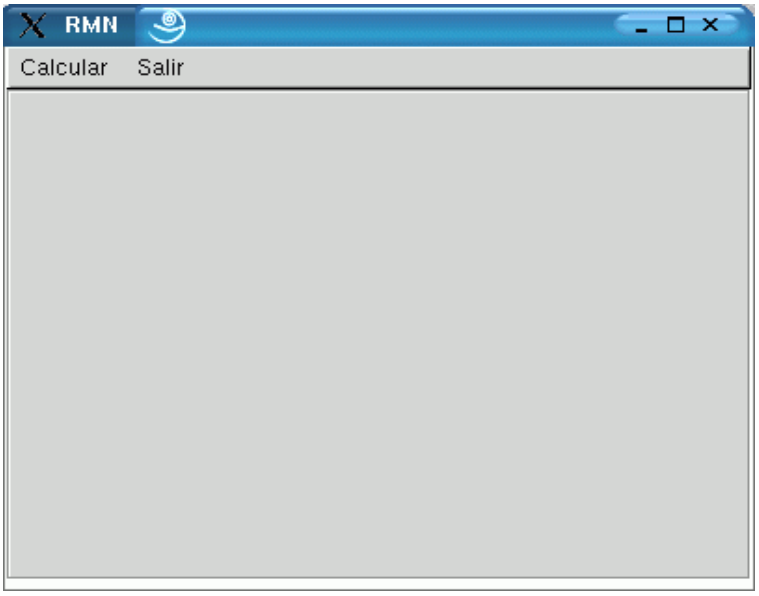


Con todo esta acción, la ventana que nos muestra Glade tiene ya la siguiente apariencia y va tomando la forma deseada.

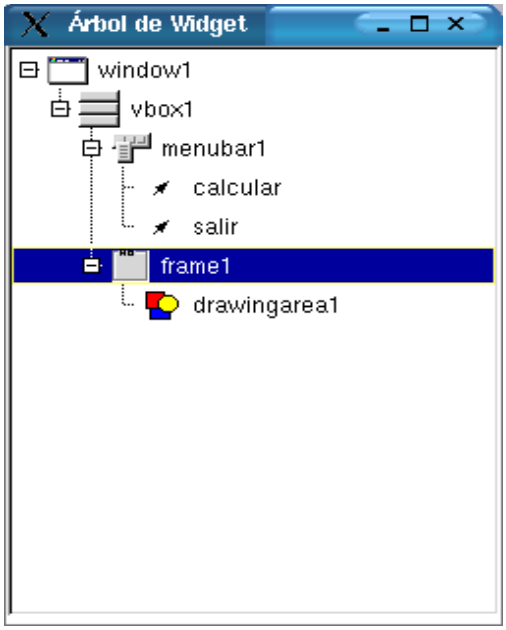


Finalmente, para terminar nuestra interfaz vamos a agregar un marco como adorno y un Área de Dibujo en el cuadro inferior de la **Caja vertical**, presionando sucesivamente sobre los iconos indicados abajo (recordemos que tras pulsar un icono de la **Paleta de widgets** debemos también pulsar sobre la ventana de la aplicación que estamos desarrollando para indicar a Glade dónde queremos ubicar dicho widget)

siendo el aspecto final el que se muestra a la derecha.



Antes de terminar esta fase vamos a retocar ligeramente las propiedades del marco para destacarlo un poco más. Para allo podemos presionar sobre él, o bien acudir a la **Ventana principal** de Glade, pulsando sobre **Ver** y a continuación sobre **Mostrar árbol de widgets.**. Tras hacerlo aparece una nueva ventana que contiene el árbol de nuestra aplicación, y podemos seleccionar cada uno de los widgets por separado. Si presionamos sobre **frame1** veremos que sus propiedades aparecen en el **Editor de Propiedades**. Posteriormente modificaremos el valor del **Ancho del borde** (por ejemplo, 10):

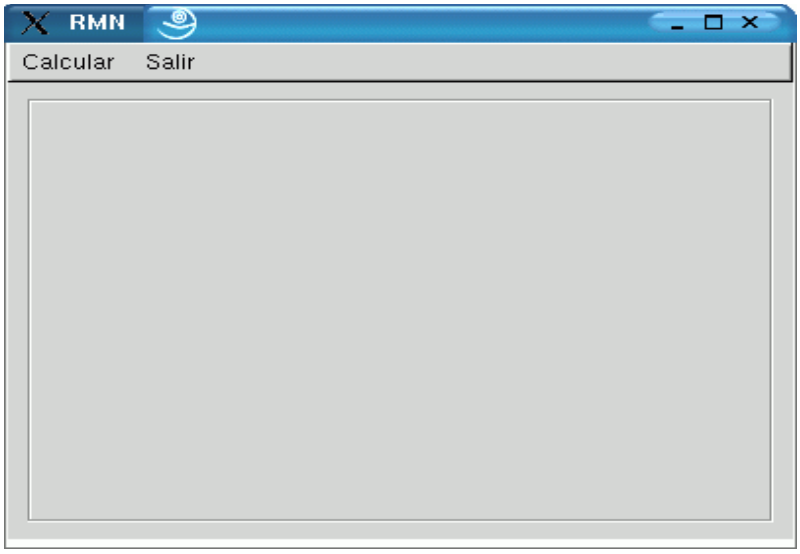


Árbol de widgets

Propiedades del marco

También podemos modificar el nombre de la ventana principal de la aplicación (**window1**) y escoger alguno más representativo, por ejemplo **RMN**. Dicho título será el que aparezca en la ventana principal de nuestra aplicación una vez terminada. Asimismo, en la pestaña llamada Comunes se puede seleccionar el tamaño deseado de la ventana (por ejemplo, 500 x 400). Y con esto hemos terminado el diseño de nuestra aplicación. Ahora debemos salvar el proyecto y generar el código fuente (**Archivo → Escribir código fuente**).

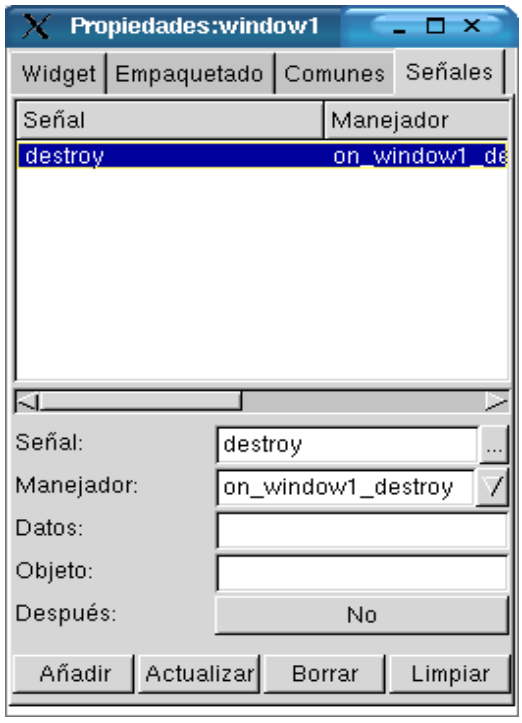
Llega el momento de compilar la aplicación. Para ello, nos dirigimos al directorio donde indicamos a Glade que queríamos guardar el proyecto. Además del fichero rmn.glade encontraremos, entre otros, un único ejecutable llamado **autogen.sh**, que debemos ejecutar para obtener los ficheros **configure** y **Makefile**.. Si introducimos en la línea de comandos el clásico **make** obtendremos el ejecutable src/**rmn**. Ahora podemos ejecutar el comando **./src/rmn** y aparece la interfaz:



Como podremos comprobar no hace aún gran cosa. De hecho, si intentamos teminar el programa pulsando sobre la × de la barra superior veremos que desaparece la misma, pero el programa sigue ejecutándose (se ejecuta en primer plano porque no hemos utilizado el símbolo **&** y no se devuelve el control al terminal). Sólo nos queda como solución introducir Ctrl-C o bien abrir otra ventana terminal y utilizar el comando **kill PID** (PID, identificación del proceso).

Este comportamiento anómalo es fácil de entender pues no hemos indicado en ningún momento cómo debe responder el programa a los distintos eventos (pulsaciones del ratón, teclado, etc). Así que ha llegado la hora de enfrentarse al manejo de señales y eventos (la explicación detallada está fuera del alcance de este breve manual, por lo que se recomienda la consulta de alguno de los enlaces o libros que indicamos en las referencias).

Para conseguir que el programa responda a la indicación de finalizar, volvemos a Glade y seleccionamos la ventana principal (**window1**) en el **Árbol de widgets**.. En la pestaña denominada **Señales** encontramos lo que necesitamos. Elegimos la señal **destroy** y pulsamos el botón **Añadir** para incorporarla. Una vez hecho, podemos volver a salvar el proyecto Glade y generar de nuevo el código fuente.



En la interfaz tenemos entonces tres elementos sobre los que podemos actuar, a saber, los dos elementos del menú (**Calcular** y **Salir**), y el símbolo × de la barra superior. Si editamos el archivo *src/callbacks.c* veremos que precisamente Glade ha creado tres funciones, una relacionada con cada elemento. El programador debe entonces completarlas para que la aplicación se comporte de acuerdo con el diseño previsto. Lo primero que haremos es modificar la función **on_window1_destroy** del archivo *callbacks.c* , incluyendo la sentencia gtk_main_quit(); con lo que quedaría

```
void
on_window1_destroy    (GtkObject      *object,
                       gpointer        user_data)
{
gtk_main_quit();
}
```

Podemos volver a compilar, ejecutando **make**, para convencernos de que ahora sí se interrumpe el programa al pulsar el símbolo × de la barra superior. Pues, si ya funciona, por qué no hacer lo mismo con el elemento **Salir** del menú? Con ello, la función **on_salir_activate** del archivo callbacks.c quedaría así

```
void
on_salir_activate       (GtkMenuItem     *menuitem,
                         gpointer         user_data)
{
gtk_main_quit();
}
```

Para terminar el código debemos hacer que el programa resuelva las ecuaciones diferenciales y represente el resultado gráficamente sobre el área de dibujo **drawingarea1**. Para ello vamos a modificar la función **on_calcular_activate** del archivo *callbacks.c* . Puesto que no hemos previsto que la interfaz modifique ningún parámetro del problema (básicamente el valor de la constante α, el tiempo máximo de integración y el número de pasos temporales), el programa sólo debe ejecutarse una vez. Esto lo podemos controlar con una variable booleana que llamaremos calculado. Al comenzar la ejecución del aplicación la variable toma el valor FALSE, pero la cambiaremos a TRUE cuando termina el cálculo. La función **on_calcular_activate** entonces sería la siguiente:

```
void
on_calcular_activate            (GtkMenuItem     *menuitem,
                                 gpointer         user_data)
{

 /* Sólo se activa si calculado==FALSE */

 if(!calculado) {
  calcular();
  calculado=TRUE;
  dibujar((GtkWidget *) menuitem);
 }

}
```

Al principio del archivo *calbacks.c* indicaremos todas las constantes y arreglos necesarios para realizar el cálculo:

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#define ALFA    0.5       /* Valor del parámetro alfa            */
#define TMAX    2.5       /* Tiempo reducido máximo              */
#define NT   2000         /* Número de pasos temporales          */
#define MX0     1.0       /* Imanación reducida x inicial        */
#define MY0     0.0       /* Imanación reducida y inicial        */
#define PI      3.141592654

/* Arreglos empleados en el cálculos                             */

float m[2];         /* Almacena las componentes de la imanación  */
                    /* en cada paso temporal                     */
float f[2];         /* Almacena las derivadas de la imanación    */
                    /* en cada paso temporal                     */
float mx[NT];       /* Almacena la componente X de la imanación  */
                    /* en función del tiempo                     */
float my[NT];       /* Almacena la componente Y de la imanación  */
                    /* en función del tiempo                     */

/* Variable de control para comprobar si se ha ejecutado el código */

gboolean calculado=FALSE;
```

Asimismo, las funciones que realizan el cálculo numérico, basado en el método de Runge-Kutta de cuarto orden de paso fijo, son las siguientes

```
/* Defínción de las funciones a utilizar durante el cálculo          */

void funcion (float *m,float *f) {

  f[0]=(2.*PI/ALFA)*m[1]-m[0];
  f[1]=-(2.*PI/ALFA)*m[0]-m[1];

}

void runge_kutta (float *m,float *f,float h) {
  int i;
  float k2[2],k3[2],k4[2],m_aux[2];
  for(i=0;i<=1;i++) m_aux[i]=m[i]+0.5*h*f[i];
  funcion(m_aux,k2);
  for(i=0;i<=1;i++) m_aux[i]=m[i]+0.5*h*k2[i];
  funcion(m_aux,k3);
  for(i=0;i<=1;i++) m_aux[i]=m[i]+h*k3[i];
  funcion(m_aux,k4);
  for(i=0;i<=1;i++) m[i]+=h*(f[i]+2.*k2[i]+2.*k3[i]+k4[i])/6.;
}

void calcular(void) {
  int i;
  float dt=TMAX/(float)NT;
  mx[0]=m[0]=MX0;          /* Condiciones iniciales          */
  my[0]=m[1]=MY0;
  funcion(m,f);            /* Derivadas iniciales            */
  for(i=1;i<NT;i++) {
    runge_kutta(m,f,dt);
    funcion(m,f);
    mx[i]=m[0];
    my[i]=m[1];
  }
}
```

Al terminar el cáculo debemos hacer que la variable calculado tome el valor TRUE en la función **on_calcular_activate** . Es conveniente comentar que esta forma de proceder, esto es, separando las funciones de cálculo y dibujo, demanda más memoria que si hiciéramos ambos procesos simultáneamente. Se ha preferido la primera opción buscando la claridad, pero es seguro que lector sabrá cómo evitar el uso de los vectores **mx[ ]** y **my[ ]**.

La preparación de la función de dibujo no requiere demasiados ajustes porque conocemos de antemano los valores máximo y mínimo de todas las magnitudes [las componentes de la imanación reducida tienen módulo menor que la unidad y el tiempo es menor o igual que NT, en unidades del paso temporal dt definido en la función **calcular()**]. Sin embargo, no todo es tan sencillo como puede parecer a primera vista, debido a las peculiariedades del widget Área de Dibujo. El problema reside en que cada vez que la ventana de la aplicación sea ocultada por otra ventana o modifiquemos su tamaño, debemos rehacer el dibujo. La solución más rápida es utlizar un mapa de pixels (pixmap) que guarde la información en segundo plano, y cada vez que reaparezca la ventana principal utilizarlo para realizar de nuevo el dibujo. Si cambiamos el tamaño de la ventana principal, debemos actualizar también el mapa de pixels. Una vez comprendido esto, la técnica es bastante sencilla de implementar.

Lo primero que advertimos es que el Área de Dibujo debe responder a nuevos eventos. Para ello acudimos otra vez a Glade y añadimos dos señales a **drawingarea1**, a saber **configure_event** y **expose_event**. Cuando salvemos de nuevo el proyecto y generemos el código fuente veremos que han aparecido las correspondientes funciones en el archivo *callbacks.c*. El primer evento (configure_event) se envía para indicar que el Área de Dibujo ha sido creada o cambiada de tamaño. El segundo evento (expose_event) se envía cuando el Área de Dibujo necesita redibujarse, por ejemplo, si se mueve otra ventana del escritorio dejando el Área de Dibujo al descubierto.

En el archivo *callbacks.c* incluiremos todo lo necesario para tratar estos aspectos gráficos. En primer lugar, definimos una estructura **typGraphics** cuyos elementos serán el mapa de pixels y la información de color. Además, como último paso previo, deberemos diseñar la función **NewGraphics ( )** que reserva memoria para dicha estructura:

```
/* Estructura para el dibujo en segundo plano                    */

typedef struct {

   GdkDrawable *pixmap;
   GdkGC *gc;

} typGraphics;

static typGraphics *g = NULL;
```

```
/* Defínción de los colores a utilizar para el dibujo            */

GdkGC *penBlack    = NULL;
GdkGC *penRed      = NULL;
GdkGC *penBlue     = NULL;
GdkGC *penWhite    = NULL;

GdkGC *GetPen (typGraphics *g, int nRed, int nGreen, int nBlue)
{
   GdkColor *c;
   GdkGC *gc;

   c = (GdkColor *) g_malloc (sizeof (GdkColor));
   c->red = nRed;
   c->green = nGreen;
   c->blue = nBlue;

   gdk_color_alloc (gdk_colormap_get_system (), c);

   gc = gdk_gc_new (g->pixmap);
   gdk_gc_set_foreground (gc, c);

   return (gc);
}
```

```
typGraphics *NewGraphics ()
{
   typGraphics *gfx;

   gfx = (typGraphics *) g_malloc (sizeof (typGraphics));
   gfx->gc = NULL;
   gfx->pixmap = NULL;

   return (gfx);
}
```

Ahora ya están disponibles todos los elementos necesarios para definir la función de retrollamada **configure_event** del archivo *callbacks.c*, que quedarían como sigue

```
gboolean
on_drawingarea1_configure_event       (GtkWidget         *widget,
                                       GdkEventConfigure  *event,
                                       gpointer           user_data)
{

/* Creamos el pixmap para el dibujo en segundo plano          */

  if (g == NULL) {
     g = NewGraphics ();
  }

  if (g->pixmap) {
     gdk_pixmap_unref (g->pixmap);
  }

  g->pixmap = gdk_pixmap_new (widget->window,
                              widget->allocation.width,
                              widget->allocation.height,
                              -1);
  dibujar(widget);
  return TRUE;
}
```

Con la función **configure_event** inicializamos el pixmap, reservando memoria, y dibujamos sobre el Área de Dibujo llamando la función **dibujar( )**, que se tratarán posteriormente .

La función de retrollamada **expose_event** del archivo *callbacks.c* es más sencilla porque el mapa de pixel ya está creado y sólo utilizamos la función **gdk_draw_pixmap ( )** de las librerías gráficas GDK que realiza la tarea encomendada.

```
gboolean
on_drawingarea1_expose_event          (GtkWidget         *widget,
                                       GdkEventExpose    *event,
                                       gpointer          user_data)
{
  gdk_draw_pixmap (widget->window,
        widget->style->fg_gc[GTK_WIDGET_STATE (widget)],
        g->pixmap,
        event->area.x, event->area.y,
        event->area.x, event->area.y,
        event->area.width, event->area.height);
  return FALSE;
}
```

Para acabar el código tenemos que implementar la función **dibujar( )** y, para ello, conviene hacer unas aclaraciones sobre las coordenadas en la pantalla. Lo primero que debemos recordar es que dicha coordenas representa el número de pixels desde un punto en las direcciones horizontal (eje X) y vertical (eje Y, hacia abajo) a un origen, que es el vértice superior izquierdo del Área de Dibujo. Aclarado este asunto, podemos discutir ya la función de dibujo, cuyo código es el que presentamos a continuación:

```
void dibujar(GtkWidget *widget){

  int n_puntos=NT;
  GdkRectangle update_rect;
  int i;

  /* Identificamos el Área de Dibujo. La función lookup_widget()
     se encuentra en support.c */

  GtkWidget *where_to_draw =lookup_widget(widget,"drawingarea1");
```

```
  /* Se utilizará para el dibujo de texto                        */

  GdkFont *label_font=NULL;
  gchar label[50];

  /* Tamaño, anchura del borde, longitud ticks, separacion tick-etiqueta*/

  gint x_size, y_size, b_w=50, l_t=10, s_t=5;

  x_size = where_to_draw->allocation.width-2*b_w;
  y_size = where_to_draw->allocation.height-2*b_w;

  /* Crea  los lápices de colores para el dibujo  */

  penBlack    = GetPen (g, 0, 0, 0);
  penRed      = GetPen (g, 0xffff, 0, 0);
  penBlue     = GetPen (g, 0, 0, 0xffff);
  penWhite    = GetPen (g, 0xffff, 0xffff, 0xffff);

  if( label_font == NULL)
   label_font = gdk_font_load("-*-helvetica-*-r-normal-*-14-*-*-*-*-*-*-*");

/* Dibuja un rectángulo  relleno y blanco sobre el Área de Dibujo          */

  gdk_draw_rectangle(g->pixmap,penWhite,
                           TRUE,0,0,
                           where_to_draw->allocation.width,
                           where_to_draw->allocation.height);

  /* Dibuja el perímetro mediante un rectángulo azul sin rellenar            */

  gdk_draw_rectangle(g->pixmap, penBlue,
                  FALSE,0,0,
                           where_to_draw->allocation.width-1,
                           where_to_draw->allocation.height-1);
  /* Dibuja los ejes cartesianos                                 */

  gdk_draw_line(g->pixmap, penBlack,
                b_w, b_w+y_size/2,
                b_w+x_size, b_w+y_size/2);

  gdk_draw_line(g->pixmap, penBlack,
                b_w, b_w,
                b_w, b_w+y_size);

  /* Dibuja los ticks                                            */

  for(i=0;i<3;i++) {

    gdk_draw_line(g->pixmap, penBlack,
                  b_w-l_t, b_w+i*y_size/2,
                  b_w, b_w+i*y_size/2);

     gdk_draw_line(g->pixmap, penBlack,
                  b_w+2*i*x_size/5, b_w+y_size/2+l_t,
                  b_w+2*i*x_size/5, b_w+y_size/2);
  }

   /* Dibuja las etiquetas                                        */
```

```
for(i=0;i<3;i++) {

  sprintf(label,"%2d",1-i);

  gdk_draw_string(g->pixmap, label_font, penBlack,
            b_w-l_t-gdk_string_width(label_font,label)-s_t,
            b_w+i*y_size/2+gdk_string_height(label_font,label)/2,
            label);
}

for(i=1;i<3;i++) {

  sprintf(label,"%1d",i);

  gdk_draw_string(g->pixmap, label_font, penBlack,
            b_w+2*i*x_size/5-gdk_string_width(label_font,label)/2,
            b_w+y_size/2+l_t+gdk_string_height(label_font,label)+s_t,
            label);
}
label_font = gdk_font_load("-*-symbol-*-*-normal-*-18-*-*-*-*-*-*-*");
sprintf(label,"t");
gdk_draw_string(g->pixmap, label_font, penBlack,
            b_w+2*s_t+x_size,
            b_w+y_size/2+gdk_string_height(label_font,label)/2,
            label);

/* Dibuja los puntos y leyendas si calculado==TRUE                */

if(calculado) {

  for(i=0;i<n_puntos;i++) {

     gdk_draw_point(g->pixmap, penBlue,
             b_w+i*x_size/n_puntos, b_w+(gint)((1-mx[i])*y_size/2));

     gdk_draw_point(g->pixmap, penRed,
             b_w+i*x_size/n_puntos, b_w+(gint)((1-my[i])*y_size/2));
  }

    /* Leyendas                                          */

     gdk_draw_line(g->pixmap, penBlue,
                b_w+2*x_size/5, b_w,
                b_w+3*x_size/5, b_w);
     label_font = gdk_font_load("-*-helvetica-bold-r-normal-*-14-*-*-*-*-*-*-*");
     sprintf(label,"mx");
     gdk_draw_string(g->pixmap, label_font, penBlue,
                s_t+4*x_size/5,
                b_w+gdk_string_height(label_font,label)/2,
                label);

     gdk_draw_line(g->pixmap, penRed,
                b_w+2*x_size/5, 2*b_w,
                b_w+3*x_size/5, 2*b_w);

    sprintf(label,"my");
     gdk_draw_string(g->pixmap, label_font, penRed,
               s_t+4*x_size/5,
               2*b_w+gdk_string_height(label_font,label)/2,
               label);
}
```

```
  update_rect.x=0;
  update_rect.y=0;
  update_rect.width=where_to_draw->allocation.width;
  update_rect.height=where_to_draw->allocation.height;
  gtk_widget_draw(where_to_draw,&update_rect);

}
```

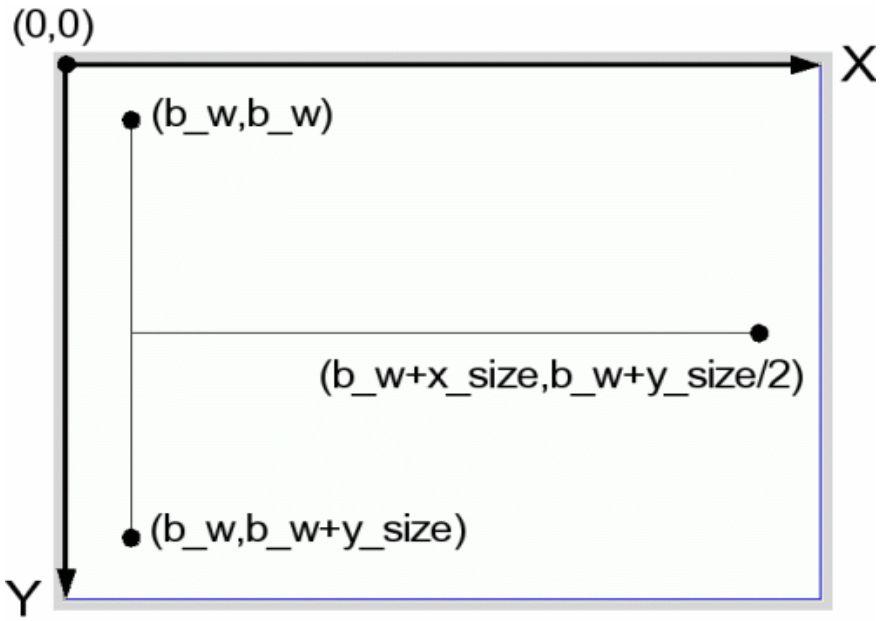
En primer lugar, debemos indicar que vamos dibujando los elementos sobre el mapa de pixels de segundo plano, y sólo al final se vuelca el contenido a la pantalla. Esto es sólo una cuestión de procedimiento, por lo que pasaremos entonces a explicar los aspectos más destacados de la rutina. En primer lugar, tenemos que identificar el widget de dibujo que se llama *drawingarea1* (por eso tuvimos que activar la casilla *Fijar el nombre de los componentes* en las Opciones del Proyecto de Glade). Para este cometido utilizamos la función **lookup_widget** definida por Glade. Definimos las variables apropiadas que nos permitan dibujar fuentes en el Área de Dibujo y seleccionamos la fuente helvetica de 14 pt (la cadena que debemos indicar como argumento, en nuestro caso "-*-helvetica-*-r-normal-*-14-*-*-*-*-*-*-*", puede obtenerse con la utilidad **xfontsel** de Linux)

```
GdkFont *label_font=NULL;
  gchar label[50];

 if( label_font == NULL)
    label_font = gdk_font_load("-*-helvetica-*-r-normal-*-14-*-*-*-*-*-*-*");
```

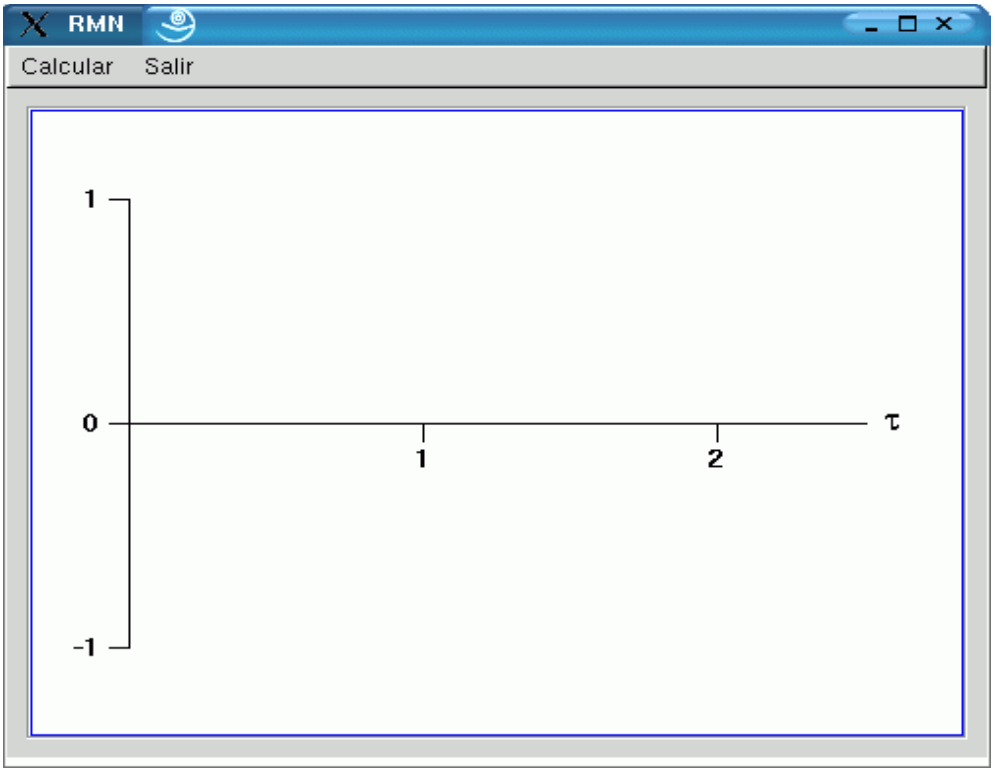
Asimismo, definimos algunos enteros, como el ancho **b_w** de un marco donde no se dibujará nada (una especie de tierra de nadie), la longitud **l_w** de los *ticks* sobre los ejes y la separación **s_t** de los *ticks* y las correspondientes etiquetas.

Independientemente de si se ha ejecutado o no el programa, siempre dibujamos dos elementos, a saber, dos rectángulos. Un primer rectángulo es blanco y relleno (aparece TRUE como uno de sus argumentos) con vértices (0,0) y (widget->allocation.width, widget->allocation.height), es decir, ocupa toda el Área de Dibujo. El segundo elemento es decorativo, y consiste en un rectángulo azul vacío (aparece FALSE en el argumento de la función) en el perímetro del Área de Dibujo.
Las sentencias para dibujar los ejes, los ticks y las etiquetas son bastante claras si consideramos la siguiente imagen aclaratoria donde se indican las coordenadas de algunos puntos significativos sobre el Área de Dibujo

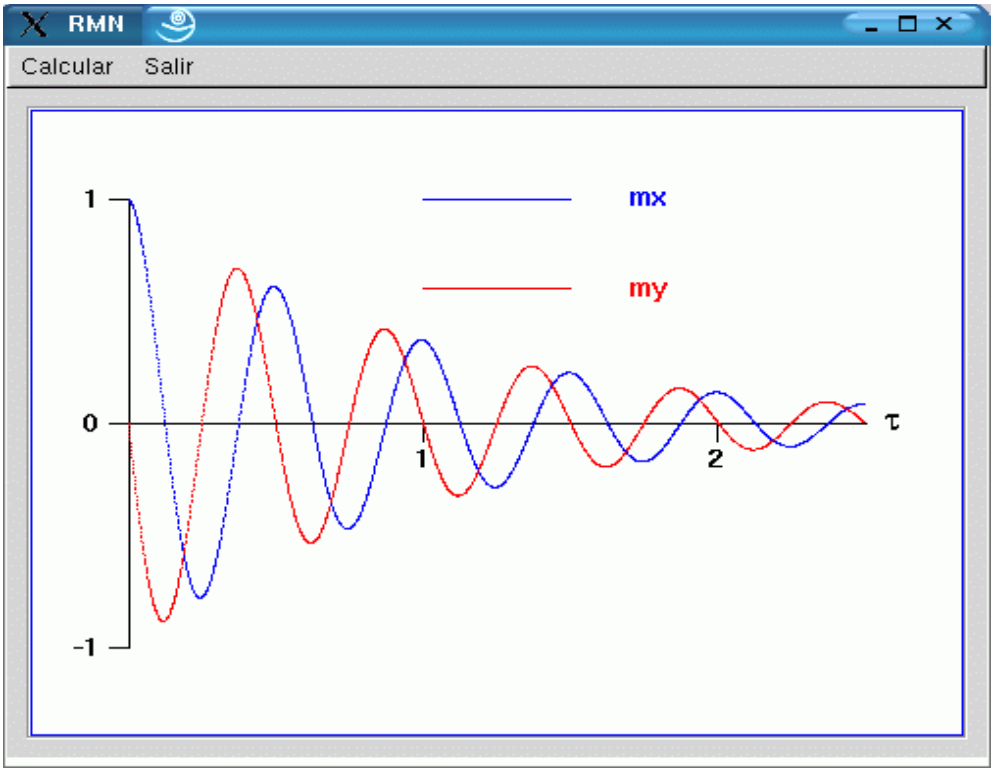


También es facil comprender que la cadena label se inicia con la función **sprintf**, que es totalmente análoga a la función **fprintf** de C. Finalmente, observamos que los puntos de las curvas y las correspondientes leyendas se dibujan una vez que se ha indicado la ejecución del programa y la variable calculado ha tomado el valor TRUE. Por tanto, antes y depués de la ejecución del cálculo las ventanas tienes la siguiente apariencia

Antes de calcular



Después de calcular



Código fuente

Las fuentes de esta aplicación y el correspondiente archivo de Glade se pueden descargar en la siguiente dirección

http://valbuena.fis.ucm.es/~adame/programacion/Glade/RMN/rmn-0.1.tar.gz

... y Windows?

Los usuarios de Windows están de enhorabuena porque tanto GTK como Glade están portados a este entorno. Puede descargar los archivos aquí los corresponientes archivos

http://valbuena.fis.ucm.es/~adame/programacion/Windows/wingtk-0.2.exe

http://valbuena.fis.ucm.es/~adame/programacion/Windows/wGLADE-0.2.exe

o bien hacerlo desde la Web http://wingtk.sourceforge.net. Para que compilen el ejemplo anterior basta reemplazar el archivo **Makefile** para Linux por el correspondiente a Windows que acompaña a **wingtk**, y ejecutar **make** del directorio **bin** creado tras la instalación de **wingtk**.

Bibliografía

E. Harlow, **Desarrollo de aplicaciones Linux con GTK+ y GDK**, Prentice Hall Iberia, 1999.

D. Martin, **SAMS Teach Yourself GTK+ Programming in 21 Days**, Sams Publishing, 2000.

F. Domínguez-Adame, **Física del Estado Sólido: Teoría y Métodos Numéricos**, Editorial Paraninfo, 2000.

Enlaces de interés

Página del autor dedicada a programación en Física del Estado Sólido: http://valbuena.fis.ucm.es/~adame/programacion/

Glade: http://glade.gnome.org

Anjuta: http://anjuta.sourceforge.net

GTK: http://www.gtk.org

Tutorial de GTK+: http://www.linuxlots.com/~barreiro/spanish/gtk/tutorial/gtk_tut.es.html

Guía de Glade: http://tigre.aragon.unam.mx/m3d/glade/glade_manual.html

Starting off in Glade/GTK+: http://www.geocities.com/ichattopadhyaya/linux/glade.htm

GTK+, Glade, LibGlade etc. for Windows: http://wingtk.sourceforge.net/