TEMA

45

Análisis y diseño de servicios de presentación en un entorno gráfico

Esteban Leyva Cortés

CUERPO DE PROFESORES TÉCNICOS DE FORMACIÓN PROFESIONAL

ÍNDICE SISTEMÁTICO

1. INTRODUCCIÓN

Este tema lo he dividido en dos partes claramente diferencias. En una de ellas hago un recorrido general del tema sin centrarme en ningún entorno (apartados del 1 al 9), y en la segunda parte (apartado 10) realizo una pequeña aplicación en Visual Basic para ilustrar los conceptos expuestos en la primera parte. He optado por emplear Visual Basic por ser uno de los entornos de programación gráficos más extendidos, de todas formas es muy sencillo alterar este ejemplo para cualquier otro entorno de programación.

Podemos definir una interfaz gráfica de usuario (GUI[1] a partir de ahora) como el software que facilita la interacción del hombre con la máquina. A partir de esta definición tan amplia podemos concretar algunas características comunes. Se realiza usando una interfaz gráfica intuitiva de alto nivel, de forma que cada acción "lógica" se representa mediante una acción "física" en la pantalla.

La mayoría de estos sistemas se basan en el concepto de escritorio, el uso de ventanas y elementos que permiten la interacción con el usuario, como botones, cajas de texto, iconos, etc. Estos entornos, además, intentan ser visualmente agradables.

Uno de los principales objetivos de este tipo de entornos es minimizar el tiempo de aprendizaje de la aplicación para el usuario. Sin embargo, desde el punto de vista del programador, ha supuesto un importante giro en la filosofía tradicional de programación. No debemos olvidar que la programación, orientada al objeto al principio, y posteriormente la programación "visual", han estado íntimamente unidas al éxito de los entornos gráficos.

En los orígenes era necesario aprender gran cantidad de objetos, con sus métodos y propiedades para crear estas aplicaciones. Actualmente, las herramientas de diseño son a su vez entornos gráficos, lo que ha facilitado considerablemente el trabajo del desarrollador.

2. INTERFACES GRÁFICAS DE USUARIO (GUI): ORÍGENES, CARACTERÍSTICAS Y EVOLUCIÓN

A lo largo de este apartado vamos a realizar un breve recorrido por la historia de las interfaces gráficas de usuario (GUI).

Para conocer los inicios de las interfaces gráficas tenemos que remontarnos al año 1968. Douglas Englebart crea un sistema denominado **NLS** o *OnLine-System* cuando trabajaba en el Centro de Investigación de Stanford. NLS se compone de un macrocomputador con cuatro dispositivos conectados, una pantalla con posibilidad de mostrar textos y líneas (basadas en sistemas de dibujo vectorial), un teclado (como el de una máquina de escribir), un teclado de acordes (tenía cinco teclas, de forma que cada combinación diferente realizaba un comando), y un ratón (de madera). El ratón disponía de un pequeño puntero que nos permitía moverlo por la pantalla. Douglas Englebart es considerado el invertor del ratón y el precursor de los interfaces gráficos.

Xerox decide dejar de ser una industria papelera e invertir en el mundo de la "oficina sin papel". Esto le lleva a fundar en 1970 el centro de investigación de **Palo Alto**. Uno de los principales inventos de este grupo fue la impresora láser; el problema es que necesitaban un ordenador gráfico para poder trabajar con documentos de oficina. Esto les lleva a crear el "Alto", un ordenador con el tamaño de una pequeña nevera que cabía debajo de la mesa. El interfaz que se desarrolló para este ordenador era gráfico, pero no trabaja con vectores, sino con píxeles. Toman como refe-

[1] GUI = *Graphical User Interface*.

rencia el ratón de Englebart, modificándolo con tres botones. El puntero del ratón toma la forma de un puntero tal cómo los conocemos actualmente. El primer software que se desarrolló era una interfaz de manejo de ficheros bastante pobre, el "Alto File Manager".

Llegado a este punto se dieron cuenta que era necesario desarrollar software para esta máquina y la forma de desarrollarlo debía ser diferente; no podía usarse la programación clásica para desarrollar GUI. En 1974 nace **SmallTalk**. SmallTalk ("pequeña charla" en castellano) es un lenguaje de programación orientado a objetos, un lenguaje ideado para crear interfaces gráficas de usuario que a su vez funcionaba bajo un interfaz gráfico de usuario.

Con la combinación de Alto y SmallTalk se empezó a desarrollar el primer software completamente gráfico. En 1981 Xerox presenta el procesador de textos "Xerox Star 8010". Sin embargo se le habían adelantado otras compañías.

Seguramente ya habrá adivinado que estábamos pensando en Apple. Esta compañía fundada en 1979 en un garage por Steve Jobs y Steve Wozniak empezó a construir ordenadores a mano para venderlos al por menor en reuniones de forofos de la electrónica. El primer ordenador gráfico desarrollado por la compañía recibe el nombre de **Apple II**. Esta máquina mostraba en pantalla texto y gráficos, pero se manejaba mediante comandos.

El gran éxito de Apple II permite que la compañía crezca y contrate a ingenieros que trabajaban en Palo Alto. Esto posibilita la creación de **Lisa**[2]. Lisa sí es un ordenador basado en un entorno gráfico, manejado por el ratón, y donde aparece la idea de arrastrar y soltar. Usaba un ratón de dos botones, por lo que se inventa el doble click para suplir al tercero. Este proyecto se inicia en 1979 y la presentación oficial es en 1983. Debido al alto precio de la máquina (10.000 $) no tuvo el éxito esperado. Steve Jobs se encargó de crear una versión económica con una pantalla de menor calidad, menos memoria (128 Kilobytes) y sólo una disquetera por 2.400 $.

El impacto de Lisa fue tal que todas las compañías intentaron crear interfaces similares. Los años ochenta se caracterizaron por la aparición de una gran cantidad de entornos como VisiOn, Windows 1.0 y 2.0, DeskMate, GEM, Amiga WorkBench, GEOS, Acorn, NeXT STEP, OS/2, X Window System, etc.

Si los años ochenta se caracterizaron por la gran cantidad de entornos gráficos, es en los noventa cuando se consolidan estos sistemas, o fracasan. Entre los que consiguen éxito podemos mencionar **Windows** 3.0 y 3.1. Lanzados en 1992 con algunas carencias consiguió hacerse muy popular. Es realmente el lanzamiento de Windows 95 el que convierte a Microsoft en la creadora del sistema operativo más vendido de todos los tiempos.

Apple desarrolla un entorno denominado Aqua para su sistema operativo **Mac OS X**. Este entorno es visualmente agradable. Todas las ventanas usan doble buffer para evitar el problema del parpadeo o que veamos cómo se dibujan delante nuestra.

Otros sistemas desarrollados en esta época que merece la pena mencionar son OS/2 y BEOS.

Como ya bien sabrá, actualmente el pastel de los entornos gráficos se lo reparten entre Microsoft y Apple. Los entornos libres de desarrollo como Linux son un factor a tener en cuenta en un futuro próximo.

3. DISEÑO DE INTERFACES: CRITERIOS DE DISEÑO

Existen muchas reglas que podemos seguir en el momento del diseño de la interfaz de nuestro programa. Muchas compañías desarrollan sus propios criterios como si se tratase de un "sello de calidad" de las mismas. Vamos a estudiar algunas de las más importantes:

---

[2] Bautizado así por ser el nombre de la hija de Steve Jobs.

3.1. Reglas de oro de Theo Mantel

Destacaremos las tres reglas de oro de Theo Mantel[3] por su sencillez y claridad. Estos axiomas son los siguientes:

1. **Ceder el control al usuario**: realmente no hay nada más frustante que una aplicación que tome el control del ordenador y no nos deje darle órdenes, o que realice acciones automáticas indeseadas sin haberlas solicitado. El usuario ha de tener la agradable sensación de controlar el programa en todo momento. Para realizar esta empresa hemos de:
   - **Permitir parar acciones y deshacerlas**: al manejar el sistema el usuario es muy posible que se equivoce, por lo que hemos de darle la opción de deshacer la última o últimas acciones realizadas, o detenerlas en cualquier momento sin perder el trabajo realizado hasta el momento.
   - **Crear interacciones directas con objetos visuales**: nuestro interfaz debe funcionar como si fuese físico, los objetos se podrán mover, desplazar, apretar. Cada acción "lógica" debe tener una representación "física" en la pantalla.
   - **Permitir una interacción flexible**: cada usuario tiene preferencias sobre la manera de interacción con la aplicación. Algunos preferirán manejar sólo el ratón, a otros les gustará más el teclado, otros preferirán la combinación de ambos, o en programas muy específicos otros sistemas como joysticks, pantallas táctiles, lápices ópticos, tabletas digitalizadoras, etc. Conseguiremos ganar puntos si dejamos que cada usuario emplee el modo de interacción que prefiera.
   - **Acelerar y personalizar la interacción**: cuando el usuario ya esté familiarizado con nuestro programa deseará usarlo de forma más rápida. En este punto será interesante que el usuario pueda cambiar las teclas o comandos asociados para ponerlos a su gusto o crear "macros" a su estilo.
   - **Ocultar detalles técnicos al usuario novato**: nunca se debe preguntar al usuario novel detalles técnicos que seguramente desconocerá, como el hardware de su máquina, su sistema operativo, sobre la forma de almacenamiento de su sistema, la IRQ que usa su tarjeta de sonido, etc.
2. **Reducir la carga de memoria del usuario**: hemos de conseguir que el usuario no memorice datos "inútiles" que le hagan pesado el uso del programa. Hemos de intentar:
   - **Proporcionar valores por defecto**: el usuario podrá introducir los valores que desee, sin embargo, la aplicación debe proporcionarle unos valores por defecto útiles para evitarle esa tarea. También es conveniente que el usuario pueda volver a los valores por defecto cuando ya haya introducido los suyos.
   - **Usar metáforas del mundo real**: Debemos emplear objetos visuales similares a los del mundo real para desarrollar nuestro interfaz, de manera que el comando para imprimir tenga la forma de una impresora, el icono para guardar de un disco duro, etc.
   - **Desglosar la información**: la información aparecerá agrupada en diferentes niveles de abstracción, empezando por el nivel más alto hasta el más bajo. Para grabar un archivo existe un botón que lo hace directamente. Si queremos cambiar el formato del archivo, la ubicación por defecto, accedemos a otro menú donde tenemos más opciones.

[3] Si desea más información le remitimos a la bibliografía.

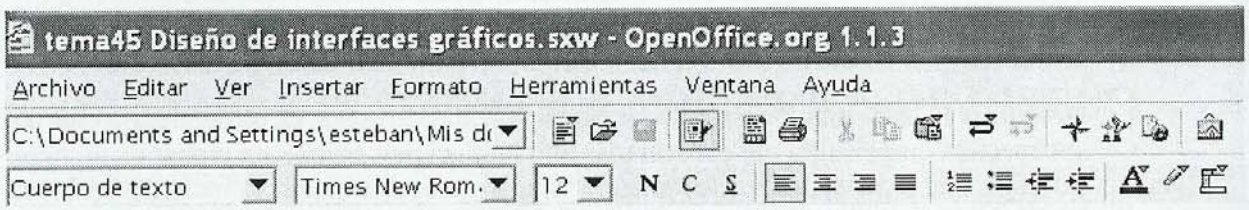
- **Crear asociaciones intuitivas**: si tenemos que asignar teclas o comandos que sean fáciles de recordar, por ejemplo, si queremos grabar un archivo en algunos programas, hemos de pulsar CTRL+S, si siempre hemos usado CTRL+G.
- **Reducir la memoria a corto plazo**: nuestra aplicación debe ser capaz de indicar las últimas acciones realizadas por el usuario de una forma visual clara.

3. **Diseñar una interfaz coherente:** nuestra aplicación debe ser homogénea y consecuente con su propia filosofía. La información visual debe ser similar en toda la aplicación, la forma de introducir datos, cambiar de tareas o realizar acciones debe ser igual durante el programa. En este caso hemos de tener en cuenta:
   - **Mantener la coherencia en todos los productos de nuestra empresa y las diferentes versiones del mismo programa**: al usuario que esté familiarizado con una versión antigua o con un programa de la misma empresa no debe costarle mucho trabajo hacerse con el control de una versión nueva o un programa diferente de la misma empresa.
   - **No realizar cambios al usuario a menos que sean beneficiosos para el mismo**: por ejemplo, si el botón izquierdo del ratón se suele emplear para seleccionar elementos, no lo cambiaremos.
   - **Posibilitar al usuario ubicarse en el contexto adecuado**: el usuario ha de poder localizar la tarea que está realizando en cada momento y debe conocer las posibilidades de cambiar de una tarea a otra.

3.2. Criterios de Benjamin Roe[4]

Benjamin Roe es un desarrollador de software libre vinculado desde hace años al diseño de interfaz de usuarios. Su experiencia se resume en cinco criterios que hemos de tener en cuenta cuando preparemos el diseño de nuestra página:

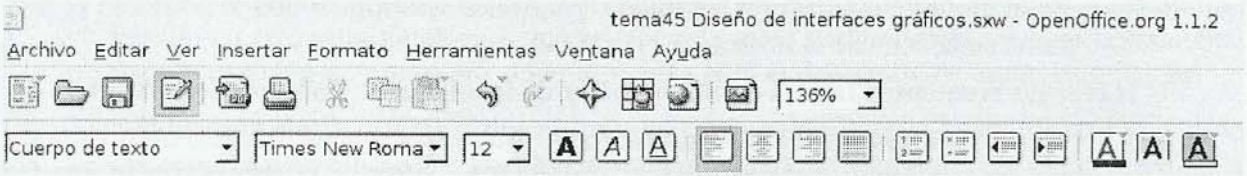
- **El usuario no está usando su aplicación**: el usuario no está usando su aplicación, simplemente quiere realizar el trabajo tan pronto como pueda, de la manera más sencilla, y por lo tanto irse a casa o ponerse a hacer otra tarea cuanto antes. El tiempo que pierda el usuario aprendiendo a manejar la aplicación es tiempo de producción perdido. Todos los criterios que estudiamos a continuación se desprenden de esta afirmación.
- **La ley del ajuste**: Esta ley es tan sencilla como pensar que cuando más importante sea un elemento de la pantalla más fácil debe ser acceder a él. Esta ley tiene tres premisas:
  1. **Cuando más se use un control, más grande debe ser, y más diferenciado debe estar**. Veamos un ejemplo de lo que no hay que hacer:



[4] Benjamin Roe es desarrollador de software libre, ha trabajado en proyectos como "siEd", un editor de texto para Palm Os, y trabaja actualmente para "Centre Process for System Engieneering" del "Imperial College".

Aquí tenemos la parte superior del editor de textos del OpenOffice[5] para Windows. Están todos los controles juntos, son pequeños y son muy parecidos, por lo que es muy fácil equivocarse al marcar. ¿Usamos igual el estilista que los botones de negrita, cursiva o subrayado? ¿No sería mejor tener los botones más usados en una parte de la pantalla con un tamaño considerable, y de fácil acceso?

Sin embargo, la versión de Linux tiene otro aspecto:



Donde los iconos más importantes son más grandes y están mejor situados. Puede comprobar la diferencia por sí mismo.

2. **Nunca situemos elementos a un pixel del borde de la pantalla**: de esta forma sólo conseguirá que el usuario se equivoque y mueva la ventana, le cambie el tamaño, etc.
3. **Situar los botones importantes en las esquinas y borde de la pantalla**: simplemente porque es más sencillo mover el ratón en diagonal que en línea recta. La mayoría de las ventanas tienen los botones de minimizar, cerrar o maximizar en la parte superior derecha y los comandos más importantes en la parte superior izquierda.

- **Interferencia inútil**: el usuario está trabajando, no lo molestemos con información innecesaria o irrelevante. Si es necesario mostrar información siempre se puede realizar de forma gráfica mediante una animación, color, u otra forma que no paralice su trabajo. Este criterio se divide en tres partes:

1. **Sólo usar diálogos cuando sean imprescindibles**.
2. **No tapar la zona de trabajo del usuario a menos que sea estrictamente necesario**.
3. **Si es posible usar indicadores gráficos en vez de indicadores textuales**.

- **Use la potencia del ordenador**: los ordenadores actuales son máquinas potentes que han mejorado bastante en los últimos años, desgraciadamente, nosotros no hemos evolucionado mucho: nos distraemos con facilidad y no tenemos una memoria portentosa, por lo tanto no debemos sobrecargar al usuario con decisiones irrelevantes que la máquina puede hacer por nosotros. Se puede resumir en las siguientes reglas:

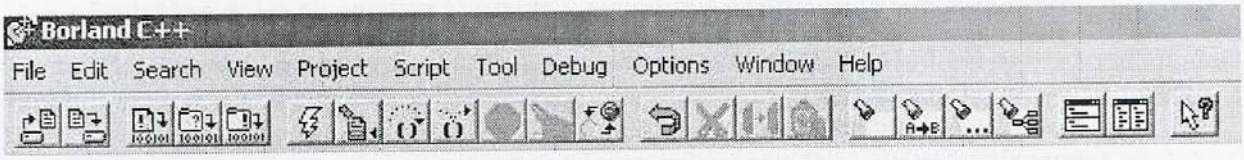
1. **El ordenador es potente: use su potencia para ayudar al usuario**: use la potencia del ordenador para hacerle la vida más fácil y agradable al usuario, no para complicársela.
2. **Haga los elementos similares fáciles de distinguir**. En este caso voy a poner un ejemplo en Windows Xp. Supongamos que tengo cuatro aplicaciones funcionando: una el procesador de textos del OpenOffice, el programa Paint, y dos ventanas de MS-DOS. Los botones que corresponden a aplicaciones diferentes son claramente diferenciables, sin embargo las dos ventanas de MS-DOS son iguales y en una tengo abierto el editor de textos "edit" y en la otra no. ¿Cuál de las dos será?

[5] Las críticas al diseño del interface de programas son constructivas y didácticas, y nunca responden a un interés destructivo o comercial.

3. **Las aplicaciones deben recordar la configuración.** En algunos programas es necesario cambiar la configuración cada vez que se entra, o buscar en algún menú un poco escondido la opción de guardar las preferencias del usuario. Lo ideal es que la propia aplicación "recuerde" cómo dejamos los elementos de pantalla la última vez que lo usamos. En este caso vamos a poner el ejemplo del entorno de programación "Anjunta", donde cada vez que entramos hemos de seleccionar el lenguaje de programación y las opciones de configuración seleccionadas la última vez, el archivo con el que estábamos trabajando, etc. ¿No sería más lógico que al entrar en el programa se encargase él mismo de "recordar" esos detalles?

- **Hacer los elementos fáciles de distinguir y de localizar**: lo podemos resumir en:

1. **Hacer distintos los iconos que realizan diferentes tareas.** En este caso vamos a usar como ejemplo del compilador Borland C++ 3.5 para Windows. En su interfaz gráfica, como puede comprobar, todos los iconos son muy parecidos. El icono de grabar y cargar son casi iguales (los dos primeros a la izquierda), los tres iconos siguientes son prácticamente idénticos y el resto también. Además, estos iconos no representan objetos de la vida real ni son intuitivos.



2. **No sobrecargar al usuario con muchas opciones donde elegir.**
3. **Hacer los elementos seleccionados fáciles de distinguir, y de leer.**

4. PROCESO DE DISEÑO DE LA INTERFAZ DE USUARIO

El proceso de diseño de la interfaz de usuario se divide en las siguientes tareas:

- **Análisis y modelado de usuario, tareas y entornos**: en primer lugar hemos de estudiar el perfil de los usuarios que van a emplear nuestra aplicación. Debemos conocer la experiencia del usuario usando aplicaciones, su nivel de conocimiento, edad, etc. A continuación estudiamos las tareas que va a realizar nuestra aplicación. Debemos

identificarlas, describirlas y estudiar cómo se realizarán. Por último vemos el entorno, dónde se ubicará el interfaz, dónde se situará el usuario, si compaginará el uso de esta aplicación con otras, etc.

- **Diseño de la interfaz**: a partir de la información recabada en la primera parte del diseño podemos empezar el diseño del interfaz. El objetivo del diseño del interfaz es definir un conjunto de objetos, acciones, y su representación gráfica.
- **Implementación de la interfaz**: normalmente en este apartado creamos un prototipo basado en el diseño anterior. Para realizar esta tarea podemos usar alguna herramienta para la creación de prototipos o herramientas para el desarrollo de entornos gráficos.
- **Evaluación de la interfaz**: por último debemos comprobar si la interfaz ha cumplido las expectativas esperadas. Hemos de tener en cuenta la facilidad de uso del interfaz, la aceptación del interfaz por los usuarios, si se han conseguido acoplar correctamente las acciones, tareas, etc.

Además hemos de comentar que estas tareas requieren refinamiento, es decir, si la evaluación de la interfaz no es satisfactoria hemos de volver a realizar todas las tareas, de esta forma se crea un proceso en espiral, ya que estamos creando prototipos.

A continuación estudiaremos cada tarea independientemente.

4.1. Análisis y modelado de tareas

El análisis y modelado de tareas básicamente puede recibir dos enfoques, por un lado tenemos el enfoque tradicional, y por otro el orientado a objetos. Nosotros, lógicamente, nos decantaremos por el enfoque orientado a objetos.

El objetivo de esta fase consiste en definir las diferentes tareas y clasificarlas. Nuestro punto de partida debe ser las tareas humanas, de manera que estableceremos un paralelismo entre los objetos reales y los objetos "simbólicos" que vamos a manipular en la aplicación. Esta fase también necesita realizar sucesivos refinamientos hasta que las tareas se definan con el nivel de detalle necesario.

Por ejemplo: supongamos que vamos realizar un estudio de grabación de sonido virtual. Debemos, entonces, ir a un estudio de grabación y observar cuáles son las tareas necesarias para grabar una canción. Normalmente primero se graban los instrumentos uno a uno, después se mezclan y se comprueba el resultado. Si no es satisfactorio, quizás sea necesario remezclar los sonidos o regrabar algún instrumento. Una vez finalizada la grabación la pasaremos a otro soporte, por ejemplo un CD. Muy bien, pues tenemos las cuatro tareas básicas ya definidas:

- Grabar un instrumento.
- Mezclar todos los instrumentos
- Escuchar el resultado final.
- Grabarlo en algún soporte.

Cada tarea se puede dividir en subtareas si la analizamos de forma independiente. Por ejemplo, al grabar un instrumento es necesario realizar algunas pruebas de sonido, ecualizarlo, añadirle reverb, limpiar la señal original, aplicarle algún efecto, etc. De esta manera podemos ir estudiando las diferentes tareas y subtareas que va a permitir nuestra aplicación.

4.2. Diseño de la interfaz

Una vez que conocemos las tareas que debe realizar nuestra aplicación es el momento de diseñar nuestra interfaz.

En primer lugar definiremos los **objetos** y las **acciones**. Este enfoque es similar a la definición de las diferentes clases en un diseño orientado al objeto. Lo primero que debemos hacer es realizar una descripción textual del escenario de nuestra aplicación. En esta descripción separaremos los sustantivos y los verbos. A continuación comprobaremos qué sustantivos definen objetos de nuestra interfaz y qué verbos se corresponden con posibles acciones.

Los objetos los clasificaremos en:

- **Objetos origen**: son elementos de la pantalla que se arrastran y se dejan caer sobre un objeto destino. Por ejemplo, en la aplicación musical que comentábamos anteriormente, la mezcla final (que tendría una representación de onda) la podemos dejar caer en un icono en formato de CD si queremos grabarla en un soporte externo. Esto provocaría la acción de grabar un CD con el tema seleccionado.
- **Objetos destino**: los objetos destino recogen a los objetos origen.
- **Objetos de la aplicación**: son objetos "internos" de la aplicación que no se manejan directamente. Volviendo al ejemplo anterior, podemos tener un listado de todas las pistas grabadas, de manera que podemos seleccionar mediante un menú si queremos insertarlas en la mezcla actual, duplicarlas, borrarlas, etc.

A continuación debemos definir el formato de pantalla. Éste también es un proceso iterativo donde hemos de estudiar el diseño gráfico que vamos a emplear, la colocación de los iconos, los menús que usaremos, los textos descriptivos, los mensajes de error, etc.

Una vez finalizado el diseño debemos realizar una revisión del mismo. Entre los muchos problemas que podemos encontrarnos suelen ser frecuentes los siguientes:

1. **Tiempo de respuesta del sistema**: el tiempo de respuesta encierra en sí dos conceptos: la duración y la variabilidad. La duración se puede definir como el tiempo que tarda nuestro sistema en realizar una acción. Este tiempo no interesa que sea demasiado largo, porque puede aburrir al usuario, ni demasiado corto (quizás mostremos información en pantalla y al usuario no le dé tiempo a leerla). La variabilidad es la diferencia de tiempo entre la realización de las diferentes acciones. Es interesante que la variabilidad se aproxime a cero para que el uso de la aplicación, por decirlo de alguna forma, tenga siempre el mismo "ritmo". Dicho de otra manera, nos interesa que la duración de las diferentes acciones sean similares.
2. **Servicios de ayuda al usuario**: normalmente se proporciona al usuario dos tipos de ayuda: por un lado están las ayudas integradas en la aplicación y por otro las ayudas complementarias. Las ayudas integradas se van creando a medida que vamos generando la aplicación. Son ayudas contextuales en la mayoría de los casos, guías que se activan al iniciar una tarea, etc. Las ayudas complementarias se crean al finalizar la aplicación, son manuales de usuario insertados en la aplicación, manuales a través de internet, etc. Estos manuales permiten realizar consultas introduciendo una palabra clave y suelen tener un pequeño índice agrupado por tareas.
3. **Manipulación de la información de errores**: desgraciadamente, por perfecta que sea nuestra aplicación fallará en algún momento. Normalmente se soluciona mostrando un mensaje de error. El mensaje de error que muchas veces nos encontramos es del tipo: "ERROR: 114". La verdad es que no da demasiada información sobre lo que ha sucedido. Seguramente en el manual impreso de la aplicación vendrá explicado qué significa esto. ¿No sería mejor incluir la descripción del error también?, algo como "ERROR: 114. Modo de vídeo incorrecto".

Otro problema con el que nos encontramos con más frecuencia de la deseable es cuando usamos alguna aplicación en "español". Curiosamente, cuando se produce algún error nos aparece una parrafada en inglés bastante difícil de comprender. No debemos emplear nunca una jerga informática en los mensajes de errores, para que cualquier persona lo entienda. "ERROR: 114. Resolución de pantalla insuficiente". También debemos indicar la posible solución al problema. "ERROR: 114. Resolución de pantalla insuficiente. Cambie la resolución del monitor a 1024 × 748 puntos con 256 colores".

4. **Etiquetado de órdenes**: otro tema que hemos de revisar es el etiquetado de órdenes en los menús, botones, etc. Debemos buscar la palabra adecuada para cada acción, buscar atajos de teclado significativos y fáciles de recordar, emplear teclas de función para las órdenes más usadas, etc.

4.3. Implementación

Si queremos realizar la implementación de nuestra interfaz debemos emplear una herramienta de desarrollo rápido o de construcción de prototipos. Estas herramientas han de permitirnos crear ventanas, menús, manejos de errores, cuadros de diálogo, etc. En el siguiente apartado de este mismo tema se habla de algunas aplicaciones de este tipo.

4.4. Evaluación

Existen mucho métodos diferentes para evaluar el diseño de la interfaz. Nosotros vamos a ver uno basado en tres pasos:

1. **El usuario evalúa el interfaz**: mediante un cuestionario se comprueba si la interfaz que hemos diseñado es del agrado del usuario. Este cuestionario puede ser cualitativo o cuantitativo. Si es cualitativo el usuario responderá con sí/no a preguntas del tipo: "¿Le ha parecido fácil usar la aplicación?, ¿las acciones eran claras?, ¿los mensajes de error eran compresibles?", etc. Si realizamos un estudio cuantitativo podemos estudiar el tiempo que tarda el usuario en aprender a usar la aplicación, el tiempo que ha estado aprendiendo su manejo, el tiempo que ha estado leyendo la ayuda, etc.
2. **La evaluación es estudiada por el diseñador**: el diseñador debe comprobar si esta interfaz cumple los requisitos planteados al comienzo del diseño.
3. **Se modifica el prototipo**: si la interfaz no ha cumplido las expectativas, es necesario volver a revisar el diseño, modificar los errores y defectos del mismo. Una vez modificado debemos volver al primer paso, hasta conseguir un interfaz que cumpliese los requisitos fijados por el cliente.

5. HERRAMIENTAS PARA EL DESARROLLO DE ENTORNOS GRÁFICOS

Existen infinidad de herramientas para el desarrollo de entornos gráficos. Las podemos clasificar en herramientas comerciales y libres. Las herramientas comerciales son aquéllas desarrolladas por empresas para su venta a programadores. Dentro de esta categoría podemos citar a dos grandes empresas: Microsoft y Borland. Por otro lado no debemos olvidar proyectos abiertos de desarrollo libre como Mono, Glade, etc. El objetivo de este apartado no es realizar un estudio exhaustivo de los diferentes entornos, sino efectuar un estudio global de las posibilidades de programación si se decide profundizar en este tema.

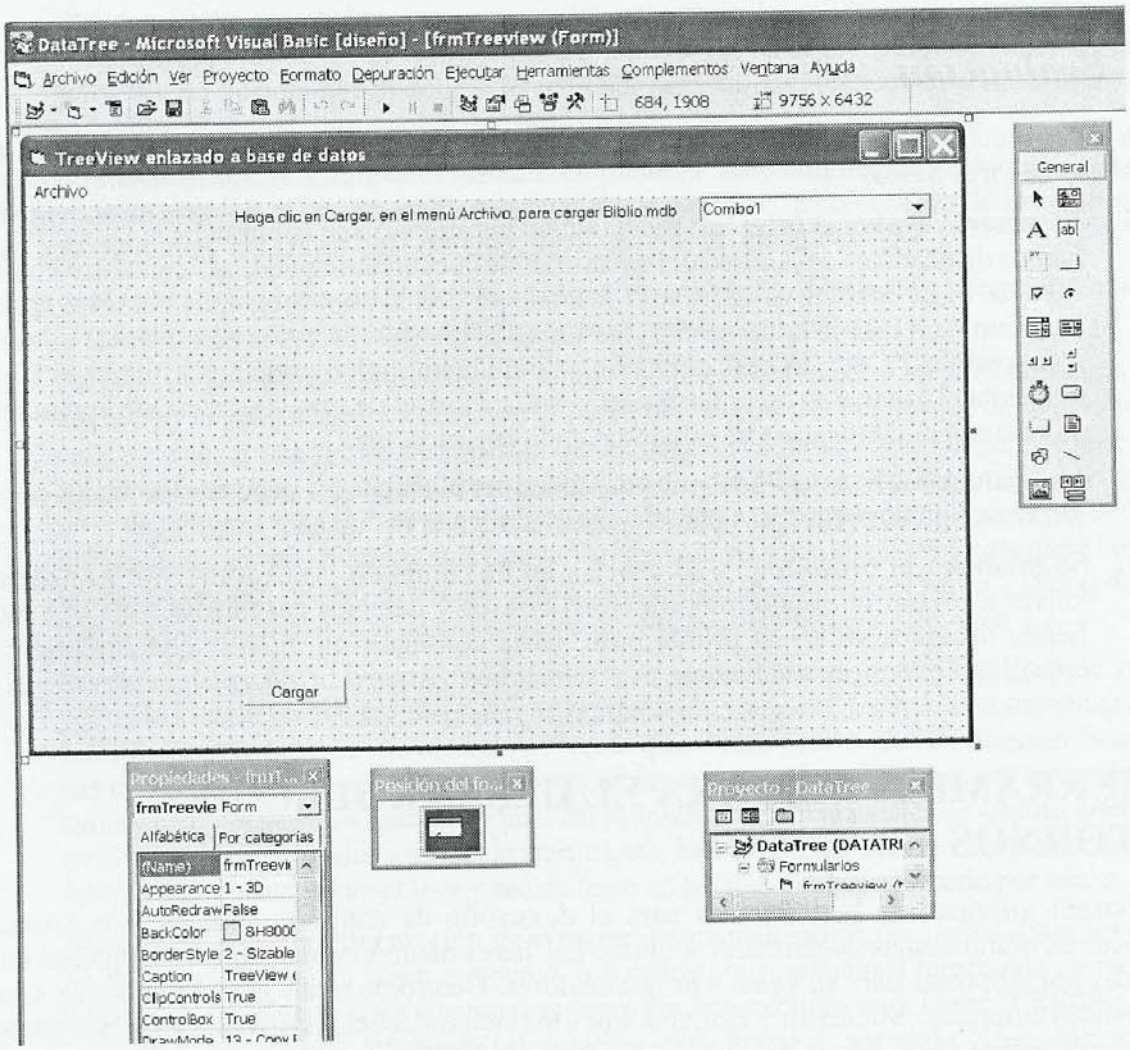
5.1. Microsoft

Esta empresa siempre ha apostado por el desarrollo en entornos gráficos comercializando herramientas similares a las que emplea para el desarrollo de Windows. En la evolución de estos entornos podemos distinguir tres fases: los primeros entornos de desarrollo de Microsoft, Visual Studio y la plataforma .NET.

Microsoft desarrolla los siguientes lenguajes de programación:

- **Visual Basic**. Uno de los primeros en aparecer. Como su nombre indica se basa en el lenguaje de programación BASIC (bastante limitado como lenguaje en sí). Como ventaja podemos decir que es muy fácil de manejar y que con unas nociones elementales de programación podemos empezar a realizar nuestros primeros proyectos. Otra ventaja es que trabaja muy bien con Windows y con Office[6].

  La primera version de Visual Basic (1.0) aparece en mayo de 1991 y posibilitaba lo que por aquella época era muy complicado (requería un conocimiento profundo del funcionamiento de Windows). Basado en Qbasic, permitía introducir controles sencillos de programar y crear aplicaciones gráficas con poco esfuerzo.



6 Office fue desarrollado en Visual Basic.

La versión 3.0 aparece en el año 1993 e incluía la utilidad conocida como *Crystal Reports*, que permitía diseñar informes a medida, y el DAO (*Data Access Object*) que posibilitaba acceder a las bases de datos Access 1.0, y otros estándares de bases de datos.

La versión 4.0 se publica en setiembre de 1995 y es la primera versión que trabaja en sistemas operativos a 32 bits (Windows 95 y NT).

Las versiones 5.0 (marzo de 1997) y 6.0 (junio de 1998) añaden controles para trabajar con Internet, haciendo hincapié en el DHTML y los controles ActiveX.

La versión .NET permite usar la programación orientada al objeto en todo su esplendor usando herencia, excepciones, etc.

- **Visual C++**. Es uno de los entornos de desarrollo más importante de los últimos tiempos. No debemos olvidar que Windows ha sido desarrollado con Visual C++.

  A diferencia de Visual Basic, Visual C++ se basa en C++, lo que elimina todas las limitaciones iniciales de lenguajes obsoletos, por lo que disponemos de la potencia de la programación orientada a objetos y un entorno de desarrollo gráfico.

  Las últimas versiones vienen integradas en la plataforma .NET, por lo que permiten entre otras características usar XML, ASP, usar la MFC, etc.

- **Visual C#**. Debido a las limitaciones del lenguaje BASIC, se realizó una revisión del mismo, de forma que éste es una mezcla entre C y BASIC. De esta manera se pretende aunar la sencillez del uso del BASIC con la potencia del C.

- **Visual J#**. En este caso estamos ante la versión que hace Microsoft de Java, basada en su entorno .NET.

5.2. Borland

Borland es otra de las compañías que apostó fuerte por el desarrollo de herramientas de programación en este tipo de entornos.

- **Delphi**. Una de las primera herramientas en aparecer. Es muy similar a Visual Basic, sin embargo se basa en el lenguaje PASCAL (concretamente en OBJECT PASCAL[7]), mucho menos limitado que el BASIC y fácil de manejar.

  La primera versión de Delphi apareció en el año 1995 y funcionaba en Windows 3.1. La última versión que se ha desarrollado es la denominada 2005, que permite trabajar con la plataforma .NET, aplicaciones Windows a 32 bits, y soporta C#.

- **C++ Builder**. Basado en unos de los mejores compiladores de C++ (Borland C++) es la versión mejorada empleando un entorno gráfico de desarrollo. Siempre ha sido un entorno de desarrollo gráfico, sin embargo, en las últimas versiones se pueden ejecutar en la mayoría de las plataformas disponibles actualmente, pero no son un entorno de desarrollo gráfico.

- **Jbuilder**. Entornos de desarrollo basado en Java. Es uno de los mejores compiladores de Java y está disponible para gran cantidad de plataformas (Windows, Linux, Solaris y Mac Os). Además genera código Java estándar.

---

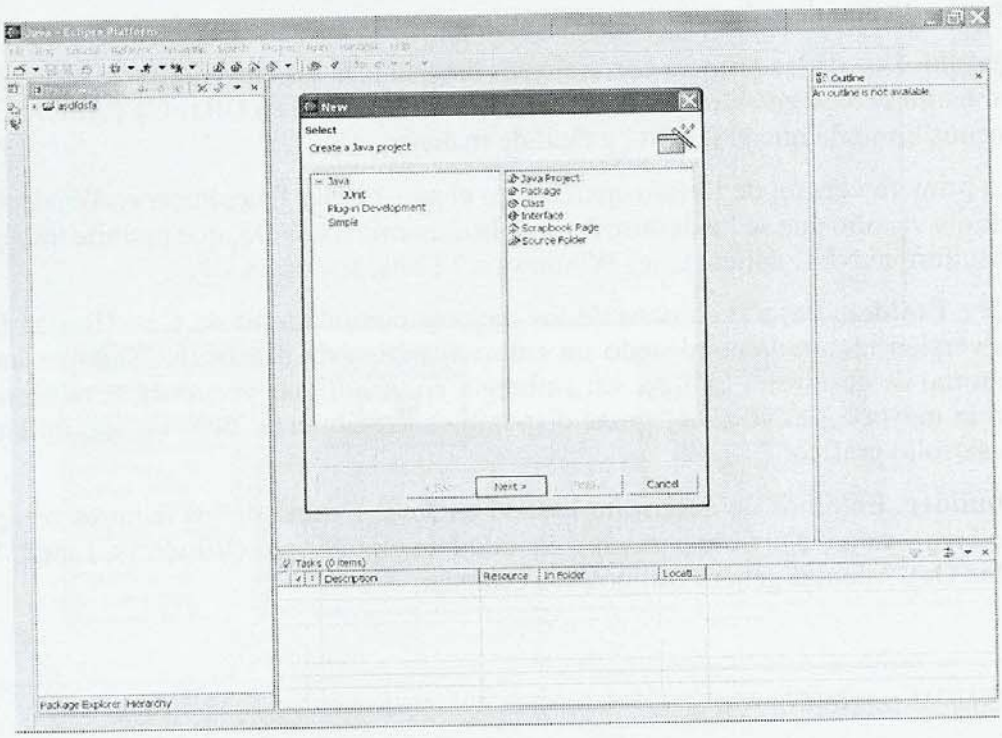
7 OBJECT PASCAL es un PASCAL orientado a objetos que permite emplear herencia, polimorfismo, etc.

- **Kylix**. Nueva apuesta de Borland. Se trata de un entorno de desarrollo muy similar a Delphi que funciona bajo Linux. Permite usar el lenguaje C++ o Delphi.
- **Glade**. Glade es un entorno de programación gratuito basado en GTK+ y Gnome. Es ideal para desarrollar aplicaciones de código abierto y permite emplear diferentes lenguajes de programación como C++, C, Jave, Perl, Python, etc.

  Muchas aplicaciones de código abierto como GIMP han sido desarrolladas con este entorno. En su página web puede ver un listado de las aplicaciones realizadas con GLADE:

  * **GTK+**. Realmente GTK+ es un conjunto de librerías que permite programar un entorno gráfico, pero GTK+ no es un entorno de desarrollo gráfico en sí mismo. GTK+ pertenece al proyecto de código abierto Gnu. El entorno Gnome fue desarrollado con estas librerías, de ahí su importancia.
  * **Mono**. Es un proyecto comunitario cuyo propósito es realizar un entorno de desarrollo compatible con la plataforma .NET de microsoft, pero gratuito. Esta iniciativa está impulsada por Ximian (comprada recientemente por Novell). Incluye varias herramientas, entre las que destaca un compilador de C#, Monodevelop (entorno de desarrollo integrado), Monodoc (visor de documentación para el desarrollador, y XSP (servidor web para ASP).

    Mono funciona bajo GNU/Linux y es compatible con Microsoft .NET, de forma que es bastante fácil migrar una aplicación de Windows a Linux. En el 2005 ha salido su versión 1.1.8, y se le augura un futuro muy prometedor. Algunas distribuciones de Linux, como Red Hat 9, Fedora Core 1 y Suse 9.0 ya incluyen Mono.
  * **NetBeans**. Entorno de desarrollo basado en Java y creado por Sun. Permite diseñar formularios o pantallas con AWT, control de las clases, crear componentes, etc. Es un entorno muy potente y recomendable si vamos a trabajar en Java.
  * **Eclipse**. Entorno de desarrollo creado por IBM, gratuito, y que soporta muchos lenguajes. Destacamos su capacidad para trabajar con Java.



6. PROGRAMACIÓN POR EVENTOS

La filosofía de programación para la creación de entornos gráficos difiere de la programación tradicional, y es similar a la programación orientada a objetos. En este caso recibe el apelativo de **programación guiada por eventos**.

En la programación guiada por **eventos** la ejecución del programa no es lineal, como en la programación clásica; en este caso escribimos bloques de código. Cada bloque se ejecuta cuando un evento lo requiere. Un evento puede ser provocado por el usuario, por el ordenador, por la aplicación, o por otras aplicaciones. Ejemplos de eventos pueden ser que el usuario pulse un botón, que se produzca un error en la aplicación, intercambio de datos entre varias aplicaciones, etc.

Programa Tradicional

Inicio
☐ Instrucción 1;
☐ Instrucción 2;
☐ Instrucción 3;
☐ Instrucción 4;
☐ Instrucción...;
☐ Instrucción n;
Fin.

Evento1. Objeto1

Inicio
☐ Instrucción 1;
☐ Instrucción 2;
Fin

Evento3. Objeto2

Inicio
☐ Instrucción 1;
☐ Instrucción 2;
Fin

En este paradigma de la programación los objetos son similares a los de la programación orientada a objetos (se componen de métodos y propiedades), pero además poseen **procedimientos de eventos**. Estos procedimientos de eventos constituyen un repertorio de eventos ante los que puede responder un objeto (al pasar el puntero sobre el objeto, pulsar el objeto, arrastrarlo, etc.).

7. COMPONENTES GRÁFICOS

Existen dos tipos de componentes gráficos muy importantes, como son los Formularios (*forms*) y los controles (*controls*).

Los **formularios** son las diferentes ventanas que componen nuestra aplicación. A estas ventanas le podemos asociar un menú, un archivo de ayuda, un fondo, y realmente se puede ver como un objeto contenedor. Un formulario da cabida a controles. Las aplicaciones pueden ser de dos tipos:

- Aplicaciones de una sola ventana donde sólo tenemos un formulario que contiene todos los controles necesarios para manejarla. Se suele usar para aplicaciones sencillas.
- Aplicaciones de múltiples ventanas, también conocidas como MDI (*Multiple Document Interface*). En este caso es necesario indicar cuál va a ser el formulario principal, o padre, y cuáles van a ser los formularios hijos. El formulario padre podrá ir creando o destruyendo los formularios hijos según los vaya requiriendo.

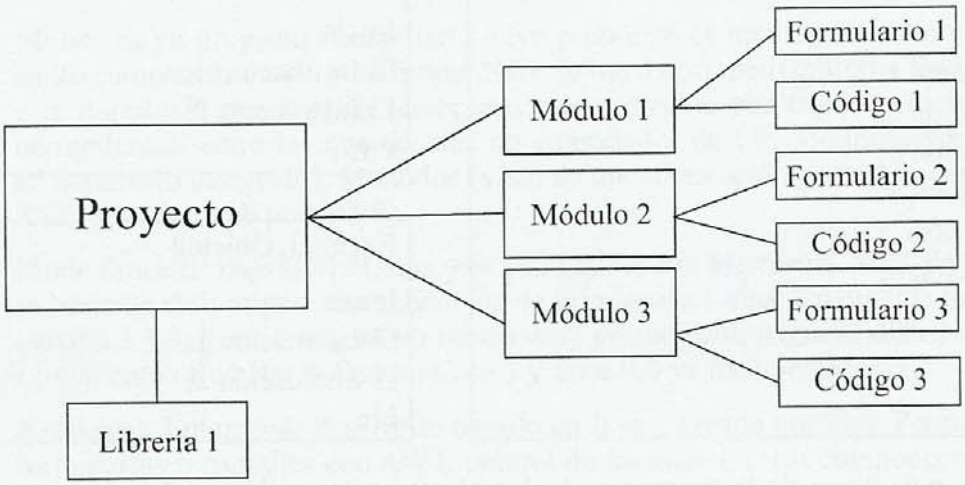
Los **controles** son objetos visuales con sus propiedades, métodos y eventos asociados. Estos entornos contienen gran cantidad de controles, normalmente agrupados por categorías. En cierta medida el éxito del producto depende de la cantidad y calidad de los controles suministrados con la herramienta. También el programador puede crear sus propios controles a partir de los ya existente usando la herencia.

8. ESTRUCTURA DE UN PROGRAMA BAJO UNA INTERFAZ GRÁFICA DE USUARIO

Un programa de estas características se trata de forma diferente a la programación tradicional. En este caso no tenemos un solo fuente, sino muchos. En este caso el código fuente de un programa se agrupa en proyectos. Un proyecto contiene los módulos y librerías que forman nuestro programa.

A su vez, cada módulo se divide en dos partes: uno o varios formularios con sus propiedades y los controles que contiene, y el código correspondiente a los eventos de estos formularios.

De esta manera, un programa de este tipo se compone de muchos archivos fuente, siguiendo el siguiente esquema:



9. HERRAMIENTAS

La mayoría de los entornos de programación modernos se componen de una serie de herramientas similares:

- **Inspector de objetos**: como su nombre indica nos permite examinar las propiedades, métodos y eventos definidos en cada objeto. También podemos alterar las propiedades, métodos y eventos y movernos a través de ellos.
- **Editor de código**: permite escribir el código correspondiente a un evento. Suelen ser editores sintácticos, con funciones como autocorrección y autocompletación del texto que escribimos.
- **Explorador de proyectos**: contiene la lista de módulos que componen nuestro proyecto. Desde esta ventana podemos abrir/cerrar módulos, acceder a un formulario determinado, a un control, etc.
- **Diseñador de formularios**: esta herramienta facilita la creación de formularios, inclusión de controles, alineación de los mismos, etc.
- **Editores de menús y barra de herramientas**: al igual que cualquier aplicación podremos diseñar nuestros propios menús y barras de herramientas.
- **Creador de informes**: Microsoft incorpora *Crystal Reports*, *Borland Report Smith*, etc. Estos entornos de programación incluyen herramientas para facilitar la creación de informes desde distintas fuentes y de forma casi automática.

- **Asistentes**: estas aplicaciones suelen tener muchos asistentes para ayudarnos en tareas como enlazar con una base de datos, diseñar un formulario, crear un control nuevo, etc.
- **Plantillas**: estos programas incluyen plantillas ya predefinidas para facilitar el trabajo al programador. Como plantillas nos suelen permitir crear aplicaciones estándar para un determinado sistema operativo, librerías, nuevos componentes, etc.
- **Generadores de ayuda**: incluyen ayuda que genera información sobre el propio proyecto. Normalmente se usa el HTML como lenguaje para crear esta ayuda.

10. EJEMPLO DE CREACIÓN DE UNA PEQUEÑA APLICACIÓN USANDO VISUAL BASIC 5.0

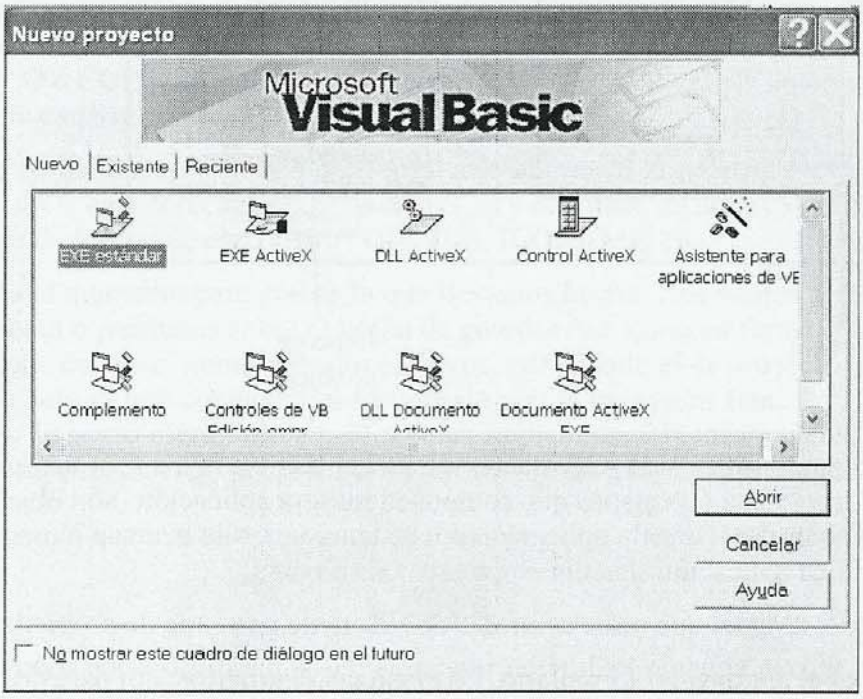
Para ilustrar la filosofía de la programación en entornos gráficos vamos a realizar una pequeña aplicación, en este caso, un visor de gráficos explicando paso a paso la creación de la misma.

La idea básica de este apartado es realizar un enfoque global del tema, que es demasiado extenso y requeriría un libro para sí solo. De todos modos, si quiere ampliar conocimientos de este tema le remito al tema 48 de este mismo temario.

He elegido Visual Basic por ser el entorno que cuenta con más difusión, aunque se puede adaptar a cualquier otro entorno sin demasiado esfuerzo.

Lo primero que debemos hacer es lanzar Visual Basic. Simplemente nos vamos al menú Inicio > Programas > Microsoft Visual Basic > Visual Basic 5.0[8].

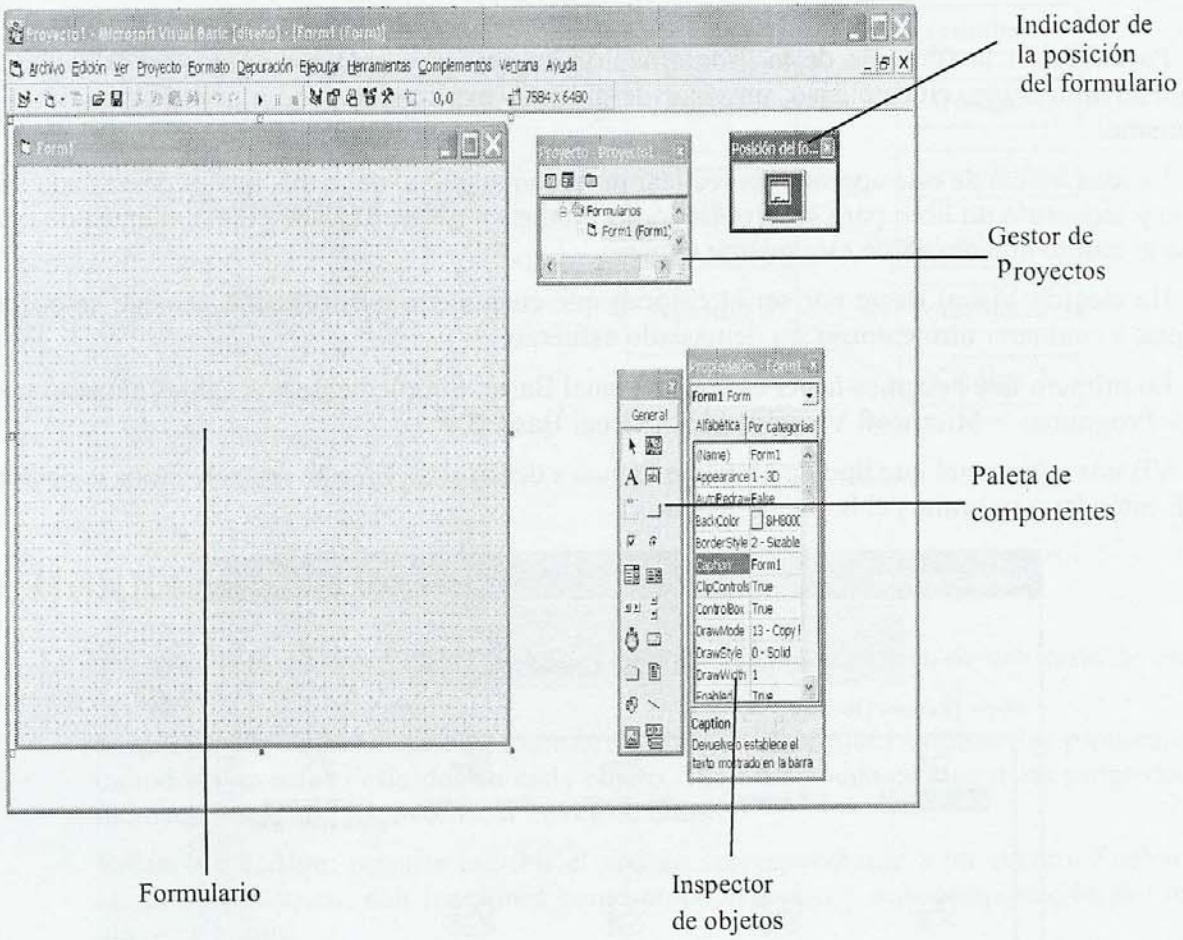
VB nos preguntará que tipo de proyecto vamos a desarrollar. En este caso elegimos la opción EXE estándar y pulsamos el botón de **Abrir**.



[8] Como convenio usaremos el símbolo mayor que (>) para indicar las opciones del menú que hemos de usar.

Una vez elegido el tipo de proyecto accedemos al entorno de programación. Vamos a comentar brevemente sus partes:

- **Inspector de objetos**: permite ver las propiedades, métodos y eventos asociados al componente seleccionado.
- **Paleta de componentes**: contiene todos los controles de los que dispone VB.
- **Gestor de proyectos**: permite movernos por los diferentes módulos que componen nuestro programa.
- **Formulario**: es la ventana sobre la que vamos a diseñar el programa que vamos a hacer.
- **Indicador de la posición del formulario**: como su nombre indica podemos situar el formulario en la pantalla a nuestro gusto.



Uno de los componentes más importantes de Visual Basic son los formularios o forms. Los formularios son la ventana o ventanas que componen nuestra aplicación. Son objetos contenedores que agrupan controles. Nuestra aplicación puede tener una sóla ventana o más de una. Como vamos a realizar una aplicación sencilla emplearemos sólo una.

Un formulario, al igual que otros controles de VB, tiene una serie de propiedades:

- **Name**: es el nombre del formulario. Lo empleará el programador para designar la ventana cuando queramos realizar operaciones sobre ella.
- **Caption**: título de la ventana. La mayoría de los controles de VB tienen dos nombres: el que usa el programador (*name*), y el título o nombre que se le da al usuario (*caption*).

- **Enabled**: activado. Si está establecido a TRUE podemos usar el control, si está a FALSE el control aparecerá en la pantalla pero no se podrá usar.
- **Height**: altura de la ventana medida en twips. 20 twips equivalen a un pixel.
- **Width**: ancho de la ventana medido en twips.
- **Visible**: si está a TRUE nuestra ventana podrá verse, por lo que si está a FALSE se ocultará a la vista del usuario.
- **MaxButton, MinButton**: están activas si queremos que aparezca el botón de maximizar y el de minimizar.
- **WindowState**: estado de la ventana, que puede ser 0 (normal), 1 (mininizado), 2 (maximizado).
- **Icon**: icono asociado al formulario que aparecerá cuando se minimice.

Existen muchas propiedades más, como el tamño de la fuente (*FontSize*), el dibujo de fondo del formulario (*Picture*), el color de la ventana (*ForeColor*), el estilo del borde (*BoderStyle*), la apariencia de la ventana (*Appeareance*), etc.

Un formulario tiene asociado muchos eventos, entre los que destacamos:

- **Load**: se activa al cargar el formulario.
- **Unload**: se llama al descargar el formulario.

Lo primero que vamos a hacer es darle un nombre a la ventana principal del programa. Buscamos la propiedad **Caption** y pulsamos en la caja que aparece a su derecha, donde le habrán designado un nombre por defecto (Form1). Lo cambiaremos por **Minivisor**. Como podrá comprobar, automáticamente se cambiará el nombre de la ventana.

A continuación vamos a asociar a nuestra aplicación algún icono. Vamos a la propiedad **Icon** y a la derecha nos aparecerá un botón con tres puntos (...). Pulsamos sobre ese botón una vez con el botón izquierdo del ratón y nos aparecerá un explorador desde el cual podremos elegir cualquier archivo ICO o CUR para representar nuestra aplicación. He buscado como icono un ojo que simboloza que nuestro programa es un visor.

Por último he cambiado el fondo del formulario alterando la palabra **Picture**, al igual que antes pulsamos en la caja derecha en los 3 puntos (...) y elegimos un archivo gráfico. VB permite un amplio rango de formatos, como BMP, GIF, JPG, ICO, WMF, etc.

Ahora es buen momento para grabar lo que llevamos hecho. Nos vamos al menú **Archivo > Guardar Proyecto** o pulsamos sobre el botón de guardar (un icono en forma de disco flexible). VB nos pedirá que demos el nombre de dos archivos, por un lado el de proyecto con la extensión **vbp**, y por otro lado el que contendrá el formulario con la extensión **frm**. En ambos casos hemos optado por llamarlo **ejemplolibro**. El archivo vbp contiene la información sobre todos los módulos que componen nuestro proyecto y el archivo frm las propiedades, métodos y eventos de nuestro formulario.

Vamos a probar nuestro programa. En principio no realiza ninguna acción, simplemente podremos mover la ventana, redimensionarla, minimizarla, maximizarla, etc. Para ejecutarla podemos pulsar el botón de ejecutar (similar al botón de play), pulsar **F5** o en el menú Ejecutar > Iniciar.

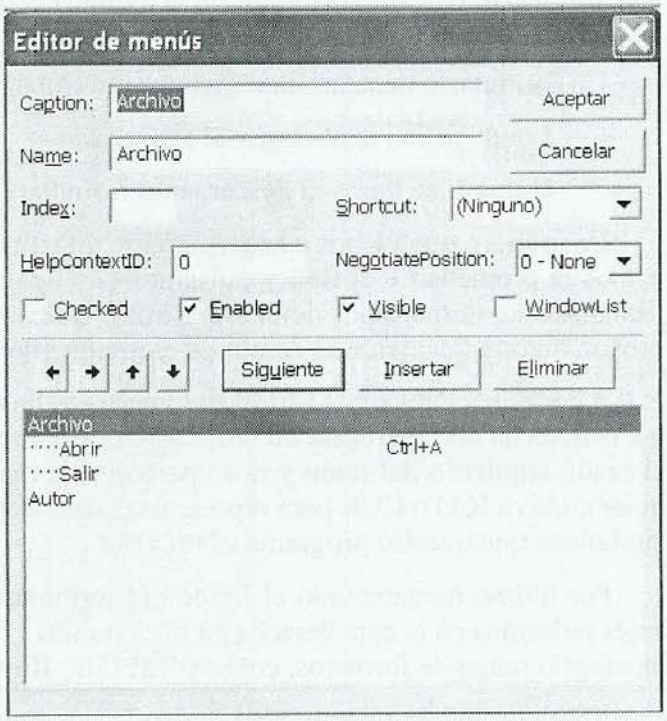
Al cerrar la aplicación volveremos al modo de edición de Visual Basic. Como ya habrá podido imaginar, VB dispone de dos modos de funcionamiento:

- El modo de **ejecución**, que se activa cuando ejecutamos la aplicación. En este modo podemos probar nuestro programa y comprobar si hace lo que debería.
- En el modo de **edición**, por defecto, podemos escribir código, modificar las propiedades de los controles, escribir eventos, etc.

El próximo paso va a ser añadir en el formulario un control para cargar la imagen que deseemos visualizar. Vamos a usar el control denominado **Image**. Este control contiene una imagen en un formato estándar que pueda leer VB (JPG, BMP, GIF, etc.). Para insertarlo lo marcamos en la paleta de componentes y posteriormente marcamos en el formulario dándole el tamaño que consideremos adecuado. Las propiedades de este control son similares a la de los formularios, con algunas diferencias no tenemos la propiedad Caption ya que no incluye ninguna etiqueta identificativa. La propiedad **Picture** nos da el nombre del archivo del que vamos a extraer la información.

El siguiente paso va a consistir en crear un pequeño menú para nuestra aplicación. Hemos de activar el editor de menús. Activamos la opción **Herramientas > Editor de menús** o pulsamos **CTRL+E**. En el editor de menús debemos insertar la siguiente información:

- **Caption**: título del menú.
- **Name**: nombre del menú.
- **Shortcut**: atajo de teclado para la opción del menú.



Dentro de los menús podemos crear submenús. En la zona de abajo tenemos botones para identar las opciones de forma que podemos crear submenús.

Vamos a crear un menú con el de la imagen. Con las opciones Archivos, dentro de archivo Abrir (ctrl + A) y salir. Y la opción Autor.

Cuando terminemos pulsamos el botón **Aceptar**.

El siguiente paso de nuestro proyecto va a consistir en crear una *barra de herramientas*. En las versiones Profesional y empresarial existe un control específico denominado Toolbar. Sin embargo, en la versión Personal no existe ese control y debemos crearlo combinando varios controles.

En primer lugar necesitaremos un objeto contenedor que tenga la propiedad align para poder situar el objeto en la posición que queramos de la pantalla. En este caso vamos a emplear un cuadro de imagen o picture box. Pulsamos con el botón izquierdo del raton en el control PictureBox y en el formulario dibujamos la dimensión y situación que queremos que tenga.

Posteriormente alteraremos la propiedad **Align**. Align puede tomar varios valores:

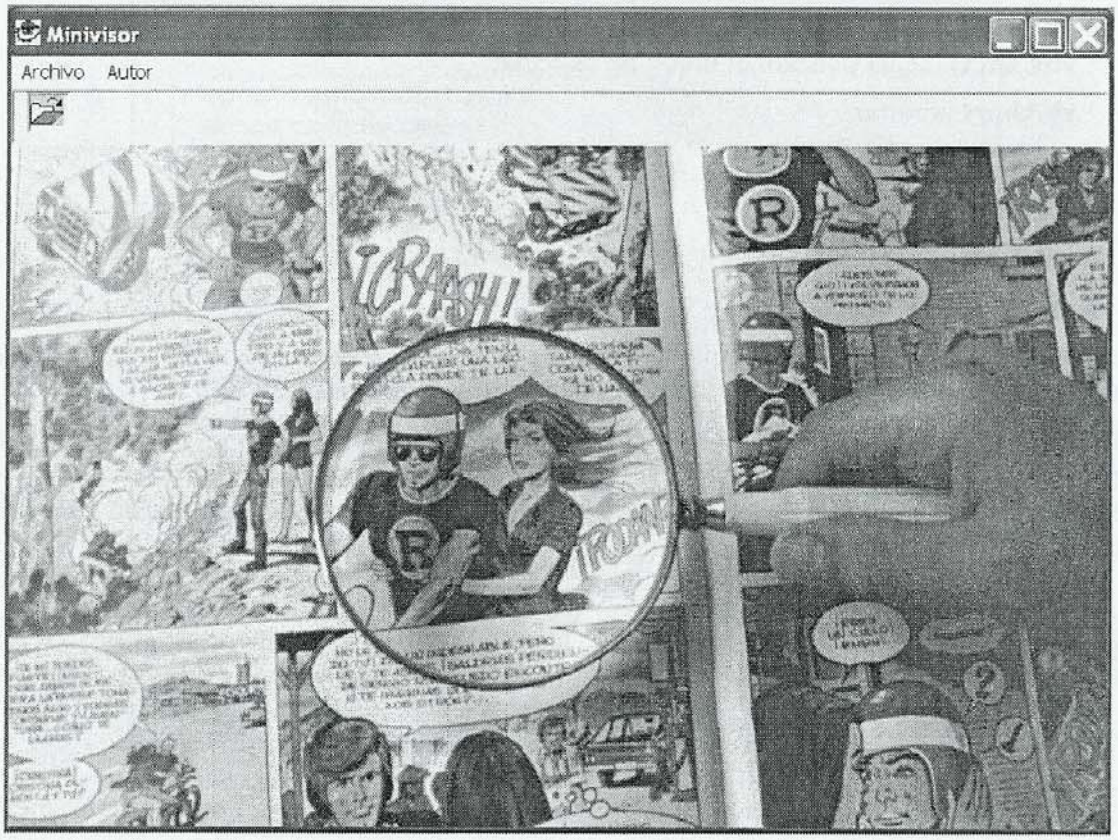
- **Ninguno, none o 0**: el control permanece en la posición en la que se ha situado inicialmente.
- **Arriba, top o 1**: el control ocupa la parte superior de la pantalla.
- **Abajo, botton o 2**: el componente se localiza en la parte inferior de la pantalla.
- **Izquierda, left o 3**: el objeto se va a la parte izquierda de la pantalla.
- **Derecha, right o 4**: el objeto se ubica a la derecha de la pantalla.

En nuestro caso le daremos el valor de **Top**. Simplemente seleccionamos el control PictureBox en la lista desplegable que aparece a la derecha de la propiedad Align y seleccionamos Align Top.

Ahora tendremos que insertar los botones en la barra de herramientas, hasta ahora vacía. Para realizar dicha empresa emplearemos el control conocido como **PictureBox**. Para agregar el botón de comando pulsamos en la paleta de componentes y, a continuación, vamos al formulario y sobre el PictureBox que acabamos de añadir le damos la dimensión adecuada al mismo.

Nuestra aplicación sólo va a tener un botón para abrir un archivo. A este control le ponemos como nombre (propiedad name) Babrir[9], y buscaremos un icono que será la propiedad **Picture**[10].

Si todo ha ido bien nuestra aplicación tendrá un aspecto similar al siguiente:



9 Normalmente en programación visual se suele bautizar a los controles con el nombre precedido de una letra que indica el tipo de componente al que pertenece. En este caso, B de botón y abrir por la acción que realiza.

10 En el CD de VB exite una carpeta denominada Graphics/bitmaps donde encontrará gran cantidad de gráficos disponibles.

Vamos a añadir un cuadro de diálogo, de forma que cuando el usuario cierre el programa se le pregunte si lo ha realizado por error. Existen muchos tipos de cuadro de diálogo ya predefinidos, aunque podemos crear los nuestros. Para crear un cuadro de diálogo usaremos la función **MsgBox (Mensaje, [Botones][, Título] [, ArchivoAyuda, Contexto])**.

En **Mensaje** escribimos la frase que queremos que muestre.

En **Botones** indicaremos qué opciones de respuesta tendrá el usuario. Las respuestas más usadas son:

- **0 o vbOKOnly**: botón de aceptar.
- **1 o vbOKCancel**: botón de Aceptar y Cancelar.
- **2 o vbAbortRetryIgnore**: terna anular, reintentar o ignorar.
- **3 o vbYesNoCancel**: terna sí, no cancelar.
- **4 o vbYesNo**: sí o no.
- **5 o vbRetryCancel**: retintentar o cancelar.

El título será el título del cuadro de diálogo.

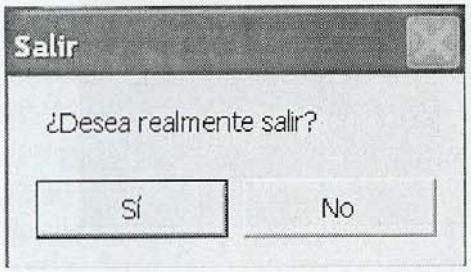
Los argumentos ArchivoAyuda y Contexto nos permiten asociar un archivo de ayuda sensible al contexto del cuadro de diálogo.

Esta función devuelve un valor según la opción que elija el usuario, que puede ser:

- **vbOk**: el usuario ha pulsado el botón de aceptar.
- **vbCancel**: se ha pulsado el botón de cancelar.
- **vbAbort**: abortar.
- **vbRetry**: reintentar.
- **vbIgnore**: ignorar.
- **vbYes**: sí.
- **vbNo**: no.

Una vez explicado el funcionamiento vamos a escribir el código necesario para que cuando el usuario desee salir el programa pida confirmación.

Ahora queremos que cuando el usuario salga del programa usando el menú se le solicite confirmación. En este caso simplemente solicitamos la descarga del formulario usando el evento UnLoad. Para escribir el código sólo tenemos que pulsar en el menú del formulario en la opción salir que se encontraba en archivo.

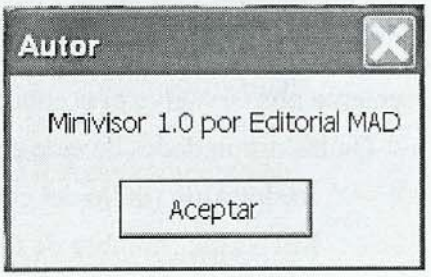



```
Private Sub Salir_Click()
        If MsgBox("¿Desea realmente salir?", vbYesNo,
        "Salir") = vbYes Then
    Unload Me
  End If


End Sub
```

El siguiente paso va a consistir en ser un poco ególatras y ponernos como autores de este programilla. Existe muchas formas de realizar esta tarea, pero la más sencilla es usando un cuadro de diálogo.

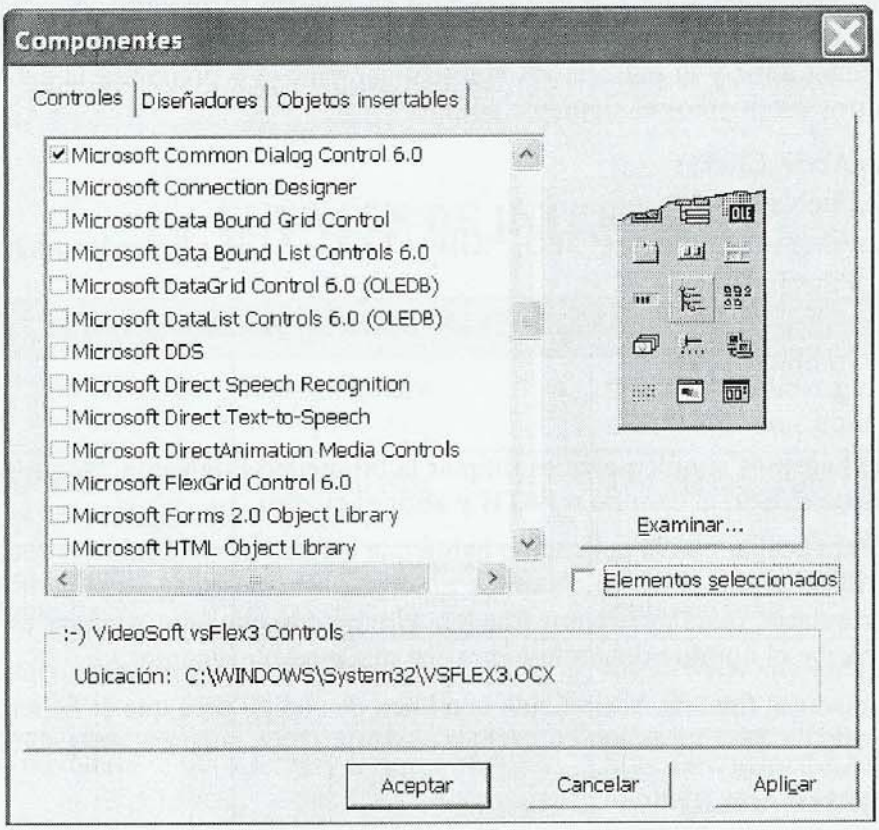
Al igual que hicimos anteriormente en el modo de edición, abrimos en el menú la opción de autor. VB nos creará automáticamente la cabecera para que escribamos el código dentro. Seguro que puede adivinar lo que vamos a escribir:




```
Private Sub Autor_Click()
respuesta = MsgBox ("Minivisor 1.0 por Editorial MAD",
vbOKOnly, "Autor")
End Sub
```

En este caso el cuadro de diálogo sólo tiene el botón de Ok y la respuesta la guardamos en la variable "respuesta" aunque no nos interese su resultado.

Ahora deberíamos crear la opción Abrir, que va a permitir ver un archivo gráfico seleccionado por el usuario. Para realizar dicha función debemos usar el control denominado *Common Dialog* (Diálogo Común). Este componente contiene los diálogos más usados en Windows, como el diálogo de abrir archivo, guardar como, imprimir, etc. Es posible que este diálogo no aparezca en la paleta de componentes; pruebe a ampliarla para visualizarlo. Si sigue sin aparecer pulse el botón derecho del ratón sobre la paleta, seleccione la opción **Componentes** y busce Microsoft Common Dialog Control[11]; márquelo y pulse el botón Aceptar para añadirlo a su paleta.



Este control no es visible en tiempo de ejecución. Dicho de otra forma, al iniciar el programa se carga y hasta que no lo reclamamos permanece "oculto". El funcionamiento de cada uno de estos componentes es distinto según su función. Nosotros vamos a estudiar el cuadro de Abrir.

---

11 En algunas versiones limitadas de VB este control no está disponible, por lo que deberíamos crearlo nosotros o buscar una versión que lo contenga.

Para mostrar este cuadro debemos asignar a la propiedad Action el valor 1. Este cuadro nos permite seleccionar un archivo de nuestro ordenador para posteriormente abrirlo o, en nuestro caso, mostrar el gráfico asociado al mismo. Este componente realmente no abre el archivo; simplemente nos devuelve el nombre del archivo que el usuario ha seleccionado.

De las propiedades de este control destacamos:

- **DialogTitle** (título del cuadro de diálogo): es el nombre del cuadro; por defecto será abrir.
- **FileName** (nombre de fichero): es el nombre inicial que aparecerá en la caja de nombre de archivo, al cerrar el cuadro nos dará el nombre del archivo seleccionado.
- **Filter** (filtro): filtro de los archivos a cargar. Se compone de dos elementos: por un lado la descripción textual del tipo de archivo y, por otro, el nombre del fichero. Se separan usando el operador "|". Si queremos incluir más de un argumento hemos de usar el operador ";".
- **FilterIndex** (índice del filtro): es un número entero que indica qué filtro actuará por omisión.
- **Flags** (banderas): nos permiten seleccionar características del archivo a seleccionar. Estos valores se describen en la ayuda de VB. Vamos a destacar el valor OF_NMUSTEXIST, que obliga a que el nombre del archivo exista, y OF_NPATHMUSTEXIST, que indica que el camino o path debe existir.

Después de este pequeño resumen teórico vamos a pasar a la acción. En primer lugar insertaremos un *Common Dialog* en nuestro formulario. A nuestro diálogo lo bautizaremos con el nombre *Dabrir* (Diálogo abrir). En el menú Archivo buscaremos la opción abrir y la pulsaremos automáticamente. Vb preparará la cabecera para que nosotros escribamos el siguiente código:

```
Private Sub Abrir_Click()
    Dabrir.FileName = ""
    Dabrir.Filter = "Gráficos  (*.JPG, *.GIF) | *.JPG ; *.GIF | Todos los archivos | *.*"
    Dabrir.FilterIndex = 1
    Dabrir.Flags = OFN_FILEMUSTEXIST or OFN_PATHMUSTEXIST
    Dabrir.Action = 1
End Sub
```

Aquí lo que hacemos simplemente es limpiar la propiedad FileName, elegir los filtros, decir que el fichero debe existir, el camino o PATH y abrir el cuadro.

Para acabar con la aplicación habrá que asignar el resultado del cuadro de diálogo anterior a la propiedad **FileName** a un objeto de tipo Image que vamos a insertar en el formulario, que llamaremos Imagen. Un método que vamos a usar es **LoadPicture**, que recibe el nombre del archivo gráfico que queremos cargar.

Hemos de añadir al final de Abrir_Click una línea de código para que el fichero seleccionado aparezca en la pantalla. De esta manera el código de Abrir_Click quedaría de la siguiente manera:

```
Private Sub Abrir_Click()
   DAbrir.FileName = ""
   DAbrir.Filter = "Gráficos  (*.JPG, *.GIF) |*.JPG;*.GIF| Todos los archivos |*.*""
   DAbrir.FilterIndex = 1
   DAbrir.Flags = OFN_FILEMUSTEXIST Or OFN_PATHMUSTEXIST
   DAbrir.Action = 1
   If DAbrir.FileName <> "" Then Imagen.Picture = LoadPicture(DAbrir.FileName)
End Sub
```