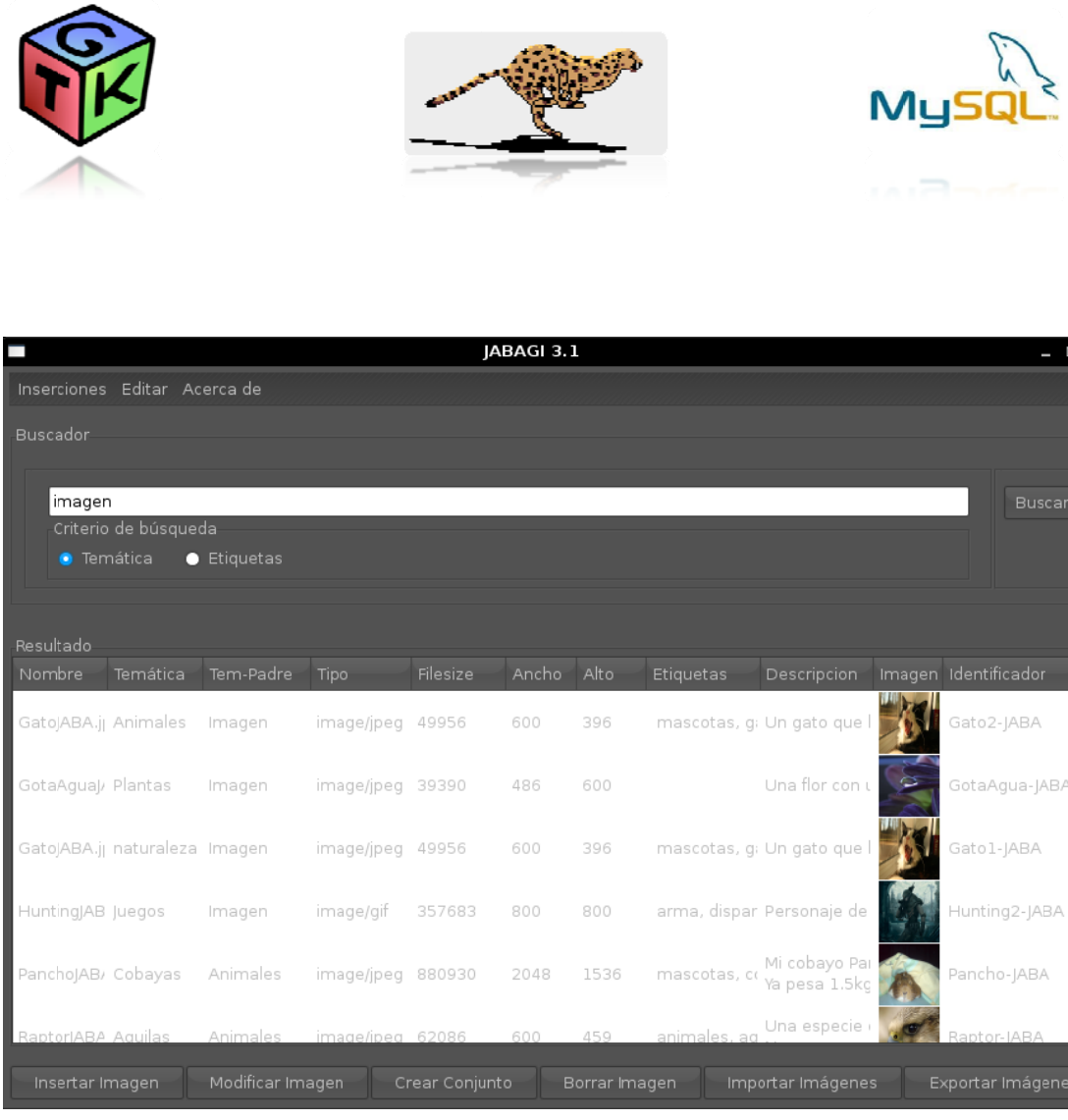
Documentación : Gestor de Imágenes.

(GTK2.0, GDK2Pixbuf, FreePascal & MySQL)

Jose Alberto Benítez Andrades



Metodología y Tecnología de la Programación

2º Ingeniería Informática, DNI: 71454586A

infjab02@estudiantes.unileon.es

Índice

PRÓLOGO

A la hora de realizar aplicaciones, un programador debe tener en cuenta muchos aspectos, tales como los tipos de datos a utilizar y las operaciones para trabajar con ellos, es decir, realizar un diseño "interno" del programa. Todo esto no le interesa al usuario final, que únicamente pedirá como requisito que aquello que le ofrecemos funcione correctamente, y lo haga de la mejor manera posible y además, cuanto más sencillo mucho mejor.

Estas razones hacen necesaria la implementación de una interfaz agradable para el usuario, que le permita realizar las tareas que solicita sin demasiado esfuerzo (tras un simple "clic" puede haber cientos o incluso miles de líneas de código...).

La interfaz de nuestro programa es la carta de presentación del mismo, y como tal debe ser capaz de llamar la atención. Es muy posible que la primera impresión del usuario ante dos programas que realicen la misma tarea, haga que se incline hacia una aplicación con interfaz gráfica frente a otra en modo texto, a pesar de no ser la que funciona mejor.

1. ARQUITECTURA DE LA APLICACIÓN.

La elaboración de una práctica mediana conlleva una buena fase de análisis del problema y diseño del programa, ya que si no seguimos bien las especificaciones del programa, podemos crear una aplicación que no hace lo que el cliente nos pide.

Otro problema muy frecuente, es la mala elección de unos tipos de datos y una mala codificación del programa, ya que eso conllevaría a un programa ineficiencia, ya sea porque posee muchas líneas de código para hacer algo sencillo, o simplemente porque a la hora de la ejecución, el programa funciona muy lentamente y con errores.

En este apartado voy a explicar el diseño de la base de datos que he utilizado para hacer el programa, y también desglosaré el funcionamiento de las unidades y del programa principal.

1.1. DISEÑO DE LA BASE DE DATOS.

Según las especificaciones del programa, debemos crear una base de datos en la que podamos almacenar imágenes que contengan la siguiente información:

- Identificador textual de la imagen (nombre descriptivo),
- descripción textual abierta de la imagen,
- nombre del fichero asociado,
- tipo de fichero,

- tamaño en bytes,
- anchura y altura de la imagen,
- temática principal de la imagen,
- temática padre de la temática principal, (la clasificación por temáticas es jerárquica, considerándose **imagen** como el término que define la raiz de la jerarquía)
- un conjunto de etiquetas libres (texto) asociadas a la imagen.

Para ello he creado 3 tablas, **imagen**, **tematica** y **subtematica**.

1.1.1. Tabla Imagen.

La tabla principal, en la que almacenaremos las imágenes en sí con toda su información, es la tabla que yo he denominado ***imagen***. Contiene los siguientes campos:

- ***Id***: Identificador de Imagen único para cada imagen añadida, es la clave primaria de nuestra tabla imagen.
- ***nombre_imagen***: el nombre del fichero imagen.
- ***descripcion***: descripción de la imagen.
- ***tipo***: se almacenará el mime type de la imagen (image/jpeg,image/png...).
- ***tematica***: se almacena la id de la imagen, con ella podremos obtener el nombre de la temática y más información desde la tabla *Tematica*.
- ***subtematica***: al igual que tematica, almacena una id con la cual podremos obtener el nombre de la subtematica desde la tabla *Subtematica*.
- ***anchura***: anchura de la imagen en píxeles.
- ***altura***: altura de la imagen en píxeles.
- ***tamano:*** tamaño de la imagen en bytes.
- ***etiquetas:*** etiquetas que pertenecen a la imagen.
- ***Identificador:*** es el id único de cada imagen que nos hará falta para la gestión de las imágenes.

1.1.2. Tabla Tematica.

En esta tabla se almacenan todas las temáticas que puede poseer una imagen, y además se almacenará si la temática tiene temáticas hijas, o no. Los campos que posee son los siguientes:

- ***id***: almacena la id única de cada temática, es la clave primaria de la tabla.
- ***nombre***: nombre de la temática.
- ***espadre***: es un booleano que almacena TRUE si la temática tiene temáticas hijas, o FALSE si por el contrario, no posee temáticas hijas.

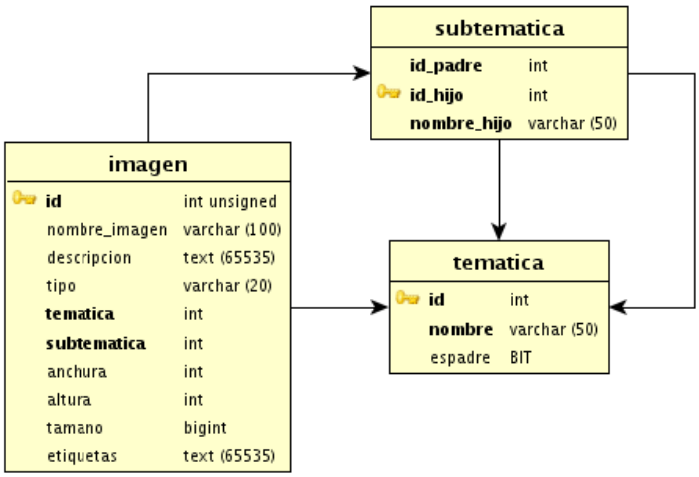
1.1.3. Tabla SubTematica.

En esta tabla se almacenarán las relaciones padre-hijo de las temáticas, tiene 3 campos y son los siguientes:

- ***id_padre***: se almacena la id de la temática padre, está relacionada con el campo *id* de la tabla *tematica*.
- ***id_hijo***: contiene la id de la temática hija, está también relacionada con el campo *id* de la tabla *tematica*.
- ***nombre_hijo***: almacena el nombre de la temática hija.

1.1.4. Diagrama Entidad-Relación.

El Diagrama Entidad-Relación de la base de datos anteriormente explicada sería el siguiente :



1.2. CODIFICACIÓN.

Una vez pensada la distribución de la base de datos, y pensado en cómo abordar el problema, comienza la etapa de codificación, en la cual, pasamos nuestras ideas de cómo hacer funcionar el programa, a código. Después de hacer la práctica, mi programa está formado por 6 unidades y el programa principal:

Unidades:

- errorsInOut.pas
- tad_cola.pas
- miMysql.pas
- tratadoXML.pas
- GestionImagenes.pas
- intGes.pas

Programa principal:

- JABAGI.pas

1.2.1. errorsInOut.pas

Esta unidad es la encargada de gestionar los errores a la hora de crear, abrir o leer ficheros. Contiene un único procedimiento:

```
PROCEDURE calcError(error:integer);
```

Recibe como parámetro un número entero que representa el tipo de error, y dependiendo de ese número muestra un mensaje u otro. Por ejemplo en nuestro tratadoXML.pas al abrir el descriptor.xml tenemos el siguiente código:

```
{$I-}rewrite(descriptor);{$I+}

    errorF := ioresult;

if (errorF<>0) then

        Begin

            calcError(errorF);

        End
```

errorF es una variable de tipo entera, en la cual se almacena el *ioresult*, que se obtiene con la creación de un fichero en este caso. Cuando *errorF* es distinto de 0, significa que hay un error, éste se envía al procedimiento *calcError* y muestra por pantalla un mensaje. Por ejemplo si no tenemos permisos para crear el fichero *errorF* sería 5 y *calcError* ejecutaría en su *case* el siguiente caso:

```
5 : writeln('Acceso denegado');
```

1.2.2. tad_cola.pas

Esta unidad está formada por 4 procedimientos y 2 funciones:

```
PROCEDURE ColaVacia(var C:tCola);
```

- Este procedimiento se encarga de crear una cola vacía.

```
FUNCTION EsVacia(C:tCola):boolean;
```

- Esta función comprueba si la cola está vacía o contiene elementos.

FUNCTION Primero(C:tCola):longint;

- Con esta función obtenemos el primer elemento de la cola.

PROCEDURE Poner(var C:tCola; tematica:longint);

- Con este procedimiento insertamos elementos en la cola.

PROCEDURE Quitar(var C:tCola; var tematica:longint);

- Este procedimiento se utiliza para eliminar elementos de la cola.

PROCEDURE Suprimir(var C:tCola);

- Este procedimiento es con el que borramos realmente el elemento que se le pasa desde el procedimiento anterior *Quitar* y actualizamos la cola.

Del uso de esta unidad en nuestro programa hablaremos en el punto *1.2.6. JABAGI.pas*

1.2.3. miMysql.pas

Esta unidad contiene todas las rutinas que vamos a necesitar relacionadas con la base de datos MySQL (conexión a mysql, creación de la base de datos, búsquedas…).

Procedure LlamadasDB(llamada:Pchar);

- Con este procedimiento produciremos llamadas en la base de datos mediante la rutina *mysql_query* pasándole como parámetro llamada que será un puntero a una cadena que contendrá la llamada.

Procedure ConectaDB(var directorio:string);

- Este procedimiento es el encargado de conectar a la base de datos, y recibe como parámetro por variable, directorio de tipo cadena, que contendrá el nombre del directorio donde se almacenan las imágenes de nuestra base de datos.

Procedure CreaDB();

- Con CreaDB crearemos la base de datos, con las respectivas tablas y relaciones, en el caso de que no existan.

Procedure BusquedaDB(busqueda:pchar;var valor:longint;mostrar:boolean;var tabla:tTablaBusqueda);

- Este procedimiento hace una llamada del parámetro *busqueda* recibe por variable el parámetro *valor* que es de tipo entero largo y en él se almacenarán el número de filas que corresponden a la búsqueda que se haya hecho.

- *Mostrar* es una bandera que determina si queremos que se muestren la búsqueda o no, en nuestro caso, no querremos mostrar búsquedas cuando estén relacionadas con temáticas, y sí las mostraremos, cuando la búsqueda sea de imágenes.
- Por último la *tabla* que recibe como parámetro por variable de tipo *tTablaBusqueda* almacenará todas las filas y columnas de la base de datos para más tarde poder mostrarlas con *GTK* en nuestro *CLIST* en el programa principal.

```
Function Num2St(entero : longint):string;
```

- Esta función simplemente se encarga de pasar un número (*entero*) a cadena de caracteres, nos servirá para añadir números en la base de datos, como tamaño, altura, anchura…

```
Procedure BusquedaTematicas(busqueda:pchar;var colaTems:tCola;
encolar:boolean;var tabla:tTablaBusqueda);
```

- Este procedimiento es similar al anterior, pero sirve únicamente para buscar imágenes por temática, en este procedimiento se hará uso de la cola que crearemos en el programa principal.

1.2.4. GestionImagenes.pas

Gracias a esta unidad podemos obtener el "*mime type*" de las imágenes que inserta el usuario de manera individual, guardar las imágenes en la base de datos y modificar las imágenes que ya estaban en la base de datos. Está formada por 3 procedimientos y una función.

```
Function gdk_pixbuf_get_file_info(filename:Pgchar; width:Pgint;
height:Pgint):PGdkPixbufFormat; cdecl; external gdkpixbuflib;
```

- Esta función se encarga de obtener, mediante el nombre del fichero imagen, el *mime type* de la imagen en sí, por ejemplo *image/png*.

```
Procedure GuardarImagen(var imagen:tImagen);
```

- Este procedimiento recibe un registro que contiene como elementos el nombre de la imagen, el tipo de imagen, el tamaño en bytes de la imagen, la temática, la temática-padre , la descripción y las etiquetas asociadas a la imagen. Es el encargado de: crear el pixbuf de la imagen, con el que obtiene el tamaño del fichero, la anchura y altura de la imagen gracias a una función *SubirImagen* y una vez obtenido esto, si el fichero existía, envía el registro a *AlmacenarDB* que se encargará de agregar la imagen a la base de datos.

Procedure AlmacenarDB(var imagen:tImagen);

- Este procedimiento se encarga de hacer la llamada de inserción de la imagen en la base de datos, los datos de la imagen los tiene en el registro *imagen* que recibe como parámetro.

Procedure ModificarImagen(var imagen:tImagen);

- Este procedimiento hace la llamada a la base de datos cambiando el valor de los datos que ha modificado el usuario mediante la interfaz gráfica, en la base de datos.

1.2.5. intGes.pas

Esta unidad pretendía ser la unidad de interfaz gráfica separada del programa principal, pero por problemas de relación entre código de freepascal con GTK no he podido adecuarla a lo que quería. Posee 4 procedimientos públicos y son los siguientes.

Procedure destroy(widget : pGtkWidget; data : gpointer); cdecl;

- Este procedimiento destruye la ventana principal.

Procedure destroy_wid(widget : pGtkWidget; data : gpointer); cdecl;

- Este procedimiento destruye el widget que recibe como parámetro, sin destruir el widget principal en el que esté contenido, a diferencia de *destroy*.

Procedure file_ok_sel(w : pGtkWidget ; fs : pGtkFileSelection); cdecl;

- Este procedimiento es el encargado de seleccionar ficheros de menús de selección. Además, si el valor de bandera_imagen es falso, recortará del fichero seleccionado el directorio que posee, con lo que mostrará en la ventana de inserción de imagen únicamente el nombre del fichero. Es decir, si el usuario selecciona una imagen que se encuentra en */home/jaba/imágenes/cacahuete.jpg*, en la ventana donde se muestra el nombre del fichero seleccionado, sólo se verá *cacahuete.jpg*.

Procedure MenuEleccion; cdecl;

- Este procedimiento muestra el menú de selección en el caso de pulsar el botón *Examinar* cuando el usuario desea insertar una imagen en la base de datos.

1.2.6. tratadoXML.pas

Con esta unidad analizaremos las importaciones y exportaciones en general, se encarga de leer el *descriptor.xml* en las importaciones y en crearlo en las exportaciones. Posee 3 procedimientos públicos:

Procedure ImportarImagenes(nombre_tar,directorio:string;var Mensaje:string);

- Este procedimiento recibe el nombre del fichero comprimido en .tar.gz (*nombre_tar)* que queremos importar, y una variable k, que mostrará un mensaje positivo cuando la importación se haga correctamente y negativo cuando se produzca un error en la lectura del descriptor.xml.

- Primero se descomprime el fichero mediante una llamada a *"shell"* ejecutando *'tar -xzvf nombre_tar'* y seguidamente haremos un '*cp imagenes/*.* directorio_imagenes'* . El primero descomprime el fichero en el *directorio_imagenes* que tenemos de nuestra base de datos, y el segundo copia el contenido de la carpeta *imagenes/* que se ha descomprimido, en el *directorio_imagenes* de la base de datos.

- Una vez hemos descomprimido las imágenes y el descriptor.xml, se procede a la lectura del mismo, mediante un bucle simple de lectura de ficheros, y dependiendo del valor de nuestra variable i, se analiza en cada línea si el fichero está bien creado o no. En caso de no estar bien creado, no se insertarán las imágenes en la base de datos, y si está bien creado, se insertarán mostrando un mensaje de éxito.

- Cabe destacar la inserción de nueva temáticas en nuestra base de datos. Cuando leemos en el *descriptor.xml* la *temática-padre* comprobamos si está ya en nuestra base de datos, si está en nuestra base de datos, no la añade, seguidamente comprueba si *espadre* está TRUE o FALSE, caso de estar FALSE, lo cambia a TRUE. Una vez comprobada la *temática-padre* comprueba si la *Tematica* está o no en la tabla *temática* caso de no estar en ella, la añade, y además de añadirla a la tabla *Tematica* la inserta en la tabla *Subtematica* poniendo en *id_padre* la id de la *temática-padre*.

Procedure ExportarImagenes(nombre_tar:string;var mensaje:string;t: tTablaBusqueda;conjunto: tTablaBorrado);

- Recibe el nombre del fichero en el que queremos exportar el conjunto de imágenes y el conjunto de imágenes a exportar en *t.*
- Este procedimiento tiene un funcionamiento similar al anterior. Primero crea una carpeta *imágenes/* donde se almacenarán las imágenes exportadas. Una vez

creada esta carpeta, trata cada elemento contenido en *t* de forma individual, añadiendo esas imágenes a la carpeta *imagenes/,* y escribiendo el *descriptor.xml* con los datos que recibe.

- Cuando ya tiene el *descriptor.xml* creado, comprime todo mediante *'tar czvf nombre_fich_comprimido imagenes/ descriptor.xml'* y cuando ya está creado, borra lo que hemos creado mediante '*rm descriptor.xml imagenes/ -rf'.*

Procedure cogerEtiquetas(cadena:string;var eti:tDatos;var num_eti:longint;var spuntuacion:tDatos);

- Este procedimiento se encarga de cortar una cadena que recibe en varias cadenas. Cuando encuentra una coma o un punto, comienza la nueva cadena, además devuelve el número de etiquetas que ha contado.

1.2.6. JABAGI.pas

JABAGI es el programa principal, el nombre surgió de JABA = Jose Alberto Benítez Andrades y GI = Gestor de Imágenes. El programa principal contiene X procedimientos y X funciones, y son:

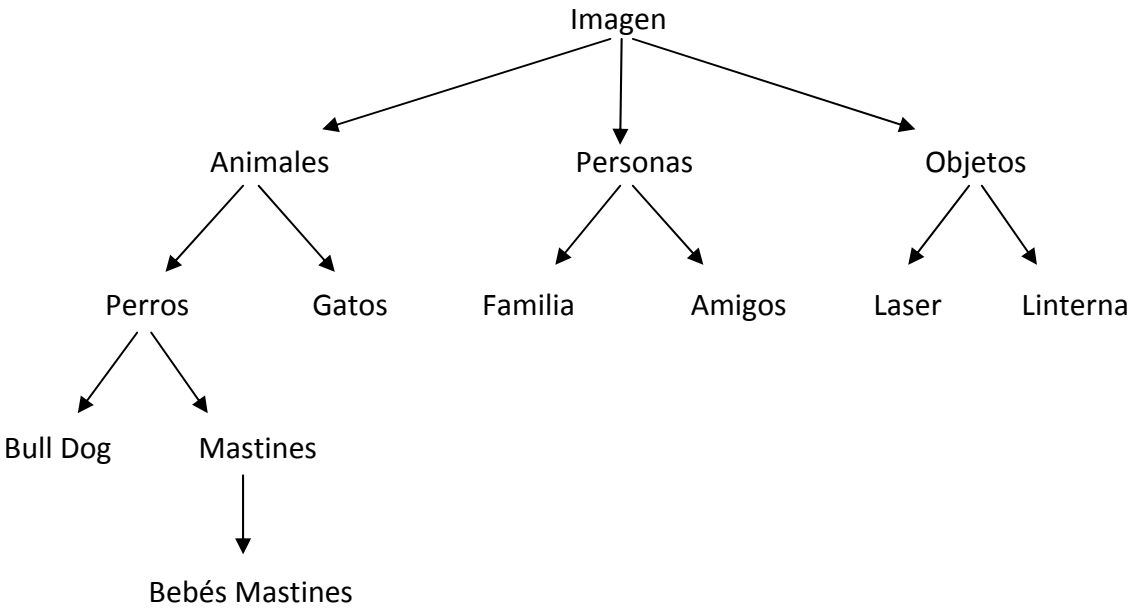
Procedure asignar_criterio(data: gpointer); cdecl;

- Este procedimiento sirve para elegir qué criterio queremos de búsqueda, si el usuario de sea buscar por *temática* o por *etiquetas.* Dependiendo de cual se elija, la búsqueda se hará con un '*WHERE imagen.etiquetas =* ' o '*WHERE imagen.tematica =*'.

Procedure buscar_imagen(data: pGtkCList); cdecl;

- Con este procedimiento buscaremos imágenes por etiqueta o por temática, si es por etiquetas buscará lo siguiente. Por ejemplo, si el usuario introduce *'perro, azul. pistolas. casa, grande'* el buscador buscará todas las imágenes que contengan como etiquetas (perro y azul) O (pistolas) O (casa y grande) , las comas son 'y-es' y los puntos son 'oes'.

- En caso contrario, si buscamos por temáticas: Por ejemplo, supongamos que tenemos el siguiente árbol de temáticas en nuestra base de datos, donde *Imagen* es la raíz del árbol.



Si nosotros queremos mostrar las imágenes que tienen como padre, la temática *Imagen* realmente deberemos mostrar, todas aquellas imágenes que como padre, abuelo, bisabuelo… tienen a Imagen, por ejemplo, *Bebés Mastines* tiene como padre *Mastines*, *Mastines* tiene como padre *Perros*, *Perros* tiene como padre *Animales* y *Animales* tiene como padre *Imagen*, con lo que *Bebés Mastines* pertenece a *Imagen*.

Esto lo conseguimos buscando en la tabla *Subtematica,* la relación padre hijo, de manera que, si Imagen tiene 3 hijos, almacena en la cola esos 3 hijos, seguidamente saca el primer hijo y busca los hijos que este tiene, y los almacena en la cola, y así sucesivamente hasta que acaba con todos.

```
Procedure VentanaInformacion(w : pGtkWidget ; fs : pGtkFileSelection); cdecl;
```

- Este procedimiento lo que hace es, crear una ventana de 400 x 70 en la cual mostrará mensajes. Por ejemplo, cuando importamos un fichero y el descriptor está mal formado, mostrará "*El fichero importado es erróneo*"

```
Procedure visualizar_imagen(tamimg:boolean;nombreimg:string;wid:pGtkWidget);
```

- Procedimiento encargado de mostrar en una ventana, la imagen en solitario. También es el encargado de meter en un recuadro , la imagen con una dimensión de 200 x 200.

Procedure crear_conjunto;

- Este procedimiento crea una ventana nueva centrada en la que muestra la información de la primera imagen seleccionada del CLIST, junto con una imagen en miniatura, la cual se puede agrandar pulsando el botón *agrandar* y con 2 botones que pasan a la imagen anterior y siguiente, de la selección que ha hecho el usuario en el clist.
- Además de mostrar la información, modificando el identificador y la temática, inserta una imagen con los mismos datos que tenía, pero con temática e identificador diferentes.

Procedure aceptar_imagen;

- Para poder insertar una imagen con las mismas características que la que tenemos en nuestro recuadro seleccionada, pero con distinta temática, al pulsar aceptar, se llama a este procedimiento que es el que inserta en la base de datos la imagen con los cambios realizados por el usuario.

Procedure rellenar_valores(imgelegida:longint);

- Para poder mostrar las imágenes anteriores y siguientes necesitamos actualizar todos los datos que tenemos en nuestra ventana de diálogo, esto se consigue mediante este procedimiento, que recoge la información de la siguiente o anterior imagen y lo plasma en pantalla.

Function create_bbox_mod(objeto : longint; horizontal : boolean ; title : pchar ; spacing : gint ; child_w : gint ; child_h : gint ; layout : gint) : pGtkWidget;

- Este procedimiento se encarga de crear los contenedores, entrys y texts en los que mostraremos los valores de los campos de cada imagen.

Procedure agrandar_imagen;

- Al pulsar este botón, se llama al prodecimiento *visualizar_imagen* pasándole un FALSE que determina que, lo que tiene que crear es una ventana nueva con la única imagen.

Procedure siguiente_imagen;

- En este procedimiento se comprueba si la imagen tiene *siguiente imagen* o no, si tiene siguiente imagen, llama a *rellenar_valores* de la nueva imagen.

Procedure anterior_imagen;

- En este procedimiento se comprueba si la imagen tiene *anterior imagen* o no, si tiene anterior imagen, llama a *rellenar_valores* de la nueva imagen.

Procedure modificar_imagen;

- Este procedimiento crea una ventana nueva centrada en la que muestra la información de la primera imagen seleccionada del CLIST, junto con una imagen en miniatura, la cual se puede agrandar pulsando el botón *agrandar* y con 2 botones que pasan a la imagen anterior y siguiente, de la selección que ha hecho el usuario en el clist.
- Además de mostrar la información, deja modificarla, y al pulsar abajo el botón *modificar* la modifica en la base de datos.

Procedure actualizar_imagen;

- Para poder modificar las imágenes, cuando pulsamos el botón modificar, se llama a este procedimiento que es el que actualiza en la base de datos los cambios que ha realizado el usuario.

Procedure rellenar_valores(imgelegida:longint);

- Para poder mostrar las imágenes anteriores y siguientes necesitamos actualizar todos los datos que tenemos en nuestra ventana de diálogo, esto se consigue mediante este procedimiento, que recoge la información de la siguiente o anterior imagen y lo plasma en pantalla.

Function create_bbox_mod(objeto : longint; horizontal : boolean ; title : pchar ; spacing : gint ; child_w : gint ; child_h : gint ; layout : gint) : pGtkWidget;

- Este procedimiento se encarga de crear los contenedores, entrys y texts en los que mostraremos los valores de los campos de cada imagen.

Procedure agrandar_imagen;

- Al pulsar este botón, se llama al prodecimiento *visualizar_imagen* pasándole un FALSE que determina que, lo que tiene que crear es una ventana nueva con la única imagen.

Procedure siguiente_imagen;

- En este procedimiento se comprueba si la imagen tiene *siguiente imagen* o no, si tiene siguiente imagen, llama a *rellenar_valores* de la nueva imagen.

Procedure anterior_imagen;

- En este procedimiento se comprueba si la imagen tiene *anterior imagen* o no, si tiene anterior imagen, llama a *rellenar_valores* de la nueva imagen.

Procedure borrar_imagen (data : pGtkClist); cdecl;

- Con este procedimiento borramos las imágenes que ha seleccionado el usuario en el clist. Se borran de la base de datos, y se borra el fichero imagen que está en el directorio_imágenes.

Procedure selection_made(thelist : pGtkCList ; row, column : gint ; event : pGdkEventButton ; data : pchar); cdecl;

- Este procedimiento se encarga de rellenar nuestro *array de booleanos* de forma que, cuando el usuario selecciona una fila del CLIST, en el *array de booleanos*, en la posición de la fila seleccionada, se pone a TRUE, en caso de deseleccionarla, se pone a FALSE.

Procedure insertar_imagen;

- Este procedimiento abre una ventana de diálogo nueva en la cual se pide introducir el nombre del fichero imagen, una temática, una descripción y etiquetas.
- Después de tener todo seleccionado y escrito, al pulsar *Aceptar* , el programa comprueba que el usuario ha insertado bien todo , o si hay algún tipo de error en la temática elegida o en el nombre del fichero.

Procedure aceptar_imagen;

- Este procedimiento, al igual que en *modificar_imagen* es el encargado de enviar todos los datos a la base de datos para insertar la imagen en ella.

Procedure rellenar_valores(imgelegida:longint);

- Se utiliza para mostrar los valores de : tamaño, tipo, anchura y altura de la imagen, que se obtienen automáticamente de la imagen seleccionada.

Function create_bbox(objeto : longint; horizontal : boolean ; title : pchar ; spacing : gint ;child_w : gint ; child_h : gint ; layout : gint) : pGtkWidget;

- Este procedimiento crea la ventana de diálogo, con los frames necesarios, las virtual box necesarias y los botones.

Procedure Importar_imagenes;

- Con este procedimiento se abre una ventana de menú de selección en la cual el usuario elegirá el fichero que quiere importar.

Procedure Exportar_imagenes;

- Al igual que con *importar_imagenes* se abre una ventana de menú de selección en la cual el usuario elige el nombre del fichero en el que desea exportar su conjunto de imágenes.

Procedure AbrirImagen; cdecl;

- Este procedimiento abre un menú de selección para que el usuario pueda insertar una imagen nueva a la base de datos. Una vez seleccionada, abre el menú de *insertar_imagen.*

Function crear_botones_buscador(tipoboton:longint; horizontal : boolean ; title : pchar ; spacing : gint ;

- Este procedimiento se encarga de crear los frames necesarios y vertical boxes necesarios para poner la *entrada de búsqueda*, el *botón de búsqueda* y los *radio buttons* para elegir si buscamos por *temática* o por *etiquetas*.

1.3.DICCIONARIO DE DATOS.

- Además de explicar los procedimientos y funciones utilizadas para hacer funcionar nuestro programa *Gestor de Imágenes* expondré los tipos de datos utilizados en cada unidad.

1.3.1. Unidad tad_cola.pas

```
UNIT tad_cola;
INTERFACE
TYPE
        tTematica = longint;                { Id de la temática que almacenamos en la cola }

        PNodoCola=^TnodoCola;   { Puntero a TnodoCola }
        TCola=record
                cab, fin:PnodoCola;         { Registro que almacena el primer elemento y el }
                                            { último de la cola. }
                end;
        TnodoCola=record
                Info:longint;               { Temática almacenada en la cola }
                Sig:PNodoCola;   { Puntero al siguiente elemento de la cola }
                End;
```

1.3.2. Unidad miMysql.pas

```
TYPE
{ Tabla que almacenará la info de las imágenes cuando hacemos un busquedaDB }
{ ID,NOMBRE, TEMATICA, SUBTEMATICA.... etc }
    tTablaBusqueda = array [0..N,1..11] of string;
{ Tabla con la que sabremos qué imágenes seleccionan desde la tabla de búsqueda CLIST }
{ La usaremos para borrar imágenes a bloque y crear conjuntos... }
 tTablaBorrado = array [0..N] of boolean;
```

1.3.3. Unidad GestionImagenes.pas

```
UNIT GestionImagenes;
{$H+}
INTERFACE
USES miMysql,gtk2,glib2,Gdk2PixBuf,sysutils,DOS;

CONST
INSIMG2 = 'INSERT INTO imagen (descripcion,image,id_subtematica,anchura,altura,tamano) VALUES (';

TYPE
tImagen = Record          { Registro en el que almacenaremos la información completa de la imagen }
                id_imagen       : integer;      { Almacena la ID de la imagen }
                id_desc         : string;       { Almacena el Identificador Textual }
                titulo          : string;       { Campo que almacenará la id de la imagen}
```

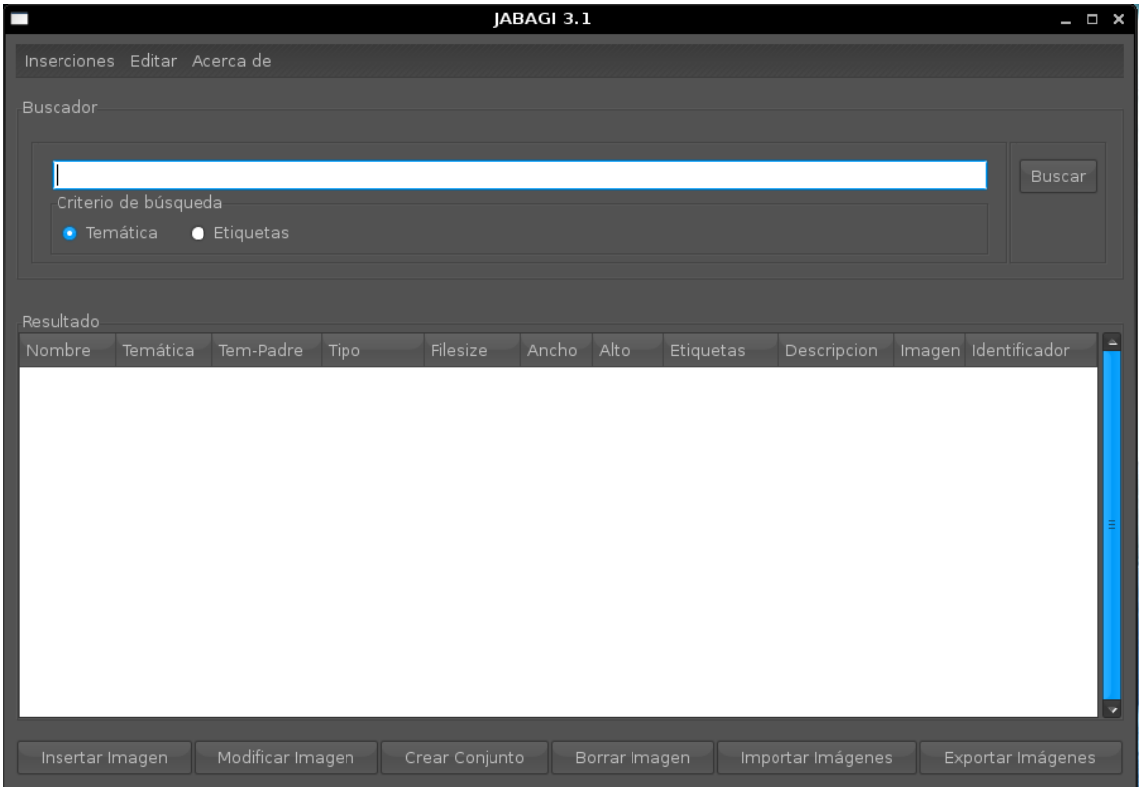
```
                descripcion     : string;       { Almacena la DESCRIPCION de la imagen }
                fichero         : string;       { Almacena el NOMBRE DEL FICHERO imagen }
                tipo_fichero    : pchar;        { Almacena la MIME TYPE de la imagen }
                tamano_img      : longint; { Almacena la TAMAÑO EN BYTES del fichero imagen }
                ancho_img       : longint;      { Almacena la ANCHURA de la imagen }
                alto_img        : longint;      { Almacena la ALTURA de la imagen }
                etiquetas       : string;       { Almacena las ETIQUETAS de la imagen }
                tematica        : string;       { Almacena la SUBTEMATICA de la imagen }
                subtematica     : string;       { Almacena la SUBTEMATICA de la imagen }
                idsubtematica   : longint;      { Almacena la SUBTEMATICA de la imagen }
                idtematica      : longint;      { Almacena la TEMATICA de la imagen }
                existefich      : boolean;      { Almacena la EXISTENCIA DEL FICHERO de la
        imagen }
                malformato      : boolean;      { Nos dice si la imagen posee un FORMATO valido
para gdkpixbuf }
                End;

    PGdkPixbufModulePattern = ^TGdkPixbufModulePattern;
    TGdkPixbufModulePattern = record    { Los tipos declarados abajo hasta el final, nos sirven para
obtener el mime type en nuestra funcion }
        prefix          : ^byte;
        mask            : ^byte;
        relevance       : longint;
    end;

  PGdkPixbufFormat  = ^TGdkPixbufFormat;
  TGdkPixbufFormat = record
        name            : ^gchar;
        signature       : PGdkPixbufModulePattern;
        domain          : ^gchar;
        description     : ^gchar;
        mime_types      : ^Pgchar;
        extensions      : ^Pgchar;
        flags           : guint32;
        disabled        : gboolean;
        license   : ^gchar;
    end;
```

2.FUNCIONAMIENTO DEL PROGRAMA.

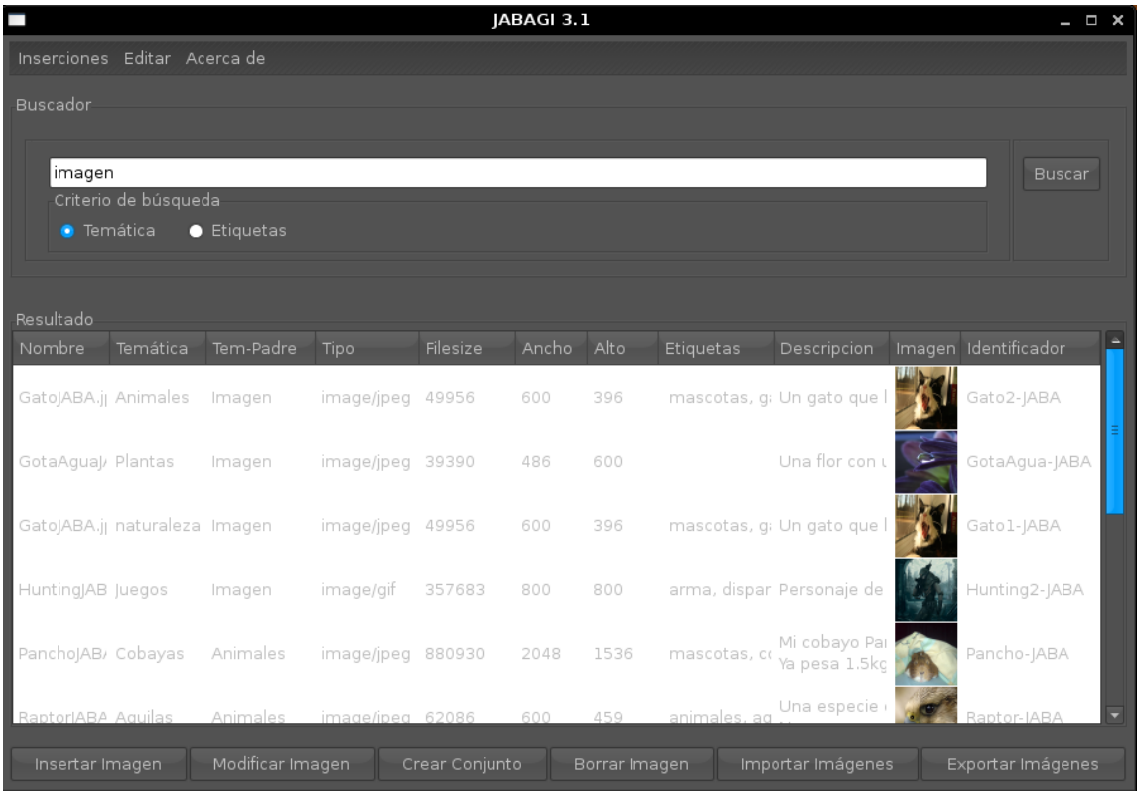
- La ventana principal del programa contiene un menú superior, con menús desplegables, una entrada de texto para buscar las imágenes, botones para elegir el tipo de búsqueda (por etiquetas o por temática), el clist donde se mostrarán todas las imágenes y 5 botones inferiores que realizan la misma tarea que los menús superiores.



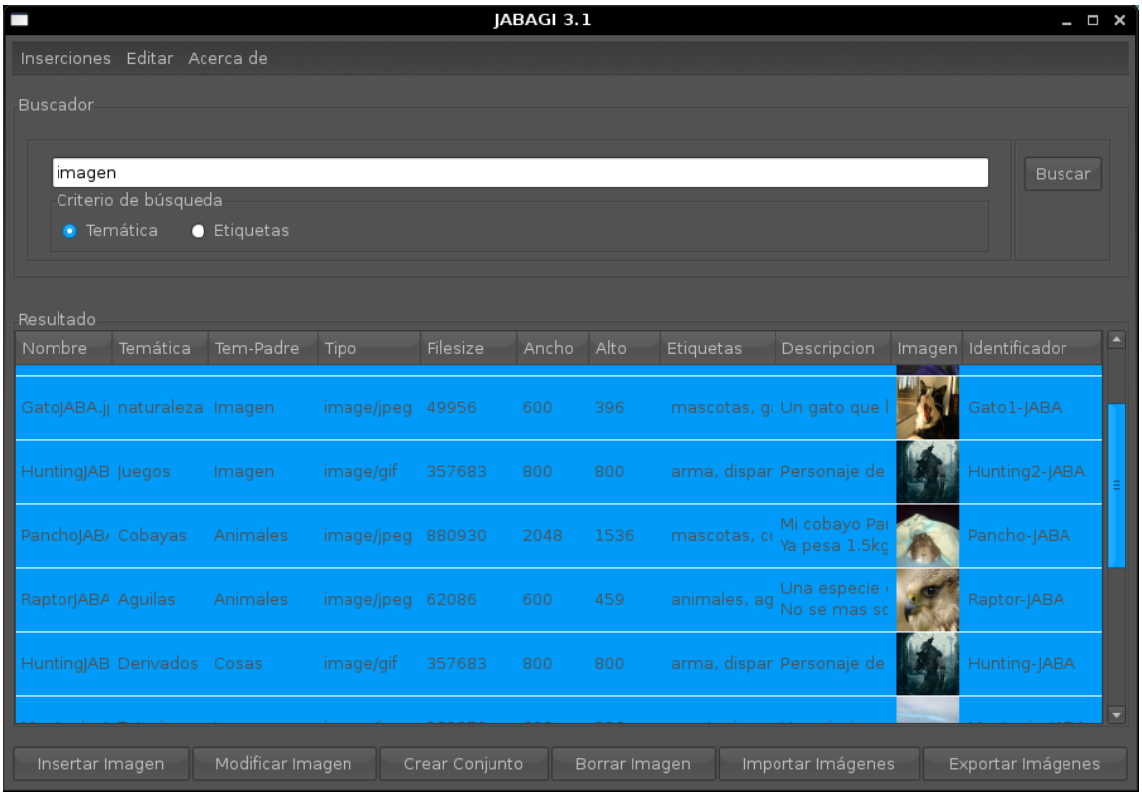
- Para realizar búsquedas, el usuario puede seleccionar imágenes mediante la temática, o las etiquetas que posee. Por ejemplo, si el usuario quiere mostrar las imágenes que contienen como etiquetas (perros y azules) o (palos de madera) o (flores y verdes) el usuario deberá escribir la siguiente sintaxis:

perros, azules. palos de madera. flores, verdes

- Si por el contrario, el usuario desea buscar por *temática*, simplemente deberá escribir el nombre de la temática en el buscador. Por ejemplo, en la siguiente imagen podemos ver el resultado de una búsqueda por temática, y en concreto, que posean como temática *Imagen.*

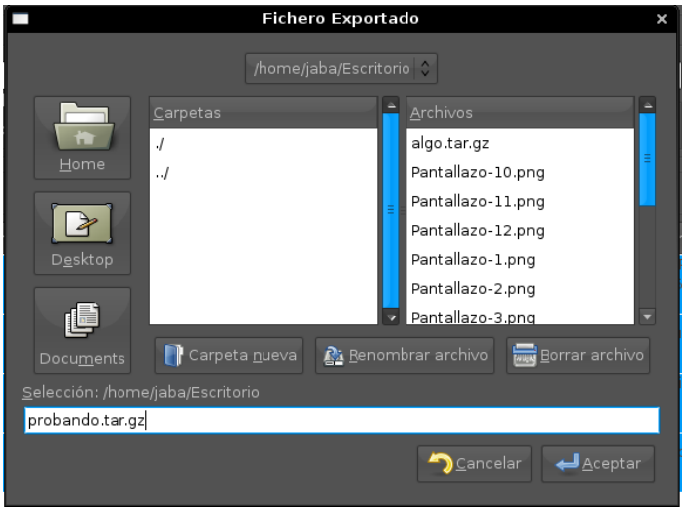


- Una vez realizada la búsqueda podemos seleccionar las imágenes que deseemos. Si queremos hacer una selección múltiple, debemos pulsar *Ctrl + Botón Izquierdo del Ratón*.



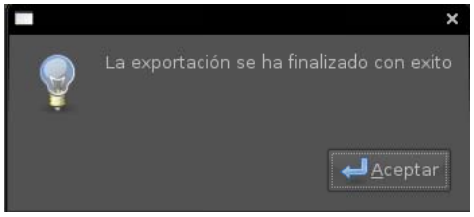
- Con las imágenes ya seleccionadas, tenemos 3 opciones.

- Borrar las imágenes, simplemente con pulsar en *Borrar Imagen* borrará las imágenes seleccionadas.
- También podemos exportar las imágenes, pulsando *Exportar Imágenes*. Se nos abrirá una ventana en la que elegiremos el nombre del fichero comprimido.

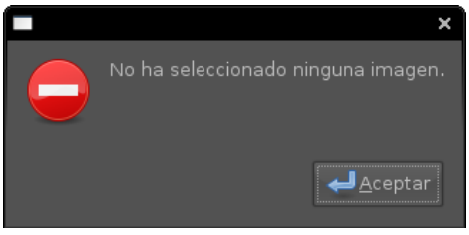


Si el usuario inserta el nombre del fichero exportado sin la extensión .tar.gz, el programa la escribirá de manera automática.

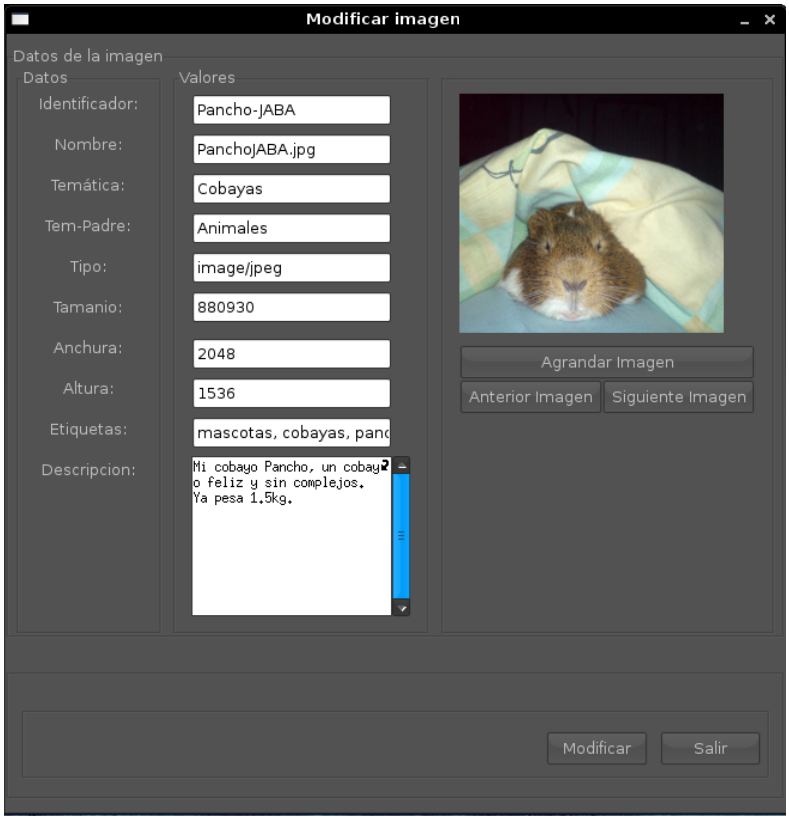
Si la exportación de produce satisfactoriamente se mostrará en pantalla el siguiente mensaje :



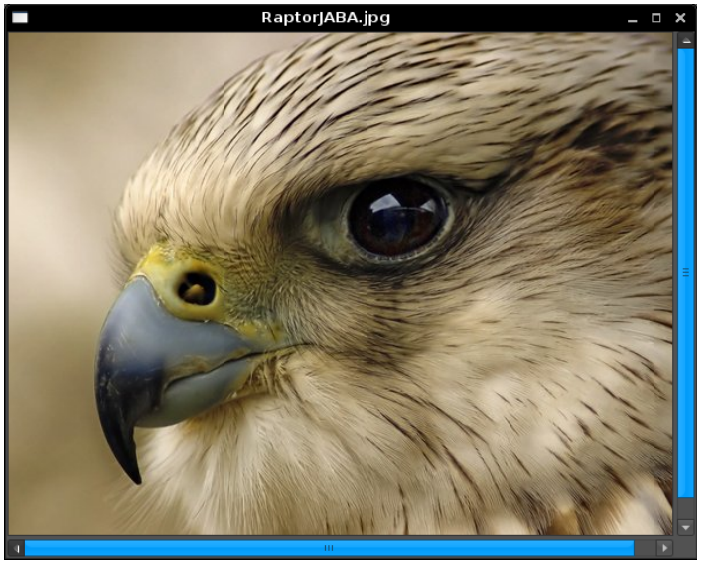
De lo contrario se mostrará el siguiente mensaje en pantalla:



- La 3ª y última opción es la de modificar/visualizar imágenes, pulsando el botón *Modificar imagen*. En este caso se nos abrirá una nueva ventana en la que podremos ver los datos de la primera imagen seleccionada, junto con la imagen en miniatura (200x200px), la cual podemos agrandar pulsando *Agrandar Imagen*. Para pasar a las siguientes imágenes y anteriores, deberemos clicar en *Siguiente Imagen* y en *Anterior Imagen.*

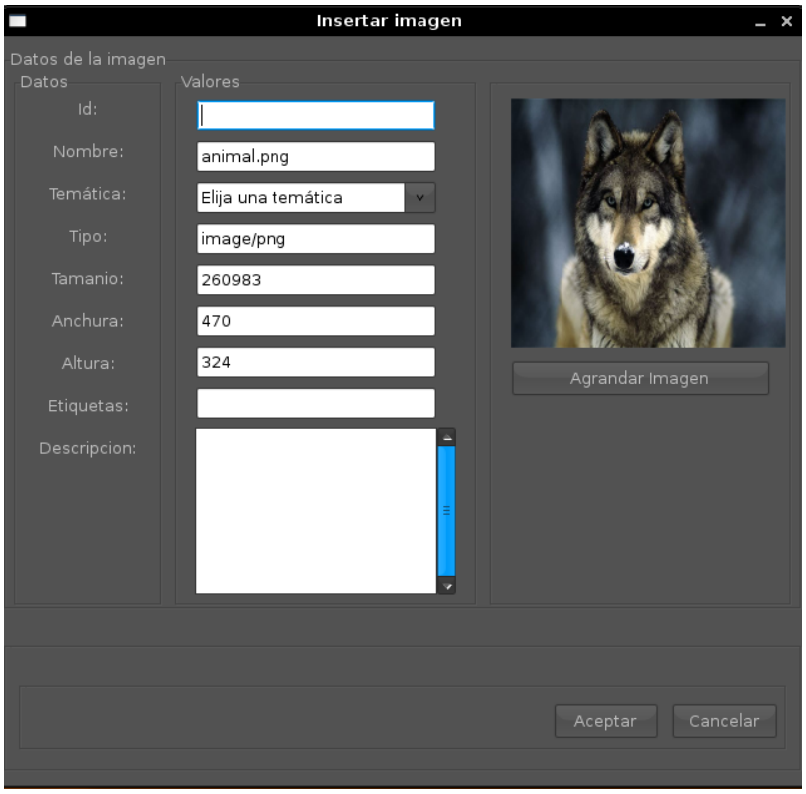


Si pulsamos el botón *Agrandar Imagen* se abrirá una ventana con la dimensión de la imagen de la siguiente manera:

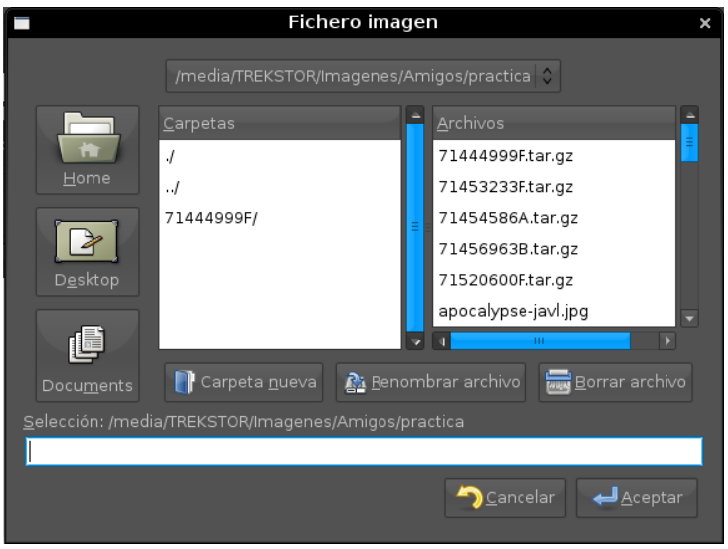


- Otras opción que tenemos en nuestro programa, es la de insertar imágenes de manera individual, esto se consigue pulsando el botón *Insertar imagen.* Esto abrirá una nueva ventana con los datos asociados a la imagen que debe introducir

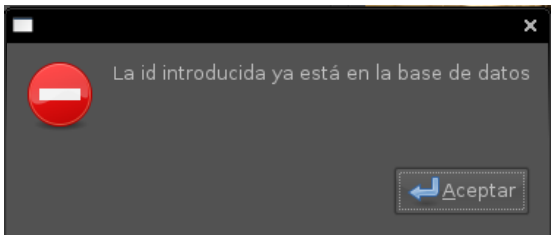
el usuario. Pulsando el botón examinar podrá seleccionar mediante un menú de selección la imagen que desee añadir a la base de datos.

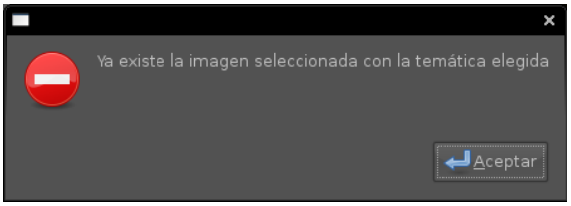


El menú de selección que se nos abre al pulsar examinar, es el siguiente:



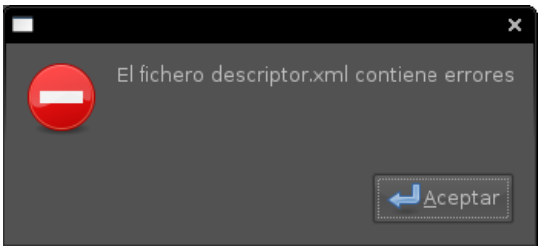
- Si el usuario introduce mal la id o introduce mal la temática, el programa mostrará una ventana con el mensaje de error correspondiente de la siguiente manera.





- Otra opción que tenemos es la de importar imágenes, al pulsar *Importar imagenes* se nos abrirá un menú de selección en el cual deberemos seleccionar el fichero que deseamos importar.

Si el descriptor.xml que posee nuestro fichero a importar, tiene errores, se mostrará una ventana con el siguiente mensaje de error:



De lo contrario se mostrará el siguiente mensaje :