Creación de objetos y modelado en Object Pascal y Model Maker

En el presente ejemplo fue creando las clases y las interacciones entre ellas (Diag. de secuencia) poco a poco. Primero se colocaron algunas clases que se consideraron esenciales en la jerarquía de acceso a los objetos como son:

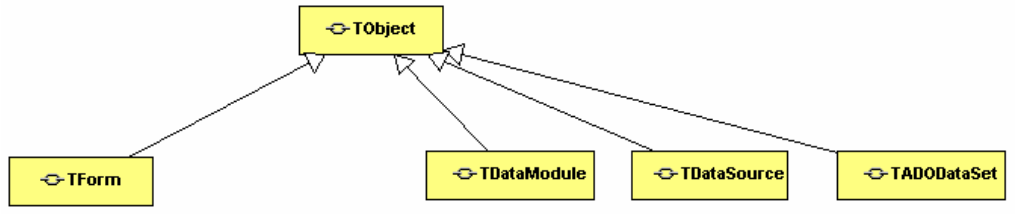

Fig. 1, Clases base

Explicación:

- TForm es la clase base para aquellos objetos que formaran nuestra GUI.
- TDataModule es la clase base para nuestro modulo de datos
- TDataSource y TADODataSet son dos clases necesarias para el ligado y retracción de los datos. Se puede agregar opcionalmente TADOConnection el cual generalmente es empleado en nuestro método de conexión.

Ahora será necesario que pensemos en que objetos pueden existir en nuestro modelo de identificación del usuario. Permítase la definición de las clases siguientes:

- TGUI_ID, será la clase interfaz encargado de solicitar los datos de autentificación. Su papel será precisamente recibir el login y el password
- ControldeAcceso, será la clase encargada de validar que los datos ingresados desde nuestra interfaz de usuario sean válidos. También deberá permitir el acceso a la aplicación si los datos son válidos
- Persistor, será la clase encargada de conocer todo lo relacionado con el acceso a la base de datos. En términos generales la clase persistor será nuestro *DataModule*.
- TGUI_main, será la clase que represente a la aplicación una vez que nuestros datos hayan sido validados.

Establecido lo anterior crearemos nuestros objetos en nuestro diagrama de clases teniendo el resultado siguiente:

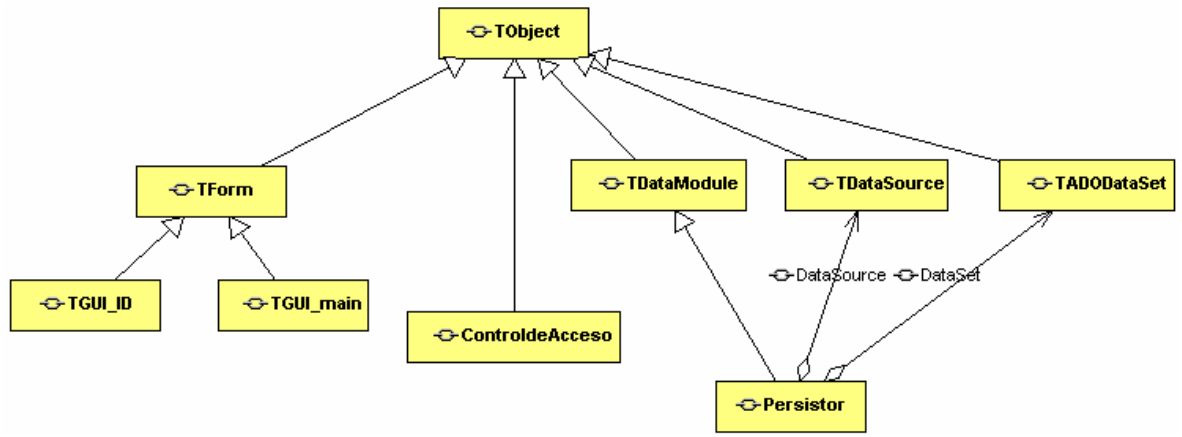

Fig. 2, Jerarquía completa de objetos para el ejemplo

Explicación:

- Como se puede observar las dos clases cuyos nombres inician con TGUI heredan de la clase TForm ya que son objetos de interfaz de usuario.
- La clase ControldeAcceso hereda directamente de la clase base TObject ya que es una clase general.
- Finalmente la clase Persistor hereda de TDataModule, observamos que también existe una agregación de TDataSource y TADODataSet ya que nuestro modulo de datos contendrá a otros objetos de dichas clases para recuperar y almacenar datos.

El siguiente paso para establecer los métodos lo encontraremos al definir nuestro diagrama de secuencia. La figura que se muestra a continuación es el resultado de un proceso iterativo a través del cual se fueron estableciendo los mensajes de acuerdo al papel que tiene cada clase del ejemplo:

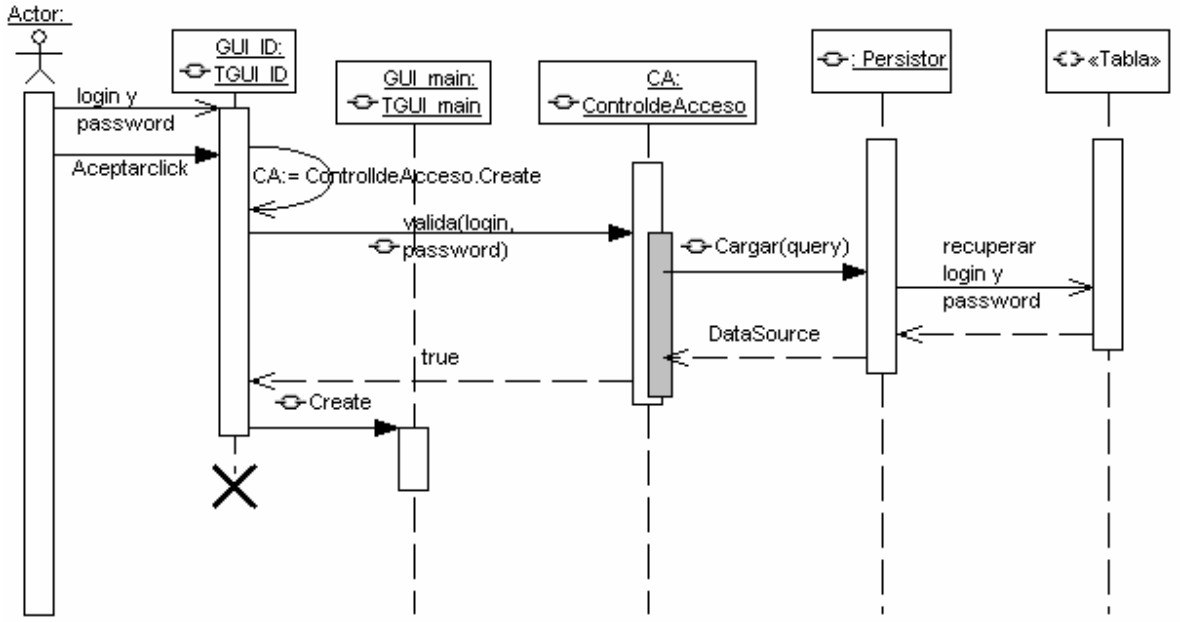

Fig. 3, Diagrama de secuencia Identifica usuario

Explicación:

1. El usuario ingresará su login y password en la interfaz
2. El usuario da un clic en el botón aceptar
   2.1. El evento crea una instancia del objeto ControldeAcceso llamada CA
   2.2. a través de la instancia se tiene acceso a **valida**, método con el cual se verificará si los datos ingresados son correctos
3. La clase ControldeAcceso es la encargada de verificar que los datos ingresados sean válidos, para ello:
   3.1. Empleará los métodos contenidos en la clase persistor, para este ejemplo el único método empleado es **cargar**. Éste método recibirá el query apropiado para recuperar los datos apropiado, por ejemplo aquí sería de la tabla *usuarios*.
   3.2. La clase Persitor solicita la ejecución del query y recibe los datos indicados
   3.3. La clase Persitor regresará un data source, para este ejemplo, en él el resultado de la consulta
4. Nuevamente la clase ControldeAcceso procesa el resultado de la consulta y válida si los datos son correctos
   4.1. Si los datos son correctos entonces invoca la creación de una nueva ventana GUI_main para la aplicación principal

Notas:

- El rectángulo en gris en la figura 3, solo es una de las formas básicas del Model Maker y se uso para representar que dentro del método valida está ocurriendo una serie de acciones
- Faltan también dos mensajes importantes, uno es la creación de la instancia para la clase Persitor y otro es el mensaje de destrucción para la interfaz GUI_ID. *Se deja al estudiante la modificación de este diagrama en Model Maker ya que el mismo se adjuntó a este documento*.
- Los nombre GUI_ID y GUI_main son los nombres de las instancias para las respectivas clases. Estos nombres de instancias son los que se asignaran a cada forma respectiva. En la siguiente sección se explicará como asociar el código con la forma.

Implementación en Model Maker

El siguiente paso será iniciar la codificación de los métodos hasta ahora descritos. Para ello iniciaremos creando las unidades asociadas a las clases definidas en el objeto. Este paso puede ser hecho al final de la codificación, pero se comenzó por allí, ver figura 4.

En la figura 4 se puede observar en el área de las unidades que las únicas clases para las cuales se ha creado su respectiva unidad es para aquellas que son definidas por el diseñador. Aquellas clases que forman la base (ver figura 1) no es necesario generar su unidad, sin embargo la razón por la cual se crearon es para establecer la correcta jerarquía de objetos y que de esta forma hereden los métodos apropiados.

En la imagen también se puede apreciar que para la clase Persitor se han incluido las referencias a DB y ADODB ya que es de allí en donde se encuentran definidos los objetos TDataSource y TADODataSet respectivamente. Un proceso similar se realizará en el resto de la clases en donde sea necesario que se tenga acceso a una determinada definición de los objetos.

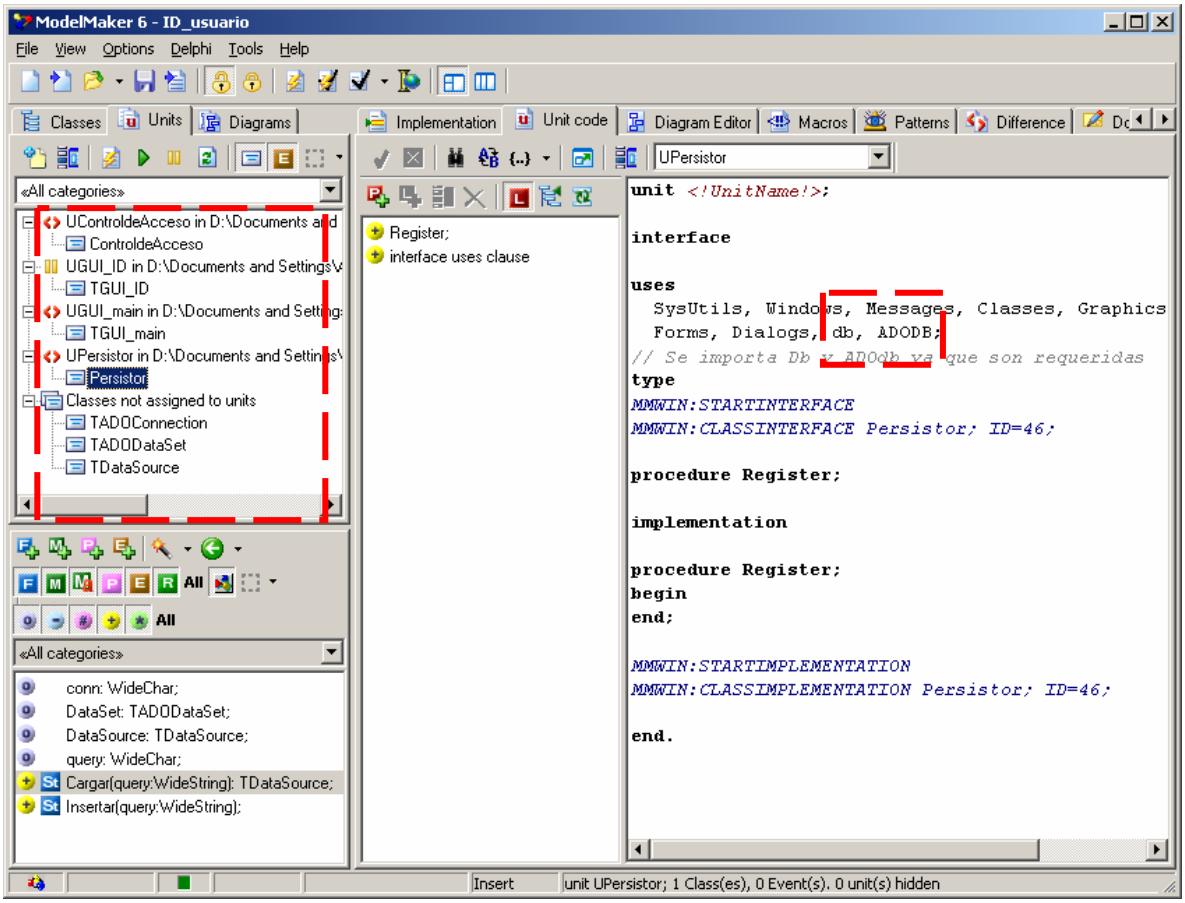

Fig. 4, Clases asociadas a su respectiva unidad

Una vez que se han generado las unidades asociadas empezaremos por agregar el código necesario para las mismas.

Como ejemplo se muestra el código insertado para la clase TGUI_ID, quien fungirá como unidad asociada a la forma inicial de autentificación. Ver Figura 5. En el cuadro discontinuo aparecen las porciones de código nuevo.

Explicación:

1. Primero se declara el nombre de la instancia GUI_ID, que es el nombre de la futura ventana y su tipo será TGUI_ID que es un objeto que deriva la la clase base TFORM:

```
var
  GUI_ID: TGUI_ID;
```

2. Segundo importamos la referencia a la clase de control de acceso

```
uses UControldeAcceso
```

3. Finalmente pero **muy importante** ponemos la directiva que ayudará al compilador a asociar la unidad con un objeto gráfico del tipo ventana.

```
{$R *.dfm}
```

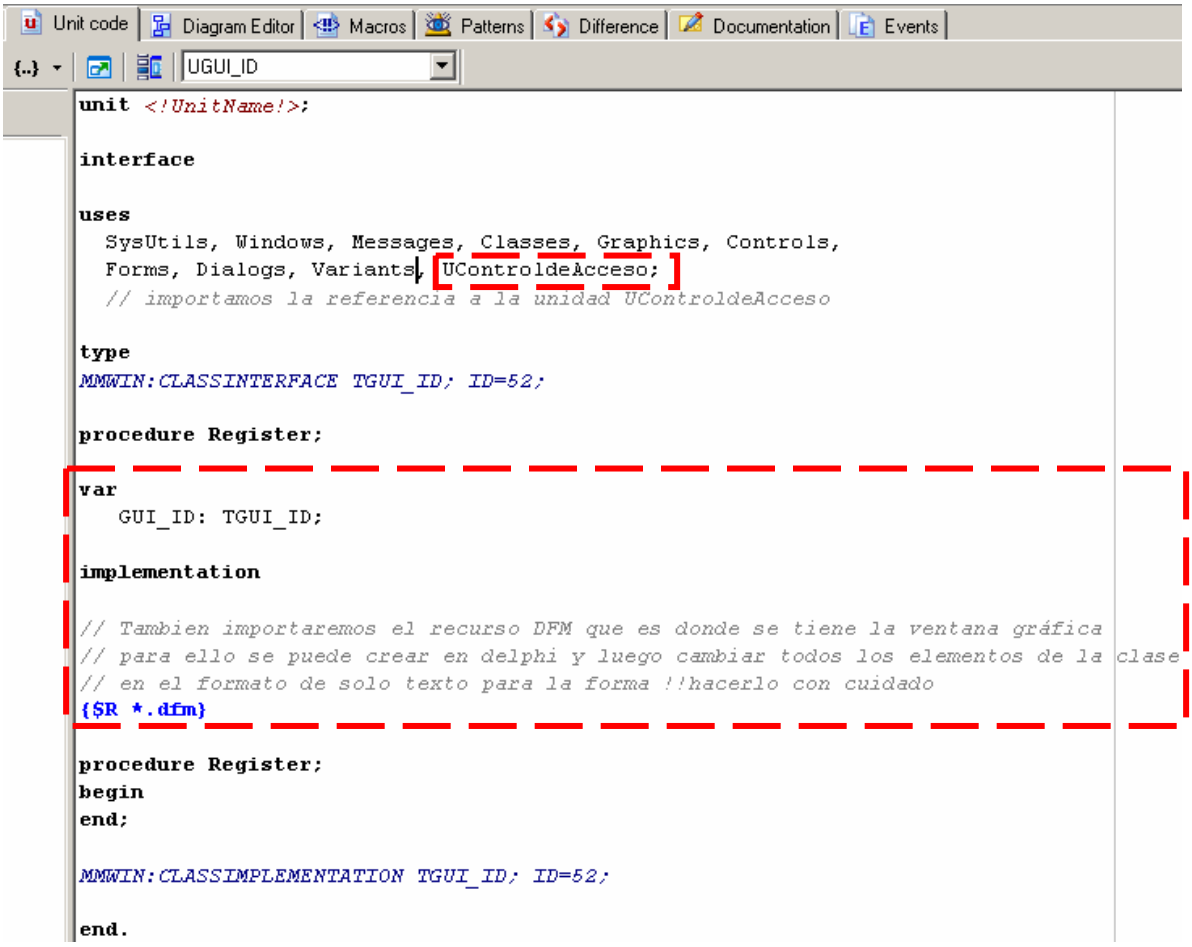

Fig. 5, Listado completo para la unidad UGUI_ID

El resto de los elementos visuales de la ventana no serán declarados desde Model Maker, sino que en su lugar serán hechos directamente desde el editor de Delphi. Luego si se requiere se puede resincronizar el código con el Model Maker e incluir la declaración de todos los componentes gráficos.

Asociar la Unidad Generada a la forma creada en Delphi

Ahora se desarrolla el último paso con el cual se podrá utilizar la unidad generada en Model Maker con una forma en Delphi

1. Primero en Delphi cree un nuevo proyecto, para nuestro ejemplo éste se llamará ID_usuario.dpr
2. El nuevo proyecto incluye una unidad llamada Unit1.pas asociada a una forma por omisión (Form1) y una clase por omisión (TForm1). En realidad en este punto lo que nos interesa es el código que tiene la forma (el objeto en si) y no la unidad, en la siguiente figura se observan los elementos descritos anteriormente:

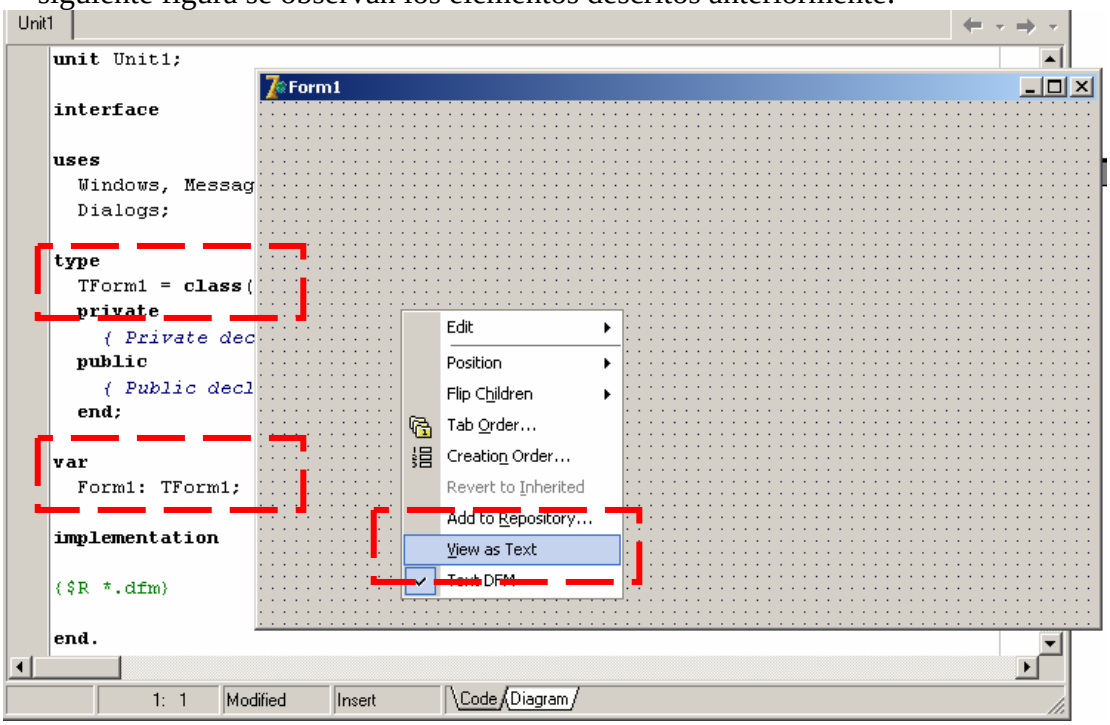

Fig. 6, Unidad y Forma creada por default

3. Modifiquemos pues el código de la forma directamente, para ello sobre la forma de clic derecho y seleccione la opción *view as Text*
4. El código mostrado se observa en la columna izquierda, su correspondiente modificación se observa a la derecha:

Código Original del objeto	Código Modificado
object Form1: TForm1 Left = 515 Top = 190 Width = 574 Height = 380 Caption = 'Form1' Color = clBtnFace Font.Charset = DEFAULT_CHARSET Font.Color = clWindowText Font.Height = -11 Font.Name = 'MS Sans Serif' Font.Style = [] OldCreateOrder = False PixelsPerInch = 96 TextHeight = 13 end	object GUI_ID: TGUI_ID Left = 515 Top = 190 Width = 574 Height = 380 Caption = 'Acceso Unico' Color = clBtnFace Font.Charset = DEFAULT_CHARSET Font.Color = clWindowText Font.Height = -11 Font.Name = 'MS Sans Serif' Font.Style = [] OldCreateOrder = False PixelsPerInch = 96 TextHeight = 13 En

4.1. **Realice las modificaciones** mostradas en el paso anterior

4.2. **Importe** la unidad generada en Model Maker mediante la opción *Add to Project*, en el menú Project. Al hacerlo Delphi mandará un mensaje de error de que no puede localizar la forma asociada, ignórelo y continué
4.3. **Guarde** la forma con sus nombres por default
4.4. **Renombre** solo el archivo .dfm con el nombre de la unidad, en este caso UGUI_ID
4.5. Abra la unidad del proyecto llamada por default Project1 y realice las modificaciones como a continuación se observan:

Código Original del objeto	Código Modificado

program Project1;
 	
uses
  Forms,
  Unit1 in 'D:\Unit1.pas' {Form1},
  UGUI_ID in 'D:\UGUI_ID.PAS';
 	
{$R *.res}
 	
begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
	program Project1;
	
	uses
	  Forms,
	  Unit1 in 'D:\Unit1.pas',
	  UGUI_ID in 'D:\UGUI_ID.PAS' {GUI_ID};
	
	{$R *.res}
	
	begin
	  Application.Initialize;
	  Application.CreateForm(TGUI_ID, GUI_ID);
	  Application.Run;
	end.

4.6. Remueva la unidad Unit1, no se preocupe si al hacerlo se cierra la forma creada. Guarde el proyecto, y vuélvalo a abrir ahora ya tendrá asociada la unidad a la forma.

Con los pasos anteriores ahora ya puede editar el resto de los componentes visuales que desee que la ventana tenga, para el presente ejemplo el diseño de la interfaz quedará como se observa en la figura 7.

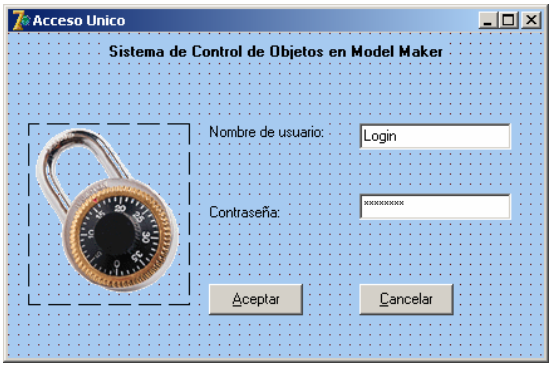

Figura 7, Interfaz de autentificar usuario

Los últimos pasos que realizaremos en nuestra forma serán crear una instancia para la clase ControldeAcceso y el método para el evento AceptarClick. La creación de la instancia se llevará a cabo al momento de crear la ventana, por lo cual será necesario agregar el método FormCreate que se activa inmediatamente cuando la aplicación crea la forma.

A continuación los métodos agregados

```
procedure TGUI_ID.FormCreate(Sender: TObject);
begin
  // Al momento de crear la forma se crea también una instancia para la clase de validación
  CA := ControldeAcceso.Create();
end;

procedure TGUI_ID.AceptarClick(Sender: TObject);
begin
  // Se pasa el control de las entradas a la instancia de la clase ControldeAcceso
  if not(CA.valida(Login.Text, Password.Text)) then
    GUI_ID.Close
  else
    // implemente la creación de la ventana principal para su aplicación ....
end;

procedure TGUI_ID.CancelarClick(Sender: TObject);
begin
  // Al término cerramos la aplicación
  GUI_ID.Close;
end;
```

Creación del código para la clase ControldeAcceso

Para la clase de control de acceso ya se había definido un método de valida, cuyo propósito será determinar si los datos entrados en la interfaz son válidos. Para lograr esto la clase necesariamente deberá enviar información a la clase Persistor, ya que es ella quien conoce como acceder a la base de datos.

En el diagrama de secuencia de la figura 3, no esta descrito quien tiene la responsabilidad de crear la instancia para la clase Persistor, de hecho esta aparece como una clase ya creada según representación del mismo diagrama. Pero ya que eso no es cierto se decidió crear un constructor en la clase ControldeAcceso en donde al momento de ser creada la instancia para ella, también invoque la creación de una instancia llamada **Almacen**.

A continuación el listado completo para el código de la clase control de acceso:

```
unit UControldeAcceso;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, UPersistor, ADODB;

type
  ControldeAcceso = class (TObject)
    Almacen:Persistor;
  public
    function valida(login:String;password:String): Boolean;
```

```
   constructor Create();
  end;

implementation

{
******************************** ControldeAcceso ********************************
}
function ControldeAcceso.valida(login:String;password:String): Boolean;
var
  l,p:String;
  query:WideString;
  DS:TADODataSet;
begin
  // Todo el control de los datos lo tendrá el método válida, es decir lo único
  // que esperamos es recibir dos cadenas, así si están en minúsculas o mayúsculas
  // será responsabilidad de este método tratarlo como se deba
  l := LowerCase(login); // se cambia a minúsculas el login
  p := password;         // el password no cambia ya qye puede ser may y min
  query := 'select * from usuario where login='" + l +
                          '" and password='" + p + "'";
  DS := Almacen.Cargar(query);
  if (DS.RecordCount = 0) then
    valida := false
  else
    valida:= true;
end;

constructor ControldeAcceso.Create();
begin
  // se crea una instancia para la clase persistora
  Almacen := Persistor.Create(nil);
end;

end.
```

Creación del código para la clase Persistor

Finalmente el código para la clase Persistor se describe en esta sección. La función como se ha explicado con anterioridad para esta clase es conocer como acceder a la base de datos determinando el controlador y los mecanismos para la ejecución de querys.

La clase Persistor contaba con la definición de dos métodos uno Cargar y otro Insertar, pero debemos recordar que esta clase contiene dos propiedades del tipo TDataSource y TADODataSet los cuales en su momento deberán ser inicializados, el proceso de inicialización de estos objetos se delegará dentro del método constructor de la clase Persistor. Por lo tanto se agregará también en este caso un nuevo método constructor el cual se encarga de la inicialización de los objetos.

El código insertar no se ha realizado en este ejemplo y solo queda bosquejado el cuerpo del método con el propósito de alentar la programación de dicha función en sucesivos ejemplos.

A continuación el listado completo del código:

```
unit UPersistor;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, db, ADODB;
// Se importa Db y ADOdb ya que son requeridas
type
  Persistor = class (TDataModule) //(TObject)//
    DataSet: TADODataSet;
    DataSource: TDataSource;
  public
    function Cargar(query:WideString): TADODataSet;
    procedure Insertar(query:WideString);
    constructor Create(O:TComponent);
  end;

implementation

{
********************************** Persistor **********************************
}
function Persistor.Cargar(query:WideString): TADODataSet;
begin
  // se crea la conexión con el origen de datos
  DataSet.ConnectionString := 'Provider=MSDASQL.1; Persist Security '+
                        'Info=False;Data Source=MyCcomputo';
  // se efectua la consulta
  DataSet.CommandText := query;
  Dataset.Active := true;
  // se regresa el resultado de la consulta como un data set
  Cargar := DataSet;
end;

procedure Persistor.Insertar(query:WideString);
begin
end;

constructor Persistor.Create(Owner:TComponent);
begin
  DataSet := TADODataSet.Create(nil);
end;

end.
```

Para terminar se muestra el diagrama de clases actualizado luego de haber terminado la escritura del código. El diagrama de secuencia no se presenta actualizado para este ejemplo, pero se recomienda tratar de actualizarlo para que manifieste los cambios realizados en los métodos, sin embargo es necesario comentar que en general el diagrama de secuencia definido conserva su función tal y como fue modelado.

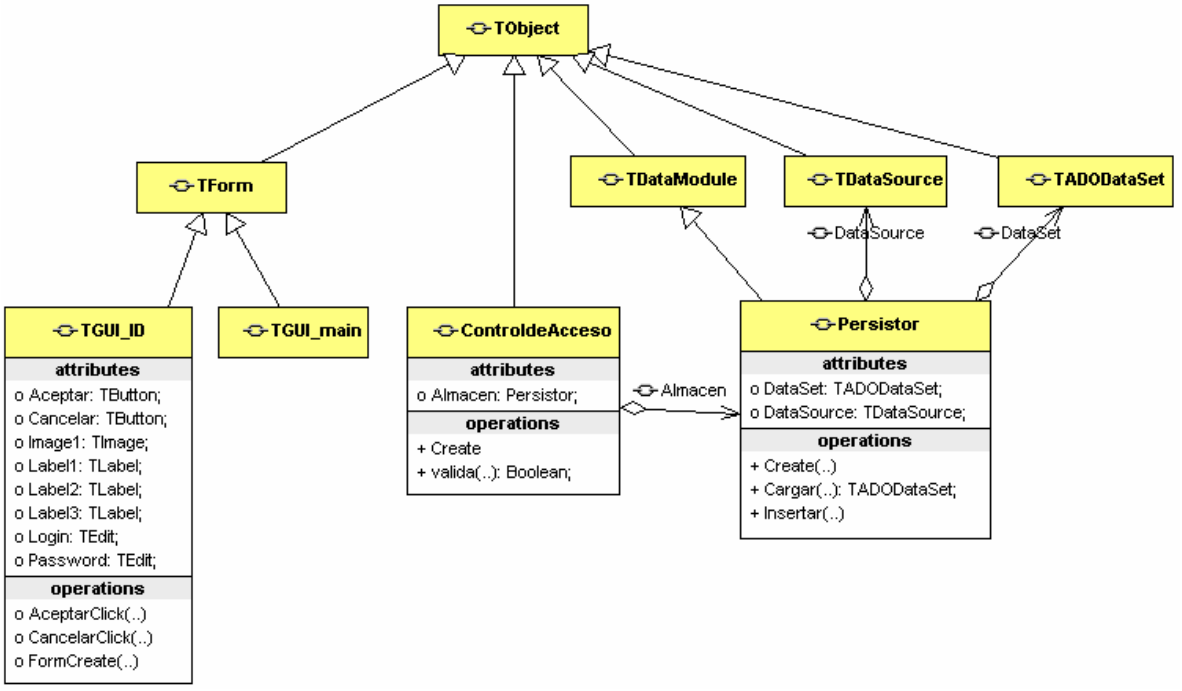

Fig. 8, Diagrama de clases actualizado

Notas finales

Para poder llevar a cabo la ejecución de este proyecto deberá crear un origen de datos desde el ODBC Data Source Administrator llamado **MyCcomputo**.

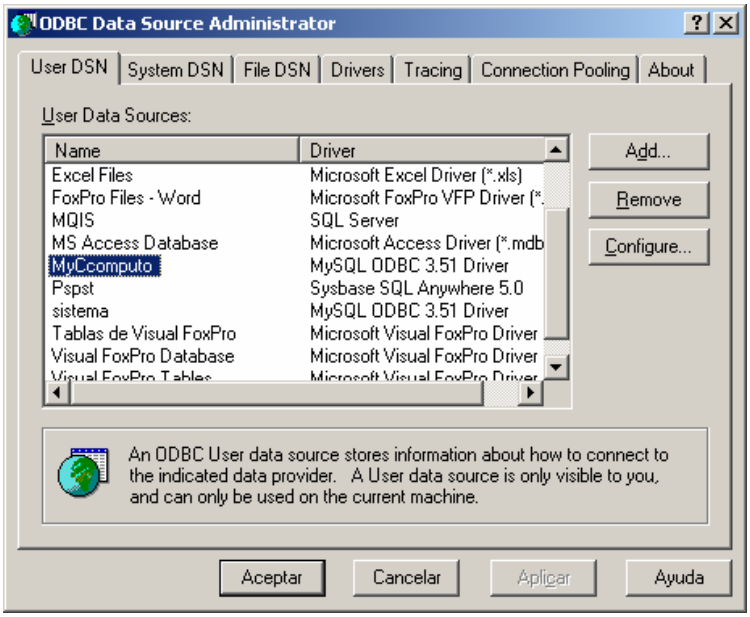

Fig. 9, Configuración del origen de datos

El nombre de la base de datos y la tabla usada para este ejemplo se adjunta a continuación pero puede ser adecuada a sus necesidades:

```
/*
SQLyog v4.07
Host - 5.0.15-nt : Database - ccomputo
**********************************************************************
Server version : 5.0.15-nt
*/
create database if not exists `ccomputo`;

USE `ccomputo`;

/*Table structure for table `usuario` */

drop table if exists `usuario`;

CREATE TABLE `usuario` (
  `ID_u` int(11) NOT NULL auto_increment,
  `nombre` varchar(40) default NULL,
  `paterno` varchar(40) default NULL,
  `materno` varchar(40) default NULL,
  `login` varchar(10) default NULL,
  `password` varchar(10) default NULL,
  PRIMARY KEY  (`ID_u`),
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Descripción de los archivos contenidos para este ejemplo:

ID_usuario.mpb	Archivo conteniendo el modelado en Model Maker
ID_usuario.dpr	Archivo del proyecto Identificación de usuario en Delphi
UGUI_ID.PAS	Clase generada en Model Maker para la interfaz
UGUI_ID.dfm	Forma asociada a la clase de interfaz de usuario
UPersistor.PAS	Clase generada en Model Maker para acceder a la base de datos
UControldeAcceso.PAS	Clase generada en MM para validar los datos de la Interfaz