3. Descripción General.

3.1. Comentarios preliminares.

3.2. Formulario facercade.

3.3. Formulario fbuscar_cita.

3.4. Formulario fconsulta.

3.5. Formulario fconfiguracion.

3.6. Formulario fmodificar_cita.

3.7. Formulario fmodificar_tabla_clas

3.8. Formulario fnavegar_clientes.

3.9. Formulario fprincipal.

3.10. Formulario finforme_citas_hoy.

3.11. Archivo sigpyme.cpp

3.12. Archivo datosBBDD.cpp

3.13. Trucos.

3. Descripción General.

En este capítulo se pretende hacer una descripción del presente proyecto de fin de carrera entrando en el nivel de detalle necesario en cada momento para que quede todo perfectamente explicado. En primer lugar se hará una descripción general del proyecto para después pasar a comentar todos los formularios (ventanas) en los que se divide la aplicación.

3.1. Comentarios preliminares.

En esta sección pretendo explicar varias cuestiones:

- Los convenios usados para nombrar formularios, archivos de código, variables, etc…
- La estructura de la base de datos.
- Características principales de la aplicación.

3.1.1. Convenios.

Empecemos primero por los convenios usados en el software:

- Por norma, a los nombres de los formularios se les ha exigido que comiencen siempre por la letra "f", de "formulario". El resto del nombre va con letras minúsculas, y si consta de más de una palabra, las distintas palabras van separadas por el carácter "_". Por ejemplo: "fbuscar_cita".
- Las "Units" (piezas de código C++ asociadas a cada formulario) tienen como nombre el mismo que el del formulario pero sin la "f" inicial. Por ejemplo: "buscar_cita.cpp". El nombre del archivo de cabecera correspondiente sería "buscar_cita.h", al cual le asigna el nombre automáticamente C++ Builder a partir del ".cpp". Además se asigna el mismo nombre pero con extensión ".dfm"

al archivo dfm, que contiene los datos del formulario y de todos los componentes que lo integran: los tamaños, colores y demás valores fijados en tiempo de diseño.

- ❖ Los controles y demás componentes de C++ Builder también tienen una técnica especial para asignar el nombre. El nombre se compone de dos partes: la primera es la de la clase de la que proviene el objeto, por ejemplo "*DataSource*", la segunda parte del nombre es algo significativo que explique la utilidad del componente, por ejemplo "Agenda". El nombre del objeto finalmente sería: *DataSourceAgenda*, que indica que es un componente de tipo *DataSource* que está relacionado con la base de datos agenda. La única excepción a esta regla la componen las tablas, en las que en la primera parte del nombre se ha usado "Tabla" en lugar de "Table".

Sigamos ahora con los convenios usados en la presente memoria:

- Los nombres de archivos se mostrarán entre comillas dobles "".
- Los identificadores (variables, propiedades, componentes…) con cursiva.
- Los valores de variables y propiedades de los componentes estarán en negrita.
- Los nombres de funciones y métodos de los componentes van subrayados.
- Los trozos de código incrustados en el programa se incluyen con el tipo de letra "Century Gothic" de tamaño 10, para diferenciarlos bien del resto de la memoria.

La programación con C++ Builder se puede ver como la programación de formularios, desde un punto de vista superficial. Por ello he decidido dividir el análisis del código de la aplicación en formularios y los siguientes puntos de esta memoria se reducen a comentar los distintos formularios que componen la aplicación. Con el análisis de todos los formularios se cubre prácticamente todo el código del programa. Lo único que queda por comentar es dos archivos: "sigpyme.cpp" y "datosBBDD.cpp". El primer archivo es el del proyecto "sigpyme.bpr", es el archivo "principal" y contiene la función "WinMain", que es equivalente a la función "main" de un programa en C clásico. El archivo "datosBBDD.cpp" es el correspondiente al DataModuleBBDD. Ambos se comentarán después de los formularios.

3.1.2. Características principales de SIGPYME.

En este apartado se van a comentar a grandes rasgos las características principales de la aplicación SIGPYME. Más tarde, en el análisis del código, se verá la implementación concreta de dichas características:

- Se guardan los valores de los parámetros de configuración de la aplicación en el registro de Windows. El registro de Windows© es como una inmensa base de datos de dicho sistema operativo. Es una base de datos jerarquizada con forma de árbol. Tiene una serie de nodos colgando del nodo raíz (llamado miPC): "HKEY_CLASSES_ROOT", "HKEY_CURRENT_USER", "HKEY_LOCAL_MACHINE", "HKEY_USER", y "HKEY_CURRENT_CONFIG". Puede haber más nodos (llamados claves en Windows) dependiendo del software instalado en la máquina. Cada una de estas claves tiene muchas más ramas y hay bastantes niveles de profundidad. El uso principal del registro de Windows es guardar la configuración del sistema y de todas sus aplicaciones, y, por eso, decidí usarlo con tal propósito en SIGPYME. Así se consigue que cuando el usuario establezca la configuración del programa, ésta se conserve las siguientes veces que ejecute SIGPYME.
- Capacidad de creación de las bases de datos en tiempo de ejecución. SIGPYME se instala con las bases de datos vacías para que el usuario pueda rellenarlas a su antojo. El programa, mediante la ventana de configuración, está preparado para usar más de un juego de bases de datos. Por ejemplo, el usuario puede usar las bases de datos que instala SIGPYME por defecto y además otras bases de datos situadas en un directorio distinto. Esto puede servir, por ejemplo, para gestionar dos empresas distintas con el mismo PC e instalando SIGPYME una sóla vez. Lo único que habría que hacer es entrar en la ventana de configuración e indicarle al programa donde están ubicadas las bases de datos que quiera usar en ese momento. Todo esto tiene un problema: la primera vez que se cambie la ruta de las bases de datos, inevitablemente la aplicación no encontrara la estructura (tanto de directorios como de tablas) que espera. Tal problema se soluciona creandola. Así se permite que en esa nueva ubicación haya unas nuevas bases de

datos vacías para que el usuario pueda hacer uso de ellas. Esta utilidad de SIGPYME también se aplica para el caso de que la aplicación encuentre que falta una de las tablas requeridas, en cuyo caso se crearía automáticamente.

- Uso de SQL para las consultas. Éste fue uno de los objetivos fundamentales del proyecto, hacer uso del lenguaje estándar SQL. Su página web www.sql.org. En este proyecto he incluido un apéndice sobre el lenguaje SQL. Este lenguaje tiene ya muchos años de vida y el paso del tiempo ha demostrado su utilidad y lo valioso que es. Cualquier aplicación que soporte SQL (como Borland C++ Builder) proporciona ventajas al desarrollador ya que no tiene que familiarizarse con un nuevo lenguaje de gestión de bases de datos cada vez que usa un producto distinto. Hoy en día la gran mayoría de los servidores de bases de datos soportan SQL (no suele ser el SQL estándar pero sí un subconjunto de él): Microsoft SQL Server, MySQL, PostgreSQL, etc… Esto da una idea de la importancia que ha adquirido este lenguaje. Al ver el código del formulario de consultas se puede observar cómo se hace la construcción de la sentencia SQL y esto facilita mucho la comprensión del código a cualquiera que conozca previamente dicho lenguaje.
- Detección de que el programa esté corriendo más de una vez. Esta característica sirve para avisar al usuario de que la aplicación está corriendo más de una vez, lo que podría provocar efectos no esperados y nada deseables en las bases de datos. Puede ocurrir que una versión de SIGPYME modifique los datos que la otra versión piensa que tiene, y provocar que la otra versión se interrumpa de forma brusca, e, incluso se podría llegar a una inconsistencia en las bases de datos que, desde luego, no es nada deseable.
- Gestión completa de agenda. Es una de las características principales de SIGPYME: permite la gestión de la agenda del usuario de la aplicación. Se pueden crear citas, borrarlas, modificarlas, etc… Además se dispone de herramientas que permiten buscar citas para una fecha y horas determinadas. También existe la posibilidad de buscar citas para el día actual. En dichas búsquedas se tienen en cuenta los campos *Periodicidad*, *Intervalo* y *Cantidad*. El campo *Periodicidad* indica si la cita tiene carácter periódico o no. *Intervalo* indica el intervalo de periodicidad y el programa acepta 5 posibilidades: semanal, mensual, trimestral, semestral y anual. Por ejemplo, si queremos que el

programa nos avise de un aniversario, pondríamos en la cita la fecha y en *intervalo* el valor "anual"; en el campo *cantidad* habría que poner un "1" porque queremos que nos avise cada año. Si quisiéramos que nos avise cada 4 años, habría que poner un "4" en *cantidad*. Otro ejemplo más, si queremos que la cita sea periódica cada dos meses, habría que poner en *intervalo* el valor "mensual" y en *cantidad* el valor "2".

- Gestión completa de clientes. Con los clientes también se puede hacer lo normal: dar de alta, dar de baja, modificar datos… En el caso de los clientes se ha hecho el tratamiento de forma distinta a las citas, en una sola ventana se puede navegar por todos los clientes, crearlos, borrarlos, editarlos, guardar los cambios, etc.. Los clientes tienen tres campos para clasificarlos: clasificación 1, clasificación 2 y clasificación 3. Los datos de los clientes se ponen con un color distinto, según el campo clasificación 1, para facilitar el uso de los clientes. El usuario tiene la opción de modificar las tablas de clasificación 1, 2 y 3 a su gusto. Los cambios que introduzca en estas tablas se verán reflejados en el resto del programa y los valores que introduzca el usuario serán perfectamente usables con posterioridad.
- Utilidad adicional para que en el inicio de windows se avise de las citas del día actual. SIGPYME proporciona una herramienta adicional que se ejecutará en el inicio de windows y que avisará al usuario de las citas para el día actual. Esta herramienta sólo estará disponible a partir del momento en que SIGPYME se ejecute por primera vez. Cuando se ejecute mostrará una rejilla con todos los datos de las citas para el día actual, además hará una comprobación también de las citas en las que la casilla *Periodicidad* está activada. Más adelante se hablará con más detenimiento de esta utilidad.

3.1.3. Estructuras de las bases de datos de SIGPYME.

La estructura de las bases de datos y sus correspondientes tablas se va a describir a continuación. Esta estructura es además la que crea la aplicación de forma dinámica si no encuentra el juego de bases de datos.

Las bases de datos están contenidas en un directorio. Este directorio, después de la instalación de SIGPYME, se llama BBDD y es hijo del directorio donde resida el

software, pero ninguna de estas dos cosas es un requisito obligatorio para hacer uso de un nuevo juego de bases de datos.

Dentro del directorio BBDD se encuentran otros dos directorios: “agenda” y “clientes”, que contienen las bases de datos de agenda y de clientes respectivamente. El directorio “agenda” contiene los archivos correspondientes a dos tablas, la tabla de las citas y la de los intervalos:

Tabla Agenda	agenda.db	agenda.px	Agenda.mb
Tabla Intervalo	Intervalo.db	Intervalo.px	

Los archivos con extensión “db” son las tablas en sí mismas. Los demás ayudan a constituir los índices que sirven para ordenar (indexar) las tablas.

En cuanto al directorio “clientes”, en él podemos observar los archivos correspondientes a cinco tablas:

Tabla Clientes	clientes.db	Clientes.px	Clientes.mb	
Tabla Clas1	clas1.db	Clas1.px		
Tabla Clas2	clas2.db	Clas2.px	Clas2.xg0	Clas2.yg0
Tabla Clas3	clas3.db	Clas3.px	Clas3.xg0	Clas3.yg0
Tabla Provincias	provincia.db	Provincias.px		

Todos los archivos que se ven y que no tiene extensión”db” sirven para mantener los índices de las distintas tablas.

Las tablas “provincias” (en la base de datos “clientes”) e “intervalos” (en agenda) no son modificables por el usuario, o sea, su contenido está fijado de antemano. La primera contiene todas las provincias de España, la segunda todos los intervalos de periodicidad posibles en una cita. Cuando SIGPYME se instala por primera vez, rellena estas tablas con los valores adecuados. En la creación dinámica de bases de datos, que ya se ha comentado con anterioridad, si se crean algunas de estas dos tablas, también se rellenan con todos sus datos apropiados, para que el usuario, al partir de cero, lo haga con un juego de bases de datos consistentes.

Llega ahora el momento de mostrar la estructura de cada una de las tablas. Las *tablas* están compuestas de *registros* (filas) y cada *registro* está formado por *campos* (columnas). A continuación se detallan los campos que componen cada uno de los registros de las distintas tablas.

1. Tabla Agenda.

Campo	Tipo	Tamaño	Llave
IDCaso	AutoIncremento		SÍ
Fecha	Fecha		
Hora	Hora		
Periodicidad	Lógico		
IDIntervalo	Entero		
Cantidad	Entero corto		
Cita	Memo	240	

2. Tabla Intervalo.

Campo	Tipo	Tamaño	Llave
IDIntervalo	Entero Corto		SÍ
Intervalo	Alfanumérico	20	

3. Tabla Clientes.

Campo	Tipo	Tamaño	Llave
IDCliente	AutoIncremento		SÍ
IDClas1	Entero		
IDClas2	Entero		
IDClas3	Entero		
Nombre	Alfanumérico	20	

Apellidos	Alfanumérico	40	
Tfno	Alfanumérico	12	
Email	Alfanumérico	30	
Domicilio	Alfanumérico	30	
CP	Alfanumérico	5	
Localidad	Alfanumérico	20	
IDProv	Alfanumérico	3	
Comentarios	Memo	240	

4. Tabla Clasificación 1

Campo	Tipo	Tamaño	Llave
IDClas1	Entero		SÍ
Clas1	Alfanumérico	20	

5. Tabla Clasificación 2.

Campo	Tipo	Tamaño	Llave
IDClas1	Entero		SÍ
IDClas2	Entero		
Clas2	Alfanumérico	20	

6. Tabla Clasificacion 3.

Campo	Tipo	Tamaño	Llave
IDClas2	Entero		SÍ
IDClas3	Entero		
Clas3	Alfanumérico	20	

7. Tabla Provincias.

Campo	Tipo	Tamaño	Llave
IDProv	Alfanumérico	3	SÍ
Provincia	Alfanumérico	20	

En las columnas de llave, sólo hay un campo indicado como "SÍ", ya que hay sólo una llave por tabla. La llave (o clave) sirve para ordenar la tabla según el valor de ese campo. Es muy útil para indexar la tabla y para hacer búsquedas basadas en la clave, porque así son más rápidas.

Sobre el campo tamaño, hay que reseñar que cuando está vacío se debe a que se corresponde con un tipo de variable que tiene un tamaño ya prefijado, como por ejemplo, un campo de tipo "entero"; su tamaño es fijo y al desarrollador no se le permite cambiarlo, simplemente porque no tiene sentido.

En cuanto a los tipos de los campos, se puede observar que hay varios:

- Entero, es un campo en el que sólo se pueden introducir números enteros, sin parte decimal.
- Entero Corto, es igual que el anterior pero el rango de números enteros que se puede representar es menor. Se usa en algunos campos de identificación (ID).
- Alfanumérico, es un campo que puede contener cualquier carácter.
- Autoincremento, es un tipo de campo algo especial. Este campo no lo controla el usuario de la aplicación, sino la propia base de datos. Este campo sólo contiene números enteros. Con este tipo de campo se consigue que el nuevo valor sea el del campo con mayor valor ya existente incrementado en una unidad. Por ejemplo, si el valor más grande de ese campo que existe en la actualidad es 66, el valor que se asignará a ese campo en un nuevo registro será el 67. Este mecanismo permite que cada registro de la tabla tenga un valor que le identifique de forma unívoca, el del campo de tipo Autoincremento. Además, este campo es el candidato ideal para ser la clave de la base de datos, ya que nunca va a haber dos valores iguales, que es un requisito de la llave de una tabla de una base de datos.

Una vez finalizado el estudio de las bases de datos usadas en SIGPYME, procedemos al análisis del código formulario a formulario.

3.2. Formulario facercade.

El aspecto del formulario es el siguiente:

Fig. 3.1: Apariencia del formulario *facercade*.

No hay nada que comentar en cuanto al código, ya que es prácticamente inexistente. En esta ventana se muestran tres cosas:

- El autor de la aplicación.
- La versión de la aplicación.
- La licencia bajo la cual se distribuye la aplicación, que es GPL, lo que implica que, junto con la aplicación se distribuye el código fuente. Para más información, consultar www.gnu.org, o consultar el apéndice en el que se muestra el contenido de la licencia GPL.

3.3. Formulario fbuscar_cita.

Este formulario sirve para buscar citas, como su propio nombre indica y, además, para borrar citas. Esto se justifica de la siguiente manera: primero se hizo el formulario para buscar citas. Más tarde, cuando se procedió a implementar el formulario necesario para borrar citas, se observó que primero es necesario hacer una búsqueda para poder localizarla. Entonces se pensó en coger casi todo el código de fbucar_cita y copiarlo en fborrar_cita. Pero también se barajaron otras opciones y se vio que ni siquiera eso era necesario, bastaba con reutilizar el formulario para que, una vez localizada la cita se diera la oportunidad de borrarla. Más tarde, se comentará el código asociado al formulario y, poco a poco, veremos cómo se ha hecho.

En primer lugar, se puede observar la apariencia del formulario cuando se va a buscar una cita:

Fig. 3.2: Formulario *fbuscar_cita* cuando se va a buscar una cita.

Cuando se va a borrar una cita, tiene este aspecto:

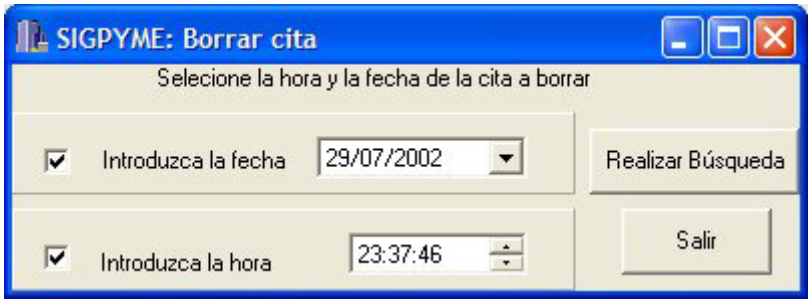

Fig. 3.3: Formulario *fbuscar_cita* cuando se va a borrar una cita.

Vemos que lo que cambia es el texto del título de la ventana y el de la etiqueta de texto. Esto sirve para orientar al usuario. Se le informa de en qué tarea está y de los pasos a seguir. Se hace para evitar confusiones, ya que el usuario ve casi la misma ventana cuando va a buscar una cita y cuando va a borrarla.

El formulario sirve para buscar una cita en un principio. Para ello dispone de dos criterios de búsqueda, la fecha y la hora de la cita. Si se rellenan ambos espacios del formulario y se activan las correspondientes casillas de verificación, se buscará usando ambos parámetros. Si sólo se activa una de las casillas se buscará sólo con el parámetro correspondiente (la fecha o la hora).

Una vez establecidos los parámetros de búsqueda se ha de pulsar el botón buscar. Si se encuentra algún registro, se mostrará una rejilla, como se ve en esta imagen:

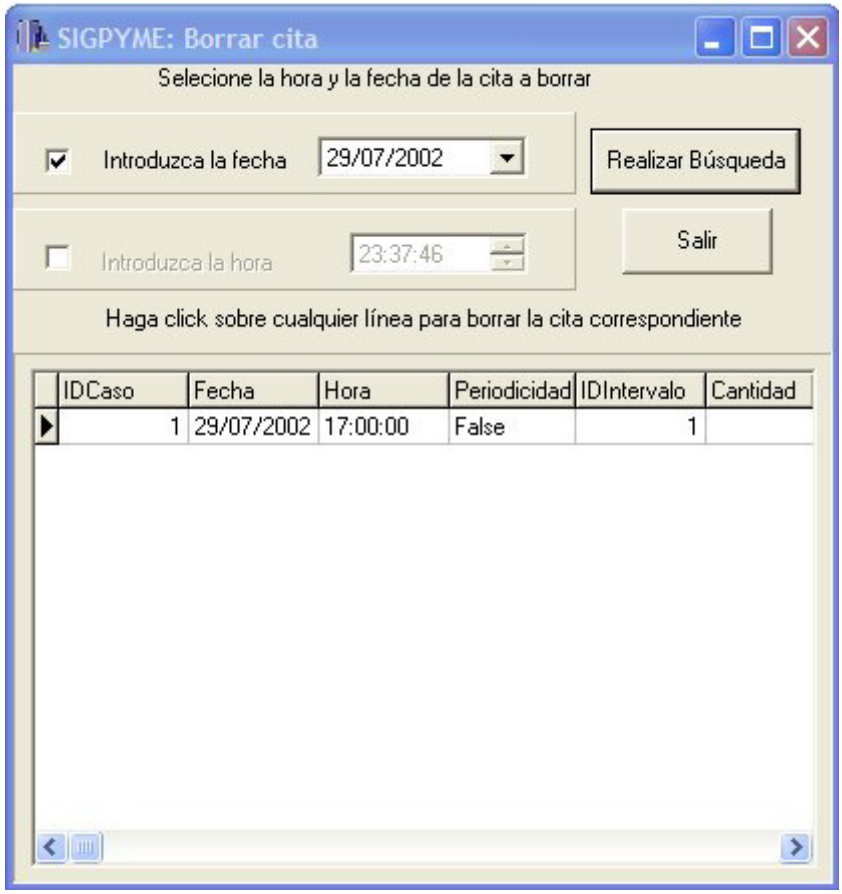

Fig. 3.4: Formulario *fbuscar_cita* cuando se ha realizado una busqueda.

En la rejilla aparecerán uno o más registros, tantos como citas cumplan los requisitos de búsqueda. Al hacer clic en uno de los registros, pueden suceder dos cosas:

- Si estábamos buscando un registro, se procederá a mostrarlo en una ventana nueva.
- Si estábamos borrando el registro, se mostrará un mensaje pidiendo confirmación para el borrado. En caso de que la confirmación sea positiva, la cita será borrada inmediatamente, informando al usuario de tal hecho una vez dada de baja.

Pasemos ahora a analizar el código:

Función BotonBuscarClick.

Lo primero que se hace en esta función es construir la sentencia SQL que se usará para hacer la búsqueda. Para ello se define en primer lugar la primera parte de la sentencia:

```
AnsiString ConsultaSql = "select * from agenda where ";
```

Luego se construye el resto de la sentencia SQL teniendo en cuenta qué casillas de verificación están activadas o no. En el código se puede ver como hay algunos `if` anidados para construir la sentencia.

Posteriormente se usa el componente QueryCitas del Módulo de datos denominado "DataModuleBBDD" para ejecutar la sentencia SQL generada anteriormente. Si dicha ejecución produce más de cero registros, se muestra la rejilla con los resultados. Si no se halló ningún registro no se muestra dicha rejilla, y en cambio, sí un mensaje informando de tal hecho (con la ayuda de la función ShowMessage de Builder).

Función CheckBox1Click.

Esta función responde al evento de hacer clic en el *CheckBox*1 (el encargado de la fecha). Si el *checkbox* está activado, habilita (poner propiedad *Enabled* fijada a **true**) los controles *LabelBuscarFecha* (etiqueta) y *DateTimePicker1* (cuadro para introducir fecha u hora). En caso contrario, deshabilita ambos controles, lo cual hace que el usuario no pueda hacer uso de ellos. El usuario lo puede apreciar debido a que el texto de la etiqueta se sombrea (efecto típico de Windows cuando una opción no está disponible), y además no puede introducir datos en el campo de fecha dispuesto a tal efecto.

Función CheckBox2Click.

Actúa cuando se pulsa el botón primario del ratón sobre el *CheckBox2* y tiene el mismo efecto sobre la hora que la función anterior sobre la fecha. Por ello no se hace necesario comentarlo aquí de nuevo.

Función FormActivate.

Se llama a esta función cuando se activa el formulario. En ella nos aseguramos de que la rejilla y el panel que la contiene están ocultos poniendo la propiedad *Visible* de ambos a **false**.

Función BotonSalirClick.

Llama al método *Hide()*, para esconder el formulario cuando se pulsa el botón salir.

Función DBGridCitasCellClick.

Esta función actúa cuando se hace clic en una celda de la rejilla. Según estemos borrando citas o buscándolas, pedirá confirmación para el borrado o mostrará la cita. Esto se hace basándonos en la variable *BorrarCita*, declarada en "buscar_cita.h".

En el formulario principal, cuando se llama a *fbuscar_cita* para buscar una cita, se pone el *Caption* correspondiente al formulario. La propiedad *Caption* del componente *Label* también se fija con la cadena adecuada y a la variable *BorrarCita* se le asigna el valor **false**. Si se llama al formulario cuando se quiere borrar una cita, esta variable se pone a **true.** Podemos observar esto en estas porciones de código de principal.cpp:

```
fbuscar_cita->BorrarCita = false;
fbuscar_cita->Caption = "SIGPYME: Buscar Citas";
fbuscar_cita->Show();
```

Se observa, como entre otras cosas, la propiedad *BorrarCita* se pone a **false**. Sin embargo, en el código de la función Tfprincipal::Buscarcita1Click, se aprecia como se pone dicha propiedad a **true**:

```
fbuscar_cita->BorrarCita = false;
fbuscar_cita->Caption = "SIGPYME: Buscar Citas";
fbuscar_cita->Show();
```

Volviendo a la función que estamos tratando (DBGridCitasCellClick), lo primero que se hace es chequear el valor de la propiedad *BorrarCita*. Si es **false**, se llama simplemente a modificar_cita, para mostrarla. Antes de ello, hay que localizar en la tabla original el registro en el que se ha hecho clic en la rejilla de la consulta. Si el valor de *BorrarCita* es **true**, se hace uso de la función *MessageBox* para pedir la confirmación de borrado. Si la confirmación es positiva, se construye una sentencia SQL para el borrado de la cita y se ejecuta dicha sentencia con ayuda del *QueryCitas*, con lo que se hace el borrado definitivo del registro que se había localizado antes. Para dejar la rejilla en un estado consistente, volvemos a construir la sentencia SQL de búsqueda de registros (que ya se construyó en la función BotonBuscarClick), según los parámetros que están fijados en los controles del formulario. Esta sentencia se ejecuta, y según se devuelvan registros, se mostrará la rejilla o no. Finalmente, se muestra un mensaje diciendo que el borrado de la cita se ha llevado a cabo.

Componentes usados en este formulario:

- *LabelBuscarFecha*, se usa para mostrar el mensaje de que se ha de introducir la fecha.
- *LabelBuscarHora*, lo mismo que el anterior pero con la hora.
- *DateTimePicker1*, se usa para escoger la fecha. Cuando se hace click en el botón del lado derecho del campo se obtiene un calendario, para hacer la elección de la fecha más cómoda.
- *DateTimePicker2*, se usa para escoger la hora. Tiene en el lado derecho dos botones, que sirven para incrementar y decrementar la hora.
- *CheckBox1*, sirve para habilitar o deshabilitar la hora como criterio de búsqueda.
- *CheckBox2*, sirve para lo mismo, pero con la hora.
- *BotonBuscar*, este botón lanza la búsqueda de los registros según los criterios fijados por el usuario.
- *BotonSalir*, para salir del formulario.
- *PanelCitas*, es el panel que sirve para mejorar el aspecto del formulario cuando está en su forma extendida. Este panel contiene el DBGrid. La pareja formada por ambos componentes es la que se hace visible cuando se lleva a cabo una consulta.
- *DBGridCitas*, es la rejilla que sirve para mostrar los resultados de la búsqueda.
- *Panel2,* es el panel superior. Contiene la casilla de verificación *CheckBox1*, la etiqueta *LabelBuscarFecha* y el control *DateTimePicker1*.
- *Panel3*, el panel que se encuentra debajo de Panel2. Contiene la casilla de verificación CheckBox2, la etiqueta LabelBuscarHora y el control DateTimePicker2.
- LabelIndicacion, sirve para poner texto en el formulario basado en el contexto de la aplicación. Este texto varía en función de si vamos a borrar una cita o sólo estamos buscando citas. Esta etiqueta sólo es visible cuando se ha llevado a cabo una cita y se ha obtenido algún resultado.
- LabelIndicacion2, al igual que la etiqueta anterior, muestra texto en función de que vayamos a borrar citas o a buscarlas. A diferencia de la etiqueta anterior, ésta se encuentra visible siempre. Es la que se puede observar en la parte

superior del formulario.

3.4. Formulario fconsulta.

El formulario tiene el siguiente aspecto:

Fig. 3.5: Formulario *fconsulta.*

El código del archivo ".cpp" y el ".h" se puede observar en el apéndice del código fuente de SIGPYME.

La primera elección que se ha de hacer en este formulario es la de la tabla en la que se desea realizar la búsqueda: en la de clientes o en la de agenda. Una vez hecho esto, se procede a establecer los criterios de consulta, en la zona superior derecha de la ventana. En la tabla se observa que inicialmente sólo hay habilitada una fila. Cada fila de la tabla supone una restricción en la búsqueda (o consulta) a realizar. Se pueden añadir tantas filas (restricciones) como se deseen pulsando el botón con el símbolo "+". Para borrar la última restricción se usa el botón con el símbolo "-". Cada restricción consta de 4 partes:

- Activo, sirve para especificar si la restricción correspondiente está activa o no. Si no está activa, no se incluirá a la hora de la búsqueda. Esto le otorga una gran flexibilidad al usuario a la hora de implementar búsquedas, ya que puede realizar

varias búsquedas consecutivas basadas en el mismo juego de restricciones, pero activando unas restricciones y desactivando otras en cada consulta.

- Campo, es un menú desplegable en el que se encuentran a disposición del usuario los campos sobre los que se pueden aplicar restricciones. En el caso de que se seleccionase en un principio la tabla agenda para la búsqueda, serán sus campos los que se encuentren en este menú desplegable. Si la tabla es clientes, entonces son sus campos los que están a disposición del usuario.

- Criterio, este menú desplegable ofrece el operador para el criterio correspondiente. El operador depende del tipo del campo seleccionado en el menú desplegable anterior.

  - Si el campo es **Alfanumérico**, los operadores disponibles en este menú son "CONTIENE" y "ES".
  - Si el campo es de tipo **Lógico** (o Booleano) las opciones serán "ES" y "NO ES".
  - Por último si el campo es N**umérico**, las posibilidades ofrecidas serán: "MAYOR QUE", "MAYOR O IGUAL QUE", "IGUAL QUE", "MENOR O IGUAL QUE" y "MENOR QUE".

- Valor, aquí se ofrece lo único que falta para completar una restricción, que es el valor que va a la derecha del operador y pieza clave de la restricción.

Una vez que se han establecido todos los requisitos de búsqueda, se puede hacer clic en el botón "buscar", y si se encuentran resultados, la ventana pasará a tener este aspecto:

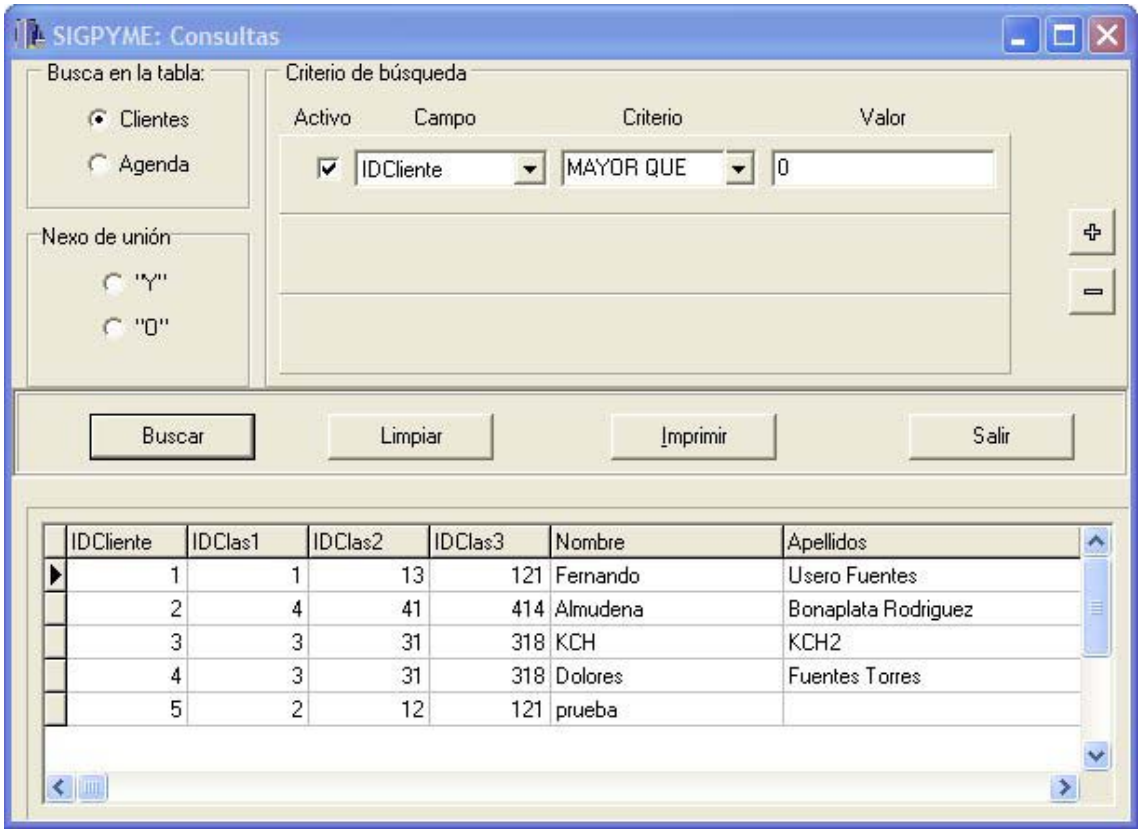

Fig. 3.6: Formulario *fconsulta* una vez realizada una consulta.

En él se muestra una rejilla con los resultados obtenidos en la consulta. Al pulsar el botón “Imprimir”, se obtiene un listado por pantalla de los registros resultantes de la consulta. Este listado es el paso previo a sacarlo por impresora, que se puede hacer de forma automática pulsando el botón con el icono de una impresora, que se puede observar en la botonera de la parte superior de cualquier informe que tiene SIGPYME.

En cualquier momento se puede pulsar el botón "Limpiar" y desaparecerán todos los criterios de búsqueda y también la rejilla. Así se puede pasar a realizar una búsqueda que no tenga nada que ver con las anteriores. El botón "Salir" sirve para volver a la pantalla principal de la aplicación SIGPYME.

Pasemos ahora a comentar el código fuente de "consultas.cpp":

Función IniciarDatos.

Se llama a esta función cada vez que se activa el formulario y cuando se pulsa el botón “Limpiar”. Lo primero que hace es no hacer visible la rejilla de resultados, porque al principio del formulario no se ha hecho ninguna consulta.

Después se procede a comprobar el archivo "consulta.db" que contiene la tabla denominada "TablaConsulta". Esta tabla sirve para guardar los criterios de la búsqueda que se va a realizar. Es importante observar que los criterios de búsqueda se almacenan en una tabla. Esto permite gestionar las restricciones de una forma mucho más cómoda. Si el archivo no existe, se crea y si existe, se vacía la tabla.

La siguiente tarea consiste en rellenar el menú desplegable "Campo" de los criterios de búsqueda. Se rellena con los nombres de los campos de la tabla "Clientes" o los de "Agenda", según esté seleccionado en los botones de radio de la parte superior izquierda del formulario.

También se comprueba si existe una consulta anterior, y en caso afirmativo, se establece como la consulta actual. Si no existe una consulta anterior y la Tabla de Consultas se encuentra vacía, se crea la primera entrada (restricción) con valores por defecto. Esto servirá para orientar al usuario.

Función BotonSalirClick.

Controla la pulsación del botón "Salir". Lo único que hace es cerrar el formulario y guardar en disco la consulta actual, ya sea de clientes o de agenda.

Función BotonBuscarClick.

Esta función responde al evento de pulsación del botón "Buscar". No realiza una comprobación de la sintaxis de los datos introducidos por el usuario porque dicha comprobación se hace en la función DBEValorKeyPress, que se considera un lugar más adecuado. En esta función lo primero que se hace es construir la parte de la sentencia SQL que refleja las restricciones. Esto se hace con la sentencia:

```
Restricciones += campo + criterio + valor + "AND";
```

La variable *campo* se extrae de la tabla TablaConsulta. La variable *criterio* se obtiene también de la tabla y se transforma a algo que se pueda introducir en una sentencia SQL. Por ejemplo, el criterio "MAYOR O IGUAL QUE" se transforma en ">=". La variable valor se obtiene directamente de la tabla TablaConsulta.

Tenemos que hacer una breve parada para analizar el caso de la variable *campo*. Existen algunos campos que se ofrecen para construir las restricciones que son campos "artificiales". Es el caso del campo "intervalo" en la base de datos Agenda y de los campos "Clasificacion1", "Clasificacion2", "Clasificacion3" y "Provincia". Si se comprueba las estructuras de las tablas, vemos que en lugar de estos cinco campos los que realmente existen son: "IDIntervalo", "IDClas1", "IDClas2", "IDClas3" e "IDProv". Todo esto implica dos cosas:

1. Hay alguna forma en C++ Builder de construir campos artificiales en las tablas. Son campos que no existen físicamente, pero sí se pueden usar. ¿Cómo se han creado?
2. De alguna forma hay que mapear un criterio basado en un campo artificial con un criterio basado en un campo real.

Respondamos a estas dos cuestiones:

1. Los campos denominados artificiales, se llaman en realidad campos de búsqueda (o lookup) y se crean en tiempo de diseño. La explicación de cómo se hace se deja para la penúltima sección de este capítulo (Trucos).
2. El mapeo de la restricción se hace teniendo en cuenta cada caso de campo lookup. Para ello se incluyen 5 `ifs` en el código, uno tras otro, pero no anidados:

```
if (campo == "Intervalo" )
{
……
}
```

```
if (campo == "Provincia" )
{
……
}
if (campo == "Clasificacion1" )
{
……
}
if (campo == "Clasificacion2" )
{
……
}
if (campo == "Clasificacion3" )
{
…..
}
```

Dentro de cada `if` se hace el tratamiento adecuado para cada campo. Básicamente lo que se hace es buscar el valor del campo ID correspondiente en la tabla adecuada (Intervalo, Clasificacion1, Clasificacion2, Clasificacion3 o Provincia). Más tarde con ese valor del campo ID y con el nombre del propio campo se construye la restricción (para la Tabla Agenda o para la tabla Clientes).

Por ejemplo, supongamos que una restricción es “Provincia = Sevilla”. En la tabla Provincia se busca el registro cuyo campo “Provincia” sea igual a “Sevilla”, se obtiene de ese registro el valor del campo ID correspondiente: “IDProv”, que en el caso de Sevilla vale **SEV**. Y se sustituye en la restricción “Provincia” por “IDProv” y “Sevilla” por “SEV”, quedando así: “IDProv = SEV”. En el código podemos ver cómo se hace esto, dentro del `if` de Provincia. El campo se cambia sí:

```
campo = "IDProv";
```

Para cambiar el valor de la restricción:

```
AnsiString buscar =  TablaConsulta->FieldByName("VALOR")->AsString;
```

```
if ( !DataModuleBBDD->TablaProv->Locate("Provincia",buscar, Opts) )
   ShowMessage("El intervalo no se corresponde con ninguno conocido. \
                                                    Inconsistencia en BBDDs");

valor="\""+DataModuleBBDD->TablaProv->FieldByName("IDProv")->AsString+"\"";
```

En la primera línea se define el elemento a buscar *en TablaProv* (en el caso de nuestro ejemplo sería "**Sevilla**"). En la segunda línea, con la ayuda de `Locate`, se busca en *TablaProv* el registro cuyo campo "Provincia" se corresponda con el elemento a buscar. Si no se encuentra se da un mensaje de error. En la última línea, a la variable valor se le da el valor del identificador *IDProv* del registro actual de *TablaProv*, que en el caso de nuestro ejemplo sería "**SEV**". Además el valor se mete entre comillas dobles para que sea más fácil la construcción de la sentencia SQL.

Una vez construida la sentencia SQL completa, se procede a realizar la consulta, y sólo se muestran los resultados si se ha obtenido como mínimo un registro. Para mostrar los resultados se hacen visibles los controles *Panel1* y *DBGridConsulta*. Como el formulario tiene la propiedad *AutoSize* a **true** (se puede observar en "consulta.dfm"), al hacerse visibles los dos controles anteriores, el formulario adapta su tamaño de forma automática.

En esta función se evita hacer la búsqueda si no se introdujo ningún criterio para llevarla a cabo.

Función BotonLimpiarClick.

Responde al evento de hacer click sobre el botón Limpiar. Lo que se hace es borrar la tabla en la que se almacena la consulta actual (TablaConsulta) y hacer no visible tanto la rejilla de resultados como el panel que la contiene. También se llama a la función IniciarDatos.

Función FormActivate

Sólo se encarga de llamar a IniciarDatos. No se pone aquí el código de IniciarDatos() porque a esta función se le llama también desde otras, como BotonLimpiarClick. Más tarde se comentará esta función.

Función SPAnadirClick

Controla el evento de pulsación del SpeedButton SPAnadir, el botón con el icono "+". Lo primero que se hace es añadir un registro a la tabla de consultas y luego se preparan los valores por defecto de la nueva restricción, para orientar al usuario. Esto último también se hace al final de la función IniciarDatos.

Función SPEliminarClick.

Esta función se ejecuta cuando se pulsa el botón del signo "-", que sirve para eliminar una restricción del criterio de búsqueda. Antes de llevar a cabo dicha operación, se comprueba si la tabla estaba en modo edición o modo inserción para cambiar su estado.

Función DBComboBoxCampoChange

Esta función responde al evento de cambio en el componente ComboBoxChange, o sea, actúa después de que se haya cambiado el contenido de dicho componente.

Es necesario actuar puesto que hay que cambiar el valor del ComboBox de criterio, ya que no tendrá las mismas opciones el comboBox de criterio si el campo a tratar en la restricción es de Alfanumérico que si es de tipo Numérico o Entero.

Para llevar a cabo el cambio necesario en el control DBComboBoxCriterio, se define una variable llamada columna que es una referencia a la columna que haya seleccionado el usuario en DBComboBoxCampo. Luego con esta variable se hace una secuencia de ifs para ver su tipo, y en función de él, rellenar el DBComboBoxCriterio con el valor adecuado, como se puede ver en el código:

```
if (columna->DataType==ftString || columna->DataType==ftMemo)
{
  //Si campo es string
  DBComboBoxCriterio->Items->Add("CONTIENE");
  DBComboBoxCriterio->Items->Add("ES");
}
else if(columna->DataType==ftBoolean)
{
  DBComboBoxCriterio->Items->Add("ES");
  DBComboBoxCriterio->Items->Add("NO ES");
}
else
{
  //Si campo es numérico
  DBComboBoxCriterio->Items->Add("MAYOR QUE");
  DBComboBoxCriterio->Items->Add("MAYOR O IGUAL QUE");
  DBComboBoxCriterio->Items->Add("IGUAL QUE");
  DBComboBoxCriterio->Items->Add("MENOR O IGUAL QUE");
  DBComboBoxCriterio->Items->Add("MENOR QUE");
}
```

Podemos ver que si el campo es de tipo alfanumérico (ftString en el código) los valores del DBComboBoxCriterio serán "CONTIENE" y "ES". Si el campo es de tipo booleano, los valores serán "ES" y "NO ES". Por último, si el campo es numérico se dan las posibilidades: "MAYOR QUE", "MAYOR O IGUAL QUE", "IGUAL QUE", "MENOR O IGUAL QUE" y "MENOR QUE".

En este formulario, existe un componente que es DBEditCriterio, que se superpone al DBComboBoxCriterio. Esto se hace porque si hay dos campos criterios distintos en dos restricciones, no se puede reflejar adecuadamente con el control DBControlGrid, así que se decidió usar otro componente superpuesto que mostrase lo que fuese necesario.

Lo último que se hace en esta función es actualizar el valor del componte DBEditCriterio y si el campo elegido es Booleano, poner por defecto el valor verdadero en la restricción.

Función RadioButtonClientesClick.

Esta función responde al clic en el botón de radio de Clientes. Cuando se pulsa en el *RadioButton*, es porque se ha cambiado la tabla sobre la que se va a hacer la consulta. En ese caso, se procede a salvar la consulta que se había hecho sobre la otra tabla y a reiniciar el formulario con la función IniciarDatos.

Función DBGridConsultaCellClick.

Cuando hacemos clic en alguna celda del DBGridConsulta, se ejecuta esta función, que lo único que hace es llamar al formulario feditar_campo_cita para mostrar el campo sobre el que se ha hecho clic si es de tipo Memo. El formulario feditar_campo_cita se creó en un primer momento para editar el campo Cita de la Tabla Agenda, pero vemos que aquí se ha reutilizado. También se tiene en cuenta si la consulta que estamos haciendo es sobre la tabla Agenda o la tabla Clientes mediante el if adecuado.

Función DBComboBoxCriterioClick.

Esta función actúa cuando se pulsa el DBComboBoxCriterio. Lo único que hace es actulizar el valor del componente DBEditCriterio, que se usa con la misma finalidad que el componente DBEditCampo comentado en la función DBComboBoxCampoChange.

Función SalvarBBDD.

Esta función ya se ha mencionado antes. Se la invoca desde RadioButtonClientesClick. Su misión es guardar la consulta actual en una tabla auxiliar, para permitir posteriormente su recuperación.

Lo primero que hace es comprobar que la tabla de consultas no tenga ninguna operación pendiente, y si la tiene, fuerza a que se lleven a cabo.

Después, hace el volcado desde TablaConsulta a TablaConsultaA o TablaConsultaC, según si el parámetro que se le pasa a la función es "agenda" o "clientes" respectivamente. Para hacer el volcado, hay que comprobar si existen o no TablaConsultaA o TablaConsultaC. Si no existen se crean. Por útlimo se hace el volcado, se vacía el contenido previo de la tabla (si lo hubiera tenido) y se hace la copia de seguridad con ayuda de la función BatchMove.

Función BotonImprimirClick.

Se ejecuta cuando se hace clic sobre el botón "Imprimir". Sirve para sacar un informe de los resultados de la consulta efectuada. Se hace simplemente llamando al método Preview del objeto QuickRep1. Antes de dicha llamada, hay que comprobar si la consulta actual es de clientes o de agenda, para llamar al objeto QuickRep1 del formulario adecuado: finforme_consulta_clientes o finforme_consulta_cita.

Componentes usados:

- *TablaConsulta*, tabla en la que se guardan todas las restricciones de la consulta actual. Facilita mucho la gestión de las consultas.
- *BotonBuscar*, para realizar la búsqueda.
- *BotonLimpiar*, para limpiar la zona de rellenado de restricciones y hacer desaparecer la rejilla de resultados.
- *BotonSalir*, para salir del formulario y volver a la pantalla principal.
- *TablaConsultaC*, es una tabla para guardar la anterior consulta a la tabla de clientes, su contenido se recupera si se activa el formulario y está marcado que se va a realizar una consulta de clientes.
- *TablaConsultaA*, equivalente a la anterior, pero refiriéndonos a la tabla agenda.
- SpeedButtonAnadir, botón que sirve para añadir un criterio de búsqueda más a la consulta, o sea, habilita una fila más para que el usuario pueda introducir los

datos correspondientes a una nueva restricción.

- *GroupBox1*, es un componente que se usa para agrupar controles relacionados en un formulario. Éste contiene dos: RadioButtonClientes y RadioButtonAgenda.
- *RadioButtonClientes*, es un botón de radio que se usa para indicar que la consulta se va hacer sobre la tabla de Clientes.
- *RadioButtonAgenda*, igual que el anterior pero con la tabla Agenda.
- *GroupBox2*, este componente contiene el DBCtrlGridConsulta (con todos los componetes que lo conforman), el SpeedButtonAnadir y el SpeedButtonEliminar.
- *DataSourceConsulta*, está asociado a la tabla TablaConsulta.
- *QueryConsultaClientes*, es un componente TQuery para hacer consultas SQL en la tabla de Clientes.
- *QueryConsultaAgenda*, lo mismo que el anterior pero con la tabla Agenda.
- *DBCtrlGridConsulta*, es el componente que contiene todas las restricciones de la búsqueda actual y las guarda en una tabla.
- *DBCheckBoxActivo*, es la casilla de verificación que sirve para indicar si esa restricción está activa o no de cara a la consulta que se va a realizar.
- *Label1*, *Label2*, *Label3* y *Label4*. Son las cuatro etiquetas de texto que aparecen en el formulario: Activo, Campo, Criterio y Valor.
- *Panel1*, es el panel que contiene la rejilla de resultados y que sólo es visible cuando se ha efectuado una consulta y se ha obtenido algún resultado.
- *DBGridConsulta*, es la rejilla de resultados de la consulta.
- *DatabaseConsultas*, es el componente que permite definir en este formulario una base de datos llamada consultas y trabajar con ella.
- *SpeedButtonEliminar*, botón con el signo "-" que permite eliminar la última restricción que haya en el *DBCtrlGrid*.
- *DBComboBoxCampo*, es el menú desplegable que existe en cada restricción para elegir el campo en el que se quiere basar la restricción correspondiente.
- *DBComboBoxCriterio*, lo mismo pero con el criterio de cada restricción.
- *DBEditValor*, lo mismo pero con el valor de cada restricción.
- *DBEditCriterio*, control superpuesto al *DBComboBoxCriterio*. Se usa para poder personalizar el criterio en cada restricción ya que el *ComboBox* no lo permite. El

ComboBox se sigue utilizando para tener un menú desplegable que facilite la elección al usuario.

- *DataSourceQuery*, es el *DataSource* del que se obtienen los resultados de la consulta. En tiempo de ejecución cambia su asociación a *QueryConsultaClientes* o a *QueryConsultaAgenda*, según se necesite.
- *GroupBox3*, es el componente que contiene *RadioButtonY* y *RadioButtonO*, los botones de radio para elegir el nexo de unión entre restricciones.
- *RadioButtonY*, botón de radio para elegir el nexo "Y".
- *RadioButtonO*, botón de radio para elegir el nexo "O"
- *Panel2*, es el panel que contiene los botones "Buscar", "Limpiar", "Imprimir" y "Salir".
- *BotonImprimir*, para obtener un informe de los resultados de la consulta.

3.5. Formulario fconfiguracion.

En este formulario se pueden observar las opciones de configuración de SIGPYME. La ventana que verá el usuario es la siguiente:

Fig. 3.7: Formulario de configuración de SIGPYME.

Antes de empezar a comentar la utilidad de cada una de las opciones disponibles, es necesario comentar que SIGPYME guarda su configuración en el registro de Windows. La justificación de guardar la configuración de la aplicación en el registro reside en que es necesario que estén guardadas de forma permanente, para que vuelvan a estar disponibles tal y como lo estaban la última vez que se ejecutó el programa.

Pero, una vez establecido el requisito de querer que las opciones de configuración estén guardadas de forma permanente, se ha de reseñar que la única posibilidad no es guardar dichos valores en el registro de Windows. Es más, existe otra posibilidad, que es la de guardar las opciones en un archivo en el disco duro. Sin embargo, se decidió tomar la pimera opción por varios motivos:

- Uno de los usos del registro de windows es precisamente ése, guardar la configuración de programas que corran en el sistema, como es el caso de SIGPYME.
- Un archivo corre el riesgo de ser manipulado por un usuario que esté navegando por el disco duro con el explorador de archivos de windows. Esto no sería nada conveniente para la aplicación. Para modificar el registro, sin embargo, el usuario tendría que ejecutar el programa “Regedit”, que no está disponible en el menú inicio de windows, ya que no se considera un programa para un usuario medio y también porque una manipulación incorrecta del registro puede llevar a un fallo general de windows, que en su caso más grave, implicaría una reinstalación del sistema operativo completo. Como vemos, el registro es mucho menos accesible para un usuario que no sepa lo que hace que un archivo cualquiera, y éste es otro motivo para guardar las opciones de configuración de SIGPYME en el registro de windows.
- El último motivo es que si se usa un archivo, hay que decidir el formato del archivo y escribir mucho código para leer el archivo cada vez que se quiera recuperar una opción de configuración, porque habría que hacer algo así como un analizador léxico para recorrer el archivo. El registro, gracias a C++ Builder es muy fácil de usar, como se puede ver en la función IniciarDatos de "principal.cpp" y en la sección “Trucos”, en la que se explica cómo hacerlo.

Volviendo al registro de Windows, las opciones de configuración de SIGPYME se guardan en la ruta "HKEY_LOCAL_MACHINE\SOFTWARE\sigpyme". Si se entra con regedit en dicha ruta, se pueden observar cuatro "valores" (siguiendo la nomenclatura del registro): *Opcion1*, *Opcion2*, *Opcion3* y *Opcion4*. Estos valores se corresponden con las cuatro opciones de configuración que se pueden observar en SIGPYME:

- "Activar cuadro de diálogo al salir de la aplicación", si se activa esta opción el programa preguntará al usuario si realmente desea salir de la aplicación cuando haga clic sobre la opción salir.
- "Geométrica", dentro de "Elección de colores". SIGPYME usa dos algoritmos distintos para la generación de los colores en la ventana de navegación de clientes. Si el usuario quiere usar el algoritmo que produce una asignación de colores geométrica, ha de marcar esta opción.
- "Artimética", dentro de "Elección de colores". Sirve para usar la opción de asignación artimética de colores. La elección de colores se comenta con más detalle en la información concerniente al formulario *fnavegar_clientes*.
- "Directorio donde se encuentran instaladas las bases de datos", esta opción sirve para indicar al programa dónde están instaladas las bases de datos. SIGPYME tiene la cualidad de poder trabajar con distintas bases de datos, aunque el esquema de dichas bases de datos ha de ser siempre el mismo. Si un usuario decide crear unas bases de datos (por ejemplo con el "Database Desktop" del paquete de C++ Builder) por su cuenta, puede usarlas siempre que el formato de las bases de datos sea el adecuado. Dicho formato se comenta en el apartado 3.1.3 de esta memoria. El proceso es muy delicado y cualquier incompatibilidad con el formato correcto puede ocasionar que SIGPYME no funcione de forma adecuada. Junto con esta opción se disponen de "controles adicionales" para que el usuario pueda seleccionar la ruta, en lugar de tener que teclearla completa.

Sobre esta última opción, queda por comentar algunos detalles. Si se le da al programa una ruta para las bases de datos, donde no se encuentran dichas bases de datos, él las crea automáticamente para poder seguir trabajando. También se puede

interpretar como que el usuario quiera comenzar a usar la aplicación desde cero, entonces puede elegir una ruta nueva para las bases de datos y la aplicación las creará en ese lugar vacías. En realidad lo que está vacío es las tablas en las que insertará datos el usuario, hay ciertas tablas a la que el usuario no tiene acceso y cuyos valores se rellenan automáticamente después de la creación de las bases de datos. Dichas tablas son la de intervalos ("intervalos.db") en la base de datos **agenda** y la de provincias ("provincias.db") en la base de datos **clientes**. Existe la posibilidad de que lo que falle en la ruta indicada a SIGPYME sea la no existencia de un directorio de la estructura de las bases de datos, en tal caso la aplicación crearía dicho directorio. También se puede dar la circunstancia de que lo que no exista sea un archivo concreto perteneciente a alguna base de datos, esto también está previsto por la aplicación y también lo crearía para poder seguir trabajando.

Toda esta funcionalidad relacionada con la ruta donde se encuentran las bases de datos se implementa en la función IniciarDatos de "principal.cpp", que se encuentra convenientemente comentada en la explicación del formulario *fprincipal*.

Una vez que el usuario ha hecho los cambios que considere oportunos, si pulsa el botón "Aplicar", los cambios de configuración se guardarán en el registro de windows y se cerraría la ventana. Si pulsa el botón "Cancelar", los cambios se ignoran y se cierra el formulario.

Pasemos ahora a comentar el código del formulario:

Función FormActivate.

Se ejecuta cuando se activa el formulario. Esta función se encarga de que los controles que hay en el formulario tengan los valores adecuados, que son los de configuración que actualmente está usando el sistema. Estos valores son variables del formulario *fprincipal*, y se llaman *GOpcion1*, *GOpcion2*, *GOpcion3* y *GOpcion4*:

- *GOpcion1*, es de tipo booleano, y dice si está o no activada la opción de "Activar cuadro de diálogo al salir de la aplicación".

- *GOpcion2*, es de tipo cadena y da la ruta donde se encuentran las bases de datos actualmente en uso.
- *GOpcion3*, dice si está usada la opción "Geométrica" o no, en el apartado "Elección de colores".
- *GOpcion4*, lo mismo con la opción aritmética.

Función BotonAplicarClick.

Esta función actúa cuando se hace clic en el botón "Aplicar". Se encarga de actualizar las variables *GOpcion1*, *GOpcion2*, *GOpcion3* y *GOpcion4*, pertenecientes a fprincipal. Luego llama a la función IniciarDatos (comentada en el formulario fprincipal), que se encarga de guardar los valores en el registro de windows y hacer todas las comprobaciones necesarias relacionadas con una posible nueva ruta fijada por el usuario para las bases de datos. Estas comprobaciones son las mencionadas más arriba: comprobar si existe la estructura de directorios adecuada y comprobar si existen todas las tablas que conforman las bases de datos "agenda" y "clientes".

Función BotonCancelarClick.

Se llama a esta función cuando el usuario pulsa el botón "Cancelar". Lo único que hace es ocultar el formulario actual.

Función DirectoryListBoxOpcion6Change.

Esta función actúa cuando se produce un cambio en el directorio seleccionado en el control *DirectoryListBoxOpcion6*. En ese caso lo que se hace es poner dicho directorio en el control TEdit, para que el usuario tenga una "realimentación visual" y aprecie que su elección ha sido tenida en cuenta. Después de esto, sólo restará pulsar el botón "Aplicar" si el usuario considera adecuados los valores asignados a las opciones de configuración.

Función BotonImpresoraClick.

Con esta función conseguimos que aparezca un cuadro de diálogo para la configuración de la impresora. Sólo consiste en llamar al método Execute() del objeto *PrintDialog1*.

Componentes usados:

- *Label1*, es la etiqueta con el texto: “Opciones de configuración de SIGPYME”.
- *CheckBoxOpcion1*, es una casilla de verificación para poner a verdadero o falso la opción de "Activar cuadro de diálogo al salir de la aplicación".
- *BotonAplicar*, es el botón “Aplicar”.
- *BotonCancelar*, es el botón “Cancelar”.
- *GroupBox1*, es un componente que contiene estos otros dos: *RadioButtonGeometrica* y *RadioButtonAritmetica*.
- *RadioButtonGeometrica*, se consigue con este componente que se verifique o no la opción "Geométrica" de "Elección de colores".
- *RadioButtonAritmetica*, lo mismo que el anterior pero con la opción "Aritmética".
- *Panel1*, es el panel sobre el que se encuentran todos los componentes para la elección de la ruta de las bases de datos.
- *LabelOpcion2*, es la etiqueta que contiene el texto: “Directorio donde se encuentran instaladas las bases de datos”.
- *EditOpcion2*, sirve para mostrar la ruta de las bases de datos actualmente seleccionada.
- *DirectoryListBoxOpcion6*, es un control para mostrar la jerarquía de directorios y hacer más fácil la elección de una carpeta para el usuario.
- *DriveComboBoxOpcion5*, es el menú desplegable que permite seleccionar la unidad donde se encuentran las bases de datos que se quiere seleccionar. Este componente está conectado al *DirectoryListBoxOpcion6*, y así se consigue que se actualice cuando el usuario cambie la unidad. Cuando se selecciona una unidad en el *DriveComboBoxOpcion5*, el control *DirectoryListBoxOpcion6* se actualiza para mostrar la estructura de directorios y el directorio activo actual en esa unidad.
- *BotonImpresora*, es el botón “Impresora”.

- *PrintDialog1*, es para poder visualizar una ventana de configuración de la impresora. Es un componente que no tiene representación visual como la tiene un botón o una etiqueta. Sólo se manifiesta cuando se llama a su método Execute.

3.6. Formulario fmodificar_cita.

El aspecto de este formulario es el siguiente:

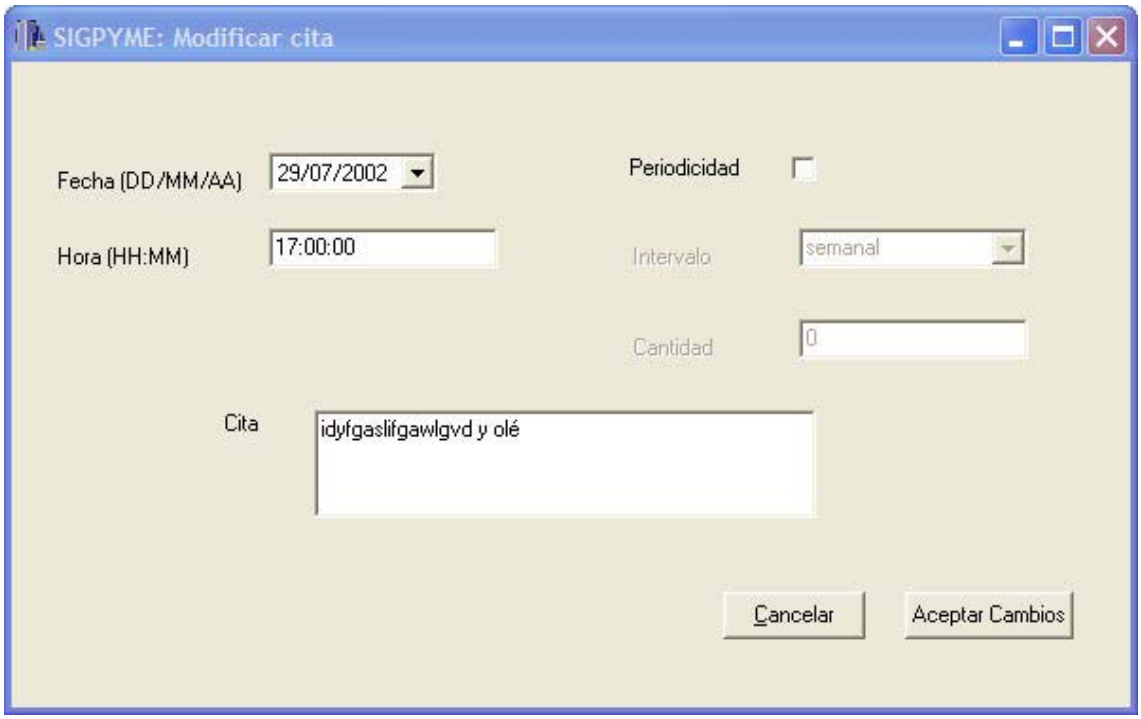

Fig. 3.8: Formulario fmodificar_cita.

Observamos que se disponen en el formulario los componentes necesarios para modificar todos los campos de cualquier cita: Fecha, Hora, Periodicidad, Intervalo y Cantidad. Una vez hechos los cambios pertinentes en la cita, el usuario ha de pulsar el botón "Aceptar cambios" para que dichas modificaciones se guarden en la base de datos, si se arrepiente puede pulsar el botón "Cancelar" en su lugar.

Si el campo periodicidad no está activado, los campos Intervalo y Cantidad no se podrán editar. Se observará que sus respectivas etiquetas están en gris y además no se

puede escribir en los recuadros de texto. En el mismo momento en que se active la casilla de verificación del campo "Periodicidad", "Intervalo" y "Cantidad" pasarán a ser modificables, hecho que observará el usuario porque las etiquetas se escribirán en color negro.

Pasemos a hablar del código:

Función FormActivate.

Esta función se ejecuta cuando se activa el formulario. Se puede ver en el código que lo primero que se hace es rellenar el menú desplegable (*ComboBox*) del campo intervalo. Para ello se vacían los elementos del Combo, y se vuelca en la lista de elementos el contenido de la tabla Intervalo con el siguiente bucle:

```
for (int i=0 ; i < DataModuleBBDD->TablaIntervalo->RecordCount ; i++)
{
        ComboBoxIntervalo->Items->Add( DataModuleBBDD->TablaIntervalo-> \
                                        FieldByName ("Intervalo")->Value );
        DataModuleBBDD->TablaIntervalo->Next();
}
```

Más tarde comienza un tratamiento del registro actual de la tabla Agenda. Hay que observar que este formulario hace la modificación del registro actual de la tabla, por ello hay que asegurarse de que antes de llamar a este formulario hayamos situado la tabla en el registro que queremos modificar, es decir, hacer activo el registro que queremos tratar.

Lo que se hace es volcar en los controles de visualización del formulario los datos del registro actual. Se muestra la fecha (con el control *DateTimePicker1*), la hora (con *EditHora*), la cita (con *MemoCita*), la periodicidad (con la casilla de verificación *CheckPeriodicidad*), y si procede, el Intervalo (con *ComboBoxIntervalo*) y la cantidad (con *EdiCantidad*).

Para controlar si procede o no mostrar, el intervalo y la cantidad se utiliza el siguiente if:

```
if(DataModuleBBDD->TablaAgenda->FieldByName("Periodicidad")->AsBoolean==True)
```

Vemos, que, en definitiva, lo que se consigue con esta función es mostrar en el formulario los datos del registro actual.

Función CheckPeriodicidadClick.

Esta función actúa cuando se hace clic en la casilla de verificación de Periodicidad. Mediante un *if*, se encarga de revisar si está activada o no y actuar en consecuencia:

- Si está activado, habilita los controles referentes al Intervalo (LabelIntervalo y ComboBoxIntervalo) y a la cantidad (LabelCantidad y EditCantidad).
- Si está desactivado, deshabilita los mismo controles.

Al deshabilitar los controles, el usuario lo nota en que el texto de las etiquetas se sombrea en lugar de aparecer en color negro.

En el código vemos el if al que hacía referencia antes:

```
if (CheckPeriodicidad->Checked)
```

Función BotonAceptarCambiosClick

Esta función se ejecuta cuando se pulsa el botón de "Aceptar Cambios". Al inicio de la función, la tabla se activa, se refresca y se edita el registro actual. El resto de la función, lo único que hace es volcar el contenido de los controles del formulario a la base de datos, teniendo en cuenta que el campo Periodicidad esté activado o no con un *if* similar al anterior.

Si el campo "Periodicidad" está activado, se procede a guardar en la base de datos los valores que se hallan en el formulario, como se puede observar en el código:

```
DataModuleBBDD->TablaAgenda->FieldByName("Periodicidad")->Value = True;
DataModuleBBDD->TablaAgenda->FieldByName("IDIntervalo")->Value = \
                                        (ComboBoxIntervalo->ItemIndex) + 1;
DataModuleBBDD->TablaAgenda->FieldByName("Cantidad")->Value = \
                                          StrToInt(EditCantidad->Text);
```

Vemos que los términos de la parte derecha de las igualdades hacen referencia a los controles del formulario (ComboBoxintervalo y EditCantidad).

Si el campo "Periodicidad" se encuentra desactivado, vemos que el campo "IDIntervalo" se rellena con un 1 (por poner algo) y el campo cantidad se rellena con un cero:

```
DataModuleBBDD->TablaAgenda->FieldByName("Periodicidad")->Value = False;
DataModuleBBDD->TablaAgenda->FieldByName("IDIntervalo")->Value = 1;
DataModuleBBDD->TablaAgenda->FieldByName("Cantidad")->Value = 0;
```

Para finalizar, se hace un Post, que sirve para volcar los cambios a disco, de modo que realmente se lleven a cabo. Luego se refresca la tabla y se esconde el formulario.

Componentes usados:

- Label1, Label2, Label3, Label7, LabelIntervalo, LabelCantidad. Son todas las etiquetas de texto que aparecen en el formulario.
- EditHora, EditCantidad. Son campos de edición, para poder modificar los campos Hora y Cantidad de la base de datos.
- MemoCita, sirve para editar el campo cita, que es un campo de tipo Memo, y por lo tanto necesita un control TMemo para poder ser editado.

- BotonAceptar Cambios, es el botón que ha de pulsar el usuario para guardar los cambios que haya hecho en la cita que está modificando.
- CheckPeriodicidad, es la casilla de verificación correspondiente al campo Periodicidad.
- Button1, es el botón "Cancelar", sirve para cancelarlas modificaciones y dejar la cita tal y como estaba.
- ComboBoxIntervalo, es el menú desplegable correspondiente con el campo "Intervalo" y que refleja los posibles valores de esta opción:semanal, mensual, trimestral, semestral, anual.
- DateTimePicker1, es el control que facilita la elección de una fecha. Si se hace click en la parte derecha del recuadro, se despliega un calendario para la elección de fechas.

3.7. Formulario fmodificar_tabla_clas

El aspecto de este formulario es el siguiente:

Fig. 3.9: Formulario fmodificar_tabla_clas

Este formulario sólo sirve para editar y gestionar los registros de las distintas tablas de clasificación. Existen tres tablas de clasificación: la 1, la 2 y la 3. Es posible modificar cada una de las tablas con el mismo formulario. Para ello, lo que se hace es

cambiar la propiedad *DataSource* del control *DBGrid* cuando se va a llamar a este formulario. Esto se puede ver en las funciones que actúan cuando se hace clic en los botones "Clasificacion 1", "Clasificacion 2" y "Clasificacion 3" del formulario principal. He aquí dichas funciones:

```
void __fastcall Tfprincipal::BotonClasificacion1Click(TObject *Sender)
{
    DataModuleBBDD->TablaClas->Active = False;
    fmodificar_tabla_clas->Caption = "SIGPYME: Modificar tabla de clasificaciones 1";
    DataModuleBBDD->TablaClas->TableName = "clas1.db";
    DataModuleBBDD->TablaClas->Active = True;
    fmodificar_tabla_clas->Show();
}

//---------------------------------------------------------------------------

void __fastcall Tfprincipal::BotonClasificacion2Click(TObject *Sender)
{
    DataModuleBBDD->TablaClas->Active = False;
    fmodificar_tabla_clas->Caption = "SIGPYME: Modificar tabla de clasificaciones 2";
    DataModuleBBDD->TablaClas->TableName = "clas2.db";
    DataModuleBBDD->TablaClas->Active = True;
    fmodificar_tabla_clas->Show();
}

//---------------------------------------------------------------------------

void __fastcall Tfprincipal::BotonClasificacion3Click(TObject *Sender)
{
    DataModuleBBDD->TablaClas->Active = False;
    fmodificar_tabla_clas->Caption = "SIGPYME: Modificar tabla de clasificaciones 3";
    DataModuleBBDD->TablaClas->TableName = "clas3.db";
    DataModuleBBDD->TablaClas->Active = True;
    fmodificar_tabla_clas->Show();
}
```

Vemos que en las tres funciones lo que se hace en primer lugar es desactivar la tabla de ayuda *TablaClas*. Luego se cambia la propiedad *Caption* del formulario *fmodificar_tabla_clas*, según qué tabla de clasificación se vaya a modificar. Posteriormente, se modifica la propiedad *TableName* de *TablaClas*, para establecer la tabla sobre la que se va a trabajar. Por último, se muestra el formulario *fmodificar_tabla_clas*.

Vemos que en el formulario hay botones que permiten hacer una gestión completa de registros: Borrar, Guardar, Crear, etc... Otros botones, permiten moverse por los registros de las bases de datos: "Anterior", "Primero", "Siguiente" y "Último". Este juego de botones es bastante común en toda la aplicación, se usa con asiduidad para facilitar el acomodo del usuario a la aplicación y conseguir así que la curva de aprendizaje sea menos pronunciada.

Una vez que el usuario ha hecho todos los cambios que considere oportunos, puede pulsar el botón Aceptar para ocultar el formulario.

En cuanto al código fuente, sólo hay una función que comentar, que es la llamada *Button1Click*, que se ejecuta cuando el usuario pulsa el botón "Aceptar", y lo único que hace es cerrar el formulario.

Componentes usados en el formulario:

- *DBGrid1*, es la rejilla que se puede observar en el formulario. Su propiedad DataSource se modifica antes de abrir el formulario para especificar con qué tabla vamos a trabajar.

- *Button1*, es el botón "Aceptar" que se puede observar en el formulario.

- *dbNavBtnFirst*, *dbNavBtnPrior*, *dbNavBtnNext*, *dbNavBtnLast*, *dbNavBtnCancel*, *dbNavBtnDelete*, *dbNavBtnEdit*, *dbNavBtnNew*, *dbNavBtnSave*, *dbNavBtnRefresh*. Son una serie de botones que funcionan como

si fuera el control *TDBNavigator* que incorpora C++ Builder. Estos componentes son del tipo *TDBNavigationButton*, sólo hay una clase de botones de este tipo, pero se puede cambiar la acción que lleva a cabo, así como el texto que aparece sobre él. Esto componente ha sido encontrado en Internet, en la página www.vclcomponents.com. Para cambiar la acción que realizan los botones hay que modificar la propiedad *DataButtonType* y para cambiar el texto hay que cambiar *Caption*. Para hacer uso de un componente descargado de Internet, hay que indicarle a C++ Builder que incorpore este control al entorno de desarrollo, lo cual se puede hacer con la opción "Install Component" del menú "Component" y siguiendo las instrucciones que se dan. Así se consigue que el componente aparezca en la paleta de componentes "Data Control", como si fuera un componente estándar de C++ Builder. Como curiosidad, cabe decir que este componente está escrito en el lenguaje ObjectPascal del entorno de desarrollo Delphi, que es de Borland, al igual que C++ Builder. De hecho, la gran mayoría de los componentes de C++ Builder derivan de la biblioteca VCL (Visual Components Library) de Borland, escrita en ObjectPascal. Esto explica que el hecho de bajarse un componente de Internet escrito en ObjectPascal e incorporarlo tanto a la aplicación como al entorno de desarrollo no entrañe ninguna complejidad y sea un proceso casi automático.

3.8. Formulario fnavegar_clientes.

En la imagen se muestra el formulario:

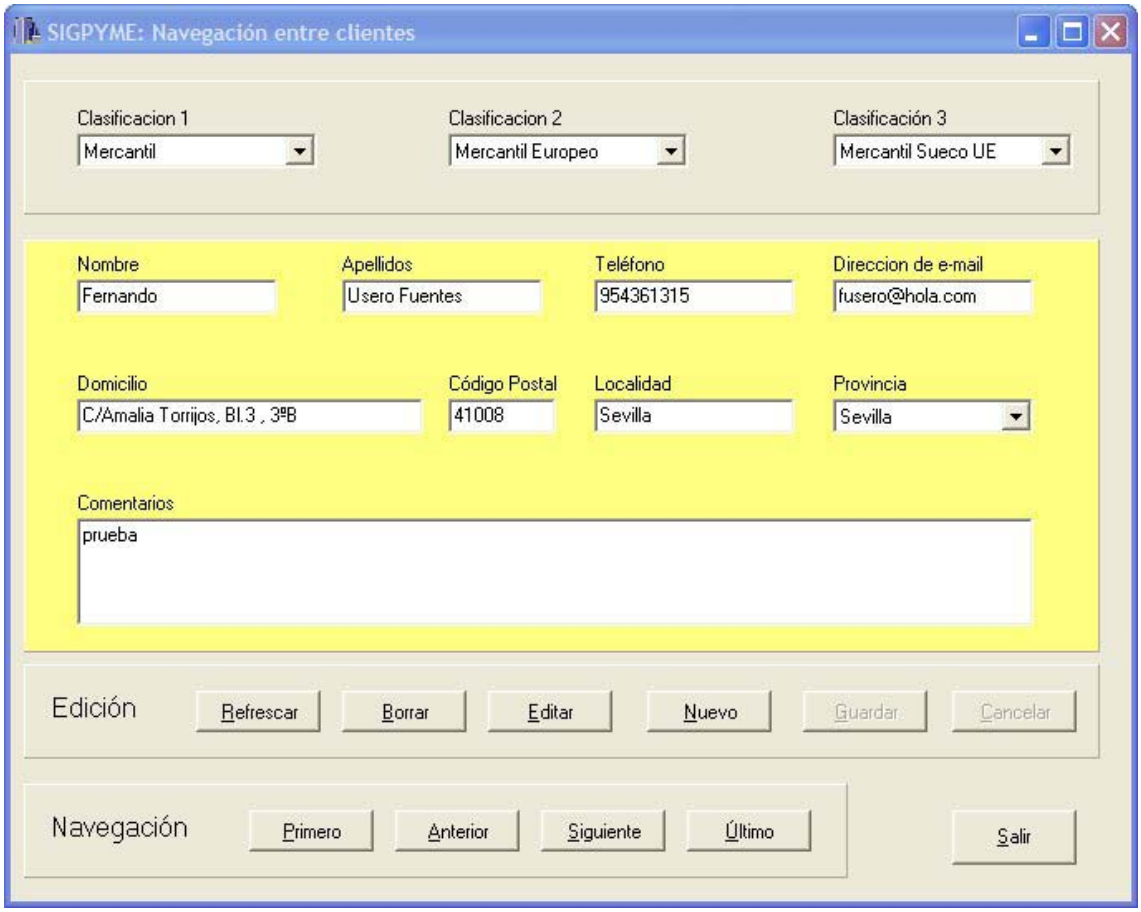

Fig. 3.10: Formulario *fnavegar_clientes*.

En este formulario se muestran lo datos del cliente actual de la base de datos de clientes. Vemos que el formulario está dividido en varias secciones. En la sección superior, se muestran los datos de clasificación del cliente, es decir, los campos "Clasificación 1", "Clasificación 2" y "Clasificación 3".

La segunda sección, muestra los demás datos relacionados con el cliente. Esta sección aparece con un color distinto, dependiendo del campo Clasificación 1 de cada registro. Así todos los registros con el mismo campo Clasificación 1 aparecerán con el mismo color. El programa usa tantos colores distintos como registros haya en el campo Clasificación 1. Para generar los colores se pueden usar dos técnicas: geométrica o aritmética:

- Geométrica: divide el espectro de colores en n colores, calculando la raíz n-ésima. Este método se basa en que los colores del espectro van desde el 0 hasta el FFFFFF y si se hace la raíz cúbica se obtienen estos colores:
  - 0000FF, rojo puro.

- 00FF00, verde puro.
- FF0000, azul puro.

Se puede pensar que haciendo la raíz n-ésima para los n colores se pueden obtener buenos resultados. De hecho es así, cuando el número de colores a usar no es demasiado alto.

- Aritmética: divide el espectro de colores en n partes iguales. Este método es recomendable usarlo cuando hay un mayor número de registros en la tabla de clasificación 1.

El usuario tiene la posibilidad de cambiar el algoritmo de elección de colores en la ventana de configuración cuando quiera. Puede ser que una elección no funcione bien para un determinado de colores, puede probar con la otra elección a ver si le resulta mejor.

Después de la segunda sección de datos, se encuentra la sección de edición, en la que, como su propio nombre indica se puede editar las bases de datos con las operaciones usuales: refrescar la tabla, borrar un registro, editar un registro, crear uno nuevo, guardar los cambios o cancelarlos.

En la parte inferior de la pantalla tenemos los controles de navegación, son unos botones que nos permiten movernos entre los distintos registros: ir al primero, al anterior, al siguiente y al último.

Pasemos ya a comentar el código de la aplicación:

Función BotonSalirClick.

Se ejecuta cuando se llama a la función salir. Lo único que hace es llamar al método Hide del formulario para esconderlo.

Función FormActivate.

Lo primero que se hace es activar y refrescar todas las tablas que se van a usar: *TablaClas1*, *TablaClas2*, *TablaClas3*, *TablaProv* y *TablaClientes*.

Luego se puede observar un if, que se usa para ver si la asignación de colores que reside en la configuración de la aplicación es geométrica o aritmética. Se basa en la propiedad GOpcion3 del formulario fprincipal:

```
if (fprincipal->GOpcion3 == true )
```

Por último, se asigna al componente Panel (en el que se encuentran el grueso de los datos de usuario) el color que le corresponda de entre los que se acaban de calcular.

3.9. Formulario fprincipal.

Este formulario puede tener dos aspectos distintos, dependiendo de si en el instante actual se trabaja con la agenda o con los clientes. En el caso de estar usando la agenda se puede observar esto:

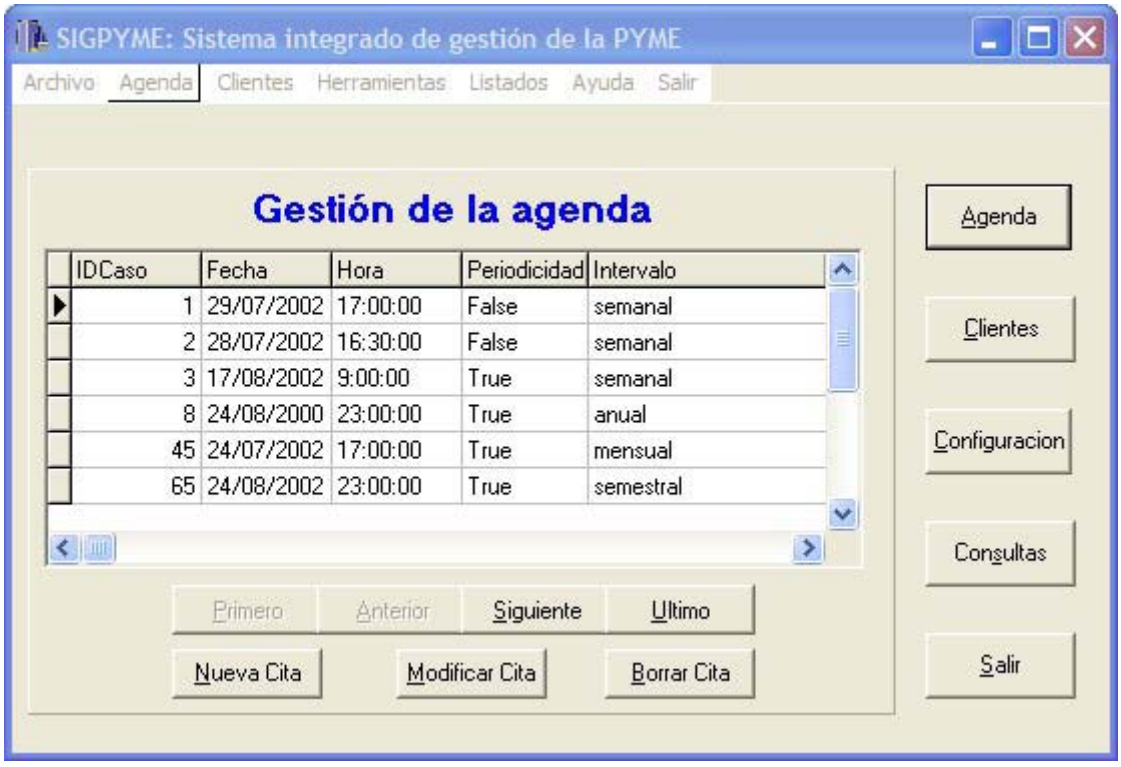

Fig. 3.11: Formulario fprincipal cuando se está trabajando con la agenda.

Si estamos gestionando clientes, entonces veremos algo así:

Fig. 3.12: Formulario fprincipal cuando se está trabajando con clientes.

Los dos aspectos del formulario se obtienen pulsando el botón "Agenda" y el botón "Clientes". También pulsando en el menú "Herramientas", en las opciones "Agenda" y "Clientes".

En la parte de la derecha de la ventana, se disponen cinco botones que siempre están presentes, independientemente de que estemos trabajando con la agenda o con los clientes. Dos de ellos ya se han comentado: "Agenda" y "Clientes", los otros tres son:

- Configuración, para acceder a la ventana de configuración de SIGPYME.
- Consultas, sirve para visualizar la ventana de consultas.
- Salir, para salir de la aplicación.

La otra parte del formulario *fprincipal* que no cambia nunca es la del menú, que está compuesto por las siguientes opciones:

- "Archivo", sólo dispone en su menú desplegable de la opción "Salir".
- "Agenda", tiene todas las opciones correspondientes a la "agenda": "Nueva Cita", "Modificar Cita", "Borrar Cita", "Citas de hoy" y "Buscar Cita".
- "Clientes", que contiene opciones para modificar las tablas Clasificación 1, Clasificación 2 y Clasificación 3.
- "Herramientas", tiene las opciones:
    - Agenda, para mostrar la rejilla de gestión de agenda.
    - Clientes, para hacer la gestión de clientes.
    - Configuración, para ir a las opciones de configuración.
    - Consultas, para hacer búsquedas.
- "Listados", sólo se contemplan dos opciones: listado de todos los clientes y de la agenda completa. Si el usuario desease obtener listados de otro conjunto de registros, lo que debería hacer es ir a consultas, realizar una búsqueda con los registros oportunos y, finalmente , darle al botón "Imprimir".
- "Ayuda", contiene la opción "Acerca de".
- Salir, sirve para salir de la aplicación.

Analicemos ahora el caso en que estamos trabajando con la parte de "Agenda". Podemos observar una rejilla con los datos de la agenda, así se permite un acceso rápido a los registros y sus campos. Por otra parte observamos los botones de navegación que son tan frecuentes en SIGPYME, con los que el usuario se familiarizará muy rápido. Sirven para ir hasta el primer registro, hasta el último, al siguiente y al anterior. El resto de los botones permiten realizar las opciones más frecuentes en bases de datos: Añadir una cita, modificarla y borrarla.

Veamos ahora qué se encuentra el usuario cuando está realizando la gestión de clientes. Verá una rejilla, con la misma utilidad que en el caso de la agenda. Los botones de navegación (Primero, Anterior, Siguiente y Último) también están a disposición del usuario. Debajo de éstos podemos ver el botón "Navegación clientes", que da paso al formulario *fnavegar_clientes* comentado en el punto anterior de este capítulo, y que da toda las posibilidades al usuario en lo que se refiere a la gestión de clientes. Por último, queda reseñar que a la derecha de la rejilla hay tres botones: "Clasificación 1",

"Clasificación 2" y "Clasificación 3". Dan la posibilidad de gestionar las respectivas tablas de nombre igual al de los botones.

Comentemos el código:

Función BotonAgendaClick.

Actúa cuando pulsa el botón "Agenda". Lo que hace es hacer no visible el panel de Clientes y hacer visible el panel de la Agenda. Así se consigue el efecto que tiene SIGPYME de tener dos aspectos distintos: con dos paneles superpuestos y haciendo visible el que nos interese en cada momento.

Función Agenda2Click.

Esta función corre cuando se selecciona la opción "Agenda" del menú "Herramientas". Hace exactamente lo mismo que la función anterior.

Función Salir1Click.

Sirve para salir de la aplicación, actúa cuando se pulsa la opción "Salir" dentro del menú "Archivo".

Función Acercade1Click.

Muestra el formulario *facerca_de* cuando se pulsa el botón "Acerca de" del menú "Ayuda".

Función BotonSalirClick.

Sirve para cerrar la aplicación cuando se pulsa el botón "Salir".

Función BotonClientesClick.

Sirve para cambiar el aspecto del formulario y pasar a trabajar con los clientes. El código que contiene lo único que hace es ocultar el panel de la agenda y mostrar el de clientes. Se ejecuta la función cuando se pulsa el botón "Clientes".

Función Configuración1Click.

Sirve para mostrar el formulario de configuración de SIGPYME. Responde al evento de pulsar en la opción "Configuración" del menú "Herramientas".

Función BotonModificarCitaClick.

Esta función muestra el formulario para modificar citas cuando se hace clic en el botón "Modificar Cita". Sólo lo muestra cuando hay algún registro en la tabla de la agenda, y, para ello usa un `if` en el que se llama a la propiedad *RecordCount*, que contiene el número de registros en la tabla:

```
if (DataModuleBBDD->TablaAgenda->RecordCount)
```

Función BotonBorrarCitaClick.

Esta función responde al evento de hacer clic en el botón "Borrar Cita". Ya se comentó en el apartado del formulario *fbuscar_cita*, por lo que no se hablará de ella de nuevo.

Función DBGridAgendaEditButtonClick.

Actúa cuando se hace clic en el DBGridAgenda en el campo Cita, entonces sale un botón con puntos suspensivos, si se hace clic sobre este botón, se obtiene una pequeña ventana que sirve para editar el contenido del campo cita. La ventana que se obtiene es la correspondiente al formulario *feditar_campo_cita*.

Función FormCreate.

Se ejecuta sólo una vez, cuando la el formulario *fprincipal* se crea, que es sólo una vez. Es distinta la creación de la activación. Esta última ocurre cada vez que el formulario se hace la ventana activa. Sólo se hace una cosa dentro de esta función: poner la variable *primeravez* a **false**, ya que es el momento en el que se crea el formulario y es la primera vez que le formulario ejecuta algo de código correspondiente a él. A continuación veremos el uso que se le da a esta variable.

Función FormActivate.

Esta función se ejecuta cada vez que el formulario se activa. Al principio de la función hay un if que abarca todo el código de la misma. O sea, si no se cumple el if, la función no ejecuta nada de código. El if es el siguiente:

```
if (primeravez)
```

Esto quiere decir que el código de esta función sólo se ejecuta si *primeravez* es **true**. Esto ocurre cuando se ejecutó *FormCreate* antes, porque después *primeravez* ya siempre tendrá el valor **false**, ya que es una de las cosas se hace dentro del `if`. Veamos todo el código de dentro de la función:

```
if (primeravez)
{
 IniciarDatos(this);
 primeravez = false;
 reg = new TRegistry;
 reg->RootKey = HKEY_CURRENT_USER;
 reg>OpenKey("\\Software\\Microsoft\\Windows\\CurrentVersion\\Run", \
                                                                    true);
 if ( !reg->ValueExists("citas") )
   ShowMessage("No existe la clave citas. Error fatal");
 else
 {
  if (reg->ReadString("citas") == "") //si no tiene información
  {
    ShowMessage("Ésta es la primera vez que se ejecuta SIGPYME");
```

```
    ShowMessage("A partir de ahora windows le avisará de sus citas \
                                                        al iniciarse.");
    PathApp = ExtractFilePath(Application->ExeName);
    AnsiString PathCitas = PathApp + "citas\\citas.exe";
    reg->WriteString("citas",PathCitas);
   }
  }
  reg->CloseKey();
  delete reg;
}
```

Vemos que dentro del if, también se llama a la función IniciarDatos, que se comentará más tarde. El resto del código comprueba con la ayuda de la variable *reg*, si ya está instalado el programa de aviso de citas. Para ello revisa en el registro de windows (accesible mediante el programa regedit de windows) si el valor citas del path HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run del registro tiene información o no. Este valor lo crea la instalación de SIGPYME y lo deja sin información. De esta manera, cuando SIGPYME arranca comprueba que si esto es así y en caso afirmativo se da cuenta de que es la primera vez que arranca SIGPYME, y, entonces, como información del valor citas introduce la ruta del programa citas. Así se consigue que este programa se ejecute siempre en el inicio de windows, ya que es la particularidad que tiene el path del registro que estamos usando. Para más información sobre el uso del registro de windows, recurrir a la sección de trucos al final de este capítulo 3.

Función IniciarDatos.

Se le llama desde FormActivate. Lo primero que hace es dar el valor necesario a PathApp, que es una variable global que contiene el path del sistema en el que está corriendo la aplicación. Esta variable se usa para muchas cosas a lo largo de todo el programa.

Más tarde se empieza a tratar los valores del registro de windows. Ya sabemos que SIGPYME guarda en el registro las opciones de configuración. Como ejemplo, vamos a ver lo que se hace con la primera. El código correspondiente es:

```
if ( !reg->ValueExists("Opcion1") ) //Si no existe en el registro
{
  reg->WriteBool("Opcion1",false); //La creo
  GOpcion1 = false; //Le doy el mismo valor a la correspondiente
                                    //variable global
}
else
{
  if (primeravez)
    GOpcion1 = reg->ReadBool("Opcion1");
  else
    reg->WriteBool("Opcion1",GOpcion1);
}
```

Antes de este código ya se ha establecido el path en el que SIGPYME guarda sus opciones de configuración. En el primer if comprobamos si existe el valor *Opcion1* en el registro. Si no existe se crea y se le da un valor por defecto. El usuario podrá modificarlo más tarde. Si ya existía el valor Opcion1 en el registro, hay dos opciones:

- Si es la primera vez que se ejecuta IniciarDatos (o sea, viene de la llamada de FormActivate), se guarda la información del valor Opcion1 del registro en la variable global de SIGPYME *GOpcion1*, que luego se usará en otras partes, como por ejemplo en el formulario *fconfiguracion*. Tiene sentido, ya que es la primera vez que se ejecuta este función, estamos en la parte inicial del programa, y es necesario dar valores a las variables globales de configuración que se usarán por todo el programa.
- Si no es la primera vez, es porque se ha llamado a la función probablemente desde el formulario fconfiguracion, y lo que se pretende es actualizar los valores del registro.

Este mismo tratamiento se hace con las cuatro opciones de configuración. Después de esto, viene el control de las bases de datos, para saber si existen o no, y también se controlan los directorios que necesita SIGPYME. Veamos un ejemplo de control de existencia de un directorio:

```
AnsiString DirectorioClientes = GOpcion2 + "clientes";
if ( !DirectoryExists(DirectorioClientes) )
{
  ShowMessage("No existe el directorio Clientes, procedo a crearlo");
  if (!CreateDir(DirectorioClientes))
    throw Exception("No puedo crear el directorio clientes necesario \
                                                    para la ejecucion");
}
```

Primero se construye una variable que tenga la ruta completa del directorio a examinar. En este caso se quiere ver si existe el directorio donde residen las tablas de clientes. Se construye dicha ruta como la concatenación de la opción de configuración 2 (que contiene la ruta donde se encuentran las bases de datos) y el nombre del directorio ("clientes"). Después con el primer if se comprueba si existe o no el directorio. Si la comprobación es positiva se pasa a otra tarea. Si no es así, se muestra un mensaje de aviso al usuario y se crea el directorio en cuestión (con la función CreateDir). Sino se puede crear dicho directorio se lanza una excepción.

Veamos ahora cómo se hace la comprobación de la existencia de una tabla, por ejemplo la de clientes. El código correspondiente sería:

```
AnsiString PathArchivo = GOpcion2 + "clientes" + "\\clientes.db";
if ( !FileExists(PathArchivo) )
{
  ShowMessage("La tabla clientes.db no existe, procedo a su creación");
  DataModuleBBDD->TablaClientes->FieldDefs->Clear();

  //Creamos todos los campos
  NuevoCampo = DataModuleBBDD->TablaClientes->FieldDefs->       \
                                              AddFieldDef();
  NuevoCampo->Name = "IDCliente";
```

```
  NuevoCampo->DataType = ftAutoInc;

  NuevoCampo = DataModuleBBDD->TablaClientes->FieldDefs->      \
                  AddFieldDef();
  NuevoCampo->Name = "IDClas1";
  NuevoCampo->DataType = ftInteger;

.....

  //Creamos los índices
  DataModuleBBDD->TablaClientes->IndexDefs->Clear();
  DataModuleBBDD->TablaClientes->IndexDefs->Add("","IDCliente",  \
                  TIndexOptions() <<ixPrimary << ixUnique);

  //Creamos la tabla
  DataModuleBBDD->TablaClientes->CreateTable();
}
```

Donde van los puntos suspensivos es porque no es necesario mostrar más ejemplos de creación de campos, pero sí es interesante lo que viene después.

Comencemos a hablar sobre el código. Primero se construye una variable con el path completo del archivo que se quiera tratar. En el `if` se comprueba si no existe el fichero, en cuyo caso entra dentro. Se muestra un mensaje al usuario advirtiéndole sobre la inminente creación de la tabla.

Para crear la tabla en el disco duro disponemos del componente TablaClientes. Lo primero que hacemos es vaciar la lista de campos. Luego mediante el método AddField obtenemos el puntero a un nuevo campo, al que llamamos NuevoCampo. Haciendo uso de los métodos de NuevoCampo, podemos asignarle un nombre al campo, especificar el tipo del campo y el tamaño.

Más tarde se crean los índices de la tabla, primero se vacía la lista y después se añade el campo que hace de índice. Por último, mediante el método CreateTable, llevamos a cabo la creación efectiva del archivo en disco duro.

Este mismo proceso se hace con las siete tablas que se usan en SIGPYME. Al final de la creación de todas las tablas de clientes, se activan dichas tablas y se conecta la base de datos. Lo mismo se hace con la agenda.

Función BotonConfiguracionClick.

Actúa cuando se pulsa el botón de Configuración. Sólo se encarga de mostrar el formulario *fconfiguracion*.

Función BotonNavegarClick.

Actúa cuando se pulsa el botón de NavegarClientes. Si no hubiera registros en la tabla de clientes se avisa de ello al usuario.

Componentes usados:

- *BotonAgenda*, *BotonSalir*, *BotonClientes*, *BotonNuevaCita*, *BotonModificarCita*, *BotonClasificacion1*, *BotonClasificacion2*, *BotonClasificacion3*, *BotonConfiguracion*, *BotonNavegar* y *BotonConsulta*. Son los distintos botones que hay a lo largo de todo el formulario. Los nombres ayudan a identificarlos.
- *MainMenu1*, es el menú que se puede observar en la parte superior del formulario.
- *Agenda1*, *Edicion1*, *Herramientas1*, *Ayuda1*, Agenda2, *Clientes1*, *Acercade1*, *Salir1*, *Configuracion1*, *Listados*, *Todoslosclientes1*, *Todaslascitas1*, *NuevaCita1*, *Modificarcita1*, *BorrarCita1*, *Citasparahoy2*, *BuscarCita1*, *Clientes2*, *Navegacin1*, *Clasificacin11*, *Clasificacin21*, *Clasificacin31*, *Consultas1*. Son todos los elementos del menú principal y de todos los submenús.
- *PanelAgenda* y *PanelClientes*. Son los paneles con los que se juega (haciéndolos visibles o no) para cambiar el aspecto de la ventana principal.
- *DBgridAgenda* y *DBGridClientes.* Son las dos rejillas: la de la agenda y la de clientes.
- *GroupBox2*, es el componente que agrupa los botones de las tres tablas de clasificación.

- *LabelClientes* y *LabelAgenda*. Son las dos etiquetas que hay en cada panel con el texto: “Gestión de clientes” y “Gestión de la agenda”, respectivamente.
- *dbNavBtnFirst*, *dbNavBtnPrior*, *dbNavBtnNext* , *dbNavBtnLast*, *dbNavBtnFirst1*, *dbNavBtnPrior2*, *dbNavBtnNext3* y *dbNavBtnLast4*. Son los 8 botones de la clase TDBNavigationButton de la que ya se ha hablado aquí y que sirven para navegar por la tabla.

3.10. Formulario finforme_citas_hoy.

Este formulario en fase de diseño, tiene esta forma:

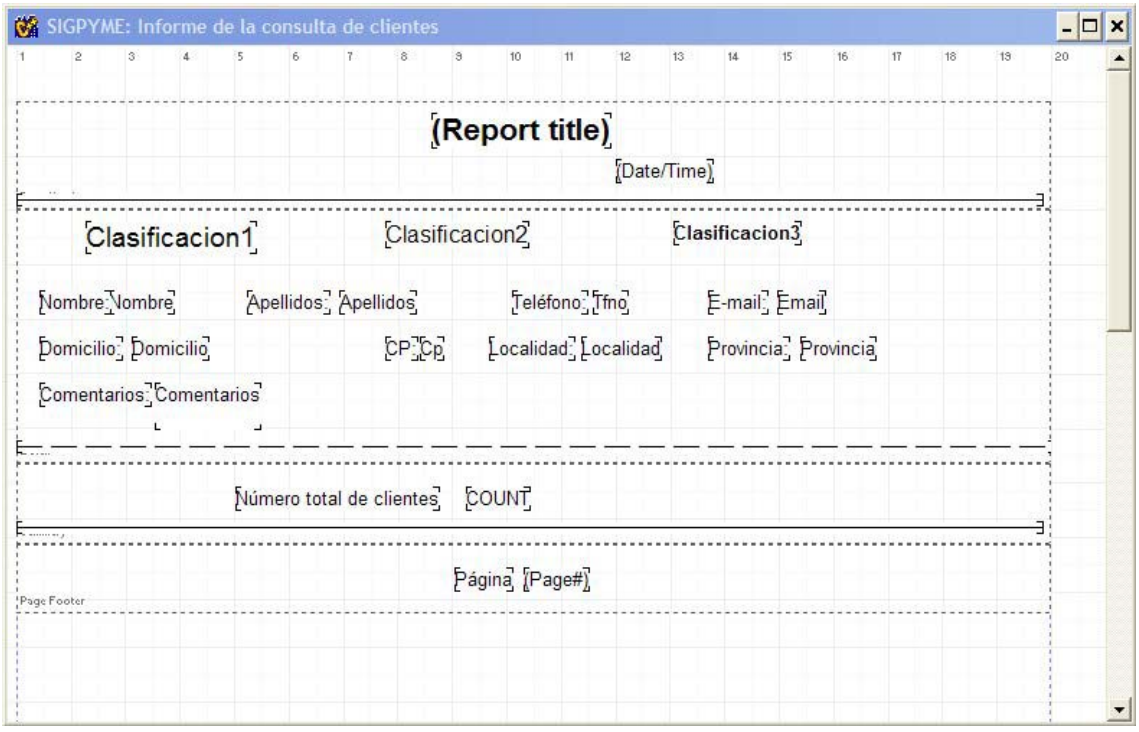

Fig. 3.13: Formulario informe_consulta_clientes en fase de diseño

En ejecución tiene esta forma:

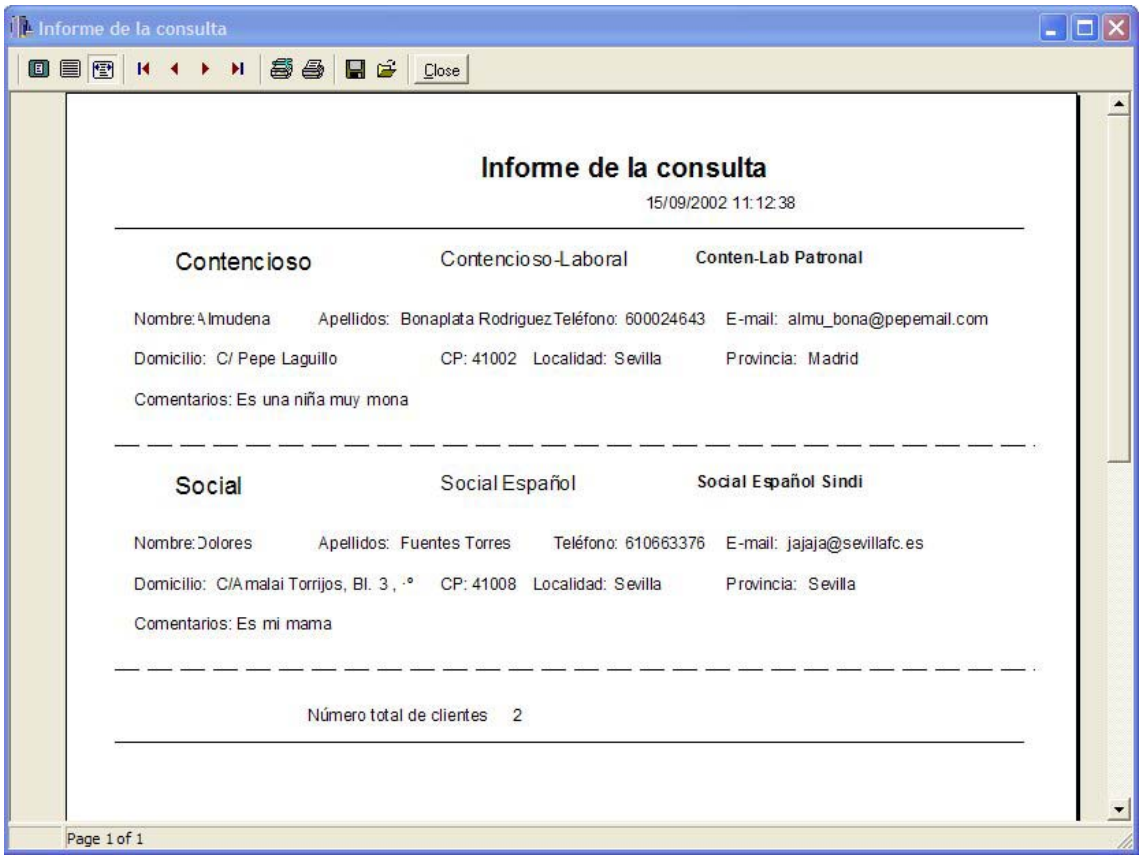

Fig. 3.14: Formulario finfome_consulta_clientes en tiempo de ejecución

El formulario se lanza cuando se pulsa el botón "Imprimir" de la ventana de consultas y se está realizando una búsqueda de clientes. Los demás formulario de informes de SIGPYME no se van a comentr pues son todos exactamente iguales. Lo único que cambia es el número de componentes de cada tipo y las propiedades.

En cuanto al código no hay nada que reseñar, ya que es casi inexistente. Esto ocurre con todos los informes.

Pasemos a comentar los componentes usados:

- *QuickRep1*, es el componente principal del formulario, es el que le da el aspecto de informe.
- *PageHeaderBand1*, es el componente que hace que la página tenga una cabecera. En él se incluyen el título del formulario y la fecha y hora en que se realizó el informe.

- *DetailBand1*, es el cuerpo del informe: donde van los datos de cada registro que se imprime en el formulario. Contiene varios controles *TQRLabel* (para poner etiquetas) y *TQRDBText* (para poner los campos de las tablas).
- *SummaryBand1*, es una parte del informe con características de sumario, y que sólo aparece en la última página del informe. Contiene un *TQRLabel* (con el texto "Número total de clientes") y un *TQRExpr*, que da el número total de registros presentes en el informe, esto se consigue haciendo que su propiedad *Expresion* contenga el valor **COUNT**.
- *PageFooterBand1*, es el pie de página del informe. Contiene un *TQRLabel* (con el texto "Página") y un *TQRSysData*, que muestra el número de página del informe debido a que tiene su propiedad *Data* con el valor **qrsPageNumber**.
- *QRLabel6*, *QRLabel10*, *QRLabel5*, *QRLabel9*, *QRLabel8*, *QRLabel4*, *QRLabel3*, *QRLabel7*, *QRLabel11*, *QRLabel1*, *QRLabel2*. Son todas las etiquetas presentes en el informe.
- *QRDBText*, *QRDBText2*, *QRDBText3*, *QRDBText7*, *QRDBText11*, *QRDBText10*, *QRDBText6*, *QRDBText9*, *QRDBText5*, *QRDBText4*, *QRDBText8* y *QRDBText12*. Son todos los componentes que enlazan con un campo de la tabla y muestran su contenido.
- *QRShape1*, *QRShape2* y *QRShape3*. Son todos los componentes que muestran una línea horizontal (continua o discontinua) en el informe.
- *QRExpr1*, sirve para mostrar una expresión. Se utiliza para mostrar el número total de páginas.
- *QRSysData1*, *QRSysData2* y *QRSysData3*. Componentes usados par mostrar datos como el número actual de página del informe, o el título del informe.

3.11. Archivo sigpyme.cpp

Este archivo es el código correspondiente al proyecto "sigpyme.bpr". Contiene la función WinMain, que es la primera que se ejecuta de toda la aplicación. Vemos, que lo primero que se hace es declarar una variable denominada *mutex* del tipo *HANDLE*.

Esta variable servirá para implementar un sistema de bloqueo, para conseguir que la aplicación no corra simultáneamente más de una vez. La técnica usada se explica más adelante en este mismo capítulo, en la sección de trucos.

En la función WinMain también se puede apreciar como se crea de forma dinámica el formulario *fsplash* y se muestra. Este formulario muestra una pantalla splash al inicio de la aplicación, a modo de presentación. Para obtener más información sobre esta técnica, recurrir a la sección de trucos.

Por último, vemos también el código generado por C++ Builder, con líneas del tipo: `Application->CreateForm` para crear cada uno de los formularios que componen SIGPYME. El único que no se crea así es *fsplash*.

3.12. Archivo datosBBDD.cpp

Es el archivo que corresponde al componente *DataModuleBBDD*, instanciado de la clase *TDataModule*, que sirve para centralizar la gestión de componentes no visuales de la aplicación.

Veamos el aspecto que tiene DataModuleBBDD:

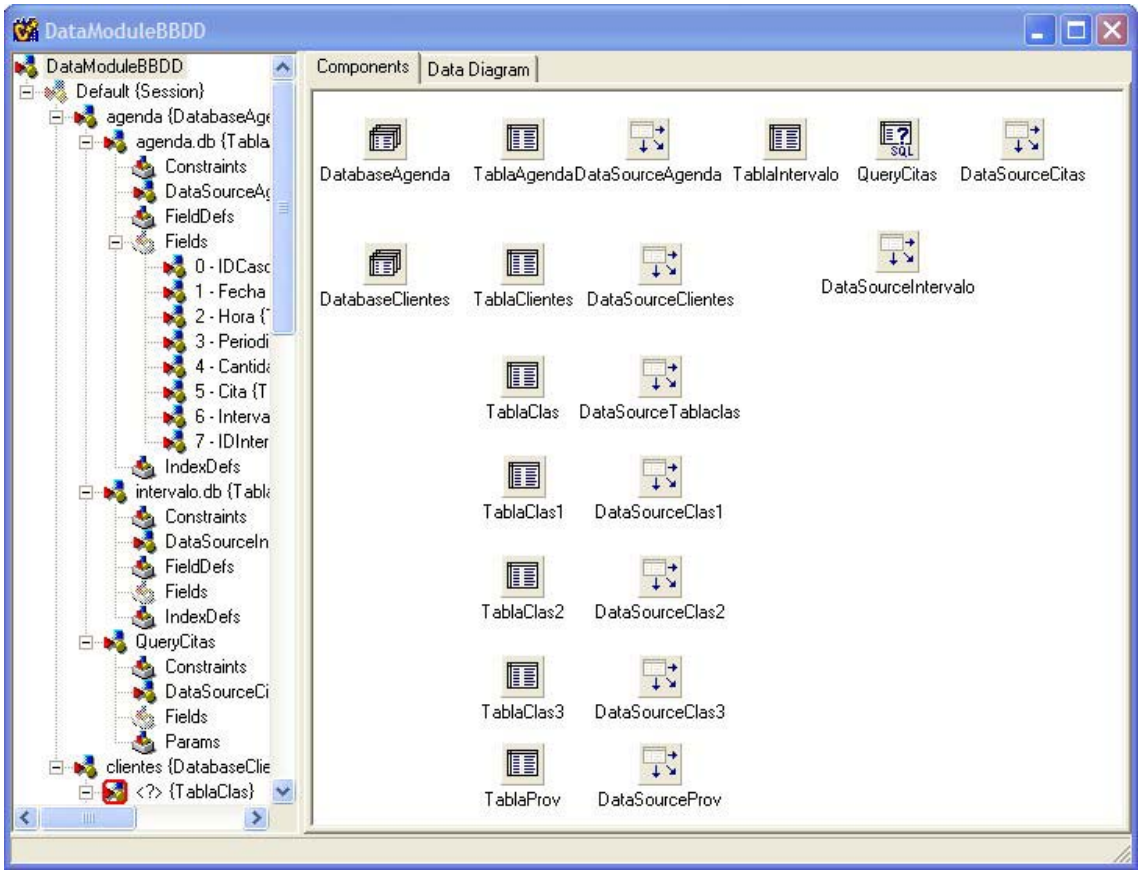

Fig. 3.15: Componente DataModuleBBDD

Un objeto instanciado de TDataModule se usa en una aplicación para proporcionar un lugar para la gestión centralizada de componentes no visuales. Lo más normal es que estos componentes sean de acceso a datos (TSession, TSessioList, TDataBase, TTable, TQuery, TStoredProc, y TBatchMove), pero también pueden ser otros componentes no visuales, como TTimer, TOpenDialog, TImageList o TOleContainer.

En tiempo de diseño un objeto TDataModule proporciona un contenedor visual en el que un desarrollador puede colocar componentes no visuales, fijar sus propiedades y escribir los correspondientes manejadores de eventos. Para crear un modulo de datos en tiempo de diseño, elija la opción “New Data Module” del menú “File”.

En el caso de nuestro módulo de datos, su propio nombre indica que se incluyen en él todos los componentes relacionados con bases de datos de la aplicación

SIGPYME. Para hacer uso de ellos desde cualquier otro formulario hay que anteponer "DataModuleBBDD->" al nombre del componente.

3.13. Trucos.

3.13.1. ¿COMO PREGUNTO AL USUARIO SI REALMENTE QUIERE CERRAR UN FORMULARIO?

PROBLEMA

Para evitar la pérdida de datos, quiero pedir a los usuarios confirmación cuando intenten salir de mi aplicación. Quiero que los usuarios tengan la opción de cancelar la acción y volver al programa. ¿Como puedo añadir esta sencilla característica a mi programa C++ Builder?

TÉCNICA

OnCloseQuery es lo que buscas. El Inspector de Objetos muestra *OnCloseQuery* como una de los eventos disponibles para un formulario. Puedes crear una función *OnCloseQuery* que determine si un formulario se puede cerrar o no. Esta función puede prevenir que un formulario se cierre estableciendo uno de sus argumentos como **false**.

PASOS

Este truco solo describe como usar *OnCloseQuery* para preguntar a tus usuarios si quieren salvar sus datos. El programa realmente no salva ningún dato.

1. Crea un nuevo proyecto utilizando la opción File/New. Pulsa "File/Save project as", nombrando a la unidad como MAINFORM, y al proyecto como "canclose.mak".
2. Sitúa un control *RichEdit* en el formulario y establece su propiedad *Align* a **alClient**. El control *RichEdit* se encuentra en el paleta de componentes Win95.
3. Coloca un componente *MainMenu* en el formulario. Haz doble clic en el componente del menú para lanzar el "Menu Designer". Añade un objeto de

menú *File* y un objeto *Exit* dentro él. Crea una función *OnClick* para el objeto de menú "Exit", y añade este código:

```
void __fastcall TForm1::Exit1Click(TObject *Sender)
{
        Close();
}
```

4. Cierra el "Menu Designer", y selecciona Form1 en el Inspector de objetos. Pulsa la pestaña *Events* del Inspector de objetos, localiza el objeto *OnCloseQuery* y haz doble clic en el. Escribe este código en la función FormCloseQuery que C++ Builder construye en tu lugar:

```
void __fastcall TForm1::Exit1Click(TObject *Sender, bool &CanClose)
{
  //Añade código para comprobar si los cambios de editor se han realizado.
  //Si no hay cambios, establece CanClose para verificar y retroceder.

  int ExitCode = Application->MessageBox("¿Salvar los cambios de editor \
          antes de cerrar?", "Peligro", MB_YESNOCANCEL / MB_ICONWARNING);
  switch(ExitCode)
  {
        case IDCANCEL:
                CanClose = false; // No guarder y no salir.
                break;
        case IDYES:
                CanClose = true; // Añadir código para salvar.
                break;
        case IDNO:
                CanClose = true; // No guardar y salir
                break;
        }
  }
```

5. Compila y prueba el programa.

COMO FUNCIONA

El manejador *OnClick* del objeto de menú "Exit" (paso 3) llama a la función VCL TForm::Close . *Close* no cierra el formulario por su cuenta. Primero pide permiso al manejador *OnCloseQuery*, y este debe aprobar la petición antes de que el programa pueda salir. FormCloseQuery se ejecuta cuando seleccionas "File / Exit" o cuando haces clic en el icono de cerrar en la barra de titulo del programa. El formulario no se cerrará si FormCloseQuery establece que *CanClose* es **False**. El establecer *CanClose* como **True** permite que el programa se cierre, pero es tarea tuya salvar los cambios antes de que FormCloseQuery cierre la aplicación.

NOTA: Observa que sucede si cambias la llamada *Close* en el paso 3, por Application->Terminate(). Terminate provoca un modo suicida al cerrar un programa.

3.13.2. ¿COMO PUEDO MOSTRAR UNA PANTALLA SPLASH CUANDO SE INICIA EL PROGRAMA?

PROBLEMA

Las aplicaciones profesionales despliegan una atractiva pantalla splash cuando se inician. Estas pantallas splash suelen mostrar logos de empresa, avisos de copyright, información sobre el programa, y los términos del registro. ¿Como puedo añadir una pantalla splash a mi aplicación de C++ Builder?

TÉCNICA

Las pantallas splash no son mas que formularios decorados que responden a eventos temporales. La función WinMain puede lanzar la pantalla splash cuando el programa se inicia. La pantalla splash se cierra a sí misma después de un tiempo.

PASOS

1. Crea un nuevo proyecto. Salva la unidad como "mainform.cpp" y salva el proyecto como "splash.mak".
2. Añade un menú a Form1 usando la herramienta *MainMenu*. Haz doble clic en el componente del menú y utiliza el "Menu Designer" para crear un objeto menú *File* y un objeto *Exit* dentro él. Crea un manejador de evento *OnClick* para el objeto *Exit*, e inserta una llamada a *Close* en la nueva función manejador. Cierra el "Menu Designer" cuando hayas acabado.

```
void __fastcall TForm1::Exit1Click(TObject *Sender)
{
        Close();
}
```

3. Añade un segundo formulario al proyecto utilizando "File / New form" y guárdalo como "splashform.cpp". Este formulario, *Form2*, será la pantalla splash.
4. Por omisión, los programas en C++ Builder crean automáticamente sus formularios durante su comienzo. Estos objetos de formulario existen durante toda la vida del programa. Este proceso se llama auto-creación. Es una pérdida de tiempo auto-crear el formulario splash porque solo se mostrará durante un instante. Elige "Option / Project" y borra *Form2* de la lista de auto-creación.
5. Sitúa un control *Image* y un control *Timer* en *Form2*. El control *Image* se encuentra en la pestaña **Additional** de la paleta de componentes. El control *Timer* se puede encontrar en la pestaña *System*. Establece las propiedades como en la tabla siguiente:

Componente	Propiedad	Valor
Form2	BorderIcons	(all false)
	BorderStyle	bsDialog
	FormStyle	fsStayOnTop
	Position	poScreenCenter
Image1	Align	allClient

	Stretch	true
Timer1	Interval	4000

NOTA: El valor de Interval de *Timer1* establece el tiempo del pantallazo en milisegundos. Modifica este valor según tus necesidades.

6. Haz doble clic sobre el componente imagen para activar el "Picture Editor", el cual permite cargar un dibujo dentro de una imagen. Pulsa el botón "Load" y localiza el archivo gráfico que quieras usar. Selecciona este archivo. C++ Builder proporciona una gran colección de archivos gráficos. Se pueden encontrar en el subdirectorio IMAGES / SPLASH de C++ Builder.
7. Crea un manejador *OnTimer* para el control *Timer* en *Form2*. Este manejador cierra la pantalla splash cuando lo indica el temporizador.

```
void __fastcall TForm2::Timer1Timer(TObject *Sender)
{
        Close();
}
```

8. Abre "splashform.h" y añade un prototipo de función CreateParams para la clase *TForm2*.

```
private: // User declarations
      void __fastcall CreareParams(TCreatePArams &Params);
```

9. La mayoría de las pantallas splash no tienen barra de título. La función CreateParams te permite quitar la barra de titulo modificando el estilo de ventana del formulario. Usa el "Editor de Código" para entrar en la función CreateParams dentro de splashform.cpp.

```
void __fastcall TForm2::CreateParams(TCreateParams &Params)
{
        TForm::CreateParams(Params); // Primero llamar a la clase base
        Params.Style = WS_CAPTION; // Después quitar el caption
}
```

NOTA: Los pasos 8 y 9 borran la barra de titulo de la pantalla splash. También se puede borrar estableciendo la propiedad *BorderStyle* del formulario splash como **bsNone**, pero esto podría borrar también el borde del formulario. Muchas pantallas splash, incluyendo la del C++ Builder, utilizan esta técnica de quitar el borde.

10. Ahora la aplicación puede activar la pantalla splash. Abre "splash.cpp", y modifica la función WinMain para abrir la pantalla splash cuando comience el programa. Puede acceder rápidamente a la función WinMain seleccionando View / Project Source en el menú de C++ Builder.

```
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
        try
        {
                Application->Initialize();
                Form2=new TForm2(Application);
                Form2->Show(); //Mostrar pantalla splash ahora.
                Form2->Update(); //Pintar pantalla splash ahora.
                Application->CreateForm(__classid(TForm1), &Form1);
                Application->Run();
        }
        catch (Exception &exception)
        {
                Application->ShowException(&exception);
        }
        return 0;
}
```

11. La función WinMain no puede llamar a `Form2->Show` sin conocer la declaración de clase de *Form2*. Dentro de "splash.cpp" añade un `#include` referente al archivo de cabecera de *TForm2*.

```
#include <vcl\vcl.h>
#pragma hdrstop
#include "splashform.h"
```

12. Alguien tiene que borrar el puntero a *Form2* después de que la pantalla splash se cierre. La forma mas fácil de hacerlo es utilizando el argumento *Action* del manejador *OnClose* del formulario. VCL borrará automáticamente un objeto formulario si el manejador *OnClose* establece *Action* como **caFree**. Crea un manejador *OnClose* para *Form2*, e inserta este código (asegurate de que el manejador *OnClose* es para *Form2*, el formulario de la pantalla splash):

```
void__fastcall TForm2::FormClose(TObject *Sender,TCloseAction &Action)
{
        Action = caFree;
}
```

13. Compila y prueba el programa.

COMO FUNCIONA

C++ Builder genera una función *WinMain* que contiene código para la construcción de los formularios del programa. *WinMain* normalmente llama a Applicatio->Run después de que el formulario haya sido creado. Run muestra el formulario principal, además de hacer una cantidad ingente de de cosas más.

La pantalla splash se puede mostrar creando el puntero a *Form2* y llamando Form2->Show justo antes de que *Form1* sea creado (paso 10). Show muestra la pantalla splash sin modo, lo cual permite al formulario de la aplicación principal aparecer justo después de la pantalla splash. Form2->Update pinta la pantalla splash inmediatamente antes de que se ejecute la siguiente afirmación. El formulario principal aparecerá mientras se muestra la pantalla splash, la cual permanece en primer plano porque se ha establecido su propiedad *FormStyle* como **fsStayOnTop** (paso 5).

La pantalla splash debe decidir cuando cerrarse a sí misma, y realizaa esta tarea esperando que el temporizador expire. *Form2* llama a Close tan pronto como el evento temporizador ocurre (paso 7).

3.13.3. ¿COMO USO EL REGISTRO PARA ALMACENAR INFORMACION SOBRE LA CONFIGURACION DE LA APLICACION?

PROBLEMA

He escrito una aplicacion de un solo formulario. Cada vez que se ejecuta la aplicación, el usuario cambia su tamaño y posición en la pantalla. Se que es muy facil guardar estos cambios en un archivo INI, de forma que al iniciar la aplicación de nuevo, ésta recuerde las preferencias del usuario, pero me gustaría escribir las preferencias en el registro en su lugar. ¿Como puedo hacerlo en C++ Builder?

TÉCNICA

Aquí se muestra como usar la clase TRegistry para almacenar las preferencias. También se mostrará algo sobre cuál es el lugar adecuado para los datos de registro de tu aplicacion.

PASOS

1. Crea una nueva aplicacion. Guarda el formulario como “remembermain.cpp” y el proyecto como “remember.mak”.
2. Cambia el título del formulario a "Remember...".
3. Visualiza “remembermain.h” . Añade la siguiente cabecera de manera que la aplicacion pueda usar el objeto TRegistry:

```
#include <vcl\Forms.hpp>
#include <vcl\Registry.hpp>
```

4. Añade la siguiente declaracion a la sección privada de TForm1:

```
private:// User declarations
        TRegistry *Registry;
```

5. Abre el Inspector de objetos para el formulario *Form1*. Haz doble clic sobre el evento *OnShow*, y añade el código que lee el tamaño y la posición del formulario en el registro:

```
void __fastcall TForm1::FormShow(TObject *Sender)
{
 int iLeft, iTop, iHeight, iWidth;

 //Crea un nuevo objeto del registro
 Registry = new TRegistry;
 //La clave raíz (root key) es HKEY_LOCAL_MACHINE
 Registry->RootKey = HKEY_LOCAL_MACHINE;
 //Abre la llave de registro (crear si no existe)
 Registry->OpenKey("SOFTWARE\\MyCompany\\Remembber", TRUE);
 //Intenta bloquear, si algo va mal, control sera transferido
 //al catch(...)
 try
 {
   //Consigue las entradas de regidtro
   iLeft = Registry->ReadInteger("Left");
   iTop = Registry->ReadInteger("Top");
   iHeight = Registry->ReadInteger("Height");
   iWidth = Registry->ReadInteger("Width");

   //La ejecucion solo logra esto si las llaves de registro existen
   //Posicion y tamaño para
   Left = iLeft;
   Top = iTop;
   Height = iHeight;
   Width = iWidth;
 }
 //Captura errores y los ignora
 //el caso usual es en el que el registro no exista la primera vez
 catch (...)
 {
  }
}
```

6. Haz doble clic en el evento *OnClose*, y añade el código que guarda el tamaño y la posición en el registro.

```
void __fastcall TForm1::FormClose(Tobject *Sender, TCloseAction &Action)
{
        //Escribe el tamaño y la posición actuales en el registro
        Registry->WriteInteger("Left", Left);
        Registry->WriteInteger("Top", Top);
        Registry->WriteInteger("Height", Height);
        Registry->WriteInteger("Width", Width);
        //Libera el objeto de registro
        Registry->Free();
}
```

7. Compila y ejecuta la aplicación. Si la aplicación corre por primera vez desde un entorno de C++ Builder, el depurador se parara en la excepción causada cuando la aplicación no puede encontrar algo en el registro. Si esto ocurre, haz clic en el botón OK, y entonces pulsa F9 para continuar con la ejecución del programa. El mensaje de error no aparecerá si el programa funciona fuera del entorno del C++ Builder.

CÓMO FUNCIONA

Antes de entrar en detalles sobre qué hace el programa, es importante entender que apariencia tiene el registro. Comienza ejecutando tu editor de registro de sistema. En Windows 95 se llama regedit.exe. En Windows NT se llama regedt32.exe.

Como puedes ver, el registro de sistema está compuesto de varios árboles. Hay algunas diferencias entre Windows 95 y Windows NT, pero el árbol en el que estamos interesados, HKEY_LOCAL_MACHINE , es el mismo en ambos sistemas operativos.

Haz doble clic en HKEY_LOCAL_MACHINE en tu editor de registro. Este árbol del registro maneja información que se aplica a la máquina, sin importar los usuarios que hayan abierto sesión en la máquina. La clave en la que está interesada esta

aplicación, llamada SOFTWARE, mantiene la información de configuración de las aplicaciones de la máquina. Abre la clave SOFTWARE haciendo doble clic en ella. Deberías ver que hay una entrada llamada "MyCompany". Ábrela y después abre la clave *Remember* de dentro de ella. Ahora deberías poder ver los elementos de configuración de la aplicación.

EL CÓDIGO

FormShow (paso 5) intenta leer la información de configuración del registro. Si el intento es exitoso, cambia las propiedades *Left*, *Top*, *Height* y *Width* del formulario para ponerse en concordancia con los valores recuperados del registro. Para hacer esto, primero construye un objeto *TRegistry*, y establece como valor de *RootKey* el siguiente: **HKEY_LOCAL_MACHINE**. Luego abre la verdadera llave del registro que contiene la información de la aplicación pasando el nombre de la clave del registro al método *OpenKey* de *TRegistry*. El segundo parámetro de *OpenKey* es true, para indicar al objeto que debería crear la clave del registro si no existe.

El resto de las operaciones de registro de *FormShow* están englobadas en un bloque *try*. Si ocurre un error dentro del *try*, el control se transfiere al match, que está vacío, lo que significa que el error será ignorado. El motivo de esto es sencillo. Si es la primera vez que se ejecuta la aplicación en esta máquina, la información de configuración no existirá, y las llamadas al registro fallarán. No hay ninguna necesidad de informar de estos errores al usuario porque son completamente normales.

FormClose (paso 6) hace lo contrario que *FormShow*. Escribe en el registro las propiedades *Left*, Top, *Height* y *Width*. La próxima vez que se ejecute la aplicación, la configuración estará en el registro, y *FormShow* será capaz de posicionar y fijar el tamaño del formulario adecuadamente.