CAPÍTULO 7: Conclusiones

7.1 INTRODUCCIÓN

El objetivo de este capítulo es realizar avances en lo referente a la actualización de la interfaz vía software, y no de forma manual como se indica en el manual de mantenimiento del capítulo anterior.

Ya vimos como la interfaz tiene una sección encargada de la actualización de las fichas de puesto para el caso de que éstas sufrieran modificaciones; mientras que si lo que ocurría consistía en la eliminación o la inserción de un puesto debíamos entrar en el editor de Visual Basic para modificar el código y los Userform directamente.

Por razones de comodidad, además de no correr el peligro de que la persona que modifique el código fuente de la aplicación lo deje por alguna circunstancia de forma inconsistente; lo más apropiado sería realizar esas modificaciones interactuando directamente con una interfaz mejorada para dar dichos servicios.

Por cuestiones de tiempo no se desarrollará con completo detalle toda la programación necesaria para realizar todo lo anterior. En su defecto lo que se intentará realizar será una especie de guía orientativa y de investigación, para que en el futuro otra persona retome la aplicación en este punto con intención de completar esta mejora.

Además en el último apartado de este capítulo se comentará algunas de las razones por las que resultaría interesante enfocar todo el proyecto a la utilización de una base de datos distribuida, lo cual no se ha realizado por no intentar encarecer el uso de esta aplicación, ya que necesitaría la compra de una licencia por puesto de algún tipo de software como puede ser Access de Microsoft.

7.2 SOLUCIÓN ADOPTADA

El primer tema en investigar fue la creación y eliminación de controles (en nuestro caso botones) en tiempo de ejecución. Con relación a lo anterior debemos aclarar los conceptos tiempo de ejecución y tiempo de diseño:

- ✓ **Tiempo de diseño,** periodo dedicado en exclusiva por el programador al diseño de la aplicación objetivo. Aquí se incluye tanto la generación de código fuente, su depuración y la edición de los formularios de usuario.

- ✓ **Tiempo de ejecución,** periodo durante el cual se está ejecutando la aplicación.

La eliminación e inserción de controles en tiempo de ejecución presenta una serie de inconvenientes para nuestros objetivos:

- ✓ No permite guardar cambios realizados sobre los formularios de usuario o UserForm en tiempo de ejecución.

- ✓ No es posible la eliminación en tiempo de ejecución de controles añadidos durante tiempo de diseño.

Ante esta serie de contratiempos la solución ideada consiste en que todos los controles tipo CommandButton sean añadidos en tiempo de ejecución, de forma que cada vez que se inicialice un formulario se añadan los controles necesarios.

La forma de implementar lo anterior sería la siguiente: en un formulario se recogería la información relativa al nuevo puesto a añadir o a eliminar; como consecuencia de esto habría que modificar la ficha de puesto prevista añadiéndole nuevos campos relacionados con la inserción dinámica del CommandButton asignado a ese puesto. Los dos casos con los que nos podemos enfrentar se solucionan así:

- ✓ En caso de la eliminación de un puesto, la operación a realizar consistiría únicamente en el borrado de la ficha de puesto. Cada vez que se carga un formulario se recorren todas las fichas de puesto; de forma que si no está, no se creará ningún control.

- ✓ En caso de la creación de un puesto, se procede creando una nueva ficha de puesto de modo que cuando recorramos todas las fichas al inicializar el formulario se crea el botón asociado.

7.2.1 CREACIÓN DE CONTROLES CON EL MÉTODO ADD

El código asociado para llevarlo a cabo es:

```
'Esto debe ir en la sección de declaraciones
Public WithEvents boton As CommandButton

'En alguana otro rutina se invoca al metodo Add
Set boton = Controls.Add("Forms.CommandButton.1", "nombre", Visible)

Private Sub boton_Click()
        .
        .

End Sub
```

Si no añadimos la palabra clave WithEvents el control añadido no captará los eventos relacionados con él, como por ejemplo la pulsación del botón.

WithEvents es una palabra clave que especifica que la variable es una variable de objeto utilizada para responder a eventos desencadenados por un objeto ActiveX. Su utilización solamente es válida en módulos de clase. Puede declarar tantas variables individuales como desee, pero no puede crear matrices con WithEvents.

Cuando declaramos una variable con WithEvents**,** lo que estamos es creando una variable que apunta a un control o a cualquier otro objeto ActiveX que desencadene eventos, (si no desencadena eventos no se puede usar WithEvents). Esa variable podrá acceder a los métodos y propiedades del objeto, (lo mismo que una variable declarada normal), pero además nos permite acceder a los eventos producidos por ese objeto, incluso se crea una entrada con el nombre de la variable en el combo de los objetos usados en el módulo de clase en el que se ha declarado... (un formulario también se considera una clase, por tanto, también se puede usar WithEvents en los módulos FRM, pero no se podrá usar en los módulos BAS).

Pero está solución presenta un grave inconveniente, debido a que solo en último control creado con WithEvents será el único que pueda interceptar eventos, mientras que los anteriores funcionan como si en su declaración no se hubiera utilizado la palabra clave WithEvents.

Además otro inconveniente presentado consistía en que Visual Basic no permitía la generación de controles en la rutina de inicialización de los formularios. Esto se solucionó mediante la adición de otro botón en tiempo de diseño que arranca la generación de los controles en tiempo de ejecución que fuese necesario.

7.2.2 CREACIÓN MEDIANTE EL MÉTODO LOAD

Mediante la utilización de este método resolvemos los dos problemas planteados por la solución anterior, es decir:

- ✓ Permite la adición de controles en tiempo de diseño en la rutina asociada a la inicialización de los formularios (UserForm_Load).

- ✓ Y sobre todo, se puede interceptar eventos con todos los controles añadidos en tiempo de ejecución.

Sin embargo, esta programación no es admitida por la versión de Visual Basic del editor de Excel; por ello la hemos desarrollado con un editor independiente a Excel, en concreto el Microsoft Visual Basic 6.0.

La forma de hacerlo consiste en crear un único control tipo pulsador (Command1) y asignarle a la propiedad Index el valor cero (por defecto no tiene ningún valor asociado), con esta acción se consigue poder tratar a ese botón como el primero (Command1 (0)) de una tabla con nombre Command1. El resto de Command que necesitemos en ese formulario ya si los podemos crear en tiempo de ejecución en la rutina del evento de carga del formulario con el siguiente código:

```
Private Sub Form_Load()

   Dim i As Long
    i = Command1.Count
   'Cargo el nuevo botón
   Load Command1(i)
   'Le asigno una serie de propiedades
   With Command1(i)
      .Top = Command1(i - 1).Top + 1000
      .Visible = True
   End With

End Sub
```

Ahora todos los eventos Click de cualquiera de los Command de la tabla (se distinguen por la variable Index) son interceptados por la siguiente rutina:

```
Private Sub Command1_Click (Index As Integer)
        .
        .
        .
End Sub
```

Para ilustrar los objetivos conseguidos de forma gráfica pondremos una imagen inicial del formulario y otra en la que se han creado varios botones en tiempo de ejecución:

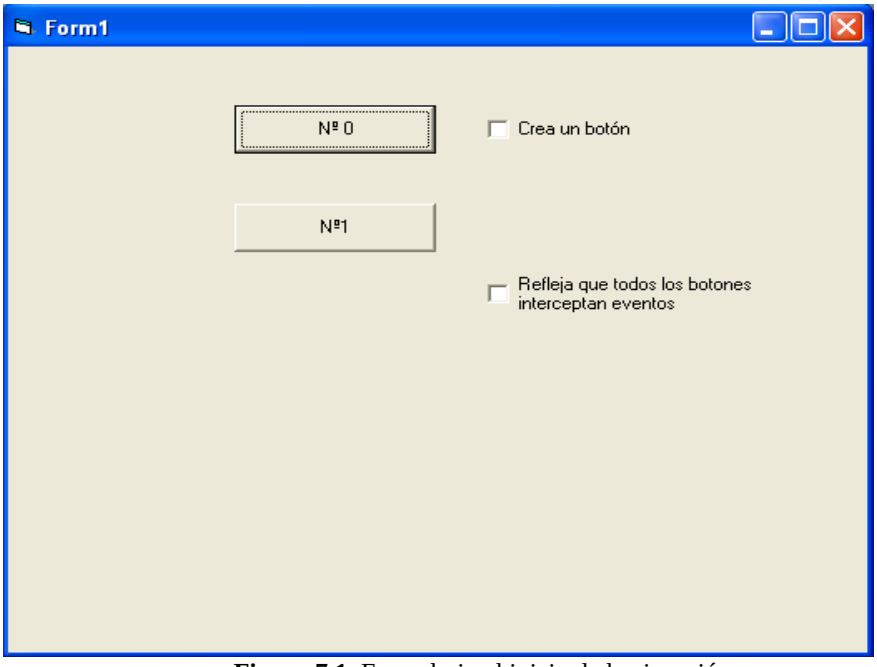

Figura 7.1: Formulario al inicio de la ejecución.

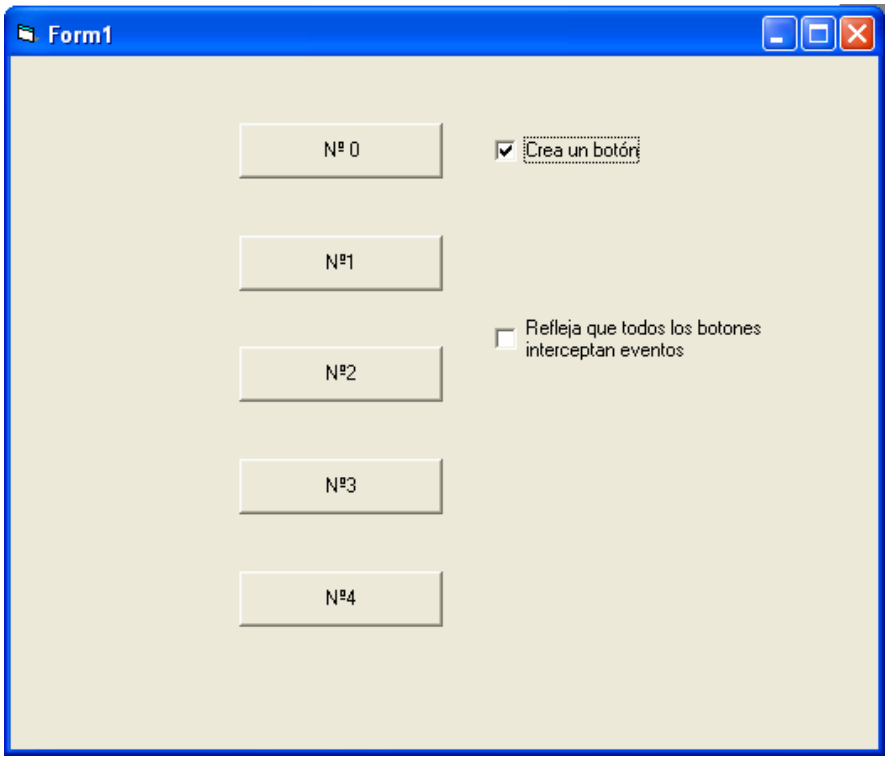

Figura 7.2: Formulario tras la creación de controles dinámicamente.

El usar Visual Basic de forma aislada a Excel no supone que no podamos seguir comunicando nos con las hojas de cálculo donde guardamos las fichas de puesto. Para ello hay que cargar una biblioteca o paquetes asociados en: Proyecto – Referencias - Microsoft Excel 10.0 Objetc Library.

La forma de utilizar un archivo Excel en un proyecto de Visual Basic 6.0 es la siguiente:

```
'Esto debe ir en la sección de declaraciones
Dim MiLibro  As Workbook ' Declara una variable de objeto.

Sub USAR_EXCEL ()

    'Establece después la variable con el método GetObject.
    Set MiLibro = GetObject  ("Ruta completa del archivo Excel a usar")

    'Acceso a los datos
    Variable = MiLibro.Worksheets(1) .Cells(1, 1).Value

End Sub
```

7.3 MIGRACIÓN A BASE DE DATOS

Lo más usual cuando se trabaja con grandes cantidades de datos como es nuestro caso, es la utilización de una base de datos. Las mejoras en el funcionamiento del proyecto que obtendríamos en caso de realizar esta migración serían:

- ✓ Al tratarse de una base de datos distribuidas no habría problemas en caso de que dos o más usuarios accedieran a ella de forma simultánea, tanto si quieren realizar una simple consulta o una actualización. Por el contrario, en la situación actual si dos usuarios abren el archivo colgado de la red de la factoría de forma simultánea, el primero tendrá todos los privilegios mientras que el segundo solo el de lectura.

- ✓ Las bases de datos realizan consultas selectivas en función de algún campo de forma más optimizada que la forma de llevarlas a cabo con nuestra interfaz actual, la cual consistía en diferenciar los distintos tipos de búsquedas posibles mediante la actualización de una variable habilitada para tal objetivo.

- ✓ No habría que crear una estructura de datos para cada puesto (hoja de cálculo). Esto se solucionaría distinguiendo todas las entradas a la base de datos (correspondiente a una palabra o byte accedida en lectura o escritura) en base a un nuevo campo que sería el número de puesto. El único pero a esta solución sería la repetición del uso de los números de puesto que se produce dentro de las líneas de montaje.

La razón por la cual no se orientó el proyecto de esta forma al inicio de su desarrollo, consistió en cumplir el objetivo de no encarecer el uso de esta aplicación; ya que necesitaría la compra de una licencia por puesto de algún tipo de software, como puede ser Access de Microsoft.