39
24.

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
CAMPUS ARAGÓN

"DESARROLLO DE SISTEMAS CON VISUAL OBJECTS"

T E S I S

QUE PARA OBTENER EL TITULO DE
INGENIERO EN COMPUTACION
P R E S E N T A

JUAN CESAR MARTINEZ PEREZ

ASESOR:ING. ERNESTO PEÑALOZA ROMERO

MÉXICO 1997

PAGINACION VARIA

COMPLETA LA INFORMACION

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES

CAMPUS ARAGON

TESIS TITULADA

DESARROLLO DE SISTEMAS CON VISUAL OBJECTS

POR EL ALUMNO

JUAN CESAR MARTINEZ PEREZ

A mis padres

" No encuentro la forma de agradecer
el apoyo que siempre me han brindado,
solamente hago también suyo este logro.
Gracias por todo. Los quiero mucho "

A mi familia

" Deseo que sientan como suya esta meta
que he alcanzado, porque siempre me han
ayudado. Gracias a todos "

A mis hermanas

" Yo se que ustedes también van a
sentir la alegría de terminar una carrera.
Cuenten conmigo siempre "

INDICE

CAPITULO III

CAPITULO IV

INTRODUCCION

La computación es cada vez más importante en la vida de los seres humanos y por lo tanto son más las personas que ocupan un sistema, por tal motivo los usuarios de sistemas se introducen mas al fondo de este gran universo que es la computación. Al conocer más, los usuarios exigen que los sistemas sean cada vez más completos, amigables, de rápida construcción y con un mantenimiento más sencillo. En estos últimos años los productores de sistemas están utilizando para su construcción, herramientas de programación que soporten la Orientación a objetos, que es una nueva forma de analizar y diseñar los sistemas utilizando técnicas que agilizan su construcción y le dan un peso especial al análisis, simplificando la programación. En la actualidad los constructores de sistemas que utilizan estas herramientas de programación, necesitan utilizar el análisis y diseño orientado a objetos, pero muchos constructores de sistemas mezclan el análisis y diseño estructurado con una herramienta de programación que soporta orientación a objetos, este fenómeno se observa porque la mayoría de los ingenieros no quieren invertir tiempo en aprender estas nuevas técnicas de análisis y diseño, y en consecuencia limitan las ventajas que puede obtener con estas herramientas de programación.

Es difícil que los desarrolladores de Software cambien la forma en que construían sistemas, porque estas técnicas de análisis y programación son muy diferentes a las que se estaban ocupando, y además tienen que olvidar algunos conceptos que ocupaban, para aprender otros que asemejan a los conceptos que utilizamos cotidianamente, haciendo más fácil el entendimiento a otras personas que no son expertas en esta materia de los diagramas y procedimientos que realizamos al construir los sistemas.

Este trabajo ayuda a los Ingenieros que quieren cambiarse a esta nueva tecnología a entender estos nuevos conceptos, y emplearlos con una herramienta que ocupa la orientación a objetos como lo es Visual Objects para la construcción de sistemas, además muestra las ventajas que se obtienen haciendo un buen análisis y diseño orientado a objetos para aprovechar al máximo los beneficios que ofrecen estas herramientas de programación, y sirve como guía para los que apenas empiezan a hacer análisis y diseño de sistemas orientados a objetos, mostrando las diferencias que existen entre las dos formas de analizar y diseñar sistemas. Se hace un enlace del Análisis, Diseño y Programación Orientado a Objetos con una herramienta de programación que se llama Visual Objetcs para que además de aprender los conceptos que se requieren, puedan evaluar Visual Objects y así decidir si cumple con lo que están buscando, y se involucren en este nuevo ambiente difícil de entender para algunos programadores.

Cuando queremos empezar a programar con alguna herramienta de programación orientada a objetos, no existe un documento en el que nos podamos basar para adquirir un panorama general de lo que podemos hacer con esta tecnología nueva, y primero tenemos que consultar libros de programación orientada a

objetos, después documentamos acerca de lo que ofrece Windows para este nuevo ambiente, consultar un libro de análisis y diseño orientado a objetos (de los que existen varios enfoques) y consultar los manuales de la herramienta. Teniendo tal cantidad de información por investigar y aprender ; además del poco tiempo que tiene para desarrollar, el diseñador de sistemas se preocupa más por la programación que por utilizar nuevas formas de analizar y diseñar los sistemas, ocupando las técnicas estructuradas y programando con herramientas orientadas a objetos, este trabajo trata de aportar una guía en la que los ingenieros que empiecen a programar con estas nuevas herramientas, no tengan que buscar en varios documentos la información básica que deben conocer para entender esta nueva filosofía de producción de software.

Las ventajas que se pueden obtener empleando la programación Orientada a Objetos son : la construcción de sistemas más rápidamente, mejor presentación del software producido, sistemas más amigables, mantenimiento mas sencillo y los cambios de estructura del sistema no es complicado.

Por todas estas ventajas el futuro es la orientación a objetos y todos los nuevos productos están adoptando esta tendencia, y los que no se adapten al cambio o no quieran cambiarse a esta nueva filosofía desaparecerán en poco tiempo.

El primer capítulo del trabajo define los conceptos para la utilización de los objetos dentro de la construcción de sistemas y el ambiente que ofrece Windows para hacer más amigables y con una mejor presentación los sistemas que se construirán.

En el segundo capítulo hablo de un concepto que es muy usado dentro del ambiente de los objetos, el Repositorio. Dentro de este capítulo describo los visualizadores que tiene Visual Objects y la forma en que trabajan con el repositorio en el que almacena la información.

En la tercera parte del trabajo me ocupo del Entorno integrado de desarrollo que ofrece Visual Objects, describiendo los editores con que cuentan los programadores.

Hablo de multimedia en una parte que se le dedicada a uno de los últimos adelantos que ha integrado la computación a sus sistemas enriqueciéndolos con la mezcla de voz, imágenes y datos.

En el último capítulo describo en forma general el análisis y diseño orientado a objetos propuesto por James Martin en su libro "Análisis y Diseño Orientado a Objetos" y menciono las diferencias que existen entre esta nueva técnica de análisis y diseño con la que se ocupa para la programación estructurada.

CAPITULO

CONCEPTOS FUNDAMENTALES

1.1 PROGRAMACION ORIENTADA A OBJETOS

La Programación Orientada a Objetos(POO) surge de la inquietud de los programadores por realizar más rápidamente sistemas, ya que con la programación estructurada la producción es muy pobre y lenta en comparación al adelanto del hardware que cada vez avanza con pasos mas grandes.

Era necesario descubrir una técnica en la que se pudieran producir sistemas en serie como en las fábricas y que se trabajara simulando al mundo real en el que se ocupan objetos. Los que pensaron en la programación orientada a objetos trataron de asemejar a la industria automotriz que fabrica objetos separados, pero en conjunto realizan una tarea que es la de transportar.

La programación orientada a objetos realiza una tarea determinada con objetos independientes que actúan con sus propios datos sin que otros objetos puedan manipularlos.

Una de las grandes ventajas de la programación orientada a objetos es la reutilización de código, en donde los programadores no empiezan de cero a trabajar, pueden utilizar procedimientos de otros programas y solo modificar la parte que les interese o anexarle módulos al programa existente. Uno de los conceptos fundamentales de la Programación Orientada a objetos es la herencia, en la que se da la reutilización de código. Este método de reutilizar código es una de las tácticas clave de esta técnica de producción. Además de esta ventaja la Programación Orientada a Objetos ofrece un modelo más natural del mundo real y la programación es en menos líneas de código, menos sentencias de bifurcación y los módulos son más comprensibles.

1.1.1 Objeto, Mensaje y Método

En la Programación Orientada a Objetos, sólo existen objetos, los cuales contienen un conjunto de datos y procedimientos; los objetos son módulos en los que existen operaciones y datos, en donde la única posibilidad de acceder a estos datos es solicitar que el mismo objeto ejecute un procedimiento para modificarlos. Un objeto no puede acceder directamente a los datos de otro. Los objetos son entidades que tienen atributos y formas de comportamiento particulares.

Un objeto es un componente del mundo real que se transforma al dominio de la computación y con esto se pretende reducir los tiempos de desarrollo y hacer software en serie como en las fábricas industriales.

En una aplicación pueden existir dos clases de objetos :

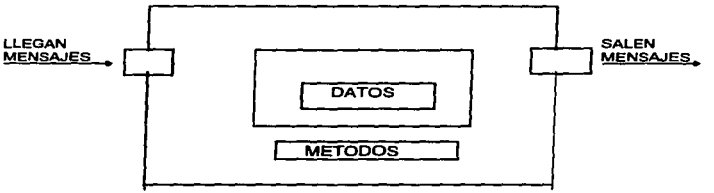
- Los pasivos, que actúan solamente bajo petición.

- Los activos, que efectúan el seguimiento de los sucesos que actúan en una aplicación y se comportan de manera autónoma. Cuando un usuario quiere que un sistema con objetos realice una tarea, el objeto recibe un mensaje del usuario o de otros objetos, estos mensajes son solicitudes que le piden al objeto que se comporten de alguna forma. Los mensajes son recibidos, interpretados y respondidos por los objetos.

Cuando un objeto emite un mensaje, no necesita saber como realizará la tarea el receptor de este mensaje. Lo que siempre tienen es un argumento que identifica al objeto que recibe el mensaje. Todos los objetos tienen un selector de mensajes, que es la parte que dice al objeto lo que ha de hacer, y al conjunto de mensajes que un objeto puede responder se le denomina protocolo.

Cuando un objeto recibe un mensaje lo identifica como una orden en la que se le indica lo que tiene que hacer, a esta acción se le llama método y estos métodos residen en el objeto y determinan cómo debe de actuar el objeto cuando recibe un mensaje.

Un mensaje es una orden que se le da a un objeto para que realice algo, y mientras tanto el que hace la llamada espera hasta que el receptor lo haya hecho y le proporcione una respuesta, es el medio de comunicación con un objeto, el cual determina su comportamiento.



1.1.2 Tipos de métodos

Los métodos son funciones, acciones que los objetos son capaces de ejecutar; y la diferencia con las funciones normales es que estan fuertemente asociadas al objeto a que pertenecen , de modo que no pueden ejecutarse de otro modo distinto. Según la función que realizan, podemos hablar de algunos métodos especiales:

Constructor: Es el método que crea una nueva ocurrencia del objeto.

Destructor: Es un método que destruye un objeto previamente instanciado.

Amigo: Es un método público que puede acceder a cualquier dato de cualquier objeto, independientemente del objeto donde se haya declarado. Los métodos Amigo rompen uno de los principios fundamentales de la Programación Orientada a Objetos que es la encapsulación que veremos mas adelante.

Virtual: Es un método que no realiza en sí mismo acción alguna, su única finalidad es facilitar la herencia. Las clases abstractas están compuestas de métodos virtuales.

Demonio: Es un método que no se ejecuta a través del mensaje enviado por otro método, sino a través de algún evento que se produce de forma no reglamentada.

1.1.3 Clases

Las clases son una agrupación de objetos con características similares, son una descripción de un conjunto de datos casi idénticos. Estas clases se emplean para aglutinar todas las características de un conjunto de objetos en una entidad abstracta de la que podamos instanciar (extraer, modelar) ocurrencias reales de la misma, es decir objetos. Instanciar, no es más que crear objetos, los cuales están agrupados conforme a sus características de clases.

En las clases existe una jerarquía en donde una clase puede tener una superclase y una subclase. Una subclase puede ser una versión especializada de la clase, de la cual se modificó información pero no se eliminó nada. Esto significa que una subclase tiene más que su clase o su superclase, no hay forma de quitarle, simplemente por que no es necesario.

Una clase puede ser abstracta, esto quiere decir que no crearemos ninguna ocurrencia física. Su cometido es facilitar la herencia, simplificando la organización jerárquica de clases.

La clase base es la que ocupa la posición superior dentro del árbol que esquematiza una jerarquía, esta clase no tiene antecesores o superclases.

Un concepto contrario al anterior es el de la clase derivada, en el que siempre existe un antecesor, no es la primera en el árbol jerárquico, en algunas ocasiones se denomina clase heredada.

1.1.4 Herencia

Es una herramienta que sirve para organizar, construir y emplear clases reutilizables, sin la herencia, cada clase sería una unidad independiente y cada una se desarrollaría partiendo de cero.

La herencia hace posible definir nuevo software tomando como base algo semejante a lo que queremos crear. Con esta ventaja ahorramos mucho tiempo al no repetir códigos y además incrementamos el grado de congruencia en todo el sistema. Gracias a la herencia, numerosas clases pueden compartir datos y programas con tan solo definirlos en la clase de la que descienden. Es uno de los pilares básicos de la POO.

La herencia sigue exclusivamente una dirección, por tanto un hijo hereda propiedades y métodos de su padre, pero no a la inversa. Por eso se dice que la herencia es unidireccional y además es el aspecto más importante de la organización jerárquica de los objetos como estructuras organizadas.

Los programas orientados a objetos constan de árboles o jerarquías de clases que, por medio de la subclasificación, llegan a ser más específicas.

Herencia simple y múltiple son dos tipos de mecanismos de herencia utilizados normalmente en la POO. En la herencia simple, una subclase puede heredar datos y métodos de una clase así como añadir o sustraer comportamiento por si mismo. La herencia múltiple se refiere a la posibilidad de una subclase de adquirir los datos y métodos de más de una clase.

1.2 CARACTERISTICAS DE LA POO

1.2.1 Encapsulación

La encapsulación es el término que describe el conjunto de métodos y datos dentro de un objeto de forma que el acceso a los datos se permite solamente a través de los propios métodos del objeto. Los datos son variables, por tanto dichas variables son locales al objeto y son inaccesibles desde el exterior, lo único que podemos hacer con él es interaccionar para ejecutar sus métodos.

La comunicación entre objetos se da solamente por medio de mensajes entre ellos.

1.2.2 Polimorfismo

El polimorfismo es tener dos métodos con el mismo nombre que ejecutan acciones distintas, un mismo mensaje puede originar acciones diferentes al ser recibido por diferentes objetos.

Con el polimorfismo un usuario puede enviar un mensaje genérico y dejar los detalles exactos de la realización para el objeto receptor sin tener que detallar el mensaje a cada receptor que active.

1.2.3 Ligamiento Dinámico

Para los sistemas Orientados a Objetos es deseable que puedan crear objetos nuevos y modificar la estructura de datos y métodos de los ya creados en tiempo de ejecución.
La ligadura es el proceso de entrelazar un programa de forma que existan todas las conexiones apropiadas entre sus componentes.
Los lenguajes Orientados a Objetos (00) emplean una estrategia llamada ligadura dinámica o tardía, en donde el proceso de ligadura ocurre cuando el programa se esta ejecutando y no antes de ejecutarse como la ligadura estática.

El proceso de ligadura es el mismo en la estática y en la dinámica, pero la ligadura ocurre posteriormente cuando el programa se está ejecutando y se hacen las conexiones entre el método y los datos locales con el objeto. La ligadura ocurre en el último momento posible.

En el caso de Visual Objects (VO)estamos ante un compilador inteligente que genera pseudo código cuando es necesario, lo que posibilita la creación y manipulación de símbolos en tiempo de ejecución, por ello Visual Objects está especialmente dotado para el ligamiento dinámico.

1.2.4 Persistencia

La persistencia se refiere a la permanencia de un objeto, es decir, al tiempo durante el cual se asigna espacio y permanece accesible en la memoria de la computadora.

Los objetos se crean cuando se ejecuta un programa y se destruyen cuando se termina esta ejecución, es necesario por seguridad y flexibilidad, que estos objetos se puedan almacenar en una base de datos 00 entre ejecuciones de un programa.
La capacidad de almacenar los objetos permite que los objetos sean compartidos en un entorno distribuido. Una base de datos 00 hace posible la persistencia y permite que solamente los objetos utilizados activamente sean cargados en memoria. Se dice que un objeto posee persistencia si continúa existiendo después que ha terminado el programa de la aplicación y esto es posible con las bases de datos 00 que son diferentes a las tradicionales.

En estas bases de datos 00 se pueden almacenar, modificar y recuperar por orden del programa de la aplicación objetos que pueden ser tan simples como números y cadenas o tan complejos como las especificaciones completas de un circuito electrónico.
Una de las ventajas que las bases de datos Orientadas a Objetso que tienen sobre las tradicionales, es su capacidad para representar como objeto cualquier entidad del mundo real.

1.3 WINDOWS

Los sistemas desarrollados bajo DOS están quedando atrás, porque los usuarios requieren de ambientes más amigables, los que se asemejen más al mundo real y no a programas de computadora.

A los usuarios que no son expertos en computación, les resulta más fácil trabajar en ambientes gráficos. Estos ambientes gráficos son implementados en plataformas gráficas, como Windows que además de ser un ambiente gráfico también es multitarea, ya no es procedural como DOS.

En Windows los estados de espera son propiedad del sistema y no de las aplicaciones. Los procesos también llamados eventos siempre se enlazan o desencadenan desde el sistema aunque se ejecuten en la aplicación. A esto se le llama orientación al evento.

El sistema en Windows siempre está interrogando sus diferentes fuentes de entrada (teclado, ratón, puertos) y, recibe y reenvía cualquier orden en forma de evento, quedando inmediatamente liberado y, por consiguiente, dispuesto a recibir la siguiente.

Las aplicaciones disponen de su propia cola de mensajes, desde donde se irán tratando los mismos por orden de entrada.

El entorno Windows para los desarrolladores en PC es el futuro por la facilidad con la que se pueden utilizar sus herramientas que facilitan la interacción de los programas con el usuario.

Es necesario, desarrollar con lenguajes que utilicen estas ventajas de Windows y además poder conjuntar la POO con estas poderosas herramientas que Windows ofrece.

Los programadores estaban desperdiciando el ambiente gráfico que Windows ofrece y es por eso que muchos lenguajes están cambiando a este ambiente más amigable para el usuario. No es fácil el cambio, principalmente para los desarrolladores bajo ambiente DOS y una programación procedural, pero es necesaria porque el mercado lo reclama.

1.3.1 Entorno de Trabajo Gráfico GUI (Interfaz de Usuario Gráfico)

Un GUI es el panel de control que gestiona la interacción entre el entorno de sistema operativo, la aplicación subyacente y el usuario. Es un entorno de trabajo gráfico, en el que las aplicaciones se ejecutan en modo gráfico dentro de un plano de coordenadas variables llamado ventana o caja.

En un GUI no hablamos de caracteres como en un entorno de texto, sino que tratamos con puntos con coordenadas (pixels). Windows trata con 640 X 480 puntos y 16 colores en su más baja definición, es decir 307 200 puntos, los cuales tienen dos coordenadas y un color concreto.

Los GUI orientados a objetos hacen que las aplicaciones se parezcan mucho más a las funciones del mundo real y cada vez menos a los procesos de programación. Una gran parte del campo de desarrollo de la mayoría de los GUI está ya basado en objetos y se aproxima mucho a los mecanismos básicos orientados a objetos.

El entorno de desarrollo de la mayoría de los GUI consta de 4 componentes principales:

- Sistema de ventanas.
- Modelo de imágenes.
- Interfaz del programa de aplicación.
- Conjunto de herramientas y marcos estructurales para crear interfaces y desarrollar aplicaciones integradas.

Los sistemas de ventanas permiten a los programadores visualizar aplicaciones múltiples al mismo tiempo, estos sistemas de ventanas incluyen herramientas de programación para construir ventanas desplazables y capaces de dimencionar menús, cuadros de diálogo y demás elementos de la presentación.

Un modelo de imágenes define la forma en que se crean en pantalla los tipos de letra y los gráficos. Por ejemplo, los modelos de imágenes manipulan el estilo y el tamaño de letra en un procesador de textos o las curvas y líneas en un programa de diseño.

El API (interfaz del programa de aplicación) es un conjunto de funciones de lenguaje de programación que permiten al programador especificar la forma en que la aplicación actual controlará los menús, las barras de desplazamiento y los iconos que aparecen en la pantalla. Al igual que los modelos de ventanas, los API se emparejan con determinados GUI.

Existe un estándar en el interfaz gráfico de usuario pero en los sistemas de ventanas todavía no existe.
Son 3 grupos de GUI que ejercen el dominio:

- System Application Architecture (SAA) de IBM, que incluye principalmente a Windows y PM de Microsoft.
- Sistemas Unix, construidos alrededor de XWindow Next.
- Macintosh.

1.3.2 Interfaces de Documentos

MDI (Multiple Document Interface)
Esta técnica posibilita el trabajo con varias estructuras de datos a la vez en una única carga del ejecutable, se diferencia de la multitarea por el hecho de que ahorra recursos del sistema porque sólo se carga un sólo ejecutable. Además lleva, casi siempre, implícita la gestión dinámica del menú principal con las ventanas disponibles, para poder pasar de unas a otras sin problemas.

SDI (Single Document Interace)
Es el que utiliza DOS, maneja un solo documento a la vez.

1.4 TECNICAS DE INTERCAMBIO DE INFORMACION

En un entorno multitarea como lo es Windows, el intercambio de datos entre diversas aplicaciones activas se puede plantear de 3 maneras distintas:

1.4.1 Portapapels

El portapapeles, permite el intercambio de bloques de datos. Por ejemplo copiar un párrafo de texto entre dos procesadores bien diferentes como Word y Word Perfect es tan sencillo como dar la orden de copiar en uno y de pegar en otro. Automáticamente el bloque viaja pasando por el portapapeles de Windows. Lógicamente hay un control automático del tipo de dato que se copia (gráfico, texto, valor numérico) con el fin de evitar posibles errores de pegado.

1.4.2 DDE (Intercambio Dinámico de Información)

El DDE es el intercambio dinámico de datos que se emplea para enlazar los datos que manejan dos o más aplicaciones en tiempo de ejecución, y se basa en el concepto de cliente-servidor. Por ejemplo, permite enlazar un documento en Word con una hoja de notas de la agenda de Lotus; al escribir en Word vemos como se va actualizando la hoja de notas en Lotus en cada pulsación del teclado. Otra de sus utilidades puede ser el enlace de una aplicación gestora de una base de datos como Visual Objects o Fox Pro, con un procesador de textos como Word Perfect, de manera que se posibilite la realización de documentos personalizados, que toman parte de su información desde la base de datos.

1.4.3 OLE (Objetos Ligados e Incrustados)

La tecnología OLE permite incrustar los resultados de una aplicación en los procesos de otra. Por ejemplo, incluir el resultado de un documento de Word en una hoja de cálculo Excel. La utilidad de OLE se percibe a la hora de hacer informes de gestión que implican diversas áreas de proceso. Prepara un documento escrito que incluye un gráfico, se puede asignar un área de nuestro documento al resultado de un proceso en Excel sobre unos datos. Sin OLE, usando el portapapeles, habría que hacer el gráfico, capturarlo en el portapapeles y, después, pegarlo en el documento. Cualquier cambio o ajuste posterior en alguna de las variables que conforman el gráfico, obliga a rehacer la operación. Con OLE, lo que se incrustaría en un área de nuestro documento , no es un gráfico, sino el resultado de otra aplicación, estableciéndose un enlace estable, que incluiría modificaciones y ajustes.

1.4.4 DLL (Librerías Preenlazadas Dinámicas)

Las DLL son librerías preenlazadas dinámicas, de uso muy extendido en Windows, contienen código complicado y preenlazado directamente utilizable por cualquier aplicación Windows que la requiera.

Son creadas por normas estándar desde cualquier lenguaje que lo permita y pueden ser utilizadas por cualquier programa Windows.

Un archivo DLL contiene código que el ejecutable carga y descarga de modo dinámico en función de sus necesidades, pudiéndolo compartir con otros ejecutables en tiempo real, esto gracias a la posibilidad de Windows de trabajar con multitarea.

Con esto se ahorra memoria, ya que una función estará cargada en memoria una sola vez, independientemente de los programas que la estén utilizando.

Los módulos más importantes de Windows son DLL y son usados al mismo tiempo por todas las aplicaciones que son ejecutadas bajo Windows.

Con la multitarea, podemos tener activas pequeñas aplicaciones cuyo cometido es lanzar tareas de impresión, envío de faxes, copias de seguridad, etc. Estas utilidades harán ejecutar otras aplicaciones por medio de enlaces DDE y se activará cualquier clase de proceso.

1.5 CLIENTE /SERVIDOR

Es un modelo de trabajo para organizar el acceso a los datos que nuestras aplicaciones realizan. Con Cliente/Servidor se organizan los componentes de un sistema de información, de forma que entre ellos mantengan una clara diferenciación de roles entre los que proveen servicios a los sistemas y los que son clientes de dichos servicios.

Este modelo para el desarrollo de aplicaciones afecta al Hardware y Software. Dentro del Software incluimos a los sistemas operativos y las bases de datos.

Es muy importante cuidar el diseño de este modelo porque si falla, podemos desperdiciar muchos recursos a la hora de poner en marcha nuestro sistema. Las ventajas de utilizar Cliente/Servidor son muchas y una de las principales es que se trata de un sistema abierto en el que se puede incorporar material de distintos fabricantes.

Otra ventaja es que el cliente y el servidor a la hora de procesar la información, se distribuye el trabajo en las dos partes, estos procesos al ser distribuidos se agilizan y se termina con los cuellos de botella y con los congestionamientos.

Con el modelo Cliente/Servidor interaccionan dos o más aplicaciones con diferentes procesos ejecutándose y cada una de las partes y colaborando en forma simultanea.

Las partes principales de un sistema Cliente/Servidor son:

Cliente

Es una aplicación, en donde se generan peticiones de acceso a los datos y de procesamiento de los mismos.

Servidor

Es una aplicación servidora dedicada, en donde se proveen los servicios que requiere el cliente

1.6 ODBC (Open Database Conectivity)

ODBC es un intermediador entre aplicaciones clientes y sistemas servidores de bases de datos. Se puede definir a ODBC como un interfaz de acceso a los datos via SQL que permite a la aplicación acceder a datos de diferentes sistemas gestores de bases de datos sin necesidad de saber sobre cual se está trabajando. ODBC es un estándar propiciado por Microsoft y aceptado por la industria como tal.

Antes de que existiera este estándar las bases de datos ya permitían a través de su API, la posibilidad de interaccionar con sistemas externos, como lenguajes independientes. Lo que ODBC aporta sobre esto, es un nivel superior de homologación, de manera que si tenemos los drivers ODBC adecuados, nuestra aplicación se escribe de forma totalmente independiente de la base de datos contra la que vaya a trabajar, interaccionando solo con el correspondiente driver ODBC. Luego cambiando dicho driver, nuestra aplicación funcionará de forma idéntica si cambia la base de datos contra la que trabajamos, ya que es el driver de ODBC el que se encarga de cambiar la parte que afecta a la base de datos manteniendo la parte que afecta a nuestra aplicación.

ODBC se encuentra muy ligado a Windows como sistema operativo, ya que ambos fueron hechos por Microsoft. Todo esta pensado para que exista un buen ensamblaje de sistema operativo (Windows) con el ODBC.

Cuando tenemos drivers ODBC, se instalan de manera automática en el panel de control a través de un icono que interacciona con un fichero denominado ODBC.INI donde se encuentra escrito lo necesario para identificar y manipular cada uno de los drivers ODBC presentes en nuestro sistema. Los drivers ODBC son DLL de Windows. Nuestras aplicaciones llaman a dichas DLL y ellas se encargan de traducir dichas llamadas a los servidores de datos requeridos. Este sistema esta pensado para que funcione en tiempo de ejecución de la aplicación, es decir que a lo largo del tiempo en que una aplicación está ejecutándose no solo se puede llamar a sentencias ODBC sino que, además, puede cambiarse de forma dinámica el driver ODBC, a través de un gestor de drivers, lo que permite trabajar simultáneamente con diversas bases de datos, distribuidas en diversos servidores inclusive.

1.6.1 Niveles de Arquitectura ODBC

En la arquitectura ODBC tenemos que distinguir varios niveles:

Aplicación

Ejecuta llamadas ODBC que invocan sentencias SQL y obtiene los resultados correspondientes.

Gestor de drivers

Carga los drivers de acceso a cada uno de los servidores gestores de bases de datos a partir de las necesidades de la aplicación. El gestor de driver puede manipularse directamente desde el panel de control de Windows y es el que nos permite seleccionar el driver adecuado para cada base de datos. Debido a este gestor que mientras una aplicación se ejecuta se puede cambiar el driver con el que trabajamos y , por tanto, cambiar "on line" la base de datos.

Driver

Procesa las llamadas ODBC, ejecuta los pedidos SQL hacia el sistema gestor de base de datos (a través de una fuente de datos) y retorna a la aplicación el resultado de la operación. Es el corazón del sistema, la DLL que tiene escrito el código para interaccionar con el API de la base de datos que controla, de modo que presenta esa información como llamadas ODBC estándar a la aplicación que lo usa. En cualquier caso hay que tener en cuenta que el driver no se entiende directamente con la base de datos sino con el último nivel de la arquitectura ODBC que es la Data Source o fuente de datos.

DATA SOURCE

El data Source o fuente de datos es la adecuación de un driver ODBC a una base de datos concreta. Un driver puede estar escrito para Oracle, pero la fuente de datos necesita saber, que ha de trabajar con dicho Sistema Gestor de Bases de datos, y con que base de datos debe interaccionar . Se puede decir que la fuente de datos es la representación de los datos que realmente ve nuestra aplicación.

1.7 ALGO QUE OFRECE VISUAL OBJECTS DE LO ANTERIOR

Visual Objects nos facilita la programación orientada a objetos porque permite su manejo como un tipo de dato más y los integra dentro de su estructura como lenguaje.

Dispone de todos los conceptos citados, lo único que no soporta es la herencia múltiple.

Encontramos mejoras con respecto a otros lenguajes orientados a objetos. Entre estas mejoras se encuentran las siguientes:

- Variables de instancia exportadas
- Métodos de acceso y asignación.

1.7.1 Definición de Clase

Visual Objects utiliza la declaración CLASS para identificar un nombre como una entidad del tipo clase dentro del repositorio de nuestra aplicación.

```
Class Almacenamiento
```

Con esta instrucción, que debemos teclear en el editor de código fuente, informamos al compilador de la existencia de una nueva entidad de tipo clase.

La visibilidad de las clases pueden ser:

1.- Global para toda la aplicación. Se puede usar en todos los módulos de nuestra aplicación para crear una instancia de la misma.

2.- Local para el módulo que ha sido definida. Para este tipo de ámbito antepondremos la palabra clave STATIC.

Una cláusula adicional a la definición CLASS es inherit. Esta cláusula es la que se utiliza cuando se desea aprovechar las características de otra clase que ya tengamos definida en nuestro sistema.

Se puede declarar la clase magnético y sería como sigue:

```
CLASS Magnetico INHERIT Almacenamiento
```

Dentro de esta entidad, tras la declaración class, podemos indicar todos los datos que contendrán la nueva clase. Una clase sin datos es válida en Visual Objects. Es posible que solo se sobreescriban métodos de la clase heredada.

En Visual Objects, los datos de un objeto son conocidos con el nombre de variables de la instancia. Las variables de una clase pueden ser de diferentes tipos, y son identificadas por distintas palabras clave.

INSTANCE	Tiene como visibilidad los métodos de la clase en la que son declaradas y sus clases descendientes. Tiene un enlace dinámico, por lo que pueden ser sobreescritas por métodos de acceso/asignación del mismo nombre en una clase descendiente.
PROTECT (INSTANCE)	Tiene como visibilidad los métodos de la clase en la que son declaradas y sus clases descendientes. Su enlace es estático, por lo que pueden ser redefinidas por métodos de Acceso/Asignación.
HIDDEN (INSTANCE)	Tiene como visibilidad los métodos de la clase para la que son definidas, pero no para las clases derivadas. Su enlace es estático y no pueden ser redefinidas.
EXPORT (INSTANCE)	Tiene como visibilidad los métodos de la clase para la que son definidas, y en el exterior de la clase. Es una de las formas de acceder a las propiedades de los objetos desde el exterior de los mismos. Su interpretación es temprana siempre que sea posible.

Las variables de la instancia pueden contener cualquier tipo de dato válido en Visual Objects, incluidos los objetos.

Al igual que las demás variables, las de una instacia pueden ser tipificadas de forma estricta, comprobando si el tipo de dato que les asignemos posteriormente es igual al que declaramos que almacenarían. Esta técnica agilizará nuestros programas en ejecución, al tiempo de que informará de asignaciones u operaciones incorrectas en tiempo de compilación.

Si no tipificamos la variable, ésta será automáticamente creada como USUAL.

Para tipificar una variable, se utiliza la cláusula AS dentro de la definición de la misma. Por ejemplo para nuestra clase Coche podemos definir la propiedad Ruedas como:

```
EXPORT Ruedas AS OBJECT
```

Con esto, el compilador de Visual Objects detectará cualquier intento de asignar a esta variable protegida un dato que no sea un objeto.

Visual Objects permite que una clase concreta pase a formar parte de su definición interna como lenguaje, con lo que podemos ampliar el lenguaje a nivel del propio compilador. Con esto podríamos tipificar la Variable Volante como:

```
EXPORT Volante AS Volante
```

Para que esta característica funcione correctamente, hemos de disponer, lógicamente, en nuestra aplicación de la clase que indica el tipo de la variable de instancia.

Al igual que las demás variable; las de una clase pueden inicializarse a un valor en el momento de ser declaradas junto a la entidad CLASS. Para esta operación se utiliza el operador de asignación de Visual Objects := .

```
EXPORT Volante := Volante ( GOMA, DIAMETRO_3 ) AS Volante
```

Si no especificamos un valor inicial para la variable, será usado el valor por defecto para las variables de tipo USUAL.

1.7.2 Métodos en Visual Objects

Los métodos de una clase son definidos como entidades independientes dentro del repositorio, indicando la clase a la que pertenece. El método Grabar de nuestra jerarquía almacenamiento para la subclase libro sería declarado en el editor de código fuente:

```
METHOD Grabar () CLASS Libro
```

Dado que los métodos son declarados como entidades independientes, varios visualizadores de Visual Objects identifican al método por su nombre completo.

Se puede comparar un método con una función, ya que, al igual que éstas, los métodos disponen de parámetros, mandatos, declaraciones del lenguaje, variables y pueden devolver valores. La diferencia principal entre un método y una función estriba en que el método deberá ser llamado siempre anteponiendo el nombre del objeto sobre el que debe actuar. Otra manera diferente con una función es que al ser un método parte de una clase concreta, podrá acceder a todas las variables en ella declaradas, con independencia del ámbito de las mismas.

Para que los métodos puedan devolver valores al punto del programa desde donde son invocados se utiliza la declaración RETURN. Adjunto a esta declaración situaremos el valor que deseamos que devuelva el método. En Visual Objects, el valor por omisión que devuelvan los métodos es SELF (el objeto en sí mismo) por lo que si deseamos que un objeto devuelva NIL, deberemos incluir la línea:

```
RETURN NIL
```

La ventaja de devolver SELF es que se ilumina la posibilidad de encadenar mensajes a objetos, aliviando el número de líneas de nuestra aplicación. Para hacer andar y girar al objeto automóvil, si no dispusiéramos de esta facilidad, deberíamos disponer de dos órdenes

```
Automóvil: Anda()
Automóvil: Gira()
```

Al devolver el objeto mismo (Self) podemos optar por las siguiente línea:

```
Automóvil: Anda(): Gira()
```

La invocación de un método sigue el siguiente patrón:

(objeto):(id Método) (((lista de argumentos)))

El operador que indica al compilador que debe prepara código para el envío de un mensaje es conocido por operador send. El operador send es representado por el signo de dos puntos (:).

Un método debe especificarse con paréntesis, se le envíe argumento o no.

Los argumentos de un método no pueden tipificarse, y son siempre del tipo USUAL.

Igualmente a las funciones, podemos asignar el valor de devolución de un método.

El envío de mensajes es la única forma de interaccionar con un objeto en la Programación Orientada a objetos, y esto no es del todo cierto en visual objects, ya que permite acceder directamente a las variables de objeto que han sido definidas como EXPORT. Es recomendable tener precauciones con estos tipos de variables ya que se ignora uno de los pilares básicos de la Programación Orientada a objetos, la encapsulación.

1.7.3 Construcción e Inicialización de Objetos

Se denomina construcción de un objeto al proceso por el cual instanciamos una nueva ocurrencia de una clase concreta.

El proceso de instanciación es indicado a Visual Objects por medio de la siguiente sentencia:

```
( ( oVariable):= ) (Nombre de la Clase) { (lista de argumentos) }
```

El operador de instanciación es identificado por las llaves {} .

No en todas ocasiones necesitaremos guardar un objeto en una variable, ya que en muchos casos únicamente lo instanciaremos para pasarlo como argumento de una función o método. También podemos crear un objeto como valor de devolución de una función o método.

```
// Código de clase
CLASS Factura
        EXPORT Lineas  AS ARRAY

METHOD Imprime ( ) CLASS Factura

Rreturn Printer (Aadd (Lineas, LineaFact "Total :", 15000 ))
```

El método imprime ( ) de la clase Factura devuelve un objeto Printer al que instancia, remitiéndole como parámetros una variable de su clase y un objeto de la clase LineaFact.

```
// Código de clase
Class Muestra Texto
        PROTECT cTexto

METHOD Init ( cCadena )
        cTexto := cCadena

METHOD Muestra ( )
        °10,10 SAY cTexto
// Código Cliente
Function Star ( )
        Local o AS MuestraTexto

o  := MuestraTexto { "Hola " }
o :=Muestra( )          // Visualizara "Hola" en la ventana terminal
```

La tarea normal del método Init( ) es almacenar en las variables del objeto los valores que le son pasados en la lista encerrada entre los operadores de instanciación, crear objetos que contendrá en sus variables, crear las relaciones entre los diferentes objetos contenidos, abrir ficheros etc.

1.7.4 SELF Y SUPER

Self es una palabra clave especial que permite que nos podamos referir al objeto en si mismo desde un método de la clase.

Siempre que deseemos invocar a un método de la clase para la que estamos definiendo otro, hemos de usar Self.

```
METHOD Grabar ( uDato ) CLASS Magnético
        Dato := uDato

METHOD Borrar ( ) CLASS Magnético
        Dato := NIL

METHOD Regrabar ( uDato ) CLASS Magnetico
        Self: Borrar ( )
        Self: Grabar ( uDato )
```

Para referirnos desde un método a las variables de la instancia, esto es optativo, siempre que no exista una variable LOCAL del mismo nombre, ya que oculta a la variable del objeto.

```
// Código de clase
CLASS SumaDos
        PROTECT Operador1 := 1 AS INT
        PROTECT Operador2 := 2 AS INT
        PROTECT Resultado  := 100 AS INT

METHOD Hazok ( ) CLASS SumaDos
    LOCAL Resultado AS INT

    Self: Resultado := Operador1 + Operador2
RETURN Self:Resultado

METHOD Hazok ( ) CLASS SumaDos
    LOCAL Resultado AS INT

    Resultado :=  Operador1 + Operador2
RETURN Resultado

// Codigo cliente
FUNCTION sTART ( )
    Local o AS SumaDos

    o := SumaDos ( )
    ? o: Hazok ( ) //  0
    ? o: Hazok ( ) //  3
```

En Hazok la variable LOCAL Resultado oculta la variable de la instancia resultado. La forma correcta de referirse a la variable de instancia resultado está en el método Hazok ( ) , que antepone a la variable de instancia Self.

Dentro de una jerarquía de clases, algunos métodos de las clases hijas sobreescriben a los de sus padres, aprovechando las características de estos en la nueva implantación.

Visual Objects tiene la palabra Super para referirnos a la clase padre. Super tiene significado siempre que heredemos, y desde un método redefinido quisiéramos invocar a la acción del método con el mismo nombre en su clase padre. Si deseamos referirnos desde un método, que no redefine a ninguno del padre, a la clase padre, no es necesario utilizar Super, ya que Self contiene la referencia a este método.

Super es de uso común cuando el método de una clase heredada no reemplaza completamente las acciones del método de su clase padre.

```
CLASS Uno
      PROTECT Var

METHOD Init ( ) CLASS Uno
      Var := 1

CLASS Dos INHERIT Uno
      PROTECT VaNew

METHOD Init ( ) CLASS Dos
      Super: Init ( )
      VarNew := 2
```

El método Init ( ) de la Clase Dos aprovecha la tarea realizada por el método Init ( ) de la clase Uno antes de comenzar su labor.

1.7.5 Acceso Y Asignación

Como se mecionó anteriormente, no existe otra forma de acceder a los datos que contiene un objeto si no es por un método. Si deseamos mantener un riguroso control sobre los accesos y las asignaciones que se realizan sobre una variable, Visual Objects facilita dos tipos especiales de métodos, que son considerados por el repositorio como entidades independientes.

Para acceder desde el exterior de un objeto a sus datos podemos definir un método que realice el acceso y otro que realice la asignación. Podemos hacer uso de estos métodos como si se tratara de variables, facilitando la utilización del objeto.

```
// Código de clase
CLASS Empleado
      EXPORT Nombre
      PROTECT Sueldo

METHOD FijaSueldo ( x ) CLAS Empleado
      IF X > 0
            Sueldo := x
      ELSE
            Alert ( ' ¿ no cobra ?' )
      ENDIF

METHOD    TomaSueldo ( ) CLASS Empleado
```

```
RETURN     Sueldo

// Código Cliente
FUNCTION Start
        LOCAL o AS Empledao
o := Empleado ( )
o: nombre := ' Luis '
o: FijaSueldo ( 15000 ) // No usamos la misma sintaxis
? o:Nombre
? o:TomaSueldo( )       // No usamos la misma sintaxis
```

Aunque con el código mostrado resolvemos el acceso a una variable protegida, hemos de utilizar una sintaxis diferente a cuando accedemos a variables EXPORT. Sería, deseable disponer de una sintaxis idéntica, pero preservando la posibilidad de control y protección que representan los métodos de acceso y asignación que implica la encapsulación.

Visual Objects facilita este deseo, con dos declaraciones especiales de métodos: ACCESS y ASSIGN. Estas, como cualquier otro método, son tratadas como entidades independientes por el repositorio.

Veamos como quedaría el ejemplo anterior codificando con estas dos nuevas declaraciones:

```
// Código de clase
CLASS Empleado
        EXPORT  Nombre
        PROTECT Sueldo

ASSIGN Sueldo ( x ) CLASS Empleado
        IF  X > 0
                Sueldo := x
        ELSE
                Alert ( ' ¿ No cobra ?' )
        ENDIF
ACCESS Sueldo ( ) CLASS Empleado
RETURN Sueldo

// Código Cliente
FUNCTION Start ( )
        LOCAL o AS Empleado

        o := Empleado { }
        o : Nombre := ' Luis '
        o:Sueldo := 1500 // Usamos la misma sintaxis
        ? o :Nombre
        ? o:Sueldo // Usamos la misma sintaxis
```

El código cliente en este caso es mucho más intuitivo que en el ejemplo anterior. Y lo más importante para el desarrollador es que esto se logra con un cambio en el código de clase.

1.7.6 Métodos Predefinidos

Visual Objects envía ciertos mensajes de forma automática a nuestros objetos, siempre que estos sean definidos por nosotros. El más importante de los métodos es Init ( ). Existen algunos otros como Axit ( ), que es el método con el que podemos controlar la destrucción de un objeto. Es utilizado en la situación opuesta a Init ( ). Aunque el sistema de Visual Objects no necesita realizar esta destrucción de un objeto, no existiendo el concepto de destructor como en otros lenguajes Orientados a Objetos, en los que se pueden realizar tareas previas a la finalización del tiempo de vida de un objeto.

El recolector de basura implementado en Visual Objects se encarga de eliminar los objetos de manera automática cuando ya no existe ninguna referencia en memoria a los mismos. En objetos completos como las ventanas, se implementa un método destructor Destroy ( ) para agilizar el uso de memoria en sistemas que no tengan una abundante disposición de la misma.

En un objeto se pueden realizar tareas externas a él. Una de estas tareas puede ser la apertura de un fichero externo para la importación de datos, el cual solo es utilizado por dicho objeto. No podemos saber cuándo el recolector de basura de Visual Objects va a descartar por completo el objeto, pero sabemos que necesitamos cerrar el fichero abierto a nivel del sistema cuando se destruya la última referencia al objeto que lo usa. Para esto se registra el método Axit ( ), y este será ejecutado instantes antes de que el recolector de basura elimine el objeto en memoria.

Para registrar el método Axit ( ) , disponemos de la función RegisterAxit ( ), la cual se encarga de definir dicho método en el sistema para el objeto que le es pasado como parámetro.

Axit ( ) es un método gracias al cual podemos conseguir un robusto sistema de clases. Las jerarquías de Visual Objects hacen un uso exhaustivo de este método. Si vamos a utilizar clases que manejen recursos externos, es conveniente definir nuestros propios métodos Axit ( ).

NoMethod ( ) es accionado siempre que intentamos ejecutar un método que no está definido para el objeto al que se lo indicamos. Es un método para gestionar los posibles errores dentro de una aplicación . Desde dentro del método

Nomethod podemos invocar la función NoMethod para conocer el nombre del mensaje que se intenta ejecutar.

NoIVarGet ( ) y NoIVarPut ( ) son ejecutados si existen, cuando intentamos acceder a una variable o asignarle un valor y esta no existe. Ambas reciben como parámetro el símbolo que identifica a la variable con la que se desea operar. En el caso de NoIVarPut ( ) se recibe como segundo parámetro el valor que se desea asignar. Gracias a estos métodos podemos solventar ciertos errores en tiempo de ejecución.

1.7.7 Sobrecarga de Operadores

Al activar la opción de métodos de operadores en la caja de opciones del compilador, se convierten ciertas operaciones con objetos en llamadas a métodos.

Esta característica requiere que el operador de la izquierda sea del tipo objeto.

Conversión de operadores a métodos

OPERACION	CONVERSION
a +b	a:Add (b)
a - b	a:Sub (b)
a * b	a:Mult (b)
a / b	a:Div (b)
a ^ b a ** b	a:Pow (b)
a % b	a:Mod (b)
a > b	a:Gtr (b)
a < b	a:Less (b)
a >= b	a:GtrEqu (1)
a <= b	a:LessEqu (1)
a++	a:=a:Add (1)
a--	a:= a:Sub (1)
-a	a:Neg ( )
!a	a:Not ( )

CAPITULO

REPOSITORIO DE ENTIDADES

En el capítulo anterior describimos los conceptos necesarios para poder trabajar con Visual Objects, en este capítulo, empezamos a conocer con lo que cuenta, pero antes de adentrarnos al ambiente de Visual Objects vamos a conocer lo necesario para que se lleve acabo una buena instalación del producto.

Visual Objects requiere los siguientes componentes de Hardware como de Software.

COMPONENTE	MINIMO	RECOMENDADO
DOS	Versión 3.0	Versión 6.2
Windows	Versión 3.0	Versión 3.1
Espacio en disco	35 MB	55 MB
Memoria RAM	4MB	8MB
Computadora	386 SX 25MH	486 DX2 66 MH

El espacio en disco es considerado para una instalación completa, si se requiere se puede hacer una instalación parcial ocupando menos espacio en disco.

Es necesario tener un margen mayor de espacio para poder trabajar con las aplicaciones que se elaboren. Es posible que contemos con una computadora con los componentes mínimos requeridos por Visual Objects y esto ocasiona que el producto sea mas lento en comparación con una computadora que tenga mayores recursos.

Para el buen funcionamiento de Visual objects es recomendable tener en el CONFIG. SYS las siguientes líneas:

FILES = 100

BUFFERS= 16, 0

INSTALL = C:\DOS\SHARE.EXE / F:4096 / L:50

La última línea es solamente si tenemos un sistema con WINDOWS versión 3.1y las especificaciones de /F y /L son independientes de cada sistema.

Si el sistema en el que se instalará Visual Objects tiene drivers instalados que usen compresores de datos como STACKER, DOUBLE SPACE o DOUBLE DISK, Visual Objects tendrá un comportamiento diferente al normal.

Teniendo instalado completamente Visual Objects veremos el grupo de Programas que a continuación se describe:

Repository Manager: Como veremos mas adelante Visual Objects no utiliza archivos, si no que todo se encuentra en el repositorio, este programa nos permite repararlo por si llegara a sufrir algún daño.

CA Visual Objects: El producto

Readme: Es un archivo en el formato de Write de Windows que contiene información sobre los últimos cambios hechos en la versión de Visual Objects.

3er Party Vendors Help: Es un archivo en el formato de ayuda de Windows que contiene referencias a libros, bibliotecas de terceros y otros servicios relacionados con Visual Objects.

ODBC Help: El archivo de ayuda para el uso de Open Data Base Conectivity (Conectividad abierta de Base de Datos)

CA Visual Objects Help: Archivo de ayuda general de Visual Objects.

Hotspot Editor Help: El editor Hotspot es un programa necesario para generar archivos de ayuda para Windows.

Creating Help Files: Archivo de ayuda que indica como crear ayudas para Windows.

Interactive Watcom SQL: Visual Objects viene con una versión de un producto llamado Watcom SQL, el cual le permite el acceso a bases de datos basadas en tablas SQL, este programa permite manipular este tipo de archivos.

Watcom SQL Help: Archivo de ayuda para el uso de Watcom SQL

Watcom SQL DB tools: Herramienta para el manejo de tablas SQL.

2.1 EL REPOSITORIO

Un repositorio es una mesa o banco donde se ubica de manera ordenada todo lo que el programador necesita y usa (código y vínculos) tanto si pertenece a librerías o aplicaciones, es un sistema almacenador y organizador de recursos con muchas ventajas sobre un sistema de desarrollo tradicional, ya que no es necesario lidiar con archivos cuando se trabaja en VISUAL OBJECTS. En lugar de que una aplicación esté relacionada con uno o ,más archivos (.PRG o .CH, etc.)ahora consistirá de uno o más módulos. (fig 1)

Siendo el repositorio un ente único, hace que desaparezca el concepto de fichero individualizado tal como se percibe en la mayoría de los lenguajes: fuente, accesorios, obj y ejecutable.

En Visual Objects las aplicaciones, módulos y entidades están almacenadas en el repositorio. A pesar de esto todas estas piezas que conforman un ente único, son manejables y editables, pero con la diferencia de que no se basan en archivos, el repositorio los contiene a todos.

Otra de las ventajas del repositorio es que cada vez que tiene que compilar una aplicación, el repositorio sabe qué cosas compilar basado en los cambios que se han hecho y construye una nueva aplicación de la manera más eficiente sin tener que compilar toda la aplicación, esto recibe el nombre de "compilación incremental", en la cual solo las piezas de código que han sido modificadas vuelven a compilarse, ello reduce considerablemente el tiempo de desarrollo. Tal grado de automatización elimina la elaboración de archivos para compilación y encadenado.

Básicamente existen dos componentes del repositorio, la parte que se ve, que recibe el nombre de Ambiente Visual de Trabajo y la parte que no se ve, que es realmente la más importante y está constituida por los componentes internos, como el compilador de código nativo y la maquinaria de almacenamiento y organización.

2.2 ORGANIZACION DE LA MESA DE TRABAJO

La desaparición de ficheros acarrea la necesidad de definir de modo diferente los procesos de enlace y los vínculos con librerías.

Cuando vamos a crear una aplicación VISUAL OBJECTS, especificamos por medio de una pantalla de propiedades cuales van a ser librerías externas que incorporar y el modo de desarrollo (MDI, DEBUGGER, etc....). Si necesitamos cambiar algo a lo largo de las fases de desarrollo, podemos perfectamente acceder de nuevo a la misma pantalla de propiedades y cambiarlas.

Una librería de Visual Objects es un conjunto de módulos cuyo tratamiento se diferencia de una aplicación por el echo de que no genera un ejecutable. Es necesario compilar compilar las librerías si queremos que su código esté disponible desde otras aplicaciones.

Si en una aplicación usamos librerías, al igual que ocurre en los lenguajes para DOS, se incorpora su código a la aplicación . En caso de usar DLL las mismas se incorporan externamente a los discos de instalación de la aplicación y por consiguiente el instalador de aplicaciones se ocupa de todo automáticamente.

2.2.1 Visualizadores del Sistema

El visualizador nos proporciona información que esta almacenada en el repositorio, de manera organizada y conveniente.

La mayoría de los visualizadores de Visual Objects, permiten subextracción de ciertos datos, por ejemplo, el visualizador de clases, despliega clases en la forma de una estructura de tipo árbol expandible-colapsable permitiéndonos determinar que información vamos a extraer.

La integración de los visualizadores y el repositorio, provee el fácil acceso a diferentes editores.

Los visualizadores, se utilizan para diferentes propósitos y las visualizaciones proporcionan una foto general de la información almacenada en el repositorio; también se puede manipular dicha información, de manera que podemos cambiar el nombre a una aplicación o mover un módulo a otra aplicación. Los visualizadores proporcionan acceso a los diferentes editores existentes en Visual Objects.

2.2.2 Visualizador de Aplicación

El repositorio se presenta como una colección de módulos, que a su vez puede subdividirse en otros, bajando así hasta tres niveles básicos de abstracción; las aplicaciones comprenden módulos que a su vez contienen entidades y los visualizadores siguen esta jerarquía de arriba-abajo (top - down) y es la siguiente: el de aplicaciones, el de módulos y el de entidades.

El visualizador de aplicaciones es el primero que vemos cuando entramos a CA Visual Objects, y es como el de la figura 2.1.

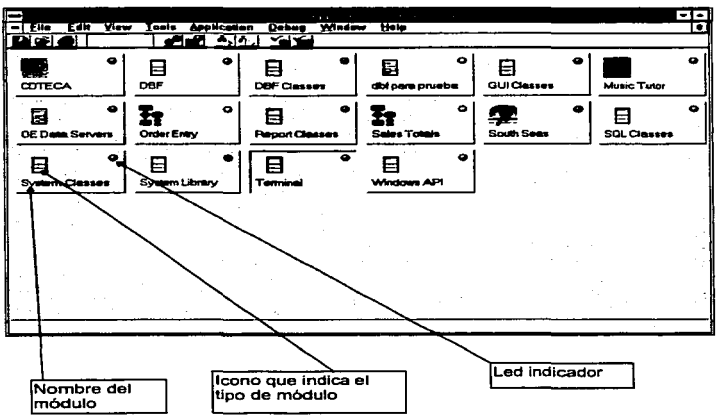


figura 2.1

En este visualizador se muestran las aplicaciones y librerías disponibles en el sistema. Independientemente de su tamaño o contenido se presentan de forma individual como un único rectángulo gráfico que lleva además de su nombre, un indicador de estado en forma de led ubicado en su parte superior derecha, y el significado es el siguiente:

- Verde: está compilado y apto para el uso.
- Amarillo: es de nueva creación, no ha sido compilado nunca.
- Rojo: contiene errores y no se puede usar.

El recuadro de la izquierda sobre el nombre contiene un icono que indica que se trata de:
Un módulo de Visual Objects (cuando es azul con flechas blancas), una librería nuestra o de terceros (cuando es marrón con flechas amarillas) o una aplicación (cuando es un diagrama de flujo).

El visualizador de aplicaciones permite realizar operaciones tales como supresión de aplicaciones, librerías y/o DLLs, mediante la opción de menú VIEW/SHOW, o bien, hacer una ordenación de botones de aplicaciones por nombre o tipo.
Solo existe un visualizador de aplicaciones en el que podemos tener abiertos al mismo tiempo, varios visualizadores de errores, módulos, entidades y clases.

2.2.3 Visualizador de Módulos

Después de efectuar un doble click en el área de alguna aplicación, dentro del visualizador de aplicaciones, se despliega el visualizador de módulos y es como el que se muestra en la figura que sigue.

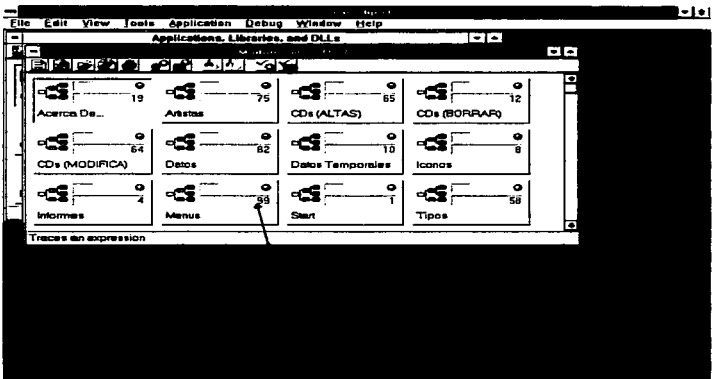


figura 2.2

De igual manera que en el visualizador de aplicaciones, el visualizador de módulos presenta cada uno de ellos como un botón. El dibujo que identifica a cada módulo es similar a un diagrama de flujo, sin embargo bajo el led de estado, aparece una cantidad numérica. Esa cantidad, nos indica el número de entidades que componen el módulo.

2.2.4 Visualizador de Entidades

El doble click en un módulo trae una nueva ventana que despliega las entidades definidas en dicho módulo, esta ventana recibe el nombre de visualizador de entidades. Aquí se nos indica la luz de estado, el tipo de entidad y su nombre. En un visualizador de entidades aparecen todas las de un módulo, ordenadas alfabéticamente independientemente de su tipo. Además, en la parte inferior de la pantalla, se observa el contenido de la primera línea de código que compone la entidad.

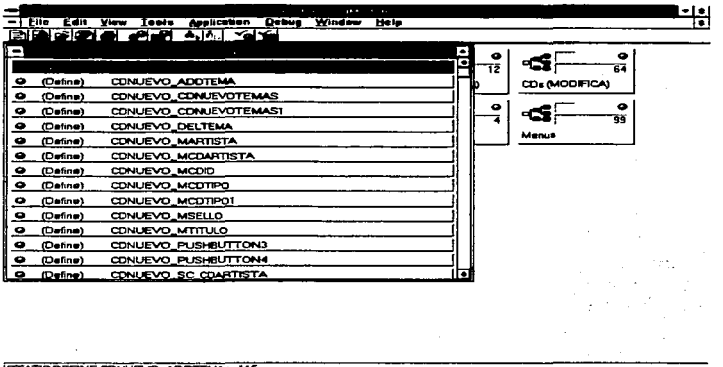


figura 2.3

La visualización de entidades de un módulo, puede mejorarse considerablemente llamando a otros visualizadores de entidades disponibles, como por ejemplo el ENTITY BROWSER del menú de herramientas, el cual nos muestra todas las entidades independientemente del módulo al que pertenecen, y las ordena por tipo permitiendo encoger y expandir los tipos.

Cada entidad es una estructura de árbol expandible-colapsable, agrupada por tipo y permitiéndonos la selección de mayor visualización. Por ejemplo, para colapsar las ramas del árbol, seleccione la opción VIEW del menú principal y luego utilice COLLAPSE ALL; para realizar la opción contraria, es decir, para expander una rama y visualizar sus entidades en mayor detalle, presione el botón signo de (+) a la izquierda de la rama.

2.2.5 Visualizador de Clases

Permite la visualización de todas las clases asociadas con la aplicación : al igual que el visualizador de entidades, es accesado por el menú con la opción TOOLS.

El visualizador de clases es similar al de entidades en cuanto a que se despliegan las clases en estructura expandible-colapsable y puede accesarse al código fuente.

2.2.6 Visualizador de Errores

Durante el ciclo de desarrollo o compilación de una aplicación, pueden surgir errores y/o warnings; para ayudarnos a encontrarlos y corregirlos de manera rápida y eficiente, VISUAL OBJECTS nos proporciona el visualizador de errores.

Para ingresar a este visualizador de errores escogeremos en el menú principal TOOLS y luego ERROR BROWSER, que nos listará todas las entidades que tienen errores.
Con un doble click en un error nos lleva directamente a la línea en código fuente que lo genera.

2.3 CLASES DE OBJETOS PREDEFINIDAS

Las clases de objetos predefinidas de Visual Objects nos proporcionan fundamentos para la programación orientada a objetos, ya que ponen a disposición del desarrollador, código previamente probado y múltiples facilidades para resolver problemas.

Las clases disponibles para nuestras aplicaciones se puede observar dentro del visualizador de aplicaciones y están representadas con un icono de un archivero con flechas apuntando hacia adentro.

Las clases predefinidas con que viene Visual Objects de fábrica son las siguientes:

- Clase GUI
- Clase DBF
- Clase SQL
- Clase SISTEMA

Proporcionan varias clases y métodos que nos ayudan a crear ventanas, agregar menús y controles a las mismas, y procesar mensajes que son enviados a las ventanas basados en eventos.

Adicionalmente, existen clases y métodos para manejar el portapapeles, funciones de arrastra-suelta entre aplicaciones, DDE y soporte para ayuda, así como métodos y clases para ayuda, para desplegar mapas de bits, iconos, plumas, brochas, cursores, generación e impresión de reportes.

Existe soporte completo en la creación de aplicaciones dependientes de datos. Los controles en las ventanas de datos pueden relacionarse directamente con campos de bases de datos, teniendo una actualización automática conforme los registros cambien, y cualquier modificación hecha a los controles va a ser reflejada en la base, después de pasar las validaciones establecidas.

Todo se concentra en la ventana, los mensajes para una ventana se mandan a métodos apropiados. Los métodos pueden ser subclasificados o sobrescritos y reemplazados por los métodos elaborados por el usuario.

2.3.1 Clase DBF

Provee el acceso a archivos .DBF (Xbase) de manera orientada a objetos, estas clases son consideradas sustituto del área de trabajo en términos Xbase.

2.3.2 Clase SQL

Permite establecer conexiones a servidores de bases de datos que manejan controladores ODBC y accesan los datos al estilo Orientado a Objetos de manera consistente al resto de nuestra aplicación.

Las clases SQL y DBF manejan numerosos métodos en común, gracias a la funcionalidad proporcionada por las clases GUI, que permiten usarlos, quizás con modificaciones mínimas, sin conocer acerca de los datos fuentes : En caso de tener funcionalidad no soportada por ODBC, el método regresará sin realizar operación alguna.

2.3.3 Clase Reporte

Proveen acceso en tiempo de ejecución a reportes escritos en el reporteador CA - RET, el cual permite crear reportes WYSIWYG o estilo procesador de palabras, en forma de columnas y renglones. Adicionalmente, puede incluir bitmaps en el reporte y cambiar atributos como tamaño y tipo de letra.

Estas clases contienen métodos que permiten abrir un reporte, hacer una presentación previa en pantalla o bien, mandarlo a la impresora directamente. Se puede especificar el número de copias, pasar parámetros a los reportes para que los acepten en tiempo de ejecución.

2.3.4 Clase Sistema

Contiene clases usadas a su vez por otras clases y por nuestras aplicaciones, se incluyen aquí las clases DBERROR e HYPERLABEL.

La clase DBERROR es un manejador de errores involucrada en ventanas de datos y servidores de datos. La clase Hyperlabel contiene cuatro campos: NAME, CAPTION, DESCRIPTION Y HELP CONTEXT y es usada durante la utilización de las clases GUI; DBF y SQL; cada ventana y control de ventaneo, cada base de datos y cada campo dentro de la Base de datos tiene HYPERLABEL.

2.4 EXPORTACION E IMPORTACION.

El repositorio permite exportar e importar módulos en sus diferentes niveles. Inicialmente y estando ubicados en el Visualizador de Aplicaciones, podemos utilizar opción de menú que permite salvar todo el repositorio a un directorio. También podemos exportar/importar aplicaciones o librerías completas teniendo la opción de hacerlo globalmente con todas ellas, o de una en una. En este caso se exportan/importan archivos con extensión AEF. El repositorio al importar una aplicación desde un sólo archivo AEF genera todos los módulos que la componen, además de sus vínculos y propiedades.

La exportación/importación desde el visualizador de módulos se hace con archivos de extensión MEF, que solamente contienen un módulo por archivo. Además, se nos permite copiar o mover un módulo de la aplicación activa a otra aplicación.

La exportación desde el editor de código fuente, se realiza en formato puramente ASCII en ficheros de extensión PRG o CH.

Una de las grandes ventajas de la Exportación/Importación consiste también en mover módulos ya depurados y probados desde aplicaciones a librerías en lugar de copiarlos de aplicación a aplicación.

Visual Objects no permite en la actualidad el desarrollo concurrente en un mismo repositorio. Es decir, que un mismo repositorio no puede estar abierto por dos usuarios a la vez en una red local. Sin embargo si pueden tener dos o más repositorios paralelos desde los que van exportando/importando módulos o aplicaciones: Aunque el fabricante ha prometido resolver este problema, en la actualidad es imposible hacerlo. Esta limitante se puede substituir con la exportar/importar módulos o aplicaciones completas incluyendo sus vínculos facilitando considerablemente el trabajo de equipo. Basta con desplazar un solo archivo en disquete para que todo funcione en la máquina destino olvidándose de paths, LNKs, RMKs, etc.... La ausencia de una librería en el repositorio destino hace que se elimine su nombre de la tabla de vínculos establecida en la pantalla de propiedades de la aplicación. Si se introduce posteriormente esa librería en el repositorio habrá que restablecer el vínculo manualmente ya que habrá desaparecido.

El repositorio puede crear conflictos si no se administra su contenido debidamente. Su corrupción puede llevarnos en el peor de los casos a irremediables perdidas de código. Para prevenir estas situaciones es recomendable exporta frecuentemente las aplicaciones a discos flexibles. El repositorio es una base de datos con indices y los procesos más comunes que permite son el empaquetado y la regeneración de los índices, por tal motivo se sugiere hacer copia de seguridad de todo el repositorio para evitar que una caída de tensión durante la compactación provoque pérdidas mayores de las que se pretendían corregir.

La compactación del repositorio debe hacerse periódicamente con ciertas precauciones porque es un proceso largo y tedioso.

FALTA PAGINA

No. 36 a la 37

CAPITULO

IDE (ENTORNO INTEGRADO DE DESARROLLO)

3.1 EL IDE COMO INTERFAZ DE DESARROLLO MDI

Los que provenimos de entornos de desarrollo como Clipper, estamos acostumbrados a la adquisición adicional de un editor de código fuente, y además teníamos que preocuparnos de configurar y personalizar su entorno de trabajo. Algunas herramientas como C++ ya aportaban entorno integrado, pero ninguna alcanza el poder y la potencia del IDE (Entorno Integrado de Desarrollo) de Visual Objects.

El IDE de Visual Objects además de potente es puramente MDI, ya que permite abrir varias ventanas, módulos e incluso aplicaciones en diferentes niveles de repositorio. Su potencia reside en el hecho de que incluye todos y cada uno de los pasos que debe dar el programador en la confección de una herramienta o aplicación. Por consiguiente, además de los generadores de código que veremos a continuación, el IDE incluye Ayudas, Compilador, Enlazador y Depurador.

Los generadores simplifican enormemente la vida, ya que entre otras cosas permiten diseñar visualmente menús, ventanas, etc.,,, y escriben automáticamente el código fuente.

3.2 EL EDITOR DE MENUS

Windows nos proporciona un sistema de menús estandarizado y Visual objects un potente editor para los menús de Windows. Este editor (muy sencillo de manejar, y totalmente intuitivo), incluye todo tipo de ayudas y dos menús prefabricados. Mirando el menú de cualquier aplicación hecha en Visual Objects, vemos que algunas de sus opciones tienen teclas de acceso rápido y directo (llamadas aceladores en Windos), e incluso barra de herramientas con iconos.

Todo menú de Windows se compone solamente de dos tipos de items: PopUp y Prompts. También hay separadores, pero son irrelevantes, por ser meramente decorativos y no ejecutar nada. Los PopUp se pueden definir en cascada y no activan nunca ningún proceso, solo pueden abrir otro PopUp o desembocar en uno o varios Prompts.

Usando el Visualizador de entidades, vemos que la aplicación estandar que genera Visual Objects ha sido automáticamente dotada con dos menús: Uno llamado EmptyShellMenu (el menú inicial), y otro llamado StandardShellMenu (el que se activa al abrir la edición de una DBF). El que nos interesa en este momento es EmptyShellMenu, y desde el mismo Visualizador de entidades vamos a hacer doble clic de ratón sobre la entidad de tipo menú llamada EmptyShellMenu. Ya estamos automáticamente en el editor de menús que podemos dividir en tres zonas, la zona de edición gráfica, donde se diseña el menú propiamente dicho. La ventana de propiedades que permite editar la características de cada item. Y finalmente el área de visualización práctica del menú tal como se está diseñando en la parte arriba izquierda de la figura 3.1. Independientemente de las variadas opciones de cada item, vamos a considerar el primer atributo que aparece en la ventana de características (Abajo derecha ¨) , que no es otro el nombre del código que se ejecuta (EventName) cuando un usuario elige esa opción de menú.

El EventName es en la mayoría de los casos, un Método de la clase StandardShellWindow, es decir, la ventana principal de la aplicación que lógicamente no es otra cosa que un objeto. Para crear un nuevo item de menú nos posicionamos en el que queremos que sea su predecesor y pulsamos (intro). Observemos como en la zona de visualización del menú se puede comprobar gráficamente lo que estamos haciendo, ya que se produce un refresco automático de pantalla. Intuitivamente también deducimos que el signo & convierte a la letra que le sigue en Hot Key, y por consiguiente lo colocaremos en el lugar apropiado para activar la letra deseada.

Podemos crear más items y convertirlos en prompts o pop-up usando para ellos el teclado: Teclas (Tab) o (May)(Tab). O simplemente las flechas de la ToolBar del editor.

EmptyShellMenu

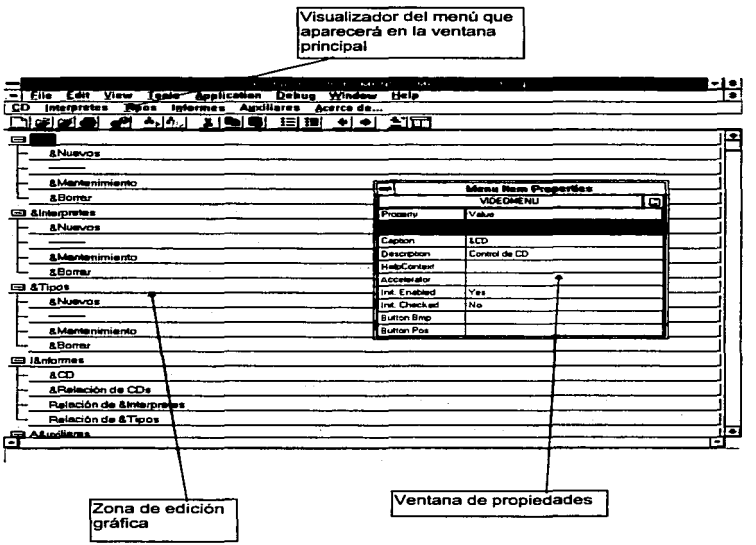


figura 3.1

Gracias al sistema gestor de mensajes de Windows, Visual Objects permite crear tantas opciones de menú como queramos sin que sea necesario que llamen a ningún evento (algo muy similar a los que se puede hacer con los prompt de Clipper), y algo más importante; se nos permite asociar a un ítem de menú el nombre de un evento que NO EXISTE, y al elegir la opción en tiempo de ejecución no ocurrirá nada (ni siquiera un mensaje de error). Se puede diseñar

un menú asociado a procesos que se irán declarando y creando a lo largo del desarrollo, sin necesidad de volver al editor de menús constantemente.

Para agilizar el uso de menús, podemos crear aceleradores, es decir, combinaciones de teclas que permitirán acceder directamente a los items sin necesidad de navegar por el menú. También es posible asignar a cada item un icono de la ToolBar, el cual nos llevará directamente a la opción pudiendo además decidir el orden secuencial de los iconos de la toolbar. Ambos sistemas de acceso rápido cubren cualquier necesidad: Los aceleradores sirven para una navegación por teclado y las ToolBars ayudan a la navegación con ratón.

Si salimos guardando los cambios, vemos como Visual Objects trabaja actualizando infinidad de entidades y provocando la aparición de la luz amarilla en los módulos de menú y shell, por lo que habrá que compilar y enlazar de nuevo. En la aplicación deben de aparecer las nuevas opciones en el menú, y los eventuales iconos en la barra de herramientas, sin embargo al intentar ejecutarlos, no ocurrirá nada, ya que tal como decíamos antes, hay que definir los eventos asociados.

Este editor cuenta con una característica de autoconfiguración, la cual es muy útil, la autoconfiguración es usada para agregar uno o más menús predefinidos estándar a una aplicación.

Adicionalmente cada uno de estos menús predefinidos, contiene un conjunto de elementos y opciones asignados por default, esto incluye eventos y una barra con botones de herramientas.

La autoconfiguración provee una forma rápida de comenzar sus propias estructuras de menús. Puede usar el menú resultante como esta o agregar sus propios elementos para una aplicación en particular. También se puede empezar desde cero si no requerimos de esta autoconfiguración.

Para cada estructura de menú que cree, puede habilitar o deshabilitar la barra de herramientas correspondiente. Si está habilitada puede escoger cuales elementos en la estructura del menú deben tener botones en la barra, como también que gráfica debe ser usada para cada elemento.

3.3 EDITOR DE VENTANAS

El editor de ventanas es uno de los más complejos, y eso es debido fundamentalmente a la gran riqueza que ofrece Visual Objects con los siguientes tipos:

1o. TopAppWindow: Ventana inicial de aplicación en un sistema SDI (Single Document Interface). Es decir una aplicación donde no se autorizará al usuario a abrir múltiples ventanas a la vez. Aun siendo Windows, es lo más parecido a una aplicación DOS, ya que resulta mucho más sencilla de controlar que MDI.

2o.ShellWindow: Ventana principal de un sistema MDI (Múltiple Document Interface). La terminología Visual Objects siempre habla de una ventaja principal " "Owner" o "Parent" (Propietaria o padre), y de ventanas "Child" (hijas).

3o.ChildAppWindow: Como su nombre indica, es una ventana "Child", es decir hija de otra. Se usa tanto en sistemas SDI como MDI. Tengamos en cuenta que en un sistema SDI las ventanas se van abriendo de una en una como hacemos en Clipper, y si bien solo tenemos una activa a la vez, podemos tener muchas ventanas abiertas. Por ejemplo tenemos la ventana inicial de la aplicación, luego la ventana de mantenimientos, y dentro de ella la ventana de clientes. En un sistema MDI, podemos tener varias ventanas de mantenimiento a la vez, aunque es aconsejable controlar los procesos concurrentes, usando atributos de tipo Modal intentando no tener que llegar a System Modal.

4o.DialogWindow: Es una típica caja de diálogo cuyas características pueden ir desde una sencillez aplastante, a una complejidad tremenda. Por defecto suelen ser de tipo Modal.

5o.DataWindow; Ventana dedicada a la visualización/edición de datos cuya explotación se produce como casi siempre en Visual Objects, en forma de clase. Tiene métodos que permiten la navegación por una base de datos como Append(), Skip( ), Delete(), etc., Además también puede contener todo tipo de controles típicos de Windows (ListBox, ComboBox, Botones, etc.).

El editor de ventanas se ofrece como herramientas básicamente intuitiva, aunque requiere ciertos conocimientos de Controles de Windows.

El editor de ventanas es usado para el diseño interactivo de todas las ventanas que requiera su aplicación.

Para diseñar estas ventanas, el editor de ventanas presenta una paleta de herramientas flotante. Para colocar un control en una ventana (como un boton, una lista o una barra de desplazamiento), simplemente presione la herramienta que desea en la paleta de herramientas y luego colóquelo en la parte que desea de la ventana.

También se pueden definir propiedades para sus ventanas y para los controles que haya puesto en ellas. Una propiedad importante de ciertos controles es un nombre de evento. Esto es porque en las aplicaciones de Windows, ciertos tipos de controles inician acciones o eventos; por ejemplo cuando el usuario presiona el botón OK en una caja de diálogo, el programa procesa la información capturada en la caja de diálogo y luego la encierra.

El editor de ventanas facilita el asociar acciones con este tipo de controles permitiendo especificar el nombre de un evento como una propiedad. También tiene la opción de usar cualquier método, ventana o reporte que sea visible para su aplicación, como un nombre de evento y puede especificar código fuente para un evento definido por el usuario desde el editor de ventanas.

La integración de los numerosos controles dentro del IDE provee de poderosos beneficios, uno de los cuales es la habilidad para crear ventanas de datos. Las ventanas de datos son datos dependientes porque conocen todas las características del Data Server sobre el cual trabajan.

Una ventana de datos sabe esto por la relación que se establece entre uno o más Data Servers. Una vez que una ventana de datos y un servidor son relacionados, puede ligar controles individuales en la ventana junto con los campos de la Data Server.

Cuando se establece una relación entre un control y un campo, este se liga automáticamente con la especificación del campo asociado.

Por su naturaleza las ventanas de datos son capaces de interactuar inteligentemente con los servidores de datos. Por ejemplo, estas pueden desplegar el contenido de un servidor de datos y tiene métodos programados para moverse sobre los registros y manipular los datos en el servidor.

Vamos a diseñar una sencilla ventana de Shell (heredada de la clase ShellWindow) a la que llamaremos SHOW:

Tras especificar el nombre y tipo de ventana, aparece una área de color gris que representa la ubicación y tamaño de la ventana que vamos a diseñar y que contendrá una rejilla de ayuda que se activará en la configuración. A la derecha

tenemos la ventana de propiedades, ratón, etc., la posibilidad de personalizar los diferentes eventos que se ejecutan asociados a la ventana. Por ejemplo podemos escribir un código que se ejecutará cada vez que se active, abra, cierre, mueva la ventana, etc...

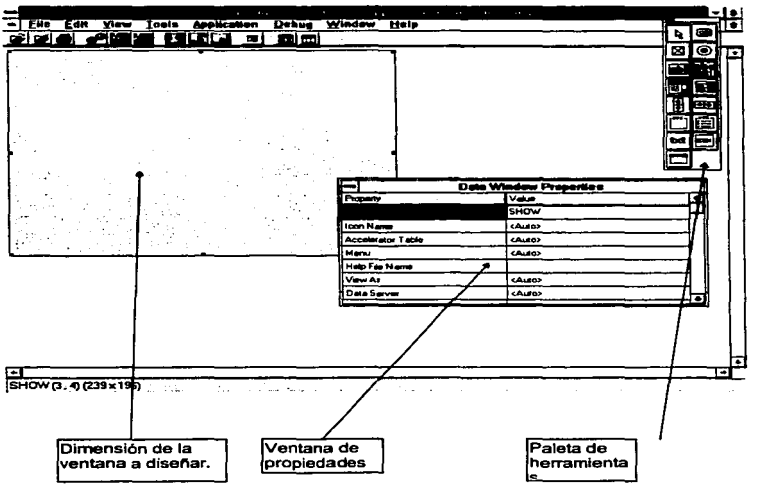


figura 3.2

Si decidimos guardar el resultado del generador se observa el siguiente código:

```
METHOD INIT (oParent) CLASS SHOW

Super: INIT(oParent)

Self: Caption := "Ventana de Shell"
Self: HyperLabel :=HYPERLABEL {#SHW, "Ventana de Shell",,}
Self: Origin:= Point {50, 430 }
Self: Size := Dimension {400,300}

return self
```

Obsérvese como ponerle título, etiqueta, Menú e Icono es muy sencillo.

Si cambiáramos la línea:

```
super:INIT (oParent)
```

Por:

```
super: INIT( )
```

Todas las ventanas de esta clase perdurarían aunque cerraremos la aplicación desde la que se las llama. Como regla general, y de manera automática, hay que mantener la jerarquía de ventanas y referenciar siempre a la ventana padre, si no queremos ventanas o cajas que se queden perdidas en el sistema al cerrar las que las llamaron.

Cajas de diálogo

La comunicación entre el usuario y una aplicación además de los menús, puede y debe hacerse usando cajas de diálogo. Estas cajas pueden servir perfectamente a la introducción de datos, tanto por teclado, como a la elección de opciones con el ratón pulsando botones y controles. Moviéndonos por la ventana de propiedades podemos observar la extraordinaria potencia del editor, ya que se permite definir hasta 31 eventos genéricos a activar la función de lo que se esté haciendo en la caja de diálogo. Por ejemplo, al entrar, al abrir, al pulsar un botón, al cerrar, etc.

Si pretendemos diseñar una caja de diálogo nos encontraremos con un editor similar al que nos aparecía inicialmente en la ShellWindow pero con una importante novedad, arriba a la derecha, tenemos una barra de herramientas que nos permite además incluir en una ventana los siguientes controles:

PushButton: Se les asocia un evento que se define como método de la misma clase que la ventana que los contiene, el cual se ejecuta al pulsar el botón. Se les puede cambiar tamaño, ubicación y fuente de texto.

CheckBox: Permite insertar la edición de valores lógicos NO EXCLUYENTES.

RadioButton: Permite insertar la edición de valores lógicos EXCLUYENTES, Normalmente se agrupan y el editor solamente permite la elección de uno de ellos, ya que la pulsación de uno desactiva automáticamente el que estuviera activo en ese momento.

SingleLineEdit: Editor de una sola línea. Permite editar todo tipo de valores, aunque a menos que se le especifique un FieldSpec, siempre trata la edición en modo carácter.

MultiLineEdit: Editor multilínea. Permite editar bloques de texto, y se le pueden asignar barras automáticas de scroll horizontal o vertical.

ListBox: Permite la edición, búsqueda y elección de elementos de una lista ya sea de una tabla de memoria (array) o una tabla perteneciente a una base de datos.

ComboBox: Permite la edición, búsqueda y elección de elementos de una lista ya sea de una tabla de memoria (Array) o una tabla perteneciente a una base de datos. Una ComboBox contrariamente a una ListBox solo se despliega cuando tiene el foco, y se le puede marcar el área máxima de visualización.

ScrollBars: Ascensores verticales u horizontales que permiten los típicos desplazamientos de Windows.

GroupBox: Permite Agrupar controles de forma genérica.

RadioButtonGroup: Permite agrupar controles específicos RadioButton.

FixedText: Permite incluir texto fijo, es decir no editable.

FixedIcons: Permite incluir iconos en la pantalla.

En el método Init() como se declaran los objetos contenedores de los controles de tipo Editor de línea, Editor multilínea, ComboBox, ListBox, etc.. Las cajas de diálogo suelen ser casi siempre MODALES, ya que en ellas se edita o se hacen preguntas que así lo requieren.

3.4 EDITOR DE INFORMES

El Editor de informes de VO es un producto comercial llamado CA-RET. Fundamentalmente se basa en un editor gráfico que permite diseñar informes, etiquetas y formatos de documento con algo tan sencillo como el ratón. Además lleva implícita una visualización previa y goza de todas las ventanas de Windows en cuanto a control de Impresoras y drivers de Impresión.

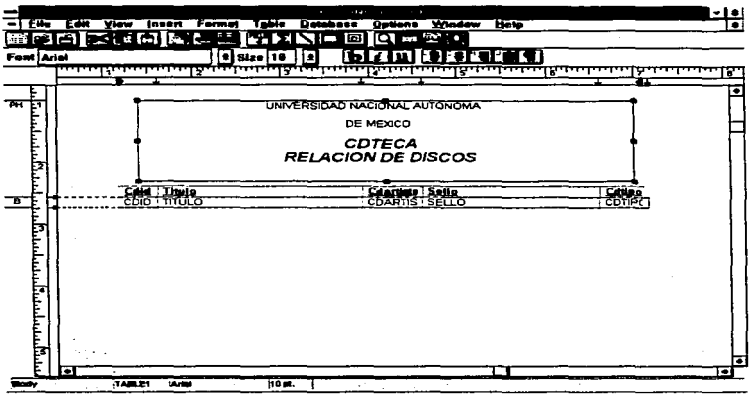


figura 3.3

Permite entre otras cosas, asociar diversos servidores de datos a un solo formato de listado y por consiguiente definir de manera simple las eventuales relaciones múltiples entre las diferentes tablas.

Sus formatos predefinidos son :

TABULAR	Listado Convencional en forma de tabla
FORM	Listado en forma de factura o nota de entrega
LABEL	Listado en forma de etiqueta
LETTER	Carta personalizada
FREE STYLE	Estilo libre que permite hacer casi cualquier cosa.

Además es posible tener tipos de letras distintos, inserción de gráficos y bitmaps en los listados, con todo esto se puede decir que el editor de informes resuelve la mayoría de las necesidades de una aplicación convencional.

Su implementación práctica se basa en lo siguiente:

1o.El programador diseña los listados, informes y documentos necesarios, salvándolos a disco. Tras la conveniente depuración, forman parte de la aplicación, aunque se ubican externamente, y pueden ser editados sin necesidad de cambiar el ejecutable.

2o.Al generar la instalación se incluyen en los disquetes los formatos de impresión y un runtime de CA-RET

3o.La aplicación activa el diálogo DDE con el runtime de CA-RET y ejecuta la previsualización con impresión por deseo del cliente/usuario o impresión directa a través del administrador de impresión de Windows y con los drivers de impresora instalados.

3.5 EDITOR DE ICONOS

El editor de iconos muy similar a los existentes en el mercado, permite crear nuevos iconos partiendo de la nada y abrir y editar iconos existentes en el disco en formato *.ICO. También permite su búsqueda en archivos EXE. DLLs, etc., por consiguiente podemos ejercer la "piratería icónica" de cualquier aplicación Windows.

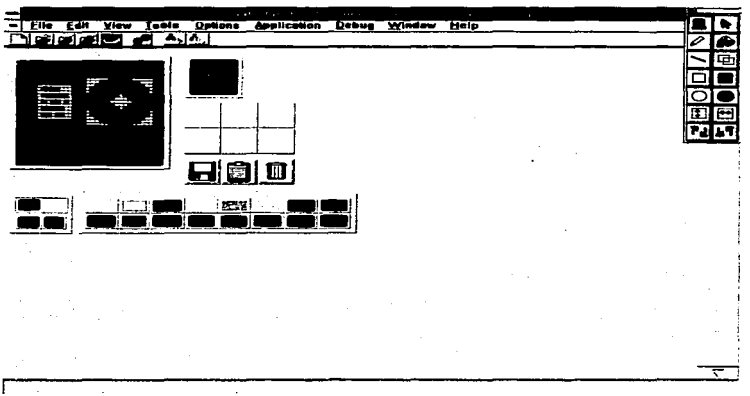


figura 3.4

Después de su edición el icono es salvado a disco, y en el repositorio se crea una entidad de clase icono, que podemos asignar a cualquier ventana posteriormente.

3.6 EDITOR DBSERVER

Visual Objects permite asignar un objeto a una base de datos, es decir, implementar todas y cada una de las acciones que se pueden emprender con ella en sintaxis OPP. Para el correcto diseño de objetos tan complejos, se nos da un editor intuitivo de sencillo manejo. Aunque podemos optar por atacar directamente a las DBF, tanto el editor de informes, como el de ventanas de datos, exigen trabajar con objetos de tipo *DBServer*.

De entrada debemos dar un nombre al *DbServer* que estamos editando, en caso de que nuestro diseño se quiera basar en un archivo de tipo DBF ya existente en el disco duro bastará con elegirlo con el ratón, para que se genere automáticamente todo lo necesario. Sin embargo si optamos por crear uno nosotros mismos, al igual que ocurre con cualquier generador de bases de datos debemos especificar:

Nombre del archivo
Ubicación en el árbol de directorios
Campos
Claves y driver de indexación

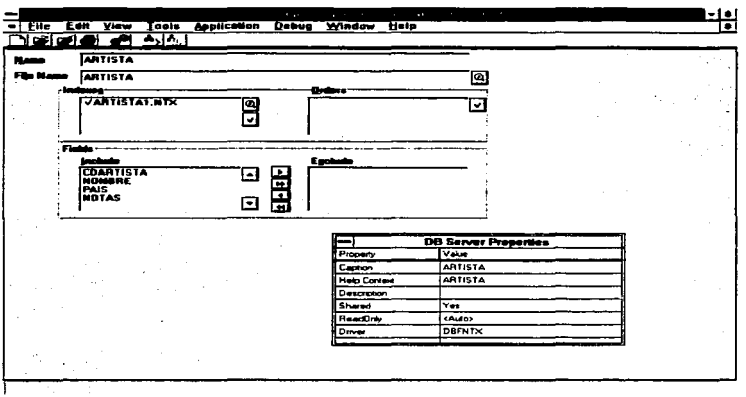


figura 3.5

Posteriormente se decide si se abre en modo de solo lectura, y/o en modo compartido, su driver de índices, etc... Bastará con exportarlo a disco para que se genere automáticamente el archivo DBF vacío y sus índices.

3. 7 EDITOR FIELDSPEC

El editor de campos planteado siempre como una clase dependiente de DBServer, permite al igual que lo haría un diccionario de datos, definir las características de los campos, picture, valid, tamaño, tipo, etc... Estas características se emplean en los controles, las ventanas de edición de datos, y en el editor de informes.

Property	Value
Name	CDARTISTA
Caption	Código del intérprete
Description	Introduzca el código del intérprete
[redacted]	[redacted]
FieldSpec	ARTISTA_CDARTISTA
FS Name	CDARTISTA
FS Caption	Código del artista
FS Description	Introduzca el código del intérprete
FS Help Context	CDARTISTA
Type	Numeric
Type Diagnostic	
Type Help	
Length	4
Length Diagnostic	
Length Help	
Decimals	0
Picture	@Z 9999

figura 3.6

FieldSpec a pesar de ser una clase aparentemente asociada a DbServer permite también la edición de variables de memoria ya que, asociados a un control de edición, permite controlar los procesos de rango, picture y validación pudiéndose asignar posteriormente el resultado de la edición a cualquier variable.

3.8 EDITOR DE CODIGO

Cuando pulsamos el ratón situados en cualquier entidad editable aparece el editor de código fuente. Goza de todas y cada una de las ventajas de un editor Windows ya que además de permitir el uso directo del NotePad para copiar y pegar bloques, permite las clásicas combinaciones de teclas para copiar, cortar, pegar, deshacer, etc...,basadas en las normas estándares.

Inicialmente basándose en la ToolBar de IDE podemos optar por abrir solamente la entidad en la que estamos ubicados (2o Icono empezando por la izquierda) o todo el código fuente de un módulo (3o. Icono empezando por la izquierda) de la figura 3.7.

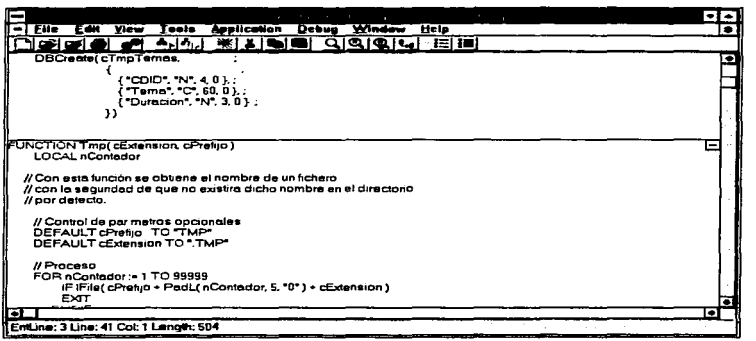


figura 3.7

Lo primero que observamos es que los colores que asigna la fuente son sensibles al tipo de código, son parametrizables y permiten adaptarlos a nuestro gusto. En la opción SETUP del menú FILE del IDE de Visual Objects podemos elegir el tipo de fuente que usaremos en el repositorio y en el editor. Los colores también se pueden asignar teniendo en cuenta que Visual Objects distingue:

BACKGROUND	-Color de fondo
KEYWORDS	-Palabras claves
COMMENTS	-Comentarios
TEXT	-Texto
CONSTANTS	-Constantes.

El editor permite realizar búsquedas y sustituciones con gran sencillez gracias a la ToolBar intuitiva y también encoger o expandir todas las entidades de manera a facilitar enormemente los procesos de depuración y escritura de código. Es más que aconsejable familiarizarse al máximo con el ya que es por regla general el entorno en el que más tiempo estamos ubicados a la hora de desarrollar.

Hay un punto negro que conviene resaltar, y es que los generadores tal como su nombre indica generan código fuente, que luego podemos ver y editar desde el propio editor de fuentes. Sin embargo cualquier modificación desde el generador elimina irremediablemente el código creado anteriormente y lo sustituye por el nuevo. Por consiguiente se recomienda NO TOCAR con el editor el código producido por los generadores ya que los mismo no leen previamente las modificaciones y borrarán irremediablemente lo creado. Este problema no está en modo alguno provocado por el editor, pero hay que tenerlo en cuenta.

3.9 OTRAS HERRAMIENTAS DEL IDE

3.9.1 Compilador

El IDE entre otras cosas es un compilador incremental. Eso quiere decir que de forma automática solo se recompilan los módulos que han sufrido modificaciones desde la anterior compilación, esto permite agilizar considerablemente los procesos de ensamble de aplicaciones.

3.9.2 Enlazador

El enlace también es incremental, y además permite la creación de ejecutables virtuales. Eso permite reservar la creación del ejecutable para la fase de entrega, y trabajar siempre con virtuales a la hora de desarrollar. Esta rápida ventaja consume enormes cantidades de memoria y debe usarse con prudencia en máquinas cortas de recursos, aunque evidentemente siempre se llega antes a ejecutar por ese medio que creando físicamente el fichero EXE. No obstante los ejecutables virtuales son siempre mas lentos que los reales.

3.9.3 Depurador

El depurador permite hacer un seguimiento puntual del código fuente, del contenido de las variables y constantes en tiempo de ejecución y por supuesto de las clases activas.

Para activar sus funciones, hay que pasar por la ventana de propiedades de una aplicación, y aplicarle el debugger, automáticamente todos los módulos de nuestro código se engalanan con la D mayúscula. Si quitamos o ponemos el debugger, deberemos siempre recompilar toda la aplicación, ya que el propio IDE invalida las anteriores compilaciones.

Si estando el cursor del ratón situado sobre un módulo o entidad pulsamos la tecla derecha del mismo podemos optar por activar el debugger en modo local y específico por módulo o entidad.

Sus funciones básicas son:

- Seguir el código paso a paso.
- Ejecutar hasta el cursor.
- Marcar puntos de ruptura en el código fuente.
- Ver expresiones.
- Ver clases.
- Ver datos.
- Ver contenido de Servers.

Una herramienta de la importancia y potencia de Visual Objects resultaría absolutamente inmanejable sin un depurador potente y de fácil aprendizaje.

CAPITULO

MULTIMEDIA

La descripción de multimedia es sencilla, ya que únicamente se debe interpretar la palabra y obtendremos el significado de este término: varios medios.

Un sistema multimedia es aquel capaz de manejar a un tiempo sonido, video, animaciones, gráficos y datos típicos. Igualmente es necesario que se produzca una interacción con el usuario y no ser una simple secuencia de acciones preprogramadas. De igual forma es deseable que todo esto se produzca en tiempo real.

Para poder considerar a un sistema como realmente multimedia, ha de ser capaz de utilizar todos los nuevos tipos de datos a un tiempo. Los distribuidores de software se encargan de lanzar las campañas, para que se adquieran los nuevos avances tecnológicos. Un programa de animación o un juego no pueden considerarse 100% multimedia puesto que carecen de algunas de las premisas que fijamos anteriormente.

Como en el concepto de multimedia mencionamos audio, animación, imágenes y video es necesario hablar acerca de esto.

4.1 AUDIO

Es posible tener con Windows 3.1 computadoras que reproduzcan sonido. Existen 3 tipos de audio que soporta Windows 3.1

4.1.1 Audio en formato Wave

El formato Wave es audio que ha sido convertido a formato digital estándar usando una tarjeta de sonido. Esta información digital puede ser vista y manipulada usando un editor de Waveform audio. La forma Wave de audio puede ser reproducida a través de hardware en la tarjeta de sonido que toma la información digital convirtiéndola en una señal analógica y la reproduce a través de bocinas. Estos archivos en forma Wave tiene extensión .WAV.

Los archivos wave son creados fácilmente a través de un micrófono en la grabadora de sonidos.

4.1.2 MIDI

Es un estándar internacional establecido en 1982 que especificó el cableado y el hardware para conectar computadoras e instrumentos musicales electrónicos.

Este estándar también incluye protocolos de comunicación para pasar sonidos de un lado a otro. Un archivo de sonido midi contiene información que describe las notas que son reproducidas por un sintetizador midi. Para que se puedan reproducir archivos MIDI es necesario seleccionar la configuración correcta para el dispositivo que reproducirá los archivos. Windows provee algunas de estas configuraciones para sintetizadores MIDI, el manual contiene indicaciones para un nivel base de un sintetizador o un nivel extendido. Si un sintetizador es nivel básico o extendido lo determina el número de instrumentos que puede reproducir simultáneamente y como toca algunas notas.

TIPO DE SINTETIZADOR	NIVEL	DISTRIBUCION
Instrumentos melódicos	Base	6 notas 3 instrumentos
	Extendido	16 notas 9 instrumentos
Instrumentos percusivos	Base	3 notas 3 instrumentos
	Extendido	16 notas 8 instrumentos

El nivel base puede tocar 6 notas distribuidas entre 3 instrumentos melódicos y 3 notas en 3 instrumentos percusivos. Los instrumentos melódicos están cada uno en diferente canal MIDI y todos los instrumentos recursivos son una llave base en un canal simple.

Si Windows no provee la configuración para el sintetizador que se quiere conectar o se tiene uno que no es soportado por Windows es necesario crear una nueva configuración MIDI. Una ventaja que tienen estos archivos sobre los Wave es que son más pequeños.

4.1.3 CD Audio

El tercer tipo de audio soportado por Windows es el Compact Disk Digital Audio (CD-DA), este es un formato de sonido digital, estándar usado por audio compacts. CD-DA puede ser reproducido por un CD-ROM drive y Windows como software. Red Book Audio es otro nombre que se le da al CD audio, es mejor que los archivos wave, pero requieren mucho más espacio en disco y solamente debe ser usado cuando la aplicación multimedia requiere un buen sonido, además no es almacenado como un archivo específico en el disco y los usuarios finales a la hora de que ejecuten su aplicación requieren un CD-Rom drive en la máquina en la que estén corriendo su aplicación.

Cuando se adquiere un CD-ROM drive la mayoría de las veces se adquiere junto con una tarjeta de sonido que interactua con el CD-ROM, para utilizar esta tarjeta se debe de tener Windows con todos los componentes de audio de las extensiones multimedia. Los drivers para tarjetas de sonido que no son soportados por Windows son proporcionados por el fabricante.

Existen dos tipos de CD-ROM drivers, los internos y los externos, pero los mas comunes y usados son los internos.

Para convertir la información de los directorios de los archivos a DOS es necesario tener cargado en la computadora el driver MSCDEX que es una abreviación de Microsoft CD Extensión. Este driver es proporcionado por fabricantes de Kits y si se instaló una unidad de CD-ROM driver por separado es necesario instalar este driver.

Windows 3.1 incluye un driver que facilita tocar audio Cds en CD-ROM drivers que tienen la capacidad de codificar audio internamente, este driver es llamado MCI CD Audio. Este driver debe ser instalado antes de que se pueda tocar audio en Cds en un CD-ROM.

Para que puedan interactuar la tarjeta de sonido con el CD-ROM es necesario que se conecten a través de una interface, y esta es otra de las fallas de las especificaciones MPC en los estándares para las interfaces que se usan entre la tarjeta de sonido y el CD-ROM, porque no son compatibles todas las tarjetas de sonido con los CD por que la interface que usan no es estándar.

Los kits multimedia que venden los fabricantes consisten en una tarjeta de sonido y un CD-ROM drive con una interface. La ventaja de comprar todo esto junto en un kit es que la tarjeta de sonido y el CD-ROM son compatibles porque no tienen problema con la interface que el fabricante proporciona, además de que también incluye en este kit el software para instalar la tarjeta y el CD-ROM.

4.2 IMAGENES

Desde que se crearon las computadoras se han utilizado imágenes, un ejemplo son las pantallas que crearon los primeros programadores, estas imágenes han ido evolucionando, en la actualidad la resolución de los monitores nos permite observar imágenes casi como si fueran una fotografía y además se utilizan gráficas, dibujos, fotografías captadas por un escáner etc. todo para que el hombre tenga mayor facilidad para trabajar con computadoras y le sean cada vez más útiles y con una mejor presentación.

Estas imágenes que en la actualidad se utilizan en multimedia tienen 2 formatos y se describen a continuación.

4.2.1 Bitmaps

Las imágenes bitmap están compuestas de una serie de bits en la memoria de la computadora que definen el color y la intensidad de cada pixel en una imagen. Aunque un gran número de paquetes gráficos pueden crear y ver formatos diferentes de archivos bajo Windows, los formatos recomendados para usar imágenes en aplicaciones Windows multimedia son bitmaps estándar (archivos con extensión .BMP), Microsoft device independent bitmaps (archivos con extensión .DIB) y archivos que han sido comprimidos usando una técnica de compresión de archivos con Run-Length Enconding (con extensión .RLE). El Paintbrush de Windows crea estas imágenes bitmap. El kit multimedia desarrollado por Microsoft incluye un programa editor de bits, aunque este editor no puede crear imágenes, puede convertir imágenes de otros formatos de archivos, cortar y rotar imágenes y cambiar la paleta usada para una imagen específica.

Bitmap incluye la mayor parte de imágenes usadas en ambiente Windows y la mayoría de imágenes usadas en el desarrollo multimedia.

4.2.2 Vector Graphics

Estas imágenes son almacenadas como un conjunto de instrucciones y deben ser desplegadas en formato metafile de Windows (archivos con extensión .CGM), son usadas cuando las secciones o dimensiones de una imagen deben ser modificadas frecuentemente.

Los paquetes de software que crean vector Graphics, a menudo retratan un esqueleto o instala un formato de la imagen. Los componentes individuales que estructuran la imagen pueden ser movidos o modificados antes de que la imagen sea reproducida. Reproducir es el proceso crea los colores y dimensiones de la imagen final.

4.3 ANIMACION

Animación en una PC es una serie de imágenes que son desplegadas en una secuencia rápida , engañando al ojo humano haciéndole pensar que está en movimiento. El adherir animación, simplifica ideas complejas que no son muy fáciles de comprender con imágenes simples.

4.3.1 Diseño de animación

Es el proceso de designar un diseño para cada pantalla vista y después hacer un movimiento ligero a través de una sucesión rápida. El diseño de animación es similar a una caricatura en que los cambios mínimos ocurren diseño a diseño.

4.3.2 Cast animation

Es el proceso de designar todos los movimientos de los objetos en un proceso separado, y después asignar un carácter particular a cada objeto. El carácter dirige la posición, característica, tamaño y coloración del objeto. El diseño de la película compuesta es formada por cada objeto, y el diseño se forma haciendo un movimiento ligero a través de una sucesión rápida que activa la ilusión del movimiento. Gold Disk´s animation Works son un ejemplo de productos que utilizan Cast Animation, esto hace simple para un artista novato crear animación después de haber incorporado multimedia.

Los archivos de animación dependen de los intérpretes usados para ejecutar secuencias animadas en Windows. La herramienta de animación que crea la secuencia animada determina el intérprete usado en tiempo de ejecución. Los intérpretes tienen diferentes capacidades, algunos son capaces de desplegar 256 colores y facilitan la sincronización de sonido para las secuencias animadas. Un intérprete es posible que se ejecute en una pequeña ventana con una resolución de 200 x 320, otros pueden tomar la pantalla entera. El paquete de animación que se use depende del tipo de aplicación que se empleará con la animación.

4.4 VIDEO

La diferencia entre el video y las animaciones es que el video describe imágenes de eventos reales almacenados en forma digital. Los archivos de imágenes de video siempre contienen audio tracks y son más largos que las imágenes animadas. El video debe de ser accesado de un disco duro o de un CD-ROM, una de las desventajas de estos archivos es que para 30 segundos de full motion video se requieren 500 mb de almacenamiento. Por lo regular estos archivos están comprimidos y se descomprimen mediante un software especial cuando van a ser vistos o almacenados. Aunque existen algunas aplicaciones buenas que usan imágenes o animación o ambas, nada es mejor que observar video en una

PC. El video en la actualidad es usado en teleconferencias, la siguiente generación de videos caseros, o tutoriales sofisticados.

Esta tecnología es costosa porque se tiene que considerar desarrollo, integración dentro de aplicaciones y espacio en disco además de una gran inversión en software y tiempo.

Existen algunos productos nuevos que facilitan la incorporación de video dentro de Windows, uno es Video para Windows que es un producto de Microsoft y ayuda en el desarrollo de captura, digitalización y compresión de video. Video para Windows es escalar y puede ser reproducido en una PC con un descomprimidor adicional de video. La calidad de la reproducción del video depende del poder del procesador que tenga el usuario y de su tarjeta de video.

4.5 MULTIMEDIA A TRAVES DE LOS AÑOS

Existen unas especificaciones mínimas para computadoras personales para que se les pueda considerar como Multimedia PC, estas especificaciones fueron desarrolladas por Microsoft y un conjunto de fabricantes de computadoras.

Este grupo formó Multimedia PC Marketing Council y establecieron un estándar en Hardware, dieron capacitación para usuarios finales y soporte para desarrolladores de productos Multimedia . Los miembros originales MPC Marketing Council fueron Compu Add, Creative Labs, Fujitsu, Headland Tecnologyl, Video Seven Inc., Olivetti, Media Vision, Microsoft, NCR, NEC, Philips, Tandy y Zenith Data Systems.

Los fabricantes miembros de esta asociación pagan por etiquetar sus productos con el logotipo de MPC Marketing Council creado por Microsoft que fue transferido en 1991 a MPC Marketing Council, esta asociación esta compuesta de los principales vendedores de computadoras personales y fabricantes de Kits para computadoras con procesadores 386 y 486.

Las primeras reuniones fueron coordinadas por Microsoft, pero después fueron transferidas a Software Publishers Association (SPA). Aunque SPA es oficialmente responsable del MPC Marqueting Council, Microsoft juega un papel importante para promover exhibiciones, conferencias y dar información a usuarios acerca de Multimedia PC.

En noviembre de 1990, las especificaciones de multimedia PC fueron anunciadas en la conferencia de desarrolladores de Microsoft Multimedia. Estas especificaciones describen estándares mínimos para Multimeda PC. Estos estándar incluyen dispositivos de hardware para facilitar el acceso a sonidos y

datos de un CD ROM drive y poner sonidos en formatos de archivos Wave y archivos MIDI.

Incluyeron en las especificaciones requerimientos para títulos MPC (aplicaciones desarrolladas para correr en MPC) que corren bajo Windows 3.0 con Extensión a Multimeda.

En diciembre de 1991, MPC Marketing Council mejoró las especificaciones MPC base, las cuales fueron originalmente basadas en una computadora con un procesador 80286 a 10 MHZ, al analizar los sistemas con multimedia se observó que este procesador era inadecuado para las aplicaciones desarrolladas y se cambiaron las especificaciones estándar a un procesador 80386 SX a 1 MHZ con 2Mb en Ram, un monitor VGA, un disco duro con 30 Mb, una tarjeta de sonido y un CD ROM.

En la actualidad MPC Council se dedica a capacitar a usuarios finales Multimedia, promover exposiciones para desarrolladores independientes y examinar periódicamente algunos productos multimedia.

4.6 QUE ES MULTIMEDIA PC

Multimedia PC se refiere a una computadora personal que contenga el Hardware necesario para tener las especificaciones mínimas basadas en los requerimientos de la extensión multimedia 1.0 de Windows 3.0 que a su vez han sido tomadas de las especificaciones MPC.

4.6.1 Hardware Requerido para Multimedia PC

Las especificaciones MPC definen un estándar para usar computadoras y reproducir, no hacer aplicaciones multimedia. El hardware para producir títulos multimeda y videotapes es mucho mayor que el que se requiere para reproducir aplicaciones.

Las últimas especificaciones MPC de 1992 no son capaces de reproducir completamente los videos para computadora por falta de capacidad, en la actualidad estas especificaciones no sirven para satisfacer las necesidades de las nuevas aplicaciones multimedia que cada vez crecen en capacidad y en número.

Las especificaciones minimas para una PC multimeda son las siguientes

- Una computadora con procesador 80386 SX
- 2Mb en memora Ram
- Monitor SVGA
- Mouse
- Tarjeta de sonido
- CD-ROM drive

4.6.2 Requerimientos Software para Multimedia PC

Las especificaciones multimedia aseguran a los desarrolladores máquinas para correr títulos multimedia. Como mencionamos anteriormente en los requerimientos de hardware, las necesidades de un desarrollador son diferentes de las de un usuario final. Un usuario final solo necesita tener Windows 3.1 para tener multimedia en una computadora. Windows 3.1 incorpora Extensiones multimedia, una colección de Dynamic Link Libraries (DLL), drivers. Antes de la versión 3.1, las extensiones Multimedia eran vendidas como un producto separado de Windows 3.0.

Para los usuarios finales, Windows 3.1 provee un número de drivers que instalan dispositivos multimedia a través del panel de control y además se encuentra una ayuda para explorar nuevas opciones de medios. Los drivers que accesan a la tarjeta de sonido y reproducciones de archivos en forma wave son llamados MCISEQ.DRV. Los drivers de Windows que accesan el CD-ROM drive y facilitan la reproducción de CD audio y usar archivos de datos basados en discos compactos son los MCICDA.DRV.

Windows 3.1 incluye los siguiente:

Panel de Control.-	Instala y configura dispositivos multimedia
Media Player .-	Reproduce archivos multimeda y facilita el control externo de dispositivos multimedia
Sound Recorder.-	Facilita tocar, registrar y editar archivos en forma Wave

Con Windows 3.1 un usuario final puede reproducir wave, midi y archivos de video. Con OLE (Object Linking y Embedding), el usuario puede incorporar esos elementos media dentro de otros documentos. Además el usuario puede incorporar cualquier software multimedia con la seguridad de que puede trabajar sin ningún problema.

Algunos títulos multimedia pueden necesitar drivers que no han sido instalados en el sistema, la mayoría son accesados del disco CDROM y todos los títulos tienen una rutina para instalar el software en el disco duro. Este software es siempre una colección de ejecutables, drivers y otras DLL, los cuales no requieren un espacio muy grande en disco, pero son necesarios para ejecutar el programa.

Un desarrollador de software necesita además de Windows 3.1 una copia de Microsoft Multimedia Developer's kit (MDK) si va a crear un título, y si quiere adherir elementos media a una aplicación Windows estándar es necesario que adquiera el Windows Software Developer's kit (SDK).

Si un desarrollador tiene estas extensiones multimedia dentro de Windows 3.1 tendrá acceso a los siguientes servicios:

Media Control Interface:	MCI es una interface de alto nivel que facilita el desarrollo de recursos para control de medios, como tarjetas de audio, reproductoras de películas y reproductores de videotapes.
Audio:	Los servicios de audio proveen un dispositivo de interface independiente del Hardware para audio,facilitando el desarrollo de aplicaciones multimedia con sonido adherido. Los desarrolladores pueden introducir Waveform, MIDI o sonido de disco compacto a sus aplicaciones, sin preocupaciones acerca de las especificaciones del hardware requeridas que proveen de capacidad de sonido.
Multimedia File I/O:	Estos servicios facilitan una I/O mediante bufers o sin ellos de ficheros multimedia. Es un perfecto soporte para el Standard RIFF(Resource Interchange file formato, formato de fichero para intercambio de recursos). Estos servicios son ampliables hacia procedimientos configurables de entrada/salida, que pueden ser compartidos por múltiples aplicaciones.
Joystick y temporizador:	Mediante estos servicios, windows pone a nuestra disposición soportes de mandos de juegos y eventos temporizados de alta resolución.

Los desarrolladores quienes adquieren el MDK reciben las siguientes herramientas:

Bit Edit.-	Facilita la edición de mapas de bits gráficos .
Pal Edit.-	Facilita crear o editar paletas de colores.
Wave Edit.-	Facilita editar y reproducir archivos en forma wave.
File Walker.-	Facilita visualizar y editar diferentes de archivos en diferentes formatos.
Convert.-	Facilita la conversión de archivos de datos de un formato a otro.

MDK contiene el Multimedia Viewer, una herramienta que se utiliza para crear títulos multimedia. Viewer facilita incorporar elementos media dentro de los títulos que se crearon incluyendo sonido, animación y video. El MKD incluye también lo siguiente:

C-Header Files and Librarys:

Contiene definiciones especificas multimedia y declaraciones requeridas por programas en código fuente de lenguaje C.

Sampple Aplications:

Incluye algunas aplicaciones simples como las siguientes: Incluye Bouncer.- un salvador de pantallas, Joyton.- una demostración de una aplicación joystick, Lava.- demostración de la paleta de animación, Mmplay.- sirve para examinar las funciones reproductoras de películas, RELApp.- crea una secuencia simple de una serie de mapas de bits.

On Line Reference Files:

Provee información completa acerca de funciones multimedia, mensajes, comandos y estructuras de datos.

La parte mas valiosa de MDK es la documentación de los manuales, en ellos se explica todo acerca de la interface del control de medios, servicios de audio de alto nivel, tipos especiales de video, servicio de jostyck y relojes, las funciones multimedia para mensajes, datos y tipos de estructuras y formatos de archivo multimedia. Esta información es crítica para cualquier desarrollador que crea aplicaciones multimedia. Antes de adquirir MDK, un desarrollador necesita

seleccionar el software para crear el titulo y el software editor de medios que servirá para introducir los media clips.

4.7 ALGO DE MULTIMEDIA DESDE VO

Vamos a realizar un ejemplo de multimedia, utilizaremos una DLL realizada en C para interaccionar de un modo más cómodo con el API multimedia de Windows. Es necesario explicar porque se ha utilizado C y no VO para codificar esta DLL. El motivo básico que nos ha llevado a programar esta DLL en C es el tamaño de la misma, ya que si bien VO permite generar un código tan compacto como el de C y tan eficiente como este, una DLL foreignhosted incluye de manera automática rutinas para el control de errores en runtime.

4.7.1 Elementos previos

No se va a describir la construcción de la DLL MMEDIA gracias a la cual podremos visualizar de un modo cómodo ficheros AVI desde VO.

Antes de poder usar una DLL externa desde VO es necesario definir los prototipos de todas las funciones, aquellos que deseemos utilizar, de la misma. Esta tarea es la realizada por el módulo que en parte es mostrado en el fuente siguiente.

```
_DLL FUNCTION VideoBottom () AS short PASCAL:mmeda.VideoBottom
_DLL FUNCTION VideoClose () AS void PASCAL:mmedia. VideoClose
_DLL FUNCTION VideoEnd () AS void PASCAL:mmedia. VideoEnd
_DLL FUNCTION VideoEndStart () AS void PASCAL:mmedia. VideoEndStart
_DLL FUNCTION VideoHome () AS void PASCAL:mmedia. VideHome
_DLL FUNCTION VideohWnd () AS HWND PASCAL:mmedia. VideohWnd
_DLL FUNCTION VideoLeft () AS shortint PASCAL:mmedia. VideoLeft
_DLL FUNCTION VideoOpen (      1pstrFile AS psz,   ;
                                hWndParent AS WORD )  ;
          AS BOOL PASCAL:mmedia.VideoOpen
_DLL FUNCTION VideoPause () AS void PASCAL:mmedia. VideoPause
_DLL FUNCTION VideoPlay () AS void PASCAL:mmedia. VideoPlay
_DLL    FUNCTION    VideoReversePlay    ()    AS    void    PASCAL:mmedia.
VideoReversePlay
_DLL    FUNCTION    VideoReveresStep    ()    AS    void    PASCAL:mmedia.
VideoReverseStep
_DLL FUNCTION VideoRight () AS sorthint PASCAL:mmedia. VideoRight
_DLL FUNCTION VideosShow () AS void PASCAL:mmedia. VideoShow
_DLL FUNCTION VideoStart () AS BOOL PASCAL:mmedia. VideoStart
_DLL FUNCTION VideoStep () AS void PASCAL:mmedia. VideoStep
```

```
_DLL FUNCTION VideoEndTop () AS shortint PASCAL:mmedia. VideoTop
_DLL FUNCTION VideoWindow (
                                AS void PASCAL:mmedia. VideoWindow
```

Veamos detenidamente la sintaxis empleada en cada una de las entidades mostradas en el fuente anterior.

_DLL FUNCTION <Func> ((<Para>)) AS <Ret>: <DLL>.<Nom>| <Pos>

_DLL es el prefijo que aplicado sobre FUNCTION (si estamos declarando una función) METHOD (si es la declaracón de un método) indica que se está definiendo una entidad residente en una DLL.

< Func> Nombre que utilizaremos desde el código fuente de VO para referimos a la función ó método.

< Para > Nombre de los parámetros, separados por comas, que identificará los diferentes valores que acepta la función.

< Ret> Tipo de dato retomando por la función, todas las funciones de API de Windows siguen la conversión pascal de llamada a función. Lo que nos obliga a declarar la función al menos como PASCAL.

< DLL > Nombre del fichero que contiene la DLL (incluida su rutra no se encuentra en uno de los directorios de la ruta de búsqueda del DOS y Windows)

< Nom\ Pos> Nombre o posición ordinal de la función dentro de la DLL si desconoce el número y se desea referirse por él, se recomienda que se adquiera alguno de los múltiples analizadores de ejecutables, una DLL no es otra cosa que un nuevo tipo de ejecutable. Nosotros solemos utilizar EXEHDR que es facilitado por Microsoft junto al compilador de C++.

Precisada la sintaxis de esta unidad podemos interpretar cualquier módulo de definición de DLL para VO que caiga en nuestras manos, así como crear aquellos que necesitemos para nuestros desarrollos.

Expliquemos brevemente para que sirve cada una de las funciones construidas en la DLL.

Video Start() AS void PASCAL

Situa el AVI en el último frame (fotograma) del mismo.

VideohWnd() AS HWND PASCAL

Retorna el handle de la ventana creada para visualizar el AVI.

Video Window (iLet AS SHORTINT)
! tOP as shortint
! Right AS SHORTINT
! bOTTOM as shortint) AS void PASCAL

Fija la posición y dimensiones del rectángulo donde se visualizará el AVI. Le indicamos coordenadas de la esquina superior izquierda (Left, !Top) y esquina inferior derecha (!Right, !Bottom).

VideoOpen (lpstrFile AS psz, hWndParent AS word) AS BOOL PASCAL

Realiza la apertura del fichero AVI indicado como primer parámetro, dentro de la ventana, de la que indicamos su handle como segundo parámetro.

VIDEO sHOW() as Void PASCAL

Visualiza el primer frame del AVI, en la superficie que hayamos indicado o en la que fue grabado.

Video Close () AS void PASCAL

Cierra el dispositivo de video.

VideoLeft() AS int PASCAL

Retorna la coordenada en X de la esquina superior izquierda.

Video Top() AS int PASCAL

Retorna la coordenada en Y de la esquina superior izquierda original del AVI.

VideoRight() AS int PASCAL

Retorna la coordenada en X de la esquina inferior derecha original del AVI.

Video Bottom() AS int PASCAL

Retorna la coordenada en Y de la esquina inferior derecha original del AVI.

VideoPlay() AS void PASCAL

Comienza la reproducción inversa del video y audio almacenados en el AVI.

VideoPause() AS void PASCAL

Paraliza la reproducción del AVI, mostrando el frame sobre el que se encuentre.

VideoHome() AS void PASCAL

Situa el AVI al principio del mismo.

VideoEnd() AS void PASCAL

Situa el VI al principio del mismo.

VideoStep() AS void PASCAL

Avanza un frame dentro de la secuencia lógica del AVI.

VideoReverseStep() AS void PASCAL

Retrocede un frame de la secuencia lógica del AVI.

Para utilizar esta DLL desde nuestras aplicaciones únicamente habremos de situar el módulo con las declaraciones de funciones dentro de nuestra aplicación.

4.7.2 Ventanas de Visualización.

Nuestra primera tarea será definir la ventana donde interaccionaremos con el fichero AVI.

Para crear la ventana principal hemos entrado en el editor de ventanas (Window Editor) y seleccionado la creación de una nueva Dialog Window. A esta ventana le indicamos Video como nombre de la entidad.

Posteriormente haciendo uso de las características del editor de ventanas, realizamos un diseño.

Creamos un GroupBox dentro del cual se visualizará el AVI. Añadimos cuatro PushButtons en los que habremos situado la lógica de nuestra ventana. Para esta tarea presionaremos sobre la propiedad Click event dentro de la ventana de propiedades. Tras esto accedemos al editor de código fuente donde especificaremos el código que deseamos se ejecute.

La misión del botón Play es comenzar desde el principio de la secuencia la reproducción de la misma, para tal fin insertaremos el siguiente código:

```
METHOD Play ( ) CLASS Video
videohome ( )
videoPlay ( )
```

El botón Stop detendrá la ejecución del AVI mediante el siguiente código:

```
METHOD Stop ( ) CLASS VIDEO
        VideoPause ( )
        VdeoHome ( )
```

La lógica más compleja es la especificada en el botón pausa:

```
METHOD Pausa ( ) CLASS VIDEO
        STATIC 1Estado := TRUE AS LOGIC
        IF 1Estado
                Self:oCCPausa:Caption := 'Continua'
                1Estado := FALSE
                VideoPause ()
        ELSE
                Self:oCCPausa:Caption := 'Pausa'
                1Estado := TRUE
                VideoPlay ( )
        END IF
```

Este botón cambia el caption (texto del boton) hacia Continua o Pausa, dependiendo del estado del flag almacenado en la variable estática al método Estado. Las acciones tomadas al presionar el tercer botón lógicamente varian en función de la bandera.

El último botón de la caja de diálogo comienza, desde la secuencia donde se encuentra la reproducción de AVI. el discurrir inverso de la misma.

Una vez definida toda la lógica dentro de nuestra caja de diálogo, abandonamos el editor de ventanas almacenando todos los cambios realizados. En este momento el generador de VO realizará parte del trabajo pesado por nosotros, definiendo las entidades necesarias para que nuestra ventana funcione como hemos indicado en el editor.

4.7.3 Método start

Toda aplicación VO ha de comenzar bien por una función Start() o el método Start de la clase App, en este caso hemos seleccionado el método.

```
METHOD Start( ) CLASS App
        LOCAL oVW AS Video

Enable3dControls ( )
ovw := Video  ( )
// Permitimos Drag & Drop
ovw:= EnableDragDropClient ( TRUE )

VideoStart ( )
ovw:=Show ( )
ovw:= EnableDragDropClient ( FALSE )
```

Tras declarar la variable oVW como LOCAL y tipificarla de forma que solo pueda contener objetos de la clase Video, activamos la visualización tridimensional mediante la llamada a la función Enable3dControls( ).

Asignamos a oVW el objeto retomando por la construcción e inicialización de la clase Video. Esta clase fue generada de forma automática por el generador cuando almacenamos nuestro diseño de caja de diálogo.

Enviamos el método EnableDragDropClient() de modo que nuestra ventana será sensible a la llegada de ficheros AVI mediante la técnica de arrastrar y soltar. Esto añade la posibilidad de que VO llame al método Drop() enviándole como argumento un objeto DragEvent.

Iniciamos el dispositivo de video mediante la función VideoStart() situada en la DLL media.

Mostramos mediante el método Show() la ventana. Este método realiza en bucle de proceso de eventos de la ventana, ya que oVW es una ventana modal a nivel de aplicación.

Cuando creamos la aplicación se ejecuta el método EnableDragDropClient() remitiéndole como parámetro FALSE para que desactive oVW como cliente del servicio Drag&Drop de Windows.

Veamos a continuación cuales son las acciones a realizar cuando nuestra ventana recibe un evento que le indica la recepción de ficheros vía Drag&Drop. En esta situación VO ejecuta el método Drop de nuestra ventana.

```
METHOD DROP(oDragevent) CLASS Video
      LOCAL nFiles:= oDragEvent : FileCount

IF nFiles > 0
      IF File (oDragEven:FileName (1), 4) = = '.AVI'
            VideoClose ()
            Self:VideoOpenFile (oDragEvent:FileName(1)
            Self: Play ( )
      ELSE
            MessageBox(Self:Handle(), ;
                        " No es un fichero AVI", ;
                        " Error ", ;
                        MB_ICONSTOP )
      END IF
END IF
```

En el parámetro oDragEvent disponemos de un evento DragEvent el cual dispone, entre otra información, del número de fichero entrantes vía el servicio D&D. Asignamos este número a la variable nFiles.

Comprobamos que existan ficheros, que el recibido en primer lugar exista y que sea un fichero con extensión AVI. Para acceder al nombre de fichero que ocupa una posición determinada en la lista de ficheros entrantes, utilizamos el método FileName() del objeto oDreagEvent. Este método recibe como parámetro el número del fichero entrante del cual deseamos recuperar su nombre

En caso de no cumplirse todas las condiciones anteriormente expuestas, mostramos una caja de diálogo mediante la función del API de Windows MessageBox(). Si todo fue correctamente, cerramos el posible AVI que se encontrará en visualización y abrimos el nuevo mediante la ejecución de método VideoOpenFie(). Este método será mostrado a continuación. Por último enviamos el método Play() para comenzar la ejecución del video.

4.7.4 Apertura del AVI

Mediante el método VideoOpenFile() realizamos la apertura y situación del AVI dentro del GroupBox de nuestra ventana.

```
METHOD VideoOpenFile ( cFile ) CLASS Video
        VideoOpen ( cFile, Self:Television:Handle () )
        VideoWindow( (Self:Television:Size:Height - VideoBottom () /2,:
                (Self:Television:Size:Height - VideoBottom () /2,:
                                VideoRght () ,:
                                VideoBottom () )
VideoShow ()
```

Mediante el cálculo realizado en la llamada a la función Videowindow() centramos el AVI con respecto al GroupBox. La clase GroupBox dispone de un dato llamado Size, en este dato se almacena un objeto de la clase BoundingBox. Dicha clase es utilizada en toda la jerarquía de VO para contener información acerca de zonas rectangulares de pantalla. Dos de los datos de esta clase son los que utilizamos ahora: Width y Height (ancho y alto)-

Cuando cerramos la ventana, y por tanto la aplicación, se captura el evento de petición de cierre, reescribiendo parcialmente el método gestor de eventos QueryClose().

```
METHOD QueryClose ( oQCE ) CLASS Video
        VideoClosse ()
        VideoEndStart ()
RETURN TRUE
```

Mediante la llamada a VideoEndStart(), liberamos el dispositivo de video consumido al arrancar la aplicación por medio de la función VideoStart()

4.7.5 Construir y Ejecutar

Tan sólo nos queda compilar y ejecutar la nueva aplicación creada. No olvidar situar la DLL media en un lugar localizable por Windows, por ejemplo el directorio System que cuelga del directorio donde tenga instalado Windows.

CAPITULO

APLICACION CON VO

Un sistema orientado a objetos no se puede analizar y diseñar como si fuera un sistema programado estructuralmente. Este tipo de sistemas tienen una metodología diferente, puesto que no son iguales la metodología tampoco lo es, y es por esa razón que hablaremos un poco de esto, y quiero decir un poco por decir casi nada, porque si profundizamos en el análisis y diseño orientado a objetos nos llevaríamos un trabajo completo de este tema.

Vamos a empezar con una explicación general de los componentes del Análisis y el Diseño según lo que plantea James Matin en su libro "Análisis y Diseño Orientado a Objetos", para después hacer una analogía con el método tradicional.

En el Análisis y en el Diseño se construyen 2 tipos de modelos que están relacionados:
- Un modelo de los tipos de objetos y sus estructuras.
- Un modelo de lo que ocurre a los objetos.

Estos modelos se representan mediante diagramas llamados esquemas. Los esquemas de objetos muestran las estructuras del objeto y los esquemas de eventos muestran lo que ocurre a los objetos.

El análisis y Diseño Orientado a Objetos se puede resumir en la siguiente figura 5.1.

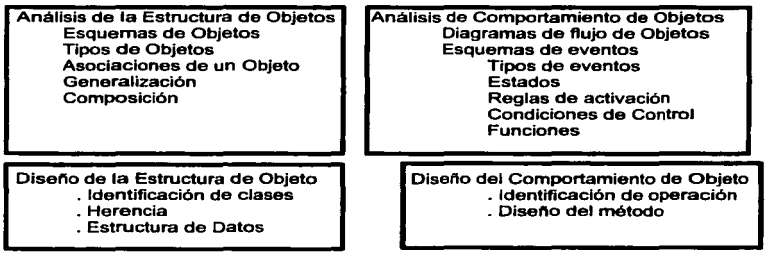


figura 5.1

El Análisis y Diseño Orientado a Objetos, tiene dos aspectos. El primero se refiere a los tipos de Objetos, clases, relaciones entre los objetos y herencia; se le conoce como Análisis de la Estructura de Objetos (AEO) y Diseño de la estructura de Objetos (DEO). El otro aspecto se refiere al Diseño de Comportamiento de Objetos (DCO).

Ya que hemos visto los componentes del Análisis y Diseño Orientado a Objetos vamos a hacer una analogía con el Análisis estructurado.

En las metodologías tradicionales, los analistas diseñadores y los programadores tienen distintos modelos conceptuales.

TECNICAS TRADICIONALES

Análisis	Diseño	Programación
Diagramas de relación entre entes	Diagramas de flujo de datos	Cobol PL / I
Descomposición funcional	Tablas de estructura	Fortran
Diagramas de dependencia entre los procesos	Diagramas de acción	C

Las técnicas Orientadas a Objetos utilizan el mismo modelo conceptual para el análisis, diseño y programación.

TECNICAS ORIENTADAS A OBJETOS

Modelo de los objetos
Declaración de Objetos
Manejo de Objetos

figura 5.2

Cuando el desarrollo tradicional pasa de una etapa a otra, a veces se pierde la información y aparecen conflictos y ambigüedades. La transición consume tiempo y a menudo reduce la calidad del producto final.

En las técnicas Orientadas a Objetos, todos utilizan el mismo modelo conceptual. Analistas, diseñadores, programadores y los usuarios finales. Todos piensan en los tipos de Objetos, los objetos y su comportamiento. Establecen jerarquías de tipos de objetos o de clases donde los subtipos comparten las propiedades que se le heredan. Piensan en términos de objetos compuestos de otros y utilizan la generalización y el encapsulado. Piensan en eventos que cambian los estados de los objetos y activan ciertas funciones y operaciones.
La transición del Análisis al Diseño es tan natural que a veces es difícil especificar el punto final del Análisis y el punto inicial del Diseño.
Solamente hemos mencionado las partes que componen el Análisis y el Diseño OO, ahora hablaremos un poco acerca de cada una de las etapas sin profundizar.

5.1 ANÁLISIS DE LA ESTRUCTURA DE OBJETOS (AEO)

El análisis de la estructura de objetos define las categorías de los objetos que percibimos y las formas que los asociamos. En este análisis se crean modelos que son una guía para la definición de clases y sus estructuras de datos en el diseño de la estructura y comportamiento del objeto.

5.1.1 Objetos y tipos de Objetos

Durante el AEO el equipo de Análisis se preocupa más por identificar los tipos de objetos que por identificar los objetos individuales en un sistema. Los tipos de objetos son categorías de objetos.
Los tipos de objetos son importantes, puesto que crean los bloques conceptuales de construcción para el diseño de sistemas. En la programación OO estos bloques de construcción guían al diseñador en la definición de las clases y sus estructuras de datos. Los tipos de objetos son un índice para los procesos del sistema.
Los tipos de objetos que definimos y utilizamos son variados, puesto que los elegimos con base en la comprensión de nuestro mundo. Así funciona nuestra mente, por ejemplo una persona puede considerar al objeto César como Hombre. Su jefe lo considera un empleado, etc.

5.1.2 Asociaciones de Objetos

Los tipos en que se clasifican los objetos de nuestro entorno son vitales en el análisis OO, otra cosa importante es modelar la forma como los objetos se asocian entre sí, por ejemplo, en la figura 5.3 se representa una forma de expresar la asociación entre dos tipos de objetos.

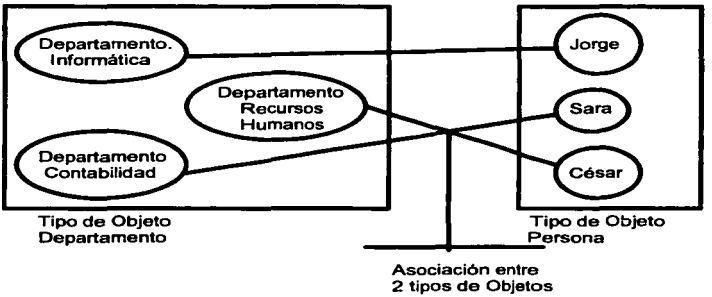


figura 5.3

En el análisis, es útil nombrar a las asociaciones e indicar la cantidad de objetos de un tipo dado que se deben asociar con los objetos de otro tipo, puesto que esto le da significado y aumenta la comprensión de la asociación.

5.1.3 Jerarquías de Generalización

La generalización es el resultado de distinguir un tipo de objeto como más general. Todo lo que se aplique a un tipo de objeto también se aplica a sus subtipos. Cada instancia de un tipo de objeto es también una instancia de sus supertipos, un tipo puede tener subtipos y de esto se observa la jerarquía de un tipo con varios.

Ejemplo:

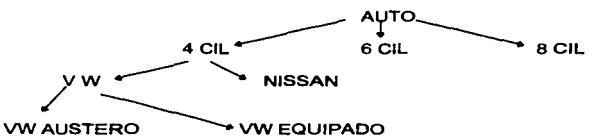


figura 5.4

Las jerarquías de generalización son importantes para el desarrollo de OO, por 2 razones. La primera es que el uso de supertipos y subtipos proporciona una herramienta útil, para describir el mundo del sistema de aplicación. La segunda es que indica las direcciones de herencia entre las clases en los lenguajes de programación Orientada a Objetos.

La comprensión de un modelo suele ser más sencilla si los tipos de objetos y relaciones se representan mediante un diagrama de relación entre objetos.

5.2 ANÁLISIS DEL COMPORTAMIENTO DE OBJETOS (ACO)

En el Análisis del Comportamiento de Objetos (ACO) realizamos esquemas de eventos que muestran eventos, la secuencia en que ocurren y como los eventos cambian el estado de los objetos. Así, los esquemas de eventos se deben expresar en términos de esquemas de objetos, puesto que los eventos cambian el estado de determinados tipos.
El AEO y el ACO están íntimamente relacionados. No se dan en forma separada, sino que se desarrollan juntos para formar modelos y diseños integrados.

5.2.1 Estado de un Objeto

Un objeto puede existir en varios estados. Por ejemplo: un objeto, reservación de habitación, puede ser una instancia de alguno de los siguientes tipos de objeto:

- Reservación solicitada
- Reservación en lista de espera
- Reservación confirmada
- Reservación cancelada
- Reservación satisfecha
- Reservación archivada

Tales tipos de objetos se suelen percibir como estados posibles del ciclo vital de un objeto.
El estado de un objeto es la colección de los tipos de objeto que se aplican a él.
Esta es una definición del estado de un objeto pero cuando se implanta en un lenguaje de programación Orientada a Objetos, el estado de un objeto es registrado en los datos almacenados en relación con el objeto. El estado de objeto se determina mediante las clases y valores de los campos de datos, asociados con el objeto. Por lo tanto una definición de estado utilizada comúnmente por los programadores 00 es:
El estado de un objeto es la colección de las asociaciones para ese objeto. Estas dos definiciones no tienen conflictos entre ellas, sino que son dos formas de ver el mismo estado del objeto.

5.2.2 Eventos

En el análisis orientado a objetos el mundo se describe en términos de los objetos y sus estados, así como de los eventos que modifican esos datos.
Un evento es un cambio en el estado de un objeto.
Sin los eventos el mundo no cambiaría. En un mundo sin eventos podríamos construir y generalizar bases de datos sin preocuparnos por actualizarlas. Sin embargo en las aplicaciones, si debe cambiar el contenido de las bases de datos y por lo tanto existen cambios y como deseamos saber de estos, cambiar y reaccionar de una forma adecuada, ante ellos, es necesario entender y modelar los eventos.

5.2.3 Tipos de Eventos

No es necesario, conocer cada evento que ocurra en una organización: solo los tipos de eventos. Así como hablamos de tipos de objetos e intancias de tipos de objetos, podemos hablar de tipos de eventos e instancias de tipos de eventos.
Los tipos de eventos indican los cambios sencillos en el estado de un objeto, ejemplo: cuando se deposita en una cuenta bancaria o se actualiza el salario de un empleado.

Los tipos de eventos describen las siguientes formas de estado:

- Un objeto se crea
- Un objeto se termina
- Un objeto se clasifica como una instancia de un tipo de objeto
- Un objeto se descalifica como una instancia de un tipo de objeto
- Un objeto cambia de clasificación
- El atributo de un objeto se cambia.

Los objetos pueden asociar un objeto con otro, por ejemplo, cuando un objeto se clasifica como empleado, debe estar asociado con un Departamento. Un evento clasificará al objeto como empleado. Otro evento creará una asociación entre el objeto empleado, y un objeto Departamento.

5.2.4 Operaciones

En el análisis Orientado a Objetos, el término operación se refiere a una unidad de procesamiento que puede ser solicitada. El procedimiento se implanta, mediante un método. El método es la especificación de como llevar a cabo la operación. Es el guión de la operación a nivel de programa, el método es el código que implanta la operación.
Las operaciones se invocan y la invocación es una instancia de una operación. Una operación puede o no cambiar el estado de un objeto. Si lo cambiará, ocurriría un evento. Las operaciones se representan mediante cuadros con esquinas redondeadas. Los tipos de eventos se representan mediante triángulos conectados a la caja.

figura 5.5

5.2.5 Reglas de Activación

Cuando ocurre un evento, lo usual es que el cambio de estado active el llamado a una o más operaciones por ejemplo, si se retira mercancía de un almacén y la cantidad conservada en éste baja de cierto nivel, ello puede activar una operación para volver a realizar un pedido.

Las reglas de activación definen la relación entre la causa y el efecto. Siempre que ocurra un evento de cierto tipo, la regla de activación invoca a una operación ya definida.

Un tipo de evento puede tener varias reglas de activación, cada una de las cuales invoca a su operación en paralelo. Las operaciones paralelas pueden producir diferentes cambios de estado en forma simultánea.
Además, una operación puede ser invocada por varias reglas de activación.

Por ejemplo, en la figura 5.6 Se activa la operación generar cheque cuando ocurre un evento cheque solicitado o fin de mes. En otras palabras, cualquier forma de activación hace que se ejecute la operación.

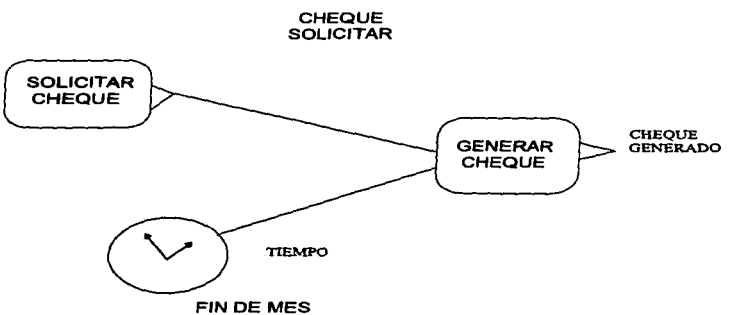


figura 5.6

5.2.6 Condiciones de Control

Una operación puede ser invocada por una o varias reglas de activación. Antes de invocar a una operación se verifica su condición de control. Si los resultados de evaluación de la condición son verdaderos, se invoca a su operación, si son falsos, no se invoca la operación.
Siempre que haya que verificar una condición de control antes de invocar a una operación, ésta se representa mediante un símbolo de forma de rombo antes de la operación.

Las condiciones de control también pueden actuar como puntos de sincronización para el procesamiento en paralelo. Las condiciones de control garantizan que un conjunto de eventos, esta completo antes de proceder con una operación.

figura 5.7

5.2.7 Aislamiento de la Causa y el Efecto

Cada operación lleva a cabo su tarea sin importar lo que ocurre en otra parte. Una operación es invocada por varios mecanismos de activación, ejecuta su método y se espera que ésta modifique el estado de un objeto. La operación no sabe "cuando" la activó ni por que, además no sabe si se activarán otras operaciones a partir de su evento. No reconoce su causa o efecto, solo sabe que es invocada para producir un cambio de estado en un objeto dado.

Este aislamiento de las consideraciones de causa y efecto es necesario para que la operación pueda volver a utilizarse en muchas otras aplicaciones.

En el análisis de la estructura de objetos trazamos diagramas para presentar la estructura de los tipos de objetos. En el análisis del comportamiento trazamos diagramas para mostrar la interacción dinámica de los eventos. Estos diagramas se conocen como esquemas. Un esquema de objetos expresa el tipo de objetos y sus asociaciones en un sistema dado. Un esquema de eventos expresa un guión de procesamiento que cambia los estados de los objetos.

5.2.8 La Analogía del Análisis y el Diseño

Los analistas, diseñadores y técnicos deben poseer el mismo modelo del sistema. Cuando el modelo pasa del análisis a la implantación, se añade mayor detalle, pero en esencia es lo mismo. Esto no sucede en lo programación estructurada. En las técnicas OO, el analista identifica los tipos de objetos y su herencia y piensa en los eventos que cambian el estado de los objetos.

El proceso de pensamiento fluye de manera tan natural, del análisis al diseño que es difícil decir donde termina el análisis y donde comienza el diseño.

Los usuarios finales también deben pensar en términos de tipos de objetos, de eventos, de cambios de estado en los objetos y de reglas de la empresa que activan y controlan a los eventos. Es por esto que debemos de garantizar que los esquemas que se diseñen ayuden a los usuarios a pensar en todos estos términos ya mencionados.

5.2.9 Diagramas de Flujo de Objetos

Los esquemas de eventos son adecuados para la descripción de procesos en términos de eventos, de reglas operacionales. Pero para procesos grandes y complejos no es lo adecuado. En ocasiones un área del sistema es demasiado compleja como para expresar la dinámica de los eventos y las formas de activación. En situaciones como esta es cuando se utiliza un diagrama de flujo de objetos (DFO).

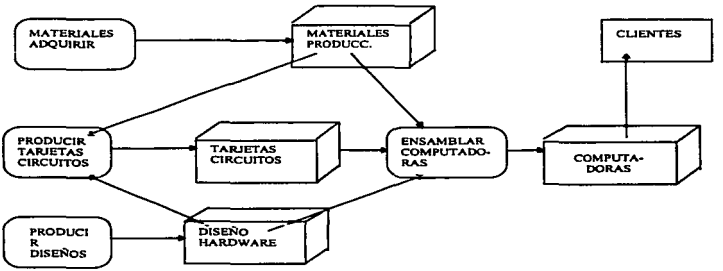


figura 5.8

Los diagramas de flujo de objetos son parecidos a los diagramas de flujo de datos, puesto que muestran las actividades que interactúan con otras. En los diagramas de flujo de datos, una interfaz transfiere datos. En la técnicas OO, no se limita a la transferencia de datos, sino que el diagrama debe representar cualquier tipo de cosa que se transfiera de una actividad a otra, ya sean pedidos, artículos, diseños, servicios, hardware o datos. El diagrama de flujo de objetos indica los objetos que se producen y las actividades que los producen e intercambian información.

Los diagramas de flujo de objetos tienen como los diagramas de flujo de datos, cajas de actividades, con esquinas redondeadas, cajas sombreadas de agentes externos y la dirección de líneas de flujo. Lo que cambia es que no existe el símbolo de almacenamiento de datos. En su lugar, se utiliza una caja tridimensional.

Los diagramas de flujo de objetos describen los objetos y cómo se producen y consumen.

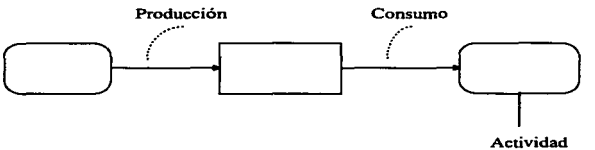


figura 5.9

El producto es el resultado final que satisface el propósito de la actividad. Los productos son el consumo de otras actividades que le añaden valor al producto consumido.

A cada paso se crean cualidades nuevas, mas complejas y sutiles. De esta forma el diagrama de flujo de objetos se puede utilizar para la planeación estratégica de la empresa.

Los diagramas de flujo de objetos se pueden utilizar para la planeación estratégica de la información, además pueden representar el modelo en forma ascendente o descendente.

Los diagramas de flujo de objetos son útiles para los modelos de organización en forma descendente a nivel estratégico porque las actividades se pueden descomponer en diagramas de flujo de objetos. En un nivel más detallado del comportamiento, también es correcto expresar los aspectos dinámicos de los esquemas de eventos. Una actividad se puede expresar en términos de un diagrama de flujo de objetos, un esquema de eventos, o ambos.

Para expresar un proceso de forma más rigurosa y poder generar un código, lo adecuado es un esquema de eventos. Un diagrama de flujo de objetos es útil para representar las estructuras básicas de control y el flujo del procesamiento, cuando no se entienda totalmente la dinámica de eventos y las claves de activación.

5.3 DISEÑO DE LA ESTRUCTURA Y COMPORTAMIENTO DE UN OBJETO.

El análisis Orientado a Objetos tiene dos aspectos:

El análisis de la estructura de objetos (AEO) y el análisis del comportamiento de objetos (ACO).

En el diseño orientado a objetos, se puede describir la misma división: el diseño de la estructura de objetos (DEO) y el Diseño del Comportamiento de Objetos (DCO). Los lenguajes de programación Orientada a Objetos tienen estructuras de datos y métodos, ambos sujetos a herencia y combinados en unidades llamadas clases. Por esto, el (DEO) y el (DCO) están entrelazados.

En el diseño de la estructura y comportamiento de objetos se identifican los componentes siguientes:

* ¿Qué clases se implantarán? Los tipos de objetos en el AEO serán la guía en esta decisión.
* ¿Qué estructura de datos utilizará cada clase? Se puede hacer un diagrama para representar la estructura de datos.
* ¿Qué operación ofrecerá cada clase y donde estarán sus métodos? Se enumeran las operaciones y se especifican sus métodos en determinado momento.
* ¿Cómo se implantará la herencia de clases y cómo afectará ésta las especificaciones de los datos y las operaciones?
* ¿Cuáles son las variables? Se identifican las probables variantes de las clases.

5.3.1 Clases a Implantar

En el análisis de estructura de objetos, identificamos los tipos de objetos; en el diseño de la estructura de objetos nos centramos en la implantación de esos tipos de objetos. Una clase especifica, la estructura de datos y los métodos operativos permitidos que se aplican a cada uno de sus objetos.
La figura 5.10 muestra una clase. La clase especifica la estructura de datos de cada uno de sus objetos y las operaciones que se utilizan para tener acceso a los datos.

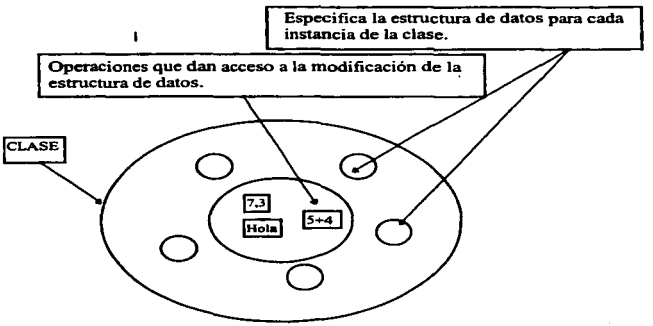


figura 5.10

Un Objeto es una Instancia de la Clase.Los datos y operaciones que encapsula quedan especificados por su clase. En el objeto se almacenan datos y se tiene acceso a ellos y se les modifica mediante sus operaciones. La reestricción al acceso de datos es lo que denominamos encapsulado.Estas operaciones forman la interfaz del objeto con sus usuarios.

Existe una diferencia entre las operaciones y los métodos. Las operaciones son procesos que se pueden solicitar como unidades. Los métodos son especificaciones del procedimiento de una operación dentro de una clase. Es decir, la operación es un tipo de servicio solicitado, y el método es su código de programación.

Ejemplo

Una operación asociada, con la clase ventas podria ser aquella que calcule el total de las ventas.

El método especificaria la forma de calcular el todo de las ventas. Para esto, el método podria obtener el precio de cada articulo del pedido al enviar una solicitud a los objetos venta por Departamento. A su vez cada objeto regresaria el total de las ventas por departamento mediante un método de la clase venta por Departamento.

La herencia establece que las propiedades de un tipo se aplican a sus subtipos. La herencia de clase hace que la estructura de datos y operaciones sean disponibles para su reutilización por parte de sus subclases. La herencia de una superclase permite que las clases compartan el código.

Ejemplo : "Ventas por departamento" hereda las operaciones "A" y "B" de "Ventas". Además tiene sus propias operaciones. "Ventas por Artículo" también hereda las operaciones "A" , "B" "C" y "D" , y también tiene sus propias operaciones "E" y "F".

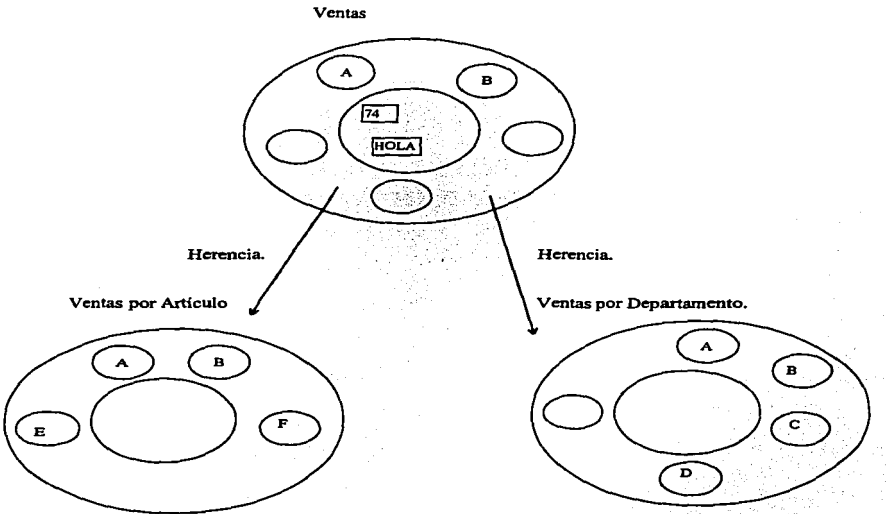


figura 5.11

Cuando se envía una solicitud a un objeto el software selecciona los métodos por utilizar.

El método no se almacena en el objeto, puesto que causaría una réplica múltiple. en vez de esto, el método se asocia con la clase. El método no puede estar en la clase de la que el objeto es una instancia, sino que en una superclase.

La herencia permite que una clase reutilice las características de sus superclases. De esta forma, los usuarios sólo deben especificar qué se debe hacer, dejando que sea el mecanismo de selección el que determine la forma de localizar la operación y la ejecute. El mecanismo de selección se deja en manos de la aplicación orientada a objetos, la que localiza la operación correcta a partir de la fuente de la solicitud.

Como uno de los objetivos principales de las técnicas orientadas a objetos es utilizar otra vez el código , algunas de las operaciones requieren adaptación para resolver necesidades particulares, por ejemplo : En una clase empleado se define una operación de retiro y en las implantaciones orientadas a objetos todas las subclases empleado heredan esta operación en forma automática, sin embargo una organización, puede tener distintos métodos para reactivar a un ejecutivo y a un empleado. En este caso, el método para el retiro de un ejecutivo esta por encima del método para el retiro de los empleados en general. Aunque sean distintos, los métodos llevan a cabo el mismo propósito operativo y es a lo que llamamos polimorfismo; concepto que analizamos en capítulos anteriores y es aquí en donde lo aplicamos.

Una de las ventajas del polimorfismo es que se puede hacer una solicitud de una operación sin conocer el método que debe ser llamado. Estos detalles quedan ocultos para el usuario, la responsabilidad la tiene el mecanismo de selección de la implantación orientada a objetos.

Las técnicas orientadas a objetos deben permitir la adaptación de las clases. Se debe poder tomar una clase de un repositorio y adaptarla a las necesidades del sistema.

CONCLUSIONES

El primer sistema que se construye con la programación orientada a objetos tarda más tiempo en construirse que con las técnicas estructuradas. Es mas tardado porque se tienen que asimilar y entender todos los conceptos nuevos que se describieron en el primer capítulo, para aplicarlos en nuevas herramientas de programación, y si estabamos acostumbrados a la programación estructurada cuesta aun más trabajo todavía. Es muy grande el tema de análisis y diseño orientado a objetos , y antes de empezar a construir un sistema con cualquier herramienta que soporte la programación orientada a objetos es necesario que se domine el análisis y diseño orientado a objetos, que es muy diferente al estructurado, además las herramientas que soportan programación con objetos tienen muchas herramientas que facilitan la programación, pero como no estamos acostumbrados a ellas nos cuesta trabajo en el primer sistema, utilizar el nuevo ambiente que nos ofrecen estas nuevas herramientas de programación.

La herramienta que evalué es Visual Objects y por la experiencia que obtuve con la realización de este trabajo me di cuenta que, en comparación con otras herramientas como los es Delphi y Visual Basic, es mas difícil para un programador que apenas empieza a trabajar con la programación orientada a objetos hacer un programa en el ambiente de Visual Objects porque en las herramientas que mencione todo se programa dentro de formas en las que se pegan componentes y se relacionan estos componentes entre sí, y es fácil que un programador que apenas empieza programe con esta facilidad que le ofrecen estas herramientas, pero se corre el riesgo de que los programadores no ocupen las técnicas de análisis y diseño adecuadas para sistemas que estén construidos con programación orientada a objetos y ocupen la herrami9enta de programación con técnicas estructuradas de para programar los sistemas y no exploten la herramienta al máximo.

Una desventaja que encontré de Visual Objects contra Delphi (que es con el que tengo más experiencia) es que tenemos que estar navegando por diferentes editores, y para los programadores que apenas empezamos es fácil que nos perdamos dentro de estos.

El pequeño sistema que se construyó, se tardó más en el diseño que en la programación, como se esperaba, y para explotar todas las ventajas que ofrece la orientación a objetos se deben de construir más sistemas para obtener la experiencia necesaria y diseñar cada vez mejor los sistemas para que la programación sea mas sencilla cada vez. Al crear un proyecto nuevo dentro de Visual Objects, el programador ya tiene funciones disponibles parque no empiece a programar desde cero y pueda reutilizar otras funciones de sistemas que creó antes, y ahorrarse el escribir mucho código. Visual Objects cuenta con funciones de altas, bajas y consultas que están disponibles, solo se necesita adherirlas a nuestro proyecto. Al principio es difícil utilizar todas estas facilidades porque son

nuevas para muchos de los programadores que no estábamos acostumbrados a trabajar con ventanas, eventos, propiedades etc.

La mayoría de estas herramientas trabajan de la misma manera y si entendemos una y ocupamos el análisis y diseño del sistema con las técnicas adecuadas, el aprender cualquier otra herramienta de programación orientada a objetos es más sencillo, es por esto que este trabajo es útil para las personas que empiecen a construir sistemas con esta nueva filosofía que es la orientación a objetos y todavía no hallan decidido con cual herramienta empezarán a programar.

No es recomendable que una empresa pequeña cambie sus sistemas con esta nueva tecnología, porque la inversión es muy fuerte, los equipos que se recomiendan para ejecutar estos nuevos sistemas, son máquinas con procesadores 486 de memoria y 12 mb en RAM, además Visual Objects es de un costo elevado y no creo que puedan absorber este gasto por el momento, pero si es necesario que todos los programadores empiecen a estudiar las técnicas de análisis y diseño orientados a objetos para que cuando baje el costo del Hardware y las empresas tengan la posibilidad de adquirir equipo nuevo para poder implementar esta nueva tecnología.

La crisis económica que estamos pasando frenó el avance que estaban teniendo las pequeñas empresas en la adquisición de Hardware y Software, en la actualidad las pequeñas empresas no pueden adquirir equipo nuevo y no tiene las posibilidades económicas para invertir en tecnología que tiene un mayor costo, es por este motivo que están trabajando con sistemas que ya tenían y que les están funcionando, estas empresas tiene que esperar a que el Hardware y el software baje de precio y que se recuperen económicamente para que puedan invertir en tecnología como esta que demanda más recursos.

DESCRIPCION	COSTO
1 Computadora 486 a 100 Mh 12 Mb Ram Windows 3.1	$10,000.00
CA Visual Objects	$ 3,200

Si una empresa hace esta inversión, justificara el costo cuando empiece a lograr los beneficios que se alcanzan al producir sistemas con esta nueva tecnología y algunos de los beneficios que capté al construir el pequeño sistema de ejemplo fueron: Cuando se dominan las técnicas de análisis y diseño orientadas a objetos, los sistemas se construyen en menos tiempo, el mantenimiento es mas sencillo, los sistemas son más fáciles de modificar, los diagramas que se utilizan en el diseño orientado a objetos son mas fáciles de entender, los programadores son más fáciles de sustituir porque el código no es complicado de entender, los

sistemas tienen una mejor presentación bajo ambiente Windows y son más amigables para el usuario final.

En algunas empresas, principalmente en las de gobierno, se adquieren las herramientas de programación orientadas a objetos y no se da la capacitación necesaria para que se ocupen estas herramientas adecuadamente y se desaprovechan las ventajas que ofrece la programación orientada a objetos y los desarrolladores mezclan la técnicas estructuradas con herramientas orientadas a objetos.

A las empresas que inviertan en esta tecnología les convendrá esta inversión, porque en la actualidad las empresas que crecen más rápido, son las que procesen la mayor cantidad de información bien organizada, y para lograr esto tienen que tener lo último en tecnología para que sus sistemas sean mejores y liberados en poco tiempo. Como mencionamos anteriormente, los primeros sistema que se construyen se tardan más en ser implementados porque es necesario antes de construirlos se tiene que capacitar al personal y se lleva tiempo esta capacitación.

Un problema con el que me enfrenté al hacer los programas del pequeño sistema que se construyó, y que es importante mencionar, es que bajo WINDOWS95 la versión 1.0a de Visual Objects, que es con la que trabajé, tiene problemas con algunos archivos DLL al ejecutar las aplicaciones que genera Visual Objects. Cuando se liberó esta versión en 1995, WINDOWS95 no salía al mercado y probablemente la nueva versión de Visual Objects no tenga estos problemas.

APENDICES

APENDICE A
REPORTES DEL SISTEMA

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

CDTECA

RELACION DE DISCOS

Cdid	Titulo	Cdartista	Sello	Cdtipo
1	MI TIERRA	5	SONY	SA
2	LOCAL HERO	4	PHONOGRAM	RR
3	BENEFIT	1	CHRYSALIS	RR
4	FOREVER YOUNG	2	CLASSIC JAZZ	JZ
5	TUBULAR BELL	3	VIGING RECORDS	RS
12	BUT SERIUSLY	0	ATLANTIC	RR

CDTeca

Catalogación de Compacts Disk

Relación de Tipos de CDs

Código	Descripción
BL	BLUES
CE	CANCION ESPAÑOLA
FK	FOLK
FL	FLAMENCO
JZ	JAZZ
MC	MUSICA CLASICA
ML	MUSICA LIGERA
RR	ROCK & ROLL
RS	ROCK SINFONICO
SA	SALSA

CDTeca

Catalogación de Compacts Disk

Relación de intérpretes

Código 1
Nombre JETHRO TULL
País INGLESA
Notas

Código 2
Nombre ELLA FITZGERALD
País AMERICANA
Notas

Código 3
Nombre MIKE OLDFIELD
País INGLESA
Notas

Código 4
Nombre MARK KNOPFLER
País INGLESA
Notas

Código 5
Nombre GLORIA ESTEFAN
País AMERICANA
Notas

- Titulo MI TIERRA

  Relación de temas: **Duración**

  CON LOS AÑOS QUE ME QUEDAN
  MI TIERRA
  AYER
  MI BUEN AMOR
  NO HAY MAL QUE POR BIEN NO VENGA
  ¡SI SEÑOR! ...
  VOLVERAS
  MONTUNO
  HABLEMOS EL MISMO IDIOMA
  HABLABAS DE MI
  TRADICION

- Titulo LOCAL HERO

  Relación de temas: **Duración**

  THE ROCKS AND THE WATER
  WILD THEME
  FREEWAY FLYER
  BOOMTOWN (VARIATION LOUIS' FAVOURITE
  THE WAY IT ALWAYS STARTS
  THE ROCKS AND THE THUNDER
  THE CEILIDH AND THE NORTHERN LIGHT
  THE MIST COVERED MOUNTAINS
  THE CEILIDH: LOUIS' FAVOURITE / BILL'S TUNE
  WHISTLE THEME
  SMOOCHING
  STARGAZER
  THE ROCKS AND THE THUNDER
  GOING ON: THEME OF LOCAL HERO

- Titulo BENEFIT

  Relación de temas: **Duración**

  WHIT YOU THERE TO HELP ME
  NOTHING TO SAY
  ALIVE AND WELL AND LIVING IN
  SON
  FOR MICHAEL COLLLINS, JEFFREY AND ME
  TO CRY YOU A SONG
  A TIME FOR EVERYTHING?
  INSIDE
  PAY IN TIME
  SOSSITY: YOU ARE A WOMAM

- Titulo FOREVER YOUNG

Relación de temas: **Duración**

SHINE
DEDICATED TO YOU
I WANT TO BE HAPPY
IF DREAMS COME TRUE
HALLELUYAH!
BEI MIR BIST DU SCHOEN
A TISKET-A-TISKET
SAVING MYSELF FOR YOU
ELLA
UNDECIDED
T AINT WHAT YOU DO (IT'S THE WAY THAT YOU DO IT)
MY HEART BELONGS TO DADDY
IT'S WONDERFUL
GOOD NIGHT MY LOVE
ORGAN GRINDER'S SWING
MY LAST AFFAIR

- Titulo TUBULAR BELL

Relación de temas: **Duración**

PART ONE
PART TWO

- Titulo BUT SERIUSLY

Relación de temas: **Duración**

FATHER TO SON

APENDICE B
DICCIONARIO DE DATOS

TABLA: CD

ESTRUCTURA DE DATOS:

CAMPO	TIPO	LON.	DESCRIPCIÓN
CDID	Numérico	4	Clave del disco(Llave primaria)
TITULO	Carácter	60	Titulo del disco compacto
CDARTISTA	Numérico	4	Clave del artista
SELLO	Carácter	25	Compañía que produce el disco
CDTIPO	Carácter	2	Clave del tipo de disco

EJEMPLO: 1*MI TIERRA*1*MELODY*SA.

TABLA: TIPO

ESTRUCTURA DE DATOS:

CAMPO	TIPO	LON.	DESCRIPCIÓN
CDTIPO	Carácter	2	Clave del tipo de música (Llave primaria)
TIPO	Carácter	25	Descripción del tipo de música

EJEMPLO: SA*BALADAS*

TABLA: ARTISTA

ESTRUCTURA DE DATOS:

CAMPO	TIPO	LON.	DESCRIPCIÓN
CDARTISTA	Numérico	4	Clave del artista(Llave primaria)
NOMBRE	Carácter	40	Nombre del artista
PAIS	Carácter	25	Pais de origen del artista.
NOTAS	MEMO	10	Comentarios

EJEMPLO:
1*GLORIA ESTEFAN*CUBA*ARTISTA CUBANA QUE RADICA EN E.U.*

TABLA: TEMAS

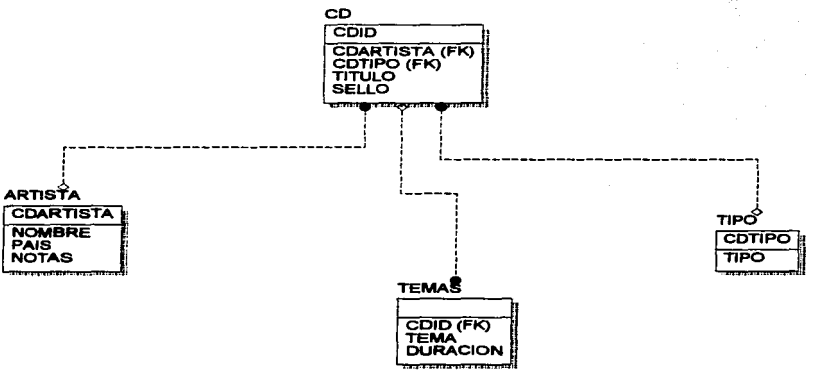
ESTRUCTURA DE DATOS:

CAMPO	TIPO	LON.	DESCRIPCIÓN
TEMA	Carácter	60	Nombre de la canción
CDID	Numérico	4	Clave del disco
DURACION	Numérico	3	Duración de la canción

EJEMPLO: CON LOS AÑOS QUE ME QUEDAN*1*3*

APENDICE C

DIAGRAMA DEL SISTEMA CDTECA



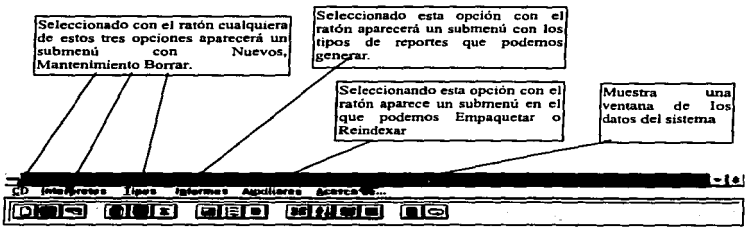
APENDICE D

MANUAL DE USUARIO SISTEMA CDTeca

Este sistema ayuda a mantener la información de Discos actualizada en una CDTeca para agilizar la búsqueda y las consultas en los ficheros.

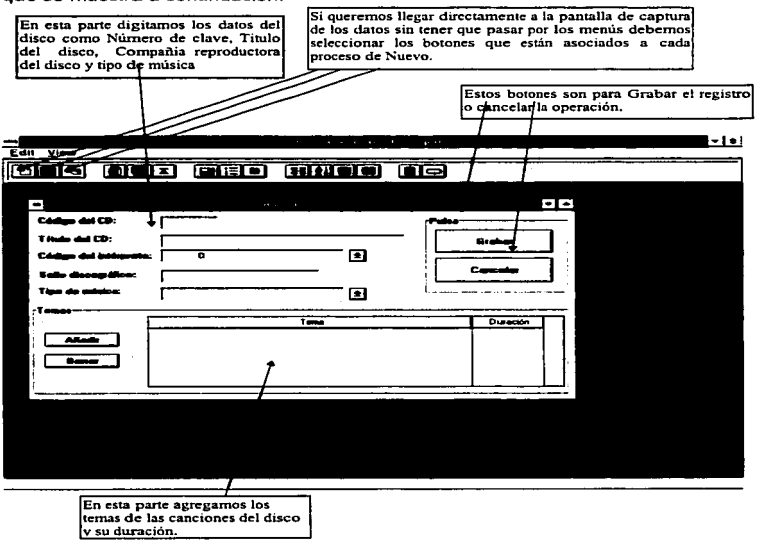
El sistema tiene asociados catálogos que pueden ser modificados mediante los procesos que se activan seleccionando los menús CD, INTERPRETES ,TIPOS. Estos menús tienen un submenú en el que podemos dar alta, baja, consulta o mantenimiento a los registros de los catálogos.

PANTALLA PRINCIPAL DEL SISTEMA



ALTA DE UN DISCO, ARTISTA O TIPO DE DISCO

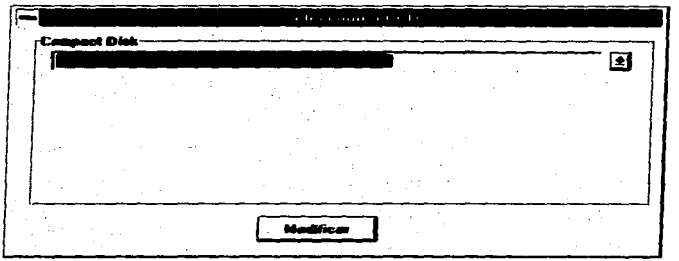
Para dar de alta un registro en nuestro sistema solamente tenemos que seleccionar el catálogo que deseamos actualizar en el menú principal y después seleccionar **Nuevo**, después el sistema mostrará una ventana de edición como la que se muestra a continuación.



Para los demás catálogos aparecen pantallas de captura semejantes en los que podemos insertar nuevos registros al sistema.

MANTENIMIENTO A LOS FICHEROS

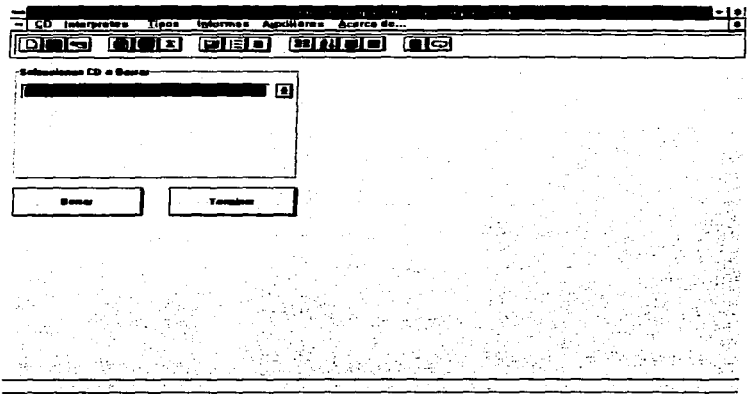
Dentro del mantenimiento podemos consultar los ficheros y modificar la información que tienen los catálogos. Para realizar esta operación tenemos que seleccionar en el menú principal el catálogo que queremos modificar o consultar y aparecerá una pantalla como la que se muestra a continuación.



En esta pantalla tenemos que elegir el Disco,Interprete o Tipo de disco según sea el catálogo que queremos modificar o consultar. Después de seleccionar el botón de Modificar aparece la misma pantalla de cuando damos de alta un nuevo registro, pero con los datos del disco que seleccionamos.

BORRAR UN REGISTRO DE LOS CATALOGOS

Para borrar un Disco, Artista o Tipo de disco, tenemos que seleccionar el comando borrar y en seguida aparecerá una pantalla en la que seleccionamos el registro que queremos borrar . Para borrar tenemos que presionar el botón de borrar, también podemos cancelar la operación con el botón terminar.



REPORTES DEL SISTEMA

Para generar un reporte e imprimirlo solo tenemos que seleccionar con el ratón en el menú principal Informes y después seleccionar el tipo de reporte que deseamos imprimir. Los reportes que genera el sistema son:

- CD
- Relación de CD
- Relación de Intérpretes
- Relación de Tipos de CD

CD :
Este reporte imprime el título de CD y los Temas que incluye.

Relación de CD:

Imprime la clave del disco, titulo, artista que lo interpreta, compañía que lo produce y el tipo de CD.

Relación de Intérpretes:

Imprime la clave del artista, nombre y el país del artista.

Relación de Tipos:

Imprime la clave del tipo de música y su descripción

Estos reportes se pueden observar en el **ANEXO A**

REINDEXAR LAS TABLAS

Para dar mantenimiento al sistema es necesario actualizar las tablas cuando se les hacen modificaciones a los registros. Por esto es necesario que el sistema regenere sus índices y la información sea la correcta al visualizarla; hay que recordar que estamos trabajando con archivos DBF, y con este tipo de archivos la reindexación es necesaria. El sistema tiene un comando de reindexación automático en el menú principal y seleccionando en el menú **Auxiliares** y despues el comando de Reindexar se ejecuta el comando.

BIBLIOGRAFIA

Análisis y Diseño Orientado a Objetos
James Martin
Ed. Kathyn Gollin
1995

C A Visual Objects Getting Started
Computer Associates
Agosto 1994

C A Visual Objects Programers Guide Volumen I
Computer Associates
Agosto 1994

C A Visual Objects Programers Guide Volumen II
Computer Associates
Agosto 1994

C A Visual Objects Programers Guide Volumen III
Computer Associates
Agosto 1994

Guide to Multimedia
Green Barbara
1956
Ed. New Riders

Microsoft Education and Certificatios
Microsoft Windows Operating Systems and Services Architecture
Microsoft Corporation
1995

Multimedia Madness
Jerram Peter
Ed. Random House
1993

NET El medio de las comunicaciones Volumen I
¿Ya sabes que es Multimedia ?
Fourier G. Ma de Lourdes
1996

Programación Orientada a Objetos
Brad J. Cox
Ed. Addison

Seminario de Introducción a CA Visual Objects
Revueltas Ana Isabel
Cibernética y Tecnología
1994

Software Orientado a Objetos
Ann L. Winblad
Ed. Adisson

Visual Objects
Grupo Eidos
1995

Windows 3.1 Multimedia
Jennings
1994