UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

IMPLEMENTACION DE UN SISTEMA DE PRESTADORES DE BIENES Y SERVICIOS EN IXTAPA ZIHUATANEJO

T E S I S

QUE PARA OBTENER EL TITULO DE

INGENIERO EN COMPUTACION

P R E S E N T A N :

ANABELLE FERNANDEZ ALCANTARA
FRANCISCO GAMEZ VARGAS
MARTHA PATRICIA PELAEZ FLORES

DIRECTOR DE TESIS: M. EN I. JUAN CARLOS ROA BEIZA

MEXICO, D. F. 1995

FALLA DE ORIGEN

INDICE

CAPÍTULO II

PLANTEAMIENTO DE LA PROBLEMATICA Y PROPUESTA DE SOLUCIÓN

CAPÍTULO III

DESARROLLO DEL SISTEMA

OBJETIVO:

- El sistema deberá contemplar la solución de los siguientes requerimientos:

- Se utilizarán metodologías de programación Orientada a Objetos, tomando como base de desarrollo el lenguaje de programación Visual C + +.

- Uso de un menú gráfico para el acceso a otros menús, utilizando para ello dos dispositivos de acceso, el ratón y el teclado.

- Información sobre la Historia general del lugar.

- Manejo de los mapas de las ciudades para la pronta localización de Bienes, Servicios y Atracciones Turísticas.

- Localización e Información especifica sobre un punto geográfico de interés deseado, utilizando tres radios de acción de diferente longitud.

- Información sobre Bienes, Servicios y Atracciones Turísticas, de las ciudades mediante un ambiente gráfico.

- Opción de un formato de Quejas donde el interesado pueda indicar algún Bien, Servicio o Atracción Turística que no haya encontrado, o alguna característica que le gustaría que el sistema tuviera.

- Opción de un menú de ayuda en donde se explique el manejo del sistema.

- Obtener un sistema amigable y confiable.

Introducción

Dado que el uso de la computadora se ha extendido hacia la mayoría de las actividades que el ser humano realiza, eliminando con ella en lo posible, el trabajo manual, se penso entonces, en crear un programa que fuera útil en el sector turístico, ya que en este campo se necesita un sistema que le permita al visitante de algún lugar, localizar de manera agil y eficiente, sitios de interés en un mapa, y que además le proporcione información turística sobre el lugar de una manera fácil.

Con el propósito de cubrir parte de la idea anterior, se implemento el siguiente programa **"Sistema para la Prestación de bienes y servicios en el puerto de Ixtapa Zihuatanejo"**, dicho sistema sólo proporcionará información turística de las ciudades de Ixtapa y Zihuatanejo.

El sistema anterior fue desarrollado para ser ejecutado en un ambiente que facilita su uso; y este es el ambiente Windows, tomando en cuenta la importancia que ha cobrado la Programación Orientada a Objetos; es decir; que mediante objetos graficos se nos indique las funciones que realiza el programa.

En la elaboración del sistema se utilizó el lenguaje de programación Visual C++, ya que este cuenta con unas poderosas herramientas que nos facilitan la programación, otorgandonos el beneficio de crear un código de programación formal que la verdadera Ingeniería de software requiere.

CAPITULO I

CONCEPTOS TEORICOS

1.1 Teoria de Programación Orientada a Objetos

1.1.1 CONCEPTOS BASICOS

La programación orientada a objetos fue desarrollada, esencialmente, por las limitaciones que otras técnicas de programación tenían y tienen, como es el caso de la programación estructurada.

La idea fundamental de la programación estructurada es romper un programa en unidades más pequeñas denominadas funciones en C (procedimientos, subprogramas y suburbanas en otros lenguajes). Cada una de las funciones (al menos idealmente) tienen un propósito claramente definido así como una interfaz a las otras funciones del programa.

La programación estructurada utiliza el método descendente y de refinamiento sucesivo, y cada una de las funciones se invocan sucesivamente. Los programas estructurados son más fáciles de escribir y mantener que los programas no estructurados. Sin embargo, a medida que el tamaño de los programas aumenta y se hacen muy grandes y más complejos, el énfasis se pone en los tipos de datos que se procesan.

Las estructuras de datos en un programa se hacen tan importantes como las operaciones realizadas por ellos. Los tipos de datos se procesan en muchas funciones, dentro de un programa estructurado y cuando se producen cambios en esos tipos de datos, las modificaciones se deben hacer en cada posición que actúa sobre esos tipos de datos dentro del programa.

Esta tarea además de consumir gran tiempo puede ser frustante en programas que contengan millares de lineas de código y centenares de funciones. Esta circunstancia se produce esencialmente porque muchas funciones comparten datos (variables globales).

Así, por ejemplo, si se añaden nuevos datos se necesitarán modificar todas las funciones que acceden a los datos, de modo que ellas también puedan acceder.

El último inconveniente que consideramos se produce cuando un equipo de programadores trabaja en una aplicación. Ya que en un programa estructurado, a cada programador, se le asigna construir una tarea especifica de funciones y manejar los tipos de datos de esta. Dado que diferentes programadores manipulan funciones independientes que comparten datos, los cambios que un programador hace a los datos deben ser reflejados en el trabajo del resto del equipo. Los problemas y los errores de comunicación se reflejarán en el diseño final.

PROGRAMACION ORIENTADA A OBJETOS

La programación orientada a objetos enfatiza en los datos, al contrario de la programación estructurada que enfatiza en los algoritmos. La programación orientada a objetos permite organizar los datos de su programa al igual que los objetos que forman el mundo real; por ejemplo: los departamentos de una gran compañía (ventas, contabilidad, personal, técnico, etc.), o los diferentes vehículos que se construyen en una fábrica de automóviles, etc.; todos ellos son objetos.

En la programación orientada a objetos coches (carros), árboles, departamentos, publicaciones, se conocen en general como objetos. Un objeto contiene en una sola entidad y con un único nombre las estructuras de datos y las acciones (procedimientos y las funciones) que actúan sobre este.

Así pues, una clase (objeto en general) es una plantilla o modelo que define los datos y las funciones que actúan sobre esos datos (llamados métodos). Por ejemplo, una clase puede describir las características fundamentales de un ejecutivo de una empresa de computadoras (nombre, título, salario, cargo, departamento al que pertenece, etc.), mientras que un objeto representará un ejecutivo específico (Andrés Montes, Físico, 5000.00, jefe de departamento, etc.).

Cuando un elemento dato es miembro de una clase se llama objeto. Se pueden asemejar las clases a los tipos de datos definidos por el usuario en el lenguaje de programación tradicional, ya que los tipos definidos por las clases se comportan como los tipos incorporados a un lenguaje de programación.

IDENTIDAD DE OBJETOS

Los objetos en una base de datos orientada a objetos, normalmente corresponden a una entidad en la empresa que está modelando la base de datos. Una entidad conserva su identidad aun cuando algunas de sus propiedades cambien con el tiempo. Igualmente, un objeto conserva su identidad aun cuando algunos o todos los valores de las variables o las definiciones de los métodos cambien con el tiempo. Este concepto de identidad no se aplica a las tuplas de una base de datos relacional. En los sistemas relacionales, las tuplas de una relación se distinguen únicamente por los valores que contienen.

La identidad de objetos es una noción más fuerte que la que se encuentra normalmente en los lenguajes de programación a objetos. A continuación se presentan varias formas de identidad.

- Valor. Se utiliza un valor de dato por identidad. Esta es la forma de identidad que se usa en los sistemas relacionales.

- Nombre. Se utiliza un nombre facilitado por el usuario por identidad. Esta es la forma de identidad que normalmente se usa para variables en los procedimientos. A cada variable se la da un nombre que identifica de manera única a la variable sin importar el valor que contenga.
- Incorporación. Una noción de identidad es incorporar en el modelo de datos el lenguaje de programación, y no se requiere que el usuario proporcione ningún identificador. Esta es la forma de identidad que se usa en los sistemas orientados a objetos.

Una clase encapsula (encierra) en sí misma datos y métodos (funciones) que manipulan los datos de los objetos de la clase. Si una clase es como un tipo de dato, un objeto es como una variable. Con frecuencia, a los objetos se les llama instancias, casos, modelos o ejemplos de una clase, y también variables de instancia.

Las funciones llamadas métodos se suelen denominar funciones miembro en C++. Las funciones miembro definidas en una clase se pueden invocar por objetos de esa clase; esta acción se conoce como envío de un mensaje al objeto.

En general un objeto tiene asociado:

- Un conjunto de variables que contiene los datos del objeto. El valor de cada variable es un objeto.
- Un conjunto de mensajes a los que el objeto responde.
- Un método que es un trozo de código para implementar cada mensaje. Un método devuelve un valor como respuesta al mensaje.

El término mensaje en un contexto orientado a objeto no implica el uso de un mensaje físico en una red de computadoras, sino que se refiere al paso de solicitudes entre objetos sin tener en cuenta detalles específicos de implementación.

Puesto que el único interfaz externo que presenta un objeto es el conjunto de mensajes al que responde, es posible modificar la definición de métodos y variables sin afectar a otros objetos. También es posible sustituir una variable por un método que calcule un valor. Por ejemplo, un objeto de documento puede contener una variable de tamaño que contenga el número de bytes de texto en el documento o bien un método de tamaño que calcule el tamaño del documento leyendo el documento y contando el número de bytes.

La capacidad de modificar la definición de un objeto sin afectar al resto del sistema está considerada como una de las mayores ventajas del modelo de programación orientado a objetos.

El proceso de programación en un lenguaje orientado a objetos se caracteriza por:

- Crear clases que definen la presentación y el comportamiento de los objetos.
- Crear objetos a partir de la definición de la clase.
- Realizar la comunicación entre objetos a través del envío de mensajes (invocación a las funciones miembro).

Así por ejemplo, en C++ el tipo de clase (class) es una estructura (struct) más funciones, que manipulan los campos de datos. Su sintaxis es similar a una estructura, sustituyendo struct por class y añadiendo una posibilidad de controlar el grado de acceso a los datos y métodos de la clase manifestados en declaraciones public (pública), private (privada) y protect (protegida).

PROPIEDADES FUNDAMENTALES DE LA PROGRAMACION ORIENTADA A OBJETOS.

Las propiedades fundamentales de la programación orientada a objetos son:

- **Abstracción de datos**
- **Encapsulación (encapsulamiento y ocultación de datos).**
- **Herencia**
- **Polimorfismo**
- **Reutilización**

ABSTRACCION DE DATOS

La abstracción fue un concepto introducido en programación estructurada. Se define como la capacidad para examinar algo sin preocuparse de sus detalles internos. En un programa estructurado, es suficiente conocer la tarea específica que hace un procedimiento dado, y no es tan importante cómo se realiza esa tarea. Esta propiedad se conoce como abstracción funcional.

La abstracción de datos es a los datos lo que la abstracción funcional es a las operaciones. Con la abstracción de datos, las estructuras y los ítems de datos se pueden utilizar sin tener que preocuparse sobre los detalles exactos de su realización (implementación).

La abstracción de datos libera al programador de preocuparse sobre detalles no esenciales. La abstracción de datos a existido formalmente en todos los lenguajes de programación para elementos complicados. Sin embargo, han sido los lenguajes modernos como Modula-2, Ada y los lenguajes orientados a objetos (puros Smalltalk o híbridos C++ y Turbo Pascal 5.5/6.0/7.0) quienes han permitido definir sus propios tipos abstractos de datos.

Tipos abstractos de datos (TAD) es un tipo de dato definido por el programador que se puede manipular de un modo similar a los tipos de datos definidos o incorporados al sistema. Al igual que con los tipos definidos en el lenguaje, un TAD pertenece a un conjunto de valores legales de datos (rango) y se pueden realizar sobre esos valores un número de operaciones primitivas. Los usuarios pueden crear variables con valores cuyo rango esta en el conjunto de valores legales y pueden operar sobre esos valores las operaciones definidas.

Un objeto es simplemente un tipo abstracto de datos y, por ello, las nociones de programación orientada a objetos se construyen sobre la idea de dicho tipo de datos, añadiéndoles otras propiedades, como pueden ser la ocultación y la reutilización de código.

ENCAPSULAMIENTO Y OCULTACION DE LA INFORMACION

La abstracción y el encapsulamiento son conceptos complementarios: la abstracción se enfoca a la vista exterior de un objeto y el encapsulamiento (también conocido como ocultación de la información) previene a otros objetos ver su interior, donde el comportamiento de la abstracción se ha realizado (implementado).

Un concepto fundamental en las clases es la ocultación de los detalles de su implementación. En la práctica esto significa que cada clase debe tener dos partes: un interfaz y una implementación.

El interfaz de una clase de dedica solo a la vista exterior, y la implementación comprende la representación de la abstracción, así como los mecanismos que realizan los comportamientos deseados.

De este modo, las clases suelen tener dos partes: parte pública (accesible fuera de la clase) y parte privada (solo accesible dentro de la clase). Los datos y métodos se pueden definir, indistintamente, en cada parte. C++ define una tercera sesión en una clase: la parte protegida. Es aquella a la que pueden acceder no solo las funciones miembro (métodos)

dentro de la misma clase, sino también funciones miembro de clases derivadas o descendientes de esa clase.

HERENCIA

La idea de clases conduce a la idea de herencia. En nuestra vida diaria el concepto de clases se divide en subclases. Así, las clases animales se dividen en: mamíferos, anfibios, insectos, etc. El principio en que se basan las sucesivas subclases es que cada subclase comparte características comunes con la clase con la que se deriva o desciende, por ejemplo, todas las clases de vehículos tienen un motor, ruedas y un volante o manillar. Además de las características comunes, cada subclase tiene sus propias características particulares, por ejemplo, número de ruedas, número de asientos, diferente portaequipajes, etc.

En la programación orientada objetos, las clases se pueden subdividir en subclases dicho en otro modo, una clase se puede deducir o derivar de otra clase.

La clase original se denomina clase base y, merced a la propiedad de herencia, una clase derivada o descendiente puede heredar las estructuras de datos y los métodos de su clase base.

Además, la nueva clase puede añadir nuevos elementos datos y métodos a los que hereda de su clase base. Cualquier clase base (incluida una descendiente) puede tener cualquier número de clases descendientes.

Mediante el mecanismo de la herencia se puede construir una jerarquía de clases que adoptará la forma típica de árbol, donde la clase base se llamará clase padre, y la clase descendiente o derivada, clase hija.

Existen dos tipos de herencia: simple y múltiple. La herencia simple, es aquella en la que una clase solo puede tener un ascendiente. Herencia múltiple se refiere a la capacidad de las clases para heredar variables y métodos de múltiples superclases, o es aquella en que un objeto puede tener más de un descendiente. La mayoría de los lenguajes soportan herencia simple. Solo algunos incorporan la propiedad de herencia múltiple: C++ (a partir de 2.0) y Eiffel. Un caso típico de herencia múltiple es el popular término multimedia. Supongamos dos tipos de clases, sonido y gráfico; si se combinan ambos pueden dar lugar a una nueva clase multimedia con propiedades de ambas clases.

Cuando se emplea la herencia múltiple es posible que se de ambigüedad en el caso en que pueda heredarse la misma variable o método de más de una superclase. No todos los casos de herencia múltiple conducen a ambigüedad.

POLIFORMISMO

Poliformismo es la propiedad que permite a una operación tener diferente comportamiento en objetos diferentes. En otras palabras, objetos diferentes reaccionan de modo diferente al mismo mensaje. El poliformismo juega un papel importante en la simplificación de la sintaxis al realizar la misma operación sobre una colección de objetos.

El poliformismo requiere herencia; simplifica la tarea del programador generalizando la sintaxis de comunicación que permite tratar objetos de un tipo diferente de modo similar. El poliformismo se aplica únicamente a los métodos heredados de una clase base.

Esta propiedad significa que un método puede tener un nombre que es compartido a lo largo de una jerarquía de clases. Cada clase puede tener una implementación diferente para ese método.

El poliformismo depende de la ligadura, que es el proceso por el cual un método se asocia con una función real. Cuando los métodos polifórmicos se utilizan, el compilador no puede determinar a qué función llamar, la función específica llamada depende de la clase del

elemento a la cual se envía el mensaje; en consecuencia, la función a llamar se determina en tiempo de ejecución. Esta operación se conoce como ***ligadura tardía o postergada*** (late binding) ya que ocurre cuando el programa se está ejecutando. La ***ligadura temprana o previa*** ocurre para métodos no polifórmicos; el compilador conoce exactamente cuál es el método que se invoca, de modo que se puede construir una llamada directa en tiempo de compilación. Aunque la ligadura previa es muy eficiente la mayoría de los lenguajes orientados a objetos utilizan ligadura postergada para todos los métodos.

REUTILIZACION.

Una vez que una clase se ha escrito, creado, depurado y comprobado, se puede distribuir a otros programadores para utilizarla en sus propios programas. Es decir, se puede utilizar como bloque básico para construir un programa. Esta operación se llama ***reutilización***. Los programas orientados a objetos se construyen a partir de componentes reutilizables de software. Esta operación es similar al modo en que una biblioteca de funciones, en un lenguaje procedural se puede incorporar en diferentes programas.

El concepto de herencia en programación orientada a objetos proporciona una extensión importante a la idea de reutilización. Un programador puede tomar una clase existente, perteneciente a una biblioteca de clases y, sin modificarla, le puede añadir características y propiedades adicionales.

En resumen la programación orientada a objetos se basa en el concepto de encapsulamiento de datos y código que opera sobre esos datos en un objeto. Los objetos estructurados se agrupan en clases. El conjunto de clases está estructurado en sub y superclases basado en una extensión del concepto del modelo entidad-relación. Puesto que el valor de un dato en un objeto también es un objeto, es posible representar el contenido del objeto dando como resultado un objeto compuesto.

1.1.2 ARQUITECTURA DE UN SISTEMA CON PROGRAMACION ORIENTADA A OBJETOS.

El estado actual de la mayor parte de la ingeniería de software está algo atrasado con respecto a las demás áreas de la ingeniería. Cuando la mayoría de los ingenieros completan su trabajo y éste se vende, esperamos que funcione de manera correcta. No esperamos que los motores de los aviones exploten o que los edificios se derrumben, pero no nos sorprende que cierto software se comporte de manera extraña. La mayoría de los productos de hardware tienen garantía, pero la mayoría de los productos de software llevan una renuncia a la garantía.

El software que se vende no esta libre de errores, pero puede decirse que cuando estos aparecen lo hacen con una frecuencia bastante baja.

Cuando las personas empiezan a programar, en general sus maestros les dicen que piensen como una computadora. Esta técnica parecía servir cuando aprendíamos a programar, puesto que escribíamos programas muy sencillos. Sin embargo, los ciclos y las ramificaciones nos proveen de una gran cantidad de combinaciones de formas, de modo que no podemos pensar en todos ellos como una computadora. La operación de la mayoría del software real depende de condiciones que no se conocerán hasta ejecutar el software. En el caso de la multiprogramación, el equipo hace varias cosas a la vez, por lo que es imposible pensar como una computadora.

Para ayudarnos a lidiar con la complejidad, se comenzó a utilizar la programación estructurada. Hubo una reducción en el código, pero la programación seguía basándose en una secuencia esperada de instrucciones de ejecución. El esfuerzo por diseñar y depurar programas, pensando en el orden que la computadora sigue para hacer las cosas, desembocó en un software que nadie entendía del todo. El hecho de pensar como una computadora está más allá de nuestra capacidad mental.

El mundo de las técnicas orientadas a objetos es muy diferente. El diseñador piensa en términos de objetos y su comportamiento, y se genera el código. El generador de código no debe tener errores, de modo que no hemos eliminado la programación manual. Sin embargo la mayoría de los sistemas se pueden construir sin tener que pensar en ciclos, ramificaciones y estructuras para el control del programa. El constructor del sistema aprende otro tipo de forma de pensar. Los eventos producen cambios en el estado de los objetos. La mayoría de estos cambios de estado requiere de pequeñas partes de código, por lo que la codificación está menos propensa a errores. Se construyen tipos de objetos a partir de tipos de objetos más sencillos. Una vez que los tipos de objetos funcionan bien, el diseñador los considera como cajas negras, de modo que nadie pueda ver su interior. La ingeniería de software adquiere así más características de la ingeniería del hardware.

En la programación convencional, los datos asumen cualquier estructura y los procesos hacen de los datos todo lo que el programador desee. En el mundo orientado a objetos, las estructuras de datos se relacionan con los objetos y sólo pueden ser utilizadas mediante los métodos diseñados para ese tipo de objeto

Dan Ingalls de Smalltalk, famoso por ser el causante de la "Violación de procesos mediante su decomposición en bits y la devastación de las estructuras de datos", describe elocuentemente la orientación por medio de los procesos. Para él, la orientación a objetos proporciona una solución que conduce a *"Un universo de objetos bien educados que se piden, de manera cortés, cocederse mutuamente sus deseos"*.

El contraste entre la orientación por medio de procesos y la orientación a objetos se puede resumir de la manera siguiente. El procesamiento convencional de los datos se centra en los tipos de objetos cuya estructura de datos sólo pueda controlarse mediante los métodos de la clase del objeto. Ocurren eventos que modifican el estado de un objeto. Por lo general, la propia programación de cada cambio de estado es sencilla, por lo que dividimos la programación en partes relativamente sencillas. De hecho cada objeto lleva a cabo una función específica e independiente de los demás objetos. Responde a mensajes, sin saber la

razón del envío de estos, ni las consecuencias de su acción. Puesto que los objetos actúan en forma individual, cada clase se puede modificar, en gran medida, de manera independiente a las demás clases. Esto facilita la prueba y modificación de las clases. El mantenimiento de los sistemas orientados a objetos es mucho más sencillo que el mantenimiento de los sistemas convencionales.

El mundo orientado a objetos tiene una mayor disciplina que el de las técnicas de estructura convencional. Esto nos lleva a un mundo de clases reutilizables, donde la mayor parte del proceso de construcción de software consistirá en el ensamblaje de clases ya existentes y probadas.

Las técnicas orientadas a objetos constituyen el mejor camino conocido para construir una verdadera ingeniería de software.

Beneficios de la Tecnología Orientadas a Objetos

- **Reutilización.** Las clases están diseñadas para que se reutilicen en muchos sistemas. Para maximizar la reutilización, las clases se construyen de modo que se puedan adaptar. Un depósito debe estar poblado de una creciente colección de clases utilizables. Es probable que las bibliotecas de clases crezcan rápidamente. Un objetivo fundamental de las técnicas orientadas a objetos es lograr la reutilización masiva al construir software.
- **Estabilidad.** Las clases diseñadas para una reutilización repetida se vuelven estables, de la misma manera que los microprocesadores y otros chips de hardware cuando sea posible.
- **El diseñador piensa en términos del comportamiento de objetos y no en detalles de bajo nivel.** El encapsulado oculta los detalles y hace que las clases complejas sean fáciles de utilizar. Las clases son como cajas negras; el investigador utiliza la caja negra y no ve hacia el interior de ésta. Sólo debe entender el comportamiento de la caja negra y cómo comunicarse con ella.

razón del envío de estos, ni las consecuencias de su acción. Puesto que los objetos actúan en forma individual, cada clase se puede modificar, en gran medida, de manera independiente a las demás clases. Esto facilita la prueba y modificación de las clases. El mantenimiento de los sistemas orientados a objetos es mucho más sencillo que el mantenimiento de los sistemas convencionales.

El mundo orientado a objetos tiene una mayor disciplina que el de las técnicas de estructura convencional. Esto nos lleva a un mundo de clases reutilizables, donde la mayor parte del proceso de construcción de software consistirá en el ensamblaje de clases ya existentes y probadas.

Las técnicas orientadas a objetos constituyen el mejor camino conocido para construir una verdadera ingeniería de software.

Beneficios de la Tecnología Orientadas a Objetos

- **Reutilización.** Las clases están diseñadas para que se reutilicen en muchos sistemas. Para maximizar la reutilización, las clases se construyen de modo que se puedan adaptar. Un depósito debe estar poblado de una creciente colección de clases utilizables. Es probable que las bibliotecas de clases crezcan rápidamente. Un objetivo fundamental de las técnicas orientadas a objetos es lograr la reutilización masiva al construir software.
- **Estabilidad.** Las clases diseñadas para una reutilización repetida se vuelven estables, de la misma manera que los microprocesadores y otros chips de hardware cuando sea posible.
- **El diseñador piensa en términos del comportamiento de objetos y no en detalles de bajo nivel.** El encapsulado oculta los detalles y hace que las clases complejas sean fáciles de utilizar. Las clases son como cajas negras; el investigador utiliza la caja negra y no ve hacia el interior de ésta. Sólo debe entender el comportamiento de la caja negra y cómo comunicarse con ella.

- **Se construyen clases cada vez más complejas.** Se construyen clases a partir de otras clases, las cuales a su vez se integran mediante clases. Así como los bienes manufacturados se fabrican a partir de una serie de materiales y de partes y subpartes ya existentes, tambien el software se crea mediante una serie de materiales de clases ya existentes y probadas. Esto permite construir componentes complejos de software, que a su vez se convierten en bloques de construcción de software más complejo.
- **Confiabilidad.** Es probable que el software construido a partir de clases estables ya probadas tenga menos fallas que el software elaborado a partir de cero.
- **Nuevos mercados para el software.** Las compañías de software deben proporcionar bibliotecas de software para áreas específicas, que se adapten con facilidad a las necesidades de la organización que las utiliza. La era de los paquetes monolíticos viene siendo reemplazada por software que incorpora clases y paquetes encapsulados de muchos proveedores distintos.
- **Un diseño más rápido.** Las aplicaciones se crean a partir de componentes ya existentes. Muchos de los componentes están construidos de modo que ya se puedan adaptar para un diseño particular.
- **Diseño de mayor calidad**. Los diseños suelen tener mayor calidad, puesto que se integran a partir de componentes probados, que han sido verficados y pulidos varias veces.
- **Integridad.** Las estructuras de datos sólo se pueden utilizar con métodos específicos. Esto tiene particular importancia en los sistemas cliente-servidor y en los sistemas distribuidos, en los que usuarios desconocidos podrían intentar el acceso al sistema.
- **Programación más sencilla.** Los programas se conjuntan a partir de piezas pequeñas, cada una de las cuales, en general, se pueden crear fácilmente. El programador crea un método para una clase a la vez. El método cambia el estado de los objetos en formas que suelen ser sencillas cuando se les considera en si mismas.

- **Mantenimiento más sencillo.** El programador encargado del mantenimiento cambia un método de clase a la vez. Cada clase efectúa sus funciones independientemente de las demás.
- **Invención.** Los implantadores eficientes encuentran que se pueden generar ideas con rápidez con las herramientas orientadas a objetos más poderosas, ejecutandose en una estación de trabajo. Las herramientas los animan a inventar e implantar con rápidez sus ideas. Las personas brillantes pueden ser muchos más creativas.
- **Ciclo vital dinámico.** El objetivo del desarrollo de un sistema cambia con frecuencia durante la implantación. Las herramientas orientadas a objetos facilitan los cambios a la mitad del ciclo vital. Esto permite a los implantadores de software conocer mejor a los usuarios finales, adaptarse a los cambios en las empresas, afinar los objetivos conforme el sistema se consolida y mejora de manera constante el diseño durante la implantación.
- **Esmero durante la construcción.** Las personas creativas, como los escritores y los novelistas, modifican de manera constante su trabajo durante la implantación. Esto lleva a mucho mejores resultados finales.
- **Modelado más realista.** El análisis orientado a objetos modela la empresa o área de aplicación de manera que sea lo más cercana posible a la realidad de lo que se logra con el análisis convencional. El análisis se traduce de manera directa en el diseño y la implantación. En las técnicas convencionales, el paradigma cambia cuando pasamos del análisis al diseño y del diseño a la programación. Con las técnicas orientadas a objetos, el análisis, diseño e implantación utiliza el mismo paradigma y lo afinan de manera sucesiva.
- **Mejor comunicación entre los profesionales de los sistemas de información y los empresarios.** Los empresarios comprenden más fácilmente el paradigma orientado a objetos. Piensan en términos de eventos, objetos y políticas empresariales que describen el comportamiento de los objetos. Las metodologías orientadas a objetos apoyan una

mejor comprensión, ya que los usuarios finales y creadores de software comparten un modelo común.

- **Modelos empresariales inteligentes.** Los modelos empresariales deben describir las reglas con las que los ejecutivos desean controlar su empresa. Estas se deben expresar en términos de eventos y la forma en que éstos deben modificar el estado de los objetos de la empresa. A partir del modelo de empresa se deben deducir diseños de aplicación con la mayor automátización posible.
- **Especificaciones declarativas y diseño.** Las especificaciones y el diseño construidos mediante los formalismos de las herramientas CASE, deben ser declarativos, en la medida de lo posible. Esto permite al diseñador pensar como un usuario final en vez de pensar como una computadora..
- **Una interfaz de pantalla sugestiva para el usuario.** Hay que utilizar una interfaz usuario gráfica, como la de la Macintosh, de modo que el usuario apunte a iconos o elementos de un menú desplegado, relacionados con los objetos. En determinadas ocasiones, el usuario puede, de hecho, ver un objeto en la pantalla. Ver y apuntar es más fácil que recordar y escribir.
- **Imagenes, video y expresión.** Se almacenan objetos binarios de gran tamaño (BLOB) que representan imagenes, video, expresiones, texto sin formato o algunos otros flujos de bits de gran tamaño. Con el objeto se utilizan métodos como compresión o descompresión, cifrado o descifrado y técnicas de presentación..
- **Independencia del diseño.** Las clases estan diseñadas para ser independientes del ambiente de plataformas de hardware y software. Utilizan solicitudes y respuestas con formato estándar. Esto les permite ser utilizadas en múltiples sistemas operativos, controladores de bases de datos, controladores de redes, interfaces de usuario gráficas, etc. El creador de software no tiene que preocuparse por el ambiente o esperar a que éste se especifique.

- **Interacción** El software de varios proveedores puede funcionar como conjunto. Un proveedor utiliza clases de otros. Existe una forma estándar de localizar clases e interactuar con ellas. La interacción de software de varios orígenes es uno de los objetivos más importantes de los estandares orientados a objetos. El software desarrollado de manera independiente en lugares ajenos debe poder funcionar de forma conjunta y aparecer como una sóla unidad ante el usuario.
- **Computación cliente-servidor.** En los sistemas cliente-servidor, las clases en el software deben enviar solicitudes a las clases en el software servidor y recibir respuestas. Una clase servidor puede ser utilizada por clientes diferentes. Estos clientes sólo pueden tener acceso a los datos del servidor a través de los métodos de la clase. Por lo tanto, los datos están protegidos contra su corrupción.
- **Computación de distribución másiva.** Las redes a nivel mundial utilizarán directorios de software de objetos accesibles. El diseño orientado a objetos es la clave para la computación de distribución masiva. Las clases de una máquina interactuarán con las de algún otro lugar sin saber donde residen tales clases. Ellas envían y reciben mensajes orientados a objetos en formato estándar.
- **Computación paralela.** La rapidez de las máquinas mejorará en gran medida mediante la construcción de computadoras paralelas. El procesamiento concurrente se efectuará al mismo tiempo en varios chips de procesadores. Los objetos de procesadores distintos se ejecutarán de manera simultanea, actuando cada uno en forma independiente.
- **Mayor nivel de automatización de las bases de datos.** Las estructuras de datos en las bases de datos orientadas a objetos estan ligadas a métodos que llevan acabo acciones automáticas. Una base de datos orientada a objetos tiene integrada una inteligencia, en forma de métodos, en tanto que una base de datos de relación básica no.
- **Eficacia de la máquina.** Ha quedado demostrado que las bases de datos orientadas a objetos tienen un rendimiento mucho mejor que las bases de datos de relación para ciertas aplicaciones con estructuras de datos muy complejas. Esto aunado al computo

concurrente con el diseño orientado a objetos, promete un salto enorme en el desempeño de las máquinas. Los sistemas cliente-servidor basados en LAN utilizarán máquinas servidoras con concurrencia y bases de datos orientadas a objetos.

- **Migración.** Las aplicaciones ya existentes, sean orientadas a objetos o no, pueden preservarse si se ajustan a un contenedor orientados a objetos, de modo que la comunicación con ellas sea a través de mensajes estándar orientadas a objetos.
- **Bibliotecas de clases para las industrias.** Las compañías de software venden bibliotecas para diversas áreas de aplicación. Las bibliotecas de clases independientes de la aplicación también son importantes y se proporcionan mejor como una capacidad de las herramientas CASE.
- **Bibliotecas de clases para las empresas.** Las empresas deben crear sus propias bibliotecas de clases, que reflejen sus estandares internos y necesidades de aplicación. La identificación descendente de los objetos en la empresa es un aspecto importante de la ingeniería de la información.

Complejidad de un programa

La complejidad de un programa es una medida de su comprensibilidad. Cuanto más complejo sea un programa, más difícil será de comprender. La complejidad es una función de la cantidad de posibles rutas de ejecución del programa y de la dificultad de rastreo de las rutas. Si se evita la complejidad excesiva de un programa, mejorará su confiabilidad y se reducirá el esfuerzo requerido para su desarrollo y mantenimiento.

La métrica de complejidad ciclomática McCabe, llamada así por su creador, es la medida de complejidad de más uso. Tiene programas medidos en lenguajes FORTRAN, COBOL, PL/1, C y Pascal. Ha sido automátizada por la ingeniería y por muchas herramientas CASE. La complejidad ciclomática es una medida de la complejidad de la teoría de gráficas, basada en el conteo de la cantidad de rutas lógicas individuales contenidas en el programa.

Los analistas de programas que miden de manera automática el número de McCabe revelan que muchos programas tienen una métrica McCabe de 15. McCabe recomienda que el número no debe exceder de 10. El autor a encontrado que los módulos y los programas que contienen módulos cuya complejidad ciclomática era mayor de 10 tenían más problemas y eran menos confiables.

El diseño y la programación orientados a objetos dan como resultado menores métricas McCabe y, por lo tanto, facilitan la depuración y el mateniemiento de software. Por ejemplo, el desarrollo de software COOPERATION en NCR era un proyecto masivo realizado con técnicas orientadas a objetos. Las técnicas produjeron un métrica McCabe de 3. En proyectos anteriores de software no orientados a objetos, NCR, tenía una métrica McCabe con un promedio superior a 10. La razón de tan radical diferencia entre números es que cada método es relativamente sencillo y autocontenido.

Análisis y diseño de un sistema orientado a objetos

Cuando analizamos sistemas, creamos modelos del área de aplicación que nos interesa. Un modelo puede incorporar un sistema, centrarse en un área de la empresa o abarcar toda la empresa. El modelado de empresas es importante para la planeación de la automatización de las mismas.

El modelo representa un aspecto de la realidad y se construye de modo que nos ayude a comprender a ésta. El modelo es mucho más sencillo que la realidad, al igual que un avión a escala es mucho más sencillo que un avión de verdad. Podemos manejar el modelo y esto nos ayudará a idear sistemas o rediseñar áreas de la empresa.

Con el análisis orientado a objetos, la forma de modelar la realidad difiere del análisis convencional, modelamos el mundo en términos de tipos de objetos y lo que le ocurre a éstos. Esto implica también diseñar y programar sistemas de forma orientada a objetos.

La figura 1.1.2.1 ilustra nuestra forma de construir sistemas. El analista crea un modelo del área de interes. El modelo se convierte en un diseño y más adelante en un código. El modelo debe representar como perciben los usuarios finales el área o lo que los usuarios desean que realice el sistema. En la medida de lo posible, este modelo debe tener una forma comprensible para los usuarios y ayudarlos a ser creativos con repecto a sus necesidades. El técnico utiliza el modelo y crea un diseño a partir del cual se pueda escribir o generar un código.

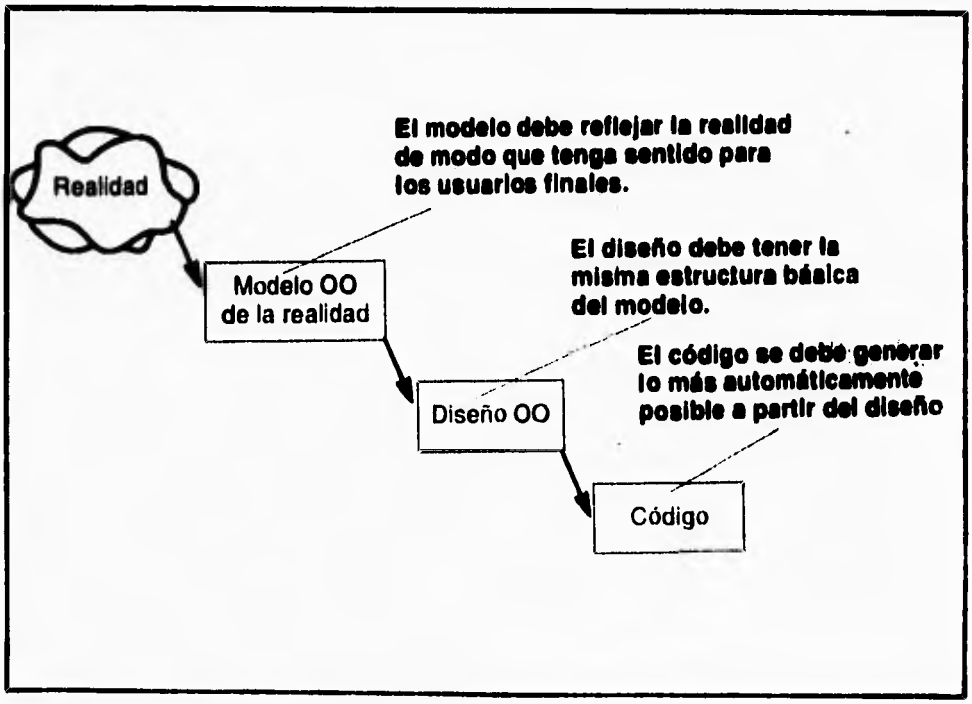


Fig. 1.1.2.1. muestra la forma de construir sistemas.

Los modelos que construimos en el análisis orientado a objetos reflejan la realidad de modo más natural que los del análisis tradicional de sistemas. Después de todo, la realidad consta de objetos y eventos que cambian el estado de dichos objetos. Mediante las técnicas orientadas a objetos, construimos software que modela más fielmente el mundo real. Cuando el mundo real cambia, nuestro software es más fácil de cambiar, lo que es una ventaja real.

Quisieramos capturar el punto de vista de los usuarios con respecto al mundo y traducirlo en software de la manera más automática posible. Entonces, cuando cambien las necesidades de los usuarios, el software cambiará con ellas.

Por ejemplo, el diseñador de un microchip no pensaria en trabajar sin poderosas herramientas computarizadas. De la misma forma, se necesitan las herramientas para el diseño de software.

Con las herramientas Orientadas a Objetos, el diseño y código avanzan de manera natural a partir del modelo de alto nivel. Todo se relaciona con los tipos de objetos, los métodos que utilizan los objetos y los eventos que activan las operaciones con objetos. El depósito almacena muchas clases y nos ayuda a localizar las que sean utiles en el diseño.

Dos tipos de modelos

En el análisis Orientado a objetos construimos dos modelos estrechamente relacionados: un modelo de los tipos de objetos y sus estructuras; y un modelo de lo que les ocurre a los objetos. Los 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 tiene dos aspectos los que se ilustran en la figura 1.1.2.2. El primer aspecto se refiere a los tipos de objetos, clases, relaciones entre los objetos y herencia; se le conoce como análisis de la estructura de objetos (DEO) El otro aspecto del comportamiento de los objetos y lo que les ocurre al paso del tiempo; se conoce

como análisis del comportamiento de objetos (ACO) y diseño del comportamiento de objetos (DCO)

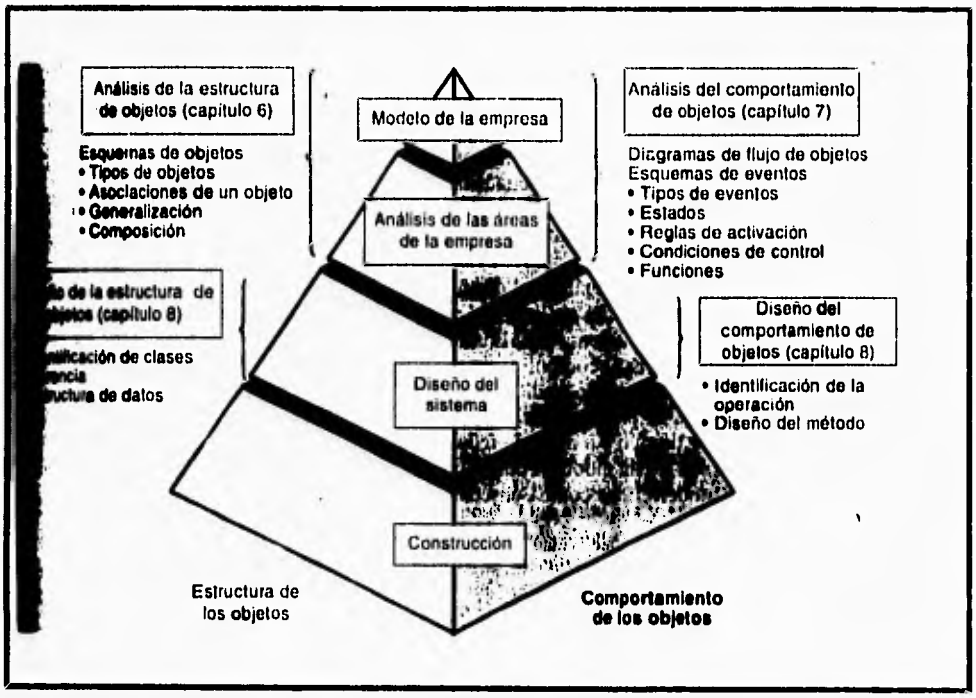


Fig. 1.1.2.2 Análisis y diseño se dividen en modelos relacionados entre sí; a la izquierda están los modelos de la estructura de un objeto y a la derecha están los modelos para el comportamiento de objetos.

Análisis de la estructura de objetos

El análisis de la estructura de objetos (AEO) define las categorías de los objetos que percibimos y las formas en que los asociamos.

Durante el análisis de la estructura de objetos, 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 importantes, puesto que crean los bloques conceptuales de

construcción para el diseño de sistemas. En la programación orientada a objetos, estos bloques de construcción guían al diseñador en la definición de las clases y sus estructuras de datos. Además, los tipos de objetos son un índice para los procesos del sistema. En otras palabras, un objeto sólo debe ser controlado por medio de las funciones asociadas con su tipo. Así, las operaciones no se pueden definir de manera adecuada sin los tipos de objetos.

Los tipos de objetos que definimos y utilizamos son variados, puesto que los elegimos con base en la comprensión de nuestro mundo.

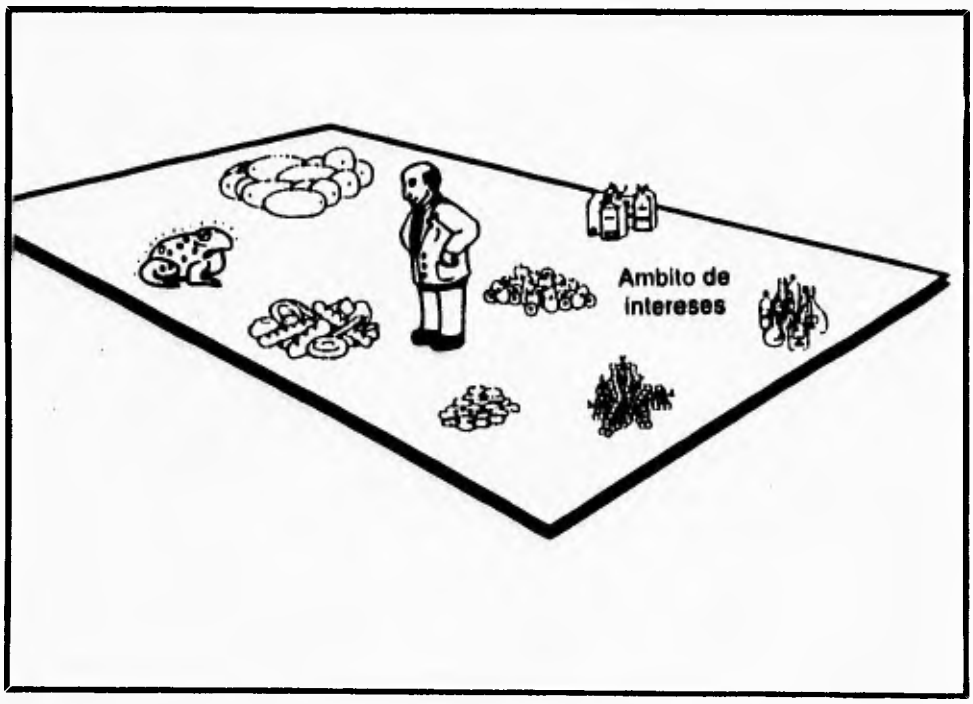


Fig. 1.1.2.3 Nosotros elegimos la forma de dar categorías a nuestro mundo.

Aunque esto pudiera parecer muy arbitrario, así funciona nuestra mente. De hecho, un objeto se puede categorizar en más de una forma.

Por ejemplo,en la figura 1.1.2.4 una persona puede considerar al objeto ***Isabel*** como **mujer**. Su jefe la considera un **empleado**. El señor que le arregla el jardín la considera su **patrona.** La agencia local para el control de los animales la considera **dueña de una mascota.** El banco informa que ***Isabel*** es una instancia del tipo de objeto llamado de **buen sujeto de crédito**, etc.

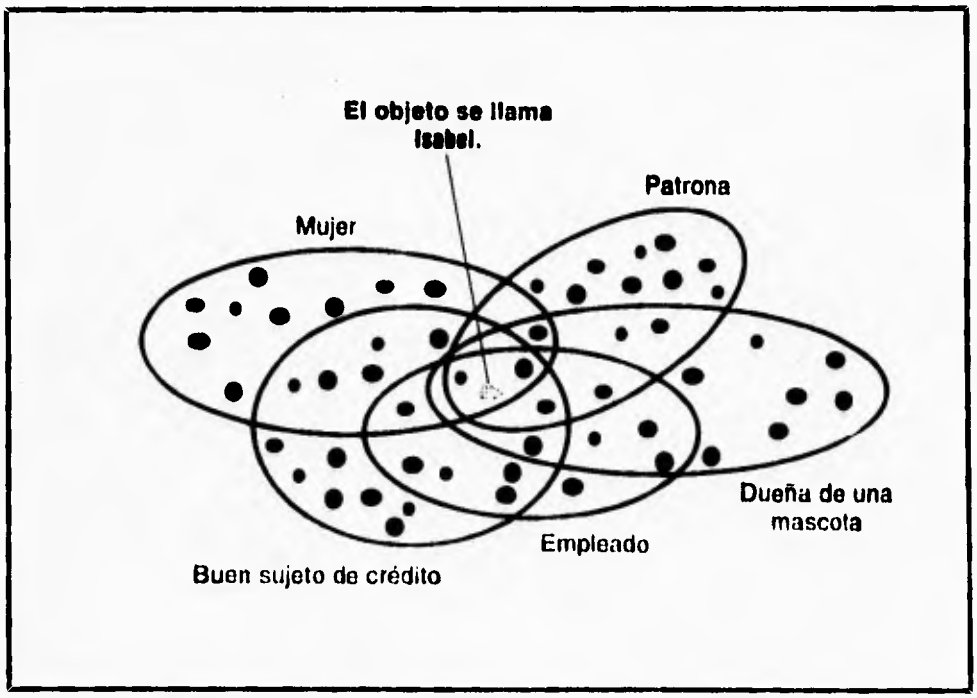


Fig 1.1.2.4 El mismo objeto se puede categorizar de varias formas.

Asociaciones de objetos

Así como se ha mencionado, los tipos de objetos que clasifican los objetos de nuestro entorno son vitales en el análisis orientado a objetos. También es importante **modelar** la

forma en como los objetos se asocian entre sí. En el análisis, es útil nombrar de alguna forma las asociaciones e indicar la cantidad de objetos de otro tipo, puesto que esto le da significado y aumenta la comprensión de la asociación.

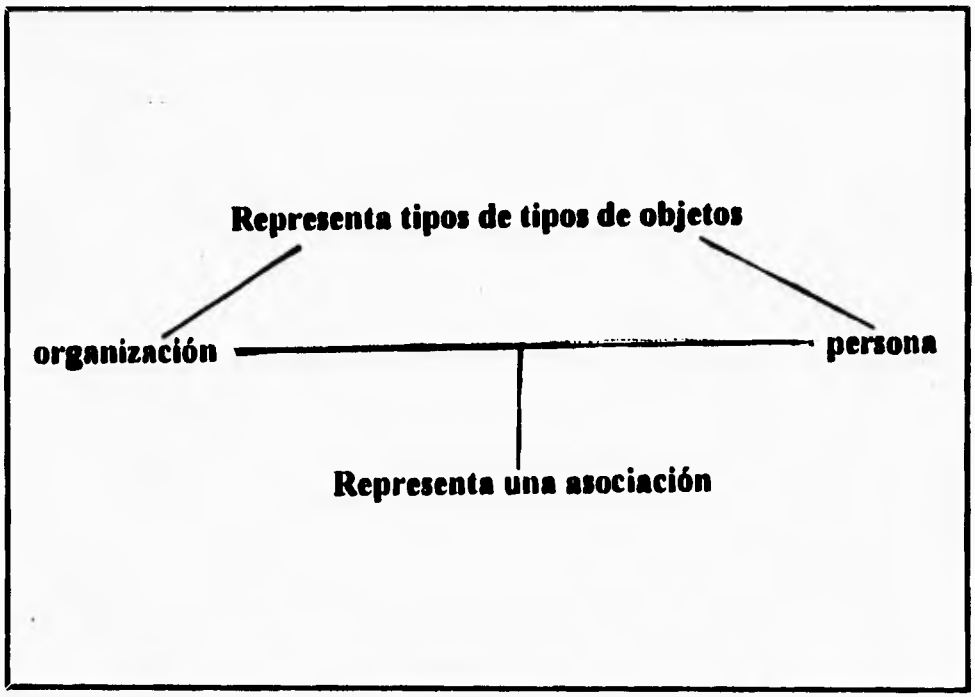


Fig. 1.1.2.5 Los objetos de un tipo se asocian con los objetos de otros tipos

Aunque la figura 1.1.2.5 ilustra las asociaciones entre dos tipos de objetos no se indica el significado de la asociación. Además en el significado, tampoco se indica la cantidad de objetos con los que un objeto dado puede y debe asociarse. En el análisis, es útil nombrar de alguna forma 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.

Generalización

Una de las vías de sentido común por lo que el hombre organiza su volumen de conocimiento es el de las jerarquías, de lo más general a lo más específico. Por ejemplo, la siguiente figura muestra una jerarquía con el conocimiento del tipo objeto persona en la parte superior. Esto indica que persona es un tipo de objeto más general que empleado y estudiante. Esto significa que empleado y estudiante son subtipos de persona; o bien que persona es un supertipo de empleado y estudiante.

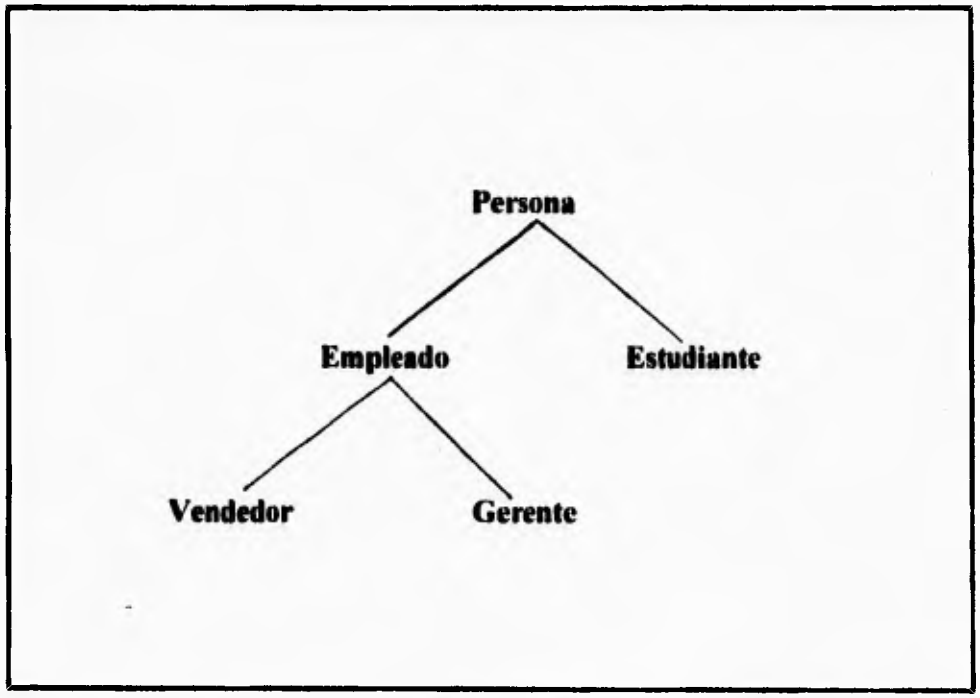


Fig. 1.1.2.6 Una jerarquía de generalización de tipo de objetos que indica que persona es un supertipo de empleado y estudiante. Empleado es un supertipo de vendedor y gerente.

La *generalización* es el resultado de distinguir un tipo de objeto como más general, o incluso, que es más que otro. 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 de objeto puede tener subtipos, sub-subtipos, etc. Por ejemplo, ácido es subtipo de líquido y ácido nítrico es un subtipo de ácido. El producto 739 es un subtipo de ácido nítrico la siguiente figura muestra esta jerarquía de generalización.

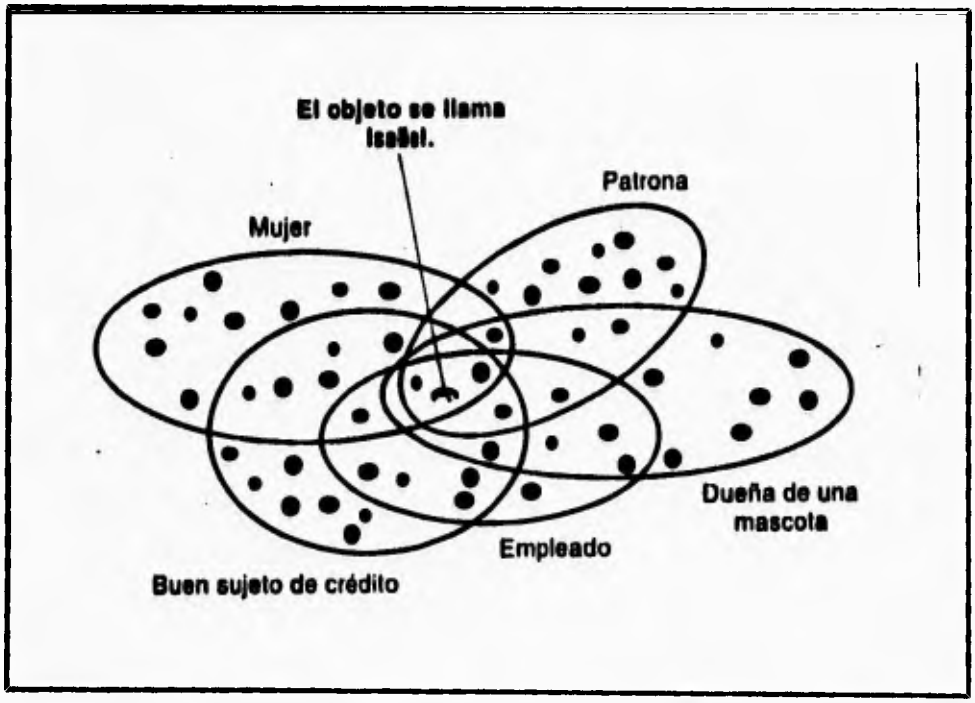


Fig. 1.1.2.7 Jerarquía de generalización.

La figura muestra una jerarquía en la que cada tipo no tiene más de un supertipo. Sin embargo es frecuente que un tipo tenga varios supertipos. Así, las jerearquías de generalización no tienen que ser necesariamente jerarquías de árbol.

Las jerarquías de generalización son importantes para el desarrollo orientado a objetos por dos 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 programción orientada a objetos.

Jerarquías Compuestas

Algunos tipos de objetos se consideran complejos. Por complejo queremos indicar que determinados objetos estan formados por otros. Por ejemplo, un auto se puede describir como algo formado por chasis, llantas y motor. A su vez el motor esta compuesto por el monoblock, valvulas, pistones, etc. Además el objeto complejo formado por anillos, una biela y una cabeza.

En el análisis orientado a objetos, la composición de un objeto nos ayuda a describir el hecho de que los dibujos estan formados por determinada configuración de símbolos, los trabajos lo están por tareas específicas, las organizaciones por otras organizaciones, y así sucesivamente. En los sistemas de tecnología avanzada, el análista describirá la forma en que un pedido no sólo puede constar de elementos en cada renglón, sino también contener instrucciones verbales del cliente o un diagrama hecho a mano. Los pedidos de este tipo se llaman objetos complejos. Cada pedido se puede controlar como un objeto unico que conste de otros, los cuales a su vez, se pueden controlar de forma independiente, en caso necesario.

Diagramas de relación entre los objetos

Los tipos de objetos están relacionados con otros tipos de objetos. Un empleado trabaja en una sucursal. Un cliente realiza un pedido de varios productos.

Los diagramas de relación entre entes se han venido utilizando por años en el análisis convencional de sistemas. Un diagrama de relación entre objetos es esencialmente igual a un diagrama de relación entre entes.

Esquemas de 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; los supertipos y subtipos se representan en un diagrama de jerarquía de generalización y las estructuras compuestas en un diagrama compuesto. Sin embargo, para los usuarios más sofisticados puede ser útil presentarlo todo en el mismo diagrama. Este tipo de diagrama se llama esquema de objetos. como se muestra en la figura. 1.1.2.8.

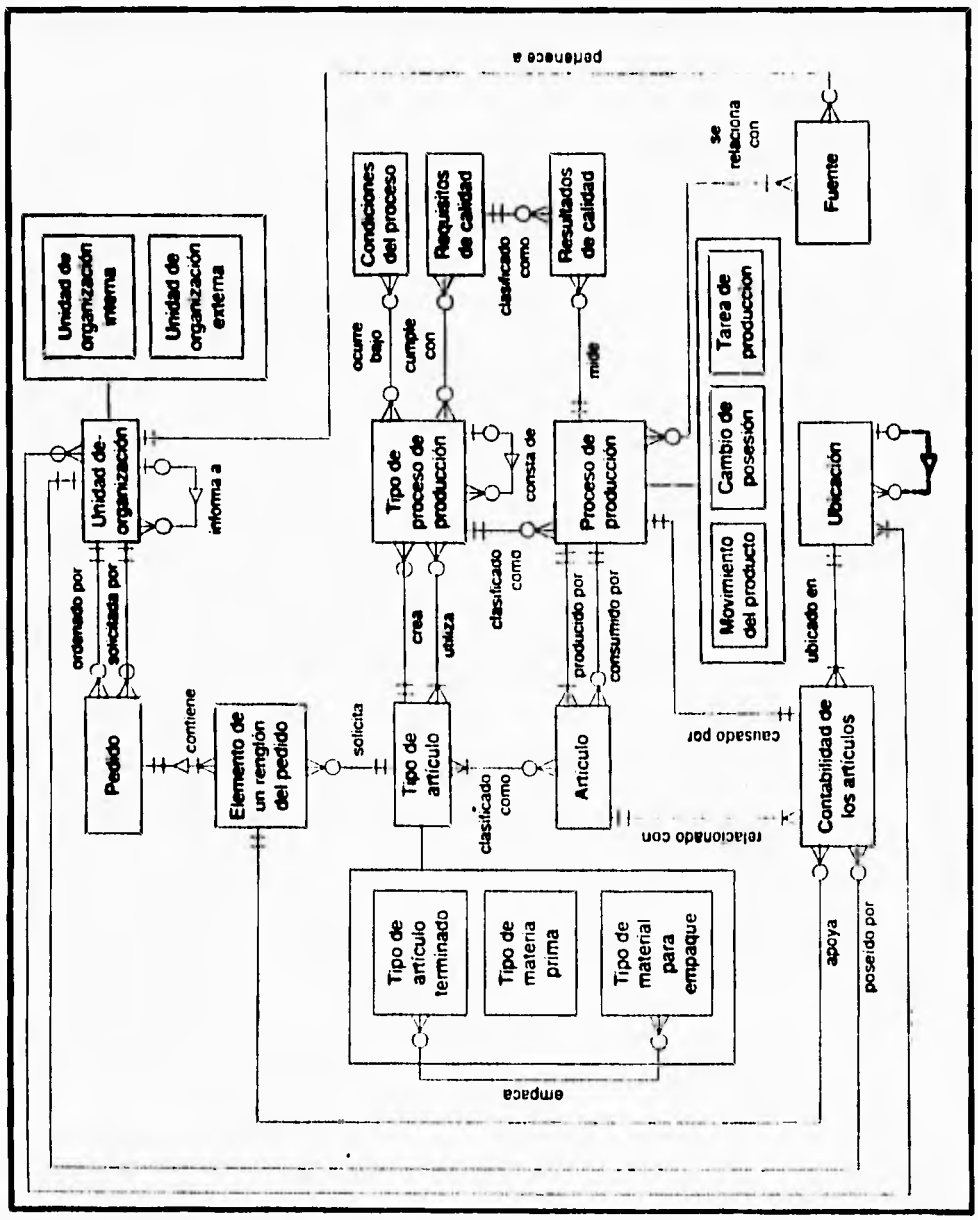


Fig. 1.1.2.8 Ejemplo de esquema de objetos para una aplicación en producción.

Análisis del comportamiento de objetos

En el análisis del comportamiento de objetos (ACO) realizamos esquemas de eventos que muestran eventos, la secuencia en que ocurren y cómo 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 de objetos.

Estados de un objeto

Un objeto puede existir en varios estados. Por ejmplo, un objeto reservación aérea 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. Sin embargo, un objeto puede tener una gran variedad de perspectivas de ciclos vitales. Por ejemplo, el mismo objeto reservación aérea también puede tener los siguientes estados relacionados con el pago:

- Reservación no liquidada
- Reservación con un pago de depósito
- Reservación totalmente pagada
- Reservación para reembolso
- Reservación reembolsada

En algún momento, un objeto podría ser reservación en lista de espera y reservación totalmente pagada. Al mismo tiempo, el mismo objeto podría ser una instancia de reservación de junio y reservación de la compañía IBM. En otras palabras, un objeto puede ser al mismo tiempo una instancia de varios tipos de objeto.

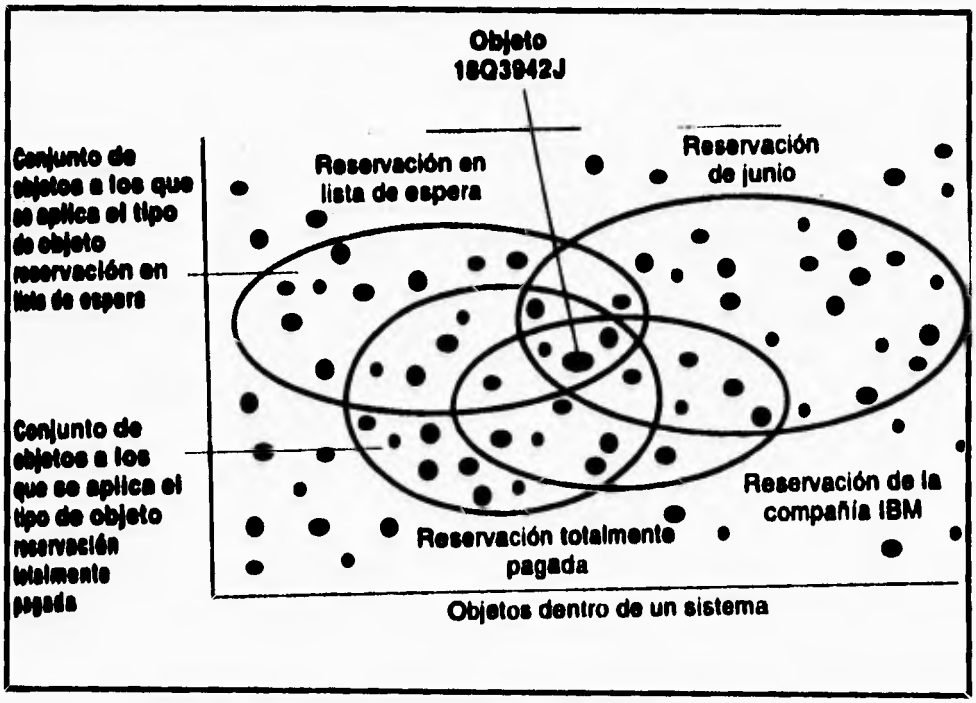


Fig. 1.1.2.9 Al ocurrir un cambio de estado, un objeto pasa de una categoría a otra.

El estado de un objeto se puede definir como la colección de los tipos de objeto que se aplican a él.

Al implantarlo en un lenguaje de programación orientada a objetos, el estado se registra en los datos almacenados con relación al objeto. Se determina mediante las clases y valores de los campos de los datos asociados con el objeto. Así en general, los programadores orientados a objetos utilizan otra definición de estados.

El estado de un objeto es la colección de asociaciones que tiene un objeto.

En el lenguaje orientado a objetos, las solicitudes se envían y provocan la activación de los métodos. Los métodos cambian el estado del objeto. El estado se registra en los datos del objeto.

Tipos de 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 estados. Un *evento* es un cambio en el estado de un objeto.

Por ejemplo en la siguiente figura un estado del objeto pasa a ser una reservación en lista de espera, a una reservación confirmada. El nombre de este evento podría ser reservación en lista de espera para el objeto 18Q3942J confirmada.

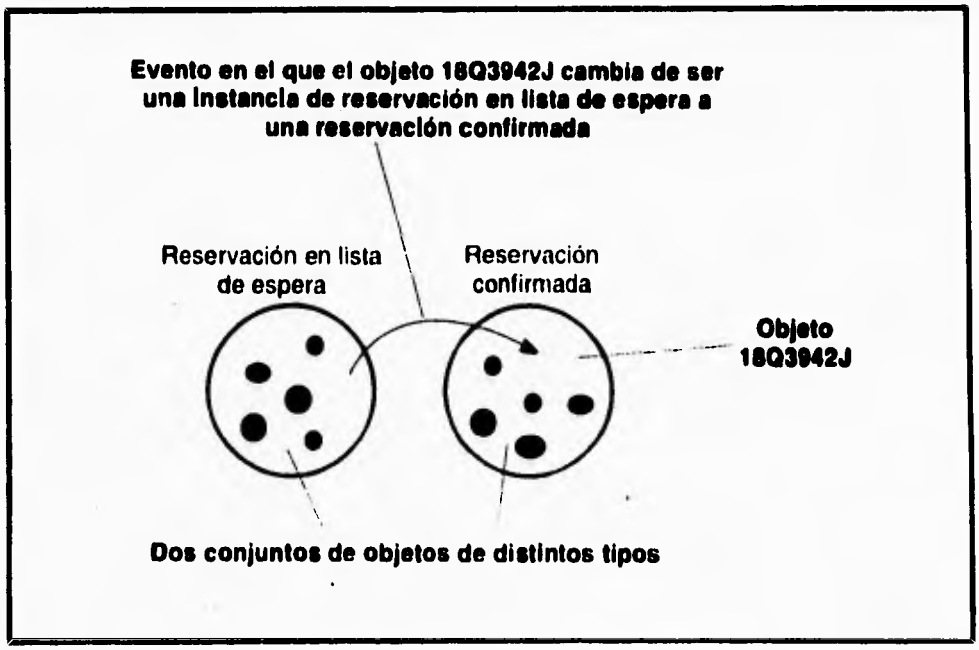


Fig. 1.1.2.10 Estado de un objeto que cambia de un tipo de objeto a otro.

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 la mayoría de las aplicaciones, sí debe de cambiar el contenido de las bases de datos. Puesto que deseamos saber de esos cambios y reaccionar en forma adecuada ante ellos, debemos entender y modelar los eventos.

Un análista orientado a objetos no desea conocer cada evento que ocurra en una organización: sólo los tipos de eventos. Así como hablamos de tipos de objetos e instancias de tipos de objetos, podemos hablar de tipos de eventos e instancias de tipo de eventos.

Los tipos de eventos indican los cambios sencillos en el estado de un objeto; por ejemplo, cuando se deposita dinero en una cuenta bancaria. Básicamente, los tipos de eventos describen las siguientes formas de cambios de estado:

- Un objeto se crea. Por ejemplo se crea una reservación aérea.
- Un objeto se termina. Por ejemplo, un producto se destruye o un contrato se termina.
- Un objeto se clasifica como una instancia de un tipo de objeto. Por ejemplo, un empleado se convierte en gerente.
- Un objeto se desclasifica como una instancia de un tipo de objeto. Por ejemplo, una firma deja de ser cliente; un producto sale del catalogo de ventas.
- Un objeto cambia de clasificación. Por ejemplo, un abogado pasa de ayudante a socio; una cuenta cambia de normal a atrasada.
- El atributo de un objeto se cambia.

Los eventos pueden asociar un objeto con otro. Por ejemplo, en la mayoría de las organizaciones, cuando un objeto se clasifica como **empleado**, debe estar asociado con un Departamento. Un objeto clasificará al objeto como **empleado**. Otro evento creará una asociación entre el objeto **empleado**, y un objeto **Departamento**. Si el empleado, cambia de departamento, se crea una nueva asignación **empleado-departamento** y se termina la anterior.

Una operación hace que los eventos ocurran. Dibujamos la operación como un cuadro con esquinas redondeadas, puesto que los eventos indican los puntos en el tiempo en que se da el cambio de estado de un objeto. Los tipos de eventos se representan como triángulos negros llenos, generalmente unidos a la caja de operación. Al representarse de esta forma, parecen apuntar a un tipo particular de "punto en el tiempo".

Según el área que se modele, puede ocurrir más de un evento al terminar una operación, y cada uno de éstos puede activar operaciones independientes.

Por ejemplo en un almacén que tiene anaqueles se puede expresar el cambio de tres eventos con el siguiente esquema:

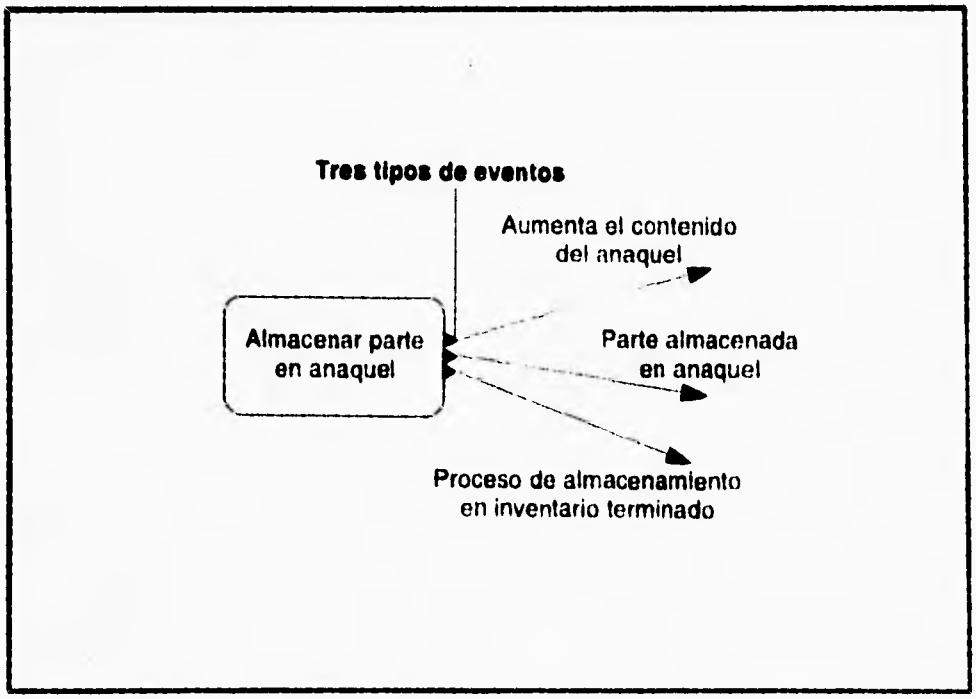


Fig. 1.1.2.11 Tres tipos de eventos.

Reglas de activación

Cuando ocurre un evento, lo usual es que el cambio de estado active el llamado a una o más operaciones.

En el análisis orientado a objetos el termino 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 cómo llevar a cabo la operación. Es el guión de la operación. A nivel programa, el método es el código que implanta la operación.

Las operaciones se invocan. Una operación invocada es una instancia de una operación. Una operación puede o no cambiar el estado de un objeto. Si lo cambiara ocurriría un evento. Las operaciones se representan mediante cuadros con esquinas redondeadas. Los tipos de eventos se representan mediante triángulos sólidos negros conectados a la caja.

Por ejemplo la figura 1.1.2.12 describe que 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.

Así las reglas de activación definen la relación entre la causa y el efecto. Siempre que ocurre un efecto de cierto tipo, la regla de activación invoca a una operación ya definida. Un tipo de operación 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.

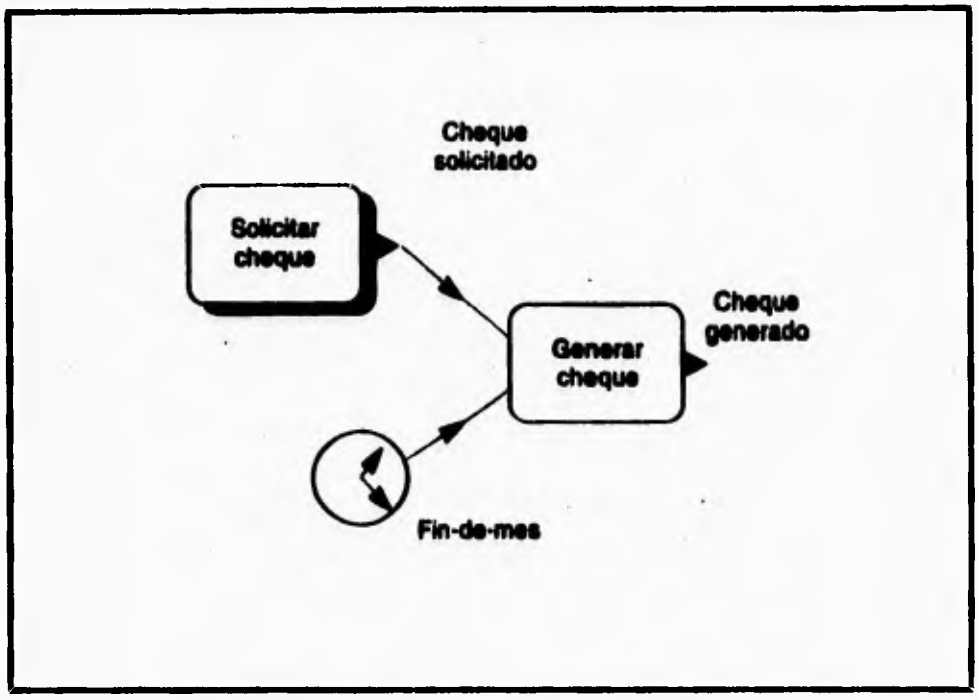


Fig. 1.1.2.12 La operación producir cheque.

Condiciones de control

Como se mostró anteriormente, una operación puede ser invocada por una o varias reglas de activación. Sin embargo, antes de invocar de hecho a la 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 a 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.

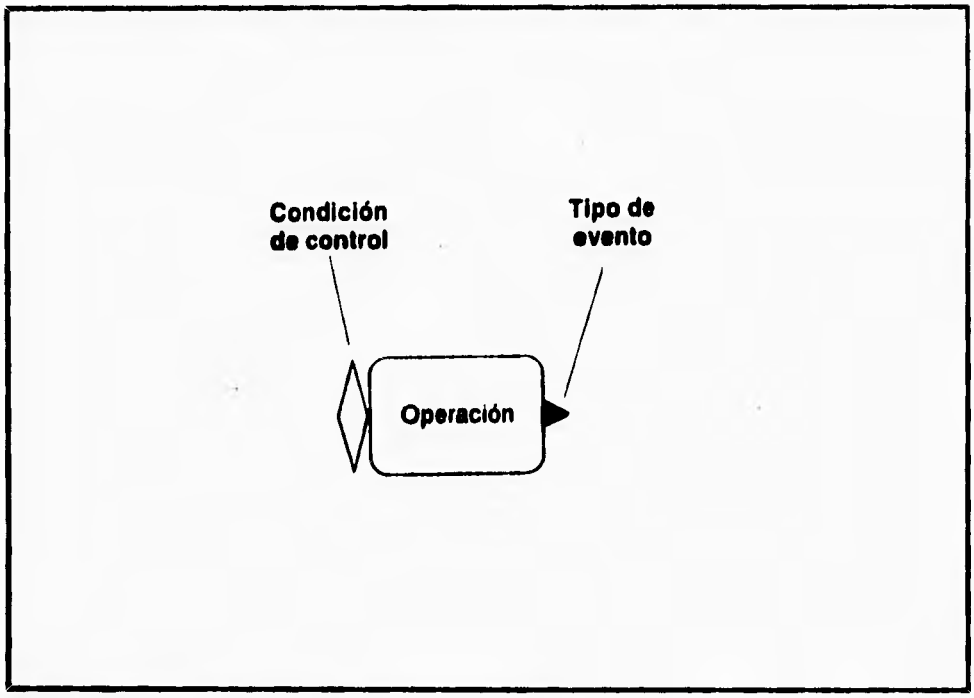


Fig. 1.1.2.13 Condición de control.

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

Diagrama de flujo de datos

Los esquemas de eventos son adecuados para la descripción de procesos en términos de eventos, de reglas de activación, de condiciones de control y de operaciones.

Sin embargo, podría no ser adecuado expresar así procesos grandes y complejos. En ocasiones, un área del sistema es demasiado vasta e intrincada como para expresar la dinámica de los eventos y las formas de activación. además es posible que sólo se necesite

un alto nivel de comprensión. Esto es cierto sólo a nivel estratégico. En situaciones como esta es cuando es útil un diagrama de flujo de objetos.

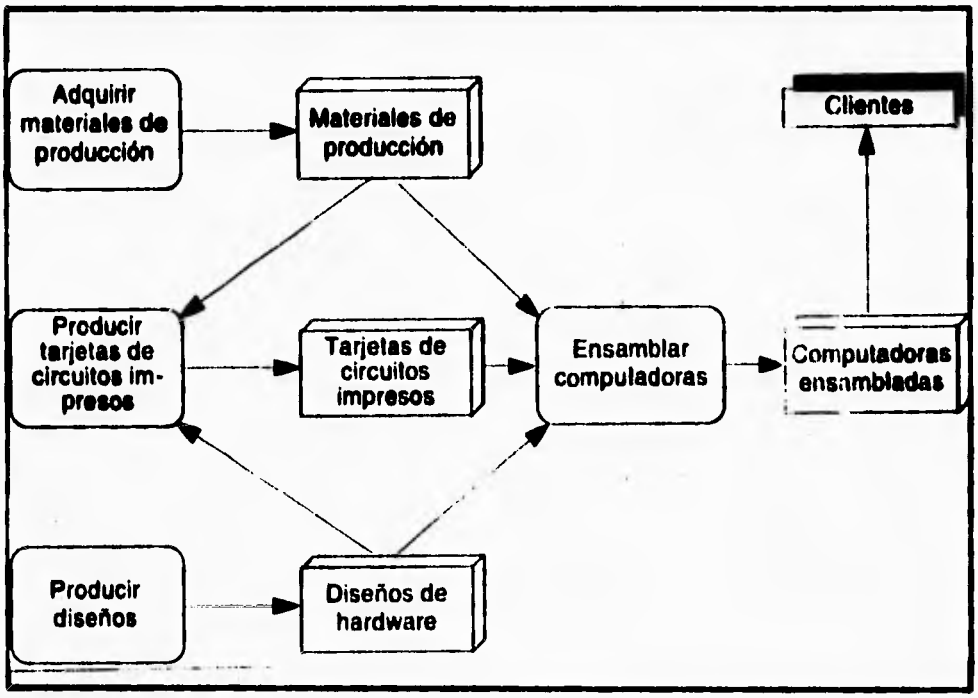


Fig. 1.1.2.14 Diagrama de flujo de objetos

Los diagramas de flujo de objetos son parecidos a los diagramas de flujo de datos, puesto que muestran las actividades que interactuan con otras. En los diagramas de flujo de datos, una interfaz transfiere datos. En las técnicas orientadas a objetos, no queremos limitarnos a la transferencia de datos, sino que el diagrama debe representar cualquier tipo de cosa que transfiera de una actividad a otra, ya sean pedidos de partes, artículos terminados, diseños, servicios, hardware, software, o datos. En resumen, el diagrama de flujo de objetos indica los objetos que se producen y las actividades que los producen e intercambian. Las personas familiarizadas con los diagramas de flujo de datos, reconocerán

enseguida las cajas de actividades redondeadas y de las direcciónes de las líneas de flujo. Sin embargo no esta presente el símbolo de almacenamiento de datos. En su lugar se utiliza una caja tridimensional. La tridimensionalidad del símbolo indica que el diagrama de flujo de objetos representa el hecho de que los objetos de la vida real fluyen entre las actividades.

Los diagramas de flujo de objetos y cómo se producen y se consumen:

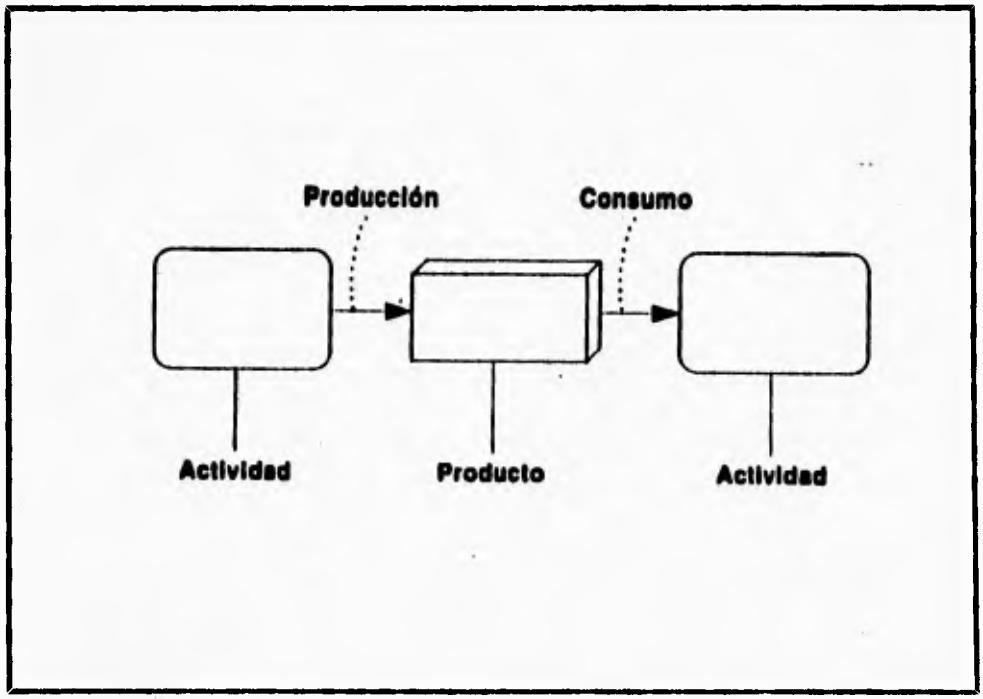


Fig. 1.1.2.15 Función de un diagrama de flujo.

El producto es el resultado final que satisface el propósito de la actividad, sin embargo, los productos no sólo se producen en aras de la producción. Se producen para el consumo por parte de otras actividades que le añaden valor al producto consumido. A cada

paso se crean cualidades nuevas, más complejas y sutiles. De esta forma el diagrama de flujo de objetos, se puede utilizar en la planeación estratégica de la empresa, así como para la planeación estratégica de la información.

Así, los diagramas de flujo de objetos pueden representar el modelo de forma ascendente o descendente. En particular los diagramas de flujo de objetos son muy útiles para los modelos de organizaciones en forma descendente a nivel estratégico. Las actividades se pueden descomponer en diagrama de flujo de objetos. Sin embargo, en un nivel más detallado del análisis 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.

Cada actividad se puede expresar en forma más detallada. Para expresar un proceso de manera 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 los eventos y las claves de activación.

Diseño de la estructura de objetos

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 división: el diseño de la estructura de objetos (DEO) y el diseño de comportamiento de objetos (DCO). Los lenguajes de programación orientado 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, y se describirán en forma conjunta.

CLASE

En el análisis de estructura de objetos, identificamos los tipos de objetos; en el diseño de estructura de objetos nos centramos en la implantación de esos tipos de objetos.

Clase **es la implantación de un tipo de objeto.**

En la siguiente figura se muestra una clase. La clase especifica la estructura de datos y los métodos operativos permitidos que se aplican a cada uno de sus objetos. La especificación de cómo se llevan a cabo las funciones de una clase se llama método. Los objetos se pueden utilizar exclusivamente con métodos específicos.

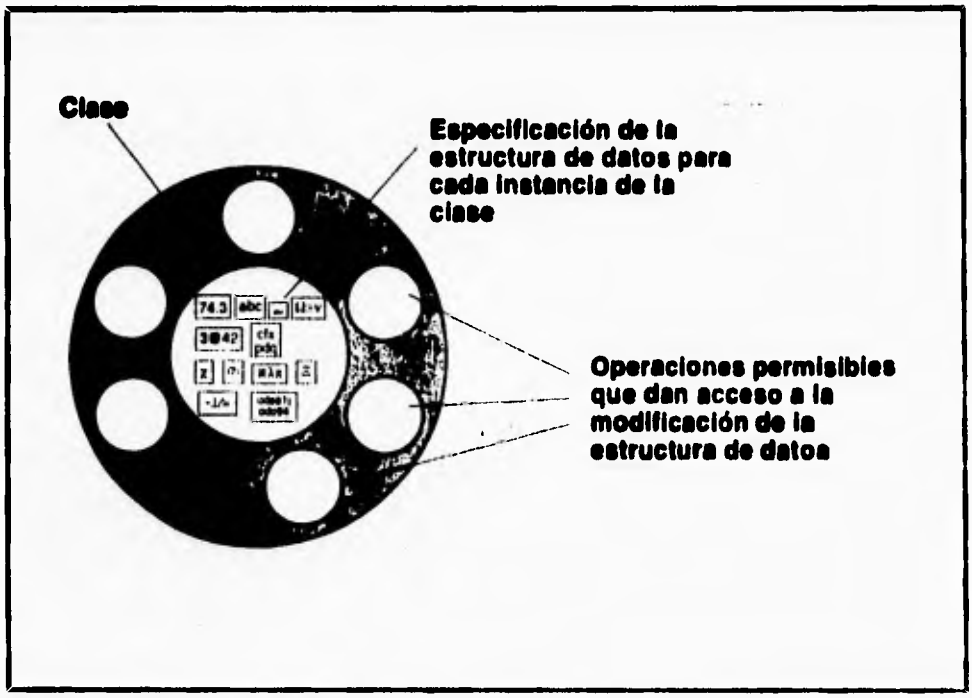


Fig. 1.1.2.16 Cada clase especifica una estructura de datos, las operaciones permisibles y los métodos operativos.

En la siguiente figura se muestra un objeto que es instancia de la clase de la figura anterior. Los datos y operaciones que encapsula quedan especificados por su clase. Los datos del objeto se almacenan dentro de él y se tiene acceso a ellos y se les modifica sólo mediante las operaciones permisibles. Esta restricción al acceso se debe al encapsulado. El encapsulado protege los datos del uso arbitrario o no pretendido. El acceso a la actualización directa de los datos de un objeto por parte del usuario violaría el encapsulado.

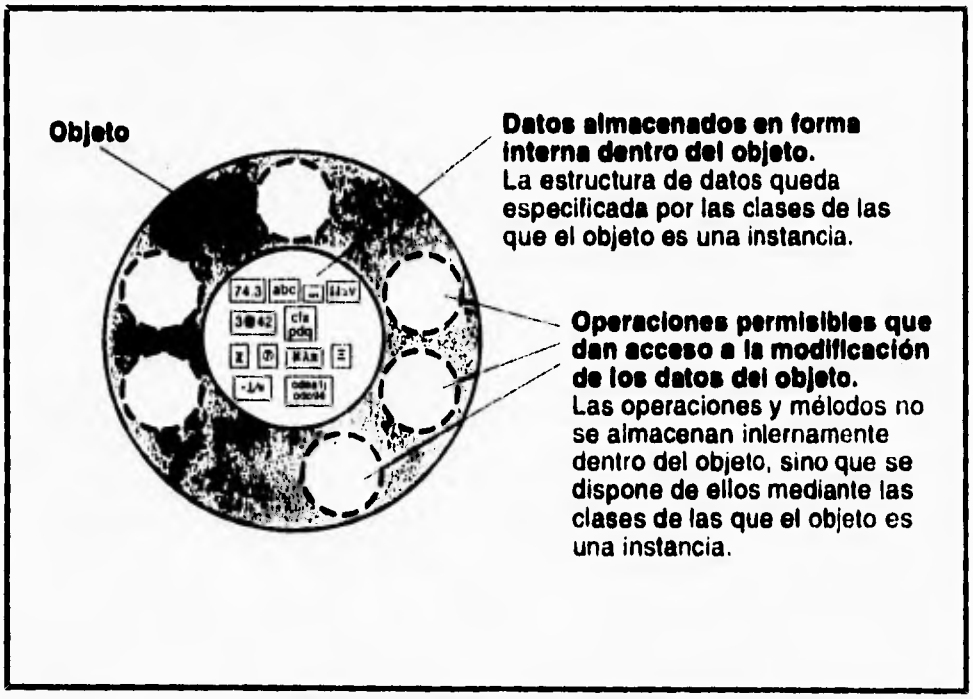


Fig. 1.1.2.17 El encapsulado oculta la implantación de la estructura de datos y métodos de cada objeto.

Los usuarios observan el comportamiento del objeto en términos de las operaciones que se pueden aplicar a los objetos, así como los resultados de tales operaciones. Estas operaciones forman la interfaz del objeto con sus usuarios.

Las operaciones son procesos que se pueden solicitar como unidades. Los métodos son especificaciones del procedimiento de una operación dentro de una clase.

Una **operación** es un proceso que se puede solicitar como unidad. Un ***método*** es la especificación de una operación.

Herencia de clase

La generalización es una noción conceptual. La herencia de clase es una implantación de la generalización. La generalización 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 las operaciones de una superclase permite que las clases compartan el código. La herencia de estructura de datos permite la reutilización de la estructura.

Herencia múltiple

En la herencia múltiple, una clase puede heredar estructuras de datos y operaciones de más de una superclase.

La herencia de clase implanta la jerarquía de generalización, y permite así que una clase comparta la estructura de datos y operaciones de otra clase.

La herencia simple es aquella en la que una clase puede heredar la estructura de datos y operaciones de una superclase.

La herencia múltiple se da cuando una clase puede heredar la estructura de datos y operaciones de más de una superclase.

Selección del método

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 esto causaría 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 una superclase.

Analogía de análisis y diseño

En la metodologías tradicionales para la generación de sistemas, los modelos conceptuales utilizados para el análisis difieren de los que se emplean para el diseño. La programación tiene también un tercer punto de vista del mundo. Los analistas utilizan los modelos de relación entre entes, la descomposición funcional y las matrices. Los diseñadores utilizan diagramas de flujo de datos, tablas de estructuras y diagramas de acción. Los programadores utilizan las construcciones COBOL, FORTRAN, C o ADA.

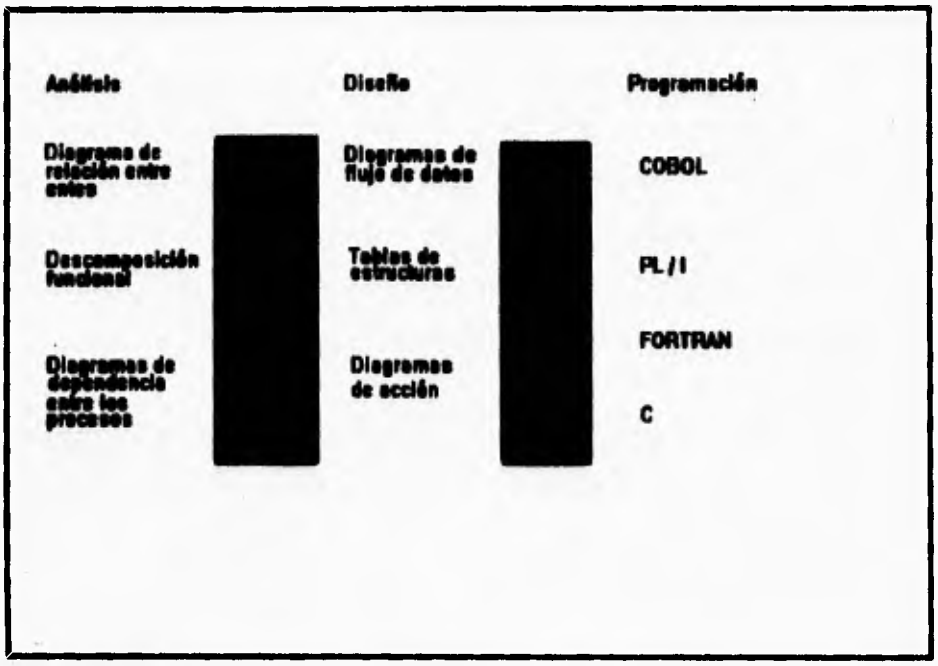
En las técnicas orientadas a objetos, todos utilizan el mismo modelo conceptual: analistas, diseñadores, programadores y, de modo particularmente importante, los usuarios finales como lo muestra la fig. 1.1.2.18.

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 de su padre. 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 u operaciones.

La transición del análisis al diseño es tan natural, que a veces es difícil especificar el punto final de análisis y el punto inicial del diseño.

Cuando el desarrollo tradicional cruza los muros que se muestran en la figura 1.20, a veces se pierde información y aparecen las incomprensiones. La transición consume tiempo y a menudo reduce la calidad del producto final.

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



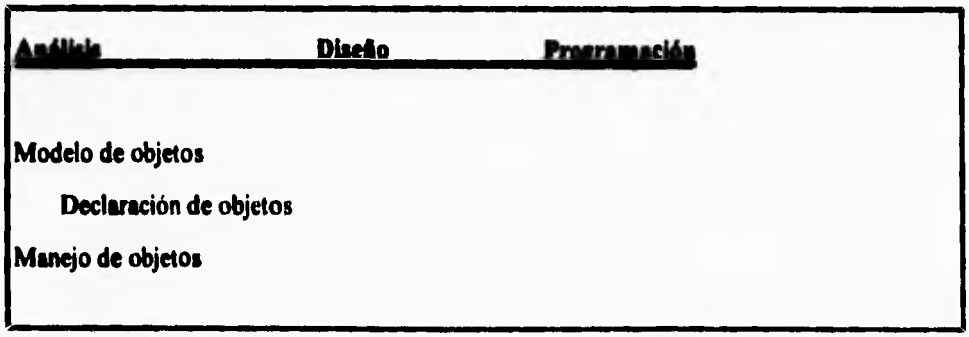


Fig. 1.1.2.18 Las técnicas Orientadas a Objetos utilizan el mismo modelo conceptual para análisis, diseño y programación Orientadas a objetos. Las técnicas Orientadas a Objetos derriban los muros conceptuales entre el análisis, el diseño y la programación (o generación de códigos).

El empleo de un modelo conceptual único tiene como consecuencias

- Una mayor productividad
- Menos errores
- Una mejor comprensión entre los usuarios, análistas, diseñadores y técnicos
- Resultados de mejor calidad
- Más flexibilidad
- Mayor inventiva

La figura 1.1.2.19 muestra los objetivos tanto para el análisis como para el diseño.

El análisis de la estructura de objetos se ocupa de definir las categorías de objetos y la forma en que los asociamos. Preguntamos: ¿Qué tipos de objetos hay? ¿Cuáles son sus relaciones y funciones? ¿Qué subtipos y supertipos son útiles? ¿Hay algún tipo de objeto compuesto por otros objetos?

Cuando el análisis pasa a la etapa del diseño de la estructura de objetos, identificamos las clases (la implantación de tipos y objetos). Se definen las superclases, subclases, rutas de herencia y los métodos a utilizar y se lleva a cabo el diseño en detalle de las estructuras de datos.

El análisis del comportamiento de objetos se ocupa de modelar lo que ocurre a los objetos al paso del tiempo. Nos preguntamos ¿En qué situación pueden estar las clases de objetos? ¿Qué tipo de eventos cambian estos estados? ¿Qué sucesión de eventos ocurre? ¿Qué funciones resultan de estos eventos y cómo se activan?

Después sigue la etapa de diseño. En ésta nos preocupamos por el diseño detallado de métodos, ya sea con técnicas por procedimientos o de no procedimientos. Se crea la entrada para los generadores de códigos. Se diseña la pantalla y se diseñan y generan los diálogos. Finalmente, se construyen y desarrollan los prototipos.

Análisis de la estructura de objetos (AEO)	Análisis de Comportamiento de objetos (ACO)
Se ocupa de los tipos de objetos y sus asociaciones	**Se ocupa de lo que le sucede a los objetos al paso del tiempo**
• Tipos de objetos y asociaciones	• Diagrama de flujo de objetos
• Diagramas de generalización	• Esquemas de eventos
• Diagramas de relación entre los objetos	• Diagramas de funcionamiento que muestran las funciones y la secuencia en que deben ocurrir
• Diagramas de componentes	• Estados de objetos y cambios en dichos estados

Diseño de la estructura de objetos (DEO)	Diseño del comportamiento de objetos (DCO)
Se ocupa de las clases, métodos y herencia	**Se ocupa del diseño de métodos**
• Clases	• Métodos y funciones
• Superclases y subclases	• Lógica de procedimientos
• Herencia	• Código de no procedimientos
• Estructuras de datos	• Entrada para los generadores de códigos
• Diseño de una base de datos	• Diseño de la pantalla y del diálogo
	• Fabricación de prototipos.

Fig. 1.1.2.19 Las dos mitades del análisis y diseño OO.

1.1.3 DISEÑO CON ARBOLES GENEALOGICOS.

Las estructuras de árboles se han utilizado en la computación por muchos años. Un árbol se puede considerar la extensión lógica de una lista enlazada, y se utiliza principalmente para facilitar la búsqueda de datos clasificados. Los nodos hacen referencia a datos y estos nodos se organizan en una estructura de árbol. Las listas tienen una estructura unidimensional, en tanto que los árboles tienen una estructura bidimensional. La figura 1.1.3.1 ilustra un árbol genérico.

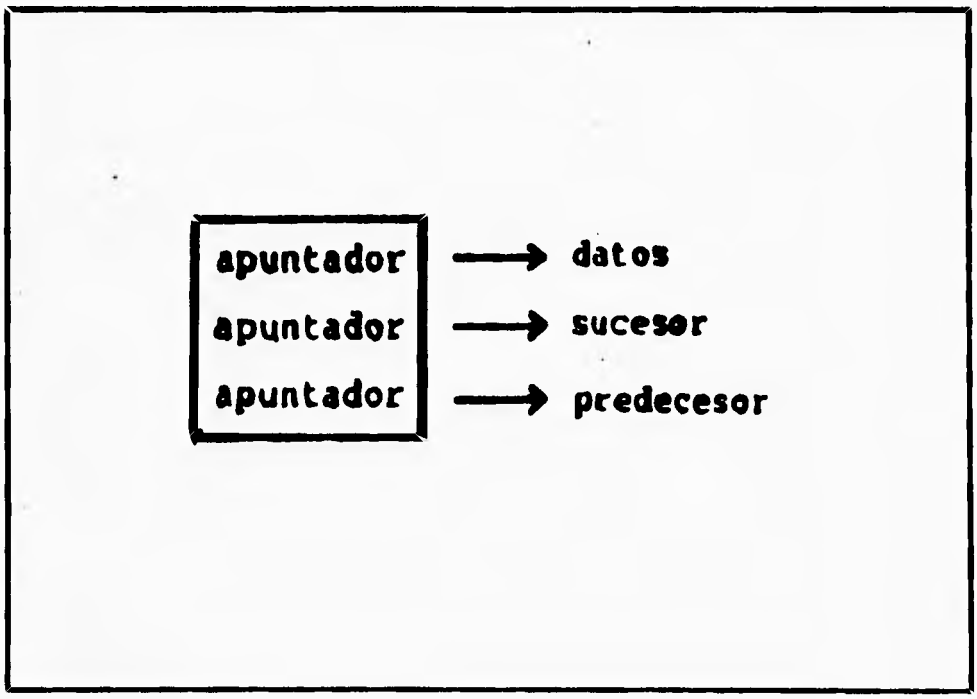


Fig. 1.1.3.1 Estructura de datos de un árbol.

El número de descendientes que puede tener un nodo se conoce como *orden* de un árbol. En un árbol binario, los nodos tienen dos descendientes. En un árbol ternario, los nodos tienen tres descendientes. En árbol *m*-ario, los nodos tienen *m* descendientes. Las diversas formas de árbol varían de forma significativa no sólo en su disposición en la memoria, sino también en las dificultades de la búsqueda, adición y remoción de elementos. Todos los árboles emplean el concepto de comparación para localizar elementos. La comparación requiere una forma de determinar si los elementos son *iguales*, *menores* o *mayores que* cada uno de los otros. En el caso de tipos de datos escalares, como los enteros y los caracteres, no hay problema. Para utilizar los objetos de clase de un árbol se debe disponer de operadores de ordenación apropiados.

Arboles binarios.

Los árboles binarios se cuentan entre las formas más simples de árboles. Cada nodo almacena un elemento (o un apuntador a un elemento), además de dos apuntadores a descendientes: un descendiente izquierdo y uno derecho. Cuando un nodo no tiene descendiente, el apuntador correspondiente es nulo. Un nodo sin descendientes se denomina nodo terminal.

Adiciones de elementos.

Cuando se agrega un elemento a un árbol binario, el elemento se compara con el del nodo raíz. Si el valor es igual al elemento de la raíz, el elemento ya está contenido en el árbol. A menudo los árboles binarios deshabilitan inserciones múltiples del mismo elemento, pero es posible utilizar una variable contadora en cada nodo para contar el número de inserciones de un elemento dato. Si el elemento es menor que el elemento raíz, se elige el apuntador del descendiente derecho. Se repite el proceso de pasar de un nodo a su descendiente hasta que se encuentra un nodo que no tenga descendiente o hasta que se

localice el elemento. Si no se localiza el elemento, este se inserta en el árbol como descendiente del último nodo visitado.

Búsqueda de elementos.

Los procesos de agregar y buscar elementos son mutuamente dependientes. No se pueden buscar elementos a menos que haya un árbol con elementos ya contenidos, y no se pueden agregar elementos a menos que se pueda buscar en un árbol. La búsqueda se realiza pasando de un nodo a uno de sus descendientes, hasta que se encuentre el elemento o el final del árbol. Si el elemento destino es menor que el elemento de un nodo dado, se selecciona el descendiente izquierdo. De lo contrario se utiliza el descendiente derecho.

Eliminaciones de elementos.

En los árboles binarios, la eliminación de elementos puede ser, en general, mucho más complicada que la adición de elementos. Existen tres tipos de nodos y cada uno de ellos afecta un árbol binario de manera diferente.

- Nodos terminales(ningún descendiente). Tan sólo se inicia el apuntador del nodo del progenitor al valor nulo.
- Nodos internos (un descendiente). El apuntador del nodo progenitor se define para que apunte al único descendiente del nodo.
- Nodos internos (dos descendientes). Aquí es donde se complican las cosas. Se tienen que ordenar los descendientes izquierdo y derecho del nodo en el árbol restante. Si los descendientes son nodos terminales, el proceso es simple. Si los descendientes son las raices de subárboles, tiene que combinar los dos subárboles de un subárbol y luego reemplazar el nodo interno borrado con el subárbol combinado.

Arboles-B.

Los árboles-B son árboles balanceados especializados. Los árboles-B no son árboles binarios, y se crearon como una solución a las deficiencias de los árboles desbalanceados. Aunque es posible balancear un árbol binario después de cualquier inserción, en la práctica se incurre en un gasto excesivo si se insertan elementos con frecuencia. Los árboles-B se utilizan cuando se desconoce la cantidad de datos que se deben manejar y se espera sea bastante grande. No es raro encontrarnos con programas comerciales que emplean árboles-b para manejar cientos de miles o incluso millones de objetos. Los árboles-B son árboles *multivía*, lo que significa que cada nodo puede tener no sólo dos, sino muchos descendientes. El número de descendientes de un nodo define el orden de un árbol-B.

Estructura de los árboles-B.

Con los árboles-b, el desempeño está garantizado porque existe la garantía de que la estructura del árbol permanezca balanceada. Lo que hace que los árboles-b sean superiores a los árboles binarios y a otros tipos de árboles es un conjunto de reglas que se emplean para expandir y contraer el árbol. Considere un árbol-b de orden 5. Dicho árbol-b tendrá nodos que contienen cinco claves (elementos dato) y seis apuntadores a descendientes. Los nodos terminales pueden contener 2*(5+1) claves. La figura 1.1.3.2 muestra un pequeño árbol-b de quinto orden que se utiliza para manejar los nombres de un directorio telefónico pequeño.

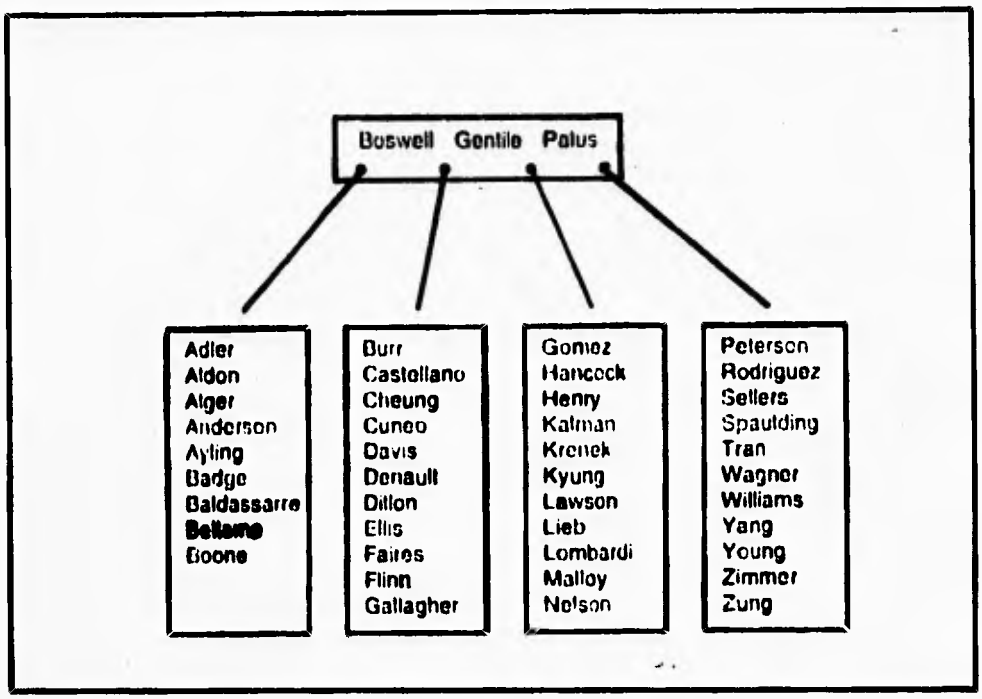


Fig. 1.1.3.2 Un pequeño Arbol-b de quinto orden que contiene cadenas.

Cada nodo terminal del árbol puede contener hasta 12 claves. Cada nodo interno puede contener hasta cinco claves. Si la adición de una clave a un nodo terminal da origen a más de 12 claves, el nodo se desborda y sus teclas se reacomodan. El número mínimo de claves permitido en un nodo terminal está dado por la variable LeafLowWaterMark. Si después de suprimir una clave de un nodo terminal, hay menos de LeafLowWaterMark claves, el nodo terminal se fusiona con otro nodo o se toman claves de un nodo hermano. El número mínimo de claves para un nodo interno está dado por la variable InnerLowWaterMark. La adición de dos entradas al nodo que contiene los nombres (Burr..Gallagher) desencadena un desbordamiento en el nodo terminal, como se ilustra en la figura 1.1.3.3.

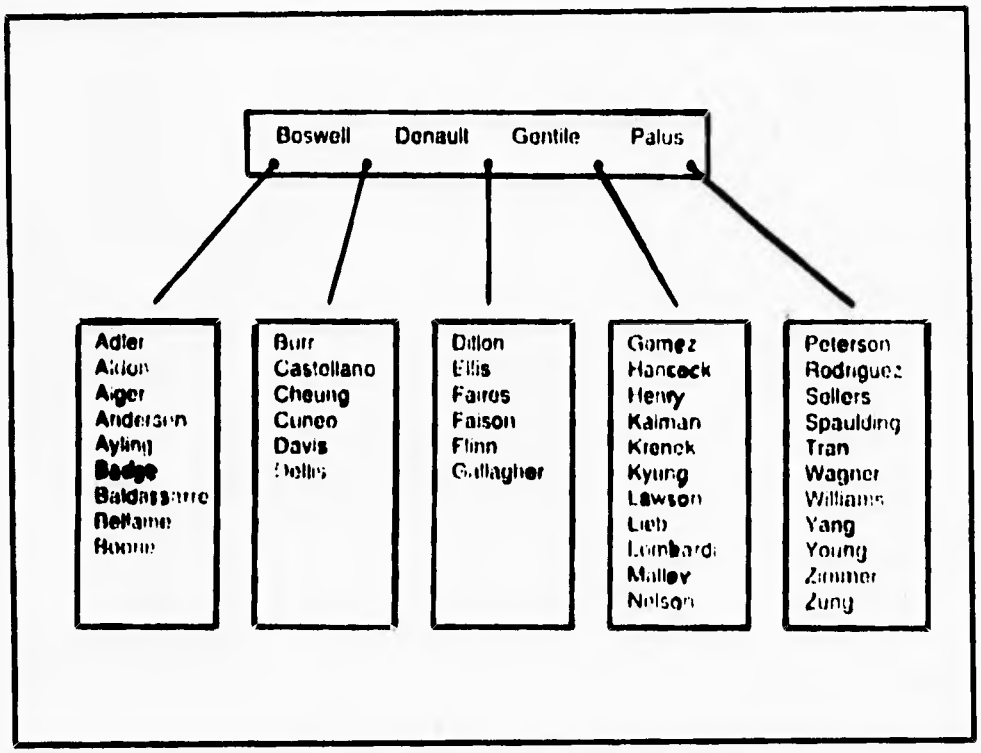


Fig. 1.1.3.3 Adición de elementos a un árbol-b, dando origen al desbordamiento de un nodo.

La adición de los nombres *Dellis* y *Faison* produce un desbordamiento en un nodo terminal. La clave del centro, *Denault*, asciende un nivel y el nodo terminal se divide en dos. Borrar una clave es tan simple como insertarla. Si la supresión de una clave produce un desbordamiento en un nodo, se toma una clave del nodo progenitor. Si también se desborda el nodo progenitor, se repite el proceso.

Desbordamiento en los nodos.

Cuando la adición de una clave a un nodo produce una condición de desbordamiento, las claves del nodo se deben reordenar en el árbol-b.

La distribución de claves en nodos hermanos se conoce como *balance de nodos*. En el caso de árboles-b de alto orden, la diferencia en tiempo de procesamiento entre balancear un nodo y dividirlo, puede ser significativa.

Cuando ocurre un desbordamiento en un nodo y éste se divide en dos nodos, la clave del centro se desplaza al nodo progenitor. Si también se desborda y se divide el nodo progenitor, se repite el proceso. Si se desborda y divide el nodo raíz, se crea un nuevo nodo ráiz, haciendo que aumente un nivel la altura del árbol. Mientras que los árboles binarios crecen de la raíz hacia abajo, los árboles-b crecen de las hojas hacia arriba.

Reglas de los árboles-B de Borland.

Estas son las reglas básicas aplicables a un árbol-b arbitrario de orden *M* de Borland:

* Los nodos no terminales pueden tener a lo sumo *M* claves y *M*+1 descendientes.

* Los nodos terminales pueden tener un máximo de 2*(*M*+1) claves. Por definición, los nodos terminales no tienen descendientes.

* Todos los nodos (menos el raíz) deben tener un número mínimo de claves, determinado por las variables InnerLowWaterMark y LeafLowWaterMark.

* Todos los nodos terminales están a la misma distancia de la raíz.

* El orden de un árbol-b es mayor que o igual a 3.

* Sólo los objetos derivados de la clase Sortable se pueden manejar como claves.

Cada nodo interno de un árbol-b de Borland de orden *M* tiene cuando mucho *M* claves($K_1..K_m$) y *M*+1 apuntadores ($P_0..P_m$) a descendientes. El apuntador P_0 apunta al descendiente que tiene claves todas menores que K_0. El apuntador P_1 apunta al descendiente que tiene claves que se encuentran entre K_0 y K_1. El apuntador P_m apunta al descendiente que tiene claves que son mayores que K_m. La figura 1.1.3.4 es una representación gráfica en la que se utiliza un árbol-b de orden 3, cuyas claves son números enteros.

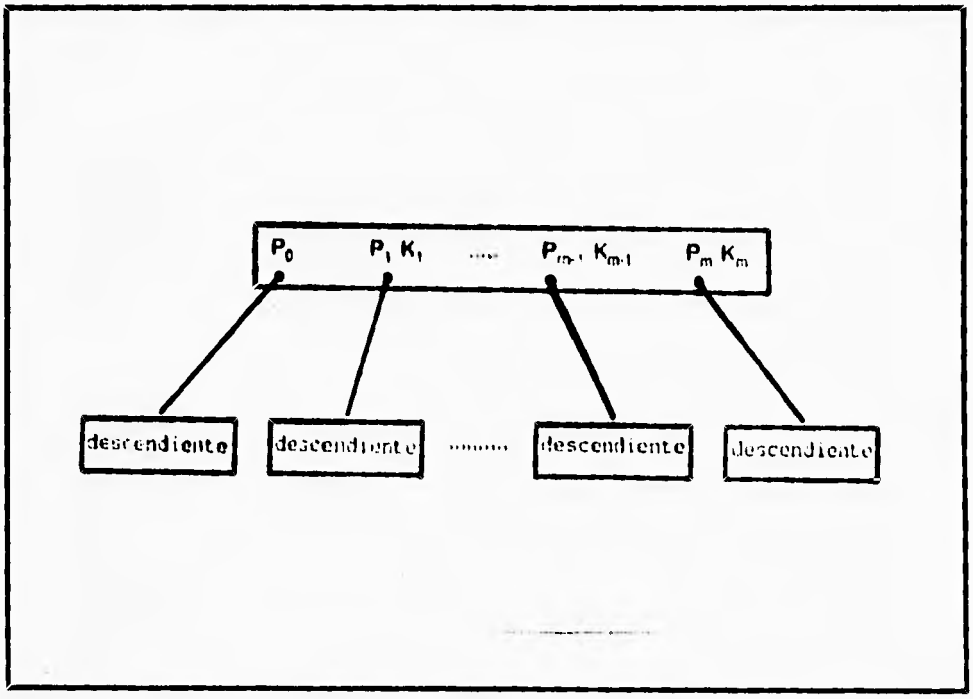


Fig. 1.1.3.4 Claves y apuntadores que utilizan un nodo interno de un árbol-b de Borland.

Cada nodo tiene un apuntador más que el número de claves. La figura 1.1.3.5 es un pequeño árbol-b de orden 3 que tiene claves que son números enteros.

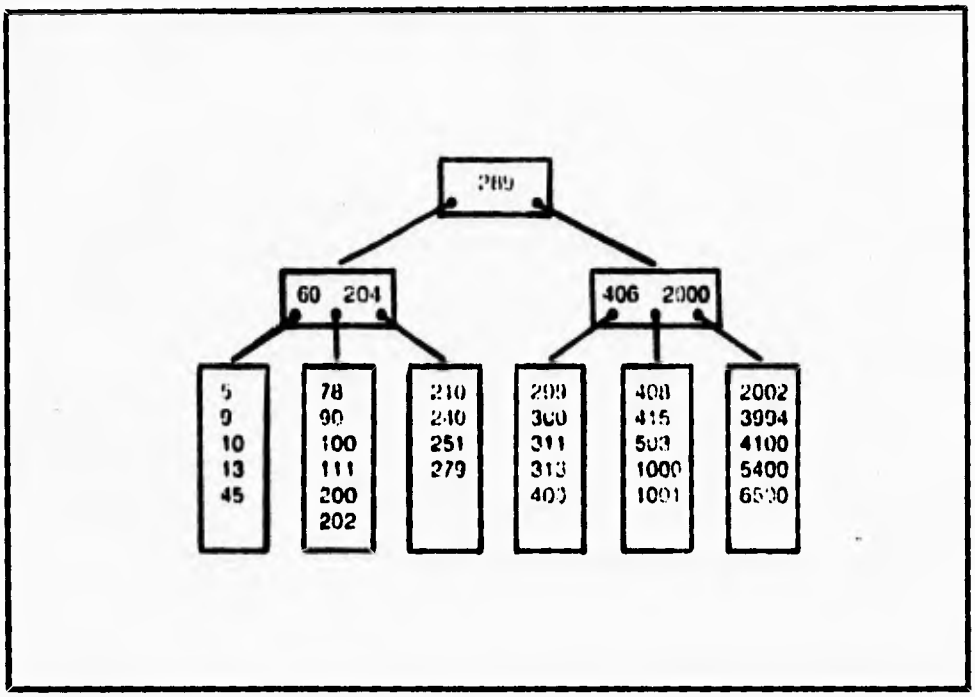


Fig. 1.1.3.5 Ejemplo de un árbol-b de orden 3 de Borland.

Los árboles-b de Borland utilizan un objeto clase llamado Node para manejar una clave y un apuntador en cada nodo. En el caso de nodos internos, la clave K_0 queda sin usarse. En el caso de nodos terminales, los apuntadores se usan para alojar claves. Se deduce que los nodos internos pueden alojar *M* claves, y los nodos terminales pueden alojar 2*(*M*+1) claves.

Aunque los árboles-b fueron diseñados originalmente para estructuras de datos que se pudieran utilizar con estructuras de datos de archivos, los árboles-b de Borland se implantan sólo para estructuras de datos de memoria primaria (RAM).

Los árboles-b no son árboles binarios.

Un árbol binario se compone de nodos, donde cada uno tiene dos posibles sucesores. La inserción de datos en un árbol binario da origen a una estructura de árbol que es función del orden de las inserciones de datos. Si se insertan datos en orden aleatorio, tiende a desarrollarse un árbol balanceado. Si se insertan datos en orden clasificado, se crea un árbol degenerado, que se parece más a una lista enlazada que a un árbol.

Un árbol-b es muy diferente. Los árboles-b son mucho más eficientes cuando se maneja una cantidad de datos grande, y son árboles que se autobalancean. Los árboles-b se utilizan por lo general con órdenes mayores que 3. Los árboles-b con órdenes por arriba de 100 no son raros en bases de datos grandes.

LA FAMILIA DEL ARBOL.

Vamos a considerar un número de temas relacionados para utilizar clases jerarquicas.

- Utilizar herencias: Llamando un metodo de ancestro que hemos sobrepuesto.
- Utilizando superclases abstractas - TObject.
- El alcance y comunicación entre objetos.
- Outlets : Una técnica de corolario para lograr la comunicación.
- Insertando superclases para propagar capacidades.
- Los objetos compuestos.
- La herencia múltiple.

La Estructura de Objetos Jerárquicos

Algunos lenguajes no permiten la herencia multiple, esto es, cada clase tiene uno y solamente un ancestro inmediato, aunque puede tener muchos ancestros sobre la jerarquía.

Debido a herencia única, un objeto jerarquico tiene una estructura distinta:

- Una jerarquía puede contener cualquier número de clases.
- La clase más alta (raíz, o base) no tiene ancestros.
- Cualquier clase en la jerarquía puede tener cualquier número de descendientes.
- Las clases descendientes heredan todas las variables y todos los métodos de todos sus ancestros (excepto aquellos métodos que son sobrepuestos).

Podemos tener varias o muchas jerarquías en una sola aplicación o en una clase de biblioteca, pero todas las jerarquías comparten la misma estructura. Una jerarquía de objeto tiene la forma de una gráfica aciclica dirigida, o DAG, un árbol en el cuál cada nado tiene solamente un padre pero puede tener muchos hijos en cada nodo. La figura 1.1.3.6 muestra tal estructura, con múltiples hijos en cada nodo.

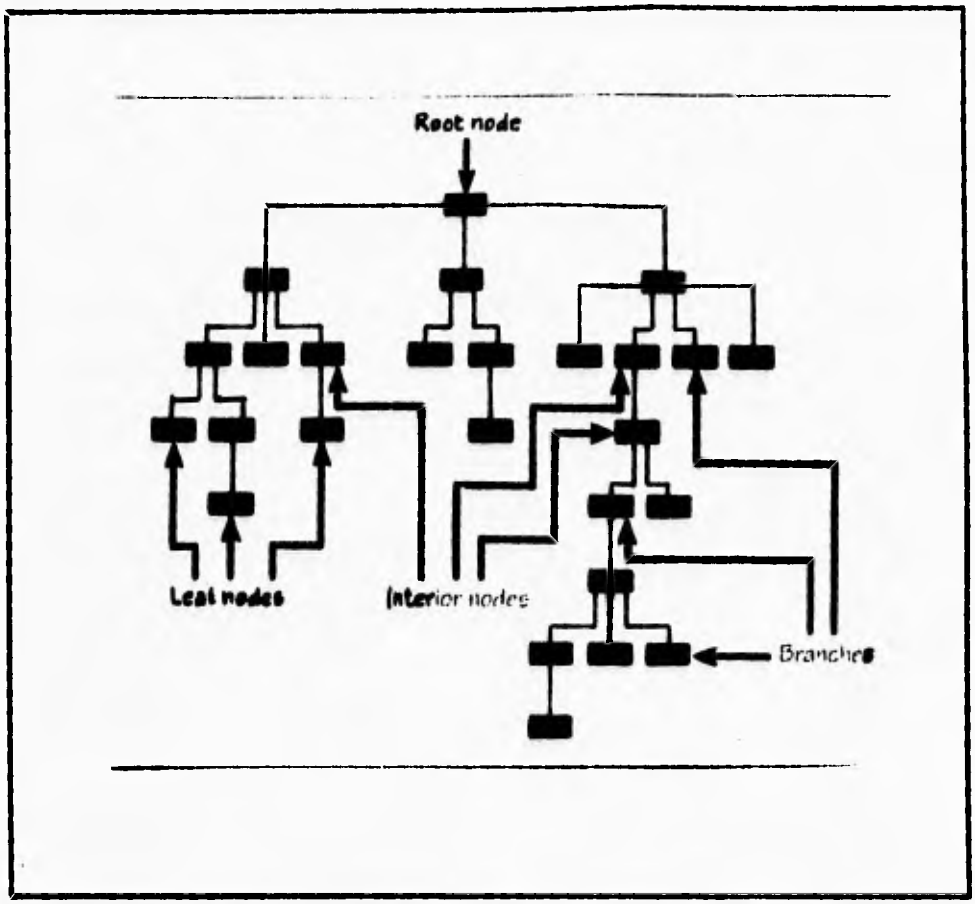


Fig. 1.1.3.6 Estructura de Objeto Jerárquico.

Si se tiene familiaridad con el sistema de archivo jerárquico de Macintosh (HFS), ó ha visto ya tal estructura. HFS es una estructura de árbol jerarquica de directorios y archivos.Un directorio HFS puede contener muchos subdirectorios pero pueden tener solamente un directorio padre. Y un archivo o subdirectorio puede ser solamente un directorio.

Otro concepto de HFS podria ayudar para entender la estructura de objeto jerarquico: Un path en el HFS desciende a tráves del archivo árbol de directorio a

subdirectorio de subdirectorio a subdirectorio.. y con el tiempo a archivos. Cualquier directorio, subdirectorio, o archivo puede ser alcanzado por uno y solamente un path del directorio principal. Dos subdirectorios o archivos podrían o no tener el mismo path de la raiz. La figura 1.1.3.7 muestra dos paths de la misma clase de raiz. Este concepto tiene una paciencia en lo que ocurre cuando multiples programadores trabajan en la misma jerarquía de objeto al mismo tiempo.

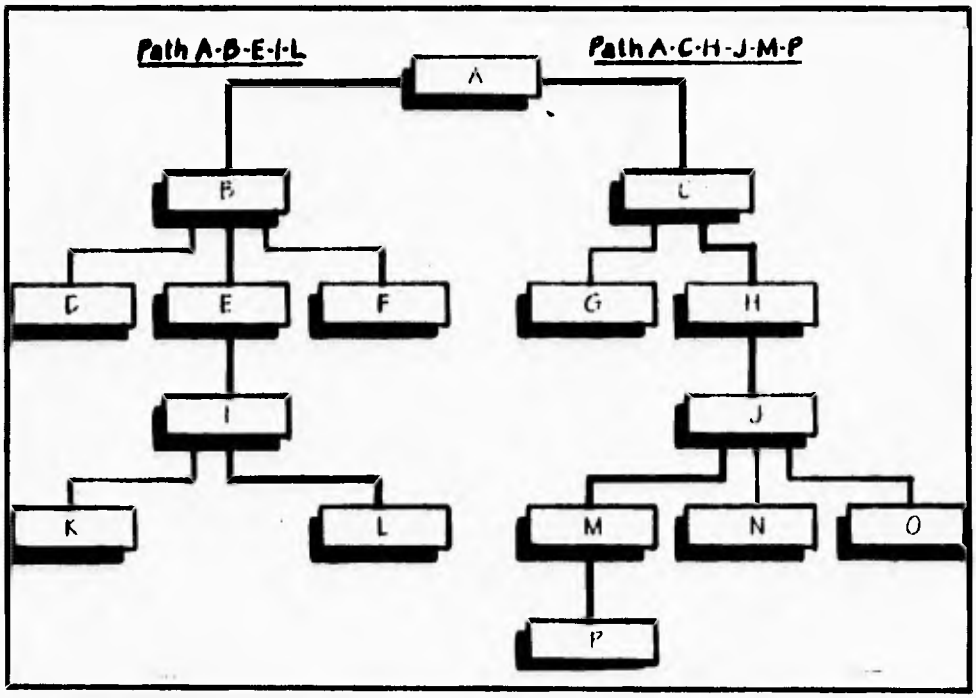


Fig. 1.1.3.7 Paths en un objeto jerarquico.

Vamos a echarle un vistazo a las clases A, B, C, D, y E, como se muestra figura 1.1.3.8. Clase A es un antecesor de B, C, D, y E, esto es,clase A es un path de la raíz a cada una de estas clases. Clase B es un ancestro de D, y clase C es un ancestro de E. Clase B,C,

D, y E hereda de A. D hereda de B, y E hereda de C. Pero clase D no hereda de clase C- C no es un path de la raiz a D.

Asi mismo, clase E no hereda de B.

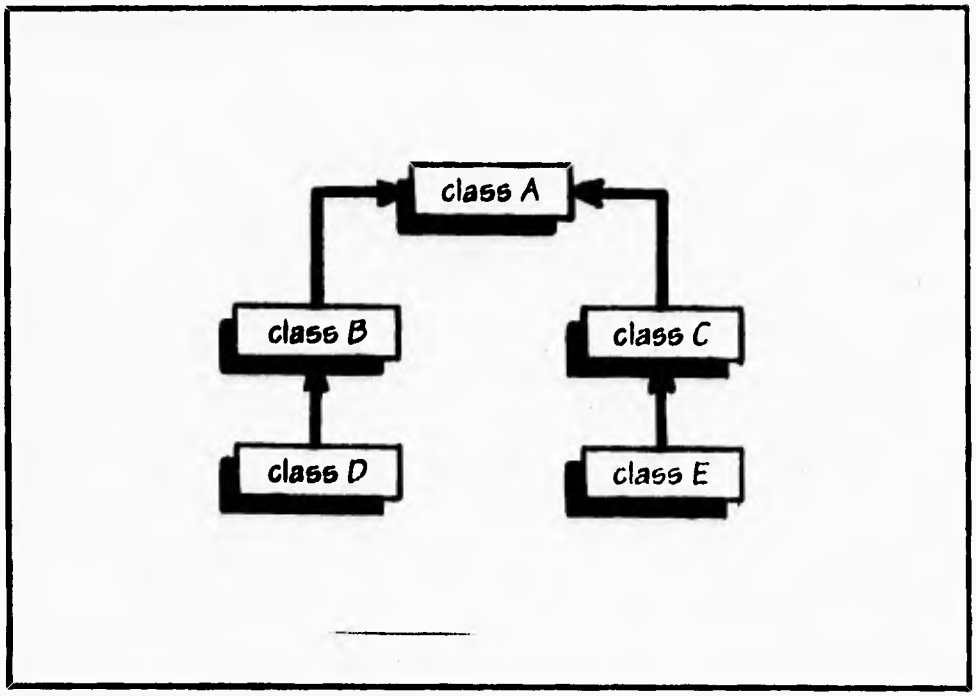


Fig. 1.1.3.8 Una simple clase de objeto jerarquico.

Toda la jerarquia descendida de uno ó más ancestros comunes. Pero cuando el árbol se ramifica más y más, algunas clases se hacen primos de otros y no comparten todos la misma herencia.

Por supuesto, muchas jerarquías son más simples, y muchas son hasta lineales: Una linea de clases descendiente podria venir directamente desde una clase de raiz en la parte superior- el árbol no tendria rama. Como ejemplo de una jerarquia lineal, se muestra en la figura 1.1.3.9.

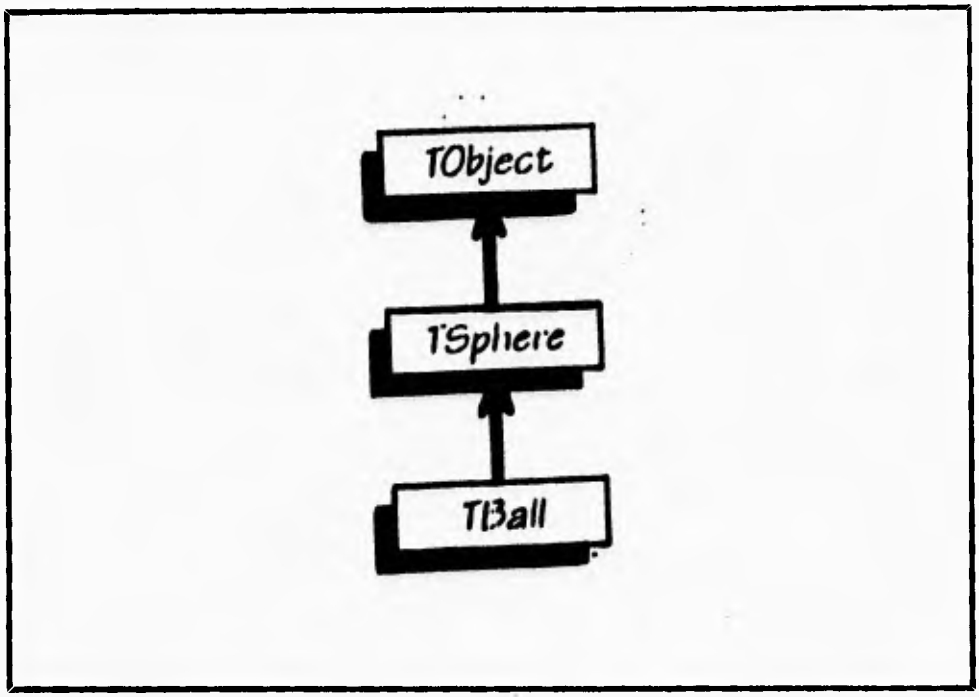


Fig. 1.1.3.9 Una jerarquía no ramificada. Cada clase tiene sólo una subclase.

Extendiendo Objetos Jerárquicos.

En una jerarquía de clase, podemos hacer cambios:

- **Podemos hacer subclases de la clase (es) más baja- las " hojas " del árbol.**
- Podemos tener subclases de clases " internas", comenzando nuevas ramas.
- Podemos insertar una nueva clase en alguna parte del interior del árbol, posiblemente hasta sobre los clase de raiz actual.
- Podemos reescribir una clase en alguna parte del interior del árbol un "nodo interno".

La figura 1.1.3.10 muestra estos puntos de cambio.

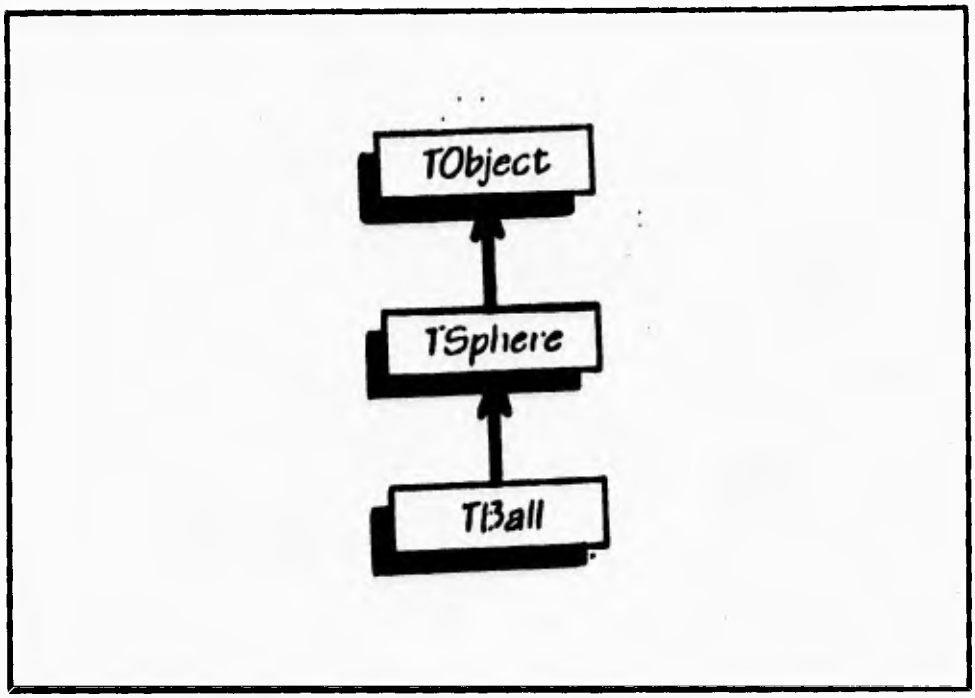


Fig. 1.1.3.9 Una jerarquía no ramificada. Cada clase tiene sólo una subclase.

Extendiendo Objetos Jerárquicos.

En una jerarquía de clase, podemos hacer cambios:

- **Podemos hacer subclases de la clase (es) más baja- las " hojas " del árbol.**
- Podemos tener subclases de clases " internas", comenzando nuevas ramas.
- Podemos insertar una nueva clase en alguna parte del interior del árbol, posiblemente hasta sobre los clase de raíz actual.
- Podemos reescribir una clase en alguna parte del interior del árbol un "nodo interno".

La figura 1.1.3.10 muestra estos puntos de cambio.

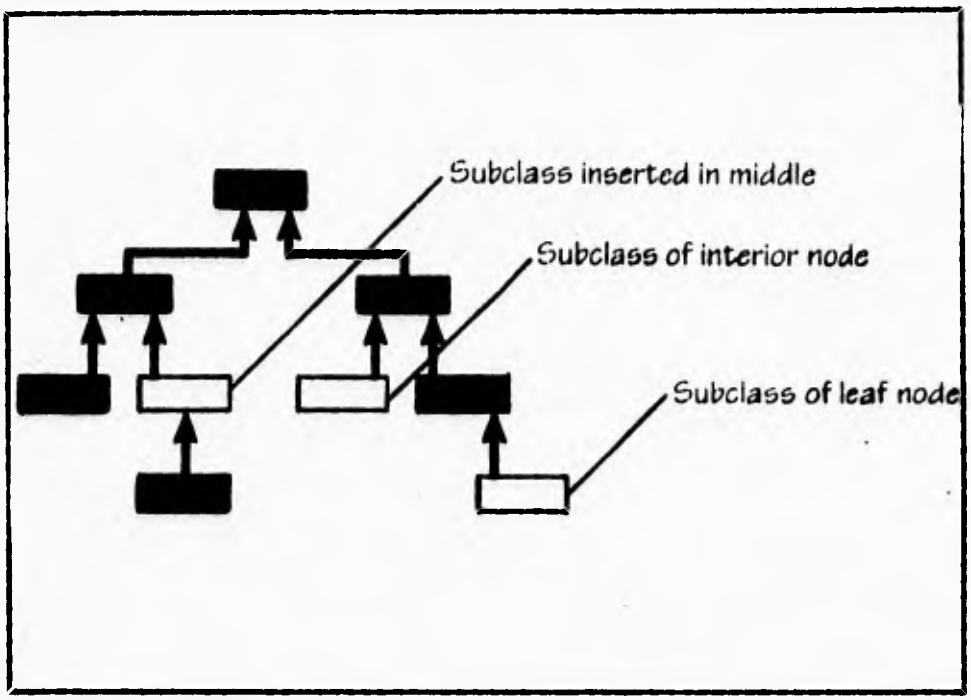


Fig. 1.1.3.10 Puntos de cambio en una jerarquía-subclasificando un nodo izquierdo y un nodo interior e insertando una nueva subclase en el interior del árbol.

1.1.4 DESARROLLO DE SOFTWARE ORIENTADO A OBJETOS.

La velocidad a la que avanza la tecnología de hardware de computadoras es sorprendente; año con año se logran avances que conducen a la construcción de computadoras más veloces, más compactas y baratas. Sin embargo, en el aspecto de software no parece haber un desarrollo similar. Mientras que los costos de hardware han disminuido continuamente, los de software han hecho lo contrario. La construcción de software no es una tarea fácil y en muchas ocasiones los proyectos de programación sobregiran los presupuestos de tiempo y dinero.

El problema del mantenimiento de software.

La construcción de software ha recibido la atención de los expertos desde hace mucho tiempo; en la década de los 70's se consiguieron avances significativos hacia el desarrollo de metodologías para construir programas en forma sistemática y a bajo costo. Como resultado de esos esfuerzos surgieron técnicas que, como el diseño estructurado y el desarrollo descendente (top-down), durante mucho tiempo han sido las herramientas utilizadas por los programadores para construir software, Aunque dichas técnicas han sido empleadas durante mucho tiempo en proyectos realmente complejos, la mayoría de los ingenieros de software coinciden en afirmar que sufren de tres grandes deficiencias:

- los productos que resultan al emplear estas técnicas son poco flexibles.

- los programadores que los usan tienden a concentrarse en el diseño y la implementación inicial del sistema, sin tomar en cuenta su vida posterior.

- no alientan al programador a aprovechar el trabajo de proyectos anteriores.

La desventaja de utilizar una metodología que se concentra en el diseño inicial del sistema se hace evidente si se toma en cuenta que la vida útil de un producto de software puede ser cinco ó seis veces más grande que el lapso en que se desarrolla; por ejemplo, un sistema que se desarrolla en uno ó dos años puede mantenerse trabajando durante un periodo que va de cinco a quince años. Los gastos que se hacen durante este último período (gastos de mantenimiento) representan alrededor del 70% del costo total del sistema. La figura 1.1.4.1 ilustra la forma en que se distribuyen estos gastos.

La gráfica muestra que cerca del 60% de los costos de mantenimiento de un sistema (alrededor del 45% del costo total) se tienen que hacer por cambios en las especificaciones del usuario ó en los formatos de los datos. Es por ello que la extensibilidad (la facilidad con que se modifica un sistema para que realice nuevas funciones) debe ser uno de los objetivos primarios de la etapa de diseño.

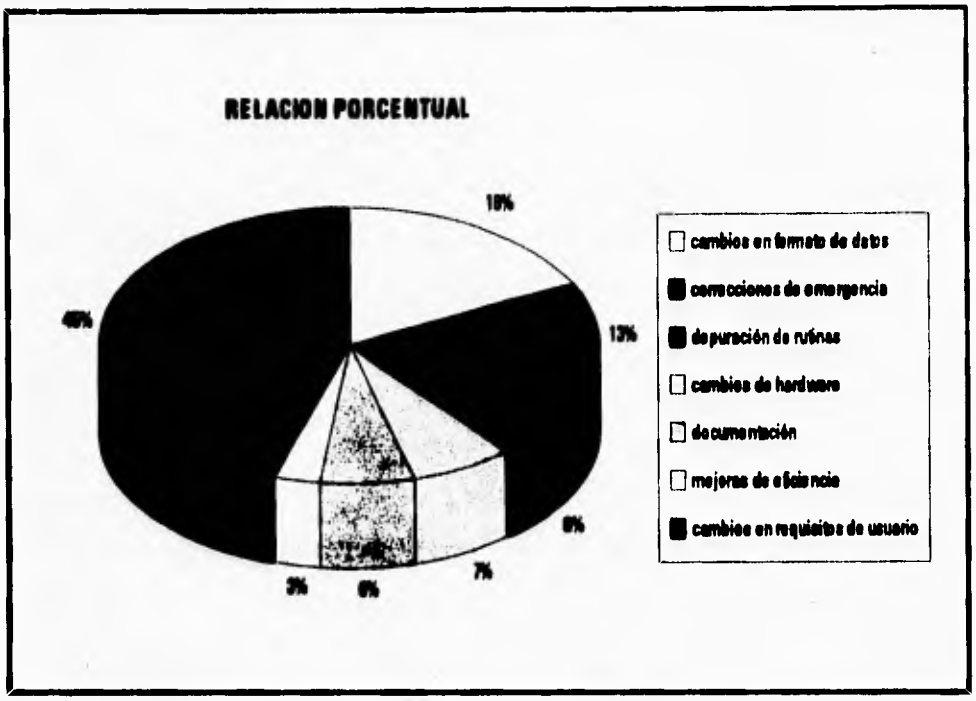


Fig. 1.1.4.1. Forma en que se distribuyen los gastos de mantenimiento de software.

La fase de mantenimiento es tan importante que qualquier método de diseño debe tener como objetivo principal producir sistemas que faciliten su propio mantenimiento. Pero, ¿cómo alcanzar este objetivo?. Los expertos recomiendan una serie de actividades y heurísticas que pueden servir como guía durante el desarrollo del sistema.

La reutilización de código es otro de los factores que no se toma en cuenta en los métodos de diseño tradicionales. Cualquier programador que desarrolla un sistema nuevo debe escribir una buena cantidad de código que ha escrito anteriormente una y otra vez. Las rutinas de búsqueda, ordenamiento, manejo de menús y despliegue de ventanas, entre otras, se repiten continuamente en proyectos diferentes, Los métodos de desarrollo de software deben alentar al programador a utilizar el código escrito previamente por él mismo o por otros programadores.

Modularidad.

La programación orientada a objetos es un método de diseño que tiene como objetivo establecer técnicas de desarrollo de software para producir sistemas modulares.

La modularidad es una de las herramientas de diseño más poderosas para facilitar el desarrollo y mantenimiento de sistemas de software. La modularidad permite definir un sistema complejo en términos de unidades más pequeñas y manejables; cada una de esas unidades (ó modulos) se encarga de manejar un aspecto local de todo el sistema, interactuando con otros módulos para cumplir con el objetivo global.

La mayoría de los lenguajes actuales alientan el uso de la modularidad: en lenguajes estructurados como C ó Pascal, la modularidad se basa en el concepto de función (también llamada procedimiento o subrutina); en lenguajes más evolucionados, como Ada ó Modula-2, los modulos corresponden a conjuntos de funciones relacionados con las estructuras de datos que manipulan esas funciones (paquetes, en terminología de Ada).

Sin embargo, para ser realmente útil, las propiedades de un módulo deberían ser las siguientes:

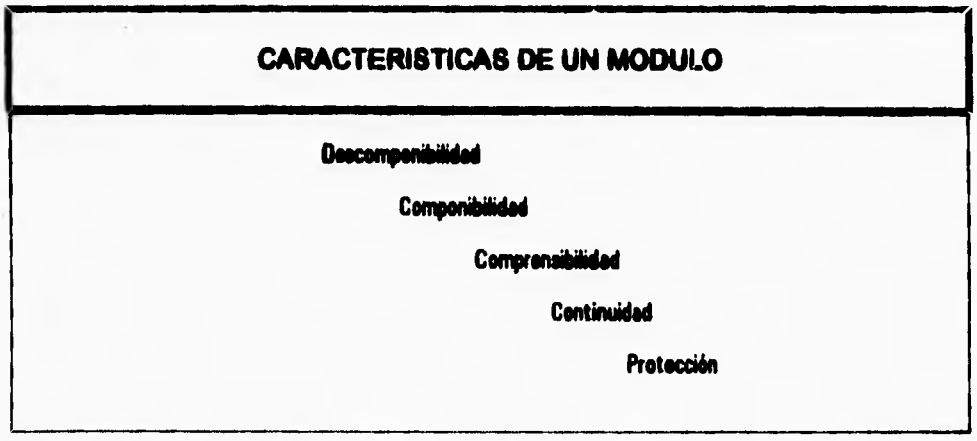


Fig. 1.1.4.2 Propiedades de la Modularidad.

a) Descomponibilidad: Un método de diseño modular debe permitir descomponer un problema de diseño en subproblemas más pequeños que pueden resolverse en forma independiente.

b) Componibilidad: Una vez que se cuenta con un conjunto de módulos que realizan una función especifica, se debe alentar al programador a usar esos módulos para construir nuevos programas.

c) Comprensibilidad: El lector de un programa o una librería debe ser capaz de entender el funcionamiento de cada módulo sin necesidad de consultar el texto de otros módulos.

d) Continuidad: Un cambio pequeño en las especificaciones de un programa debe causar cambios en un sólo módulo ó en un conjunto pequeño de ellos.

e) Protección: Un error de ejecución en el funcionameinto de un módulo no debe expandirse hacia los demás módulos.

Estas cinco propiedades pueden alentarse con las siguientes estrategias:

a) Un módulo debe corresponder con una unidad sintáctica del lenguaje (una subrutina, un paquete, una clase); esta unidad debe poder ser compilada por separado, tal vez para ser almacenada en una librería (con esto se mejoran la componibilidad y comprensibilidad del sistema).

b) Los módulos deben tener pocas interfaces (medios de comunicación con otros módulos), y éstas deben ser pequeñas. Un número pequeño de interfaces aumenta la independencia de los módulos, lo cual hace más fácil el proceso de componer nuevos sistemas a partir de módulos prefabricados, ayuda a evitar que los errores en un módulo se propaguen por todo el sistema y hace que cada módulo de un sistema sea más comprensible.

c) Cada módulo debe ocultar su implementación y algoritmos internos al resto del sistema (principio de ocultamiento de información). No se debe permitir que un módulo modifique los elementos internos de otros módulos; sino que la comunicación entre ellos

debe realizarse mediante interfaces explícitas y bien definidas (esto favorece la comprensibilidad y la protección modular).

Un sistema orientado a objetos se puede entender fácilmente, por lo que su desarrollo, depuración y mantenimiento se facilitan en gran medida.

SOFTWARE ORIENTADO A OBJETOS.

Las técnicas en las que se basa la programación orientada a objetos (ocultamiento de información, abstracción de datos, manejo automático de memoria, polimorfismo) eran conocidas y utilizadas por los ingenieros de software desde hace muchos años; lenguajes como Ada ó Modula-2 alientan a los programadores a usar algunas de esas técnicas.

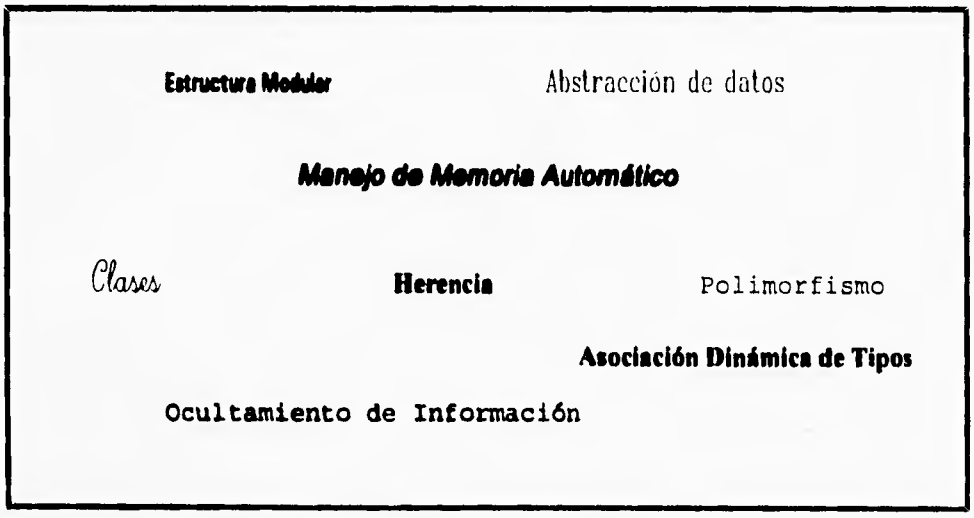


Fig. 1.1.4.3 Facilidades de la Programación Orientada a Objetos.

Entre los lenguajes orientados a objetos más populares, se encuentran Simula67, un lenguaje de simulación con facilidades para manipular eventos discretos; Smalltalk, que se ha usado principalmente para desarrollar interfaces de usuario gráficas y Eiffel, que se ha empleado en áreas diversas. Los lenguajes orientados a objetos no habían tenido una aceptación amplia entre la comunidad de programadores. Características como el manejo de memoria automático y la asociación dinámica de tipos imponen sobrecargas demasiado grandes a la ejecución de programas escritos en dichos lenguajes. Recientemente han surgido dos estrategias para disminuir esa sobrecarga: una de ellas consiste en utilizar lenguajes orientados a objetos para desarrollar los componentes de más alto nivel de un sistema y lenguajes funcionales para escribir las partes de bajo nivel críticas para la ejecución (una forma común de este tipo de combinaciones es utilizar Smalltalk y C). La otra estratégia consiste en desarrollar nuevos lenguajes que nos proporcionen todas las facilidades de la programación orientada a objetos, pero que no impongan sobrecargas de ejecución demasiado altas.

Naturalmente, la programación orientada a objetos requiere el uso de lenguajes de programación idóneos. En la Programación Orientada a Objetos (POO), existen dos tipos diferentes de lenguajes: *puros e híbridos*.

Lenguajes puros.- Son aquellos diseñados en base a las propiedades específicas de la programación orientada a objetos, y tienen unas construcciones léxicas y lógicas muy diferentes a las encontradas en los lenguajes estructurados tradicionales. Su primer representante fue Simula, un lenguaje de simulación basado en Algol; hoy día, sin embargo, su representante más genuino es Smalltalk, y la versión Smalltalk/V la más difundida. Es el lenguaje que, al decir de los especialistas en POO, debería ser elegido como primer lenguaje al comenzar a estudiar POO.

Lenguajes híbridos.- Son lenguajes que añaden, a las propiedades tradicionales estructuradas, características orientadas a objetos. Hoy día son los más difundidos y, entre ellos, podemos citar: C++, Turbo Pascal (5.5, 6.0 y 7.0), Objective Pascal y Objective-C.

El debate sobre cuál lenguaje utilizar es abierto y polémico. Los puristas sugieren lenguajes puros que fuerzan al programador a utilizar técnicas orientadas a objetos; los lenguajes híbridos permiten que otros paradigmas, como la programación estructurada, puedan ser utilizados. La decisión, muchas veces, dependerá de un conjuntos de cirscunstancias impredecibles en el principio del diseño. Hoy día, los lenguajes híbridos han proliferado, debido a la gran comunidad de programadores en Pascal y C que, lógicamente tienen, a priori, asegurada una migración más fácil a Turbo Pascal 5.5, 6.0 ó 7.0 y a C++.

SMALLTALK

Es el primer lenguaje orientado a objetos. Descendiente de Simula-67, fue desarrollado en Xerox's Palo Alto Research Center (PARC). Smalltalk trata a todo como un objeto, y representa el paradigma de la programación orientada a objetos: es el lenguaje puro POO, por excelencia.

El entorno Smalltalk es orientado a objetos con ventanas, menús desplegables y emergentes, tratamiento de menús, etc. Presenta las características fundamentales de POO, y cada objeto en el sistema Smalltalk hereda del objeto raíz. Al igual que otros POO, Smalltalk diferencia entre objetos y clases. Un objeto es una instancia o ejemplo de una clase. Todas las variables instancia de una clase Smalltalk son, por defecto, privadas a ese objeto.

Smalltalk es un lenguaje interpretado que lo hace muy flexible; pero lógicamente, limita sus posibilidades comerciales. Un programa Smalltalk debe ejecutarse desde su mismo entorno, y eso limita su transportabilidad. Esto no siempre es una desventaja. El entorno está disponible en muchas plataformas; entre ellas, DOS, OS/2 y Windows.

TURBO/QUICK/OBJECTIVE PASCAL

En 1989, Borland International y Microsoft, simultáneamente, lanzarón versiones orientadas o objetos de Pascal, Borland utilizó la filosofía C++, y Microsoft la filosofía Smalltalk, en sus versiones. Sin embargo, Borland ha progresado mucho y su última versión 7.0 incluye además, una magnífica biblioteca de clases llamada Turbo Vision, mejora de las versiones anteriores 5.5 y 6.0; por el contrario, Microsoft sigue estancada en su versión primitiva.

Estos lenguajes soportan las características mínimas orientadas a objetos, como son la creación de clases, herencia (sólo simple) y polimorfismo. La versión de Turbo Pascal es más robusta que Quick Pascal, aunque ésta es más fácil de aprender.

C++

El lengusje de programación fué diseñado e implementado por Bjarne Stroustrup de los laboratorios Bell de la AT&T como sucesor de C. Tomando ideas de los lenguajes de programación Simula 67 y Algol 68, C++ conserva la compatibilidad con programas existentes en C y la eficiencia de C. La figura 1.1.4.4 ilustra la herencia de C++.

C++ además adiciona nuevas capacidades poderosas, haciendolo compatible para un amplio rango de aplicaciones incluyendo inteligencia artificial. C++ se tiene que tomar seriamente para el desarrollo de software porque está intimamente relacionado con C, y por el potencial que tiene para graficas e interfaces, para programación de sistemas, y para el soporte a gran escala en el desarrollo de software.

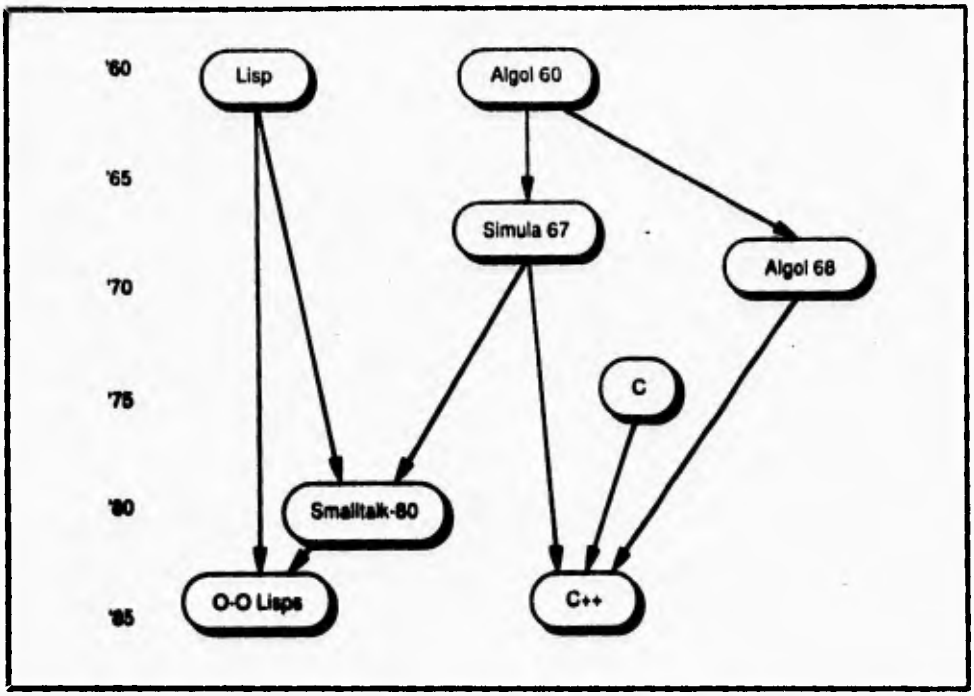


Fig. 1.1.4.4. Lenguajes que contribuyerón en la formación de C++.

El lenguaje de programación C++ permite además la abstracción de datos, en la tabla 1.1.4.1 se muestra una relación de las características de lenguajes de programación.

En ella se presenta solo Smalltalk-80 and C++ como los lenguajes en los cuales se puede realizar abstracción de datos y programación orientada a objetos.

C++ es seguramente el lenguaje híbrido más popular, existen versiones para todas las plataformas: DOS, OS/2, Windows, UNIX, XENIX, etc., y muchos distribuidores comercializan productos y compiladores C++, como es el caso de AT&T, Borland, Microsoft, Zorterch, etc.

Lenguaje POO	Programación Estructurada	Programación Modular	Abstracción de Datos	
C	Sí	Sí	No	No
Pascal	Sí	No	No	No
Modula-2	Sí	Sí	No	No
Ada	Sí	Sí	Sí	No
Smalltalk-80	No	No	Sí	Sí
C++	Si	No	Si	Sí

Tabla 1.1.4.1. C++ comparado sus características principales con otros lenguajes.

El lenguaje C++ se utiliza actualmente en el desarrollo de manejadores de bases de datos, sistemas operativos, compiladores, sistemas de comunicación, productos de CASE, redes y robótica.

Su gran difusión actual y la que se prevé en la década de los noventa se deberá esencialmente al proceso de estandarización a que se encuentra sometido, la versión 2.0 ya adoptada por el comité ANSI respectivo, es seguida, prácticamente, por todos los fabricantes. La versión 2.1 está en proceso de estandarización, y ya existen también actualizaciones de diversos fabricantes.

TURBO C++ y BORLAND C++

El compilador Turbo C++ es una implementación completa de AT&T C++ versión 2.0. Con Turbo C++, se obtienen las mismas características de C, así como de C++. Su característica fundamental es el entorno integrado de desarrollo EID (Integrated Development

Environment, IDE). En la actualidad, las versiones más populares son 1.0 y 2nd edición, aunque recientemente ha aparecido la nueva versión 3.0, que corre en entornos DOS y Windows.

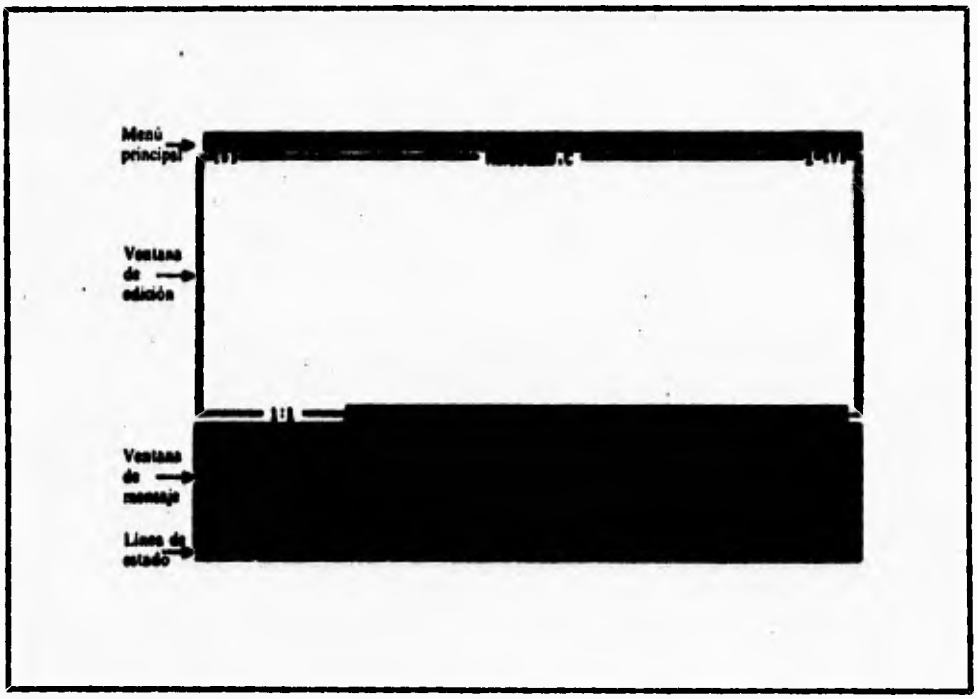


Fig.1.1.4.5. Entorno integrado Turbo C++.

Borland C++ es un paquete profesional. Añade las características básicas de los compiladores C y C++, un depurador incorporado, un profiler y un ensamblador. Otras características sobresalientes son: soporte Windows 3.0/3.1 y módulos DLL/Dynamic Link Library (Bilblioteca dinámica de enlaces). Además Borland C++ contiene el Whitewater Group's Resource Toolkit, que permite crear y mantener -programar- recursos Windows (iconos, menús, cuadros de diálogo, etc.). A finales de 1991, Borland ha presentado la

versión 3.0, que soporta las versiones 2.0 y 2.1 de AT&T C++ y, en la primavera de 1992, la versión 3.1, que además soporta Windows 3.1.

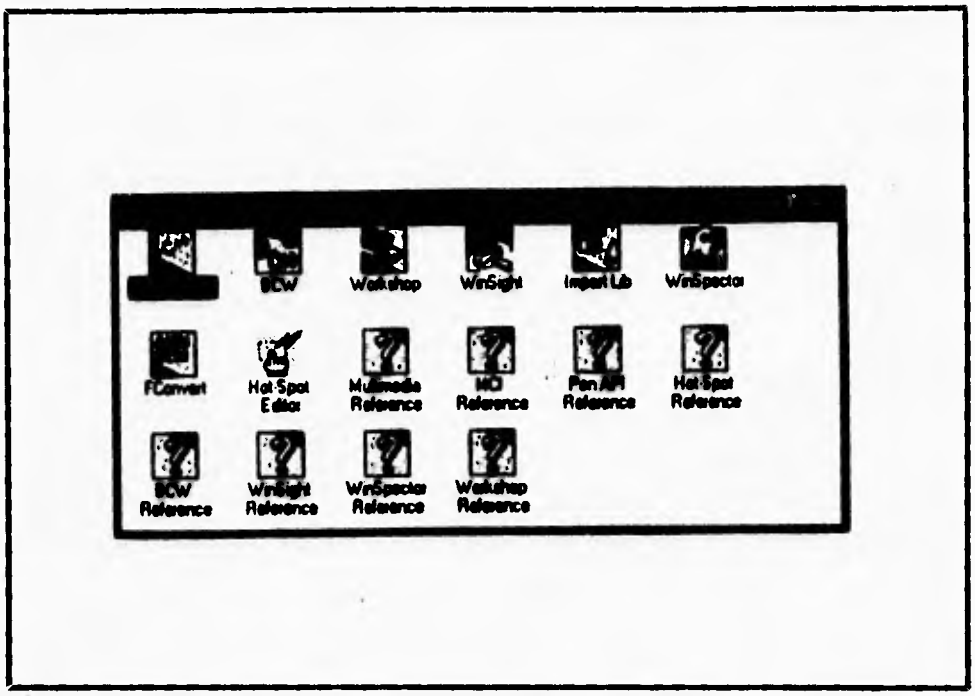


Fig.1.1.4.6. Administrador de programas de Windows.

Actualmente el compilador C++ 4.0 de Borland se nos presenta como un entorno integrado de programación para Windows que incluye todas las herramientas tradicionales para el desarrollo de una aplicación en C. Además, dispone de unas utilerias (denominadas "expertos") que facilitan de forma muy particular el desarrollo de aplicaciones bajo Windows. Por añadidura, el paquete de Borland dispone del gestor de proyectos Proyect Manager, que nos permite mantener desde una ventana el control del desarrollo de toda la aplicación.

ZORTERCH C++

Zorterch C++ es un compilador, desarrollado por la casa Zorterch, que combina las propiedades del compilador estándar C++ y un entorno integrado. Hasta la aparición de los compiladores de Borland, ha sido la estrella de la programación orientada a objetos. En la actualidad, se comercializa la versión 3.0, que soporta los entornos DOS, Windows y OS/2. Presenta dos nuevas opciones: una como entorno de desarrollo profesional y, otra, para cálculos científico-técnicos, lo que aumenta su potencia y versatilidad comparada con otros compiladores C++.

SYMANTEC C++

El compilador Symantec C++ 6.0 es la que llamaríamos, en inglés, un *outsider*, es decir, un aspirante inesperado que intenta hacerse sitio entre los dos gigantes de la programación visual en C.

Symantec C++ 6.9 es heredero directo del Zorterch. Lo primero que ha innovado es el diseño de su propio entorno integrado. De IDE ha cambiado a IDDE (*Integrated Develpment and Debugging Environment*, entorno integrado de desarrollo y depuración).

MICROSOFT C/C++ 7.0

El programa Microsoft C/C++ 7.0 (MSC) contiene un compilador C que sigue la norma ANSI. MSC necesita una PC basada en procesador 386 ó 486 con un mínimo de 4MB de RAM, un disco duro de al menos 10 MB libres, versión MS-DOS 3.3 o superior y monitores CGA, EGA, VGA o las nuevas SuperVGA (SVGA). Si se desea instalar todo el compilador C, más el kit de desarrollo SDK de Windows 3.1, se requiere entonces 55MB de disco duro.

MSC 7.0 ha sido diseñado para soportar aplicaciones DOS y Windows. El paquete contiene todo lo necesario para construir aplicaciones basadas en Windows: archivos de cabecera, bibliotecas, herramientas específicas, etc.

Una de las propiedades más notables de MSC 7.0 es la biblioteca de clases MFC (Microsoft Foundation Classes) que ofrece soporte completo para objetos Windows tales como ventanas, diálogos, mensajes, fuentes, mapas de bits, etc. Esta biblioteca es una colección de más de 60 clases reutilizables C++ y que son compatibles totalmente con Windows 3.0 y 3.1. Una característica muy notable de esta biblioteca es su trasportabilidad, que no exige más que una recompilación de aplicaciones a otros entornos. Microsoft ha pensado en el futuro y la biblioteca será compatible con el próximo sistema operativo Windows de 32 bits, el NT.

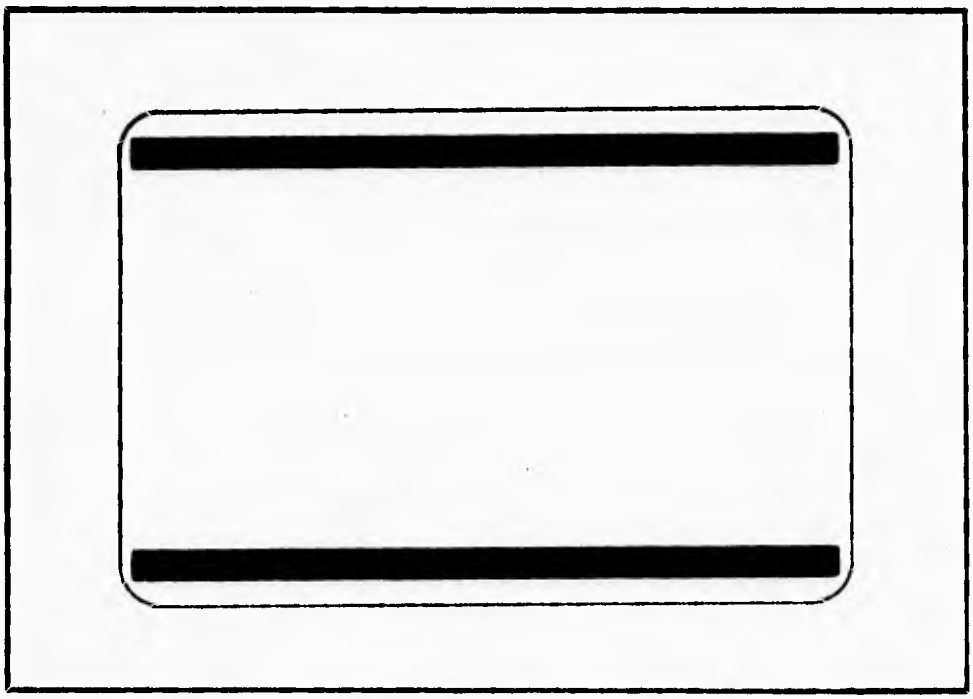


Fig. 1.1.4.7. Menú principal de PWB

MSC 7.0 incorpora un entorno integrado PWB (programmer's WorkBench) bajo DOS. Figura 1.1.4.7. Para programadores DOS ofrece, además de las herramientas citadas, una serie de propiedades notables, así como otras herramientas tales como: cabeceras precompiladas, alineación de código de máquina (posibilidad de insertar código ensamblado sin necesidad del Macro Assembler), manejador de memoria virtual, manejador de recubrimientos Overlays y una biblioteca QuickWin que permiten convertir una aplicaión MS-DOS en una simple aplicación para Windows (utilizándolas se pueden transportar muchos programas compilados con Microsoft C/C++ para ejecutarse en una ventana sólo de texto de Windows).

MICROSOFT VISUAL C++

El Microsoft Visual C++ es uno de los dos paquetes de programación más potentes del mercado, en donde se ha conseguido un compilador capaz de realizar aplicaciones muy sofisticadas con cierta facilidad, gracias a la última versión de las librerías de clases Microsoft, las MFC (Microsoft Foundation Classes).

1.1.5 IMPLEMENTACIÓN DE ALGORITMOS CON LAS ESTRUCTURAS DE DATOS

La implentación de los algoritmos con las estructuras de datos se divide principalmente en dos grandes grupos: objetos y clases.

Un objeto es aquella variable que va ha ser declarada dentro de una clase especifica un objeto puede contener una copia de todos los campos de un dato,los cuales son definidos en el campo de clases.

Las clases especifican el tipo detallado de implementación, comunmente esta implementación solamente se encuentra localizada en el campo de clase, por tal motivo también se le denomina como de **tipo privado,** cuando la implementación no se encuentra localizada en el campo de clase se le denomina de **tipo público**.

Las operaciónes también están clasificadas en públicas o privadas; las públicas son las que pueden ser accesibles fuera del campo de clase y éstas son generalmente suficientes para cualquier aplicación, las privadas, como su nombre lo indica son aquellas que solo pueden ser accesadas dentro del campo. En el lenguaje de programación orientada a objetos, a las operaciones que se definen en las clases se les denomina **metodos**, estos métodos son análogos a los procedimientos y funciones que se llevan a cabo en una programación que no está en lenguaje orientado a objetos.

Al proceso de llamar a uno o mas métodos se le conoce como "*mandar un mensaje al objeto*".

Una vez que ya se definieron las estructuras de datos como objetos y clases el siguiente paso es efectuar la implementación de los algoritmos. Esta implementación se llevará acabo siguiendo una arquitectura especifica, la cual está basada en dos dimensiones abstractas como puntos rectángulos e imagenes rectángulares llamadas "mapas de bits", se compone de tres niveles: el nivel de aplicación, el nivel de presentación y el nivel denominado terminal virtual.

1.- Nivel de aplicación.- En este nivel se implementa la funcionalidad de la interface, sin preocuparse en como será realizada está con el usuario. Sus componentes de este nivel son llamados modelos debido a que la mayoria de sus aplicaciones incluyen modelar algunos aspectos de la realidad, los modelos son los esclavos del usuario de la interface, además de que no guardan ninguna información respecto al modo en que se utiliza esta debido a que la parte principal de la presentación puede ser distinta y puede ser remplazada en cualquier momento.

	Componente y dispositivo	Característica
Aplicación	modelos	Se implementa la funcionalidad de la interface
Presentación		Se realizan los reconocimientos de los modelos
Terminal Virtual	mouse,terminal gráfica	Se encuentra la librería de los gráficos

Fig. 1.1.5.1 Características de los niveles de implementación.

2.- Nivel de presentación.- En este nivel se realiza la interface entre una aplicación específica y una clase de usuario específico. Su principal función es el reconocimiento de los modelos que existen en el nivel de aplicación, con los cuales se realiza una implementación.

Un modelo cualquiera puede tener muchos tipos de usuarios, por lo tanto existirán diferentes tipos de reconocimiento para servir a cada necesidad. Por ejemplo una misma implementación puede servir como editora en una máquina de alta resolución, en otra puede servir para imprimir reportes sobre la línea de impresora, o puede servir como soporte para un experto programador.

3.- Nivel de terminal virtual.- En este nivel se implementa una interface llamada dispositivo independiente entre el nivel de presentación y algunas clases de dispositivos de hardware. Solamente dispositivos de alta resolución podrán ser considerados en esta parte, aunque la arquitectura puede ser modificada para aplicarla a terminales con disco duro, y terminales alfanuméricas.

Los dispositivos de hardware que pueden ser utilizados son el mouse o una terminal gráfica. Los reconocimientos deben contemplar los comandos de el usuario. Los modelos nunca mantienen información que los pueda hacer dependientes de cualquier presentación particular.

La nueva arquitectura especifica que debe ser dividida la aplicación en tres distintos niveles: la de aplicación en donde se encuentran localizados los modelos, la de presentación en donde se realizan los reconocimientos, y la terminal virtual en donde se encuentra la librería de los gráficos; sin embargo se han encontrado ciertos problemas que ocurren cuando se construye un programa de una colección de archivos fuentes, cada vez que un archivo es editado debe ser compilado otra vez para crear un archivo binario, posteriormente el programa lo unirá nuevamente. Algunos programadores solucionan este problema compilando y uniendo cada archivo cada vez que vaya cambiando, otros utilizan un comando que compila a cada uno cada vez que exista un cambio; ambas opciones son caras, la primera en productividad y la segunda en recursos de máquina.

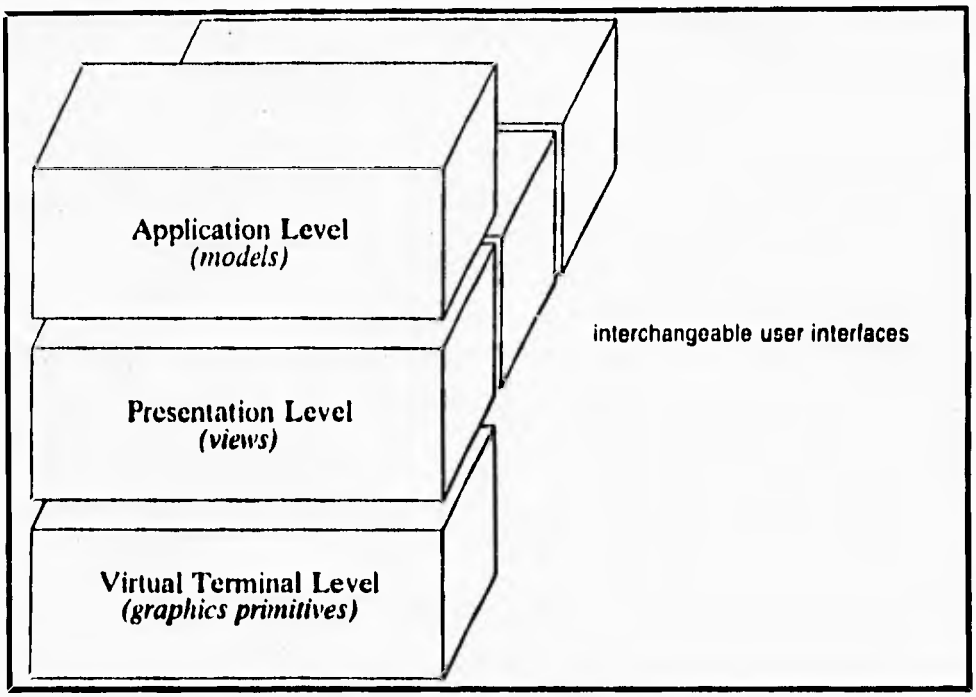


Fig. 1.1.5.2 Arquitectura específica de las aplicaciones de los niveles.

Todo esto se puede solucionar realizando el trabajo automáticamente al darle la descripción a cada uno de los archivos, dependiendo de cuales comandos se necesiten; esta solución es un caso específico de dependencia los cambios pueden ser detectados al monitorear los tiempos en que se están realizando las modificaciones. La siguiente gráfica nos muestra como se presenta una interface, en donde tres archivos son compilados para producir tres archivos binarios.

Relación entre modelos y vistas (reconocimiento).- Los modelos mantienen la información para que se realice la interface al usuario, y las vistas implementan la interface. El control de esta operación se encuentra en el usuario y no en la aplicación, la aplicación es el esclavo del usuario y nada más. El nivel de presentación contiene información del nivel de

aplicación, y nunca a la inversa, esto permite que el nivel de presentación pueda detectar cualquier cosa o pueda reemplazarla de acuerdo a cada necesidad.

Existen dos razones por las cuales se construyeron estas aplicaciónes. La primera fué para utilizarla más como modelo, por ejemplo para construir modelos que realizaran la mayor parte de su función comunincandose, lo cual produce que se corra mediante un proceso separado. La aplicación debe analizar el archivo creado para crear los modelos que describan la información que se encuentra localizada en el archivo, posteriormente crear una interface que permita que los modelos puedan ser cambiados y dar a conocer el modelo editado para que así se pueda realizar la regeneración de el archivo creado. La segunda fue simplemente reimplementar las funciones utilizando las tecnicas de dependencia grafica.

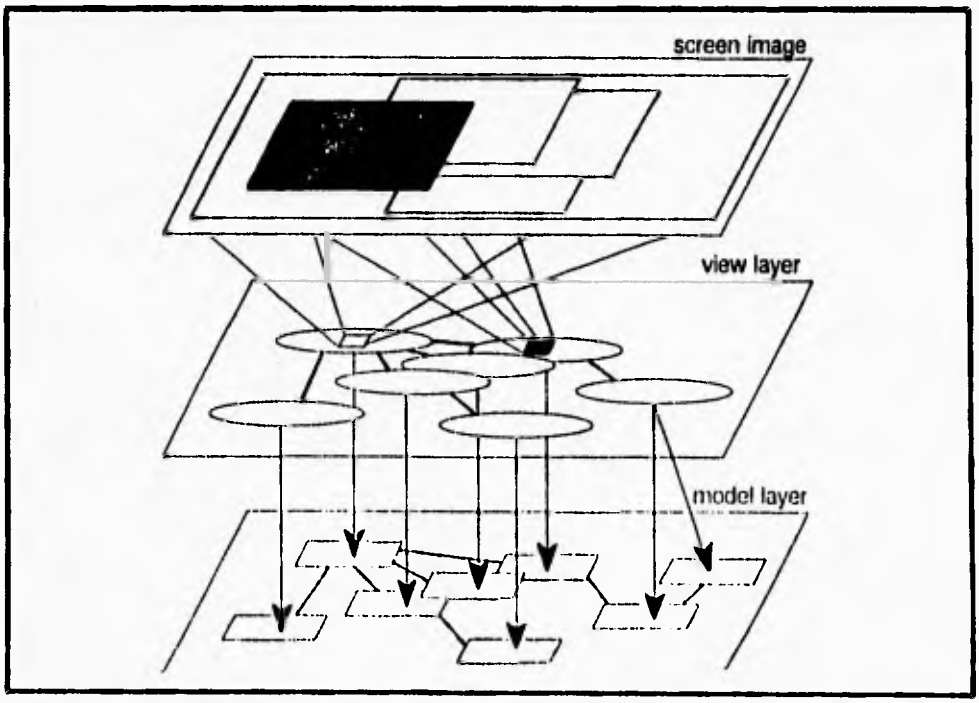


Fig. 1.1.5.3 Realción entre modelos y vistas.

Las clases que el "worckbench" define son tres variables: entidades, acciones, y dependencia.

Las entidades describen el nombre del archivo y las modificaciones de fecha; las acciones describen el comando de mando; la dependencia contiene las uniones entre las entidades y las acciones y viceversa.

Las aplicaciones de el "workbench" consisten en una rutina principal, la cual crea una instancia simple de clase. Esta instancia guarda la colección de entidades, acciones y dependencias en instancias variables; cada entidad corresponde a un archivo. Utilizando la interface el usuario puede crear instancias de acción que describirán el comando de mando y conectará todas estas acciones a las entidades.

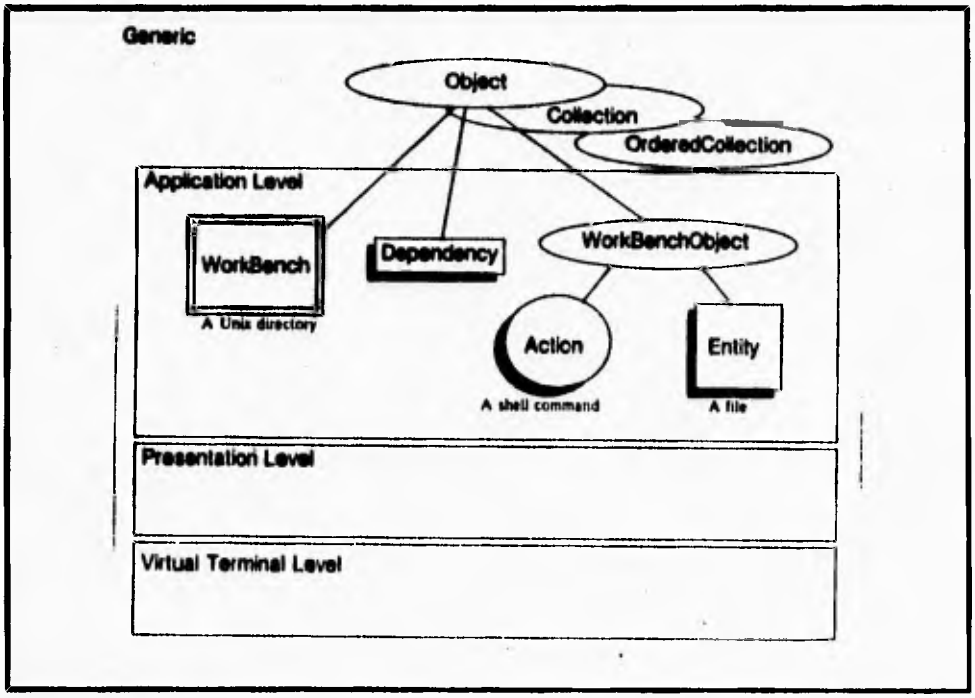


Fig. 1.1.5.4 Aplicaciones de WorkBench.

1.2 El lenguaje C ++ y la Programación Orientada a Objetos

1.2.1 EL LENGUAJE Y SU HISTORIA

El lenguaje C nació en los laboratorios de la Bell Telephone y ha sido estrechamente asociado con el sistema operativo UNIX, ya que su desarrollo se realizó en este sistema y debido a que tanto UNIX como el propio compilador de C y la casi totalidad de los programas y herramientas de UNIX, fueron escritos en C. Su eficiencia y claridad han hecho que el lenguaje ensamblador apenas haya sido utilizado en UNIX.

Este lenguaje esta inspirado en el lenguaje B escrito por Ken Thompson en 1970 con intención de recodificar el UNIX, que en la fase de arranque estaba escrito en ensamblador, en vistas a su transportabilidad a otras máquinas. B era un lenguaje evolucionado e independiente de la máquina, inspirado en el lenguaje BCPL concebido por Martin Richard en 1967.

En 1972, Denis Ritchie, toma el relevo y modifica el lenguaje B, creando el lenguaje C y reescribiendo el UNIX en dicho lenguaje. La novedad que proporcionó el lenguaje C sobre B fue el diseño de tipos y estructuras de datos.

Los tipos básicos de datos eran **int** (entero), **char** (caracter), **float** (reales en simple precisión) y **double** (reales en doble precisión). Posteriormente se añadieron los tipos **long** (enteros de 32 bits), **short** (enteros de 16 bits) y **unsigned** (enteros sin signo). Los tipos estructurados básicos de C son las estructuras, las uniones y las enumeraciones. Estos permiten la definición y la declaración de tipos derivados de mayor complejidad.

Las instrucciones de control de flujo de C son las habituales de la programación estructurada: **if, for, while, switch-case** todas incluidas en su predecesor BCPL.

C también incluye punteros y funciones. Los argumentos de las funciones se pasan por valor, esto es copiando su valor, lo cual hace que no se modifiquen lo valores de los argumentos en la llamada. Cuando se desea modificar los argumentos en la llamada, estos se pasan por referencia, esto es, se pasan las direcciones de los argumentos. Por otra parte, cualquier función puede ser llamada recursivamente.

Aunque C tiene cinco tipos básicos de datos incorporados, no es un lenguaje potente en este sentido en relación con Pascal o el Ada. C permite casi todos los tipos de conversaciones, y se puede, con plena libertad, cambiar caracteres y tipos enteros en la mayoría de las expresiones. No realiza ninguna comprobación de errores durante el tiempo de ejecución - tales como comprobaciones de los limites del arreglo o de la compatibilidad de los tipos de argumento. (Algunos compiladores pueden informar de errores tales como divisiones por cero o colisiones dentro de la pila, dependiendo de la implementación que utilice). Por tanto, la comprobación de errores durante el tiempo de ejecución es responsabilidad del programador.

C es especial, ya que permite una manipulación de los bits, bytes, palabras y apuntadores. Esto hace que se adapte perfectamente a la programación a nivel de sistema, donde este tipo de operaciones es común. Otro aspecto importante del C es que sólo tiene 32 palabras clave (27 palabras clave que provienen del estandar de Kernighan y Ritchie y 5 palabras clave del comité de estandarización del ANSI), que son los comandos que forman el lenguaje C. En comparación con el BASIC para IBM PC que tiene 159.

Una de las peculiaridades de C es su riqueza de operadores. Prácticamente puede decirse que dispone de un operador para cada una de las posibles operaciones de código de máquina.

Hay toda una serie de operaciones que pueden hacerse con el lenguaje C que realmente no están incluidas en el compilador propiamente dicho, sino que las realiza un preprocesador justo antes de cada compilación. Las dos más importantes son **#define** (directriz de sustitución simbólica o de definición) e **#include** (directriz de inclusión en el archivo fuente).

Al C se le considera comúnmente como un lenguaje estructurado con ciertas similitudes con el Algol y el Pascal. Aunque el termino lenguaje estructurado por bloques no se aplica estrictamente al C en un sentido académico, el C es una parte informal de ese grupo de lenguajes. El aspecto diferencial de un lenguaje estructurado por bloques es el de la combinación del código y los datos. Esto significa que puede seccionar y esconder al resto del programa toda la información e instrucciones necesarias para la realización de una tarea especifica. Generalmente la combinación se lleva a cabo por subrutinas con variables locales o temporales. De esta manera permite escribir subrutinas de forma que lo que ocurra dentro de ellas no provoquen efectos colaterales en otras partes del programa. El uso excesivo de las variables globales, que son reconocidas a lo largo de todo el programa, puede llevar errores no deseados, infiltrados en el programa. En C, todas las subrutinas son funciones discretas.

Las funciones son los bloques constructores del C, donde se llevan a cabo todas las actividades del programa que le permiten definir un código de labores especificas en un programa por separado. Después de depurar una función que utilice sólo variables locales, se puede hacer referencia a ellas para trabajarlas perfectamente en distintas situaciones sin crear efectos colaterales en otras partes del programa. Todas las variables declaradas en esa función serán , conocidas exclusivamente por dicha función.

En C, la utilización de bloques de código crea también la estructura del programa. Un bloque de código es un grupo de instrucciones de programas lógicamente conectadas, que puede ser tratado como unidad.

El C es un lenguaje del programador. A diferencia de la mayoría de los lenguajes informaticos del alto nivel, el C impone pocas restricciones sobre lo que se puede hacer con él. Usando el C, un programador puede evitar el uso del código ensamblador excepto en situaciones muy exigentes. De hecho, una de las razones para la invención del C fue la de suministrar una alternativa a la programación en lenguaje ensamblador.

El lenguaje ensamblador utiliza una representación simbólica del código binario que la computadora puede ejecutar directamente. Cada operación del lenguaje ensamblador diseña una labor única que la computadora puede ejecutar; aunque el lenguaje ensamblador le da a los programadores el potencial de trabajar con él cuando se esta desarrollando o depurando un programa. Además, como el lenguaje ensamblador puro esta sin estructurar, el programa final tiende a convertirse en un "código espagueti", es decir, un entramado de saltos, llamadas e indices. Esto hace que los programas con lenguaje ensamblador sean dificiles de leer, enlazar y mantener.

Inicialmente, el C fue usado para la programación de sistemas. Un programa del sistema es parte de un gran número de programas que forman parte del sistema operativo de la computadora o de sus uitilidades de apoyo. Por ejemplo, a los siguientes se les denomina comúnmente programas del sistema:

- Sistemas Operativos
- Intérpretes
- Editores
- Ensambladores
- Compiladores
- Gestores de Base de Datos

A medida que el C creció en popularidad, muchos programadores empezaron a utilizarlo para programar todo tipo de labores, por su portabilidad y eficiencia. Debido a que hay compiladores de C para casi todas las computadoras, para que se pueda tomar el código escrito para una máquina y luego compilarlo y ejecutarlo con unos pocos cambios en otra maquina. Esta portabilidad ahorra tanto tiempo como dinero. Los compiladores de C tienden a producir códigos objeto ajustados y rápidos, más pequeños y rápidos que la mayoría de los compiladores BASIC, por ejemplo.

Quizás la razón real de que el C se use en todos los tipos de labores de programación es que a los programadores les gusta C, ya que tiene rapidez de ensamblaje y la extensibilidad del FORTH, mientras que tiene las escasas restricciones del Pascal. Un programador de C puede crear y mantener una biblioteca única de funciones que han sido adaptadas a su propia personalidad. Debido a que C le permite una compilación separada, para que así se puedan manejar fácilmente grandes proyectos.

Aunque en un primer momento fue desarrollado para su ejecución bajo el sistema Operativo UNIX, el C se ha hecho tan popular que hay compiladores disponibles para casi todas las computadoras y sistemas operativos. Esto significa que el código C puede ser transferido entre computadoras y sistemas operativos, haciendo posible escribir el código una vez y utilizarlo en cualquier momento y lugar.

Finalmente, C, que ha sido pensado para ser altamente transportable y para programar lo improgramable, igual que otros lenguajes tiene sus inconvenientes. Carece de instrucciones de entrada/salida, de instrucciones para manejo de cadenas de caracteres, con lo que este trabajo queda para la librería de rutinas, con la consiguiente perdida de transportabilidad. La excesiva libertad en la escritura de los programas puede llevar a errores en la programación que, por ser correctos sintacticamente no se detectan a simple vista. Por otra parte las procedencias de los operadores convierten a veces las expresiones en pequeños rompecabezas. A pesar de todo, C ha demostrado ser un lenguaje extremadamente eficaz y expresivo.

Este lenguaje ha evolucionado paralelamente a UNIX, que a su vez ha pasado por diversas versiones entre las que destaca XENIX. Esta versión, fue desarrollada por Microsoft para las micro computadoras de 16 bits.

El lenguaje C ha seguido evolucionando sin perder sus características originales dando origen al lenguaje C++.

C++ esta basado en C y es un superconjunto de C. C++ retiene el poder y la flexibilidad de C para permitir la interfaz hardware-software con la programación a bajo nivel.

El poder de eficiencia y economía de las expresiones de C están contenidas en C++, sin embargo, C++ provee una plataforma que soporta la programación orientada a Objetos y los problemas de abstracción de alto nivel.

C++ es un lenguaje híbrido. El paradigma de la programación orientada a Objetos puede ser explotado totalmente para producir una solución orientada a objetos, a un problema, o puede ser tratado como un lenguaje procedural que contiene algunas construcciones adicionales. Esta dualidad en C++, representa un cambio significativo para un programador principiante. Ya que no solamente esta el aprender un nuevo lenguaje sino el aprender una nueva forma de pensar y de resolver problemas.

C++ deriva de la inspiración de BCPL y Simula67. La noción de subclases y las funciones virtuales están tomadas de Simula. La habilidad para sobrecargar operadores y la flexibilidad para incluir declaraciones cerradas a su primer punto de aplicación son reminiscencias de Algol68. Es claro que C++, como otros modernos lenguajes de programación, representa una evolución y refinamiento de algunas de las mejores características de los lenguajes previos.

C++ fue desarrollado por Bjarne Stroustrup en los laboratorios Bell a principios de los años 80's. El Dr. Stroustrup desarrollo C++ para mantener la escritura de algunas simulaciones complejas cuyas consideraciones de eficiencia preincluyen el uso de Simula67.

En julio de 1983 C++ fue primeramente instalado por el grupo de Stroustrup quien desarrollo el lenguaje. En el verano de 1987, C++ todavia seguia evolucionando. No hubo un lenguaje estándar o formal. El Dr. Stroustrup y sus colegas en los laboratorios AT & T continuaron haciendo refinamientos al lenguaje de acuerdo a su experiencia con él.

Una convención a lo que han llegado los desarrolladores del lenguaje es hacer compatible a C . Esto preserva la integridad de millones de lineas de código de C, manteniendo la integridad de las bibliotecas de C y las herramientas que se han desarrollado con él, lo que trae como beneficio que haya incentivos para un programador de C, el tener que aprender C++, sin tener que sacrificar o perder lo que ha realizado hasta ese momento.

Para muchos desarrolladores de software el lenguaje es como una religión así que muchos programadores de C, no tuvieron ninguna dificultad para incorporar el C + + a sus actividades.

El lenguaje C es sencillo, compacto, elegante y veloz por lo que resulta de gran utilidad para aquellos programadores que pretendan un software de altas miras y al mismo tiempo efectivo.

Podemos afirmar, sin lugar a dudas, que el lenguaje C, a través de su corta existencia, ha demostrado ser uno de los más versátiles, completos y estructurados, que posee como gran ventaja a destacar, la transportabilidad, es decir, la posibilidad de utilizarlo tanto en macrocomputadoras como en mini y micro computadoras.

Además, como muestra de su gran aceptación, son cada vez más numerosos los programas escritos en lenguaje C.

Se puede decir, que todo programador que se precie como tal, debe conocer a fondo el lenguaje C, porque se ha convertido en el más universal de los lenguajes y el que menos reformas sufra en un futuro próximo.

La evolución de los lenguajes de Programación Orientada a Objetos.

Los ensambladores fueron los primeros lenguajes de programación que introdujeron representaciones simbólicas para las instrucciones de maquina. Algunos de los primeros ensambladores fueron Soap para la IBM 650 (a mediados de los años 50's) y Sap para la IBM 704 (después de los años 50's) pero el primer lenguaje de programación de alto nivel fue sin duda FORTRAN.

FORTRAN introdujo varios conceptos importantes a los lenguajes de programación, incluyendo variables, arreglos, estructuras de control (tales como las estructuras condicionales y las iterativas). FORTRAN es todavía uno de los más populares lenguajes de programación. El lenguaje tiene una activa estandarización ANSI que periódicamente realiza cambios substanciales y extensiones al lenguaje.

Todavía, después de los años 50's se encontraron con un problema cuando se desarrollaban programas grandes, que era que los nombres de las variables entraban en conflicto en diferentes partes del programa. Los lenguajes de programación que sucedieron al fortran fueron PL/1, COBOL y Algol. Los diseñadores del lenguaje Algol decidieron proveer barreras para separar nombres de variables de los segmentos del programa. Con esto nació el BEGIN y el END en el Algol 60 (Randell y Rusell, 1964). Así sólo cuando los nombres de las variables aparecen en un bloque son conocidas unicamente en ese bloque. Este fue el primer intento de proteger o encapsular bloques en un lenguaje de programación. Las estructuras de bloques son ahora extensamente usados en una amplia variedad de lenguajes.

A principios de los años 60's, los diseñadores del lenguaje Simula-67 (Dahl y Nygaard, 1966; dahl, Myhrhaug y Nygaard, 1970) tomaron el concepto de bloque de Algol para introducir más tarde el concepto de Objeto. Aunque las raíces de Simula fueron en Algol, Simula, fue principalmente conocido como un lenguaje de simulación. Así, los objetos de Simula tuvieron una existencia propia conteniendo datos y el poder comunicarse con

otros durante una simulación. Conceptualmente un objeto contiene ambos, los datos y las operaciones que manipulan sus datos. Las operaciones fueron llamados métodos. Simula también incorporó la noción de clases, las cuales son usadas para describir la estructura y el comportamiento de un conjunto de objetos. La herencia de clase fue también soportada por simula. La herencia organiza las clases dentro de jerarquías, permitiendo parte de la implementación de las estructuras. Simula también distingue entre dos tipos de igualdad, idénticamente y superficial que refleja la distinción entre referente basado en valor y basados en la interpretación de objetos.

Otro importante desarrollo dentro de los lenguajes de programación fue el lenguaje LISP (McCarthy et al.,1965). Lisp fue introducido a finales de los años 50's y a principios de los años 60's, LISP fue uno de los más elegantes y refinados lenguajes utilizando pocas y simples construcciones de programación (listas y funciones) para ejecutar procesos complejos. LISP es uno de los principales lenguajes que por su refinamiento se utilizan en las aplicaciones de la Inteligencia Artificial.

A principios de los años 70's, el concepto de abstracción de datos fue propuesto por un número de diseñadores de lenguajes con el propósito de manejar extensos programas (Parnas,1972). Existen dos aspectos fundamentales de la abstracción de datos. Uno es agrupar las estructuras de un tipo con las operaciones definidas por su tipo. Por ejemplo, con Algol o Pascal, en donde el lenguaje no agrupa todas las operaciones en un tipo de registro del mismo modulo con la definición del tipo de registro.

El otro aspecto de la abstracción de datos es la ocultación de información, donde los detalles de implementación y representación de los objetos son escondidos y no pueden ser accesados directamente hacia los usuarios del objeto. Los lenguajes tales como Alphard (Wulf London y Shaw, 1976) y CLU (Liskov et al,. 1977) introdujeron la abstracción de datos. En CLU, por ejemplo, los tipos de abstracción de datos fueron implementados en agrupaciones con un nombre apropiado. Uno de los más importantes lenguajes que manejaron la abstracción de datos fue el ADA (Booch, 1986). El departamento de la defensa

de los E.U. comisionaron el diseño de ADA para reducir y controlar el costo de desarrollo de software. El departamento de la defensa intento que se produjeran en ADA los nuevos sistemas que ellos manejaran. El lenguaje contenía las usuales estructuras de control, y las definiciones de tipo, funciones y subrutinas. Las construcciones orientadas a objetos de ADA incluían diseño de paquetes.

Los lenguajes orientados a objetos en los 90's

En 1982 se dijo "que la programación orientada a objetos sería para los años 80's lo que la programación estructurada fue para los años 70's" (Rentsch, 1982). Los 80's serán probablemente conocidos como la década que lanzó la orientación de objetos en la era de la computación.

En el año de 1986 se dio la primera conferencia, OOPSLA (Lenguajes y Sistemas de programación orientada a objetos), completamente dedicada a la orientación de objetos, y en 1988 salió la primera publicación completamente dedicada a la orientación de objetos: The Journal of Object-Oriented Programming).

Después de este evento hubo un notable incremento de las conferencias sobre Programación orientada a objetos. Simultáneamente, surgieron los lenguajes orientados a objetos tales como C++, Objective-C, Smaltalk-80, Eiffel y extensiones de LISP orientadas a objetos que llegaron a estar disponibles comercialmente. Varias de las mejores compañías de computadoras tales como AT & T, SUN, y Microsoft comenzaron a implementar estilos de programación orientados a objetos en el desarrollo de su propio software.

Durante los años 70's y los años 80's, los conceptos orientados a objetos de Simula y otros de los primeros prototipos estuvieron influenciados por los lenguajes: Smaltalk que inicialmente fué un proyecto de Xerox Palo Alto Research park (PARC). Durante los años 70's un grupo de desarrolladores de Xerox PARC estuvieron revolucionando el futuro de la industria de la computación. Los desarrolladores PARC crearon tecnologías actualmente conocidas como tecnologías orientadas a objetos. El lenguaje Smaltalk fue desarrollado

simultáneamente en este tiempo. En términos de la interfaz de usuario surgieron las estaciones de trabajo y sus predecedores fueron influenciados por el diseño de los ambientes de la Apple Macintosh, Aldus Page Maker y Microsoft windows.

En la década de los años 90's, los lenguajes, las técnicas, bases de datos e interfaces de usuarios orientadas a objetos van a ser cada vez más populares. La mayoría del desarrollo de software se va a ver afectado por la orientación de objetos de una u otra forma. Los años 90's van a ser conocidos como los años de la proliferación de los lenguajes y tecnologías orientadas a objetos.

¿Por qué Programación Orientada a Objetos?

Hace varios años, investigadores de la ciencia de la computación observaron que los programadores podían escribir y depurar más o menos la misma cantidad de código sin importar el lenguaje que utilizaran. La cantidad de trabajo es más o menos la misma, pero no así los resultados. Escribir 100 líneas de código en lenguaje ensamblador, se obtiene cierto resultado pero con el código en C se obtiene mucho más. Con esto mente, los investigadores buscaron desarrollar lenguajes de alto nivel que multiplicaran el poder de un programador, reduciendo así el tiempo y los costos de desarrollo de los proyectos.

En la década de 1970 se volvió popular el concepto de Objeto entre los investigadores de los lenguajes de programación. Un objeto es un conjunto de códigos, datos diseñados para emular o imitar una entidad física o abstracta. Los objetos son eficientes como elementos de programación por dos razones principales: representan una abstracción directa de elementos que se utilizan comúnmente y ocultan la mayor parte de la complejidad de su implantación a los usuarios. Los primeros objetos que se desarrollaron fueron aquellos que estaban más íntimamente ligados a las computadoras, como integer, Array y stack. Se diseñaron algunos lenguajes (como Smalltalk) como lenguajes ortodoxos en los cuales todo se define como un objeto.

La programación orientada a objetos es una metodología que da gran importancia a las relaciones entre objetos, más que a los detalles de la implantación. Las relaciones son nexos entre objetos y suelen desarrollarse a través de árboles genealógicos en los que se crean nuevos tipos de objetos a partir de otros. Ocultar la implantación de un objeto hace que el usuario tenga que ver más con la relación que guarda un objeto con el resto del sistema y no con la forma en que se implanta el comportamiento de un objeto. Esta distinción es importante y representa una desviación fundamental de los lenguajes "imperativos" (como C), en los que las funciones y las llamadas a funciones son el centro de la actividad.

En C++, pocos objetos son parte del lenguaje mismo. El trabajo pesado y la responsabilidad de diseñar objetos corresponden al usuario. El poder de la programación Orientada a Objetos (POO) se aprovecha si se desarrollan grupos de objetos que suelen denominarse Jerarquías de clases. El desarrollo de estas jerarquías de clases es una actividad fundamental en POO.

La evolución de la programación orientada a objetos en el uso de interfaces

En los últimos años la comunicación de los usuarios con las computadoras se ha llevado a cabo a través de interfaces basadas en comandos.

En los primeros sistemas computacionales los usuarios tenían que memorizar comandos o en su defecto consultar grandes manuales para poder comunicarse con las máquinas. Las primeras microcomputadoras y mainframes, requerían de un gran conjunto de instrucciones para poder funcionar.

Los primeros usuarios de computadoras fueron ingenieros, a los que actualmente conocemos como expertos en ellas, éstos presentaban mucho interes en conocer más acerca de su funcionamiento y de su tecnología. El uso de las interfaces basadas en comandos fue aceptada por la mayor parte de estos usuarios ya que ellos estaban acostumbrados a utilizar un lenguaje nemotecnico para poder comunicarse con las computadoras.

En los años 70's fue entonces cuando las computadoras empezaron a ser usadas por un nuevo grupo de usuarios tales como las secretarias, directores, ejecutivos y personas en general. Los comandos basados en éstas interfaces causaron fobia a los usuarios hacia las computadoras; debido a que tenían que memorizar comandos tales como "Control-Shift-D" para borrar una palabra o "Control-Escape-D" para restaurar una palabra.

Para hacer la vida más fácil al usuario, una gran cantidad de componentes han sido inventados para comunicarse con la maquina. Al principio los unicos perifericos que existian, eran el teclado y la terminal de video, pero no fue hasta finales de los años 70's que surgió al mercado comercial un nuevo dispositivo conocido como MOUSE o "Ratón" éste cambió radicalmente el uso de las interfaces.

El mouse ya existia antes de los años años 70's, pero sólo para un selecto grupo de usuarios. En los años 60's Projects y SRI Intenational, MIT y otras universidades permitieron la invención de componentes con función de apuntadores y sistemas con ambiente gráfico (Perry y Voelcker, 1989). El mouse y el joystick son algunos de los pocos componentes apuntadores que fueron inventados en ese tiempo.

En los años 70's los diseñadores de Xerox PARC estuvieron ocupados diseñando poderosas estaciones de trabajo con ambientes gráficos. Sus experimentos se concentraron en combinar la memoria asociativa del usuario con su capacidad de manipulación directa de objetos. El objetivo básico de esas nuevas estaciones de trabajo, era que un usuario pudiera tener una poderosa computadora de escritorio totalmente dedicada al trabajo del usuario.

El concepto de utilizar la memoria asociativa del usuario es el de hacer uso de menus y cajas de dialogo que interactuen con él , en vez de tener que memorizar comandos para cada tarea, el usuario selecciona un comando de una barra de menus que a su vez este contenga un submenu con una lista de comandos disponibles. Por ejemplo la siguiente figura despliega la barra de menus de la aplicación Write de microsoft.

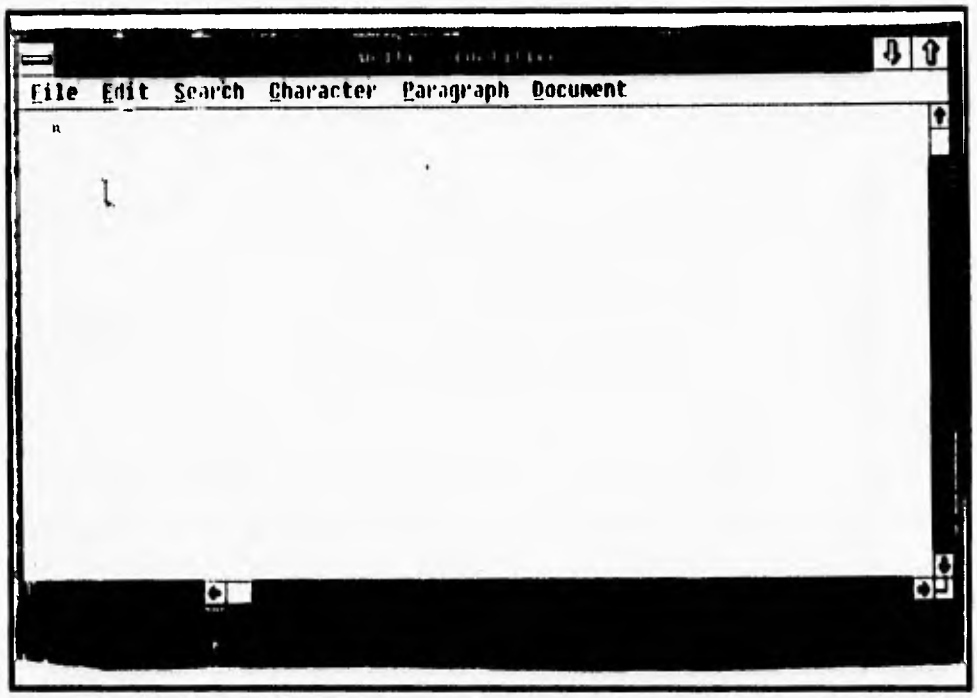


Fig 1.2.1.1 Barra de menu de Micrsoft Write.

Esta barra de menus despliega una lista de comandos disponibles tales como **FILE**, **EDIT** y **SEARCH**. Cuando el mouse apunta a cualquiera de esos comandos una determinada acción es ejecutada y aparece un submenu con otra serie de comandos. En la siguiente figura muestra el submenu del comando **caracter** apuntado. El usuario puede entonces seleccionar diferentes estilos de los caracteres disponibles.

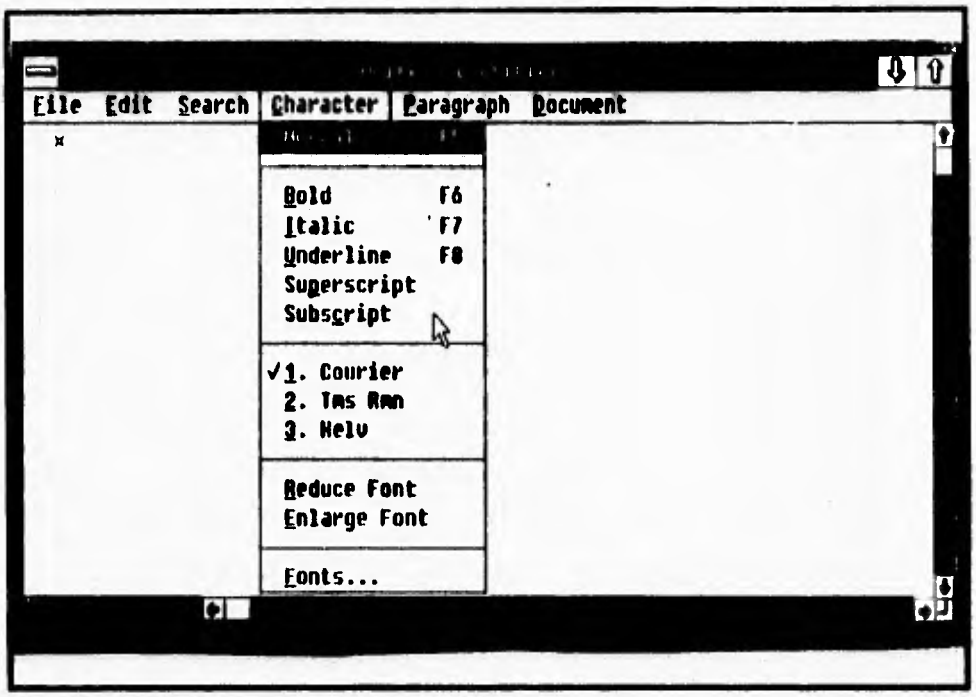


Fig. 1.2.1.2 Tipico submenu desplegable de Microsoft Write.

Las cajas de dialogo permiten la interacción entre el usuario y la computadora, las cajas de dialogo emplean una larga colección de instrucciones para el control de objetos tales como botones, barras de scroll y cajas de edición. Por ejemplo la siguiente figura muestra una caja de dialogo que se utiliza para abrir un archivo. Esta caja de dialogo esta compuesta de dos botones llamados OPEN y CLOSE, una caja de edición que permite introducir el nombre del archivo, y una región de scroll que permite navegar a través de la lista de archivos y directorios disponibles en el disco. Apuntando sobre el botón de OPEN se permite editar el archivo.

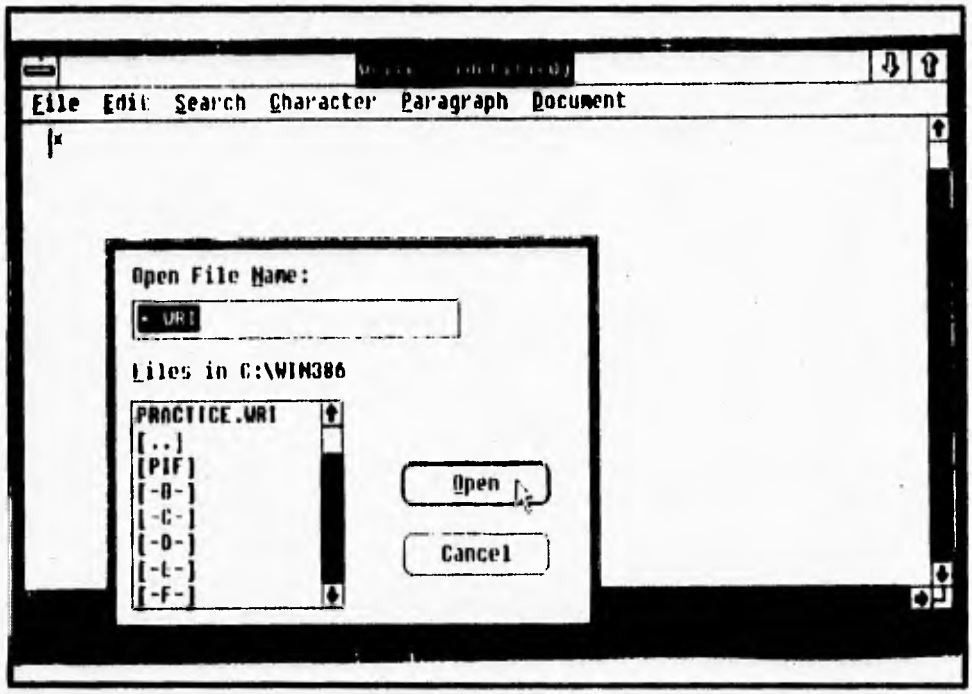


Fig. 1.2.1.3 Caja de dialogo de Microsoft.

En las interfaces gráficas, los datos textuales no son la unica forma de interacción. Los iconos representan conceptos tales como archivos de folders, cestos de basura e impresoras. Los iconos simbolizan palabras y conceptos comunmente aplicables en difererentes situaciones. La siguiente figura muestra la utilidad Microsoft Paint con una barra compuesta de iconos. Cada uno de esos iconos representan un cierto tipo de estilo de dibujo. Apuntando sobre el icono del lápiz, por ejemplo, el cursor se comporta como la punta de un lapiz para dibujar lineas.

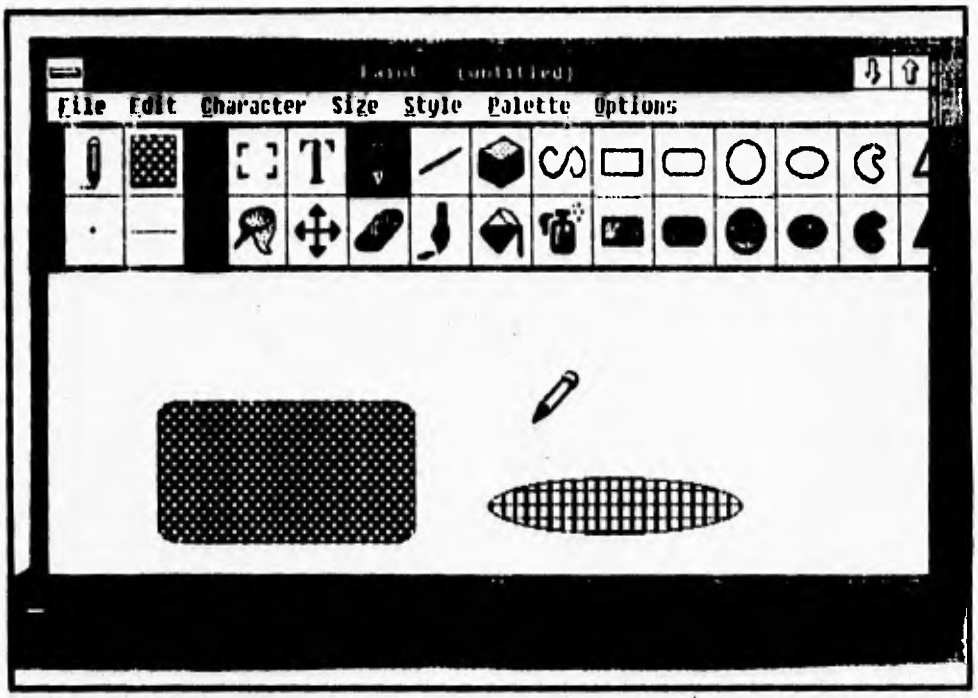


Fig. 1.2.1.4 Aplicación Paint de Microsoft.

Las aplicaciones de iconos para el diseño de interfaces estan siendo todavia exploradas en nuevos sistemas de computadoras tales como la NeXT .

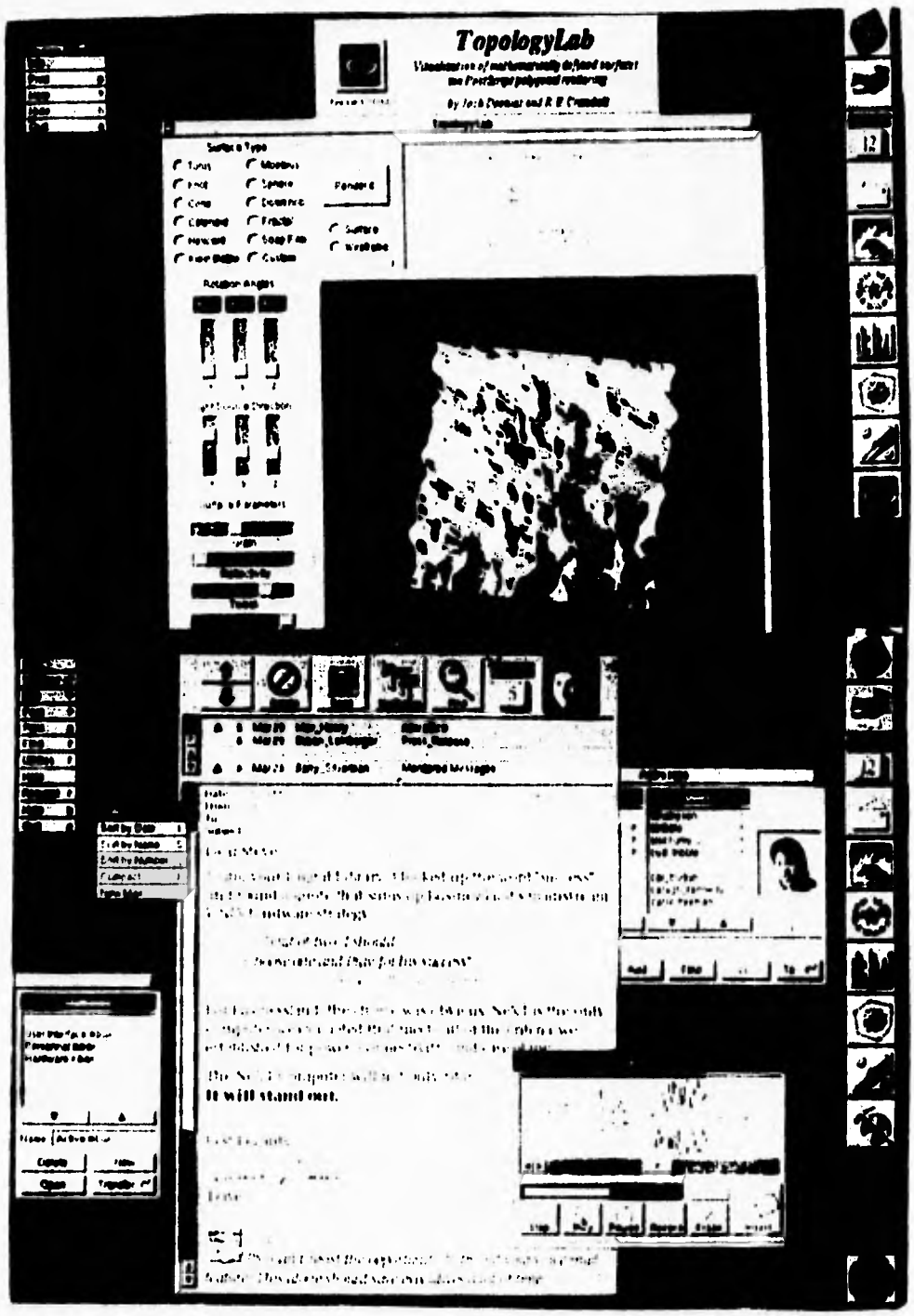


Fig. 1.2.1.5 Interface NeXT.

La idea de la metafora ha acercado al usuario a un sistema más natural. El concepto del paradigma metaforico desarrollado por Alan Kay inicio a la mayoría de los diseñadores de interfaces orientadas a objetos. "La metafora fisica es una forma de decir que los displays visuales de una computadora deberán presentar imagenes de objetos reales". Por ejemplo el icono que representa un cesto de basura puede ser usado para deshacerse de objetos de un sistema, simplemente arrastrando la imagen hacia el cesto de basura, tal como lo hiciera en la vida real.. El concepto de metafora ha sido el más famoso paradigma utilizado.

Por ejemplo en algunos de los ambientes graficos actuales han utilizado la analogia de una oficina, ya que los objetos que se manipulan, representan efectivamente objetos que se encuentran en una oficina trayendo como consecuencia que el usuario se identifique inmediatamente con el uso de cada icono. Por ejemplo en Metaphor's DIS el usuario puede crear aplicaciones graficas conectando iconos que representen tales herramientas como un manejador de base de datos, hojas de calculo, paquetes de dibujo etc.

Ademas el usuario puede producir iconos individuales que produzcan ciertos resultados. Por ejemplo, la utilidad del manejador de base de datos puede ser visualmente programado para producir el resultado de la base de datos SQL incorporada. Los usuarios pueden crear documentos compuestos de diferentes objetos tales como texto, graficos e imagenes.

La orientación de objetos rescata a la ingeniería de software, haciendo más fácil el trabajo del desarrollo de interfaces. Los nuevos conceptos de interfaces requieren el desarrollo de software más complejo para desplegar pantallas de objetos y manejar más fácilmente sus eventos. La simple tarea de desplegar las caracteristicas fisicas de una ventana requiere de varias paginas de código en un lenguaje de alto nivel tal como C. El desarrollo de interfaces graficas tales como MacApp y Actor provee de bibliotecas compuestas de objetos jerarquicos. Cada clase con sus jerarquias define los atributos necesarios a sus objetos para adherir caracteristicas de sus superclases. Cada objeto con jerarquía se comunica con otros objetos del sistema a través de la transmisión de mensajes.

Los más recientes avances en el diseño de interfaces es el NeXT Corporation. Utilizando el NeXTStep Interface Builder, el diseñador puede desarrollar rápidamente una interfaz grafica seleccionando y colocando objetos en donde desee.

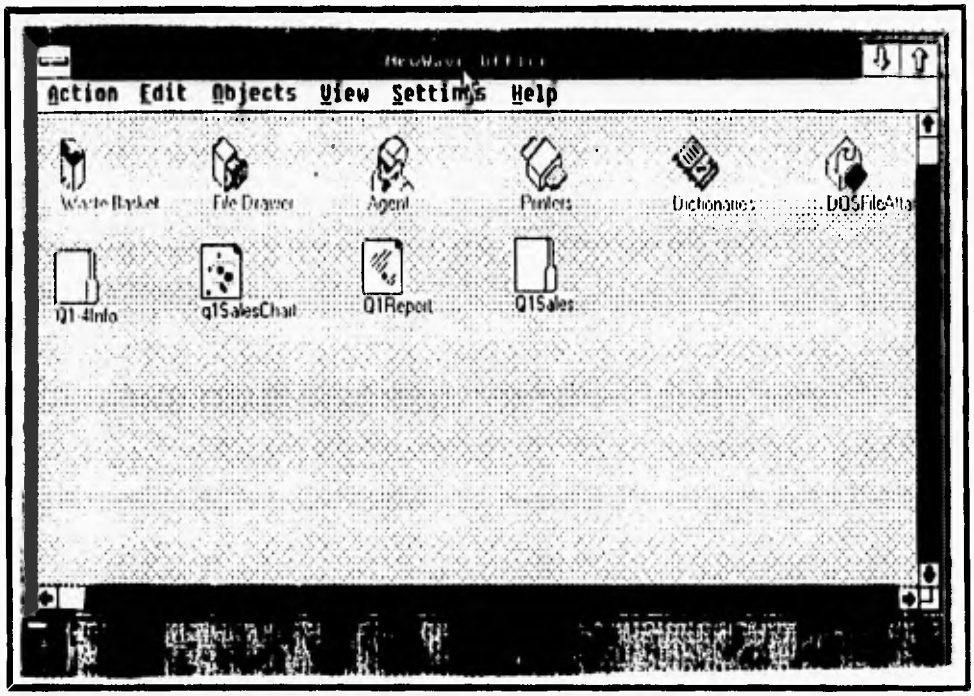


Fig 1.2.1.6 NewWave Office.

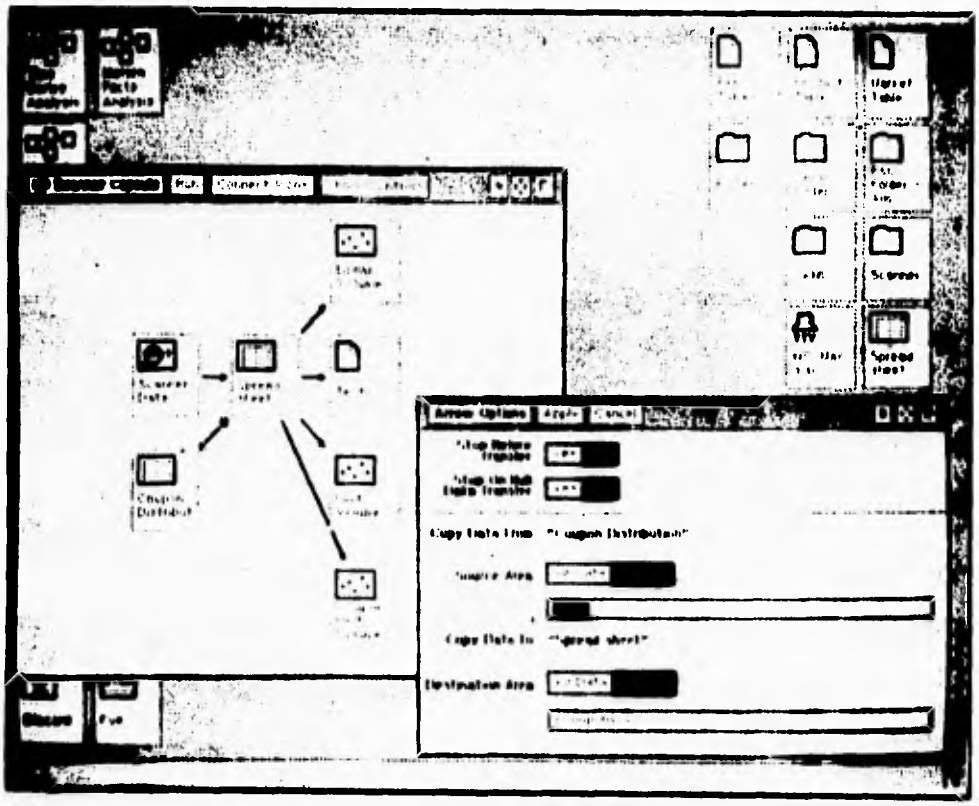


Fig. 1.2.1.7 Metaphor DIS Capsules.

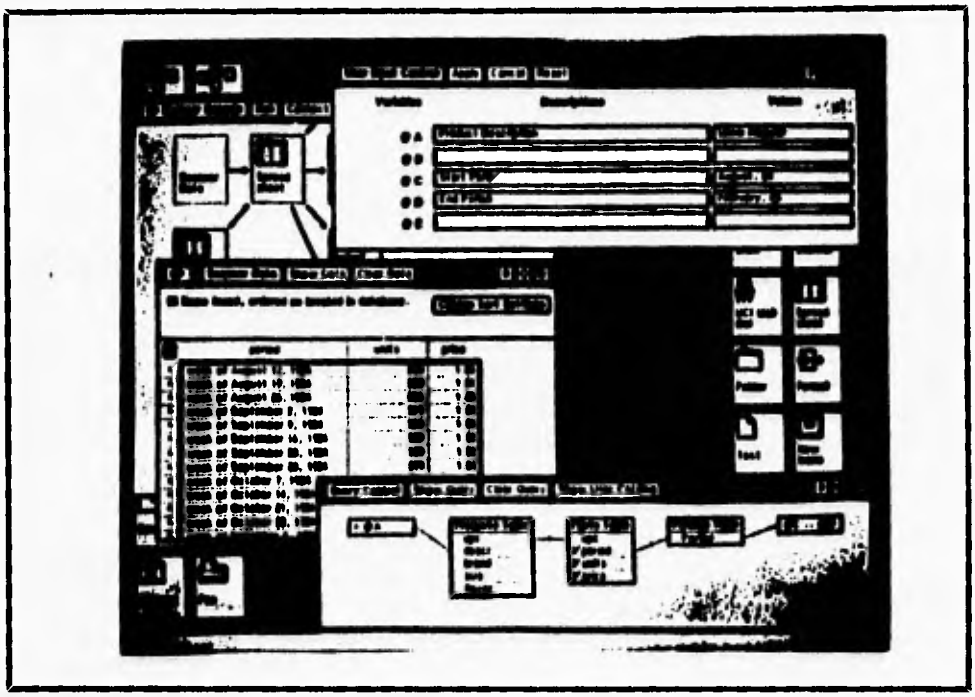


Fig. 1.2.1.8 Metaphor DIS Query.

El uso de las interfaces en los 90's

En los años 90's los sistemas computacionales incluirán procesadores más poderosos, monitores de mayor resolución y dispositivos con cada vez más memoria. Gran parte de la maquinaria del futuro utilizarán computadoras más esteticas. Las interfaces de computadoras serán más intuitivas, manejaran gran diversidad de objetos y estarán provistas de ambientes visuales.

Las interfaces jugaran un papel importante en la industria. La manipulación directa de objetos será ampliamente utilizada y las computadoras serán accesibles al público en general. En el futuro, ésta manipulación tan directa permitirá a los diseñadores utilizar interfaces seleccionando objetos directamente de la pantalla. Herramientas tales como NextStep serán comunmente utilizadas para el desarrollo de interfaces sofisticadas. Además

el usuario será apto para construir aplicaciones gráficas conectando componentees parametrizados preconstruidos.

Tecnología tales como Digital Video Interactive (DVI), la de los discos opticos y procesadores de señal digital permitirán al usuario interactuar con aplicaciones de multimedia. La orientación de objetos será ampliamente explotada para integrar diversos conjuntos de objetos no importando si son éstos, de texto, voz o video. Hypercard de apple y otras herramientas de hypermedia serán utilizadas con más frecuencia interconectando tales objetos.

La programación visual ayudará tanto a los programadores como a los no programadores para desarrollar diversas aplicaciones. Los lenguajes de programación visual permitirán a los programadores crear tipos de objetos y desarrollar rápidamente los flujos de control de tales aplicaciones. Visual SQL y QBE permitirán la realización de reportes o la extracción de información de bases de datos instaladas en mini computadoras y mainframes fácilmente sin tener que ser un experto en computadoras.

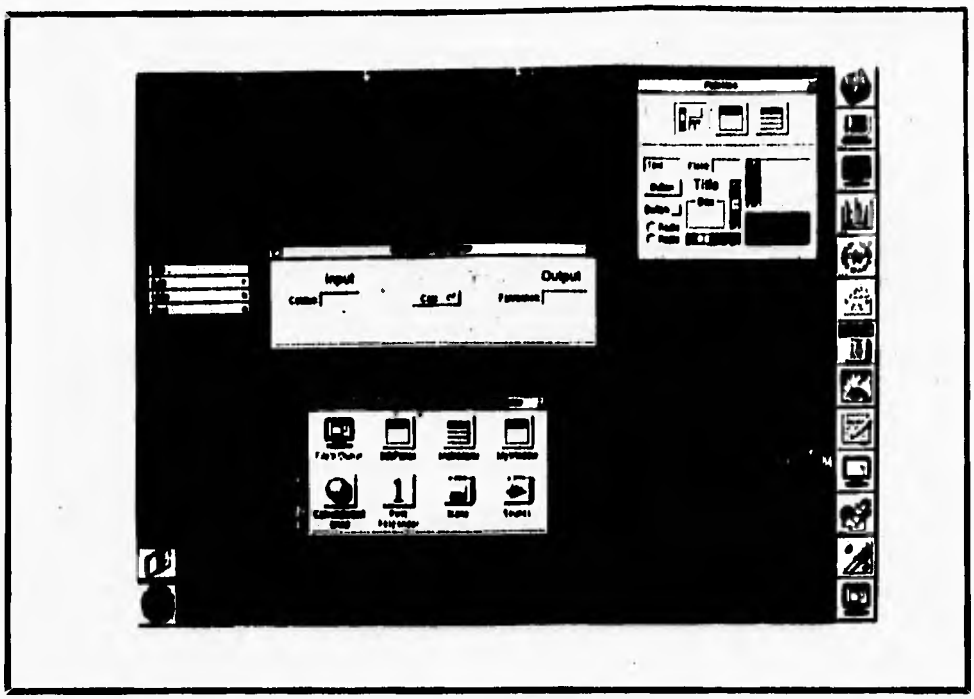


Fig. 1.2.1.9 NeXT Interface Builder.

1.2.2 VARIACIONES ENTRE C y C ++.

Se debe pensar en C++ como un subconjunto de C. La parte más importante es que cada característica que se conozca de C estará disponible en C++. Al igual que C, los programas de C++ comienzan con una función main( ). Todas las palabras reservadas de C y funciones de trabajo también las encontraremos en C++. Si siempre se han escrito programas en C que tomen las ventajas de los comandos de linea en los argumentos argc y argv, estaremos contentos de conocer que aún se encuentran en C++.

Con pocos cambios, en los programas de C, estos pueden correr en el ambiente de C++.

La siguiente tabla muestra algunos operadores de C y C++ los cuáles se utilizan en ambos casos y en otros tienen algunas diferencias que se describiran más adelante.

C	C++
-Funciones Prototipo.	-Funciones Prototipo.
Opcional	Requeridas
-Conversión de tipos automática.	-Conversion de tipos automática.
Igual	Cambios
-Alcances.	-Alcances.
Igual	Cambios
-Comentarios.	-Comentarios.
/* y */	//
	-Inicialización de argumentos por default.
-Variables por valor.	-Variables por referencia.
	-Sobrecargar el nombre de la función.
-Asignar y liberar la memoria.	-Asignar y liberar la memoria.
malloc( ) -> asignar	new -> asignar
free( ) -> liberar	delete -> liberar
	-El operador Scope (::).
	-Funciones en linea.

Tabla 1.2.2.1. Operadores de C y C++.

Se requieren Funciones Prototipo.

En C las funciones prototipo son opcionales. Si no hay conflictos entre un llamado a una función y la declaración de la misma función, el programa puede compilar.

En C++, una función prototipo es *requerida* por cada una de las funciones del programa. El programa de C++ no podra compilar a menos que siempre cada función prototipo este en su lugar. Como en C, se puede declarar una función sin teclear return. Si no se teclea return, la función asume que tiene un return interno.

En este tipo de operador se debe observar la importancia que tiene en C++, ya que se requiere siempre definir la función prototipo para cada una de las funciones de nuestros programas dentro del ambiente de C++. Las diferencias entre C y C++ son que para uno es opcional y para el otro se requiere respectivamente.

Conversión de tipos Automática.

C++ usa la mismas reglas de C para conversión de tipos automatica, pero con un pequeño cambio.

Aunque un apuntador void puede estar asignado al valor de otro tipo de apuntador sin teclearse en forma explicita, lo contrario no es verdad. Por ejemplo, aunque el siguiente codigo de compilador pertenezca a C, esto no va a compilar en C++.

```
void  *voidPtr;
void  *shortPtr;
voidPtr = shortPtr; /*  <-- This line is just
                        fine... */
shortPtr = voidPtr; /*  <-- This line is fine in C,
                        but WILL NOT compile in C++ */
shortPtr = (short *)voidPtr;  /*  <-- This works in
                         C++ */
```

Observamos en este caso que en C++ dos apuntadores no pueden estar asignados a uno de tipo void. En cambio en C el apuntador void puede tener asignados dos apuntadores.

Cambios en los Alcances.

Existen algunas diferencias entre C y C++ envolviendo el **alcance**. El alcance de una variable define la disponibilidad de la variable durante el resto del programa. Por ejemplo, una variable global esta disponible durante todo el programa, mientras una variable local esta limitada al bloque en el cuál es declarada. La mayor parte de C++ sigue las mismas reglas de alcance de C. Sin embargo hay pocas diferencias entre ellos.

La marca de comentario //.

La marca de bloque de comentario de C, /* y */, funciona de igual forma en C++. Pero, además C++ soporta una sola marca de comentario. Cuando un compilador de C++ encuentra el caracter de //, esté ignora el resto del código de esta linea. Ejemplo:

```
void main( void )
{
     short numGuppies;   // May increase suddenly !!
}
```

Como se esperaba, el carácter // es ignorado dentro de un bloque de comentario. En el siguiente ejemplo, // esta incluido como parte de un bloque de comentario.

```
void main( void )
{
     /*   Just a comment...
     //   */
}
```

Contrariamente, los caracteres de comentario /* y */ no tienen un sentido especial dentro de una linea de comentario. El inicio del bloque de comentario en el siguiente ejemplo contiene un comentario como este:

```
void main( void )
{
     //  Don't start a /* comment block
         inside a single-line comment....
         This code WILL NOT compile!!! */
}
```

El compilador va a quejarse acerca de este ejemplo.

Se puede ver con estos ejemplos que en C se empieza un comentario con /* y se debe terminar con */; pero en C++ la linea de comentarios solo utiliza el operador // y este no termina con otro operador igual, es decir únicamente utiliza un sólo operador; la desventaja es que el compilador de C++ sólo permite una línea de comentario, si el comentario excede esta linea y ocupa más, este código no será compilado y tendrá un error.

Manejando Entradas y Salidas.

En un programa estandar de C, la entrada y salida son normalmente manipuladas por las rutinas de la Libreria Estandar tales como `scanf ( )` y `printf ( )`. Se puede llamar a `scanf ( )` y `printf ( )` desde un programa de C++, esta es una alternativa elegante. El iostream facilita que el envio de una secuencia de variables y constantes a una salida normal, tal como lo hace `printf ( )`. Tambien, iostream hace fácil convertir datos desde una entrada normal hacia una secuencia de variables tal como lo hace `scanf ( )`.

`iostream` predefine tres entradas y salidas normales. `cin` es usada como entrada, `cout` para una salida normal, y `cerr` para salida de error. El operador << ("put to") es usado para enviar datos a un stream. El operador >> ("get from") es usado para recibir datos desde un stream.

Definición.

El operador << es también conocido como el operador de inserción, porque permite insertar datos dentro de un stream. El operador >> es tambien conocido como el operador de extracción porque permite extraer datos desde un stream.

Este es un ejemplo del operador << :

```
#include <iostream.h>

void main( void )
{
     cout << "Hello, world!";
}
```

Este programa envía el texto `"Hello, world!"` a la pantalla, tal como se usa `printf( )`. El archivo include `<iostream.h>` contiene todas las definiciones necesarias para usar iostream. Puesto que << es un operador binario, este requiere 2 operandos. En este caso, los operadores cout y la cadena `"Hello, world!"`. El destino de stream siempre aparece del lado izquierdo del operador <<.

Como con cualquier otro operador, se puede usar más de un << en una linea. Este es otro ejemplo:

```
#include <iostream.h>

void main( void )
{
     short     i=20;
     cout << "The value of i is " << i;
}
```

Este programa produce la siguiente salida:

```
The value of i is 20
```

`iostream` conoce todo acerca de la construcción de este tipo de datos de C++. Esto significa que las cadenas de tipo texto se imprimen como cadenas de tipo texto, `short`s como `short`s, y `float`s como `float`s, completas con punto decimal. Estos no necesitan un formato especial.

En este caso se tienen para C las entradas con el operador `scanf ( )` y las salidas con `printf ( )`; En cambio para C++ se tienen 3 tipos de operadores: `cout` para salidas, `cin` para entradas y `cerr` para salidas de error; además se debe utilizar la librería iostream.h para que el compilador entienda el manejo de dichas entradas y salidas. Además C++ utiliza los operadores << de inserción y >> de extracción, para sus entradas y salidas. Estos operadores serían la diferencia entre C y C++; mientras que `printf` y `scanf` utilizan los parentesis ( ); `cout`, `cin` y `cerr` utilizan sólo << ó >>; los parentesis indican el

inicio y final de los operadores `printf` y `scanf`, mientras que los operadores de C++ sólo utilizan un operador de inserción ó de extracción para indicar el inicio y final de estos.

Un ejemplo de Salida iostream.

Este es un ejemplo interesante de salida `iostream`. Comenzamos con un doble click en THIN C++ dentro de un archivo llamado THIN Project Manager. Vamos a encontrar THIN Project Manager dentro del folder de desarrollo THIN C++, en el folder de THIN C++. THIN C++ tiene un prompt para abrir un archivo de proyecto (Figura 1.2.2.1). Vamos dentro del folder llamado THIN C++ Projects (se tiene que mover un nivel hacia arriba para encontrar esté), entonces dentro del Subfolder llamado `Chap 04.01-cout`, y abriendo el archivo de nombre `cout`.

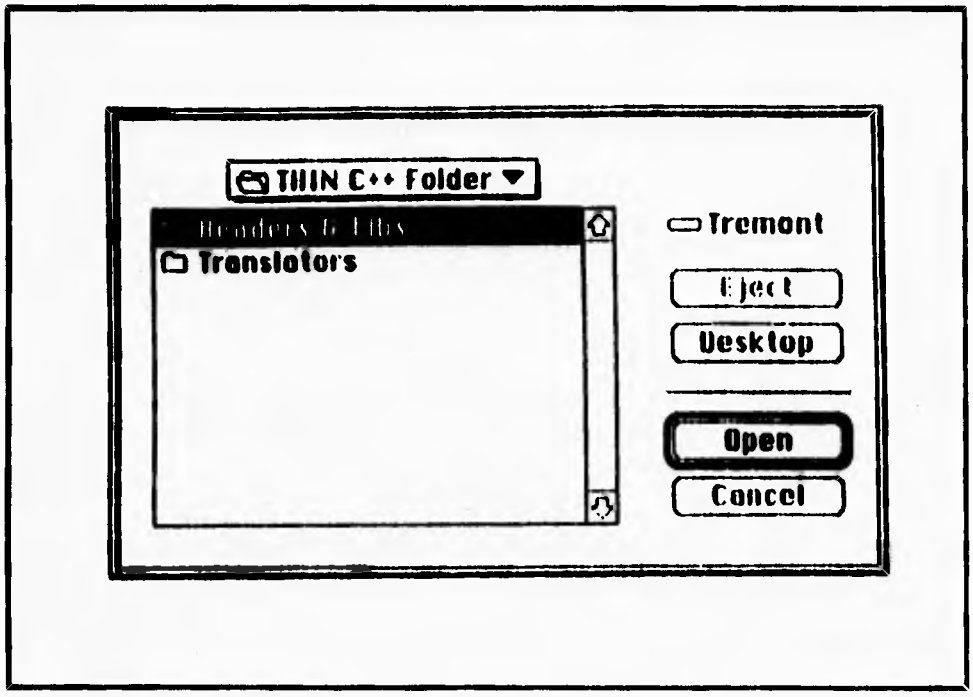


Fig. 1.2.2.1. Prompt THIN C++ para abrir un archivo de proyecto.

En este punto, aparecera una ventana de proyecto `count.`, como se muestra en la figura 1.2.2.2. Cada uno de los cuatro nombres en la ventana representan un archivo separado. `cout.cp` es el archivo que contiene el codigo fuente de C++. Los otros tres archivos son librerias que contienen varias rutinas de soporte. La libreria `iostreams` contiene todas las funciones de `iostream`, `CPlusLib` es un conjunto de rutinas de soporte para el compilador de C++, y ANSI++ es la libreria estandard de C con algunas modificaciones para soportar C++.

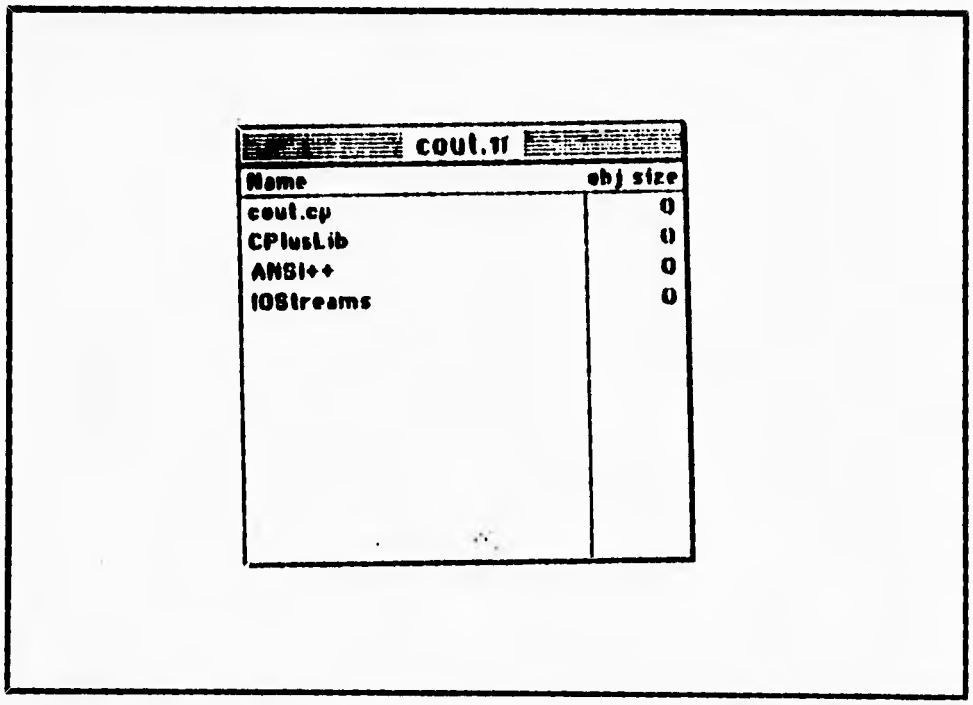


Fig. 1.2.2.2. Ventana de proyecto para cout.

Si tecleamos un doble-click sobre el nombre cout.cp en la ventana project, una nueva ventana va a aparecer, conteniendo el codigo fuente de cout, como sigue:

```
#include <iostream.h>
void main( void )
{
     char *name =  "Dr. Crusher";
     cout << "char: " << name[ 0 ] << '\n'
     << "short:      " << (short)(name[ 0 ] ) << '\n'
     << "string:     " << (unsigned long) name;
}
```

Corriendo cout.

Seleccione RUN desde el menu Project. THIN C++ va a compilar su código fuente, entonces corre el programa compilado. Cuando la ventana de la pantalla aparezca, compare su salida con la mostrada en la figura 1.2.2.3. Para salir del programa, teclee return.

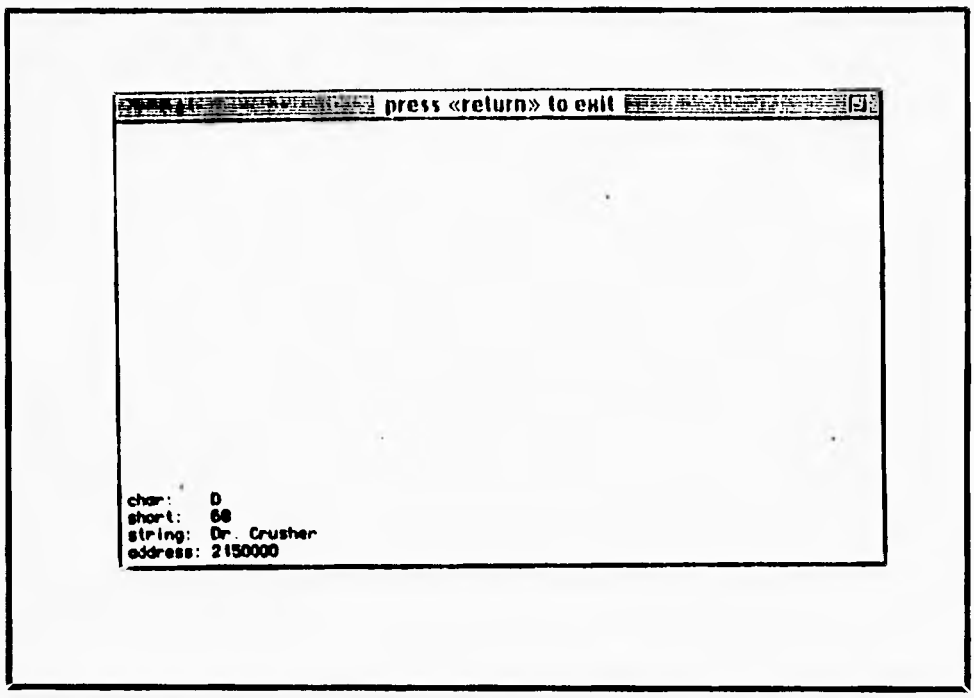


Fig. 1.2.2.3. Ventana de resultados.

La continuación del Código Fuente.

El programa empieza inicializando el apuntador nombre de tipo char, apuntando a la cadena de texto "Dr. Crusher". Después viene una declaración grande caracterizada por once diferentes ocurrencias del operador <<. Esta declaración produce cuatro lineas de salida.

El siguiente código arroja

```
cout << "char: " << name[ 0 ] << '\n'
```

produce esta linea de salida:

```
char: D
```

Como se esperaba, imprimiendo name[0] produce el primer caracter del nombre, una D.

El siguiente código que se origina es

```
<< "short:      " << (short)(name[ 0 ] ) << '\n'
```

la linea de salida asociada con este código es la siguiente:

```
short: 68
```

Este resultado fué logrado para repartir el caracter 'D' a un short. En general, iostream despliega tipos (tal como short e int) como un entero. Como se espera, un float es desplegado en formato de punto flotante.

El siguiente código que arroja es

```
<< "string:     " << name << '\n'
```

produce esta linea de salida:

```
string:   Dr. Crusher
```

Cuando el operador << encuentra un apuntador char, este asume que se requiere imprimir una cadena terminada en cero.

El código final en este ejemplo muestra otro camino para desplegar el contenido de un apuntador:

```
<< "address:  " << (unsigned long) name;
```

Otra vez, name se imprime, pero esta vez como un unsigned long. Este es el resultado.

```
address:  2150000
```

Inicialización de Argumentos por Default.

C++ permite asignar valores por default (Conocido como Inicialización de argumentos por default) a argumentos de funciones. Por ejemplo, esta es una rutina diseñada para generar un tono una frecuencia especifica:

```
void GenerateAtone( short frequency = 440 )
{
    // A frequency of 440 is equal to an A note
}
```

Se llama a esta función con un parametro, el valor anterior es usado. Por ejemplo, el llamado

```
GenerateAtone( 330 );
```

Va a generar un tono con una frecuencia de 330 beats por segundo, el cuál, en notación musical, es equivalente a una nota E. Si se llama a la función fuera especificando un valor, el valor por default es usado.

La llamada:

```
GenerateAtone( );
```

Va a generar un tono con una frecuencia de 440, el cuál representa una nota A.

El acercamiento tradicional.

Por convensión, todos los valores de los parametros por default estan especificados en la función prototipo antes que en la implementación de la función. Por ejemplo, la siguiente es una función prototipo, seguida por la misma función:

```
void MyFunc( short param1 = 27 );
void MyFunc( short param1 )
{
     // Body of the function...
}
```

Variables por Referencia.

En C, todos los parametros son pasados por valor como oponiendose al pasado existente por referencia. Cuando se pasa un parametro a una función de C, el valor del parametro es pasado a la función. Cualquier cambio hecho a este valor no es acarreado de regreso al llamado de la función.

Este es un ejemplo:

```
void DoubleMyValue( short valueParam )
{
     valueParam *= 2;
}
void main( void )
{
     short number = 10;
     DoubleMyValue( number );
}
```

main( ) pone number a 10, entonces pasa este a la función DoubleMyValue( ). Desde que number es pasado por valor, el llamado a DoubleMyValue( ) no tiene efecto sobre number. Cuando DoubleMyValue( ) regresa, number todavía tiene un valor de 10.

La Alternativa Variables por Referencia.

C++ ofrece una buena alternativa para parametros basados en apuntadores. Variables por Referencia permite pasar un parametro por referencia, sin usar apuntadores.

Esta es otra versión del programa anterior, esta vez implementada con una variable por referencia.

```
void DoubleMyValue( short &referenceParam )
{
     referenceParam *= 2;
}
void main( void )
{
     short number = 10;
     DoubleMyValue( number );
}
```

Note que este código es justo como la primera versión, con una pequeña excepción. El parametro de DoubleMyValue( ) esta definido usando el operador &:

```
short &referenceParam
```

Las marcas & y referenceParam son una variable por referencia y llaman al compilador como referenceParam y este corresponde al parametro de entrada, number y el parametro son el mismo.

Al igual que en C para C++ podemos utilizar variables por referencia dentro de una función especifica, utilizando el operador &.

Sobrecargando el Nombre de la Función.

La siguiente característica para discutir es, sobrecargando el nombre de la función, permite escribir varias funciones como parte del mismo nombre.

Suponemos que necesitamos una función que pueda imprimir el valor de una de sus variables, pueden ser de tipo long, short ó una cadena de texto. Se puede escribir una función que tenga cuatro parámetros:

```
Display( short whichType,
         long longParam,
         short shortParam,
         char *textParam );
```

El primer parámetro puede actuar como un switch, determinando cuál de los tres tipos fueron pasando e imprimiendose. El código main de la función puede observarse como:

```
if ( whichType == kIsLong )
     cout << "The long is:  " << longParam << "\n";
else if ( whichType == kIsShort )
     cout << "The short is:  " << shortParam << "\n";
else if ( whichType == kIsText )
     cout << "The text is:  " << text << "\n";
```

Otra solución es escribir tres funciones separadas, una para imprimir shorts, una para longs y una para cadenas de tipo texto:

```
void DisplayLong( long longParam );
void DisplayShort( short shortParam );
void DisplayText( char *text );
```

Los operadores New y Delete.

En C, la memoria asigna un llamado a `malloc( )` junto con un llamado a `free( )` cuando la memoria no necesita ampliarse. En C++, ocurre lo mismo para los operadores new y delete.

Se llama a new cuando se quiere asignar un bloque de memoria. Por ejemplo, el siguiente código asigna un bloque de 1024 caracteres:

```
char *buffer;
buffer = new char[ 1024 ];
```

new toma el tipo de un operando, asigna un bloque de memoria del mismo tamaño que el tipo, y regresa un apuntador al bloque. Regresa la memoria a el heap, usando el operador delete. El siguiente código libera la memoria asignada:

```
delete buffer;
```

new puede ser usado con cualquier tipo legal de C++, incluyendo los creados por nosotros. Son ejemplos:

```
struct Wobble

{
     short     papaWobble;
     short     mamaWobble;
     long littleBabyWobble;
};
short          *shortPtr;
long double    *longDoublePtr;
Wobble         *wobblePtr;
shortPtr = new short;
longDoublePtr = new long double;
wobblePtr = new Wobble;
```

Este es un ejemplo de un mal uso de *new*, garantizando la falla del compilador:

```
short     *shortPtr;
shortPtr = new 1024;     // Will not compile!!!
```

Los operadores new y delete de C++, funcionan de la misma manera que malloc( ) y free( ) para C. Estos operadores funcionan juntos usando la zona de memoria libre que queda entre el programa y la parte alta de la pila para establecer y mantener una lista de almacenamiento disponible.

El operador Scope.

La siguiente característica a examinar es el *operador scope* (::) de C++. El operador scope precede a la variable, indicando al compilador que observe fuera del bloque una variable del mismo nombre.

Suponemos que se declara una variable global y una variable local con el mismo nombre:

```
short number;
void main( void )
{
     short number;
     number = 5;     // local reference
     ::number = 10; // global reference
}
```

Dentro de *main( )*, la primera declaración se refiere a la definición local de number. La segunda declaración usa el operador scope para referirse a la definición global de number. Este código permite a number local con un valor de 5 y a number global con un valor de 10.

Este operador se utiliza en C++, para resolver conflictos entre nombres iguales definidos en una misma función; donde podemos tener definida una variable de tipo local y con el operador scope (::), podemos definir una variable de tipo global anteponiendo el operador a la variable. Este operador tambien es usado para conexión entre clases.

Funciones en linea.

Normalmente, cuando una función es llamada, el CPU ejecuta un conjunto de instrucciones que mueven el control desde la función llamando a la función llamada. Esta instrucción va al principio de la función y regresa a la dirección de return.

C++, proporciona *funciones en linea*, las cuales te permiten desviar estas instrucciones y salvar un bit de tiempo de ejecución. Así es como esta característica trabaja.

Cuando se declara una función usando el comando *inline*, el compilador copia el cuerpo de la función dentro del llamado de la función, haciendo la copia de una parte de la instrucción de la función llamada como si esta hubiera sido escrita de ese modo originalmente.

1.2.3 SURGIMIENTO DEL LENGUAJE VISUAL C++

Programación Visual.

El éxito de Windows en el entorno PC ha obligado a los programadores a reciclarse para poder desarrollar aplicaciones que respeten el "look" y las necesidades de esta interface gráfica. La llamada "programación visual" supone un paso más en la búsqueda de nuevas herramientas que faciliten el desarrollo de software para Windows.

Empresas como Microsoft y Borland se han apresurado a lanzar al mercado versiones de sus productos con las palabras "visual", pero todavía no está muy claro el significado de este término. Se prodría decir que "programación visual" se entiende como el proceso de desarrollo de forma interactiva del entorno de usuario de las aplicaciones. La idea es que la "programación visual" permite que la mayor parte del proceso necesario para crear cualquier aplicación se pueda realizar en forma automática.

Así por ejemplo, si la mayoría de las aplicaciones de Windows disponen de menús desplegables, parece razonable desarrollar una herramienta para crearlos sin tener que recurrir a páginas de código indescifrable. Pero los beneficios no acaban aquí, porque todo lo mencionado hace tiempo que estaba disponible en los editores de recursos, ahora se cuanta con mayor integración entre el desarrollo de aplicaciones propiamente dicha y el diseño de interface de usuario.

Los editores de recursos que actualmente podemos encontrar en el mercado (en los compiladores de Pascal o C para Windows siempre viene alguno) siguen estas pautas:

Primero se crea el recurso deseado -que puede ser una ventana, un menú, un icono, algún tipo de letra, un botón, un campo de texto o cualquier otro elemento típico de Windows-, después se genera el código correspondiente al recurso y, por último, con más o menos pasos intermedios, se acaba uniéndolo al resto del código fuente de nuestro programa como se ve en la figura 1.2.3.1.

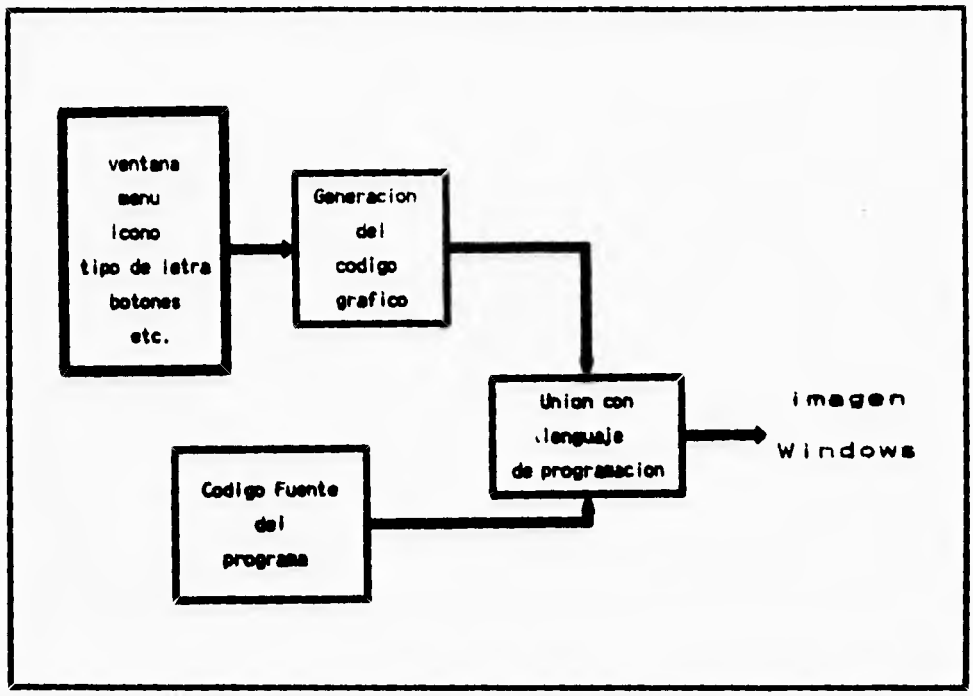


Fig. 1.2.3.1 Proceso para realizar una aplicación con imagen Windows.

Con la "programación visual" se pretende simplificar al máximo todo el proceso, evitando cuantos pasos intermedios sea posible. Lo normal es que se parta de la creación de un interface de usuario y se llegue a la escritura del código (si es necesario) en algún lenguaje de programación.

La creación de una aplicación para el entorno Windows podría ser el cuento de nunca acabar si no dispusiéramos de las herramientas adecuadas. Aunque desde luego todavia tendremos que escribir bastantes líneas de código y manejar el ratón con soltura, la "programación visual" facilita al máximo el desarrollo de aplicaciones para la interface gráfica de Microsoft.

Por el momento, la "programación visual" ha dado un paso de gigante en la tarea de facilitar la vida a los informáticos; sin embargo, todavía no ha llegado el día en el que

cualquier persona sea capaz de decirle a la computadora mediante un micrófono lo que quiere y que ésta lo haga. Hasta entonces tendremos que seguir trabajando con las herramientas disponibles, muchas de las cuales ya han comenzado su transición a la "programación visual".

Desde que la informática abandonó las tarjetas perforadas, la mayor preocupación de los programadores ha sido crear aplicaciones que "entrasen por los ojos"; es decir, que no sólo fueran fáciles de usar, sino también atrayentes. En pocos años hemos vivido una verdadera revolución en la manera de entender las aplicaciones; esta revolución ha venido de la mano de las nuevas y poderosas PCs, de los sistemas operativos y de periféricos tales como modems, impresoras o scáners.

Además, la proliferación de las interfaces gráficas de usuario -los llamados entornos de ventanas o GUI- no ha hecho más que acentuar el desfase que existía entre las técnicas de programación de los años ochenta y las actuales. Muchos han sido los desarrolladores que no han sabido adaptarse a los nuevos tiempos y se han quedado "fuera de juego"; sin embargo, aquellos que han actualizado sus conocimientos reconocen que ahora cuentan con mejores y más potentes herramientas para realizar su trabajo.

Al mismo tiempo, las herramientas de programación han evolucionado hasta convertirse en verdaderos entornos de desarrollo integrados que se basan en la misma filosofía de sencillez que gobierna los interfaces gráficos. En el campo de la integración de herramientas Borland fue pionera con su famoso Turbo Pascal, cuyas primeras versiones aparecieron para el sistema operativo CPM de la famosa familia de procesadores Z80 de Zilog. Por otra parte, Microsoft ha sido pionera con el concepto de "programación visual" con su popular Visual Basic, que ahora en su versión para Windows se ha convertido en un best-seller. Es en este punto, ahora que se menciona el término "programación visual", donde comienza una pequeña polémica.

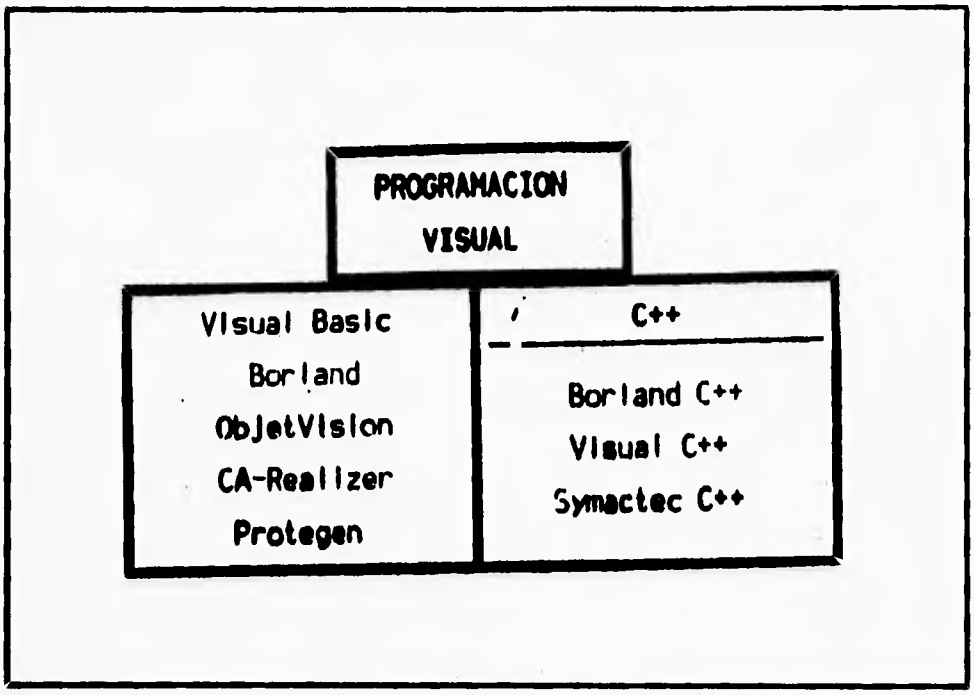


Figura 1.2.3.2 Paquetes y lenguajes que manejan la programación "Visual".

Difícil definición.

Anteriormente mencionamos que el término en sí era demasiado genérico y que procedía principalmente del nombre que las propias casas de software daban a sus IDE (*Integrated Development Environment*, entornos integrados de desarrollo) para lanzarlos publicitariamente con la idea de ser algo intuitivo y fácil de usar. Sin embargo, con la aparición de los ultimos productos de desarrollo de aplicaciones bajo entornos gráficos se va haciendo necesaria una terminología adecuada que los clasifique de alguna manera.

Un entorno integrado de programación es aquel que permite desde un único programa realizar las tareas más habituales de desarrollo y depuración de una aplicación (incluido en algunos casos la optimización y realización automática de pruebas). Si el enrono

integrado es de "programación visual", estamos hablando de un producto que integra una serie de herramientas adicionales a las habituales y que son necesarias para el desarrollo de programas bajo una interface gráfica de usuario, todo ello con la mínima intervención del programador en las tareas más repetitivas y, por tanto, más automatizables.

Podemos decir que la "programación visual" se sitúa directamente por debajo de las llamadas herramientas CASE (Computer Aided Software Engeneering). Los programas CASE permiten realizar el sueño (o pesadilla) de todo programador: pasar directamente del diseño de una aplicación (diagramas de flujo, estructuras de datos...) a producir la aplicación propiamente dicha. La pesadilla se puede hacer realidad para los programadores cuando llegue el día en que cualquier usuario normal pueda crear, gracias a estas herramientas, su propia aplicación, prescindiendo, por lo tanto, de los servicios de un profesional.

Programar la interface

Con estas premisas, está claro que la "programación visual" ha pasado de ser una curiosa novedad metodológica, a constituirse como un conjunto de herramientas imprescindibles para el desarrollo de las complejas aplicaciones basadas en GUIs que se comercializan actualmente.

La complejidad de programar bajo un interface gráfica proviene del propio diseño de los entornos. De hecho, desarrollar el programa más sencillo para un entorno gráfico puede convertirse en una tarea desquiciante si no se dispone de las herramientas adecuadas.

Para afrontar el desarrollo de una aplicación para una interface gráfica (Windows u OS/" en el mundo PC) podemos seguir dos caminos. El primero consiste en armarnos de paciencia y enfrentarnos con los conceptos básicos del entorno, lo cual resulta una tarea tediosa y poco gratificante porque a veces no es efectiva ante las funcionalidades ya existentes (OLE) o los nuevos sistemas que se lanzarán en breve (por ejemplo, Windows Chicago). El segundo camino se simplifica al utilizar las mencionadas herramientas de "programación visual".

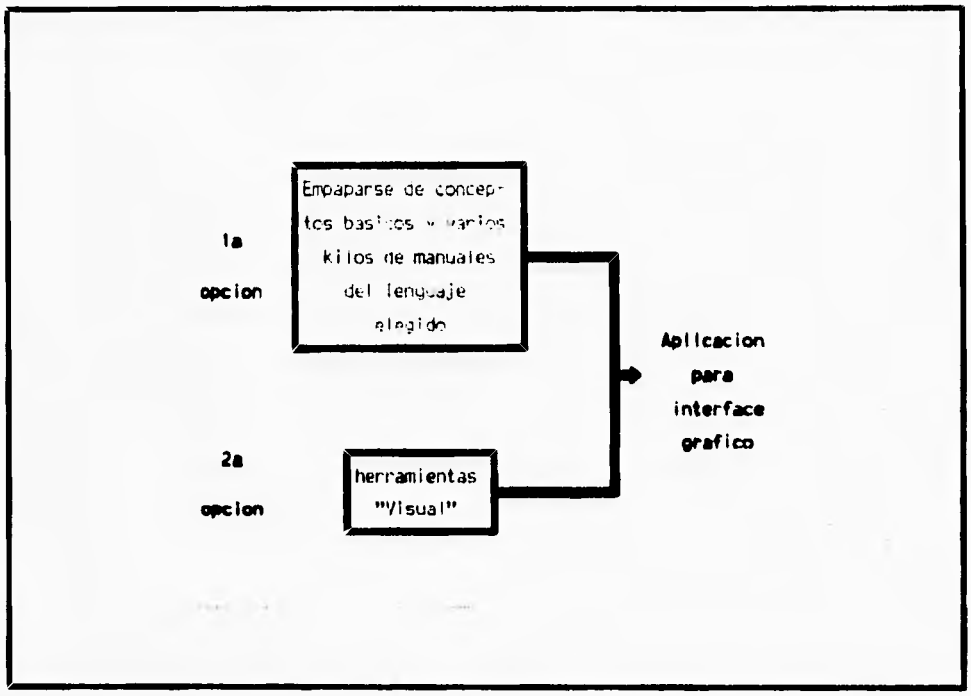


Fig. 1.2.3.3 Desarrollo de una aplicación para una interface gráfica,

Estos compiladores simplifican el diseño al proporcionarnos formatos, tipo o patrones que podemos aplicar a nuestras propias aplicaciones, al mismo tiempo que ofrecen una elevada fiabilidad porque se tratan de patrones probados en multitud de otros desarrollos. Todo ello redunda en una mayor eficiencia a la hora de programar y en un gran ahorro de tiempo de codificación.

Visual en C.

Hasta hace poco los únicos que podían disfrutar de este tipo de ayuda en su trabajo eran los programadores de Basic y Pascal gracias al Visual Basic de Microsoft y al Turbo Pascal para Windows de Borland. Sin embargo, sucede que la mayoría de los programadores profesionales prefieren el lenguaje C antes que el Basic o el Pascal. Su sencillez y poca rigidez sintáctica hacen que la construcción de compiladores sea muy sencilla, lo que facilita la portabilidad y la eficiencia de éstos, que llegan incluso a optimizar el código introducido por el programador. Se puede decir sin temor a equivocarse que existen compiladores de C para TODAS las combinaciones posibles de microprocesadores, arquitecturas de hardware y sistemas operativos.

El C++ fue cuidadosamente desarrollado con el objetivo de que cualquier programador de C pudiera aprender a utilizarlo fácilmente, como un intento más de proporcionar un estándar práctico para la programación orientada a objetos, que se convirtiera en algo más popular que lenguajes casi experimentales como el SmallTalk u otros muy particulares como el Ada.

Si el C++ ha sido una evolución natural del C hacia la programación moderna, lo que aquí llamamos "programación visual" podría ser la evolución de los entornos de programación integrados para el lenguaje C que hasta hace poco no habían abandonado el modo texto. Quede claro que no estamos hablando de una innovación en el campo del lenguaje propiamente dicho, sino de una inevitable evolución hacia formas más cómodas, y a la vez más seguras, de programar, gracias a la facilidad de uso de las interfaces gráficas.

Microsoft Visual C++

Aunque el lenguaje favorito de Microsoft es el Visual Basic para aplicaciones, son muchos los programadores que por muchos motivos prefieren el C o Pascal para sus aplicaciones. Para estos usuarios existe una herramienta muy poderoso "Microsoft Visual C++".

Hasta hace unos años, Microsoft no había adoptado la filosofía de los entornos integrados de programación en sus productos. Sus compiladores se invocaban dedes la línea de comandos, teniendo que incluir multitud de parámetros para conseguir los resultados deseados. Sin embargo, su entrada en este campo ha sido espectacular.

El nacimiento de Windows supuso un paso decisivo en general hacia el desarrollo de herramientas totalmente integradas e intuitivas, pero el más beneficiado en este aspecto ha sido Microsoft. Casi nadie esperaba que alcanzaría a la mismísima Borland -pionera en ambientes bajo DOS- en cuanto a potencia y facilidad de uso en sus entornos integrados de programación.

Visual C++ supone la competencia que Microsoft opone a Borland en este lenguaje que cuenta con tantos adeptos entre los programadores. Se supone que el inventor de Windows debe disponer de los mejores lenguajes de programación para las aplicaciones de este entorno.

1.2.3.1 ¿PORQUÉ VISUAL C ++ ?

El lenguaje visual C++ equivale a dos lenguajes Windows completos, en un solo paquete, los modelos de programación en Windows y los componentes de C++ trabajan juntos para realizar mejor las aplicaciones en un ambiente Windows.

Procesamiento de mensajes

na gran diferencia entre los programas elaborados en C a los programas elaborados bajo Windows en C++ , es que los programas en lenguaje C llaman al sistema de operaciones para utilizar una salida, en cambio Windows utiliza las salidas a través de un mensaje.

Interface de dispositivos graficos (GDI)

Los programas elaborados en C son escritos directamente a la memoria de video y al puerto de impresión, la desventaja de esta técnica es que necesita un "driver" de software para la tarjeta de display y para la impresora. Windows y C++ introdujeron un nivel de abstracción llamado interface de dispositivos gráficos, para que así el programa no necesite conocer el tipo de tarjeta de display y que tipo de dispositivo de impresión se tiene. En lugar de localizar el hardware el programa llama a la función GDI la cual hace referencia a una estructura de dato llamada dispositivo de contexto, Windows convierte el dispositivo de contexto a un dispositivo fisico y establece las instrucciones apropiadas de entrada/salida. El GDI es tan ràpido como el acceso directo a video y permite diferentes aplicaciones mientras se escribe.

Programando las bases

Para programar los datos en lenguaje C se tienen que codificar los datos inicializandolos como constantes, o se debe dar datos separados para que pueda leer el

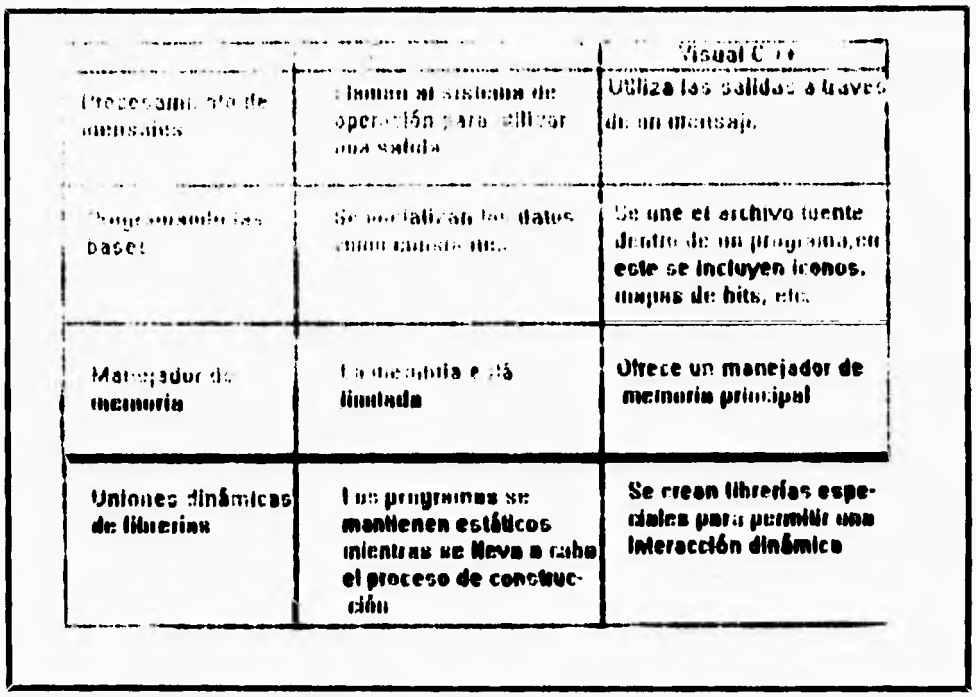
programa. Cuando se programa en Windows y C++, se colocan los datos en un archivo fuente utilizando varios formatos, Windows une el archivo fuente dentro de un programa mediante un proceso llamado "binding". En el archivo fuente pueden estar incluidos mapas de bits, iconos, definiciones del menu. Con el editor de programas del Visual C++ se pueden editar la mayoría de los formatos.

Manejador de Memorias

La memoria convencional en el MS-DOS estaba limitada a 640 Kilobytes limitando el tamaño de los programas, se pueden utilizar tecnicas para expander memoria y así permitir porgramas más largos pero ha la larga va ha ser insuficiente. Windows junto con C++ ofrece un manejador de memoria principal, el resultado es que la memoria ya no es un problema. Simplemente se coloca la memoria que se necesita y Windows se encarga de lo demás; ejecuta el programa, administra los recursos, automaticamente lo cambia al disco y despues lo coloca en memoria fisica,los cambios son tan buenos que la computadora va a tener mucha memoria disponible.

Uniones Dinamicas de Librerias (DLLs)

n los programas comunes todos los modulos de objetos en los programas se mantienen estaticos mientras se lleva a cabo el proceso de construcción. Windows y C++ permiten una interacción dinámica, lo cual quiere decir que librerías especialmente construidas pueden ser cargadas e interactuadas mientras se está corriendo un programa. La interacción dinamica incrementa la modularidad del programa debido a que se puede compilar y probar los DLLs separadamente. Algunas de las ventajas que estas librerías tienen es que se puede crear una propia, además de utilizar tanto iteracciones dinámicas como estáticas.



		Visual C++
Procesamiento de mensajes	Llama al sistema de operación para utilizar una salida	Utiliza las salidas a través de un mensaje.
Programación de bases	Se accesan los datos como conjuntos	Se une el archivo fuente dentro de un programa, en este se incluyen iconos, mapas de bits, etc.
Manejador de memoria	La memoria es más limitada	Ofrece un manejador de memoria principal
Uniones dinámicas de librerías	Los programas se mantienen estáticos mientras se lleva a cabo el proceso de construcción	Se crean librerías especiales para permitir una interacción dinámica

Fig. 1.2.3.1.1 Ventajas de Visual C ++.

Cuando Microsoft cargo el C++, Windows estaba en el último estado de prueba, este nuevo sistema (C++) tiene avanzados sistemas de archivo con multilectura, con partes de seguridad, permite el acceso a las redes. Windows puede correr tanto el Windows basado en una aplicación de 16 bits, como en la nueva versión basada en una aplicación de 32 bits, la cual está desarrollandose, utilizando también la aplicación del sistema C++. Se pueden utilizar muchas herramientas del Visual C++ para convertir Windows a un estilo de programación más fácil.

El Worckbench de Visual es un huesped de Windows que directamente desciende del Microsoft Quick C para Windows. El Visual C++ esta disponible en dos versiones: La edición Estandar y la edición Profesional.

		Visual C++
Procesamiento de mensajes	Llama al sistema de operación para utilizar una salida	Utiliza las salidas a través de un mensaje.
Programación base	Se serializan los datos como clase de...	Se une el archivo fuente dentro de un programa, en este se incluyen iconos, mapas de bits, etc.
Manejador de memoria	La memoria está limitada	Ofrece un manejador de memoria principal
Uniones dinámicas de librerías	Los programas se mantienen estáticos mientras se lleva a cabo el proceso de construcción	Se crean librerías especiales para permitir una interacción dinámica

Fig. 1.2.3.1.1 Ventajas de Visual C ++.

Cuando Microsoft cargo el C++, Windows estaba en el último estado de prueba, este nuevo sistema (C++) tiene avanzados sistemas de archivo con multilectura, con partes de seguridad, permite el acceso a las redes. Windows puede correr tanto el Windows basado en una aplicación de 16 bits, como en la nueva versión basada en una aplicación de 32 bits, la cual está desarrollandose, utilizando también la aplicación del sistema C++. Se pueden utilizar muchas herramientas del Visual C++ para convertir Windows a un estilo de programación más fácil.

El Worckbench de Visual es un huesped de Windows que directamente desciende del Microsoft Quick C para Windows. El Visual C++ esta disponible en dos versiones: La edición Estandar y la edición Profesional.

Con el original Windows SDK se tienen herramientas que editan dialogos, mapas de bits y fuentes; con el Visual C++ se puede utilizar el editor App Studio el cual incluye tanto menu de "lo que tu ves es lo que tu tienes" y una caja de dialogos que es mucho muy superior a los dialogos elaborados en el Windows SDK, Otra ventaja es que cuando se utiliza App Studio se puede incertar los controles básicos del visual dentro de las cajones de dialogos, para despues poderlos conectar con el sistema C++. App Studio fué elaborado utilizando las herramientas de C ++, el cual tambien es capaz de elaborar sus propios recursos.

El Compilador C/C++

El compilador Visual C ++ puede procesar tanto los recursos del C como los recursos del C++, determina el lenguage observando el tipo de codigo que tiene la extensión del nombre del archivo. El compilador funciona también con el ANSI versión 2.1 y tiene una extensión de Microsoft adicional.

El Enlace

Para generar un archivo EXE, el Visual C++ procesa los archivos OBJ que el compilador produce. Para realizar el enlace se utiliza la librería de Microsoft para un mejor modelo de memoria.

Los recursos del Compilador

os recursos del compilador del Visual C++ operan tanto en el modo de compilador como en el modo de enlace. En el modo de compilador un archivo en codigo ASCII del App Studio es compilado dentro de un archivo binario RES. En el modo de enlace un archivo RES es unido a un archivo ejecutable (EXE).

El Debugger

Si el programa trabaja la primera vez, no se necesita el debugger. El debugger del Visual C++ fue el primer huesped del Windows. Trabaja junto al Worckbench del Visual para asegurar con esto que los puntos de depuración sean salvados en disco.

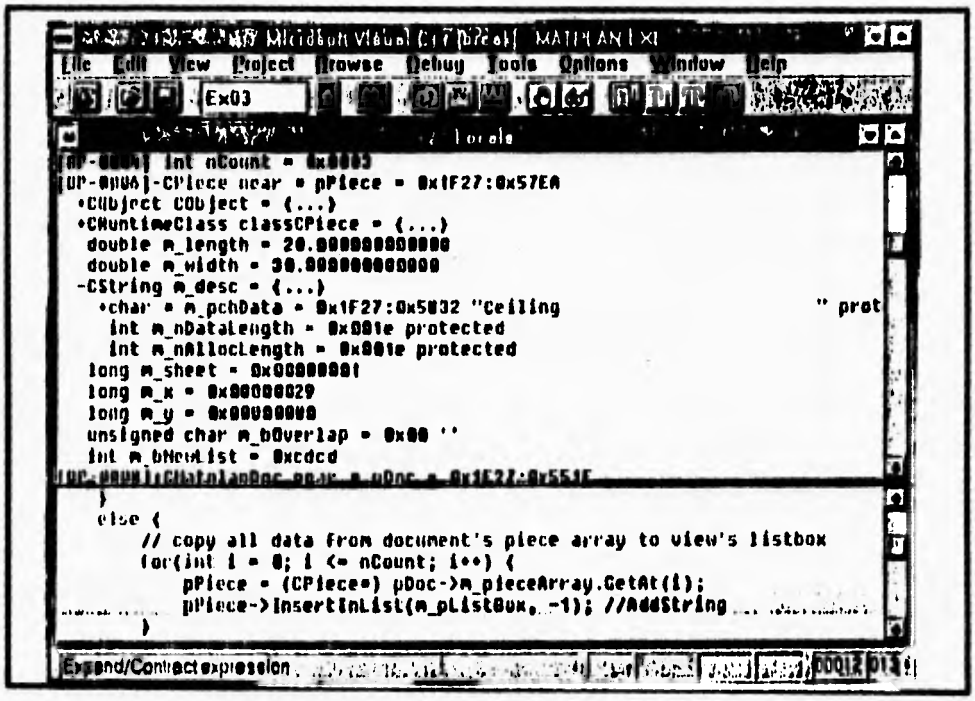


Fig. 1.2.3.1.3 La ventana del debugger de Visual C++.

Otra ventaja que tiene el Visual C++ es que con la Edición Profesional se puede tener un modo tipo caracter para el debugger en lugar de utilizar el debugger de Windows.

Derivaciones de Clases

Visual C++ provee la habilidad de definir subclases, al describir como una nueva clase difiere de una vieja superclase. Reduce el tiempo del compilador y deja más espacio de memoria.

Los Recursos

Si se está escribiendo una aplicación de algún borrador, probablemente se debe de tener una buena memoria del dibujo original, de los codigos de archivo, de las clases, de las funciones, etc; pero si se está utilizando aplicaciones de otros recursos va ha llegar el momento que se necesite ayuda. El Visual C++ permite examinar una aplicación de una clase o de una función en lugar de examinarla de todo el archivo; esto se conoce con el nombre de "browser", el Browser tiene diferentes modos de examinado:

1.- Definiciones y referencias.- Se selecciona cualquier función, variable, tipo, macro o clase y despues se ve como está definida para usarla en cualquier proyecto.

2.- Llamado Grafico.- De un grupo de funciones se obtiene una representación gráfica de las funciones llamadas o las funciones que lo llamaron.

3.- Derivaciones de clases graficas/Base de clase gráficas.- Estas son gráficas de clase con diagramas jerárquicos. De una clase seleccionada se pueden ver las derivaciones de las clases. Se podrá controlar la expansión jerárquica con el mouse.

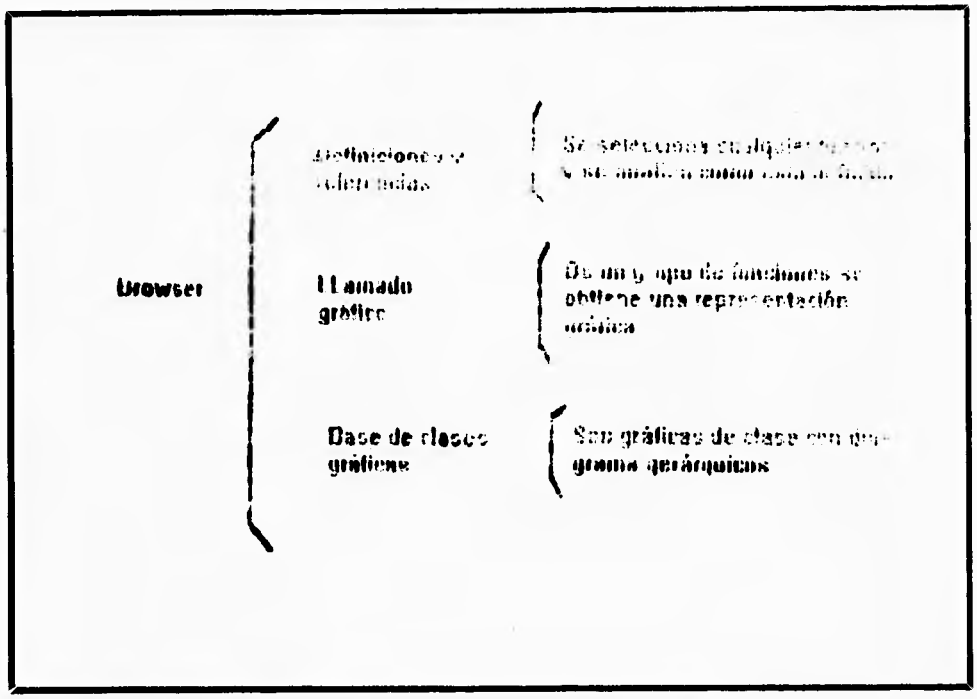


Fig. 1.2.3.1.4 Los recursos.

Lineas de Ayuda

Las líneas de ayuda también están disponibles en el Visual C++, tambien está disponible en el App Studio. Si se quiere obtener ayuda de una función simplemente se debe dar un click sobre la funcion en el Worckbench Visual y presionar F1, se verá una ventana de ayuda, como la que se muestra en la figura 1.2.3.1.5.

La ayuda de Visual C++ es util en la resolución de conflictos provocados entre la similitud que existe entre los nombres de las funciones del Windows SDK y las funciones que existen en la librería de Microsoft. Si se ha seleccionado una función que corresponde a muchas funciones en muchas clases, se podrá escoger la que mejor convenga de una lista. Si lo que se quiere es ayuda sobre una clase, se verán muchas funciones y datos en una lista colocada en orden funcional.

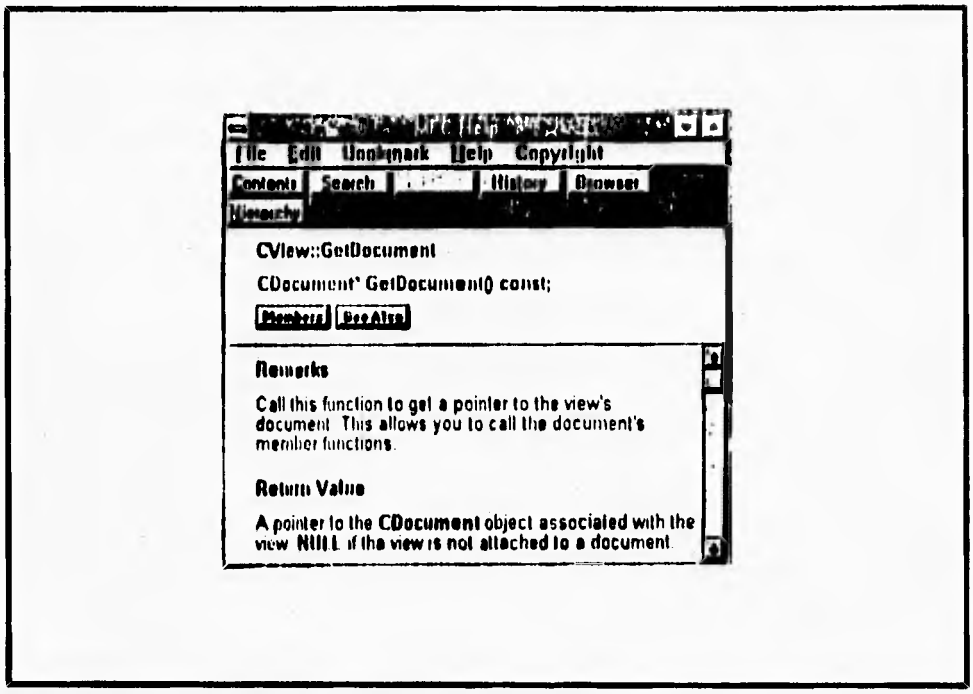


Fig. 1.2.3.1.5 La ventana de Ayuda de Visual C++.

Herramientas de Diagnostico de Windows

La Edición Profesional del Visual C++ contiene las mismas herramientas que están incluidas en el Windows SDK, cuando era un producto separado:

- **SPY** para observar los mensages de Windows,
- **HEAPWALK** para examinar la memoria,
- **HC31** para compilar los archivos de ayuda y
- **STRESS** para limitaciones artificiales de memoria.

Tanto la Edición Estandar del Visual C ++ como la Edición Profesional incluyen la utilidad DBWIN, la cual pone en pantalla el diagnostico de las salidas, y el programa NMAKER que procesa los archivos. El NMAKE es usado para versiones noestandard de la librería de Microsoft.

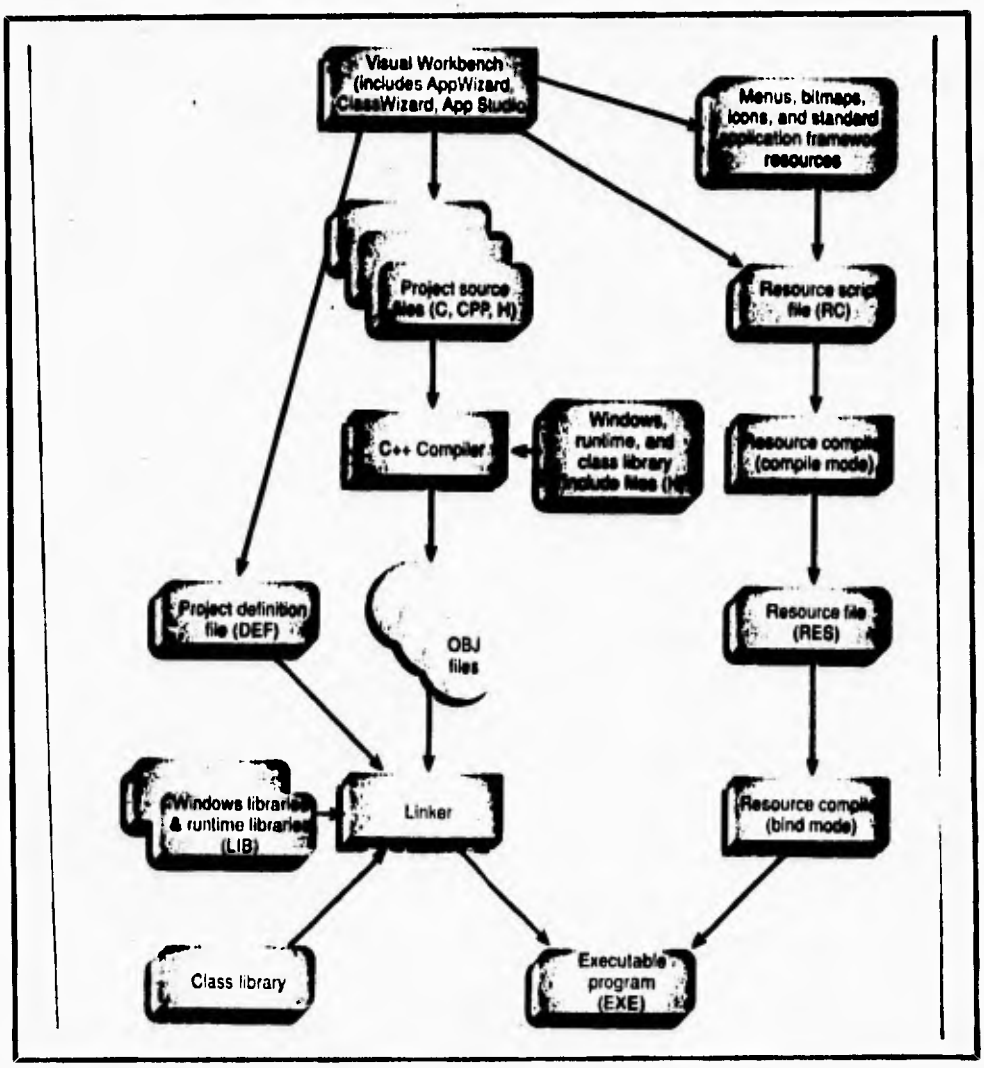


Fig 1.2.3.1.6 Proceso de Construcción de una aplicación con Visual C++.

OLE

Una de las opciones más importantes dentro de la definición de nuestro proyecto en el "AppWizard" es la incorporación de OLE (*Object Linking and Embedding*, unión e inclusión de objetos) pudiendo nuestra aplicación adoptar los papeles de cliente, servidor o ambas cosas. Esto permite el intercambio transparente y avanzado de archivos con otras aplicaiones que se acojan a las especificaciones OLE. No menos importante es la posibilidad de utilizar la ODBC (*Open DataBase Connectivity*, conectividad abierta de bases de datos) para manejar archivos y cabeceras de los programas de bases de datos más utilizados.

Una vez definidos todos los parámetros de nuestro proyecto, el "AppWizard" nos permitirá visualizar qué archivos van a ser creados y nos mostrará un resumen de las opciones que hemos incorporado. Estos archivos serán, básicamente:

- un archivo *make* (extensión .mak);
- un archivo -clw, que será usado por el "ClassWizard" para la edición de las clases;
- un archivo .def, que contendrá la información que se debe pasar a Windows para que el programa funcione correctamente;
- un archivo .rc, que almacenará los recursos que hemos incluido en muestra aplicación (tales como barras de menús o ventanas de diálogo), y que podremos modificar con el "AppStudio";
- Un archivo .cpp, que contiene el código fuente del programa;
- y un archivo .h que contiene las cabeceras necesarias para su funcionamiento.

Además, se crea un subdirectorio **RES**, que contiene otros recursos como iconos o recursos no editables por el "AppStudio". Y además un archivo README. TXT, que contiene información sobre los archivos que se han creado.

Más Magos

Una vez realizados los primeros pasos en el desarrollo de una aplicación, se intentará conseguir que se comporte como tengamos previsto. Para ello, se editarán directamente los archivos fuente. Se trata de un proceso indispensable para incluir nuestras rutinas y utilizar las herramientas de que disponemos.

Una de las herramientas que se utilizan en el "ClassWizard", que nos permite visualizar en una ventana todas las clases que incluye nuestro programa. Esto hará posible editar en todo momento el código fuente correspondiente a la definición de una clase con sólo hacer un doble-clic con el ratón. Esto resulta especialment útil, ya que el manejo de OLE y de archivos de bases de datos se ha organizado mediante la definición de clases especiales. Así, se pueden añadir o modificar las funciones contenidas en ellas con bastante facilidad.

Recursos

Otra herramienta para ir dando forma a una aplicación en el "AppStudio", que nos permitirá editar y crear los recursos que se han incluido en el proyecto. El "AppStudio" ha sido desarrollado pensando en dotar al programador de una herramienta muy intuitiva y fácil de usar. Los tipos de recursos que podemos manejar son: teclas rápidas o aceleradores, ficheros de mensaje (especialmente útiles para traducir mensajes), ventanas de diálogo, menús y gráficos (iconos, cursores o bitmaps).

La edición de las ventanas de diálogo y de los menú se realiza sobre el resultado final; es decir, en todo momento nos movemos dentro de la ventana que se va a visualizar, pudiendo mover elementos o editar sus propiedades simplemente mediante el doble click del ratón. Una vez que hayamos hecho una modificación, ésta aparecerá inmediatamente en la ventana, visualizando en la pantalla el aspecto que va tener dentro de la aplicación.

La edición de bitmaps y de iconos utiliza un pequeño pero sofisticado editor gráfico que nos permite incluir otros archivos de imágenes mediante enlace OLE, o simplemente copiando y pegando desde otras aplicaciones.

Cuando se editan las características de un recurso cualquiera, podemos dejar activa la ventana que visualiza los datos correspondientes a una zona determinada. por ejemplo una opción de menú. De este modo, nos podremos mover dentro de la ventana y, según sea la zona a la que estamos apuntando, irán cambiando los parámetros y podremos modificarlos.

También es posible invocar el "ClassWizard", pues los recursos están definidos mediante clases. Posteriormente, se pueden editar de la manera que ya se mencionó.

QuickWin

Visual C++ permite producir aplicaciones que se ejecuten en una ventaja de Windows, cuyas funciones de entrada/salida sean idénticas a los estándares para DOS (a saber: (print(), scanf(), y para C++ cin y cout). Esta posibilidad dota a las aplicaciones desarrolladas para dos de una portabilidad inmediata, ya que no se hacen llamadas directas al BIOS.

La última herramienta que puede resolvernos más de un problema es el depurador, que también está integrado con el resto de las herramientas. Se trata de una versión "Codeview" que incorpora todas las funciones básicas de un depurador, aprovechando las posibilidades del entorno. Con el depurador se puede ejecutar el programa paso a paso, inspeccionar variables, expandirlas si son estructuras de datos, y hasta detener otras aplicaciones Windows mientras el depurador está funcionando.

La biblioteca Foundation Class en Visual C++.

El paso siguiente de Microsoft C/C++7 y su biblioteca Foundation Class se presenta con Visual C++ en donde su principal característica es su interface gráfica para manejar todo

bajo Visual,, o sea ambiente Windows, con bibliotecas más potentes y grandes, todo bajo la programación orientada a objetos.

Microsoft en Visual C++ nos presenta la versión 2.0 de la biblioteca Foundation Class con aplicación en framework, que constituye su herramienta principal.

Estos son los beneficios de la biblioteca de clases:

Biblioteca Foundation Class versión 2.0 de el C++ Microsoft Windows API. La aceptación de este depende de su comprensión del Lenguaje C++. Si se tiene, entonces, será de manera natural para Windows tener una interface de programación C++.

Aplicaciones framework en la estructura estándar. Cualquier programador que empieza un gran proyecto de desarrollo piensa en la estructura del código. El problema es que cada estructura del programador es diferente y para un nuevo equipo de trabajo es difícil aprender la estructura. La biblioteca Foundation Class con aplicación de framework incluye su propia estructura de aplicaciones que han sido probadas en muchos ambientes de software y en muchos proyectos. Si se escribe un programa usando la biblioteca de clases se puede enviarlo a cualquier parte del mundo sabiendo que su subordinado puede fácilmente mantenerlo y aumentar el código.

Pero para nada las estructuras de la biblioteca de clases es inflexible para realizar otros programas. Con la biblioteca de clases sus programas pueden realizar cualquier programa que Windows SDK puede hacer.

Las aplicaciones framework son pequeñas y rápidas. Función por función los programas de la biblioteca de clases son casi igual de pequeños que los programas de Windows SDK. Al igual la velocidad en algunas circustancias es actualmente más rápida que su equivalente Windows SDK.

La biblioteca de clases de framework tiene valiosas características. La bilblioteca Foundation Class versión 1.0 con que cuanta la versión Microsoft C/C++ versión 7.0, donde esencialmente se programa con una interface para Windows que contiene:

- Clases de propósito general (no Windows necesariamente) incluyendo:
    - Coleccion de clases para listas, arrays, y mapas.
    - Una útil y eficiente clases de cadenas.
    - Tiempos, duración de tiempos y clases de fechas.
    - Clases de acceso a archivos para operación de sistemas independientes.
    - Soporte para almacenar y recuperar objetos sistemáticos y de disco.
    - Jerarquía de clases.
    - Actualizado soporte de aplicaciones para interface MDI (Multiple Document Interface).
    - Efectivo soporte para OLE (Object Linking and Embedding).

La biblioteca Foundation Class versión 2.0 de Visual C++ conserva muchas características de la versión 1.0 que se encuentran basadas en aplicaciones Windows. Con la aplicación de la arquitectura framework se suman importantes carácterísticas nuevas:

- Gran soporte para herramientas de menú como: *abrir Archivo, Salvar, y Salvar como;* con la más recientemente lista de archivos.
- Presentación preliminar y especificar impresora.
- Movimiento iterativo de pantallas y partición de pantalla.
- Barra de herramientas y control de barras.
- Acceso a los controles de Microsoft Visual Basic.

- Ayuda
- Procesamiento automático de entrada de datos en ventana de diálogo.
- Fácil programación para la interface OLE.
- Soporte DLL.

Las herramientas de Visual C++ reducen el código complicado. "AppStudio", "AppWizard", y "ClassWizard" estos reducen significativamente el tiempo necesario para escribir el código de una aplicación. Por ejemplo, "AppStudio" crea archivos de encabezados que contienen valores asignados para constantes· *#define*. "AppWizard" genera la estructura de código para una aplicación completa, y "ClassWizard" genera prototipos y cuerpos de funciones para mensajes de encabezado.

Una aplicación framework es " una colección integrada de componentes de software orientado a objetos que ofrece todas las necesidades para generar una aplicación". Una aplicación framework es superior a una biblioteca de clases. Una biblioteca de clases ordinaria está incomunicada del diseño de clases para ser incorporada dentro de cualquier programa, pero una aplicación framework define la estructura del programa en sí mismo. Windows desarrolla además bibliotecas de clases incluyendo Microsoft Foundation Class Library versión 1.0, Borland OWL, y Microsoft Foundation Class Library versión 2.0, que son considerados aplicaciones frameworks.

Sin embargo Microsoft Foundation Class Library versión 2.0 de Visual C++ ofrece significativamente más características que los otros sistemas.Con las aplicaiones framework se puede decir que C++ contiene lo necesario para llamarse Visual.

No hay duda que el Microsoft Visual C++ es una herramienta de programación muy potente, y que se ha conseguido un compilador capaz de realizar aplicaciones muy sofisticadas con cierta facilidad, gracias a la última versión de las librerías de clases de Microsoft, las bibliotecas Foundation Classes 2.0.

Lo más sobresaliente el producto es la posibilidad de incluir las especificaciones OLE dentro de nuestras propias aplicaciones. OLE se está convirtiendo en una opción casi indispensable para cualquier aplicación que necesite un intenso intercambio de datos con otras aplicaciones sin perder la transparencia del característico "pinchar y arrastrar" de Windows.

También hay que mencionar las librerías ODBC que nos permiten crear fácilmente aplicaciones compatibles con archivos Access. Btrieve, dBase, Excel, SQL y la mayoría de las bases de datos del mercado con la facilidad que proporciona el entorno.

1.3 Metodologías de la programación orientada a objetos en C++

1.3.1 METAS FUERZAS Y DEBILIDADES.

En los puntos anteriores hemos podido ver las características principales de C++,que lo hacen un lenguaje "adulto" en lo que se refiere al manejo de estructuras de datos. En esta sección se discutirá las fuerzas y debilidades de este potente lenguaje de programación. Nosotros examinaremos las mayores ventajas del lenguaje, y sus desventajas; real y potencial (figura 1.3.1.1).

Ventajas

Las mejores ventajas de C++ son:

- **La gran adaptabilidad para el lenguaje C**. El lenguaje C++ como hemos visto es diseñado siguiendo las normas ANSI (*American National Standards Institute*) de estandarización (*Kernighan y Ritchie, 1988*). Cualquier compilador C++ acepta programas estándar realizados en C. Todas las extensiones del lenguaje garantizan guardar compatibilidad con el lenguaje C existente con un mínimo de diferencias. Por ejemplo, para soportar la definición de clases, C++ extiende la definicón de la construcción *struct*.

- **Buena ejecución**. Una de las desventajas originales de la tecnología orientada a objetos fué su ejecución. Las primeras implementaciones de lenguajes orientados a objetos como Smalltalk tuvieron pobres resultados de funcionamiento. C++ no sufre problemas de ejecución asociados con las primeras implementaciones de lenguajes orientados a objetos. Algunas publicaciones demuestran que el tiempo de compilación del codigo C++ es mejor que el C estándar (Zortech,1989). Entonces se dá aquí un golpe al pobre funcionamiento, con funciones virtuales, desde que el compilador C++ los implementó con un nivel extra de indexación.

- **Popularidad y Soporte de Distribuidor.** C++ ha venido a ser el lenguaje más popular para la programación orientada a objetos. Su popularidad es mayor a la de Smalltalk y otros lenguajes de programación como el Objective C (*Cox 1986*), Eiffel (*Meyer, 1988*) o Object Pascal (*Schmucker,1986*). Un gran número de casas de software tienen a la venta compiladores de C++, preprocesadores y ambientes de desarrollo, incluyendo AT&T, Apple, Sun, Zortech, Oasys, Glockenspiel, and Oregon Software, lanzándose con entusiasmo a apoyar la programación orientada a objetos.

Desventajas

Las desventajas de C++ son:

- **Lenguaje orientado a objetos híbrido**. C++ no es un lenguaje orientado a objetos puro, por lo tanto permite un desarrollo con los tradicionales habitos de codificación C. El diseño de clases puede proveer acceso completo a las variables muestra de las clases. Por lo tanto, esto significa que las definiciones de clases C++ no necesitan respetar la información de los encabezados. El desarrollo lleva esta responsabilidad. Esto en los principios de la orientación a objetos puede ser ignorado.

- **Manejador manual para acumulación de basura**. El lenguaje no soporta automáticamante la colección de basura. Algunos lenguajes orientados a objetos como el CLOS y el Smalltalk soportan automaticamente la colección de basura.

  El manejador manual de colección de basura cambia la carga del objeto a la orilla del proceso, en vez de que lo haga el sistema. Así el proceso puede alterar al manejador de memoria para la localización y recuperación de objetos en el tiempo de ejecución. Esto requiere de cierto número de codigo de aplicación dedicado a este problema. El problema fundamental es la "semántica dual" de el lenguaje de programación de alto nivel, con la implementación de construcciones para el manejo de memoria de bajo nivel.

  Otros lenguajes proveen subsistemas de manejo de basura que trabajan en el tiempo de ejecución para manejo de objetos creados por el sistema y por el usuario.Sin embargo, mientras el controlador automático de basura libera al proceso de esta responsabilidad, el tiempo de ejecución y el código asociado con ello puede ser muy grande. Con algunas aplicaciones en tiempo real el costo de tiempo de ejecución puede ser inaceptable.

- **Falta de herramientas de desarrollo**. No cuenta con muchos ambientes de desarrollo (debugger simbólico, browsers, y cosas por el estilo).

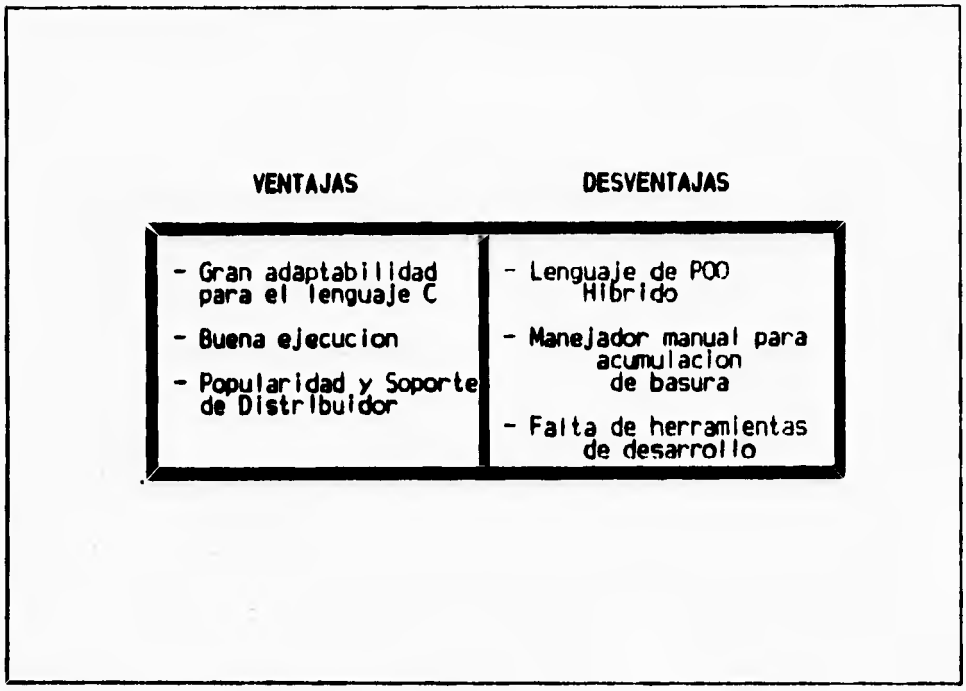


Figura 1.3.1.1 Ventajas y desventajas del lenguaje C++.

Metas

Actualmente algunas de las desventajas que tenían los primeros compiladores de C++ han desaparecido, como es el caso de falta de herramientas de desarrollo. Como hemos visto en los puntos anteriores C++ cuenta en sus versiones para Borland y Microsoft con un gran número de utilerias, incluso con librerías para ambientes DOS, OS/2 y Windows. En el caso de Microsoft tenemos actualmente su versión visual con lo que trata de lograr una programación más intuitiva.

Aúnque con sus nuevos compiladores C++ tiene un nuevo problema, que es el tiempo de compilación del gran número de librerías para una aplicación gráfica.

Sin embargo C++ y sus nuevos compiladores visual son actualmente los reyes de la programación.

Aún así, no hay lenguajes de programación perfectos. Diferentes problemas de programación requieren soluciones diferentes. La tarea es saber elegir el lenguaje idóneo para un proyecto. En un proyecto, es ésta una de las primeras decisiones que se deben tomar y será casi irrevocable desde el momento en que comience a codificar. La elección de un lenguaje de programación puede marcar la diferencia entre el éxito o el fracaso de un proyecto.

1.3.2 ENCAPSULACION, HERENCIA Y POLIMORFISMO.

ENCAPSULACION

El encapsulado protege del cambio a los usuarios de clases. Al proporcionar una especificación explícita para el empleo de un objeto de cierto tipo, los usuarios de clase y quienes la instrumentan establecen un acuerdo que define la responsabilidad de cada uno. El usuario se compromete a manipular los objetos sólo en una forma consistente con la especificación de la interfaz. El constructor garantiza que, siempre y cuando el usuario sólo se comunique con los objetos a través de la interfaz pública, los objetos siempre producirán los resultados esperados.

Algunos escritores han llegado al punto de llamar a este acuerdo un "contrato" , haciendo referencia al usuario como el "cliente" o "comprador" y al constructor como el "vendedor" o "agente". Esta metáfora muestra poco sobre las relaciones establecidas alrededor de la construcción y el uso de clases y objetos. Sin importar cuáles sean los nombres que se empleen, lo principal que debe tenerse en cuenta es la razón para distinguir entre el papel del comprador y el del usuario.

El encapsulado propicia una separación flexible de las preocupaciones, entre el constructor y el usuario. Esta separación de preocupaciones es posible a través de una interfaz pública a un objeto que puede hacer obligatoria un lenguaje de programación. Aunque es agradable que el usuario pueda contar con cierto comportamiento con base en una especificación de clase, el constructor es quien adquiere más libertad a partir de la relación.

El concepto de acceso público a funciones como una interfaz a módulos de programa no es exclusivo de la programación orientada a objetos. Ocultar datos y encapsular a través de módulos y archivos separados de código fuente se practica mucho en C.

El usuario puede construir una clase ficticia que simule los objetos con los que está trabajando el constructor. Esto permite al usuario avanzar y en realidad compilar, enlazar y ejecutar código que crea objetos y envía mensajes a los objetos que con el tiempo instrumentará el constructor. El compilador verifica que el usuario no haya violado nada de lo acordado acerca del protocolo de transmisión de mensajes. El constructor puede avanzar y probar muchas instrumentaciones distintas de una clase dada y efectuar experimentos para determinar cuáles son más eficientes en términos de espacio, velocidad y facilidades de desarrollo. El acuerdo da al constructor la máxima flexibilidad posible para probar diseños alternativos.

HERENCIA

La herencia es la característica de C++ que da mayor poder al concepto de clase. En C++, el término herencia se aplica sólo a clases y a sus características. Las variables no pueden heredar características de otras variables, y las funciones tampoco pueden heredar de otras funciones.

La herencia le permite construir y extender continuamente clases desarrolladas por cualquiera, básicamente sin límite. Partiendo de la clase más simple, se puede derivar clases cada vez más complejas que no son fáciles de depurar, sino que también son simples por sí solas.

El empuje principal en un proyecto elaborado en C++ es hacia el desarrollo de clases que resuelvan un problema dado. Estas clases pueden construirse en forma incremental partiendo de clases básicas y simples mediante el uso de la herencia. Cada vez que se deriva una nueva clase partiendo de una anterior, se puedan heredar algunas o todas las características de las clases primarias o progenitoras, agregando nuevas según se necesiten. Un proyecto completo puede tener clasificaciones o cientos de clases; pero estas clases se derivan a menudo de algunas clases básicas. C++ es diferente de algunos lenguajes orientados a objetos, ya que no sólo admite la herencia individual, sino también la herencia

El usuario puede construir una clase ficticia que simule los objetos con los que está trabajando el constructor. Esto permite al usuario avanzar y en realidad compilar, enlazar y ejecutar código que crea objetos y envía mensajes a los objetos que con el tiempo instrumentará el constructor. El compilador verifica que el usuario no haya violado nada de lo acordado acerca del protocolo de transmisión de mensajes. El constructor puede avanzar y probar muchas instrumentaciones distintas de una clase dada y efectuar experimentos para determinar cuáles son más eficientes en términos de espacio, velocidad y facilidades de desarrollo. El acuerdo da al constructor la máxima flexibilidad posible para probar diseños alternativos.

HERENCIA

La herencia es la característica de C++ que da mayor poder al concepto de clase. En C++, el término herencia se aplica sólo a clases y a sus características. Las variables no pueden heredar características de otras variables, y las funciones tampoco pueden heredar de otras funciones.

La herencia le permite construir y extender continuamente clases desarrolladas por cualquiera, básicamente sin límite. Partiendo de la clase más simple, se puede derivar clases cada vez más complejas que no son fáciles de depurar, sino que también son simples por sí solas.

El empuje principal en un proyecto elaborado en C++ es hacia el desarrollo de clases que resuelvan un problema dado. Estas clases pueden construirse en forma incremental partiendo de clases básicas y simples mediante el uso de la herencia. Cada vez que se deriva una nueva clase partiendo de una anterior, se puedan heredar algunas o todas las características de las clases primarias o progenitoras, agregando nuevas según se necesiten. Un proyecto completo puede tener clasificaciones o cientos de clases; pero estas clases se derivan a menudo de algunas clases básicas. C++ es diferente de algunos lenguajes orientados a objetos, ya que no sólo admite la herencia individual, sino también la herencia

múltiple, la cual permite la derivación de más de una clase al mismo tiempo heredando el comportamiento de todos sus ancestros.

La orientación a objetos intenta modelar aplicaciones del mundo real lo más aproximadas a él como sea posible, a su vez también intenta hacer un software reutilizable y un software extensible. Y la poderosa herramienta que posee esas características es la herencia.

A través de la herencia los diseñadores pueden construir nuevos modelos de software (tales como las clases) de una jerarquía de modulos ya existentes. Esto evita rediseñar y recodificar todo desde el principio. Estas nuevas clases pueden heredar tanto el comportamiento (operaciones y método) como la representación (variables y atributos) de las clases que ya existen.

El comportamiento de la herencia habilita el código del software a producir. La representación de la herencia habilita la estructura de datos de los objetos. Y con esto se muestra, la combinación de esos tipos de herencia que van a proveer una poderosa estrategia en el desarrollo y modelado de software. La herencia también provee un mecanismo muy natural para organizar la información.

Reutilización es una palabra que se escucha a menudo en el mundo de la programación orientada a objetos. Se refiere a tomar una clase e instanciarla directamente en sus programas o bien utilizarla como base de una nueva clase que herede algunas o todas sus características. Derivando una clase de una clase base, entonces efectivamente se reutiliza el código de la clase de base para satisfacer sus necesidades.

La derivación de una clase a partir de otra le ofrece mucha flexibilidad a un bajo costo en términos de codificación. Un vez que se tiene una clase de base sólida, sólo se necesita depurar los cambios que se hacen en las clases derivadas.

En una clase derivada de C++ se heredan caracteristicas de un progenitor, se puede hacer que la clase derivada las extienda, las restrinja, las modifique, las elimine o las utilice si ningún cambio. Todas estas variaciones se pueden agrupar en dos técnicas básicas

orientadas por objetos. La primera se denomina restricción de características, que limita o elimina una característica del progenitor.

Limitaciones de la herencia de C+ +

Cómo y cuándo se deriva una clase de otra es puramente decisión del programador, esto puede parecer obvio pero es una limitación. El diseñador de un programa debe decidir al momento de la compilación quién hereda qué y de quién, y cómo y cuándo se lleva a acabo la herencia.

Conceptualmente es posible tener un sistema en que los objetos son completamente dinámicos, en el sentido de que no sólo vienen y van de acuerdo con los requistos al momento de la ejecución, sino que también generan nuevas derivadas de clases que darían origen a bibliotecas de clases completamente nuevas, imprevistas por el programador. A este nuevo tipo de herencia se le podría llamar herencia tardía, dada su relación con el enlace tardío.

Una perspectiva diferente de la herencia

Por años se ha utilizado la programación estructurada descendente para desarrrollar proyectos de software. Empleando esta metodología, un problema se subdivide en subproblemas cada vez menores, hasta que las subtareas sean fácilmente manejables. Básicamente esta subdivisión nos lleva a una función de alto nivel que llama a un sinnúmero de niveles de nivel inferior y cada una de ellas utiliza funciones de niveles incluso inferiores. En un sentido, las funciones de nivel superior incorpora características implantadas en otras funciones. Esta noción no es tan diferente de la herencia. La siguiente figura ilustra un programa que incorpora este tipo de programación descendente.

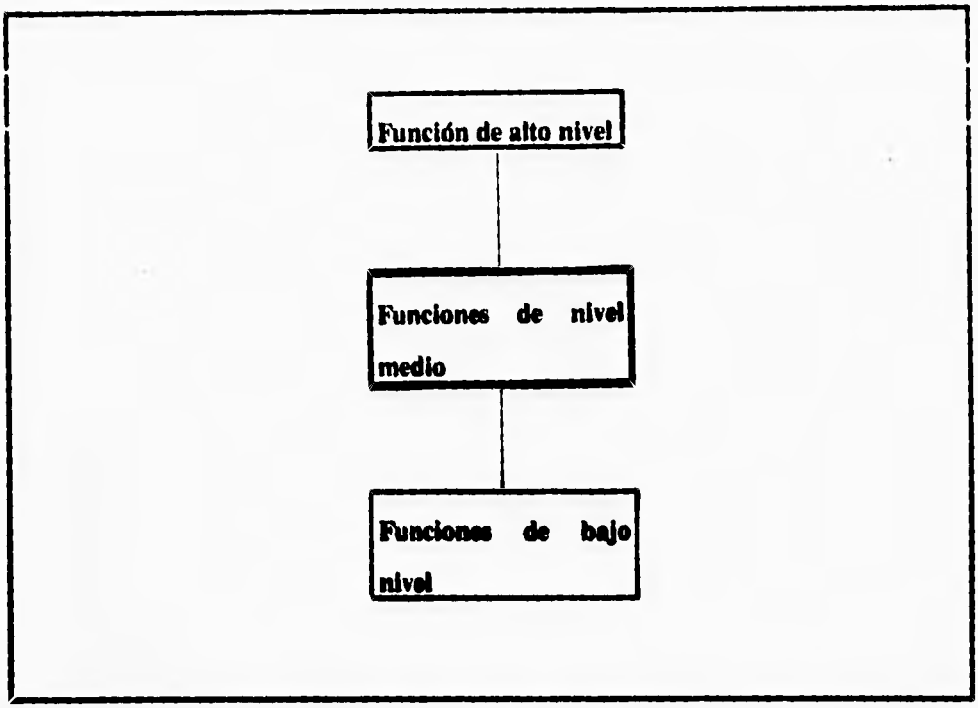


Fig. 1.3.2.1 Arquitectura de un programa desarrollado mediante el empleo de tecnicas de programación estructurada.

El árbol de un programa descendente tiene una similitud impresionante con el de la herencia que se muestra en la siguiente figura 1.3.2.2.

Considerando sólo las graficas entre las dos en perspectiva. El mundo de la programación estructurada, el árbol, se obseva desde la parte superior. Las funciones de cualquier nivel dado del árbol invocan funciones de niveles inferiores. Las funciones de nivel inferior tienen un campo de acción más estrecho y un comportamiento más especializado. En la herencia de clases, normalmente se tiene una clase que hereda características de sus progenitores. Asi se observa el árbol desde abajo para saber que caracteristica tiene.

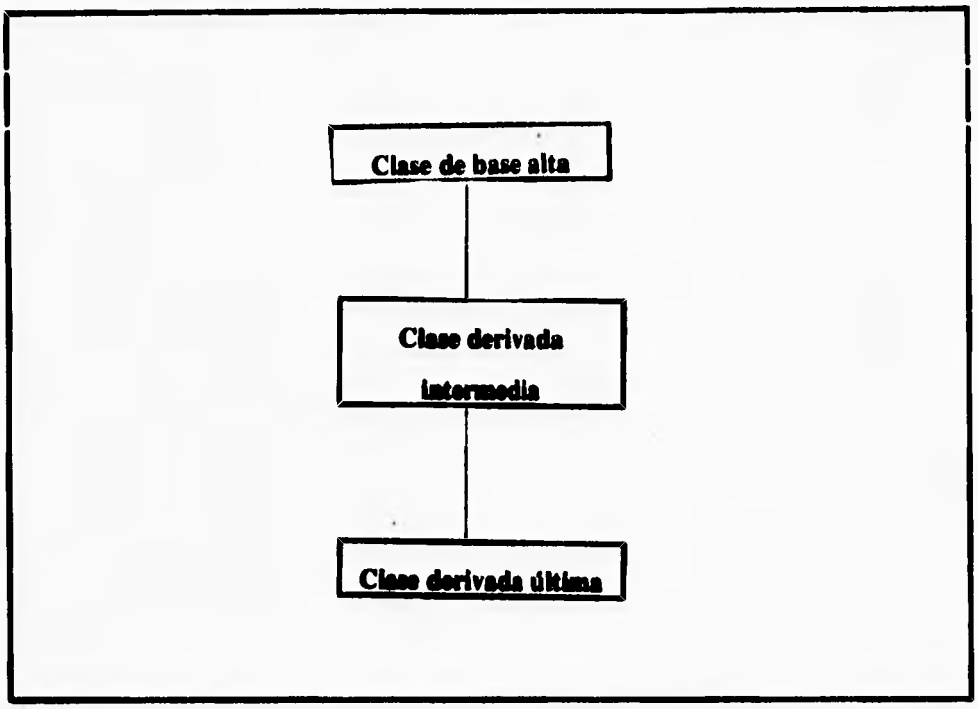


Fig. 1.3.2.2 Arquitectura básica de una jerarquía de clases.

Herencia Individual

Si una clase hereda características de alguna parte, pueden tener una o más clases progenitoras, comenzando con la herencia individual.

Cuándo heredar

Siempre que se construya una nueva clase, se debe determinar primero si se puede utilizar una clase preexistente como clase de base. A menudo se encuentran clases que proporcionan casi el comportamiento indicado. Estos son buenos candidatos, ya que se pueden heredar todas las características deseadas. Para deshabilitar una función de la clase progenitora o primaria, se emplea una técnica llamada sobrecarga de funciones.

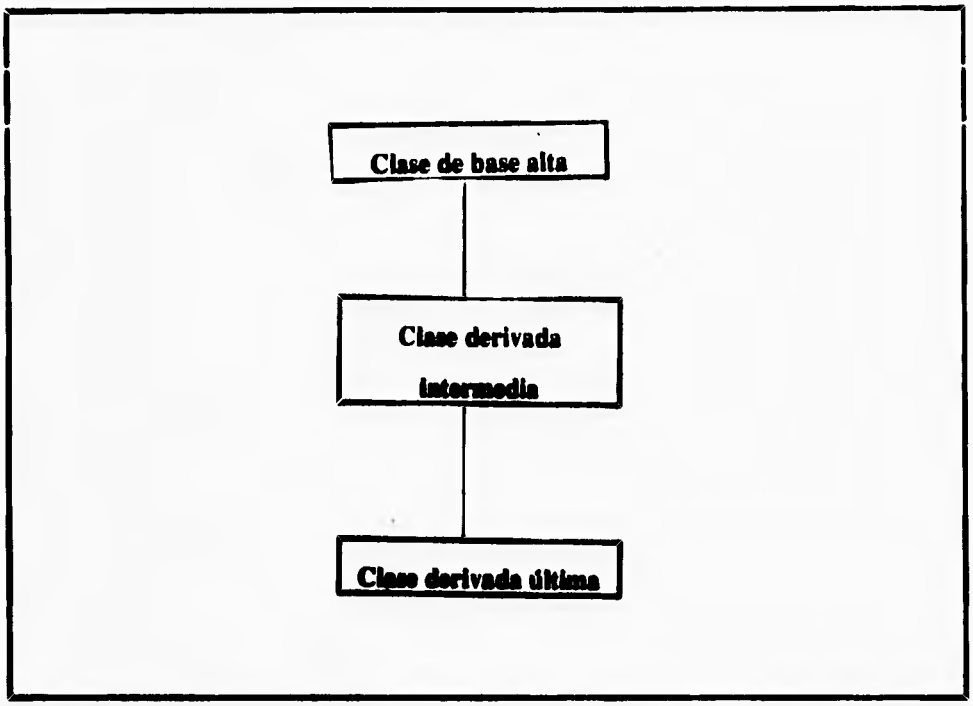


Fig. 1.3.2.2 Arquitectura básica de una jerarquía de clases.

Herencia individual

Si una clase hereda características de alguna parte, pueden tener una o más clases progenitoras, comenzando con la herencia individual.

Cuándo heredar

Siempre que se construya una nueva clase, se debe determinar primero si se puede utilizar una clase preexistente como clase de base. A menudo se encuentran clases que proporcionan casi el comportamiento indicado. Estos son buenos candidatos, ya que se pueden heredar todas las características deseadas. Para deshabilitar una función de la clase progenitora o primaria, se emplea una técnica llamada sobrecarga de funciones.

Qué no se puede heredar

Como en la vidad real, en C++ no todo se puede transmitir a través de la herencia. Esto se puede considerar en un principio como una desventaja o limitación artificial, pero en realidad sólo algunos casos especiales son inconsistentes por definición con la herencia:

1. **Constructores.**
2. **Destructores.**
3. **Nuevos operadores definidos por el usuario.**
4. **Operadores de asignación definidos por el usuario.**
5. **Relaciones Friend.**

Las clases derivadas invocan automáticamente al constructor de la clase de base cuando se instancian; pero después de ser construidas, el constructor de la clase de base queda fuera de límites. Aunque los constructores de las clases de base son heredados y sólo puden ser invocados automáticamente por el compilador cuando se construye un objeto derivado. El constructor de una clase de base no pueden ser invocados de manera explicita en una clase derivada como otras funciones heredadas.

De manera analoga, los destructores están diseñados para ser invocados automáticamente cuando un objeto del campo de acción lo llama. En un programa no se permite la invocación explicita de un destructor.

Clases diseñadas para ser heredadas

Dado el enfasis que se hace en la facilidad de reutilización y la herencia en C++, la mayor parte del tiempo se crean clases que utilizan subsiguientemente como clases de base de otras clases.

Herencia múltiple

En C++ una clase no esta limitada a un progenitor. De hecho una clase puede tener muchos padres y heredar propiedades de cada una de sus clases de base. Este tipo de herencia introduce cierta complejidad en el lenguaje y el compilador, pero los beneficios son sustanciales. Considere crear una clase, que no sólo tenga las propiedades de las tablas, sino también la característica geométrica de ser redonda, la siguiente figura muestra las relaciones.

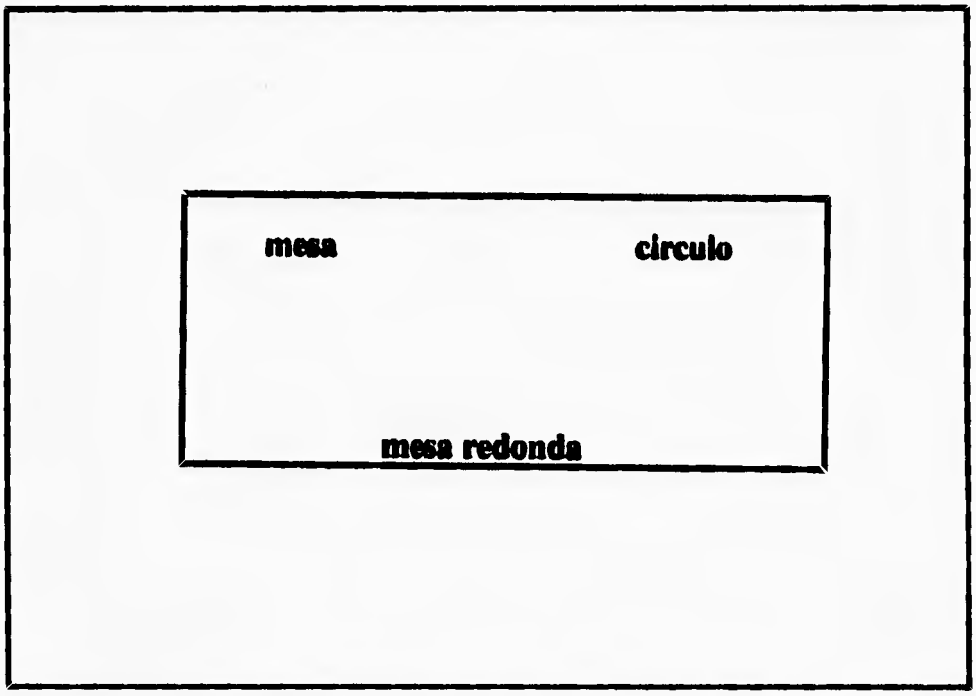


Fig. 1.3.2.3 Creación de una tabla redonda a través de la herencia múltiple.

Habiendo heredado la mayoría de sus propiedades de sus clases base, la clase se reduce a una implantación más bien trivial. En la declaración de una base derivada de múltiples progenitores, se necesita declarar no sólo las diversas clases de base, sino también

los especificadores de acceso de las clases de base. Estos especificadores se utilizan justamente como en la herencia simple.

El tema de la herencia en general no es considerado complejo por quienes se inician en el uso de C++, debido principalmente a sus similitudes con la vida real. Las reglas asociadas con privilegios de acceso necesitan a veces estudio más profundo, con la complejidad adicional las funciones miembro y miembros de datos heredados; pero con un poco de experiencia, el usuario puede dominar todos los temas. Lo que es menos obvio para los programadores, pero igualmente importante a fin de comprender el lenguaje C++, es que el codigo y los datos que genera el compilador en ciertas situaciones y cómo afecta la herencia el uso de la memoria.

Un depurador heredado

La mayor parte del tiempo en un proyecto se trabaja dentro del marco de jerarquías de clases, en las que todas las clases se derivan de una o dos clases raíz.

Herencia en la representación del conocimiento

La herencia tiene sus raíces en el sentido común que son los paradigmas del conocimiento, utilizados en la inteligencia artificial.

La herencia es sólo una clase de relaciones entre conceptos. Otras relaciones pueden representar atributos (variables) de conceptos (objetos). Minsky (1975) introdujo las estructuras llamadas frames que permitieron la definición de nuevos conceptos en términos de los frames previamente definidas. Los frames contienen segmentos los cuales son semejantes a los registros de campo o atributos. Los frames pueden heredar segmentos de otros frames.

Las diferente facetas de la herencia

La herencia es una poderosa técnica que organiza el código de bases complejas. Lo cual permite la construcción de nuevas clases para las jerarquías de clases existentes. A través de la herencia se enriquecen las relaciones semanticas entre entidades en el espacio de un objeto que puede ser expresadas directamente y naturalmente.

La herencia introduce algunas complejidades, especialmente cuando se integra con otros conceptos orientados a objetos tales como la encapsulación.

1. **Herencia y subtipos:** En la mayoría de los lenguajes orientados a objetos los dos conceptos se pueden intercambiar.

2. **Visibilidad de métodos y variables heredadas.** Algunos lenguajes orientados a objetos tales como **Simula** permite la manipulación directa. Otros lenguajes tales como **C++** distinguen entre la manipulación pública y privada. Con la herencia hay todavía una tercera alternativa que es el surgimiento de subclases visibles.

3. **Herencia y encapsulación:** La visibilidad viola la información oculta. El hecho es que si hay un conflicto entre herencia y encapsulación el uso de variables de superclases son accesados directamente. Aunque la encapsulación y la sobrecarga pueden ser usadas para mantener algunas de las funciones de la herencia. Sin embargo, la herencia es mucho más directa y natural para copiar código y estructura.

4. **Como Especializarse:** La herencia esta especializandose en las clases existentes. Las clases pueden ser especializadas extendiendo su representación o comportamiento. Alternativamente, las clases pueden ser especializadas a través de la restricción de la representación u operaciones de las clases existentes.

5. **Herencia de objetos:** La mayoría de los lenguajes orientados a objetos soportan la herencia de clases (capacidad de una clase para heredar representaciones y métodos de otra clase). Otra alternativa que soporta es que los objetos hereden de otros objetos.

6. **Herencia múltiple:** En muchas situaciones, es deseable heredar más de una clase. Esto es llamado herencia múltiple. Cuando una clase hereda más de un padre existe la posibilidad de un conflicto: Métodos o variables, con el mismo nombre pero diferente semantica son heredadas de difererentes superclases.

La herencia es quiza el concepto que en la programación orientada a objetos más se utiliza.

POLIMORFISMO

El origen del término *polimorfismo* es simple: proviene de las palabras griegas *poly* (muchos) y *morphos* (forma) -multiforme. El polimorfismo describe la capacidad del código de C++ de comportarse de diferentes maneras dependiendo de situaciones que se presentan al momento de la ejecución.

El polimorfismo no es tanto una característica de objetos como lo es de funciones member de una clase. Se implanta a través de la arquitectura de clases; sin embargo, sólo las funciones member de la clase pueden ser polimorfas, y no la clase completa. Este tipo de implantación es similar al uso de verbos de idiomas naturales, que son equivalentes a funciones member de C++.

Enlace durante la compilación y durante la ejecución.

En C++, una función polimorfa se puede asociar con una de muchas funciones posibles sólo cuando se pasa un objeto real a la función polimorfa. Dicho de otra manera, el código fuente mismo no siempre le indica cómo se va a ejecutar una sección de código, indicando que el mecanismo de llamada a funciones es diferente del de ANSI C. En C++, una llamada a una función sólo se *indica* en el código fuente, sin especificar la función exacta a la que se llama. Esto se conoce como *enlace durante la ejecución*. En la mayoría de los lenguajes de programación tradicionales, como C y Pascal, el compilador llama a identificadores de funciones fijos, con base en el código fuente. Después, el dispositivo de enlace toma estos identificadores y los reemplaza por una dirección física. Este proceso se conoce como *enlace durante la compilación*, ya que los identificadores de funciones están asociados con direcciones físicas antes del momento de la ejecución, durante el proceso de compilación y enlace.

Funciones virtual.

En C++ el enlace durante la ejecución se especifica para una función declarándola virtual. El enlace durante la ejecución tiene sentido en C++ sólo en el caso de objetos que son parte de una jerarquía de clases. La declaración de una función `virtual` para una clase que no se utiliza como clase base es sintácticamente correcta, pero da lugar a un gasto extra innecesario al momento de la ejecución. La figura 1.3.2.4 muestra el árbol de herencia simple de algunas funciones `virtual`.

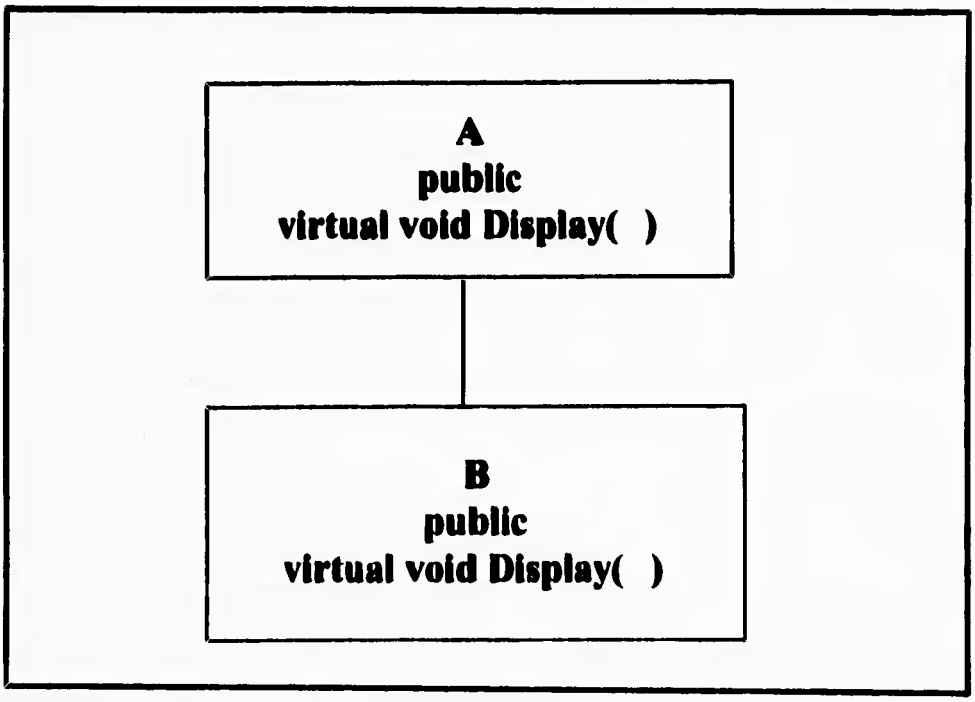


Fig. 1.3.2.4. Funciones virtual en una jerarquía simple.

Listado 1.3.2.1 Uso de funciones virtual.

```
#include <stdio.h>
class A {
public:
     virtual void Display( ) {puts( "\nClass A" );}
};
class B: public A {
public:
     virtual void Display( ) {puts( "\nClass B" );}
};
void Show(A* a)
{
     a->Display( ); //averigüe qué función utilizar
                    //durante la ejecución
}
void main( )
{
     A* a = new A;
     B* b = new B;
     a->Display( ); // utilice A::Display( )
     b->Display( ); // utilice B::Display( )
     Show(a);       // utilice A::Display( )
     Show(b);       // utilice B::Display( )
}
```

El comportamiento polimorfo de la función member `Display( )` en las clase A y B no es evidente cuando se observa tan sólo la funcion `main( )`. El comportamiento polimorfo no se hace evidente en la función `Show( )`, en la que es imposible anticipar -con sólo examinar su código fuente- si se invocará una función `A::Display( )` o `B::Display( )`. Si se eliminara la palabra reservada `virtual` en las dos declaraciones de clases, se observaría el siguiente comportamiento:

```
void main( )
{
     // comportamiento con funciones que no son virtual
     Show(a);  // utilice A::Display( )
     Show(b);  // utilice B::Display( )
}
```

La segunda llamada a función hace que se llame a `A::Display( )`, ya que el argumento `B*` se convierte en `A*` y luego se pasa a la función `Show(A*)`. La llamada `a->Display( )` se enlaza durante la compilación a la función `A::Display( )`, dando lugar al comportamiento establecido.

Declarar una función `virtual` no significa que se elimine en una clase derivada, sino que tan sólo *puede* ser eliminada. Para propagar el comportamiento polimorfo de una función hacia abajo en un árbol de herencia, cada clase derivada debe declarar la misma función como `virtual`.

Ejemplo de polimorfismo.

Para aprovechar al máximo el lenguaje C++, debe explotar el polimorfismo hasta donde le sea posible. El polimorfismo no sólo vuelve un programa más sencillo, sino también mucho más flexible y robusto.

Considere una serie de clases que se emplean para implantar una base de datos que lleva el registro de obras maestras de arte. Las clases están organizadas en una jerarquía, con la intención de tener acceso a cualquier objeto de la base de datos sin que sepa de qué tipo de objeto se trata. La figura 1.3.2.5 es el árbol de herencia de esta serie de clases.

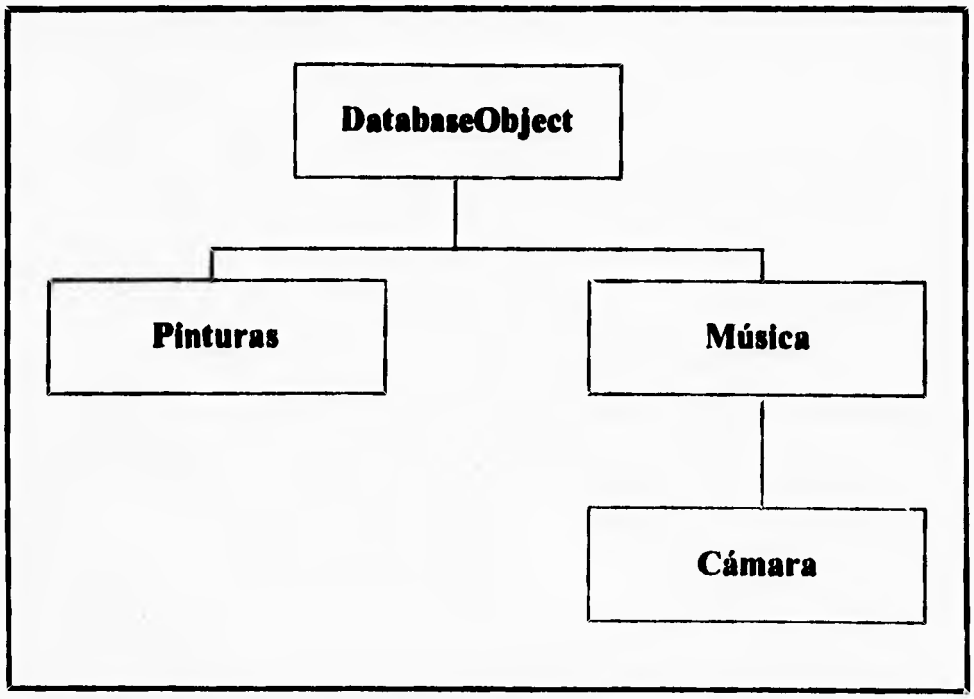


Fig. 1.3.2.5 Jerarquía de clases de una base de datos simple.

Cualquier característica común en la base de datos pertenece al nodo más alto posible del árbol. Todos y cada uno de los opus de la base de datos tienen un autor y una

fecha. Otra información puede variar entre las clases, como el método que se emplea para desplegar información acerca de un objeto. Esta función de presentación es un buen candidato para el polimorfismo y se aplica con el mecanismo de las funciones `virtual`.

El polimorfismo permite que se pasen referencias genéricas a objetos `Database` por el programa. En tanto que los objetos se deriven de Database y utilicen funciones `virtual` para realizar la tarea requerida, este método funciona eminentemente. De hecho, es muy recomendable como una forma de permitir a C++ encargarse de los detalles de la simplificación de su código.

La resolución del ámbito de acción deshabilita el polimorfismo.

Existe libertad considerable para llamar a una función `virtual`. Una función `virtual` puede ser invocada prácticamente por cualquier otra que tenga privilegios de acceso apropiados. Una función `virtual` de una clase derivada puede incluso invocar una función `virtual` en su clase base con el mismo nombre, sin ocasionar repeticiones infinitas ni dañar el sistema. Para invocar una función `virtual` en la clase base de una clase derivada, tiene que utilizar el operador de resolución del ámbito de acción para indicar de manera explícita que clase base utilizar. Este operador dice al compilador que se desea pasar por alto el mecanismo de las funciones `virtual`.

Ser o no ser virtual.

Se puede declarar cualquier función member como una función virtual, con las siguientes restricciones:

1. Constructores.
2. Funciones miembro `static`

Estas dos restricciones dejan la puerta abierta para la mayoría de las funciones de un programa ordinario; pero el gasto extra asociado con funciones `virtual` (y el mecanismo de enlace durante la ejecución que lo acompaña) vuelve pertinente utilizar funciones `virtual` sólo cuando se necesiten. Siempre se hace evidente de inmediato si se necesita o no una función polimorfa. Por ejemplo, se puede detectar de inmediato funciones `virtual` en clases que no están diseñadas para generar derivaciones. En realidad es una llamada subjetiva. Sin embargo, algunos lineamientos facilitan la elección entre funciones `virtual` y que no son `virtual`. Debe considerar las funciones virtual al menos en estos casos:

1. En clases diseñadas para estar en la cima o cerca de la cima de una jerarquía de clases.
2. Para funciones que describen atributos de clases que dependen de la estructura o tipo de una clase.
3. Para funciones que implantan entrada o salida para una clase.
4. Para funciones que tienen acciones definidas sólo para clases específicas.

Considerando el árbol de herencia de clases para objetos de una base de datos de arte. Cada clase del árbol podría ser aumentada con funciones para identificar un objeto, para leer los datos de un objeto desde una terminal, para devolver respuestas a interrogaciones del usuario, etcétera. Consideremos dotar a la clase raíz `DatabaseObject` de estas funciones y operadores:

Display

IsA

AuthoredBy

Date

PrintOn

ReadFrom

ExhibitedAt

TreeAncestors

Algunos de éstos son candidatos para observar comportamiento polimorfo y otros no.

Ahora consideremos de uno en uno.

Display La función Display imprime el contenido de una clase en la terminal del usuario. Cada clase tiene una estructura interna potencialmente diferente de su ancestro. Por lo tanto, de acuerdo con los lineamientos 1, 2 y 3, esta función necesita ser polimorfa.

IsA Cada clase es un tipo nuevo y, por lo tanto, debe devolver su identificador correctamente. El lineamiento 2 indica volver a utilizar una función `virtual`.

AuthoredBy El objeto devuelto por la función AuthoredBy es una cadena, declarada en la clase databaseObject. No es probable que el nombre del autor sea una función del tipo de objeto; de modo que tal vez no se requiera el comportamiento polimorfo.

Date Es probable que la fecha de una obra de arte no dependa del tipo de objeto en consideración; pero de nuevo aquí, no podemos estar absolutamente seguros. La función que se declara aquí devuelve un valor de año entero. Esta es otra llamada subjetiva.

PrintOn y ReadFrom Utilizando el lineamiento 3, estas funciones que se utilizan para entrada y salida del contenido de una clase deben ser siempre `virtual`.

ExhibitedAt	La función ExhibitedAt describiría probablemente el museo u otro lugar en el cual se exhiba un objeto en ese momento. Es posible que este lugar dependa de la clase; de modo que prodría requerirse una función `virtual`
TreeAntecesors	La función TreeAntecesors se podría utilizar como función depuradora, ilustrando las clases base inmediatas de una función. Después, esta función daría soporte a la característica de repaso de la jerarquía de clases. Esta es definitivamente una función polimorfa.

Mecánica del polimorfismo.

El poder del polimorfismo es considerable, en especial porque es muy sencillo utilizar las funciones `virtual`. Sin embargo, las funciones `virtual` son diferentes en sus entrañas de las funciones miembro ordinarias. El compilador genera código especial para dar soporte a funciones virtuales buscando y realizando un recorrido vectorial a través de apuntadores de funciones durante la ejecución. El código generado depende de si una función `virtual` se utiliza en una clase con una clase base o con clases base múltiples.

Polimorfismo con herencia simple.

La herencia simple no produce gasto adicional de memoria en objetos a menos que se utilicen funciones `virtual`. Aun así, la implantación de funciones virtual con herencia simple es directa. La mejor manera de estudiar las características de un compilador durante la ejecución es observar el código que se genera. Considere el árbol de herencia de la figura 1.3.2.6.

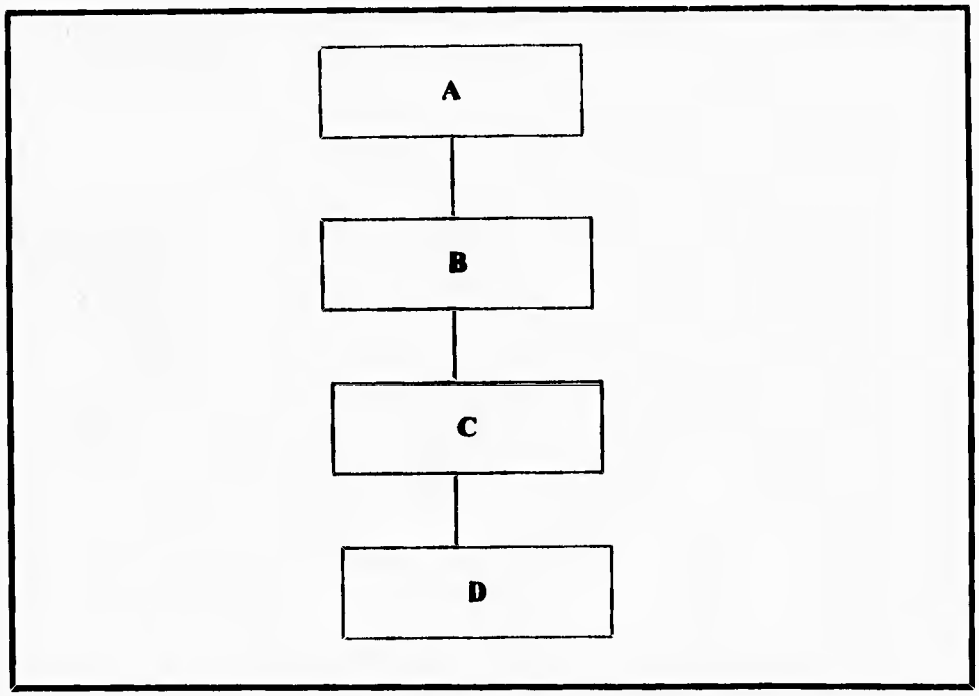


Fig. 1.3.2.6 Jerarquía de clases en la que se utiliza la herencia simple.

Cada clase tiene conjuntos idénticos propios de variables, funciones miembro y funciones miembro `virtual`.

Mediante el estudio del código compilado del Turbo Debugger, se puede determinar la estructura de cada clase y las secuencias de invocación de cada llamada a una función. La figura 1.3.2.7 es la disposición resultante en la memoria de un objeto *A* al que apunta *a*.

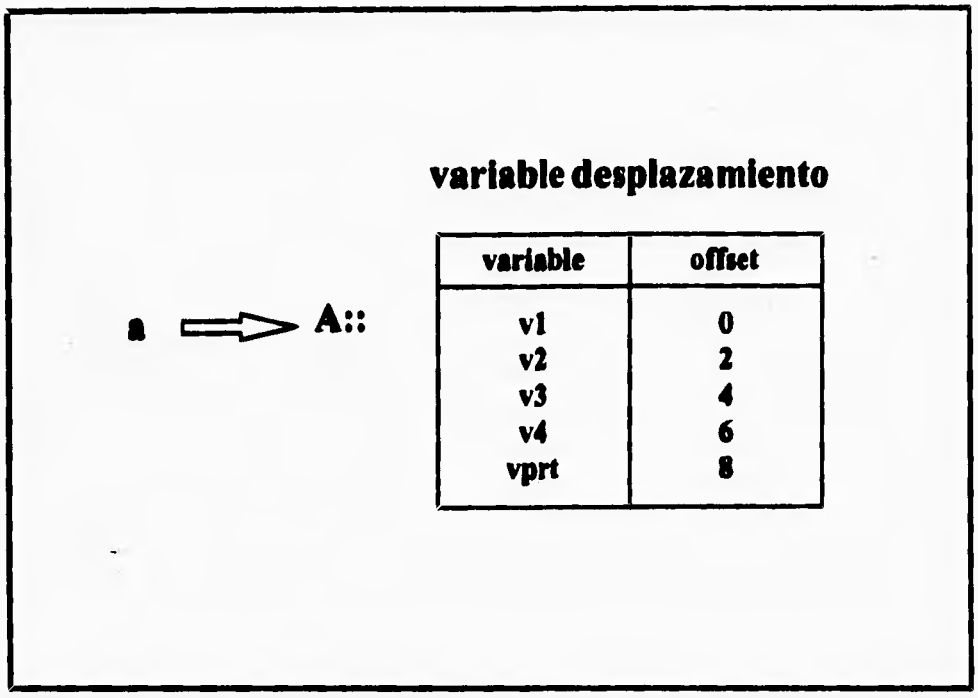


Fig. 1.3.2.7 Disposición en la memoria de un objeto de clase A.

Todo luce muy normal, salvo por el campo llamado `vptr` de desplazamiento 8. Es un apuntador a una tabla de apuntadores a través del cual el compilador recorre vectorialmente el código durante la ejecución cuando invoca una función `virtual`. La tabla de apuntadores a la que hace referencia el apuntador `vptr` se conoce como tabla de funciones `virtual`, o `vtab` para abreviar. La figura 1.3.2.8. muestra el aspecto que tiene `vtab` de la clase A.

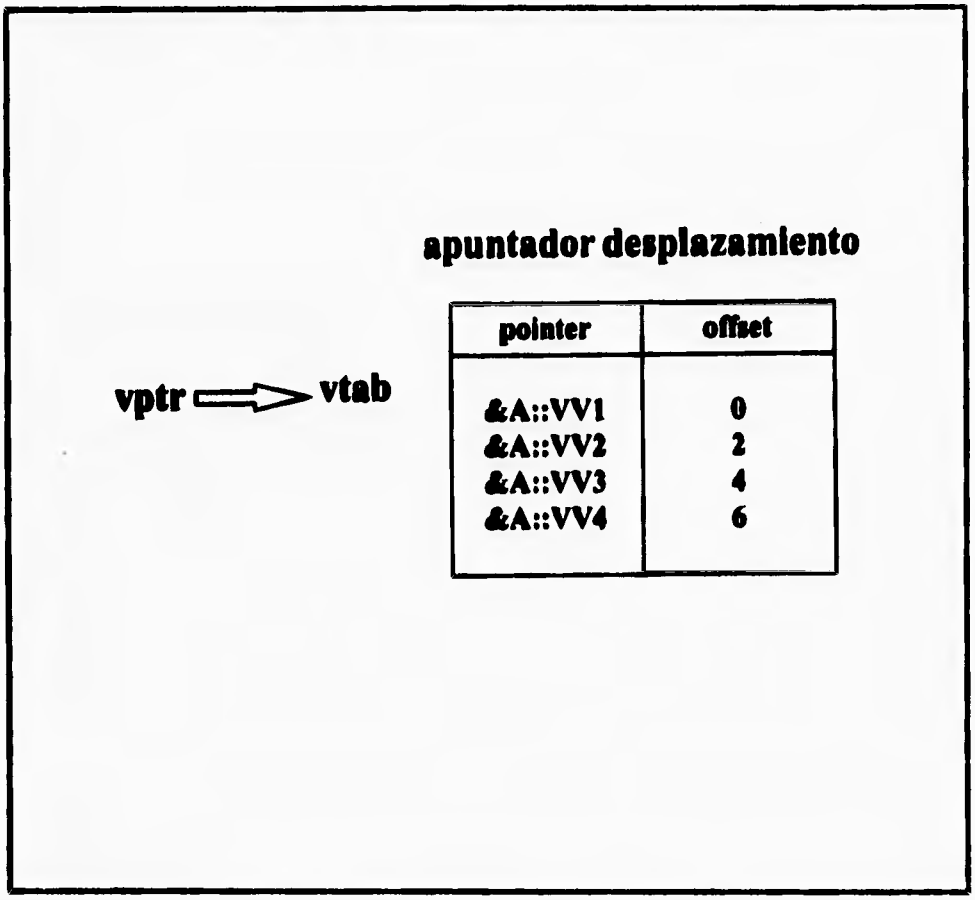


Fig. 1.3.2.8. Vtab de objetos de clase A.

A cualquier objeto de una clase con funciones `virtual` se le asigna el espacio de almacenamiento extra necesario para el apuntador `vptr` y la `vtab`. Llamar a una función `virtual` implica utilizar el desplazamiento de un apuntador de tabla `virtual` en `vtab`. Esta rama se agrega a vtab durante la ejecución para determinar la dirección de la función virtual que se debe llamar. A continuación se presentan las estructuras de las clase derivadas B, C y D (Figuras 1.3.2.9, 1.3.2.10 y 1.3.2.11).

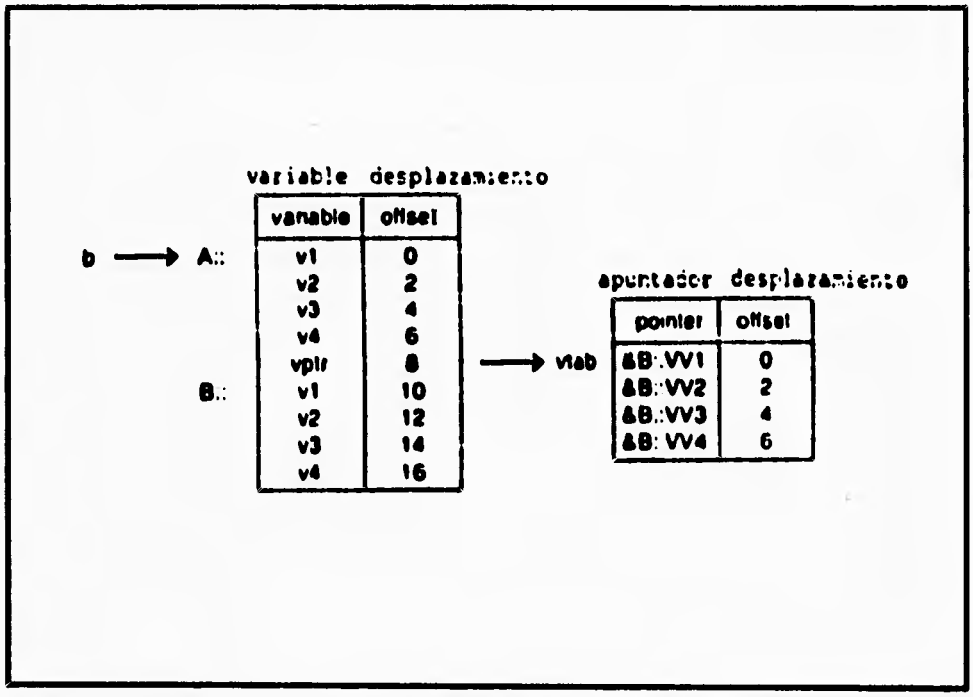


Fig. 1.3.2.9 Disposición en la memoria de un objeto de clase B.

Observando la posición de `vptr` en cada clase: `vptr` se coloca siempre al final de las variables de la primera clase base. Toda clase tiene un `vptr`, pero su valor es diferente para cada una. El `vptr` hace referencia a la `vtab` a utilizarse para cada clase. Por lo tanto, cambiando sólo el apuntador vptr, se encuentran accesibles diferentes `vtabs`, y se pueden invocar diferentes funciones `virtual`.

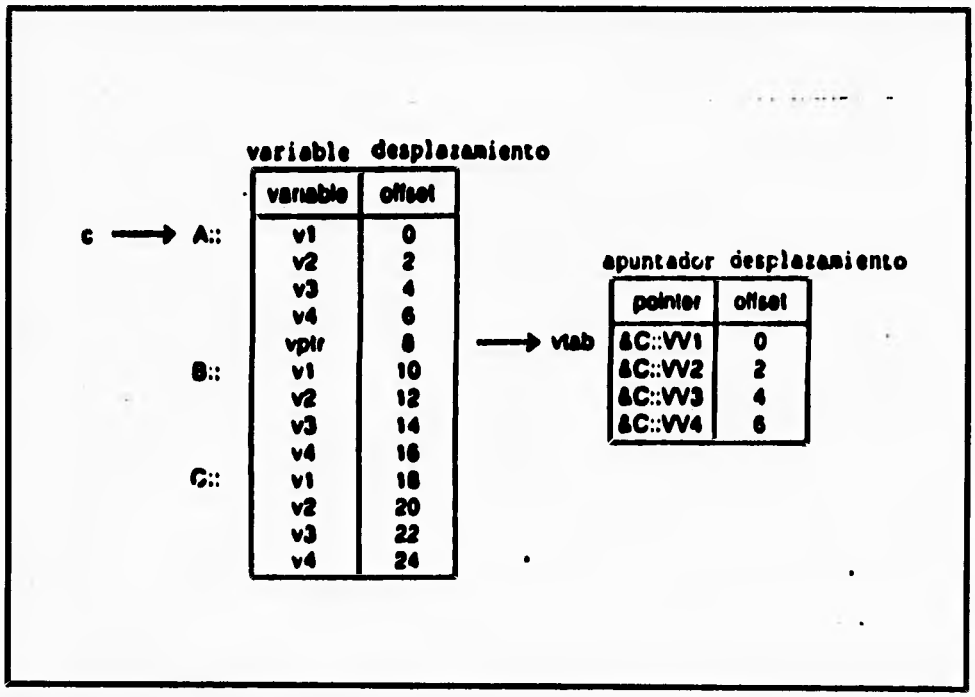


Fig. 1.3.2.10 Disposición en la memoria de un objeto de clase C.

La vtab de una clase derivada es diferente de la vtab de sus clases base. Esto incluye la clase D, que comprende los objetos integrales A, B y C. Si un objeto D se convierte en un objeto A, el objeto D sigue teniendo una vtab diferente de los objetos A.

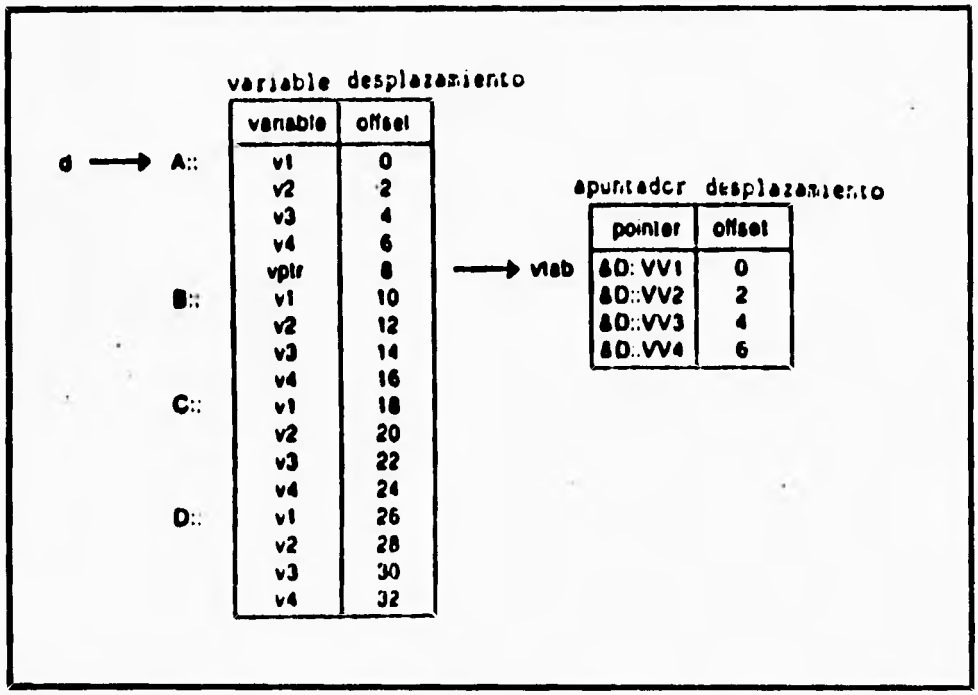


Fig. 1.3.2.11 Disposición en la memoria de un objeto de clase D.

Polimorfismo con herencia múltiple.

La herencia múltiple es evidentemente más complicada que la herencia simple. Esto sucede al nivel de programación y al nivel del compilador. Las clases que tienen múltiples clases base que emplean funciones `virtual` utilizan múltiples apuntadores `vptr` y tablas `vtab`; de modo que la mecánica del polimorfismo es similar al caso de la herencia simple. Considere el árbol de herencia de la figura 1.3.2.12.

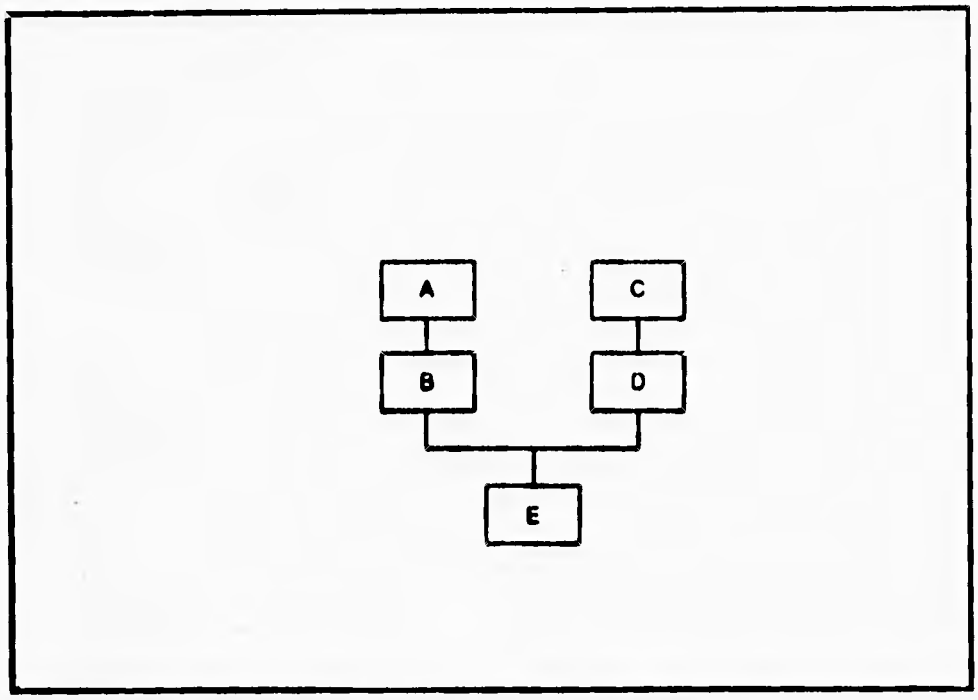


Fig. 1.3.2.12. Jerarquía de herencia múltiple.

Cuando una clase derivada tiene clases base múltiples, es posible que tenga también múltiples `vptrs`. Por cada clase base que utiliza funciones `virtual` hay un `vptr`, de modo que una clase puede tener cualquier número de apuntadores de tablas virtuales. Es evidente que todos los `vptrs` representan gasto de memoria extra en objetos; pero las buenas nuevas son que este gasto general está bajo su control directo. Si ninguna de estas clases tiene funciones `virtual`, no hay gasto extra. La cantidad de gasto extra es función de la complejidad del árbol de herencia y el uso de funciones `virtual`.

El segundo motivo por el que la herencia es útil es más sutil y eficaz. El concepto de *subclase* orientado al objeto, ayuda a crear una solución que sea más fácil de mantener y ampliar.

Cuando una *clase derivada* hereda de una *clase base*, los objetos de la **class** derivada retienen, todavía, el número de miembros en la **class** base. Mediante la derivación de muchas clases a partir de la misma **class** base, se pueden crear un grupo de **clases** que tengan el mismo interfaz, pero diferentes implementaciones. El programa principal maneja un grupo de esos objetos. Puede enviar cualquier mensaje a cualquier objeto, pero el efecto séra diferente dependiendo de cada subclase.

Por ejemplo suponga que se esta diseñando un sistema de control de tráfico aéreo. Todos los objetos en la presentación serán de **class plane**, pero cada uno tendrá su propio diseño en la pantalla. La **class** base, en este sistema es **plane**, y el interfaz a **plane** indica que "un objeto **plane** sabe como formularse y moverse por si mismo", incluso aunque un **plane** no sepa cómo desarrollar esas operaciones hasta que conozca si es un avión de una línea regular o una avioneta privada.

La utilización de interfaces idénticos con diferentes implementaciones se denomina *polimorfismo*. Como muchas de las características de C++, el polimorfismo mejora la calidad del programa. Un programa diseñado en torno al polimorfismo, es fácil de mantener y ampliar. Para ampliar un programa polimórfico, simplemente derive una nueva subclase a partir de la misma **class** base, que heredaron los otros objetos genéricos. La nueva subclase puede manejarse por el mismo programa sin modificación. Como el programa es únicamente un gestor para un conjunto de objetos genéricos, los errores se aislan automáticamente en los mismos objetos. Una vez que una **class** base es depurada, cualquiera de los errores en una nueva **class** derivada es consecuencia del nuevo código en la clase derivada.

Cuando una *clase derivada* hereda de una *clase base*, los objetos de la **class** derivada retienen, todavía, el número de miembros en la **class** base. Mediante la derivación de muchas clases a partir de la misma **class** base, se pueden crear un grupo de **clases** que tengan el mismo interfaz, pero diferentes implementaciones. El programa principal maneja un grupo de esos objetos. Puede enviar cualquier mensaje a cualquier objeto, pero el efecto séra diferente dependiendo de cada subclase.

Por ejemplo suponga que se esta diseñando un sistema de control de tráfico aéreo. Todos los objetos en la presentación serán de **class plane**, pero cada uno tendrá su propio diseño en la pantalla. La **class** base, en este sistema es **plane**, y el interfaz a **plane** indica que "un objeto **plane** sabe como formularse y moverse por sí mismo", incluso aunque un **plane** no sepa cómo desarrollar esas operaciones hasta que conozca si es un avión de una línea regular o una avioneta privada.

La utilización de interfaces idénticos con diferentes implementaciones se denomina *polimorfismo*. Como muchas de las características de C++, el polimorfismo mejora la calidad del programa. Un programa diseñado en torno al polimorfismo, es fácil de mantener y ampliar. Para ampliar un programa polimórfico, simplemente derive una nueva subclase a partir de la misma **class** base, que heredaron los otros objetos genéricos. La nueva subclase puede manejarse por el mismo programa sin modificación. Como el programa es únicamente un gestor para un conjunto de objetos genéricos, los errores se aislan automáticamente en los mismos objetos. Una vez que una **class** base es depurada, cualquiera de los errores en una nueva **class** derivada es consecuencia del nuevo código en la clase derivada.

1.3.3 FLUJO DE DATOS

Los flujos de datos son una nueva técnica en Visual C++ para el manejo de Entrada y salida de información. En muchos de los casos, la técnica de los flujos de datos es un poco diferente de las funciones de ANSI C. El antiguo lenguaje ANSI C empleaba un método basado en archivos, apuntadores de archivos y manipuladores de archivos. Los diseñadores cambiaron la parte de E/S de Visual C++ para dar a las operaciones de entrada y salida de datos una esencia más orientada por objetos fusionándolas en el resto del lenguaje. Ahora las operaciones de E/S se realizan principalmente a través del empleo de operadores sobrecargados, en vez de hacerlo a través de funciones.

El concepto de flujo de datos no es nuevo para Visual C++, de hecho el lenguaje ANSI C utiliza los llamados flujos de datos para indicar un tipo de puerto abstracto por el cual pueden fluir o circular uni o bidireccionalmente datos no estructurados ni procesados. Los flujos de datos de ANSI C se consideran una construcción de E/S de bajo nivel y encima de ella se ubica el sistema de archivos más estructurado. Visual C++ fue desarrollado para clases, pero el lenguaje mismo rara vez las utiliza e incluso entonces lo hace a través de bibliotecas. El uso de clases através de bibliotecas conserva el compilador de menor tamaño y más portatil. Los flujos de datos de Visual C++ son diferentes de la mayoría de las otras partes del lenguaje, ya que se implantan exclusivamente a través de clases completas, que es la única jerarquía importante que se proporciona con el compilador. Las versiones de C++ de AT&T hasta la 1.2 venían con una versión de clases de flujos de datos que ahora se considera obsoleta.

Borland C++ ya no dará soporte a estos flujos de datos anacrónicos en versiones futuras. En su lugar, utilizará los flujos de datos de E/S más nuevos de C++.

Los flujos de datos se implantan por separado de las funciones `stdio`. Por ello, el uso de flujos de datos y `stdio` en un programa puede causar problemas. Si obtiene un carácter con una operación de flujo de datos y luego un carácter con la función

`putc(...)` de `stdio`, es posible que los caracteres no sean emitidos en el orden correcto, dependiendo de la operación de los buffers internos y de la activación automática de los buffers.

Flujo de Datos como Filtros Generalizados

Los flujos de datos constituyen un núcleo protector que dota a las operaciones de E/S de polimorfismo y otras características orientadas por objetos. Aunque los flujos de datos están asociados a menudo con E/S en realidad son una abstracción para la transferencia genérica de datos de un objeto a otro, generalmente mediante el uso de un mecanismo de separación por buffers. Esto significa que cualquier función utilizada para mover datos de una localidad de la memoria a otra, cambiando o no los datos, puede ser considerada una operación de flujo de datos. Por ejemplo las antiguas funciones del lenguaje ANSI C

`memmove (...) sprintf (...) strcpy (...)`

se pueden implantar como operaciones de flujo de datos, ya que implican transferencias de datos de una localidad de la memoria a otra: Los datos fluyen como si estuvieran en un torrente o flujo. La biblioteca de flujos de datos tiene muchas clases, pero todas ellas encajen en una de tres categorías.

En la tabla 1.3.3.1 se muestran las nuevas categorías de flujos de datos, comparandolas con las antiguas funciones de ANSI C a las que sustituyen.

NUEVAS CATEGORÍAS Y EJEMPLOS DE FLUJO DE DATOS DE LAS FUNCIONES ANSI A LAS CUALES REEMPLAZAN

Categoría del flujo de datos	Ejemplos de las funciones ANSI
E/S estándar	scanf(...), printf(...)
E/S de archivos	fopen(...) . fread(...) . fclose(...)
Formato en la memoria	scanf(...), sprintf(...)

Tabla 1.3.3.1 Categorías y ejemplos de flujo de datos.

Aunque es posible utilizar funciones de E/S de flujo de datos y de E/S de ANSI juntas en un programa, no es recomendable hacerlo debido a los problemas de sincronizacion con la activación de buffers entre las dos; no se necesitan utilizar las antiguas funciones de ANSI con programas escritos en Visual C++ a menos que tenga programas anteriores escritos en C que desee actualizar a Visual C++.

E/S de Flujos de Datos Estandar con Tipos de Datos.

Los flujos de datos incorporan muchas de las características elegantes de la sobrecarga de operadores y orientación por objetos al sistema de E/S. El objetivo de los flujos de datos es hacer posible el uso de una notación uniforme en diferentes tipos para

entrada y salida, y el compilador debe encargarse de discernir los detalles. Una proposición de flujo de datos emplea la siguiente notación:

```
input_stream >> typed_variable;
output_stream << typed_variable;
```

El objeto del flujo de datos se coloca siempre en el lado izquierdo de la expresión. Los operadores >> y << se utilizan para indicar el flujo de datos de un objeto al otro. Cualquier tipo integrado de Visual C++ se puede utilizar con el sistema de E/S de flujo de datos.Todos los ejemplos que se muestran a continuación son válidos en Visual C++.

```
include <iostream.h>
    void main ()
    {
        int a;
        char c;
        float f;
        double d;
            cin >> a;
        cin >> c;
        cin >> f;
        cin >> d;
            cout << a;
            cout << c;
        cout << f;
        cout << d;
        cout << "\ nThis is a short string";
        cout << "\nAnd this is a three"
```

```
<< "\nline massage that"
<< "\nis a little longer";
```

Visual C++ mantiene la definición de trayectorias de datos estándar para entrada y salida, pero emplea flujos de datos en vez de archivos. El flujo de datos `cin` es el flujo de datos de entrada estándar `cin` es un flujo de datos de entrada de caracteres que suele estar conectado al teclado del operador. Reemplaza a `stdin` que se utiliza para el mismo fin en ANSI C. El flujo de datos `cout` es la contraparte de `stdout`, y normalmente está conectado a un dispositivo de salida de caracteres. En Borland C++ se definen otros flujos de datos estándar: `cerr` sustituye a `stderr` como trayectoria de salida estándar y `clog` que no tiene contraparte en ANSI C, es similar a `cerr`. Sin embargo, `clog` proporciona el uso de buffers, en tanto que `cin`, `cout` y `cerr` no tienen esta característica.

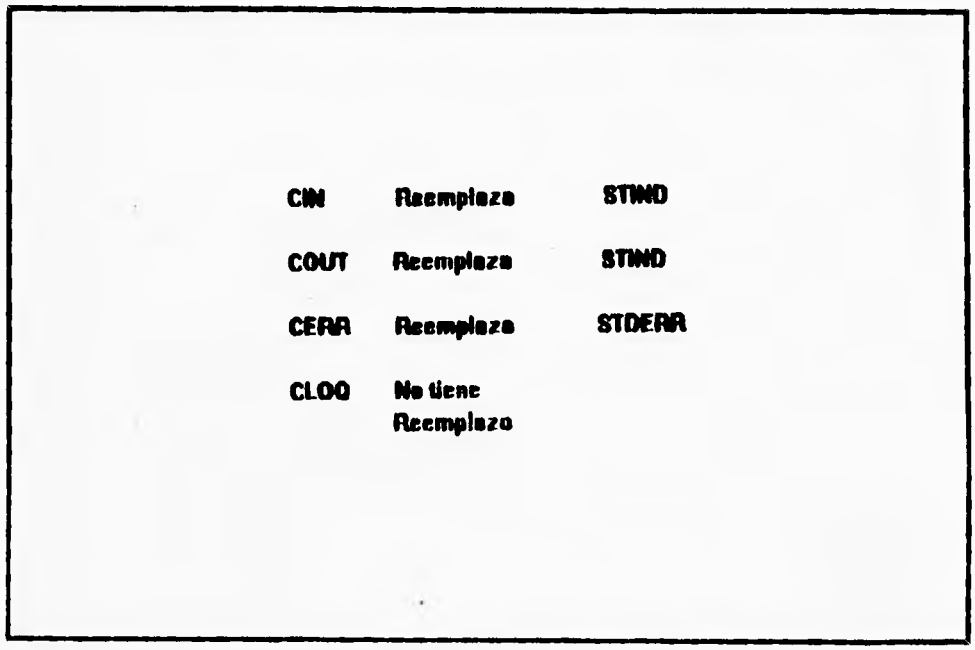


Fig. 1.3.3.1 Cambios en la E/S de flujo de datos.

Los cuatro flujos de datos estandard, `cin`, `cout`, `cerr` y `clog`, se abren automáticamente antes de que se ejecute la función `main()`; se cierran después de concluir el procesamiento de `main()`. Para utilizar cualquiera de estos flujos de datos necesita `#include` el archivo de encabezados estándar `iostream.h` en su archivo. Debe aparecer un objeto de un flujo de datos del lado izquierdo de los operadores << o >>, no obstante, se pueden ligar múltiples operaciones con flujos de datos en una misma línea aún cuando se refieran a objetos de tipos diferentes. La secuencia >> recibe el nombre de operador de extracción, ya que se utiliza para tomar caracteres de un flujo de datos. La secuencia << se denomina operador de inserción, ya que se utiliza para colocar caracteres en un flujo de datos. En el listado se describe de manera concisa:

```
void main ()
                {
     int a;
     char c;
     float f;
     double d;
      cin >> a >> c >> f >> d;          //meter todos
los          datos
      cout << a << c <<< f <<d;  // sacar todos los
datos
```

En la primera proposición, el compilador genera cuatro operaciones de entrada independientes, una para cada variable. El operador >> es asociativo de izquierda a derecha; de modo que la variable a se lee antes que la variable c. En la segunda proposición, se realizan cuatro operaciones de salida. El operador << también es asociativo de izquierda a derecha; así que se genera antes que c. Todo el formato de E/S lo proporcionan

automáticamente los flujos de datos mismos, lo cual reduce el nivel de detalle al mínimo. El formato se puede controlar de diferentes manera, empleando funciones manipuladoras especiales de flujos de datos. Por el contrario, las funciones de entrada de ANSI C requieren que el programador proporcione varios argumentos, requisito que aumenta la posibilidad de errores. La notación del flujo de datos es tan simple que es mucho menos probable que aparezcan errores ocultos, con respecto al antiguo ANSI C.

Las operaciones con caracteres en flujos de datos son muy sencillas porque no hay formato, todo califica como un carácter válido, incluyendo los así llamados caracteres de espacio en blanco. Un caracter de espacio en blanco es un espacio, un tabulador, un cambio de línea o un caracter similar al que se utiliza normalmente para separar elementos de datos. Si los caracteres que desea leer están separados unos de otros por carateres de espacio en blanco, necesita pasar por alto el espacio o bien usar un manipulador para indicar explícitamente al sistema que lo haga.

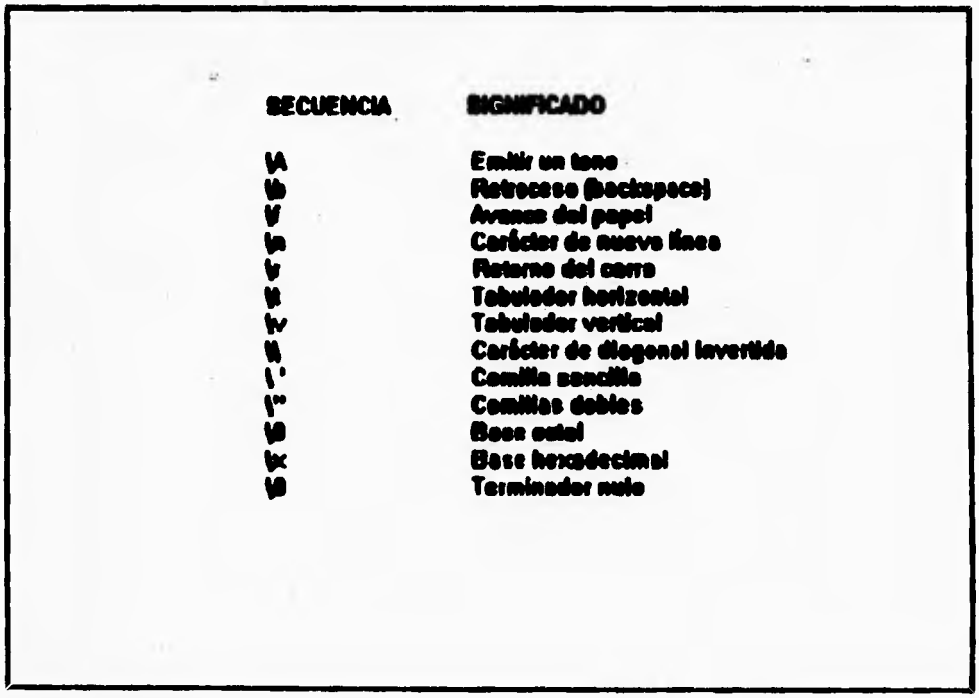
```
#include <iostream.h>
 void main ()
{
    char c;
    cout << "\nHit  a key ";
    cin >> c;
    cout << "\nThe key typed was: "<< c;
}
```

El programa presenta un mensaje en la pantalla y luego espera a que el usuario pulse una tecla seguida de un retorno del carro. Después se lee el flujo de datos `cin` con un extractor de caracteres, y se utiliza el valor en una proposición de salida. Para introducir

datos en el programa el usuario debe terminar con un retorno del carro o un cambio de línea. La lectura de caracteres de entrada antes de pulsar el retorno del carro se puede realizar recurriendo a una función que no es de flujo de datos que soporte entrada no procesada, como `getch ()` o `getche ()` ,de esta forma:

```
char c = getch ();
```

Cuando se envía caracteres a `cout` se reciben secuencias de escape, como en la biblioteca `stdio`.



SECUENCIA	SIGNIFICADO
\A	Emitir un tono
\b	Retroceso (backspace)
\f	Avance del papel
\n	Carácter de nueva línea
\r	Retorno del carro
\t	Tabulador horizontal
\v	Tabulador vertical
\\	Carácter de diagonal invertida
\'	Comilla sencilla
\"	Comillas dobles
\0	Base octal
\x	Base hexadecimal
\0	Terminador nulo

Fig. 1.3.3.2 Secuencia de escape.

La lectura y escritura de números enteros o longs es simple porque los insertores y extractores de flujos de datos hacen todo el trabajo. Cuando extrae `ints` y `longs`, el sistema pasa por alto automáticamente caracteres de espacio en blanco.

```
#include <iostream.h >
                    void main ()
        {
    cout << "\n Type a number ";
    int i;
    cin >> i;
    cout >> "\nType another number ";
    long l;
    cin >> l;
    cout << "\nThe two numbers were "
           << i << " and " << l ;
                    }
```

Se puede controlar la forma en que se leen y escriben números cambiando los siguientes atributos (Fig. 1.3.3.3).

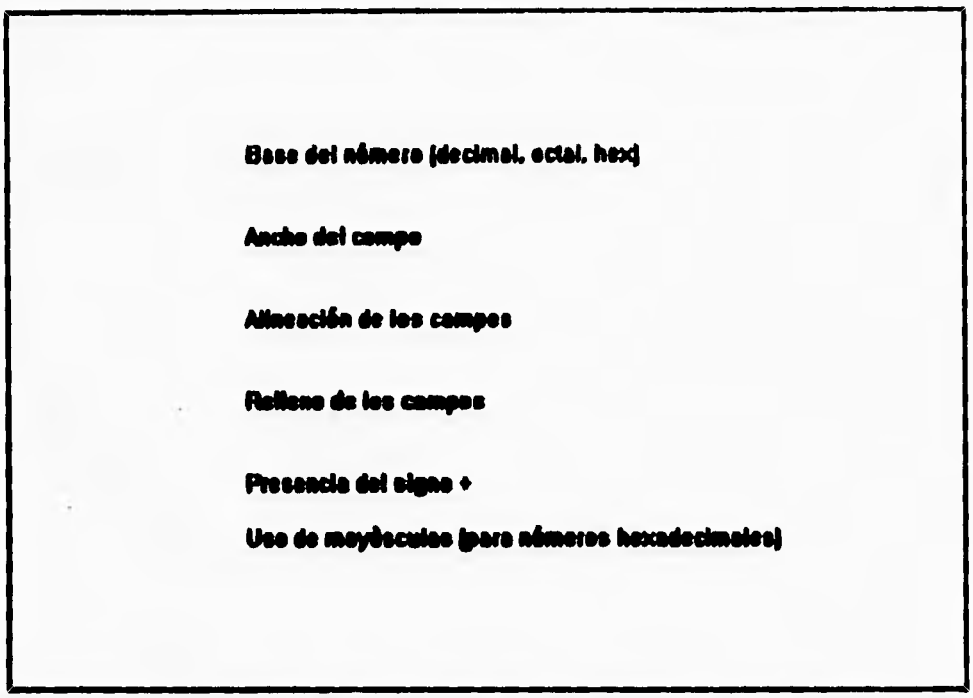


Fig. 1.3.3.3. Atributos de control.

Durante la salida, la definición es imprimir sólo los dígitos de un número, sin relleno. Durante la entrada, siempre que se encuentra un carácter que no es un dígito, un signo de más o un signo de menos, termina la operación de extracción y el valor devuelto es el número de caracteres leídos hasta ese momento.

Los tipos **floats y doubles** son tan fáciles de leer y escribir como los enteros y longs. Cuando se extraen números, primero se pasan por alto los caracteres de espacio en blanco y la extracción termina en el primer carácter que sea incompatible con números con punto flotante. Los insertores y extractores de flujos de datos integrados ofrecen comportamientos definidos que se pueden utilizar de manera directa. El formato que se utiliza para desplegar **floats y doubles (y long doubles)** se pueden cambiar utilizando manipuladores de flujos de datos. En particular, se pueden controlar los siguientes atributos:

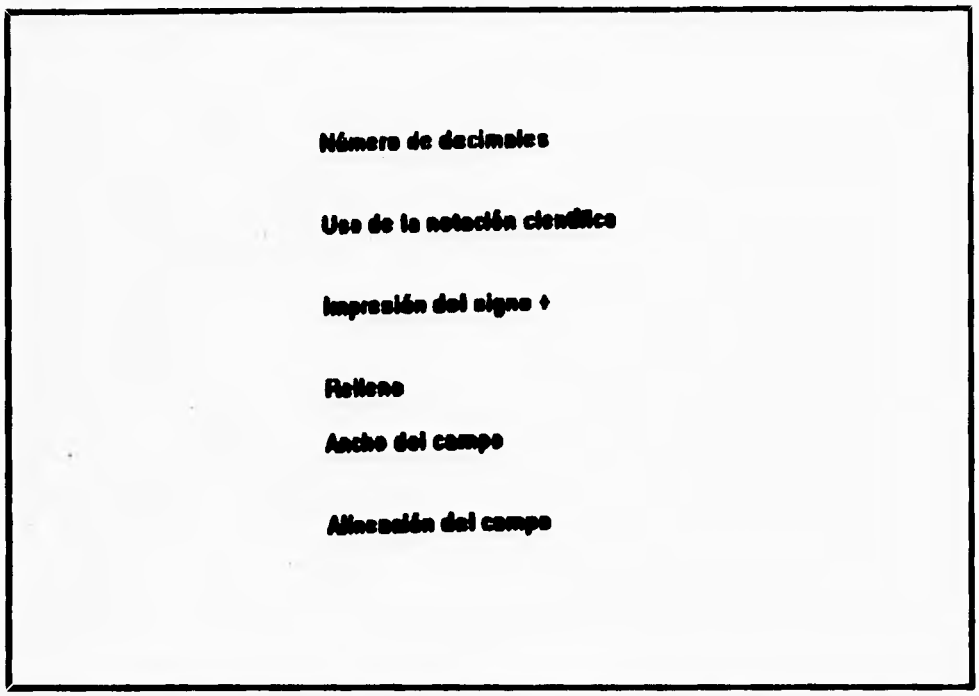


Fig. 1.3.3.4 Atributos de control para floats y doubles.

La lectura de **floats, doubles** y **long doubles** se detiene cada vez que se encuentra un carácter no válido. Este carácter no válido no se retira del flujo de datos.

E/S de Archivos con Flujo de Datos.

La elegancia y simpleza de los flujos de datos está al alcance también de archivos de usuario, tanto en modo de texto como en modo binario. Varias clases de bibliotecas manejan archivos, pero las clases que más se utilizan son **ifstream** y **ofstream.** La clase **ifstream** soporta archivos abiertos para lectura, en tanto que **ofstream** soporta archivos abiertos para escritura. Un archivo de texto se considera una secuencia de caracteres ASCII divididos en líneas. Cada línea termina con un carácter de línea nueva. Las clases de flujos de

datos de archivos tienen funciones para abrir y cerrar archivos, leer y escribir líneas de texto, y otros usos diversos.

Cuando se utilizan flujos de datos de archivos se necesita incluir el archivo estándar **fstream.h.** Para abrir un archivo para lectura, todo lo que se necesita hacer es definir un objeto **ifstream**, el cual indica el nombre del archivo. Se puede utilizar el camino absoluto de acceso; sin embargo, esto no es necesario si el archivo se encuentra en el directorio de trabajo. Se requieren dos diagonales inclinadas a la izquierda (\\) consecutivas en el nombre del archivo, ya que la diagonal se utiliza como carácter de escape dentro de cadenas, y da un significado especial al siguiente carácter. Con la proposición `ifstream` help_file (NOMBREDEARCHIVO), se define un objeto de un archivo de entrada con un nombre de archivo dado.El objeto intenta abrir el archivo nombrado en modo de entrada; no obstante, como no hay valor de retorno, no se sabe de inmediato si la operación tuvo éxito o no. Hay varias formas de manejar caracteres en un archivo de texto. Puede utilizar una proposición como

```
help_file.getline (buffer, sizeof (buffer ) ) ;
```

que toma la siguiente cadena de caracteres hasta la terminación de la línea nueva. También puede manipular caracteres individuales u otros tipos de datos integrados como:

```
char c;
char* cp;
int i ;
long l;
float f ;
double d;
help_file >> c >> cp >> i >> l >> l >> f >> d ;
```

Una de las características convenientes en Visual C++ es que los objetos tienden a encargarse de ellos mismos. Si abre un archivo con una variable **ifstream** local, el archivo es cerrado automáticamente cuando termina la función; de modo que a menudo no necesita cerrar archivos utilizando código explícito del lenguaje Visual C++. En general, cualquier archivo se cierra cuando sale del ámbito el objeto del flujo de datos asociado con él.

Prueba de errores en un Flujo de Datos.

Durante el curso de operaciones de flujos de datos pueden ocurrir errores. Se podrá intentar abrir un archivo inexistente, leer más allá del final del archivo, insertar datos con un formato no válido, etc. Todas estas condiciones causan errores, que deben detectarse y manejarse. Hay varios métodos para probar la existencia de errores en un flujo de datos. El más sencillo consiste en utilizar expresiones como éstas:

```
//pruebe si hay errores en el flujo de datos
if (istream)
  //ocurrieron errores
if (stream)
// no ocurrieron errores
```

Estos dos métodos son posibles porque se sobrecargan **operator!** y **operator void*** para flujos de datos. Se pueden utilizar funciones especiales de la clase para investigar la causa de errores, y se pueden leer con la función **ios :: rdstate ()**. Una forma alternativa de probar la existencia de errores en flujos de datos tiene una esencia más similar a ANSI C.

```
if (stream.bad ()  )
   //ocurrieron errores
if (stream. eof ()  )
    // final del archivo
if (stream. good ()  )
    //no ocurrieron errores
```

Como se puede emplear el nombre de un flujo de datos para detectar errores, es posible la expresión

```
// despliegue el archivo
while  (help_file )  {
        // copie el archivo a cout
}
```

Se ejecuta el ciclo hasta que ocurre un error de alguna clase, que es comunmente una condición de final de archivo. Como el ciclo maneja dos flujos de datos (help_file y cout) resulta aconsejable abortar la secuencia si ocurren errores en uno u otro flujo de datos. El código demuestra cómo puede probar más de un flujo de datos a la vez para detectar errores.

```
//mejor pruebe todos los flujos de datos implicados en
el ciclo
while ( help_file  &&  cout)  {
        // copie el archivo a cout
}
```

Cuando ocurre un error durante una operación con un flujo de datos, se define uno o más de los indicadores de error

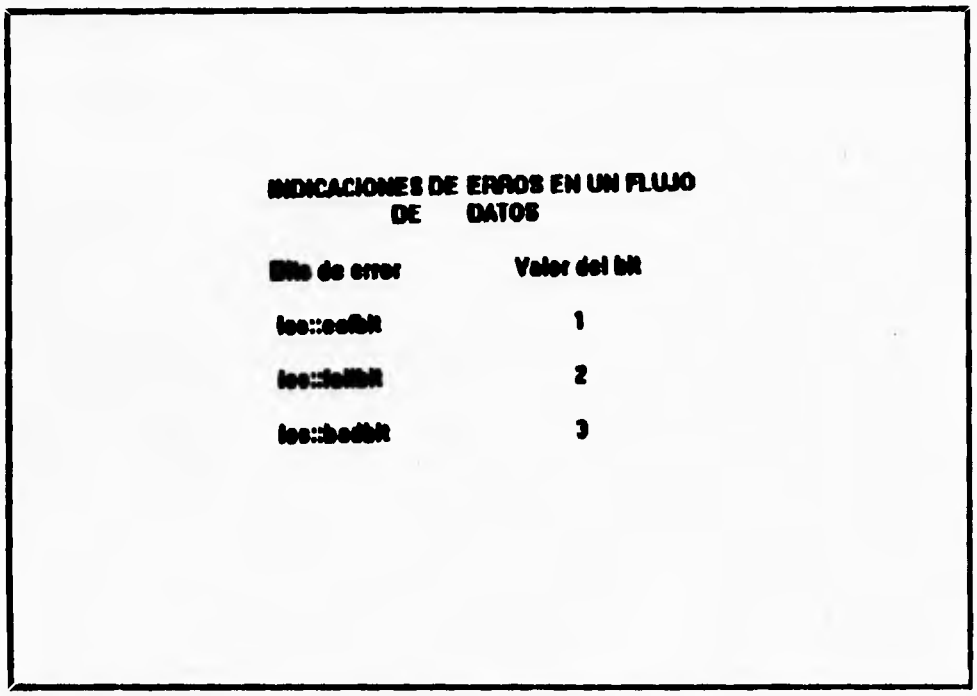


Fig. 1.3.3.5 Ejemplo de indicadores de error.

Se pueden leer los bits con errores utilizando la función **ios::rdstate().** Una vez que se define un bit con error, todas las inserciones o extracciones porteriores de ese flujo de datos fracasan hasta que se elimina de manera explícita el bit con el errores. Algunos bits con errors se eliminan utilizando la función **ios :: clear(int).**

Si se escribe un insertor o extractor propio para flujos de datos, quizá se necesite definir un bit con error.Por ejemplo:

```
//    inicie el estado de error a ios :: failbit
//    reinicie todos los otros indicadores
file.clear  (ios : : failbit );
```

Si sólo se desea eliminar un bit específico sin alterar los demás, se podría hacer esto:

```
//inicie un bit de error sin borrar los otros
file.clear (file.rdstate () | ios :: failbit);
```

A la inversa, para eliminar sólo un bit con error, esto es lo que se aplica:

```
//     Anule sólo un bit de error del flujo de datos
//      sin afectar a los otros
file.clear   (file.rdstate () & -ios :: failbit);
```

Uso de archivos de texto para salida

Escribir información en un archivo es casi tan fácil como leer su contenido. Existe una complicación ligera, en la que cuando se abre un archivo en el modo de salida se necesita saber cómo se escribirá en el archivo. La definición es truncar archivos existentes a una longitud de 0 al momento de la apertura; pero se puede cambiar la definición con parámetros operacionales.

La proposición `copy_file << buffer << "\n " ;` escribe una cadena con terminación nula en el archivo y luego agrega el carácter de la línea nueva de modo que más tarde se pueda leer el archivo con los extractores de caracteres estándar. También puede escribir caracteres individuales u otros tipos de datos integrados de esta forma:

```
char c;
char* cp;
int i ;
long l;
float f;
double  d;
copi_file  <<  c  <<  cp  <<  << i  << l  << f   << d ;
```

Para abrir un archivo y escribir en él sin que se pierda el texto que ya contienen se puede utilizar el modo de agregar

```
//abra un archivo de salida en el modo agregar
ofstream copu_file ("COPY", ios :: app);
```

Con el cambio, la ejecución del programa anterior agrega una copia de las primeras cuatro líneas de README a COPY cada vez que se ejecuta. Quizá se necesite imponer otras condiciones cuando se abra un archivo de salida, como cerciorarse de que el archivo no exista ya, o cerciorarse de que el archivo ya exista. Ambas condiciones se manejan a través de indicadores cuando se crea un objeto de un flujo de datos de un archivo.

Para detectar errores que ocurren como resultado de una condición de archivo abierto que se ha definido, se pueden emplear las técnicas de procesamiento de errores.

Los archivos binarios necesitan ser tratados de manera diferente de los archivos de texto porque no contienen caracteres de espacio en blanco, los datos no están organizados en líneas con terminaciones de línea nueva y básicamente puede ocurrir cualquier valor de ocho bits en ellos. Aun así, el manejo de archivos binarios es directo. Abrir y cerrar archivos binarios es igual que en el caso de archivos de texto, salvo por el indicador **ios::binary** que se pasa a la función de apertura.

Existen dos métodos básicos para escribir datos binarios en un archivo: escribir un carácter a la vez y escribir arreglos de caracteres (no necesariamente con terminación nula). El primer método emplea la función **put(char)** en tanto que el segundo emplea **write (char*, int)**.

Formato de Memoria

Muchas veces se necesita dar formato a datos y desplegarlos en la pantalla. Los programas escritos en lenguaje ANSI C utilizan la función **sorubtf (...)** para este fin. Los datos con formato se archivan en un buffer de la memoria para su uso posterior. La biblioteca de flujos de datos viene con la clase de formato en la memoria llamada **stream**, que puede utilizar para inserciones y extracciones.

1.3.4 ESTRUCTURAS JERÁRQUICAS DE TIPOS Y SUBTIPOS

Ya se mencionó anterormente lo que es una clase, pero es importante que se recuerde pues en eso cosisten las jerarquías de tipos y subtipos.

Una clase es la que describe el comportamiento de los tipos de datos al definirlos dentro de una interface para todas las operaciones que pueden ser desarrolladas fundamentalmente. Las operaciones que se definen dentro de las clases se les llama metodos, estos métodos son analogos a los procedimientos y funciones efectuadas dentro de un lenguaje que no es orientado a objetos.

CLASES	SUBCLASES
Describen el comportamiento de los tipos de datos	Describen el comportamiento de un grupo de objetos
Las operaciones que se definen se les llama métodos	Los objetos heredan las características de los patrones de las clases

Fig. 1.3.4.1 Características de las Clases y Subclases.

Una definición que no se ha mencionado es la de subclase; las subclases son las que definen el comportamiento de un grupo de objetos, los cuales heredan algunas de las características de los patrones de las clases, pero características especiales que no se comparten con los patrones. El costo y la complejidad del software que se desarrolla en una subclase es mucho menor; las subclases pueden incrementar la resolución de problemas en lugar de modificar los componentes del software ya existente o aún peor, reescribirlo, las subclases son creadas de un grupo de clases base,los objetos de esta nueva clase forman la arquitectura del software.

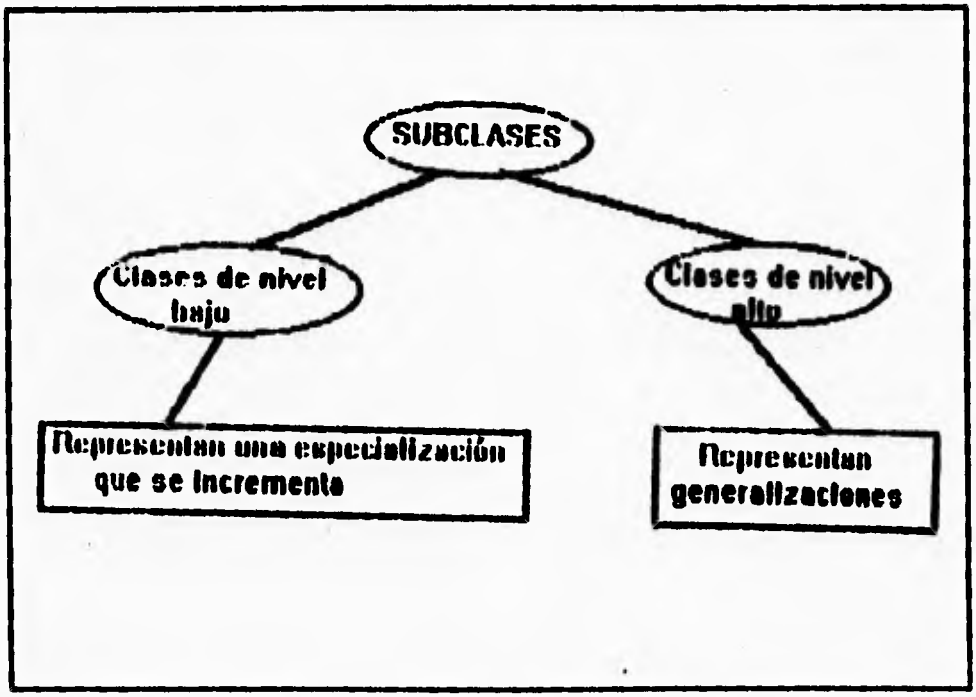


Fig. 1.3.4.2 Representación de las subclases.

Las subclases son consideradas como casos especiales de las clases bajas en donde se encuentran estructuradas jerárquicamente; las clases de nivel mas bajo casi siempre

representan una especialización que se incrementa, y las clases de nivel más alto representan mas generalizaciones. En casi todos los lenguajes orientados a objetos, si una clase P es un patron de una subclase S, entonces un objeto s de una subclase S puede ser usada en cualquier lugar que se encuentre un objeto p de un patrón de clase P; esto implica que se tengan mensajes en común y que puedan ser utilizados tanto para la clase S como para la clase P.

Algunas de las facilidades que tienen los lenguajes orientados a objetos es que pueden ser divididos dentro de jerarquías:

1.- Partición del software dentro de clases.- Modelar un proceso dentro de una clase puede parecer no muy natural. Por ejemplo, cuando se escribe un libro de bases de datos, estos programas pueden hacer modificaciones, guardar información, y borrar nombres y grados de los estudiantes e imprimir reportes de estos nombres y grados; el procedimiento común sería definir un tipo de dato y después definir el método que se va ha utilizar, modificar, borrar, e imprimir; así como también las operaciones asociadas con la base de datos.

Todas estas operaciones deben estar asociadas con un proceso llamado "editar con" , y además este proceso puede ser declarado como una clase este proceso puede ser reusado en otras aplicaciones que requieran interactuar la edición con la información de la base de datos.

2.- Agregar funcionalidades a un sistema de software ya existente.- Cada función o proceso dentro de un programa orientado a objetos está asociado con una clase;cuando una nueva función es agregada a un sistema ya existente, el programador debe decidir agregar un procedimiento particular a una clase ya existente o crear una nueva clase o más aún, crear una subclase. El procedimiento de rehuso es el factor principal para la determinación de donde se va a colocar la nueva función dentro de un sistema orientado a objetos; si la función que se agregó es rechazada por algunos objetos de las clases ya existentes es

mejor que se cree una nueva clase basada en la función que se está agregando: si la implementación de las nuevas funciones requieren acceso a detalles internos de una clase, lo que se sugiere es agregar una nueva función como un método adicional a la clase existente. Si la función que se está agregando representa una modificación de una función ya existente, lo que se debe de hacer es crear una subclase y agregar funciones a esta subclase para que funcionen como métodos similares a los de la clase.

3.- **Jerarquías en las estructuras de clases y subclases.-** Un procedimiento para crear clases y subclases es el conocido como **Top-down decomposition,** en el cual el programador identifica y modela la mayoría de los elementos en el sistema. Los componentes de mayor nivel (Clases) son divididos dentro de más subclases, y este proceso continúa hasta que se tiene la jerarquía estructurada de las clases y objetos que se construyen.

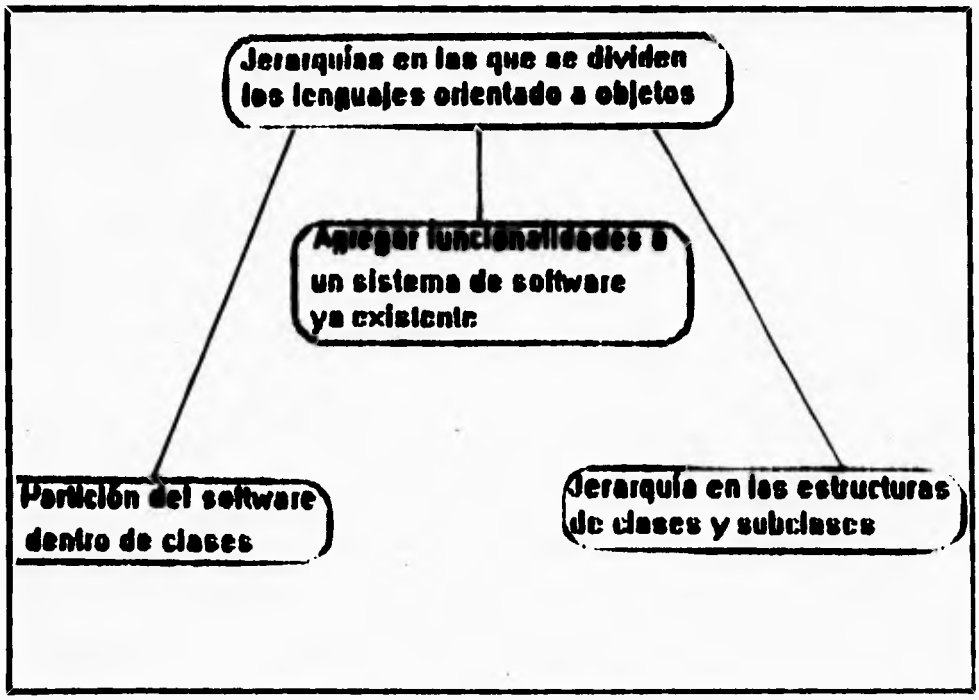


Fig. 1.3.4.3 División de Jerarquías.

Para poder explicar mejor esto se escogió el siguiente ejemplo:

Considerese un automóvil como la clase de nivel mayor en el sistema; la clase automovil puede ser dividido dentro de subclases llamadas máquina transmisión, frenos, tren de manejo, sistema de ahogamiento, suspensión y así sucesivamente. La subclase máquina así como las demás subclases pueden a su vez ser divididas en componentes como arrancador, inyección completa y encendido; la subclase arrancador puede también estar dividido en solenoide y bujías. Para cada una de las clases o subclases se tienen operaciones pertinentes (metodos) que deben ser definidas.

Usando la jerarquía top-down las clases que se encuentran hasta arriba de la jerarquía abarcan a todos los métodos y características que son comunes para todas las subclases que se encuentan abajo de la jerarquía.

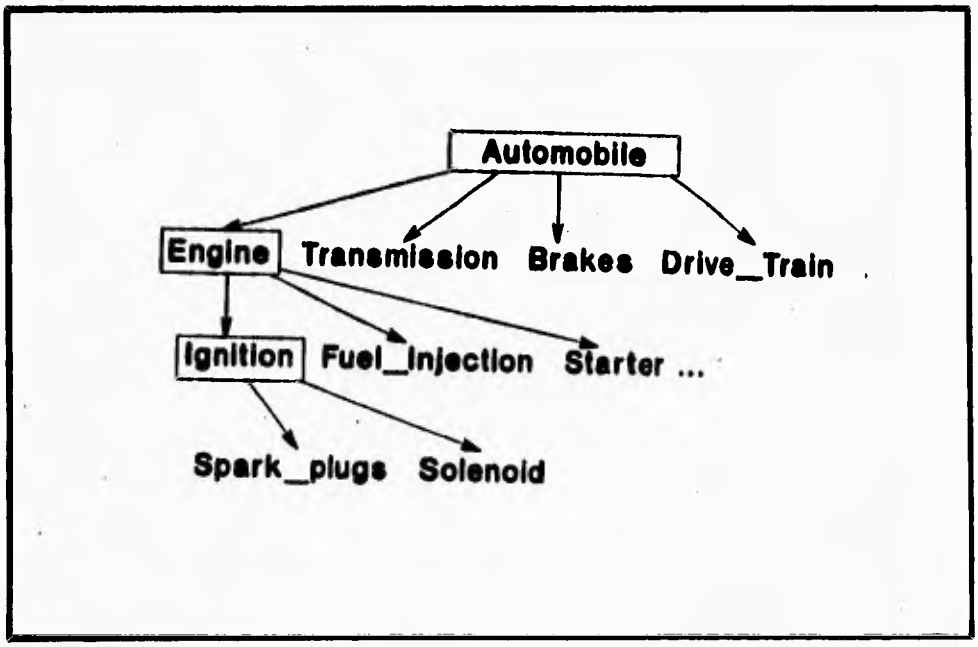


Fig. 1.3.4.4 Jerarquía top-down.

Otro procedimiento para crear clases y subclases es el conocido con el nombre de **bottom-up** de **composition**, en el cual las subclases extienden un patron de las clases y proveen gran funcionalidad a los objetos de cada subclase sucesiva.

Un ejemplo muy común para mostrar el procedimiento sería por ejemplo, una ventana pantalla como el patron de la clase. Una serie de subclases pueden ser construidas y estas a su vez agregaran funcionalidades a la clase basica, en este caso a la ventana pantalla, por medio de objetos tomados de varias subclases en donde se definirán sus atributos como por ejemplo color, calidad del borde de la ventana, y así sucesivamente.

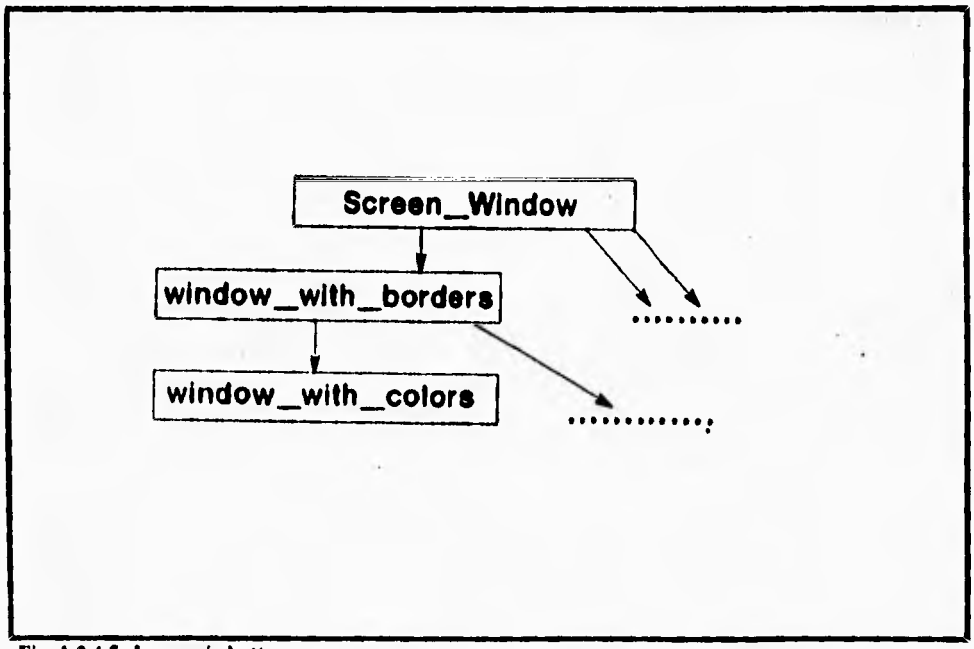


Fig. 1.3.4.5 Jerarquía bottom-up.

Otro procedimiento para crear clases y subclases es el conocido con el nombre de **bottom-up de composition**, en el cual las subclases extienden un patron de las clases y proveen gran funcionalidad a los objetos de cada subclase sucesiva.

Un ejemplo muy común para mostrar el procedimiento sería por ejemplo, una ventana pantalla como el patron de la clase. Una serie de subclases pueden ser construidas y estas a su vez agregaran funcionalidades a la clase basica, en este caso a la ventana pantalla, por medio de objetos tomados de varias subclases en donde se definirán sus atributos como por ejemplo color, calidad del borde de la ventana, y así sucesivamente.

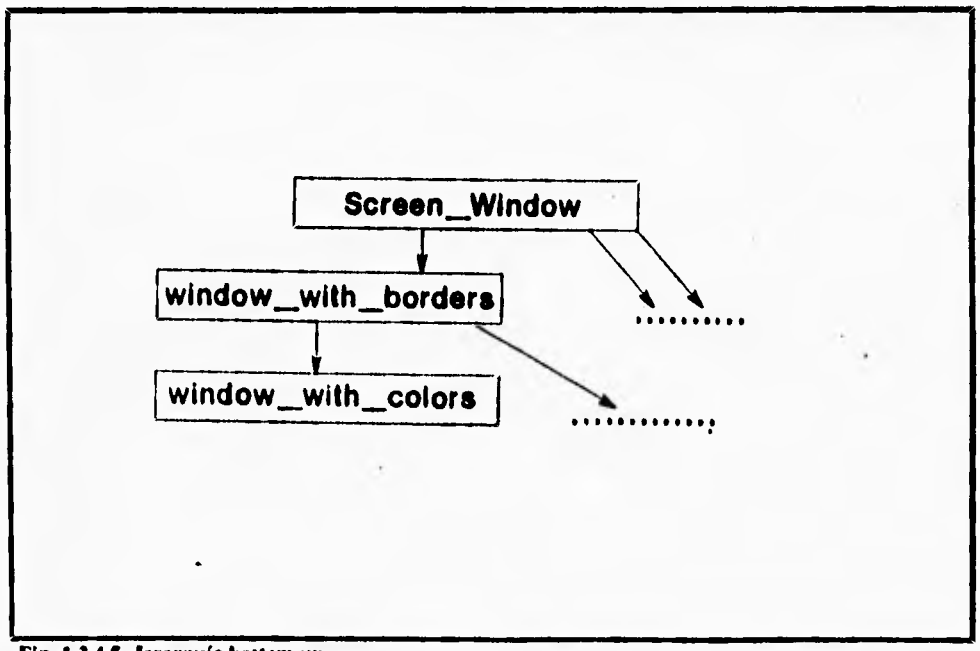


Fig. 1.3.4.5 Jerarquía bottom-up.

1.4 Caracteristicas y funcionamiento de las Bases de Datos Relacionales

El manejo y organización en una base de Datos puede ser vista en 4 niveles de Arquitectura que van desde el nivel de Organización Lógico hasta su correspondiente en el nivel de Organización Fisico, desde el más independiente del equipo hasta el más dependiente.

En el nivel 4, el más alto es donde se hacen sólo las consideraciones lógicas de Datos, donde se hace el análisis de necesidades del usuario, de los datos con que cuenta y los que necesita, se especifica el flujo de la información y sus transformaciones. Posteriormente se definen entidades u objetos y sus relaciones funcionales. A partir de éstas entidades y relaciones funcionales se construye o modela la base utilizando los modelos lógicos de organización, que serán explicados posteriormente. Cabe mencionar que éste nivel es el más independiente del computador donde se va a implementar la base de datos.

El siguiente nivel, se orienta a la implementación de los modelos de organización, se abordan acciones orientadas al desempeño, se empiezan a tomar en cuenta las limitaciones de los dispositivos fisicos. Aquí se definen las estratégias de acceso para mejorar el desempeño en el manejo de datos. Por ejemplo, en este nivel se decide qué organizaciones de archivo se utilizarán, si son de acceso secuencial, random ó por organizaciones secundarias (listas invertidas, multilistas, etc.) y sus posibles variaciones.

A los niveles 3 y 4 se les llama interface física-lógica. El nivel 2 refleja alternativas de organización Física y formatos de almacenamiento de datos en dispositivos de almacenamiento lineal para cada una de las organizaciones definidas en el nivel 3. Algunos autores sólo reconocen 3 niveles y manejan como uno solo al 3 y al 2.

El nivel 1, es más dependiente que los anteriores del equipo donde se implementa la Base de Datos, se refieren a las características y aspectos particulares de los dispositivos físicos de almacenamiento. En este nivel se ven aspectos como la organización de los

dispositivos en subdivisiones tales como bloques y sectores; el mapeo de datos dentro de estas subdivisiones; el alamcenamiento de áreas de sobreflujo y características dependientes de cada máquina, así como la localización de datos de control, verificación de integridad etc.

ARQUITECTURA CODASYL

El grupo de trabajo para bases de datos (DBTG) de Codasyl presentó su reporte a inicios de 1971. En 1973 y 1978 su propuesta fué revisada y modificada sin que cambiaran sus conceptos básicos, muchos manejadores de Bases de Datos la tomaron como estándar en la década de los setenta y parte de los ochenta. En Codasyl se proponen 3 niveles de organización de datos con su lenguaje asociado, más un lenguaje para procesar esos datos. Estos lenguajes son:

- El lenguaje de definición de Datos a nivel de esquema(Schema DDL). Donde se hace la definición lógica de la Base entera.
- El lenguaje de definición de datos a nivel subesquema (subesquema DDL). Aquí se tiene la definición lógica de vistas de usuario
- El lenguaje de manipulación de Datos (DML), estilo Cobol con comandos como OPEN,CLOSE, GET, FIND, STORE, MODIFY, etc.
- El lenguaje de control de dispositivos/medios (DMCL), que finalmente fué restructurado y renombrado como lenguaje de descripción de almacenamiento de datos (DSDL), en el que se tiene una descripción única y formal de las estrategias de almacenamiento físico y de controles expresada en instrucciones de alto nivel.

La arquitectura Codasyl se muestra en la figura.1.4.1. En ésta arquitectura el esquema es la descripción lógica de la base de datos entera, está formada por una descripción de los varios tipos de registro involucrados y de los tipos de conjuntos (SET

TYPES) que los relacionan. Un tipo de registro esta formado por uno o más elementos dato; el elemento dato es la parte más pequeña de la base de datos. Un subesquema es un subconjunto lógico del esquema, o sea un subconjunto de los tipos de relaciones (sets), tipos de registros y elementos dato de los tipos de registro del esquema

NIVEL	CODASYL 71	CODASYL 78	ANSI/SPARC
NIVEL 4 LOGICO DE USUARIO	LENGUAJE DE DEFINICION DE DATOS (DDL)	LENGUAJE DE DEFINICION DE DATOS (DDL)	MODELO EXTERNO
NIVEL 3 LOGICO GLOBAL	LENGUAJE DE DEFINICION DE DATOS (DDL)	LENGUAJE DE DEFINICION DE DATOS (DDL)	MODELO CONCEPTUAL
NIVEL 2 DE ORGANIZACION FISICA	LENGUAJE DE DEFINICION DE DATOS (DDL)	LENGUAJE DE DESCRIPCION DE ALMACENAMIENTO DE DATOS	MODELO INTERNO
	LENGUAJE DE CONTROL DE DISPOSITIVOS/ MEDIOS		
NIVEL 1 DE DISPOSITIVOS FISICOS	LENGUAJE DE CONTROL DE DISPOSITIVOS/ MEDIOS	LENGUAJE DE DESCRIPCION DE ALMACENAMIENTO DE DATOS	

Fig. 1.4.1 Arquitectura Codasyl.

En la figura 1.4.2 se muestra un ejemplo, el subesquema representa la vista o parte correspondiente de la base que le pertenece o utiliza un usuario. Todos los tipos de registro, tipos de relaciones y elementos dato que no estén en un esquema no podrán ser definidos ni introducidos a la base mediante un subesquema. En un esquema se pueden definir cualquier número arbitrario de subesquemas.

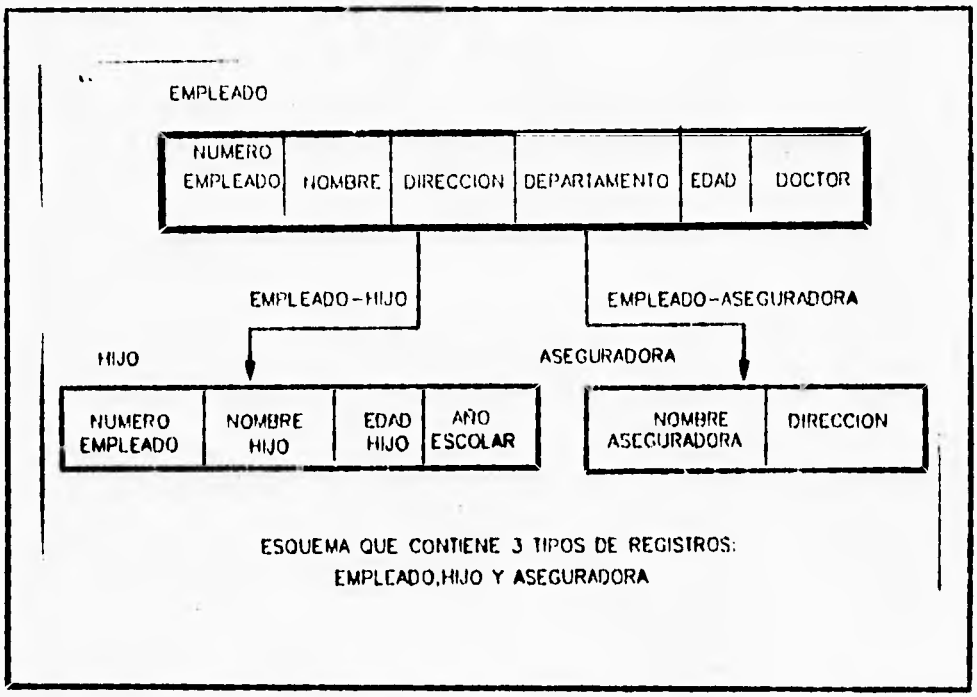


Fig. 1.4.2 Parte de la base correspondiente al empleado.

ARQUITECTURAS ANSI/SPARC

En 1972 fue fundado el grupo de estudio de ANSI/X3/SPARC sobre sistemas de Administración de Bases de Datos por el Standards Planning and Requirements Committee (SPARC) de ANSI/X3/, para determinar las áreas, si las había, de la tecnología de bases de datos para los cuales la estandarización era adecuada y producir un conjunto de recomendaciones de acción en cada una de las mismas. Este grupo definió una arquitectura generalizada que presentó en un reporte provisional en 1975.

En esta arquitectura se contemplan 3 niveles:

1.- El nivel externo. En este nivel se aborda la definición de subesquemas de la Base de Datos, es donde se revisan y se definen las vistas de usuario; se cuenta con la lista de entidades, sus atributos, sus características. Aquí se tiene el control de acceso a la Base

de Datos y cada usuario verá o accesará su información de uno o varios esquemas externos. Los esquemas externos pueden traslaparse para reflejar modelos de datos.

En este nivel se cuenta con diversos lenguajes orientado a aplicaciones, por ejemplo: Pascal y C, que cuentan con sublenguajes para la definición de datos (DDL) que sirven para la descripción de los objetos de la base tal como los ve el usuario y sublenguajes de manipulación de datos (DML) que apoyan el manejo o procesamiento de esos objetos.

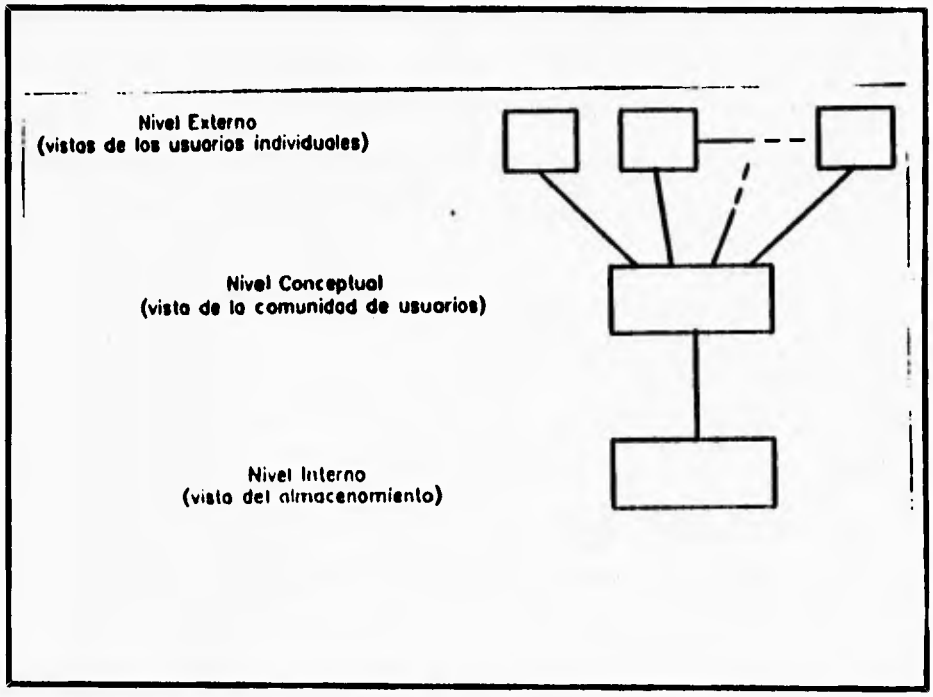


Fig. 1.4.3 Niveles de la Arquitectura ANSI/SPARC

2.- El nivel conceptual. En este nivel se implanta la Base de Datos en su forma más general, se define el esquema de la Base de Datos, sus vínculos, relaciones, etc. La función principal en este nivel es el diseño y la generación del esquema. Los esquemas reflejan el

mundo real, el negocio, la sociedad, y si éstos cambian, habrá que modificar el esquema conceptual y con esto redefinir o ajustar esquemas externos.

Todavía en este nivel se representa el contenido total de la base de datos en forma relativamente abstracta, en comparación con la forma en que los datos son almacenados físicamente. Un registo en este nivel es llamado registro conceptual y no siempre es idéntico a un registro del nivel externo ni a un registro almacenado.

Para la definición de esquemas se utiliza el lenguaje de definición de datos (DDL) conceptual. Si se desea lograr independencia de los datos, estas definiciones no deben incluir ninguna consideración sobre la estructura de almacenamiento ni sobre la estrategia de acceso, únicamente definiciones de contenido de información.

3.- El nivel interno. En este nivel se implantan las definiciones de almacenamiento a nivel registro almacenado; la vista interna aún se mantiene a un paso del nivel físico ya que no se involucra con registros físicos o bloque, ni a ninguna restricción específica de dispositivos tales como capacidades de cilindros o pistas. Aquí se define dónde están colocados los valores de atributo de la base de datos y cómo se tienen acceso a ellos; se ve el desempeño y las afinaciones para equilibrar los tiempos de respuesta de algunas transacciones, y se manejan los dispositivos físicos de almacenamiento.

Se muestra en la figura 1.4.4 un diagrama de los niveles de arquitectura de ANSI/SPARC, en relación a los niveles de arquitectura de Codasyl.

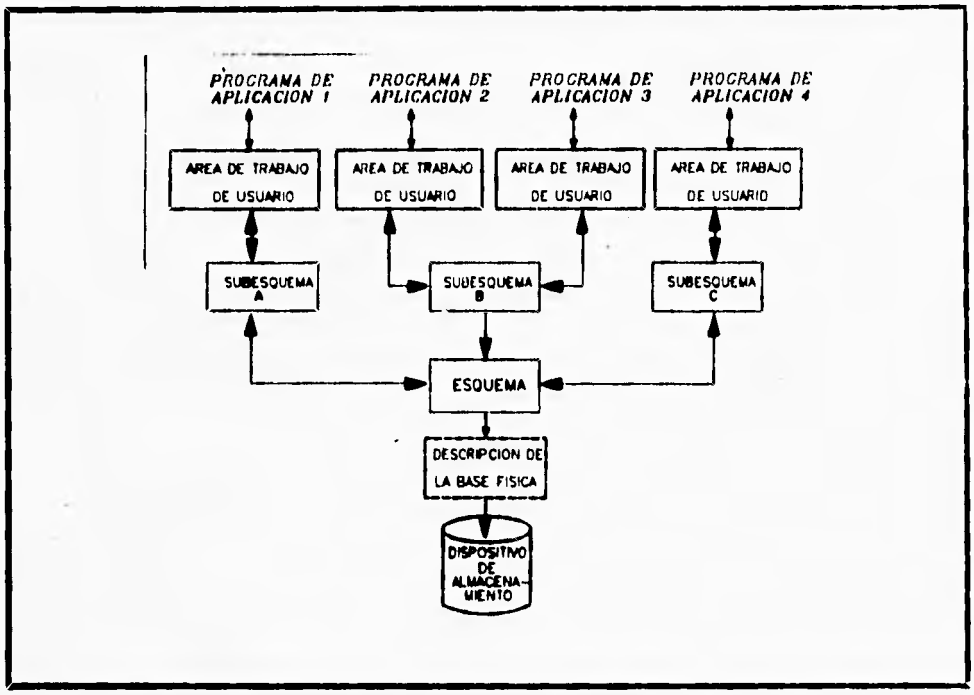


Fig. 1.4.4 Niveles de arquitectura de ANSI/SPARC.

MODELOS LOGICOS

Una base de datos es un modelo en computadora de un sistema del mundo real. El contenido de la base de datos corresponde al estado del sistema de aplicación mientras que los cambios a la base de datos corresponden a eventos del sistema. En general, si el modelo puede ser descrito en términos de estructuras naturales, el trabajo del diseñador de la base de datos se simplificará tanto para la definición inicial como para los cambios subsecuentes. Algo muy importante, es que al usuario de la base de datos le será muy fácil formular consultas a la misma si el modelo refleja el ambiente de aplicación y usa términos y conceptos familiares para él.

El mecanismo formal utilizado para expresar la estructura lógica de los datos así como la semántica asociada es llamado modelo lógico de datos. Para que un modelo sea efectivo debe representar lo más cercanamente posible, los conceptos del mundo real que son usados para estructurar la información en una organización. Los ejemplos más conocidos de modelos de datos son: el Jerárquico, el de Red, el Relacional, el de Entidad-Relación y el Semántico. Los tres primeros son los tradicionales, orientados a registros y los dos últimos son orientados a entidades.

1. **Modelo Jerárquico o de Arbol**.- Que incluye tanto estructuras jerárquicas, jerárquicas invertidas y de árbol tipo CODASYL.

2. **Modelo Reticular o de Red**.- Que incluye las estructuras de red tipo CODASYL. Cabe señalar que las estructuras de árbol o jerárquicas son un caso especial de las de red.

3. **Modelo Relacional**.- Basado en el álgebra relacional. Concibe las estructuras de datos como conjuntos de tablas, en la que se representan las entidades y sus relaciones

4. **Modelo Entidad-Relación**.- Ve al mundo en términos de entidades y relaciones entre estas.

5. **Modelo Semántico**.- Utiliza construcciones orientadas al usuario que capturan la semántica del ambiente de la aplicación. Para lograr esto, utiliza conceptos como "generalización", "agregación","datos derivados" y reglas de integridad para la definición de la base de datos.

La naturaleza de la organización de datos y sus relaciones en la mayoría de las situaciones prácticas es tal que se puede representar ya sea por árboles, relaciones sin mayor dificultad. Cualquier estructura de red puede ser convertida a una jerárquica introduciendo elementos. redundantes, también cualquier estructura de red o jerárquica puede convertirse en una estructura relacional haciendo que las relaciones sean explícitas en lugar de implícitas. Una excepción de esto son las relaciones M:N (MUCHOS A MUCHOS).

Desde las décadas pasadas y sobre todo en equipos grandes existen infinidad de sistemas manejadores de bases de datos que usan el modelo jerárquico o el de red y no ha sido hasta los 80's en que se han empezado a implementar comercialmente los manejadores relacionales tanto en computadoras personales como en equipos gandes con gran éxito.

Modelo Jerarquico o de Arbol

Haciendo una analogía con las estructuras de árbol, en las cuales se tienen nodos y una relación de jerarquía se da de un nodo padre a un nodo hijo, se define un conjunto (set) como una jeraquia de dos niveles de registros. El registro padre es llamado el propietario (owner). Cada propietario puede tener una ocurrencia del propietario y cualquier número de ocurrencias de los registro hijos llamados miembros (member). Las relaciones de propietario a miembro pueden ser 1:1 (uno a uno), 1:N (uno a muchos) pero no M:N (muchos a muchos)

Un registro propietario y uno miembro pueden ser el mismo. Una base de datos es descrita por un esquema consistente en uno o varios conjuntos (set) arreglados en una forma de árbol de muchos niveles, entonces un registro podrá ser miembro de un conjunto pero propietario de algún otro. Es fundamental, en cuanto a la vista jerárquica de los datos que cualquier ocurrencia de un registro específico se vea bajo el contexto de la relación padre-hijo.

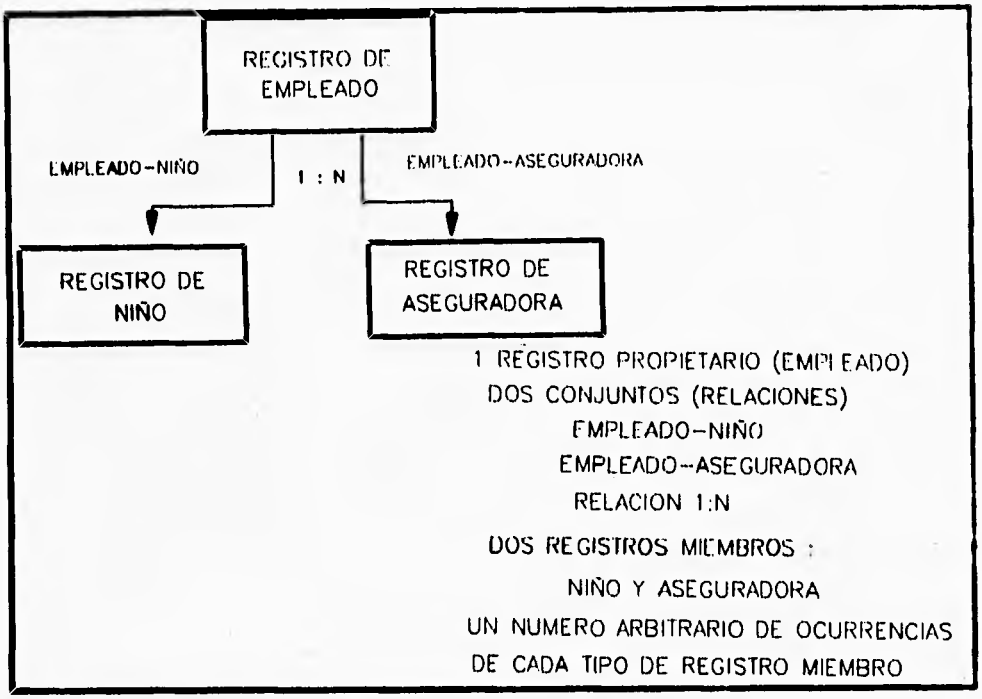


Fig. 1.4.5 Modelo jerárquico.

Las estructuras jerárquicas representan bastante bien algunas estructuras de la vida real, pero no todas las estructuras de la vida real se pueden representar con estructuras jerárquicas, sobre todo donde se dan relaciones M:N.

Modelo de Red o Reticular

En una estructura de árbol no se permite que un nodo hijo tenga más de un nodo padre. En una estructura tipo red, según CODASYL un miembro si puede tener más de un propietario siempre y cuando cada uno esté en un conjunto diferente. Una relación M:N es una red en si misma. El modelo de red permite modelar en forma directa relaciones M:N pero en éste hay un nuevo elemento que se llama conector, que se puede representar como un registro que contiene datos que describen la asociación entre propietarios y miembros.

Todas las ocurrencias de un conector para un propietario se colocan en una cadena que parte del mismo y retornan a él, igual sucede con las ocurrencias de un miembro. De esta manera cada ocurrencia del conector está en dos cadenas, en una de su propietario y en una de su miembro. Esto hace que la estructura interna de un archivo sea muy compleja, pues contiene muchos apuntadores. Un problema que presentan, es que para una misma pregunta, se puede tener acceso a la información por dos caminos, uno de los cuales, según las condiciones especificas de la pregunta, será mejor que el otro. Esto se debe tomar en cuenta cuando la programación es muy rígida.

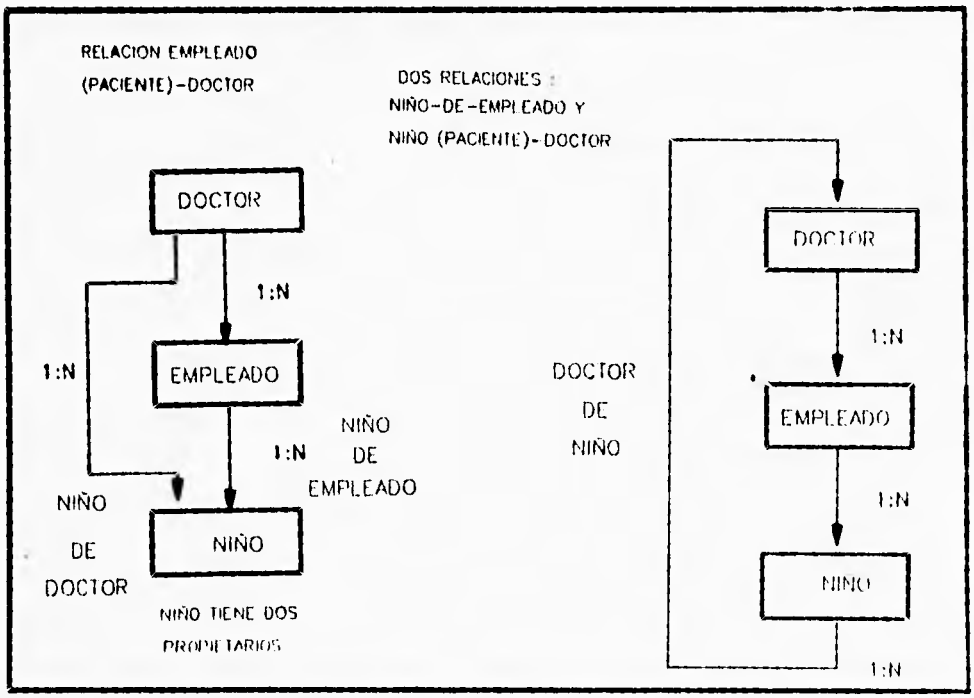


Fig. 1.4.6 Modelo Red o Reticular.

Modelo Entidad Relación (E-R)

El modelo entidad relación (E-R) está basado en una percepción del mundo real, que consiste en un conjunto de objetos básicos llamados "entidades" y las "relaciones" entre estos objetos. fué desarrollado para facilitar el diseño de las bases de datos permitiendo la especificación de un esquema. Tal esquema representa la estructura lógica de la base de datos.

Diagrama Entidad-Relación

El modelo Entidad-relación, está asociado con su respectivo diagrama de Entidad-Relación para representar las entidades y las relaciones entre estas. Esta es la mejor manera de expresar la vista global de la base de datos. Por consiguiente, se ha desarrollado una terminología para realizar diagramas de Entidad-Relación. En un Diagrama Entidad-Relación, las entidades están representadas por un rectángulo y una relación se representa con un rombo. Por ejemplo en esta se describe el número de transacciones asociadas a una cuenta bancaria en particular. El 1 y la n que están junto a los arcos indican el tipo de relación entre entidades es de 1 a muchos. Esto es, la Cuenta bancaría de un cliente puede incluir diversas transacciones (deposito, abono, etc.).

Las entidades y las relaciones poseen propiedades. Estas propiedades pueden expresarse en términos del valor de sus atributos. Por lo tanto, opcionalmente, estas pueden representarse dentro del diagrama de Entidad-Relación mediante círculos y los valores de estos se expresan en los arcos que van de la entidad al respectivo atributo. Por cada función de una aplicación, se dibuja un diagrama de Entidad-Relación para describir los requerimientos de datos. De este modo, en el mismo diagrama se describe cómo se procesarán los datos, el punto de inicio del proceso y su secuencia.

Entidades y Conjuntos de Entidad

Una entidad es un objeto que existe y es distinguible de cualquier otro. Por ejemplo, el Sr. García con el número de seguro social 890-12-3456 es una entidad, ya que el número identifica únicamente a una persona particular en el universo. Un conjunto de entidad contiene entidades del mismo tipo. El conjunto de todas las personas que poseen una cuenta bancaria, por ejemplo, puede ser definida como el conjunto de entidad denominado Clientes. De manera similar, el conjunto de entidades cuentas debe representar el conjunto de todas las Cuentas, de un banco en particular.

Los conjuntos de entidad no deben disociarse. Por ejemplo, es posible definir el conjunto de entidad de todos los empleados del banco (Empleados) y el conjunto de entidad de todos los clientes del banco (clientes).

Una entidad está representada por un conjunto de atributos. Los posibles atributos del conjunto de entidad Clientes son **nombre, seguro social, calle** y **ciudad.** Los posibles atributos para el conjunto de entidad Cuentas son el **número** y **saldo.** Para cada atributo hay un conjunto de valores permitidos, llamado el dominio de ese atributo. Por ejemplo, el dominio del atributo **nombre** debe ser el conjunto de todas las cadenas de texto de una cierta longitud. De forma análoga el dominio del atributo número debe ser el conjunto de todos los enteros positivos.

Formalmente, un atributo es una función que mapea un conjunto de entidades dentro de un dominio. Así, cada entidad está descrita por un conjunto de parejas (atributos o valores de dato), de cada atributo del conjunto de entidad. De este modo, la entidad de Clientes en particular está descrita por el conjunto {(**nombre,** García), (**seguro social,** 890-12-3456), (**calle,**Norte 4), (**ciudad** Monterrey)}, mediante la cual la entidad define a una persona llamada García con número de seguro social 890-12-3456, que reside en la calle Norte 4 en la ciudad de Monterrey. Para ilustrar la diferencia entre un conjunto de entidad y una entidad particular de un conjunto, haremos una analogía con las nociones de lenguajes de programación. Un conjunto de entidad correspondería a una definición "type". Una

variable de un "Type" determinado tiene un valor particualar en un instante determinado de tiempo. Así que, una variable de un lenguaje de programación correspondería al concepto de una entidad en el modelo E-R.

Relaciones y Conjuntos de Relación

Una relación es una asociación entre varias entidades. Por ejemplo, definimos una relación la cual asocia al cliente "García" con la cuenta 401. Esta especifica que García es un cliente con número de cuenta 401.

Un conjunto de relación es un conjunto de relaciones del mismo tipo. Formalmente, es una relación matemática sobre n >= 2 conjuntos de entidad. Si E1, E2,...,En son conjuntos de entidad, entonces un conjunto de relación R es un subconjunto de {(e1,e2,...,en)| e1 E1, e2 E2, ...en En)} donde (e1, e2,...,en) es una relación.

Para ilustrar esto, consideremos las dos entidades, Cliente y Cuentas de la figura 1.4.7. En donde se define la relación Cliente-Cuenta para denotar la asociación entre clientes y cuentas bancarias.

La relación Cliente-Cuenta es un ejemplo de conjunto de relación binaria, esto es, una en la cual se involucran dos conjuntos de entidad. Muchos de los conjuntos de relación en una base de datos son binarios. Sin embargo, ocasionalmente, hay conjuntos de relación que involucran más de un conjunto de entidad. Por ejemplo consideremos la relación ternaria (García, 401, Centro) que especifica que el cliente García tiene la cuenta 401 en la sucursal Centro. Esta relación es una instancia de una relación Cliente-cuenta-Sucursal que involucra los conjuntos de entidad cliente Cuenta y Sucursal.

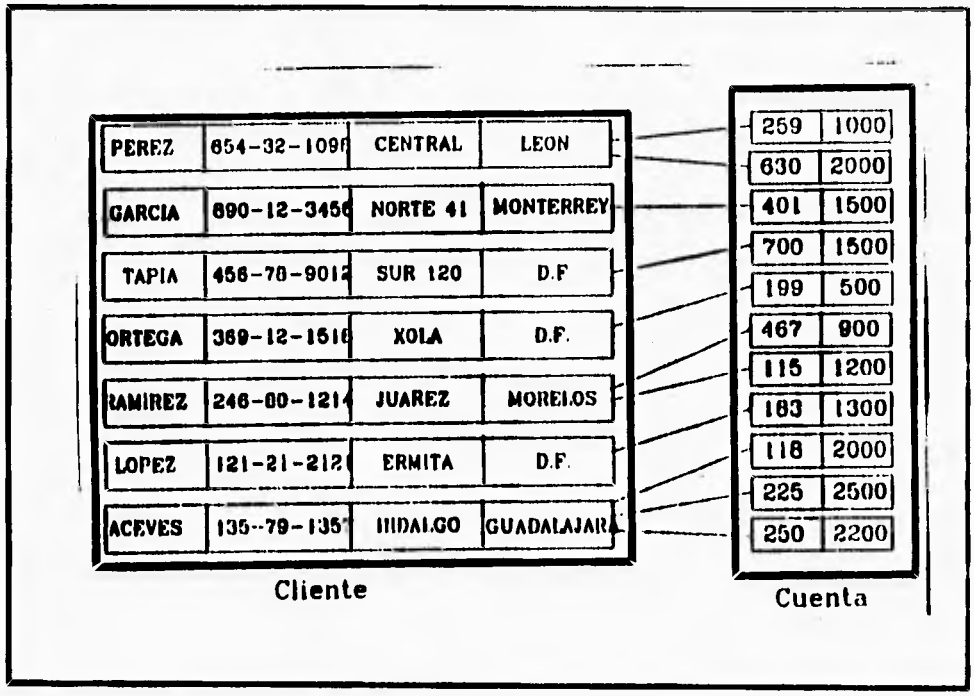


Fig. 1.4.7 Entidades Cliente y Cuentas.

La función que juega una entidad en una relación se denomina su rol. Los roles son normalmente implícitos y usualmente no son especificados. sin embargo, son utilizados cuando el manejo de una relación necesita clarificación. Tal es el caso cuando el conjunto de entidad no está bien definido. Una relación además puede tener atributos descriptivos. Por ejemplo, la fecha debe ser un atributo del conjunto de relación cliente-cuenta. Esto especifica que la última fecha en la cual el cliente tuvo acceso a la cuenta. La relación Cuenta-Cliente de (García,401) está descrita por {(fecha, mayo 23 1990)}, que indica que la última vez que García accesó la cuenta 401 fué en mayo 23 de 1990.

Tipos de Relación

Un esquema implementado por el modelo E-R define ciertas restricciones que deben conformar el contenido de la base de datos. Una de estas restricciones es el tipo de relación entre entidades, estas relaciones pueden comprenderse como un mapeo cardinal que expresa el número de entidades a la cual otra entidad puede ser asociada vía una relación.

Los mapeos cardinales son los más utilizados en la descripción de conjuntos de relación binaria, además, ocasionalmente contribuyen a la descripción de conjuntos de relación que involucran más de dos conjuntos de entidad.

Para un conjunto de relación binaria entre los conjuntos de entidad A y B, el mapeo cardinal debe ser uno de los siguientes:

- **Uno a Uno**
- **Uno a Muchos**
- **Muchos a Uno**
- **Muchos a Muchos**

Para ejemplificar lo anterior, considere el conjunto de relación Cliente-Cuenta. Si en un banco en particular una cuenta pertenece únicamente a un cliente, y un cliente puede tener varias cuentas, entonces el conjunto de relación es uno a muchos en el sentido cliente a cuenta. Si una cuenta puede pertenecer a varios clientes (como cuentas entre familiares), el conjunto de relación es muchos a muchos. Existen dependencias entre entidades. Específicamente, si la existencia de una entidad "X" depende de la existencia de la entidad " Y", entonces se dice que "X" está dependiendo existencialmente de "Y". Operativamente si "Y" es eliminada, "X" también lo será. Entonces se dice que "Y" es la entidad dominante y "X" es la entidad subordinada.

Para ilustrar esto, consideremos los conjuntos de entidad Cuenta y Transacción. Formaremos la relación Log entre estos dos conjuntos, la cual especifica que para una cuenta en particular hay varias transacciones. Esta relación es uno-a-muchos desde Cuenta a Transacción. Cada entidad de Transacción asociadas deben entonces eliminarse también. En cambio, las entidades de Transacción pueden eliminarse desde la base de datos sin afectar cualquier Cuenta. Por lo tanto, el conjunto de entidad Cuenta es dominante y el de Transacciones es el subordinado en la relación Log. La figura 1.4.8 muestra el diagrama E-R de la relación Log.

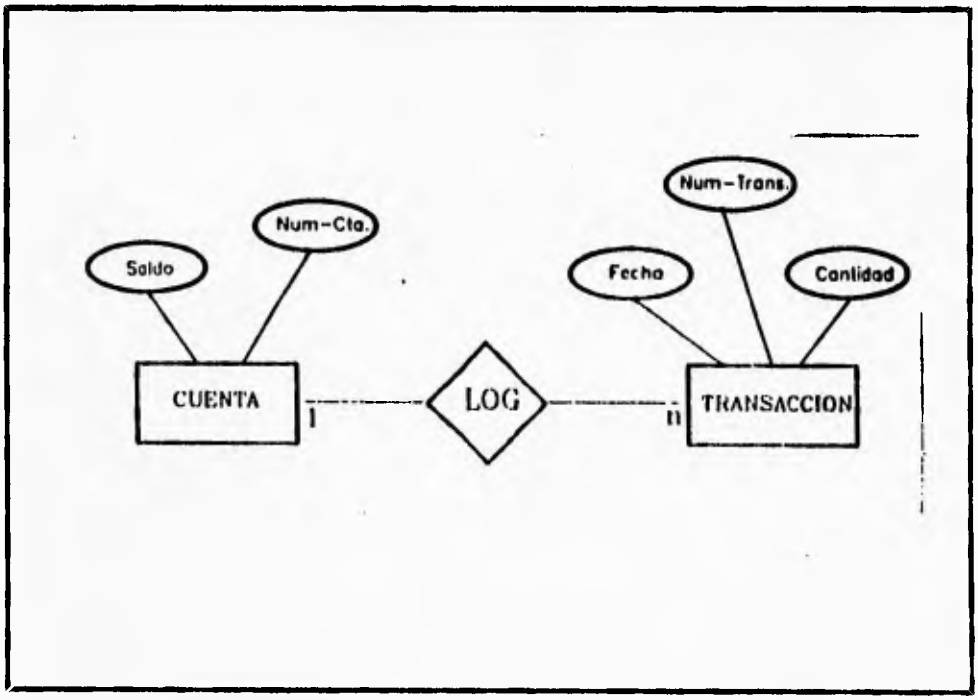


Fig. 1.4.8 Diagrama E-R de la relación Log.

En notación matemática esto puede definirse de la siguiente manera:

Sea "A" un conjunto de entidad dependiente con atributos descriptivos a1,a2, ..., ar. Sea "B" el conjunto de entidad dominante "B" sobre el cual "A" es dependiente. Si la llave primaria de "B" consiste de los atributos b1, b2,..., bs. Representamos al conjunto de entidad "A" por una tabla llamada "A" con una columna por cada atributo del conjunto:

{a1, a2, ..., ar} U {b1, b2,..., bs}

NUM-CTA	NUM-TRANS	FECHA	CANTIDAD
259	5	11 MAY 85	+50
630	11	17 MAY 85	+70
401	22	23 MAY 85	-300
700	69	28 MAY 85	-500
199	103	03 JUN 85	+900
259	8	07 JUN 85	-44
115	53	07 JUN 85	+120
199	104	13 JUN 85	-200
259	7	17 JUN 85	-70

NUM-CTA	SALDO
259	1000
630	2000
401	1500
700	1500
199	500
467	900
115	1200
183	1300
218	2000
225	2500
210	2200

Fig. 1.4.9 Relación de dependencia de Transacción respecto de Cuenta.

Para explicar lo anterior, considere el mismo diagrama Entidad-Relación de la figura correspondiente al conjunto de relación LOG. como puede apreciarse, el conjunto de entidad Transacción posee tres atributos: Num-Trans, Fecha y Cantidad. La llave primaria del conjunto de entidad Cuenta, mediante la cual Transacción es dependiente, es Num-Cta. Por lo tanto, de acuerdo con la teoría anteriormente expuesta, Transacción es representada por una tabla de cuatro columnas denominadas Num-Cta, Num-Trans, Fecha y Cantidad, la cual corresponde a la unión de sus atributos y la llave primaria del conjunto de entidad dominante "Cuenta". La figura 1.4.9 muestra la relación de dependencia de Transacción respecto de Cuenta.

LLAVES PRIMARIAS

Una tarea importante en el modelado de una base de datos es especificar como distinguir las relaciones y las entidades. Conceptualmente, las entidades individuales y las relaciones son distintas, pero desde la perspectiva de las bases de datos, la diferencia entre ellas debe expresarse en términos de sus atributos. para hacer tales distinciones, una llave se asigna para cada conjunto de entidad. Esta llave en un conjunto de uno o más atributos, que se toman colectivamente, permitiéndonos identificar únicamente a una entidad en el conjunto de entidad. Por ejemplo, el atributo Seguro-social del conjunto de entidad clientes es suficiente para distinguirlo de otra entidad Clientes. Por lo tanto, Seguro-Social es una llave. Similarmente la combinación de Nom-Clien y Seguro-Social es una llave para el conjunto de entidad Clientes. Sin embargo, el atributo Nomb-Clien no es una llave ya que varias personas pueden tener el mismo nombre. Dado que la llave puede ser una combinación de atributos, esta puede llegar a ser muy grande, por lo que se desea minimizarla. A esta llave se denomina llave candidata.

Es posible que haya varios conjuntos distintos de atributos los cuales pueden ser llaves candidatas. Usaremos el término de llave primaria para denotar a una llave candidata que haya sido seleccionada por el diseñador de la base de datos como el medio principal de identificación de entidades entre un conjunto de entidad.

Es posible que una entidad no posea los suficientes atributos para formar una llave primaria. Por lo tanto ésta debe ser dependiente de la entidad dominante o que posee la llave primaria. Esto se relaciona con la dependencia existencial que se mencionó con anterioridad, en el ejemplo de la relación LOG que se estableció entre las entidades cuenta y transacción.

El enfoque relacional de las Bases de Datos

Un sistema de Base de Datos debe ser capaz de representar y manipular entidades (registros o segmentos) y sus relaciones de manera fácil y conveniente. En el enfoque jerárquico o de árbol, se representa la relación entre dos segmentos por la posición relativa de arriba hacia abajo y de derecha a izquierda de los tipos de segmentos involucrados. En el enfoque de red, las relaciones se representan mediante mecanismos de "set" con uso de apuntadores, los cuales enlazan a un tipo de registro propietario con un tipo de registro miembro. Para el caso de una Base de Datos grande y complicada, el modelo lógico y la manera como pueden tener acceso a él los usuarios, por medio de un lenguaje de manipulación de datos (DML) pueden volverse muy complejos. Aún más, el acceso a los datos se inclina demasiado a las rutas de acceso, en términos de los enlaces o posiciones jerárquicas que estableció el diseñador. De esta manera puede ocurrir que muchos cambios a la Base de Datos violen la independencia de los datos o afecten los programas de aplicación.

El enfoque relacional a Base de Datos concebido por E.R. Cood, constituye un enfoque muy diferente para la descripción y manipulación "lógica" de los datos, Se esfuerza por evitar muchas de las desventajas que se han mencionado. En forma concisa, visualiza la Base de Datos lógica como una simple colección de tablas bidimensionales llamadas "Relaciones". Estas tablas son planas, en el sentido de que no hay grupos repetitivos. Los usuarios las comprenden y manejan fácilmente con poco o ningún entrenamiento en programación, y no implican consideración alguna sobre aspectos posicionables, de apuntadores o de rutas de acceso. Aparentemente es más fácil visualizar y manipular una tabla como la de la figura, que en su forma convencional equivalente dada en la figura 1.4.10. Así mismo, es posible transformar cualquier base de datos a una de tipo de tabla plana o relacional, mediante la introducción de redundancia adicional.

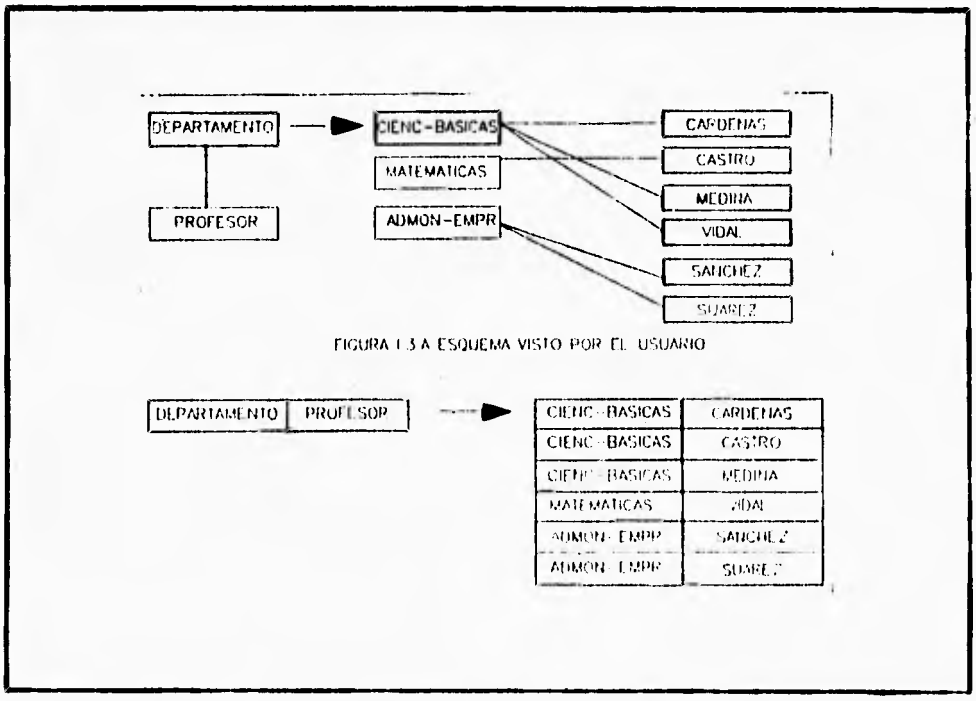


Fig. 1.4.10 Esquema visto por el usuario.

El enfoque relacional introduce terminología propia y exhibe una tendencia a usar términos poco convencionales relacionados con las matemáticas. El enfoque relacional se fundamente en la teoría matemática de las relaciones, por lo cual, posee un buen fundamento teórico.

La figura 1.4.11 muestra la arquitectura relacional que se propone. La Base de Datos Global es un conjunto de relaciones, al que se hace referencia generalmente como modelo relacional de datos, relaciones base o base de datos relacional. El modelo de datos lo define el administrador de la base de datos (ABD) mediante un lenguaje de descripción del modelo relacional de datos.

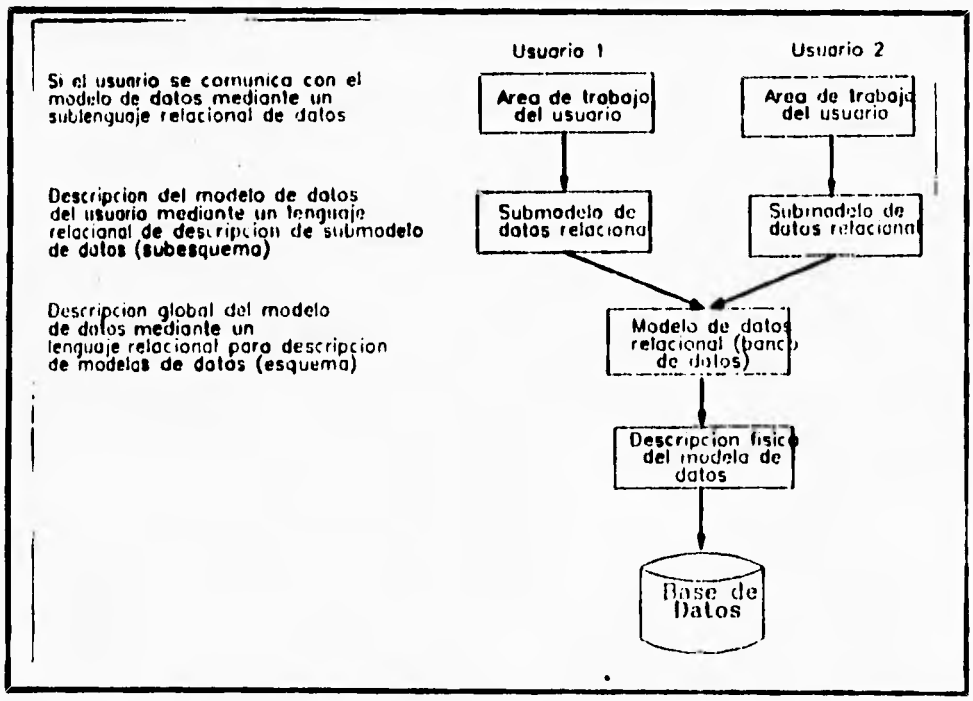


Fig. 1.4.11 Arquitectura relacional de usuarios.

Al modelo particular de datos de un usuario, que se extrae del modelo de datos global, se le llama el submodelo relacional de datos o vista. Al submodelo de datos lo define el ABD y/o lo llama el usuario vía un lenguaje de descripción del submodelo relacional de datos. El submodelo de datos es una colección de relaciones que pueden derivarse de las del modelo de datos mediante ciertas operaciones relacionales. el lenguaje de descripción del submodelo de datos incluirá las operaciones o mecanismos relacionales necesarios para formar submodelos permisibles a partir del modelo. De hecho, el lenguaje de descripciòn de submodelos es básicamente el mismo sublenguaje de datos, con ciertas adiciones para definición de datos.

Puede definirse un número arbitrario de submodelo sobre un modelo de datos dado. Un submodelo podría ser el modelo de datos completo. Un número arbitrario de usuarios puede compartir un modelo de datos dado vía los submodelos.

El Modelo Relacional

Desde una perspectiva histórica, el modelo de datos relacional es relativamente nuevo. Los primeros sistemas de base de datos fueron diseñados utilizando los modelos jerárquico y de red.

El modelo de datos relacional representa, como ya se mencionó, a la base de datos como una colección de tablas. Aunque las tablas son simples y de noción intuitiva, están en correspondencia directa entre el concepto de tabla y el concepto matemático de relación.

En años posteriores a la introducción del modelo relacional, se desarrollo una teoría para bases de datos relacionales. Esta teoria asiste en el diseño de bases de datos relacional y en el procesamiento eficiente de los requerimientos de información de los usuarios desde la base de datos.

Una base de datos relacional consiste en una colección de tablas, a cada una de las cuales se le asigna un nombre único. Cada tabla tiene una estructura similar a las representadas en el modelo Entidad-Relación. Un renglón en una tabla representa una relación entre un conjunto de valores. Dado que una tabla es una colección de tales relaciones, encierra el concepto de tabla y el concepto matemático de relación, del cual el modelo de datos relacional toma su nombre.

En términos menos formales, una relación es una tabla bidimensional de "n" columnas constituidas por un conjunto de tuplos de "n" elementos (n-tuplos). Cada una de las columnas en una relación es un conjunto de valores de elementos de datos (tipo de atributo o campo) al que se le denomina el dominio.

Una relación o tabla es un arreglo bidimensional con las siguientes características:

1. Cada entrada en la tabla es un elemento de datos o dato elemental; no hay grupos repetitivos. Es decir, cada dominio debe representar a una sola relación. se dice que una relación está normalizada si no tiene grupos repetitivos.
2. A cada columna, esto es, al dominio, se les asigna un nombre diferente y está constituido por valores del mismo dato elemental.
3. Todas las hileras o tuplos son distintas; no se permiten duplicados.
4. Las hileras y columnas pueden ordenarse en cualquier secuencia en cualquier momento, sin que esto afecte el contenido de la formación o la semántica aplicada.

Cada tuplo o relación debe poseer una llave que lo identifica univocamente y lo diferencia de otros tuplos de esa relación. La llave es un dominio simple o una combinación de dominios. Una llave constituida por una combinación de dominios es no redundante si ninguna entidad de la llave puede eliminarse o borrarse sin destruir la habilidad de identificar univocamente a cada tuplo. Puede existir más de un conjunto de dominios que pueden constituir una llave; es decir que identifica univocamente a un tuplo y que es no redundante.

A estos conjuntos se les denomina llaves candidatas. La llave primaria es el conjunto de dominios que se selecciona para identificar a los tuplos. Normalmente debería ser el que tuviera el número mínimo de dominios.

Ventajas del Modelo de Datos Relacional

La información es presentada al usuario final con un modelo de datos simple. sus requerimientos están formulados en términos del contenido de la información y no refleja ninguna complejidad en los aspectos orientados al sistema. Un modelo de datos relacional es lo que el usuario ve, pero no necesariamente lo que físicamente se implemento.

Requerimientos de no procedimiento.-Dado que no hay dependencia posicional entre las relaciones, no requiere reflejar alguna estructura preferida y por lo tanto puede ser de no procedimientos.

Independencia de Datos.- El modelo de datos relacional elimina los detalles de estructura, de almacenamiento y estrategia de acceso desde la interfaz de usuario. el modelo proporciona un grado relativamente grande de independencia de datos.

Sistemas manejadores de bases de datos basados en el modelo de datos relacional están disponibles actualmente y el adelanto de la tecnología está mejorando la rentabilidad y rapidez de estos, como respuesta a las necesidades que impone la competencia por el mercado; y la eficiencia y confiabilidad que demandan los usuarios y consumidores.

EL PROCESO DE NORMALIZACION

El enfoque relacional posee bases matemáticas rigurosas que respaldan su teoría relacional, proporcionando simplicidad en las estructuras de datos utilizadas, facilitando su uso y modificaciones. Para poder obtener estas facilidades, el proceso de normalización es la clave.

Los objetivos del proceso de normalización son:

- Eliminar en lo posible todos los datos que mantengan anomalías
- Conservar toda la información
- Maximizar la flexibilidad

1. La estructura debe ser tal que haya lugar para todos los datos requeridos.
2. La redundancia que puede existir deberá ser usada por los elementos que son identificadores o llaves. Por lo que se debe tener cuidado de elegir aquellos que no estén sujetos a actualizaciones.
3. Los efectos indeseables son las anomalías que pueden presentarse en las operaciones de actualización, inserción y eliminación.

   Anomalía de Inserción: No se debe almacenar nueva información sobre una entidad en particular hasta que se establece su relación con otra entidad.

   Anomalía de eliminación: La eliminación de un solo registro puede ocacionar la eliminación de toda una ocurrencia de una entidad.

   Anomalía de actualización: si el valor de un atributo cambia, debe cambiar en los múltiples sitios donde se encuentre definido.

4. Esta capacidad de adaptabilidad de los cambios maximizan la independencia de uso particular de los datos.

La normalización requiere tres acciones sobre un atributo de una entidad. Estas son las siguientes:

- **Primera forma normal**
- **Segunda forma normal**
- **Tercera forma normal**

DEPENDENCIA FUNCIONAL

El poder definir si una relación se encuentra en la primera, segunda o tercera forma normal, se basa en las dependencias funcionales que existan entre los atributos y los dominios particulares a esa relación; las dependencias funcionales las determina directamente el significado o la semántica del contenido de la base de datos según la interpretación del diseñador de la base de datos.

Cuando se hizo la descripción del modelo E-R, se mencionó la existencia de una entidad independiente y otra subordinada. Esta subordinación está determinada por la relación que guardan entre ellas y se ejemplificó definiendo una relación Log en la cual el conjunto de entidad Transacciones dependencia funcionalmente de cuenta, ya que si la cuenta desaparecía, las transacciones que en ella se manejaban debían desaparecer, en cambio si se eliminaba alguna transacción la cuenta no se veía afectada. el modelo relacional se basa en este concepto para establecer sus relaciones funcionales entre atributos. De esta forma, una definición formal de DEPENDENCIA FUNCIONAL en el modelo relacional sería la siguiente:

Dada la relación R se dice que el atributo B es funcionalmente dependiente del atributo A si en cualquier instante de tiempo cada valor de A no tiene más de un valor B asociado con él en la relación R. el indicar que B es funcionalmente dependiente de A es equivalente a indicar que A identifica o determina a B, lo cual se denota como A--> B esto

último concuerda con la lógica matemática en la que, A-->B significa que A identifica a B, es decir, que si A tiene un cierto valor "a", entonces B debe tener un valor "b".

PRIMERA FORMA NORMAL

Uno de los objetivos del enfoque relacional es representar las bases de datos mediante relaciones planas o tablas. Por lo tanto cada identificador en una relación debe poseer un solo valor de cada uno de los atributos y no múltiples valores de estos. si posee múltiples valores se dice que existen grupos repetitivos.

Por ejemplo consideremos la relación INF_TUR que posee la siguiente estructura y se muestra en la Tabla 1.4.1.

En esta relación se puede observar que existen multiples valores para el atributo CVE_GIRO que es el identificador, por lo tanto no es una relación plana.

Se dice que una relación está en Primera Forma Normal si para cada valor específico de un identificador existe uno y sólo un valor de cada atributo. Es decir no hay "grupos repetitivos". Pero se mantiene un alto grado de redundancia.

Para transformar la relación de la tabla 1.4.1, a la Primera Forma Normal aplicamos la conversión de la 1FN:

CONVERSION DE LA PRIMERA FORMA NORMAL (1FN) :

La redundancia puede reducirse separándola en otro grupo aparte de la entidad, debiendo contener el identificador de la entidad original para mantenerse relacionadas.

CVE_GIRO	GIRO	TIPO_GIRO	NUM_FOLIO	NOM_COM	CALLE	NUMERO	COLONIA	C.P.	TEL	R.F.C	RAZON_SOC	CATEGORIA	CON_SER
56	MOTEL	S	12H	HOTEL DORADO PACIFICO	PASEO IXTAPA	SIN	IXTAPA	[illegible]	32825	[illegible]	OPERADORA HOTELERA DORADO	5 ESTRELLAS	SERV. COMPL
56	MOTEL	S	5H	IXTAPA PALACE RESORT	PASEO DE LAS GARZAS ESQ. PASEO DEL RINCON	SIN	RESIDENCIAL III	[illegible]	31350	APT861119IR17	AMERICA PROMOTORA TURISTICA S.A. DE C.V.	5 ESTRELLAS	SERV. COMPL
56	MOTEL	S	23H	ORBI HOTEL	PASEO DEL PALOMAR	SIN	ZONA HOTELERA	[illegible]	[illegible]	[illegible]	PROPULSORA IXTAPA SUR, S.A. DE C.V.	5 ESTRELLAS	SERV. COMPL
15	RESTAURANTES	S	10R	PIZZERIA [illegible]	PASEO IXTAPA	LOC. 6	IXTAPA	[illegible]	30279	[illegible]	OBELDOX NIETO ENRIQUE	N/C	N/C
15	RESTAURANTES	S	4R	LAS MARGARITAS	PASEO IXTAPA	SIN	CENTRO	[illegible]	30070	[illegible]	ALLIA GONZALEZ FRANCISCO	N/C	N/C
15	RESTAURANTES	S	8R	DA BAFFONE	PASEO IXTAPA	SIN	IXTAPA	[illegible]	31122	[illegible]	RESTAURANTE DA-BAFONE, S.A.	N/C	N/C
73	DISCOTEQUES	S	3D	MAGIC CIRCUS	PASEO IXTAPA	SIN	IXTAPA	[illegible]	31500	[illegible]	OPERTAPA MAGICO, S.A. DE C.V.	N/C	N/C
87	TIENDA DE ROPA	S	7TR	ACA JOE	PASEO IXTAPA	SIN	IXTAPA	[illegible]	30302	[illegible]	ACA ROPA S.A. DE C.V.	N/C	N/C

Tabla 1.4.1

Por lo tanto la relación INF_TUR normalizada en la 1FN quedará de la siguiente manera :

Separando la redundancia en otro grupo aparte, obtendremos un catálogo denominado GIROS :

GIROS

CVE_GIRO	GIRO	TIPO_GIRO	CATEGORIA	PRIN_SER
56	HOTEL	S	5 ESTRELLAS	SERV. COMPL
56	HOTEL	S	5 ESTRELLAS	SERV. COMPL
56	HOTEL	S	5 ESTRELLAS	SERV. COMPL
15	RESTAURANTES	S	N/C	N/C
15	RESTAURANTES	S	N/C	N/C
15	RESTAURANTES	S	N/C	N/C
73	DISCOTEQUES	S	N/C	N/C
87	TIENDA DE ROPA	B	N/C	N/C

Tabla 1.4.2

El identificador de la relación GIROS es CVE_GIRO.

La relación de la tabla 1.4.1. quedará tal y como se muestra en la tabla 1.4.3, con la conversión de la 1FN.

SEGUNDA FORMA NORMAL

La prueba para determinar si una entidad está en Segunda Forma Normal es: que el valor de cualquier atributo que no es llave dependa de todos los atributos que forman la llave.

CVE_GIRO	GIRO_FOLIO	GIRO_COM	CALLE	[illegible]	[illegible]	C.P.	TEL	R.F.C	RAZON_SOC
66	12H	HOTEL BORREGO PACIFICO	PASEO IXTAPA	SIN	IXTAPA	[illegible]	[illegible]	[illegible]	OPERADORA HOTELERA BORREGO
66	6H	IXTAPA PALACE RESORT	PASEO DE LAS GARZAS ESQ. PASEO DEL PESCADO	SIN	[illegible]	[illegible]	[illegible]	[illegible]	AGENCIA PROMOTORA TURISTICA S.A
66	23H	[illegible] HOTEL	PASEO DEL PALMERAS	SIN	ZONA HOTELERA	[illegible]	[illegible]	[illegible]	PROPULSORA IXTAPA SUR, S.A. DE C.V.
15	16R	PIZZERIA [illegible]	PASEO IXTAPA	LOC. 6	IXTAPA	[illegible]	[illegible]	[illegible]	[illegible]
15	4R	LAS [illegible]	PASEO IXTAPA	SIN	CENTRO	[illegible]	[illegible]	[illegible]	ALUA GONZALEZ FRANCISCO
15	8R	DA-BAFFORE	PASEO IXTAPA	SIN	IXTAPA	[illegible]	31122	[illegible]	RESTAURANTE DA-BAFFORE, S.A.
73	3D	[illegible] CIRCUS	PASEO IXTAPA	SIN	IXTAPA	[illegible]	[illegible]	[illegible]	OPIXTAPA [illegible], S.A. DE C.V.
87	7TR	ACA JOE	PASEO IXTAPA	SIN	IXTAPA	[illegible]	[illegible]	[illegible]	ACA ROPA S.A. DE C.V.

Tabla 1.4.3

CONVERSION A LA SEGUNDA FORMA NORMAL (2FN)

Para normalizar una entidad en la Segunda Forma, se crea una nueva entidad de los atributos que dependen parcialmente de una llave, siendo parte del identificador de esta nueva entidad el atributo del cual depende para mantenerla relacionada con la original.

Siguiendo con el ejemplo anterior, ahora obtendremos un segundo catálogo denominado FOLIOS (Tabla 1.4.3), el cual contiene todos los atributos que dependen parcialmente de una llave.

INF_TUR

CVE_GIRO	NUM_FOLIO
56	12H
56	5H
56	23H
15	10R
15	4R
15	8R
73	3D
87	7TR

Tabla 1.4.4

En la tabla 1.4.4 se muestra como queda al final ya normaliza (con las 2 primeras normas) la ralación INF_TUR.

TERCERA FORMA NORMAL

Se dice que una entidad se encuentra en Tercera Forma Normal si el valor de cada atributo depende de toda la llave y no de cualquier otro que no lo sea.

En esta forma normal, se buscan los atributos que están dependiendo de otro que no es una llave.

CONVERSION A LA TERCERA FORMA NORMAL (3FN).

Para poner una entidad en Tercera Forma Normal, se crea una entidad con los atributos que no dependen de ningún atributo que forma la llave, siendo el identificador de la nueva entidad el atributo del cual era dependiente.

En el ejemplo, se observa que no existe este tipo de dependencia, por lo que se dice que cumple también con la tercera forma normal.

CAPITULO II

PLANTEAMIENTO DE LA PROBLEMATICA Y PROPUESTA DE SOLUCIÓN

2.1 Antecedentes

Dada la necesidad de integrar al mundo de la computación todos los campos de acción en los que interviene el ser humano, surge la necesidad de que al hacerlo, ésto no le cause ninguna molestia, sino que al contrario, que esta integración sea accesible a cualquier individuo, con mínimos conocimientos en el uso de computadoras. Es decir; que cualquier persona pueda hacer uso de ellas, sin dificultad alguna.

Pensando en hacerle la vida más fácil al usuario surgieron las interfaces gráficas, incorporadas al desarrollo de sistemas de software, con el objeto de que el usuario cuente en la misma pantalla con toda la información que requiere para hacer uso del sistema.

Es por ello que estan surgiendo al mercado los lenguajes de programación visual, los cuales cuentan entre sus características el de poder crear este tipo de interfaces, basandose en los conceptos de la Programación Orientada a Objetos, los cuales se integrarán al desarrollo del presente sistema, **"Prestadores de Bienes y Servicios para el puerto de Ixtapa Zihuatanejo".**

Este sistema se enfoca a la problematica a la que se enfrenta un turista al llegar a un sitio desconocido, en donde el tipo de información que requiere principalmente, es geográfica, económica, cultural y recreativa del lugar.

Por ejemplo, un turista que llega al puerto, puede hacer uso de este sistema si requiere información sobre las categorías de los hoteles que el puerto ofrece, el cual además le ofrecerá tanto la información económica, como sería el precio de las habitaciones así como la ubicación geográfica de los hoteles.

Para lograr lo anterior el sistema se presentará al turista mediante una interfaz gráfica en cuya pantalla se encontrarán seis menus disponibles sobre una barra luminosa, siendo estos:

Información
Ubicación
Bienes/Servicios
Radios
Ayuda
Quejas

Cada uno de estos menus presentan diferentes opciones de las que el turista puede hacer uso.

Para poder accesar a cualquiera de los menus se valdra de dos dispositivos de entrada el teclado y el mouse, siendo éste último, el de más útilidad ya que sólo se requiere apuntar hacia el menu deseado para hacer uso de él.

Entre las funciones que realizan los menus se desriben las siguientes:

INFORMACION proporciona información cultural o historica de un sitio en particular.

UBICACION permite escoger entre las dos ciudades del puerto para desplegar el mapa respectivo de cada ciudad.

RADIOS permite escoger entre los tres diferentes radios de acción para hacer un acercamiento de un punto geografico ubicado sobre el mapa de la ciudad seleccionada.

Para lograr lo anterior el sistema se presentará al turista mediante una interfaz gráfica en cuya pantalla se encontrarán seis menus disponibles sobre una barra luminosa, siendo estos:

Información
Ubicación
Bienes/Servicios
Radios
Ayuda
Quejas

Cada uno de estos menus presentan diferentes opciones de las que el turista puede hacer uso.

Para poder accesar a cualquiera de los menus se valdra de dos dispositivos de entrada el teclado y el mouse, siendo éste último, el de más útilidad ya que sólo se requiere apuntar hacia el menu deseado para hacer uso de él.

Entre las funciones que realizan los menus se desriben las siguientes:

INFORMACION proporciona información cultural o historica de un sitio en particular.

UBICACION permite escoger entre las dos ciudades del puerto para desplegar el mapa respectivo de cada ciudad.

RADIOS permite escoger entre los tres diferentes radios de acción para hacer un acercamiento de un punto geografico ubicado sobre el mapa de la ciudad seleccionada.

BIENES/SERVICIOS despliega una lista con los diferentes bienes y sevicios con los que cuenta el puerto, indicando sobre el mapa, el sitio geografico donde se ubica dicho bien o servicio.

AYUDA proporciona información sobre como utilizar el sistema, si es que el usuario tiene alguna duda al respecto del funcionamiento del sistema.

QUEJAS es una opción que se le proporciona al usuario en donde puede expresar o hacer sugerencias sobre lo que le gusta o le disgusta del sistema.

Dicho sistema esta elaborado en el lenguaje de programación **VISUAL C+ +** , ya que este leguaje proporciona, ademas de los elementos necesarios para realizar una interfaz gráfica de calidad, un código de programación formal que requiere la verdadera ingeniería de software.

2.2 ESTRATEGIA DE SOLUCIÓN

El principal objetivo del sistema es proporcionar información turística del puerto de Ixtapa-Zihuatanejo localizada en un mapa geográfico, y para ello primeramente se tiene que recabar toda la información correspondiente del lugar.

Para lograr lo anterior se buscaran fuentes de información lo más amplias que sea posible, como lo es el directorio telefonico de la ciudad, después se elaboraran diversos cuestionarios con preguntas que vayan de acuerdo con los elementos que se reunan del directorio telefonico, también se consultarán revistas y folletos con publicidad turística del lugar para terminar de recabar la información, pensando conseguir éstas en las oficinas de la secretaria de turismo y en algunas agencias de viajes.

Una vez realizado lo anterior se iniciara el diseño de las bases de datos en las cuales se vaciará toda la información recabada en los cuestionarios correspondientes.

Después se iniciará el analisis y diseño del sistema que a grandes rasgos consistirá en: Conseguir dos mapas respectivos de las ciudades para que entren en contacto directo con los diferentes bancos de datos que el sistema maneja, tales como los bancos de información económica, cultural y turística del puerto.

Los mapas se tomarán como una colección de puntos que se colocorán dentro de una estructuración determinada.

2.2 ESTRATEGIA DE SOLUCIÓN

El principal objetivo del sistema es proporcionar información turística del puerto de Ixtapa-Zihuatanejo localizada en un mapa geográfico, y para ello primeramente se tiene que recabar toda la información correspondiente del lugar.

Para lograr lo anterior se buscaran fuentes de información lo más amplias que sea posible, como lo es el directorio telefonico de la ciudad, después se elaboraran diversos cuestionarios con preguntas que vayan de acuerdo con los elementos que se reunan del directorio telefonico, también se consultarán revistas y folletos con publicidad turística del lugar para terminar de recabar la información, pensando conseguir éstas en las oficinas de la secretaría de turismo y en algunas agencias de viajes.

Una vez realizado lo anterior se iniciara el diseño de las bases de datos en las cuales se vaciará toda la información recabada en los cuestionarios correspondientes.

Después se iniciará el analisis y diseño del sistema que a grandes rasgos consistirá en: Conseguir dos mapas respectivos de las ciudades para que entren en contacto directo con los diferentes bancos de datos que el sistema maneja, tales como los bancos de información económica, cultural y turistica del puerto.

Los mapas se tomarán como una colección de puntos que se colocorán dentro de una estructuración determinada.

Para tal estructura se retomará el concepto de matriz con renglones y columnas en donde para cada pareja de puntos (X,Y), será una coordenada geográfica.

Las coordenadas geográficas estarán relacionadas con los diferentes bancos de datos y éstos a su vez lo estarán con un punto en particular del mapa.

Los bancos de datos también estarán relacionados entre si. Por ejemplo, si se señala un punto en el mapa, éste desplegará la información sobre el nombre del lugar seleccionado, el que a su vez estará en relación con la información económica, cultural y turística correspondiente, también se dará el caso contrario si se elige el nombre de un bien o servicio, el sistema se encargrá de indicar la ubicación geográfica del lugar, mediante un punto luminoso sobre el mapa de la ciudad respectiva.

Como se puede observar se utilizará el concepto de función Uno a Uno, en donde aun punto geográfico le corresponde un dato del banco de información y viceversa.

El sistema manejará tres radios de acción de diferente longitud 250, 500 y 1000 Kms para realizar los acercamientos sobre los mapas de las ciudades. Para lograr tal fecto se tomará el concepto de escalas, en donde a cada punto del mapa se le aplicará una "regla de tres" para hacer los aumentos y las disminuciones de los puntos geográficos.

Considerando que los mapas de las ciudades son mapas urbanos, es decir, son mapas que tienen definido el contorno de las calles y de las avenidas de la ciudad, a los cuales se le manejrá como objetos que es un de puntos importantes de la programación orientada a objetos.

Después se realizaran los diversos módulos de programación los cuales estaran relacionados mediante diversas pantallas de consulta de información

Y por último se realizrá un instructivo explicativo del uso del presente sistema.

2.2.1 PLAN DE TRABAJO

Considerando que el objetivo del sistema es proporcionar información turística del puerto de Iztapa-Zihuatanejo. Para elaborar el plan de trabajo primeramente se penso, en los aspectos más importantes que afectan a un turista, que son principalmente el conocimiento de los bienes y servicios que presta el lugar.

Para obtener información sobre ellos, se recurrió al directorio telefonico y a la sección amarilla del puerto, de ahí seleccionamos por orden alfabetico, los bienes y servicios que a nuestro juicio consideramos más importantes:

Abarrotes y miscelaneas
Aduanas
Agencias de Viaje
Angeles verdes
Arrendadora de vehiculos
Arrendadora de vehículos acuaticos
Artesanías
Artículos deportivos
Artículos Fotograficos y revelado
Atractivos turísticos varios
Bancos
Cajeros Automáticos
Capitania de puerto
Casas de bolsa
Centros comerciales
Centros nocturnos con variedad

Cerrajerias

Club de Golf

Club de tenis

Consulados

Cruz roja

Deportes acuaticos

Discoteques

Eléctrico Automotriz

Escuelas

Farmacias, perfumerias, boticas, droguerias

Gasolinerias

Gimnasios

Hospitales

Hoteles

Laboratorios Clinicos

Lineas de autobuses

Marinas

Mecanico Automotriz

Medicos

Mercados

Oficinas de lineas aéreas

Oficinas de migración

Playas

Playa Varadero

Playa Quieta

Playa Don Rodrigo

Playa Las Cuatas

Playa el Palmar

Playa Vista hermosa

Isla grande o de Ixtapa

Playa Cuachalalate

Playa del Coral

Playa Larga

Playa Linda

Policia y tránsito

Procuraduria de justicia

Procuraduria del consumidor

Restaurantes

Servicios postales

Servicios telefonicos

Servicios telegraficos

Tabaquerias

Tiendas, Autoservicio y Deptales

Tiendas de ropa casual

Tiendas de ropa formal

Zapaterias

2.2.2 RECOPILACIÓN DE LA INFORMACIÓN

Una vez que se tuvieron los elementos sobre los cuales buscar información, se elaborarón diversos cuestionarios, que contemplarán en forma general preguntas tanto económicas, culturales, recreativas y gráficas.

La información especifica sobre cada elemento se consiguió, consultando diversas, revista y folletos con publicidad turística, que nos fueron proporcionados por SECTUR (Secretaría de Turismo). Para la obtención de los mapas de las ciudades se consiguieron de la revista "Conozca México" en la cual por cada estado de la Republica se expide una colección de mapas de las ciudades que comprende, indicando las principales vias de comuncación de ellas.

Para buscar la información de los elementos en cuestión se formularon diversas preguntas dispuestas en cuatro cuestionarios que variaron según el tipo de bien o servicio que trataran, por ejemplo para recabar la información sobre los lugares de recreación, restaurantes, tiendas, centros comerciales, hoteles y otros servicios, se utilizaron los siguientes formatos:

Cuestionario de atractivos turísticos

Sitio de Atracción Turística : _____

Ubicación : _____

Descripción de lugar : _____

Información cultural : _____

_____

_____

Cuestionario de Tiendas y Departamentales

Nombre de la Tienda o Centro comercial: _____

Dirección : _____

Categoría : _____

Telefonos: _____

Atención : _____

Descripción de la tienda y de lo que vende (Incluyendo marcas):

_____

_____

_____

_____

Cuestionario de Restaurantes

Nombre del restaurante : _____

Dirección : _____

Telefonos: _____

Categoría : _____

Atención : _____

Horarios de servicio : _____

Tipo de comida : _____

_____

Cuestionario de Hoteles

Nombre del Hotel : _____

Dirección : _____

Categoría : _____

R.F.C. : _____

Propietario : _____

Servicios que presta : _____

_____

Telefonos : _____

Horarios : _____

Tarifas por habitación o por persona : _____

_____

Cuestionario otros servicios

Nombre del bien o servicio : _____

Dirección : _____

Telefónos : _____

Horarios de servicio : _____

Atención: _____

Descripción de los servicios que presta : _____

_____

_____

Tarifas/Servicio : _____

2.2.3 CLASIFICACIÓN DE LA INFORMACIÓN

Una vez reunida la mayor cantidad de información económica, cultural y recreativa se opto por clasificarla en tres listas:

Lista de Bienes

Abarrotes y miscelaneas

Artesanías

Artículos Fotograficos y revelado

Artículos deportivos

Centros comerciales

Farmacias, perfumerias, boticas, droguerias

Tabaquerias

Tiendas, Autoservicio y Deptales

Tiendas de ropa casual

Tiendas de ropa formal

Zapaterias

Lista de Servicios

Aduanas

Agencias de Viaje

Angeles verdes

Arrendadora de vehiculos

Arrendadora de vehiculos acuaticos

Atractivos turisticos varios

Bancos

Cajeros Automáticos

Capitania de puerto

Casas de bolsa

Centros nocturnos con variedad

Cerrajerias

Club de Golf

Club de tenis

Consulados

Cruz roja

Deportes acuaticos

Discoteques

Eléctrico Automotriz

Escuelas

Gasolinerias

Gimnasios

Hospitales

Hoteles

Laboratorios Clinicos

Lineas de autobuses

Marinas

Mecanico Automotriz

Medicos

Mercados

Oficinas de lineas aéreas

Oficinas de migración

Policia y tránsito

Procuraduria de justicia

Procuraduria del consumidor

Restaurantes

Servicios postales

Servicios telefonicos

Servicios telegraficos

Lista de Atractivos Turísticos

Playa Varadero

Playa Quieta

Playa Don Rodrigo

Playa Las Cuatas

Playa el Palmar

Playa Vista hermosa

Isla grande o de Ixtapa

Playa Cuachalalate

Playa del Coral

Playa Larga

Playa Linda

Ya que esto ayudará a la realización de las bases de datos del sistema, dado que para cada elemento de la lista se elaborará su respectiva base y la información recabada será almacenada en ellas.

2.3 REQUERIMIENTOS DEL USUARIO.

Un *requerimiento* es una característica que debe incluirse en un nuevo sistema. Esta puede ser la inclusión de determinada forma para capturar o procesar datos, producir información, controlar una actividad de la empresa o brindar soporte a la gerencia. Es así como la determinación de requerimientos vincula el estudio de un sistema existente con la recopilación de detalles relacionados con él.

Dado que los analistas de sistemas no trabajan como gerentes o empleados en los departamentos de usuarios (como mercadotécnia, compras, producción o contabilidad), no tienen los mismos conocimientos, hechos y detalles que los usuarios y gerentes de esas áreas. Por consiguiente, el primer paso del analista es comprender la situación. Ciertos tipos de requerimientos son tan fundamentales que son comunes en casi todas las situaciones. Dar respuesta a un grupo específico de preguntas, será de gran ayuda para comprender los requerimientos básicos. También existe otra clase de requerimientos que depende de si el sistema está orientado hacia transacciones, toma de decisiones o se extiende por varios departamentos. Por ejemplo, la necesidad de informar al gerente de inventarios de un pedido insólitamente grande que está por llegar subraya la importancia de eslabonar los departamentos de ventas, compras y almacén.

Esta etapa es la base en el planteamiento del diseño del sistema ya que la aceptación del sistema a desarrollar depende en un porcentaje muy valioso de una buena identificación de los requerimientos del usuario.

En la antigüedad el uso de los sistemas era muy complejo ya que requería que uno fuera un experto en el área para poder comprender los pasos a seguir para la ejecución de ciertos programas, esta ejecución exigía aprender varias instrucciones muy complejas las cuales no permitían entender exactamente lo que se hacia. Con el tiempo los desarrolladores de sistemas han ido cambiando la filosofía de uso de las aplicaciones creando sistemas de pantallas y menús más sencillos sin tener la necesidad de memorizar instrucciones nada familiares.

Con la introducción de computadoras personales el desarrollo de sistemas tomo un giro importante ya que ahora el usuario no necesita ser un experto en sistemas sino todo lo contrario, el usuario es una persona común que no tiene ni la menor idea de la que es un sistema de computo, que solo se limita a hacer uso de la computación para resolver un problema específico y se podría pensar que no está muy interesado en conocer lo que sucede a nivel software.

La competencia en el desarrollo de sistemas de software se fué haciendo cada vez mayor, esto significa, tener que captar mayor número de usuarios. Esta competencia fué benéfica, ya que exigía mayor facilidad de uso y sencillez para el usuario tratando de obtener mayor número de clientes.

Todos estos cambios dentro del diseño de los sistemas han propiciado una mejor interactividad entre el usuario y los sistemas de cómputo.

A continuación mencionaremos y describiremos algunos de los requerimientos del usuario para el desarrollo del sistema que nos hemos propuesto.

Uno de los primeros puntos a tratar sería el del desarrollo de un sistema amigable, significando que no deberá causar problemas el uso del sistema para el usuario, para tomar este punto podríamos decir que el sistema deberá estar planteado para usuarios con poco conocimiento en computación a tal vez nulo, al tener esta aseveración en la línea como premisa del sistema, podremos garantizar que el uso del sistema no causará ningún problema para el usuario y tendremos la seguridad de que será una herramiente útil para el mismo.

En la actualidad la facilidad que ofrece el uso del ventaneo en los menús hace más agradable el manejo de los sistemas, uno de los primeros pioneros en la aplicación del ventaneo fue la compañía Microsoft Corporation, este sistema de ventaneo ha sido aplicado por la mayoría de los desarrolladores de software. Los menús se deberán desarrollar en forma de bloques para tener orden en la selección de las opciones que nos proporciona el sistema y por ende orden en la operación.

Los colores también forman parte importante ya que de hacer una buena selección, el usuario no presentará síntomas de cansancio por el uso del mismo.

Las pantallas deberán ser legibles o sea que los conceptos deberán ser claros y entendibles por el usuario.

La ayuda en el sistema en forma inmediata ofrecerá una gran seguridad en el usuario, sin tener que distraerse en busca de manuales muy voluminosos y complejos.

El sistema debe tener un acceso sencillo sin tener que hacer muchas preguntas al usuario, salvo las necesarias para poder operarlo en una forma clara y concisa, ya que algunos programas tienen la particularidad de poner menús muy complejos y aunque dispongamos de ayuda de pantalla no debe de utilizarse para cada paso a ejecutar.

El sistema deberá tener capacidad de mantenimiento sencillo para futuras actualizaciones, ya que la información que se va a utilizar varía continuamente. Esto proporcionará al usuario información actualizada y veraz.

Es útil ver la determinación de requerimientos a través de tres grandes actividades: anticipación, investigación y especificación de requerimientos.

- ***Anticipación de requerimientos.*** Prever las características del sistema con base en la experiencia previa. Esto puede llevar al analista a investigar áreas y aspectos que de otra forma no serían tomados en cuenta.

- ***Investigación de requerimientos.*** Estudio y documentación del sistema actual utilizando para ello técnicas para hallar hechos, análisis de flujo de datos y análisis de decisión.

- ***Especificación de requerimientos.*** Análisis de los datos que describen el sistema para determinar qué tan bueno es su desempeño, qué requerimientosse deben satisfacer y las estrategias para alcanzarlos.

En nuestro caso, la información que requiere el turista, la mayoría de las veces es repetitiva, como es el caso de información de hoteles disponibles, lugares de interés, etc. y en otras constante, como lo es la historia del lugar, por lo tanto, un sistema de información computacional será de gran utilidad para la rápida y eficiente obtención de la información. Como el sistema será usado tanto en agencias turísticas como en kioscos informativos en zonas de concentración turística, el sistema debe de contar tanto con información básica para rápida consulta como con información específica del lugar.

En este caso se debe tomar especial atención en el sentido de que debe de ser un sistema muy amigable, ya que, la mayoría de los usuarios serán personas que nunca ó muy pocas veces traten con una computadora. Así que las instrucciones y comandos deben ser sencillos de usar.

Un ambiente gráfico para un sistema turístico es de gran ayuda porque facilita la ubicación y la visualización de los sitios de interés.

Los requerimientos específicos del usuario para el desarrollo del sistema son:

- Uso de un menú gráfico para el acceso a otros menús, utilizando para ello dos dispositivos, el mouse, y el teclado.
- Manejo de los mapas de las ciudades para la pronta localización de bienes y servicios, y lugares de recreación.
- Localización e información específica sobre un punto geográfico de interés deseado, utilizando tres radios de acción de diferente longitud.
- Información sobre bienes y servicios.
- Obtener información física del lugar e información económica (precios de las habitaciones de los hoteles, categoría de ellos, etc.).
- Obtención de un formato de quejas donde el interesado pueda indicar algún bien o servicio que no haya encontrado, o alguna característica que le gustaría que el sistema tuviera.
- Explicación breve del manejo del sistema.
- Obtener un sistema amigable y confiable.
- Sistema de tamaño pequeño (10 MB max.).
- El sistema funcione correctamente en cualquier PC (386 en adelante).

En conclusión los puntos antes mencionados nos dan un panorama de las necesidades que tiene el usuario para con el sistema (figura 2.3.1).

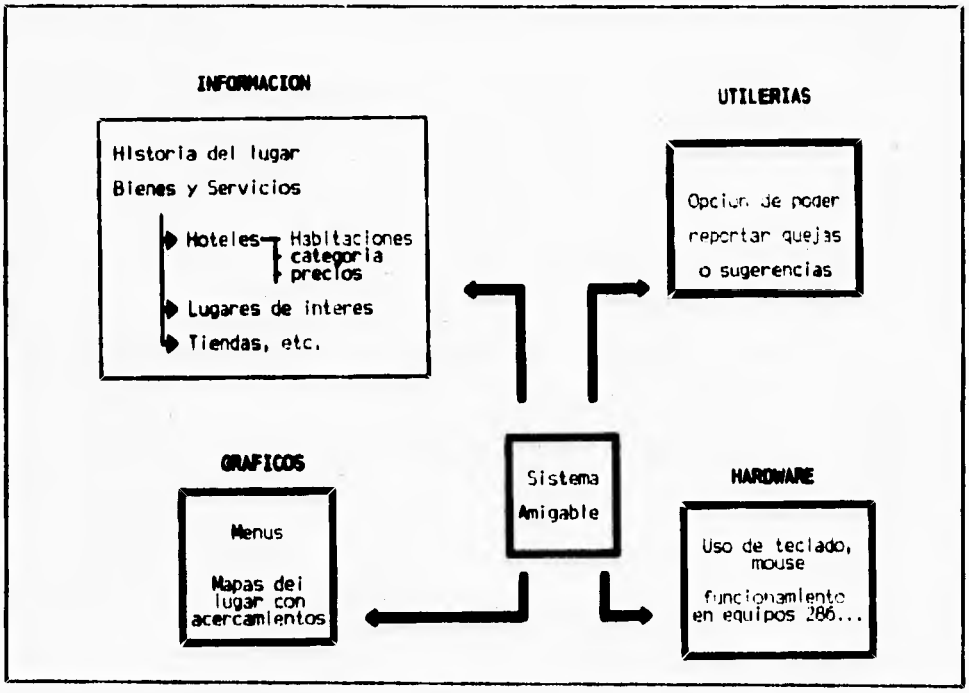


Fig. 2.3.1 Requerimientos del usuario.

No debemos perder de vista que el éxito del uso del sistema depende en gran parte de la interactividad que tenga con el usuario.

Una de las características más importantes para los usuarios de un sistema de información es la salida que éste produce. Si la salida no tiene calidad, entonces todo el sistema puede parecer a los usuarios tan poco necesario que evitarán usarlo, y esto posiblemente se convierta en causa del fracaso.

2.4 ANALISIS.

2.4.1. ESPECIFICACIONES DEL ANALISIS.

El objetivo primario del análisis de datos es el de proporcionar las bases para el diseño de una base de datos, y un enfoque disciplinado al catalogar los datos existentes en términos de las entidades y las relaciones que presentan. Sin tal entendimiento de la parte de la organización que está siendo analizada, es más difícil establecer si la base de datos será instalada eficientemente. El análisis de datos proporciona un medio muy efectivo para comunicarse con usuarios que no son profesionales en el mundo computacional, ya que se dedica solamente a aquellos en que el usuario está familiarizado.

La fase de análisis es referida algunas veces como la formulación y análisis de requerimientos, lo cual involucra el establecimiento de los objetivos y la documentación de estos requerimientos.

El análisis de datos debe ser realizado por un equipo que contenga a los usuarios, el departamento de desarrollo de sistemas, y el grupo de administración de datos.

El equipo de análisis de datos puede no intervenir en la fase de análisis de requerimientos, si esta fase está limitada a entrevistas personales con diferentes niveles de gerencia y empleados claves en el procesamiento de bienes, servicios y organización de datos. El resultado de tales entrevistas deben ser los diagramas de proceso. Los objetivos específicos y los requerimientos de la base de datos deben obtenerse de un nivel más alto de la organización.

El equipo de análisis de datos debe identificar las entidades que son necesarias para resolver el problema definido por el usuario. Durante las etapas iniciales del análisis de datos es posible que no se conozcan todos los atributos de todas las entidades. Sin embargo, a medida que estos se determinen, el equipo debe documentar la definición del atributo y su papel en un diccionario de datos apropiado.

Elaboración del Modelo de Entidad

Durante la fase de análisis se determinan las entidades mayores y sus relaciones. Estas entidades y sus relaciones se representan en modelos llamados Modelos de Entidad. El modelo es un diagrama representativo de la relación entre las clases de entidades.

La representación nos permite incluir solo aquellas entidades que se requieren para resolver un problema particular del procesamiento de datos. El modelo de entidad es esencialmente una vista del mundo real de los datos de la organización en términos de entidades atributos y relaciones.

Durante la fase de modelaje se definen las clases y relaciones de entidad más significativas, sin embargo el modelo deberá ser revisado, modificado o extendido como resultado del conocimiento sobre las nuevas entidades que se descubran. El modelo se usa para:

- Reducir la redundancia en las relaciones.
- Determinar cuales entidades son significativas al modelo y a los requerimientos del usuario
- Resolver las relaciones no binarias entre entidades.

Etapas en la integración de los Modelos de Entidad.

Las etapas requeridas para integrar los modelos de entidad son las siguientes:

- Identificar cada sinónimo u homónimo en los diferentes modelos. Esta tarea es más fácil si se usa un diccionario de datos. Los componentes con homónimos deben ser renombrados. Los componentes con sinónimos deben usar el mismo nombre.

- Los modelos de entidad para dos áreas de datos se integran superponiendo los tipos de entidad que sean idénticos o similares en los diferentes modelos de entidad. esto puede incrementar el número total de atributos del tipo de entidad, ya que las entidades idénticas pueden usar diferentes atributos.

- Como resultado de la integración, el modelo compuesto de entidad puede contener relaciones redundantes. Esta redundancia puede ser eliminada, sin embargo, determinar las relaciones que son directamente significativas y cuales son redundantes puede presentar dificultades que pueden ser resueltas solamente a través de un buen entendimiento del ambiente.

Derivación de los modelos de Entidad de Modelos tradicionales.

En realidad no existen reglas para esta derivación. La distribución de datos, con los cuales los archivos lógicos fueron construidos puede no ser la misma que se requiere para

las clases de entidad respectivas. Sin embargo, las siguientes reglas pueden seguirse cuando se trate de convertir archivos planos al modelo de entidad de la aplicación.

-listar todos los tipos de archivos en las programas relevantes.

-Listar todos los registros fisicos en los archivos.

-Listar todos los datos en los registros.

-Eliminar las redundancias e inconsistencias en los datos y los registros lógicos.

-Listar todas las combinaciones posibles de clases de entidad de los registros lógicos. El nombre del registro es un indicador de la clase de entidad.

-Listar todos los códigos en los registros que pueden dar las relaciones del modelo de entidad.

-Hacer un análisis preliminar de los datos .

-Acomodar los atributos con sus respectivas clases de entidad.

Este procedimiento debe dar como resultado un modelo de entidad que servirá como estructura para hacer otras revisiones, las cuales serán necesarias para un análisis más detallado de los datos.

Combinación de los de los Modelos de Entidad

Al convertir bases de datos existentes en su equivalente de modelo entidad, el diseñador puede llegar a diferentes modelos dependiendo de los programas o aplicaciones de los cuales los modelos fueron derivados. Se debe intentar remover las redundancias e inconsistencias al combinar los modelos de los varios programas para quedarse con un modelo integrado. Esta combinación permitirá al diseñador determinar lo siguiente:

- ¿Cuáles son las clases de entidad y los atributos comunes?
- Las inconsistencias en los nombres y uso de los atributos. Estas inconsistencias existen cuando se ve que dos entidades con diferentes nombres son la misma entidad.
- La eficacia del modelo en términos de satisfacer las necesidades del usuario.
- Si algunos atributos considerados en una entidad son realmente miembros de otra clase de entidad o de nuevas clases de entidad.
- La existencia de inconsistencias en las relaciones.

Este modelo combinado puede ser usado como la estructura para revisiones posteriores para llegar a un modelo de entidad integrado que sirva a un área de datos mayor, en lugar de varios modelos pequeños orientados a unas cuantas aplicaciones.

Agrupamiento de Clases de Entidades.

El agrupamiento de clases de entidades puede hacerse en la etapa de diseño fisico o lógico. En la etapa del diseño fisico esto puede hacerse basándose en consideraciones de desempeño. Las clases de entidades pueden juntarse o separarse en diferentes bases fisicas dependiendo de los requerimientos de acceso.

El agrupamiento lógico de clases de entidades depende de la naturaleza de los datos y sus estructuras. Es necesario, pero no suficiente, decir que los atributos se agrupan en una clase de entidad porque estos identifican y describen la clase de entidad se hace enteramente para satisfacer los siguientes requisitos:

- El área que sirven los datos o de la cual se originan.
- La estructura de datos inherente.
- La vista del usuario.
- Los usos de los datos.
- Las consultas que se hacen de los datos.
- Las necesidades de procesamiento del usuario.

Diseño del Esquema logico y vistas de aplicación.

Las vistas de la aplicación pueden definirse como el conjunto de datos que son requeridos por una aplicación particular para satisfacer una necesidad especifica de procesamiento de datos. Tenemos vistas de aplicación de:

- Una clase de entidad.
- Agrupamientos de clases de entidades.
- Agrupamientos de clases de entidad y bases de datos fisicas.
- Agrupamiento de bases de datos fisicas.

El esquema lógico puede ser definido como el mapeo del modelo de entidad en la construcción proporcionada por el manejador de la base de datos. En general el esquema lógico indica como se almacenará y accesará el modelo. En el diseño del esquema lógico tal vez sea necesario hacer algunos cambios al modelo para adecuarse al DBMS. El modelo de entidad tiene las siguientes caracteristicas:

- Es una presentación de la vista de datos del mundo real.
- Proporciona las bases para continuar con el análisis y diseño de la base de datos.
- No está restringido a ningún sistema de manejador de bases de datos (DBMS).

Diseño del Esquema logico y vistas de aplicación.

Las vistas de la aplicación pueden definirse como el conjunto de datos que son requeridos por una aplicación particular para satisfacer una necesidad específica de procesamiento de datos. Tenemos vistas de aplicación de:

- Una clase de entidad.
- Agrupamientos de clases de entidades.
- Agrupamientos de clases de entidad y bases de datos fisicas.
- Agrupamiento de bases de datos fisicas.

El esquema lógico puede ser definido como el mapeo del modelo de entidad en la construcción proporcionada por el manejador de la base de datos. En general el esquema lógico indica como se almacenará y accesará el modelo. En el diseño del esquema lógico tal vez sea necesario hacer algunos cambios al modelo para adecuarse al DBMS. El modelo de entidad tiene las siguientes características:

- Es una presentación de la vista de datos del mundo real.
- Proporciona las bases para continuar con el análisis y diseño de la base de datos.
- No está restringido a ningún sistema de manejador de bases de datos (DBMS).

- No es implementable directamente.

- Una estructura estable de referencia a la cual se puede agregar nuevas entidades, tributos y relaciones si la organización así lo requiere.

Transformación del Esquema logico a una Base de Datos Fisica.

Los detalles de esta fase dependen de las características del manejador escogido para el diseño. Esta transformación requiere las siguientes selecciones:

- **Bases de datos fisicas y tipos de relaciones lógicas, ya sea unidireccionales o bidireccionales, relacionadas fisicamente.**

- **Métodos de acceso, HISAM, HIDAM o HDAM.**

- **Segmentos, estructuras jerárquicas y representaciones de datos, incluyendo tipo y tamaño.**

- **Indices secundarios.**

- **Tipos de apuntadores en la relación.**

Adicionalmente de las selecciones anteriores, la implementación también incluye:

- Asignación de dispositivos de almacenamiento.

- Carga y organización de la base de datos.

El esquema lógico debe hacerse de manera que lo único que se deje a los diseñadores de la base física sea la selección de los métodos de acceso y los índices secundarios. Hubbard indica que deben seguirse las siguientes reglas durante el diseño físico:

- Cada clase de entidad debe ser tratada como una base de datos física.
- Si dos clases de entidades comparten una relación entre un atributo y la llave primaria por lo menos, entonces las estructuras deben consistir en dos bases de datos físicas con conexión virtual o física entre ellas.
- Las relaciones padre - hijo deben ser definidas en una sola base de datos física.
- Los segmentos que se usen frecuentemente deben ser mantenidos lo más cercanamente posible a su raíz.
- Reducir el tiempo de búsqueda de grupos de datos grandes usando índices secundarios.
- Los segmentos de tamaños variables no deben ser colocados en el mismo grupo de datos si se hacen inserciones y borrados frecuentes.

2.4.2 DIAGRAMA DE DESCOMPOSICIÓN FUNCIONAL.

La estrategia de flujo de datos muestra el empleo de éstos en forma de gráfica. Las herramientas utilizadas al seguir esta estrategia muestran todoas las caracteristicas escenciales del sistema y la forma en que se ajustan entre si. Puede ser dificil comprender en su totalidad un proceso de la empresa si se emplea para ello sólo una descripción verbal; las herramientas para el flujo de datos ayudan a ilustrar los componentes esenciales de un sistema junto con sus interacciones.

El diagrama de descomposición funcional muestra las partes fundamentales de la estructura del sistema. En el se muestran las diferentes entidades y como podemos llegar a ellas de acuerdo al flujo de información ya sea manual o automatizada, incluyendo procesos y retroceso de información.

Los usuarios y otras de la empresa que forman parte del proceso bajo estudio comprenden con facilidad anotaciones sencillas. Por consiguiente, los analistas pueden trabajar con los usuarios y lograr que participen en el estudio de los diagramas del flujo de datos. Los usuarios pueden hacer sugerencias para modificar los diagrmas con la finalidad de describir la actividad con mayor exactitud. Asi mismo pueden examinar las gráfica y reconocer con rapidez problemas; esto permite efectuar las correcciones necesarias antes de que comiencen otras tareas relacionadas con el diseño.

En la figura 2.4.2 se muestra el diagrama de descomposición de nuestro sistema. en él se pueden apreciar los difrentes modulos de funcionamiento y su flujo de datos.

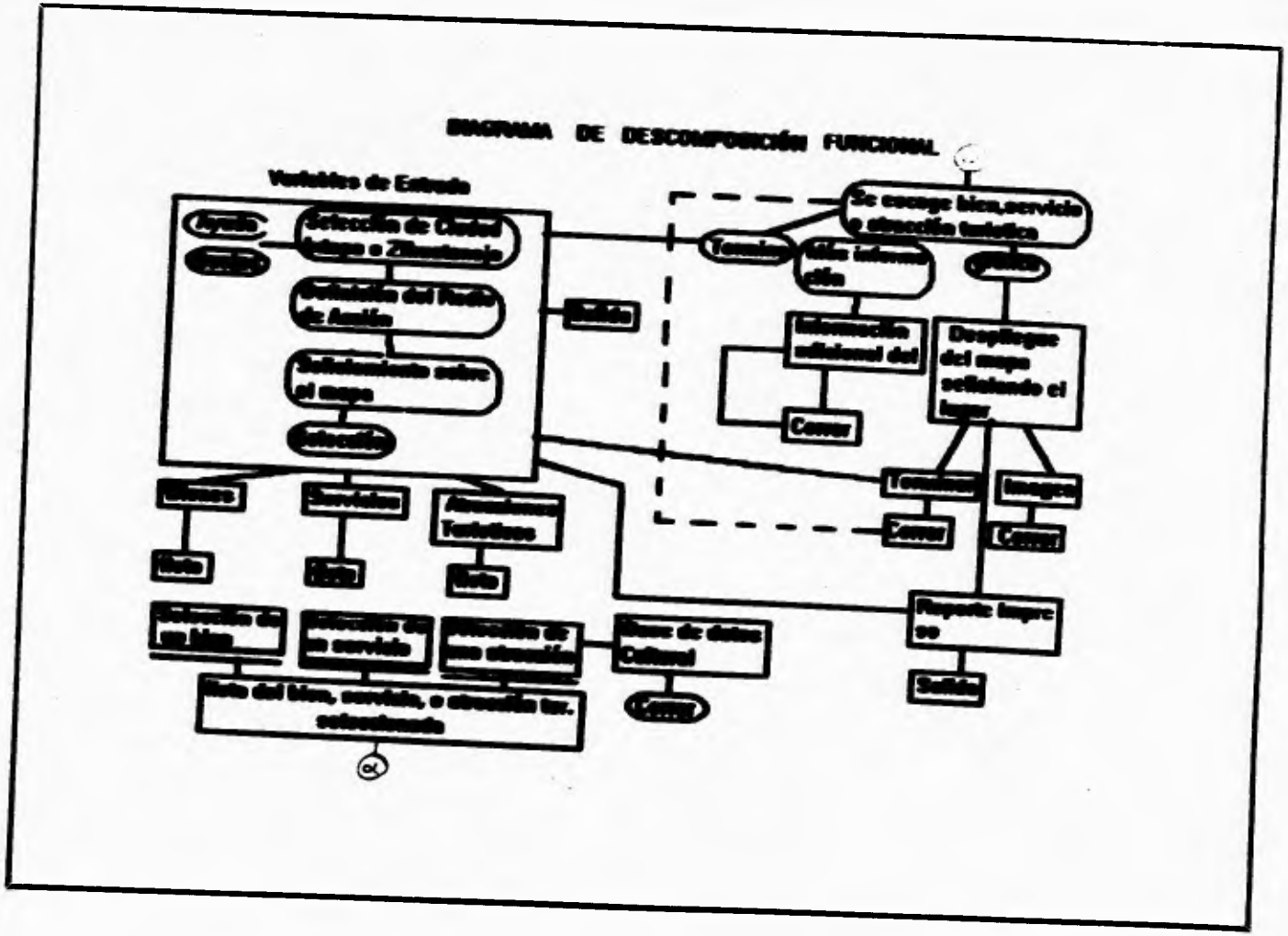


DIAGRAMA DE DESCOMPOSICIÓN FUNCIONAL
Variables de Estado
Ayuda
Salir
Selección de Ciudad
Mapa o Zhumtamiento
Solicitud del Punto de Arribo
Señalamiento sobre el mapa
Ubicación
Mapas
Servicios
Atracciones Turísticas
Info
Info
Info
Base de datos Cultural
Se escoge bien,servicio o atracción turística
Terminar
Información adicional del
Comer
Despliegue del mapa señalando el lugar
Terminar
Imagen
Comer
Comer
Reporte impreso
Salida

En la primera parte podemos ver el modulo de entrada de variables en la cual el usuario escogerá la ciudad, el radio de definición de acción y el punto de interés; teniendo esta información el sistema desplegará las listas de información de bienes y servicios disponibles. De estas listas el usuario escogerá el bien o servicio de su interés. El sistema entonces desplegará la información especifica del lugar textual y graficamente. El usuario podrá tener la opción de imprimer la información obtenida para luego volver a solicitar información o salir del sistema.

2.5 OPCIONES DE SOLUCIÓN.

CARACTERISTICAS, SELECCION DEL SOFTWARE DE BASE DE DATOS Y PAQUETE DE DESARROLLO DEL SISTEMA.

En esta época sobresalen los sistemas para el manejo de base de datos (DBMS), los archiveros de la edad electrónica. Si se busca entre el papeleo de cualquier organización en el mundo, seguramente se encontrará una gran carga de documentos solicitando el manejo de una base de datos: la lista de clientes para la sala de alquiler de videos, el inventario de cassettes en una tienda de música o los registros de personal de una empresa. De igual manera sobresalen los paquetes de computación que proporcionan las herramientas necesarias para el desarrollo de sistemas más amigables al usuario mediante un ámbiente gráfico.

A continuación se presenta una muestra típica de paquetes de bases de datos con capacidades semejantes. Es importante mencionar que lo que se va a apreciar en las descripciones no es necesariamente todo lo que puede obtenerse. En muchos casos, se pueden aplicar las capacidades básicas de las bases de datos. Además de mencionar dos paquetes de programación orientada a objetos para el desarrollo de sistemas mediante un ámbiente visual.

CLIPPER.

Este paquete para el desarrollo de bases de datos es, definitivamente para los programadores. Si bien carece de algunas de las excelentes funciones de generación de códigos que tienen los demás paquetes, Clipper ofrece una riqueza de armas y capacidad que los programadores necesitan.

Dos de estas armas son un generador de reportes (RL) y una función (DBU) para la creación y manejo de los archivos de la base de datos, escritos en el propio lenguaje de programación de Clipper, y se incluye el código fuente, el cual puede utilizarse como referencia, o modificarlo para añadirlo a las aplicaciones de la base de datos.

La pantalla de DBU enlista las opciones a través de su parte superior, junto con las teclas de funciones asignadas. El resto de la pantalla está dedicado a una representación visual del panorama de la base de datos activa, el cual consiste en una columna partida en tres secciones. El primer grupo exhibe el nombre de la base de datos activa. Si al invocar el programa DBU se invoca un argumento, aparecerá aquí la base de datos o la vista especificada. En la siguiente sección aparecerán los índices activos relacionados con la base de datos activa, y el grupo al fondo muestra los nombres de los campos para la base de datos activa.

El generador de reportes, RL, no es tan flexible , no soporta un bosquejo WYSIWYG del reporte y tampoco permite ver la salida en forma preliminar conforme se vaya trabajando. La verdadera potencia de Clipper se basa en su codificación.

El compilador del programa es muy rápido además de tener algunas funciones muy interesantes, dentro de estas se destacan las siguientes: soporta los llamados bloques de

código, pedacitos de código ejecutable que se pueden almacenar como variables, o pasar a otros programas como argumentos, para ejecutar un bloque de código, se utiliza una función EVAL( ); otra función útil es el uso hecho por Clipper de los archivos cerrados para hacer el seguimiento, a fin de determinar cuales archivos en un programa dependen de otros para operar correctamente y quedar al corriente.

Estando instalado este sistema, se puede invocar la función MiMake para llevar a cabo aquellas operaciones de compilación y enlace que se necesiten y mantener todos los archivos sincronizados. Clipper también soporta las funciones para leer los archivos binarios de DOS y escribir en ellos. También se aprecia el depurador de Clipper, el cual permite analizar el funcionamiento de código del programa, ejecutar comandos y revisar el estado de algunas variables, nombres de campos o expresiones en particular. Una ventanilla de estado enlista las bases de datos abiertas en todas las áreas de trabajo activas, como los valores de todos los comandos Set.

DBASE.

Ya pasaron a la historia los días cuando Dbase establecía las normas del manejo de las bases de datos en DOS. Dbase llegó a la cumbre principalmente por la fuerza de su poderoso lenguaje de programación, los retrasos abrieron la puerta para los paquetes con la misma potencia de programación, escondida bajo interfaces más sencillas, Dbase está llegando a ser más fácil de utilizar cada día.

El programa ya ofrece interfaces, manejado por menú que se conoce como control. Aunque el centro de control representa un mejoramiento para los usuarios intimidados por un sólo punto en una pantalla en blanco, la simple añadidura de menús accesibles no convierte esto en un producto de uso agradable.

FOXPRO.

La interface de FoxPro incluye menús presentados junto con una ventanilla de comandos conveniente, para utilizar los menús o escribir los comandos sin que ninguno de los elementos de la interface interfieran con otro. Su soporte al ratón es el mejor de todos los paquetes basados en caracteres. Asimismo, la ventanilla de comandos mantienen una historia corrida de las intrucciones (al igual que Dbase III plus), esto facilita la repetición de los comandos utilizados a través de una sesión. También se puede selecionar parte de la historia de los comandos, y anexarla a sus aplicaciones.

La ventana presenta gráficamente todas las áreas de trabajo disponibles; se puede seleccionar un área de trabajo disponible, se puede seleccionar un área de trabajo y abrir en ella una base de datos, accionando un botón de comando. FoxPro ofrece una caja de diálogo con una lista de los campos principales.

El contructor de pantalla empieza como pantalla en blanco, en la cual se puede capturar el texto y colocar los campos, también se pueden crear botones de comando, casillas, marcar con "palomas", botones de "radio" y listas de extraer. Se pueden agregar pedacitos de código a cualquier objeto, incluso a los campos. Desde la pantalla de establecimiento, se puede ejecutar código antes y después del programa de la pantalla de captura, cuando se genere el código para la pantalla diseñada, se pueden anexar otras pantallas a ella, lo cual ahorra tiempo una vez formulada una biblioteca de pantallas genéricas.

El funcionamiento de FoxPro es magnífico gracias, en parte, a su tecnología exclusiva de Rushmore, siendo su único inconveniente su voluminosa documentación, ya que es un problema localizar la información rápidamente, lo cual no obstante disminuye gracias a su fuerte función de ayuda en línea.

INFORMIX-SQL.

Informix SQL es cien por ciento una base de datos con lenguaje de consulta estructurado (SQL), el aspecto del programa es muy austero: no se encontrará con pantallas de colores múltiples con menús y ventanillas a la vista. Las pantallas del paquete, estilo Lotus, impulsadas por menús, automatizan las operaciones más significativas de la base de datos, tales como la creación de tablas, la definición y modificación de los campos, etc. Informix también incluye un generador de reportes, y un sistema para la ejecución de archivos de la definición de formas diseñadas.

En Informix se construye una forma, no moviendo un cursor sobre la pantalla con un ratón o teclas de fecha, sino escribiendo un archivo para la especificación de la forma, un tipo de definición de pantalla acompañado con instrucciones ejecutables. Dicho archivo comprende cinco partes: una sección de tablas, que identifica cuáles tablas serán accedidas por la forma; una tabla de atribuciones, la cual describe cada campo exhibido por la forma, y una sección opcional de instrucciones, que define las operaciones que habrán de ser llevadas a cabo sobre los campos dentro de la forma.

Las secciones y tablas informan al sistema que debe presentar; la sección de pantalla indica en donde presentarlo, la sección de atribuciones indica como presentarlo y la sección de instrucciones le dice al sistema que hacer antes, mientras y después de presentarlo.

Se incluye con Informix-SQL varios paquetes de servicios: BECHECK verifica la integridad de los indices, si encuentra una discrepancia entre un archivo de datos y uno de sus indices, le permite reformar el indice; DRLINK y DBLOAD son de utilidad para el traslado de los datos entre Informix y el mundo exterior de archivos de Lotus 1-2-3, Dbase o ASCII; con DBSCHEMA, se pueden elaborar las intrucciones de SQL requeridas para crear una tabla o una base de datos.

PARADOX.

Una de las características más interesantes de PARADOX es su velocidad, destacándose su rapidisimo tiempo de respuesta para la lectura y edición de las tablas.

Paradox soporta los formatos de importación y exportación más importantes, pero existen problemas en la importación de información almacenada en el formato ASCII. Con Paradox, es necesario planear con cuidado antes de precipitarse a la construcción de una forma, aplicando esto principalmente con las formas que accesan a múltiples tablas. El diseño de formas con tablas múltiples exige la creación de una forma maestra, después, existe también la introducción, en dicha forma maestra de las formas incluidas en las otras tablas. Así que, para crear la forma maestra completa, deben diseñarse las formas que se incluirán primero.

El lenguaje para el manejo de la base de datos de PARADOX es PAL, aunque es erróneo representar a PAL como simplemente un DML. Lo que dificulta la programación con PAL es que maneja la transmisión de comandos a un robot sentado en un teclado operando Paradox; sin embargo, si simplemente no se quiere tener nada que ver con PAL, podrá instalarse el programador personal. Este programa es, esencialmente un constructor de aplicaciones que guía a través de la creación de una aplicación completa con menús y formas, ofreciendo el código PAL como su producto final.

PARADOX.

Una de las características más interesantes de PARADOX es su velocidad, destacándose su rapidisimo tiempo de respuesta para la lectura y edición de las tablas.

Paradox soporta los formatos de importación y exportación más importantes, pero existen problemas en la importación de información almacenada en el formato ASCII. Con Paradox, es necesario planear con cuidado antes de precipitarse a la construcción de una forma, aplicando esto principalmente con las formas que accesan a múltiples tablas. El diseño de formas con tablas múltiples exige la creación de una forma maestra, después, existe también la introducción, en dicha forma maestra de las formas incluidas en las otras tablas. Así que, para crear la forma maestra completa, deben diseñarse las formas que se incluirán primero.

El lenguaje para el manejo de la base de datos de PARADOX es PAL, aunque es erróneo representar a PAL como simplemente un DML. Lo que dificulta la programación con PAL es que maneja la transmisión de comandos a un robot sentado en un teclado operando Paradox; sin embargo, si simplemente no se quiere tener nada que ver con PAL, podrá instalarse el programador personal. Este programa es, esencialmente un constructor de aplicaciones que guía a través de la creación de una aplicación completa con menús y formas, ofreciendo el código PAL como su producto final.

VISUAL BASIC.

Visual Basic para Windows es un sistema de programación computarizado excitante. Visual Basic ha sido tremendamente aceptado, y Microsoft puso un poco de esfuerzo para integrarlo a la versión 3 de Windows. El poder, flexibilidad y velocidad de Visual Basic está ahora a la par con el lenguaje C unicamente, y en cuanto a productividad, Visual Basic es claramente superior.

Un programador medio necesita meses para tener una velocidad de programación para Windows usando el lenguaje C, pero con Visual Basic se pueden desarrollar aplicaciones para Windows con solo algunas horas de familiarizarse con el lenguaje tanto del paquete como de programación Basic. Se podra estar satisfecho al ver la primera aplicación desarrollada para Windows con un trabajo fácil y tan rápido como sea posible.

Visual Basic es altamente interactivo, manejando un lenguaje de programación, con el cuál permite aprender de ambas formas agradable y productivamente.

Visual Basic utiliza varias formas para sus presentaciones visuales de sistemas (En terminologia de Visual Basic, una *forma* es una ventana asociada a controles, iconos, gráficos y código); la aplicación COLORBAR permite trazado de gráficas. La forma GETFILE permite al usuario seleccionar un archivo desde cualquier directorio en cualquier drive.

Mediante Visual Basic se puede generar un sistema que soporte la instalación y uso del ratón, para facilidad del propio sistema y del usuario; genera Cambios de Datos Dinamicos (DDE) para aplicaciones basadas en ventanas; trabaja con bases de datos en una interacción; y Objetos Ligados y Emprotados (OLE).

VISUAL C++.

Una parte importante de Visual C++ y el corazón de la aplicación framework es la versión 2.0 del mismo. La librería de clases consiste de una librería de clases de C++ y funciones globales con código fuente incluido. Otros componentes de Visual C++ - incluyendo AppWizard, ClassWizard, App Studio, Visual Workbench, el compilador, el ligador- son las herramientas que se pueden usar para construir sus aplicaciones.

Otra característica de las librerías de aplicación que ofrece Visual C++ es la Interface de Documentos Múltiples para Windows (MDI); además de la interface de programación de Conectividad de Bases de Datos Abiertas de Microsoft (ODBC); así como tambien Objetos Ligados y Empotrados (OLE) y Librerías de Ligados Dinamicos (DLLs).

Visual C++ debe de estar familiarizado con el lenguaje de programación C++ y el Desarrollo de Software para Windows (SDK) para lenguaje C.

Algunas de las ventajas de utilizar Visual C++ son: aparte del manejo de programación C++ que tiene mayores beneficios en cuanto a velocidad y capacidades de manejo de memoria que Visual Basic por ejemplo donde el lenguaje de programación es Basic; Visual C++ trabaja en un ámbiente Windows que muestra grandes ventajas sobre el medio ámbiente de MS-DOS, y el desarrollo de sus aplicaciones se pueden hacer en un ámbiente gráfico parecido a Windows, con iconos y ventanas para producir desarrollo de sistemas más amigables, seguros, confiables, de fácil manejo para el usuario en general y sobre todo de calidad en los productos.

Selección de herramientas de Software

La medida final del desempeño de una computadora no solamente es el diseño del hardware. Un sistema verdadero y efectivo de computadora combina un buen diseño de hardware en su arquitectura, un sistema operativo poderoso, una aplicación de software versátil que proporcione el desempeño y recursos necesarios para cualquier aplicación.

Hoy en día, el mercado demanda un ambiente de aplicaciones "estándar", donde las aplicaciones pueden ser desarrolladas y utilizadas sobre hardware de múltiples proveedores sin modificarlas o reescribirlas. Así la tecnología mejora el costo de desarrollo de aplicaciones, el mantenimiento y la ejecución. Esto implica una mayor posibilidad de encontrar algún producto que satisfaga con mayor exactitud las necesidades de los usuarios.

En cuanto al sistema que se propone, se va a utiizar el manejador de Base de Datos de Paradox for DOS, ya que el manejador servira de alguna manera para la captura de información de los principales prestadores de bienes y servicios del lugar en particular y del manejo de las mismas base de datos, para actualizar y corregir su información. En cuanto al paquete de programación que realizara la integración de todo el sistema será el Visual C++ debido a las características mencionadas, al desarrollo del sistema con un lenguaje de programación con las ventajas y beneficios de C++, así como del ambiente gráfico que se puede explotar al desarrollar Visual C++, produciendo un sistema de calidad para el usuario y facilidad en el manejo de la información.

Por lo anterior podemos concluir que utilizaremos Paradox for DOS como manejador de bases de datos, ya que cuenta con la programación orientada a objetos; y que lo verdaderamente importante será el integrar la información obtenida en un sistema de presentación visual como lo es Visual C++.

Elección del Manejador de Bases de Datos.

En cuanto al manejador de bases de datos; se pretende utilizar Paradox for DOS; cuyas caracteristicas y el porqué de la elección(información obtenida de Benchmarks) se describen a continuación:

Con la nueva versión de *Paradox for DOS*, el usuario trabaja inmediatamente y a todo vapor, ya que incluye *Expertos* integrados en su propia computadora, quienes le muestran la manera más fácil de realizar su trabajo con un solo click del ráton.

Paradox for DOS cuenta con *Expertos en Enlaces*, con lo cual el usuario puede llegar a dominar una base de datos fácilmente, debido a que podrá crear enlaces de manera visual, lo único que tiene que hacer es dibujar líneas entre las mismas y el *Experto de Enlaces* le muestra automáticamente cómo enlazar las tablas y realizar la intrincada labor de relacionarlas, no importando qué tan compleja sea la relación.

Los *Expertos en Formatos y Reportes* ayudan al usuario a elaborar bocetos con especificaciones precisas instantáneamente. Con un click del ratón, el usuario le pide al *Experto* que vaya cambiando los diseños hasta encontrar el que más le agrade, por ejemplo, el *Experto en Formatos o Reportes* podrá colocar sus campos horizontal o verticalmente, seleccionar o incluir únicamente los campos que el usuario elija o bien desplegar registros tales como una tabla dentro del diseño instantáneamente y con un click del ratón, el Experto puede acomodar sus campos en forma de etiqueta de correo si así lo solicita, cualquier cambio se puede realizar también de manera instantánea.

El *Experto en Consultas (Query Expert)*, le permite elaborar consultas basadas en sus formatos y reportes existentes. El usuario elige el formato o reporte que desee y el Experto recuperará los datos de una o más tablas y las colocará todas dentro del *"query"* o consulta, creando los enlaces automáticamente; basta con marcar la información que usted desee, pulsar luego un botón y correr la consulta.

PRUEBAS DE BENCHMARK.

Para evaluar la ejecución de las bases de datos multiusuarios, la revista PC Magazine desarrolló un test que prueba cada límite del producto, a través de poner en un ambiente multiusuario muy demandante. Como comparación previa, la revista Magazine escoge utilizar un subgrupo del AS3AP (ANSI SQL Standard Scalable and Portable). Las pruebas para sistemas de bases de datos relacionales, desarrollados por Carolyne Turbyfill, Dina Bitton, y Cyril Orji en la Universidad de Cornell. AS3AP es una industria aceptada de Pruebas que cubren un amplio espectro de operaciones comunes de bases de datos y es también escalable, los resultados válidos son proporcionados en plataformas de PC's a mainframes.

Este año, PCLabs hizo unas cuantas modificaciones a su prueba bed. Mientras que las pruebas del año pasado se enfocaron casi exclusivamente en resultados multiusuario ("9 bases de datos multiusuarios: listas para compartir", PCMagazine, Marzo 31 de 1992), Las pruebas de este año incluyen varias pruebas para evaluar la ejecución de selecciones, úne y actualiza una sola estación de trabajo. La brueba bed fué actualizada también. Para las pruebas de uso único, se usaron una Compaq Deskpro 486/33M con 8MB de RAM y 310MB de discoduro. El servidor fué una Compaq Systempro 486/33 con 32MB de RAM (sube de 16MB para el último año) y 1.6GB de alta ejecución del que almacena el disco duro mediante el uso del controlador IDA (Intelligent Drive Array) de Compaq.

Para incrementar la resistencia de la prueba de red mientras se evitan los cuellos de botella del hardware, PCLabs probó arriba de 36 estaciones de trabajo y balanceó el tráfico de red simétricamente mediante el uso de los adaptadores Token-Ring EISA de 32 bits en el archivo del servidor. Las estaciones de trabajo fueron idénticamente equipadas con computadoras 386SX/16 de Compaq Deskpro con 5MB de RAM. Se realizaron todas las pruebas usando NetWare 3.11 en una red Token-Ring a 16megabit por segundo.

A cada fabricante se le dió una prueba de especificación detallada para correr nuestras pruebas, y se requirió de un código optimizado de módulos desarrollados en el análisis del manejador de la base de datos (DBMS) para completar cada operación. Los fabricantes no permitieron usar cualquier programa desarrollado o compilado con herramientas que no están incluídos en sus productos. Además un representante de cada fabricante fué presentado en PCLabs por una semana de pruebas para asegurar que los productos eran propiamente instalados y probados.

Todos los productos fueron instalados en una configuración multiusuario. Se instaló la revisión productos DOS en el archivo del servidor de la red, mientras los productos basados en Windows fueron instalados localmente en cada estación de trabajo, con Microsoft Windows 3.1.

La Carga e Indice de prueba mide cómo cada paquete de base de datos puede importar un índice a 4 tablas de base de datos con 100000 registros por tabla. Dos tablas contienen cada una dos índices, la tercera tabla contiene 4 índices y la cuarta 5.

Cada tabla tiene sus propias características distintas de distribución de datos: una contiene valores únicos por cada columna, otra contiene 10 valores únicos porcentuales por

cada columna, la tercera contiene 100 valores distintos por cada columna y la cuarta tiene varios usos de distribuciones y es usada para las pruebas que implican actualización.

Los archivos importantes están localizados en un volumen de NetWare y disco de canal fisico que están separados del destino de la base de datos, y la importación e indexado son ejecutados desde una estación de trabajo 486/33.

Antes de la prueba, se permitió a cada fabricante especificar su preferencia por archivos desmarcados, archivos ASCII delimitados por coma ó formatos ASCII de longitud compuesta.

Ambos Microsoft FoxPro para DOS y Microsoft FoxPro para Windows superan en la carga e indice de prueba, completando ambos pasos en, alrededor de 20 minutos cada uno menos que la mitad del tiempo del siguiente-lugar del producto, Parodox para DOS. DataEase fué por mucho el producto más bajo en esta prueba. Sin embargo, DataEase fué cargado por 4 tablas en menos de una hora, el lento indexado fué producido en un tiempo total de más o menos 5.5 horas.

Paradox para Windows requiere un segundo paso para proceder a importar los datos. Mientras que la versión de DOS tubo un comando delimitado por Append al agregar datos a una tabla existente y pudo copiar los datos del archivo original de ASCII. Parados para Windows requirió de un importante operación, seguida de una tabla de operación de adición, logrando así una anotación similar. Esto duplica el tiempo de procesamiento, cada registro tuvo que ser procesado dos veces.

Cuando se importó cada una de las cuatro tablas de pruebas, descubrimos un virus que resultó en diferentes grados de datos perdidos durante este proceso. Desde que el

producto no fué exitosamente cargado las tablas e indices exactos, en su puntación en la carga e indexado de prueba fué "erronea".

La prueba **Select** mide qué tan rápido cada paquete de base de datos puede ejecutar tablas únicas de preguntas donde regresa el 10% de los registros. Ejecutamos dos tipos de preguntas. La selección numerica usa la llave primaria de la tabla y un rango secuencial como registro de criterio. La pregunta alfanumérica ejecuta una pareja-exacta en un campo de texto usando un índice secundario.

Tres de los productos probados -Microsoft Access, FoxPro para DOS y FoxPro para Windows- regresaron una lista de apuntadores a los datos seleccionados en memoria y no el valor actual de las columnas especificadas. Sin embargo, estos productos ejecutan la pregunta que es requerida casi instantaneamente, ellos deben gastar tiempo adicional para recuperar el dato real si una operación más lejana va a ser ejecutada en el conjunto de resultados. Los productos como Paradox y R:Base regresan los datos; mientras se ejecuta cada pregunta, ellos no requieren este acceso adicional a la base de datos.

Access usa un fondo de preguntas de procesamiento y regresa el control después de la primera pantalla de datos disponible, se emite un comando dentro del tiempo de la prueba que es enviada por Access al final de ésta.

Sin embargo, ambas versiones de Paradox regresan en forma rápida la porción número de la prueba Select, la selección de tiempos alfanumérica sufrió de un uso deficiente del índice secundario. Con Paradox, si el resultado de una pregunta es menor del 8%, aproximadamente, de la tabla entera, el índice secundario ayuda a la ejecución; el índice estorba la ejecución. Nuestra pregunta alfanumérica regresó 10% de la tabla.

Para la prueba de Join, se ejecutaron varias de las uniones especificadas en el AS3AP.

Sin embargo, las uniones regresaron pequeños resultados que, generalmente se procesaban rápidamente, las dos tablas de uno a muchos produjó un amplió rango de funcionamiento, dependiendo del plan de optimización utilizado.

DataEase, KnowledgeMan y Paradox para DOS optimizaron la pregunta usada en esta unión. En contraste, Paradox para Windows y Access se tornaron en tiempos más bajos. A diferencia de Paradox para DOS, Paradox para Windows usa la ingeniería de base de datos de Borland, la cual, en su primera emisión no es retornada para la ejecución.

Para la prueba Update, ejecutamos volúmenes actualizados, inserciones y supresiones (Modify, Append y Delete, respectivamente). Primero salvamos 1000 registros de la tabla a una tabla temporal. En seguida, se actualizó el rango correspondiente a la mitad de la tabla de prueba y cambiando el valor de la llave primaria la cuál forza a la base de datos a actualizar su índice. Para medir la rápidez de los productos cuando se están insertando los registros en la tabla de prueba. Finalmente se borrarón 1000 registros de la tabla.

La más difícil de las 3 operaciones para la mayoría de los productos fué actualizar (update). Advanced Revelation y KnowledgeMan ambos tomaron un largo tiempo para hacer esto, debido al índice requerido de organización. Paradox para DOS demostró ser eficiente en la ejecución de las 3 operaciones completando todas ellas en 5 segundos. Ambas versiones de FoxPro también demostraron una rápida manipulación de la tarea Modify (modificación).

Pruebas de Ejecución: Bases de Datos Relacionales.

La prueba **Random Read** (Lectura Aleatoria) dá una idea del número máximo de ocurrencias que cada paquete puede manejar. En esta prueba, cada estación de trabajo casualmente selecciona hileras de la misma tabla y las descarta dirigiendo la salida al mecanismo NUL. Los registros son accesados en el modo de browse para maximizar la cantidad de ocurrencias.

Se corré la prueba por 2 hr 10 min. Para determinar dónde ocurre la mejor etapa, incrementamos el número de estaciones de trabajo gradualmente, adicionando estaciones en intervalos de 10 minutos hasta 36 estaciones de trabajo que están corriendo o hasta que el servidor no pueda soportar más conexiones. La etapa está calculada en transacciones por segundo por cada 10 min. de intervalo.

De nuevo, no hay "tiempo para pensar" insertando entre cada repetición de la prueba, la red cargada es muchas veces lo que el mismo número de estaciones de trabajo produciría en una situación de la vida real.

Los 3 productos -FoxPro para DOS, FoxPro para Windows y Paradox para DOS- destacarón; cada uno alcanzó una etapa por encima de 3000 transacciones por segundo con una carga de 36 estaciones de trabajo. Sin embargo, Paradox para DOS llegó a 32 estaciones de trabajo, la curva de la etapa para ambas verisones de FoxPro quedó esencialmente lineal a las 36 estaciones de trabajo, indicando que el máximo momento para el producto no ha sido todavía alcanzado.

Access también funcionó consistentemente en la prueba de Random Red, un alcance de 846 transacciones por segundo con 36 estaciones de trabajo. Paradox para Windows

funcionó en nuestra de Random Read en forma desalentadora, punteando en 6 estaciones de trabajo y luego estabilizando cerca de 60 transacciones por segundo.

Para la prueba de **Report Generation** (Generación de Reporte), una estación de trabajo imprime un reporte mientras otras ocho estaciones de trabajo ejecutan la prueba de Random Read en el fondo. Para producir el reporte el paquete tiene que unir dos tablas de 100000 hileras y generar un total de 1000 hileras usando calculos de fechas, valores mínimos y máximos, subtotales y totales. Los resultados son impresos en un archivo ASCII en la estación de trabajo que está generando el reporte.

De nuevo, ciertos productos funcionan considerablemente mejor que el resto. Ambas versiones de FoxPro, Paradox para DOS y Access producen el reporte considerablemente más rápido que como lo hicieron los otros productos. Cada versión deFoxPro completó la prueba en menos de dos minutos. En el otro extremo del espectro, KnowledgeMan y Paradox para Windows cada uno tomó más de media hora en completar el reporte y R:Base fué el más lento, tomando 46 minutos para completar el reporte.

La prueba de **Random Write** (Escritura Aleatoria) da una idea del máximo número de ocurrencias actualizadas que el paquete puede ejecutar. En esta prueba, cada estación de trabajo aleatoriamente actualiza hileras individuales de la misma tabla. Se modificó el valor de un campo indexado así que ambas hileras seleccionadas y el índice deben ser actualizados. Las hileras son accesadas en modo update, el cual permite compartir el dato en la tabla, pero no admite que cualquier otra estación de trabajo tenga acceso a una hilera que está siendo actualizada.

Se corrió la prueba por 2 hr 10 min. Para determinar donde el punto de la etapa ocurre, incrementamos el número de estaciones de trabjo gradualmente, adicionando

estaciones en intervalos de 10 minutos hasta que 36 estaciones de trabajo estuvieran corriendo o hasta que el servidor no soporte por más tiempo más conexiones. La etapa es calculada en transacciones por segundo cada uno con 10 min. de intervalo. La red de carga es muchas veces lo que el mismo número de estaciones podría producir en una situación de la vida real.

Ambas versiones de FoxPro se colocaron muy aparte del paquete. La gráfica de ejecución muestra que su etapa estaba justo nivelando 36 estaciones de trabajo con cerca de 770 transacciones por segundo -casi 3 veces el nivel en la mejor ejecución en esta prueba, DataEase.

Ambas versiones de Paradox funcionan penosamente en esta prueba, debido al alto número de requerimientos simultáneos. Para obtener registros protegidos en la misma tabla, Paradox estación de trabajo debe adquirir "sección critica de protección" en el mismo archivo de protección. Para ocurrencias de requerimientos de protección a una sola tabla los clientes de Paradox deben de accesar a este mismo archivo de semáforo en un proceso serial, creando una ejecución sustancial de cuello de botella. Paradox para DOS se punteó en 3 estaciones de trabajo con un total de 42 transacciones por segundo, mientras Paradox para Windows punteó solo 5 transacciones por segundo durante la prueba.

Superbase también dió una pobre actuación en nuestra prueba de Random Write, porque estabamos obligados a usar tablas de protección en lugar de hileras de protección.

Como complemento de la prueba Random Write, PC Labs corrió una rutina de verificación para asegurar que cada registro actualizado fuerá exitosamente completado en la base de datos. Debido a que un virus en R:Base envolvió la protección de contención del producto, no fué exitosamente completado en todas las actualizaciones, así que no se

transportadora le permite mover fácilmente aplicaciones entre DOS y Windows, y la nueva capacidad de compilación permite a los desarrolladores mantener un sólo código de base para ambas versiones de sus aplicaciones.

Paradox no tiene la rapidez de FoxPro, pero trae bases de datos relacionales a un nivel completo y nuevo de sofisticación, combinando una consistente interfase de orientación de objetos, e integridad referencial, y acceso transparente a los archivos de formato múltiple. En futuras versiones, Paradox se volverá aún más fuerte.

Otros productos que revisamos son menos atrayentes pero dicnos de ser notados. Superbase toma una gran ventaja sobre la interfase de Windows y es fácil de usar, sin embargo, le encontramos carencias en los caballos de fuerza para mantenerlo en un volumen alto en el ambiente de multiusuarios. DataEase, en forma experta, combina una poderosa aplicación constructora con facilidad de uso, pero otra vez a expensas de la rapidez.

EVALUACION Y SELECCION DE LAS OPCIONES DE SOLUCION.

Un programa de computadora es el factor que viene a marcar la diferencia. La elección de como solucionar nuestro problema de implementar un sistema en computadora la podemos encontrar diferenciando primero, los tipos de sistemas que existen y, en segundo lugar, revisando las técnicas o herramientas que tenemos para el desarrollo de estos.

Los sistemas los podemos clasificar en los siguientes tipos:

SISTEMAS DE TIEMPO REAL. El software que mide, analiza y controla sucesos del mundo real conforme ocurren se llama de tiempo real. Los elementos del software de tiempo real incluyen una componente de acumulación de datos que recolecta y formatea la

información de un entorno externo, una componente de control/salida que respalda al entorno externo y una componente de monitorización que coordina a todas las demás componentes, de forma que pueda mantenerse la respuesta en tiempo real.

SISTEMA DE GESTION. El procesamiento de información comercial constituye la mayor de las areas de aplicación del software. Los "sistemas discretos" (nóminas, inventario, etc.) han evolucionado hacia el software de sistemas de información de gestión, que accede a una o más bases de datos grandes que contienen la información comercial. Las aplicaciones en esta área reestructuran los datos existentes en orden a facilitar las operaciones comerciales o gestionar la toma de decisiones. Además de las tareas convencionales de procesamiento de datos, las aplicaciones de software comercial también realizan cálculo interactivo (por ejemplo, el procesamiento de transacciones en puntos de ventas).

SISTEMAS DE INGENIERIA Y CIENTIFICOS. El software de ingeniería y científico se ha caracterizado por los algoritmos de "manejo de números". Las aplicaciones van desde la astronomía a la vulcanología, desde el análisis de la presión de los automotores a la dinámica orbital de las lanzaderas espaciales y desde la biología molecular a la fabricaciín automática. Sin embargo, la nuevas aplicaciones del área de ingeniería/científica se han alejado de los algoritmos convencionales numéricos. El diseño asistido por computadora (CAD), la simulación de sistemas y otras aplicaciones interactivas, han comenzado a tomar características del software de tiempo real.

SISTEMAS EMPOTRADOS. Los productos inteligentes se han convertido en algo común en casi todos los mercados de consumidores e industriales. El software empotrado reside en memoria de sólo lectura y se utiliza para controlar productos y sistemas

de los mercados industriales y de consumidores. El software empotrado puede ejecutar funciones muy limitadas (por ejemplo, el control de las teclas de un horno de microondas) ó suministrar una función significativa y capacidad de control (por ejemplo, funciones digitales en un automóvil, tales como control de gasolina, sistemas de frenado, etc.).

SISTEMAS DE INTELIGENCIA ARTIFICIAL. El software de inteligencia artificial (IA) hace uso de algoritmos no numericos para resolver problemas complejos que no son adecuados para el cálculo o análisis directo. Actualmente, el área más activa de la IA es la de los sistemas expertos, también llamados sistemas basados en el conocimiento. Otras áreas de aplicación para el software de IA es el reconocimiento de patrones (imágenes y voces), prueba de teoremas y juegos.

Las técnicas o herramientas para el desarrollo de sistemas con que se cuenta son:

EL CICLO DE VIDA CLASICO. La figura 2.5.1 ilustra el ciclo de vida clásico para la ingeniería de software. Algunas veces llamado el "modelo en cascada", el ciclo de vida exige un enfoque sistemático, secuencial, del desarrollo del software que comienza en el nivel del sistema y progresa a través del análisis, diseño, codificación, prueba y mantenimiento. El ciclo de vida clásico abarca las siguientes actividades:

Ingeniería y análisis del sistema. Debido a que el software es siempre parte de un sistema mayor, el trabajo comienza estableciendo los requerimientos de todos los elementos del sistema y luego asignando algún subconjunto de estos requerimientos al software. Esta visión del sistema es esencial cuando el software debe interrelacionarse con otros elementos tales como hardware, personas y bases de datos. La ingeniería y análisis del sistema abarca los requerimientos globales a nivel del sistema con una pequeña cantidad de análisis y diseño a nivel superior.

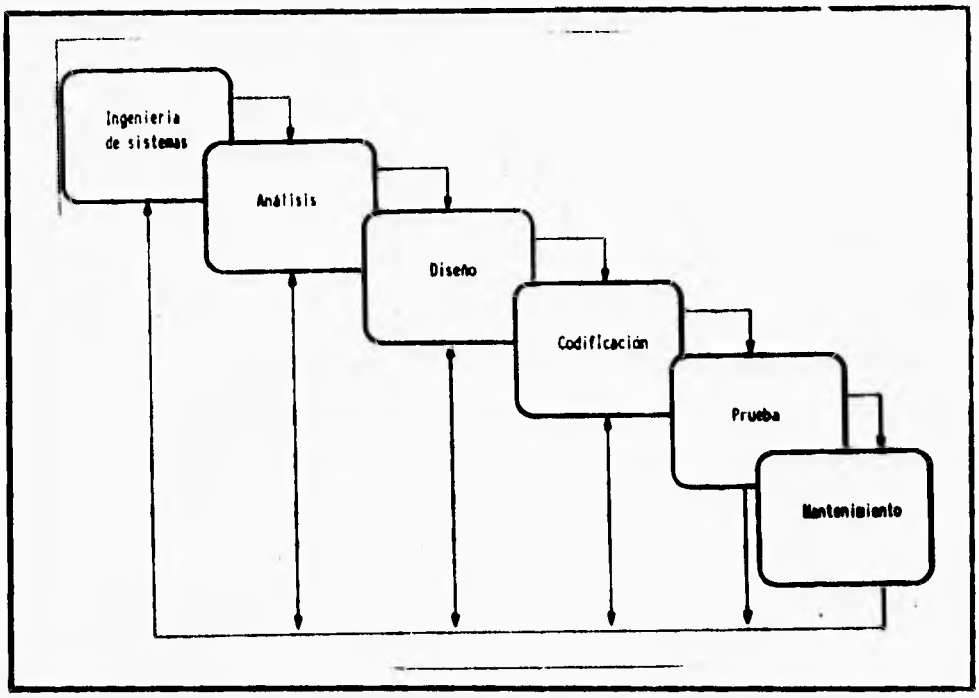


Fig.2.5.1. El ciclo de vida clásico.

Análisis de los requerimientos del software. El proceso de recogida de los requerimientos se centra e intensifica especialmente en el software. Para comprender la naturaleza de los programas que hay que construir, el ingeniero en software debe comprender el dominio de la información, así como la función, rendimiento e interfaces requeridas. Los requerimientos tanto del sistema como del software se documentan y revisan con el cliente.

Diseño. El diseño del software es realmente un proceso multipaso que se enfoca sobre tres atributos distintos del programa: estructura de datos, arquitectura del software y detalle de procedimientos. El proceso de diseño traduce los requerimientos en una

representación del software que pueda ser establecida de forma que obtenga la calidad requerida antes de que comience la codificación.

Codificación. El diseño debe traducirse en una forma legible para la máquina. El paso de la codificación ejecuta esta tarea. Si el diseño se ejecuta de una manera detallada, la codificación puede realizarse mecánicamente.

Prueba. Una vez que se ha generado el código, comienza la prueba del programa. La prueba se enfoca sobre la lógica interna del software, asegurando que todas las sentencias se han probado, y sobre las funciones externas, esto es, realizando pruebas para asegurar que la entrada definida producirá los resultados que realmente se requieren.

Mantenimiento. El software sufrirá cambios después de que se entregue al cliente. Los cambios ocurrirán debido a que se han encontrado errores, debido a que el software debe adaptarse por cambios del entorno externo, o debido a que el cliente requiere actualizar su información con el transcurso del tiempo.

CONSTRUCCION DE PROTOTIPOS. La construcción de prototipos es un proceso que facilita al programador la creación de un modelo del software a construir. El modelo tomará una de las tres formas siguientes: un prototipo en papel que describa la interacción hombre-máquina de forma que facilite al usuario la comprensión de cómo se producirá tal interacción; un prototipo que funcione e implemente algunos subconjuntos de la función requerida de software deseado; o un programa existente que ejecute parte de o toda la función deseada, pero que tenga otras características que deban ser mejoradas en el nuevo trabajo de desarrollo.

La secuencia de sucesos para la construcción de prototipos se muestra en la figura 2.5.2. La construcción de prototipos comienza con la recolección de los requerimientos. El técnico y cliente se reúnen y definen los objetivos globales para el software, identifican todos los requerimientos conocidos y perfilan las áreas en donde será necesario una mayor definición. Luego se produce un "diseño rápido". El diseño rápido se enfoca sobre la representación de los aspectos del software, visibles al usuario (por ejemplo, métodos de entrada y formatos de salida). El diseño rápido conduce a la construcción de un prototipo. El prototipo es evaluado por el cliente/usuario y se utiliza para refinar los requerimientos del software a desarrollar. Se produce un proceso interactivo en el que el prototipo es "afinado" para que satisfaga las necesidades del cliente, al mismo tiempo que facilita al que los desarrolla una mejor comprensión de lo que hay que hacer. Idealmente, el prototipo sirve como un mecanismo para identificar los requerimientos del software.

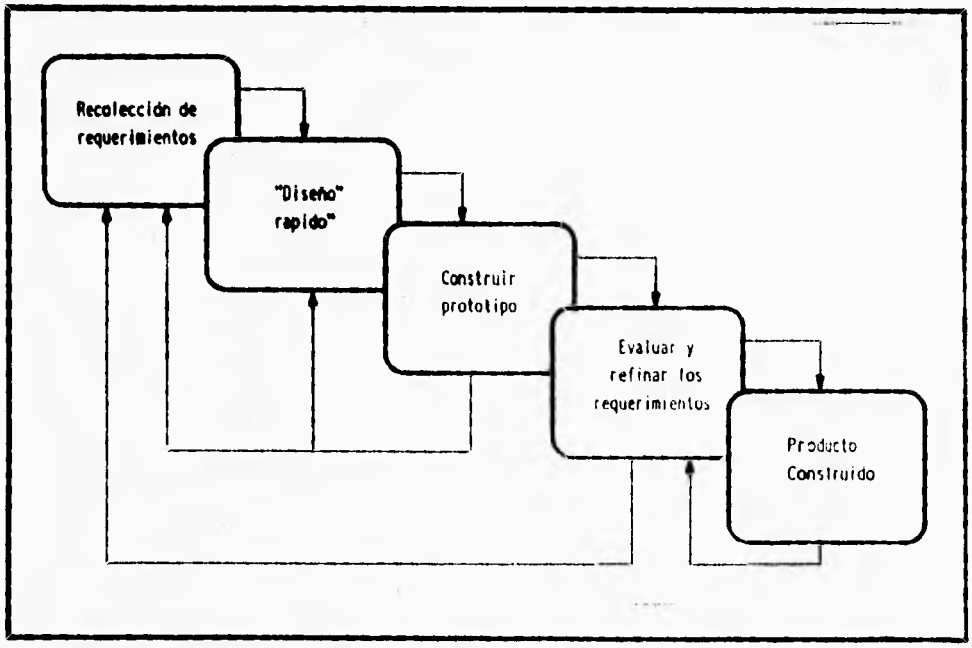


Fig. 2.5.2. Construcción de prototipos.

TECNICAS DE LA CUARTA GENERACION. El término técnicas de la cuarta generación (T4G) abarca un amplio espectro de herramientas de software que tienen una cosa en común: todas facilitan al que desarrolla el software especificar algunas características del software a alto nivel.

Actualmente, un entorno para el desarrollo del software que soporte T4G incluye algunas o todas las siguientes herramientas: lenguajes de no procedimientos para consulta a bases de datos, generación de informes, manipulación de datos, interacción y definición de pantallas y generación de código; capacidades gráficas de alto nivel; y capacidad de hoja de calculo.

La figura 2.5.3, describe los pasos de las T4G. Las T4G comienzan con el paso de recolección de requerimientos. Idealmente, el cliente debe describir los requerimientos y éstos deben traducirse directamente en un prototipo operacional; el diálogo cliente-técnico permanece como una parte esencial del enfoque T4G.

Para aplicaciones pequeñas, puede ser posible ir directamente desde el paso de establecimiento de los requerimientos a la implementación, usando un lenguaje de la cuarta generación de no procedimientos (LG4).

La implementación usando L4G facilita al que desarrolla el software, la descripción de los resultados deseados, los cuales se traducen automáticamente en código fuente para producir dichos resultados. Obviamente, debe existir una estructura de datos con información relevante y debe estar rápidamente accesible al L4G.

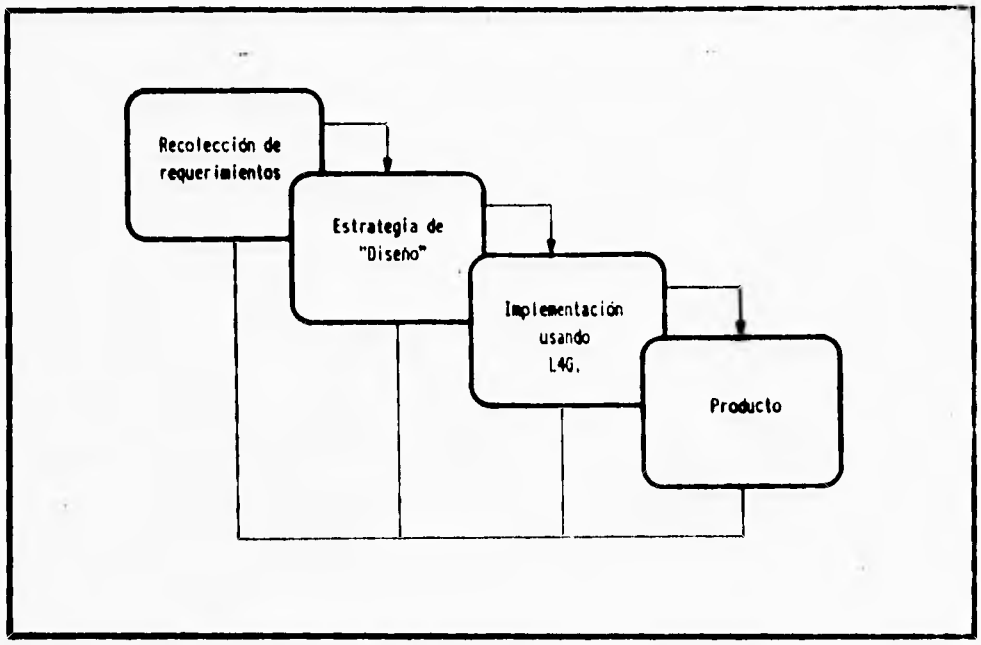


Fig.2.5.3. Técnicas de la cuarta generación.

El último paso de la figura 2.5.3 contiene la palabra "producto". Para transformar una implementación T4G en un producto, el que lo desarrolla debe dirigir una prueba completa, desarrollar una documentación con sentido para facilitar que el mantenimiento pueda ser ejecuta de una forma expedita.

Después de haber repasado los tipos de sistemas y técnicas existentes, no cabe duda que nuestro sistema a desarrollar caiga en la categoría de un "Sistema de Gestión", por lo que solo tendremos que seleccionar con que herramienta(s) habremos de implementarlo. Las técnicas pueden y deben combinarse de forma que puedan utilizarse las ventajas de cada una en un proyecto. Haremos uso entonces de las ventajas del ciclo de vida clásico, la contrucción de prototipos y las técnicas de cuarta generación (para el manejo de pantallas y capacidades gráficas de alto nivel).

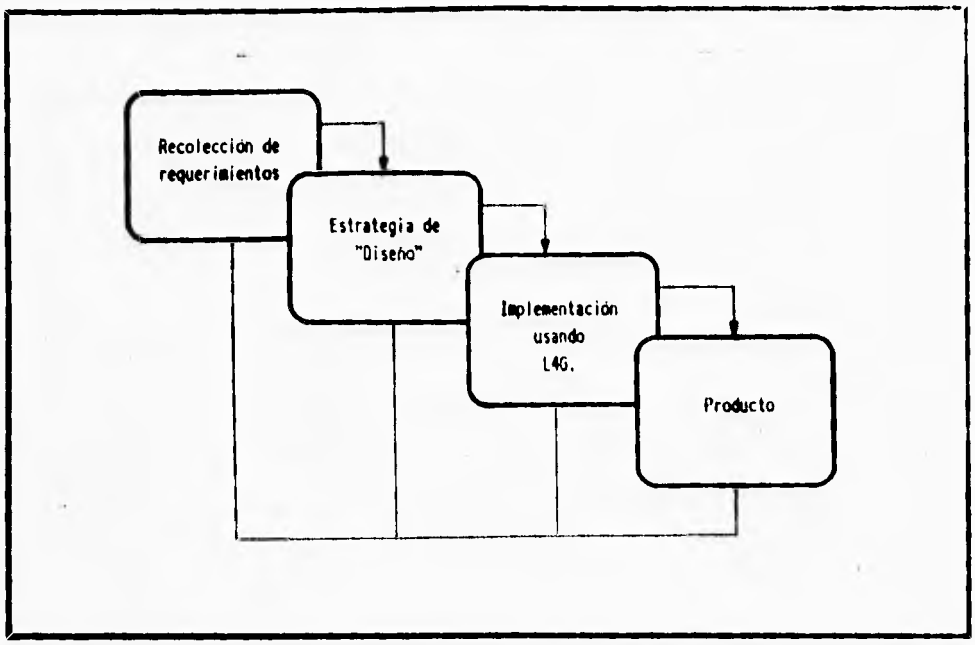


Fig.2.5.3. Técnicas de la cuarta generación.

El último paso de la figura 2.5.3 contiene la palabra "producto". Para transformar una implementación T4G en un producto, el que lo desarrolla debe dirigir una prueba completa, desarrollar una documentación con sentido para facilitar que el mantenimiento pueda ser ejecuta de una forma expedita.

Después de haber repasado los tipos de sistemas y técnicas existentes, no cabe duda que nuestro sistema a desarrollar caiga en la categoria de un "Sistema de Gestión", por lo que solo tendremos que seleccionar con que herramienta(s) habremos de implementarlo. Las técnicas pueden y deben combinarse de forma que puedan utilizarse las ventajas de cada una en un proyecto. Haremos uso entonces de las ventajas del ciclo de vida clásico, la contrucción de prototipos y las técnicas de cuarta generación (para el manejo de pantallas y capacidades gráficas de alto nivel).

Dadas las pruebas de Benchmark podemos concluir que el manejador de bases de datos que utilizaremos será Paradox, debido a que podemos emplear las herramientas de esté, sobre Programación Orientada a Objetos, indispensable para el desarrollo de nuestro sistema en cuestión. Dichas pruebas muestran las gráficas en las cuales se midieron los puntos importantes de cada manejador de bases de datos y podemos observar en ellas que tanto Paradox para DOS, como Paradox para Windows destacan entre los primeros manejadores, que no son los mejores como las dos versiones de FoxPro, pero que tienen las ventajas de la Programación Orientada a Objetos

CAPITULO III

DESARROLLO DEL SISTEMA

3.1 Diseño

3.1.1 ESPECIFICACIONES DEL DISEÑO.

Las especificaciones de diseño describen las características del sistema, los componentes o elementos del sistema y la forma en que éstos aparecerán ante los usuarios.

El diseño conceptual es un proceso donde convergen tres dimensiones en la definición de un sistema integral. Estas dimensiones son:

- **Los procesos**: la interacción dinámica de los elementos que forman el sistema.
- **La información**: la descripción del ambiente (mundo) donde operará el sistema.
- **Los datos**: las estructuras computacionales y físicas donde se almacenará la información del sistema.

El diseño conceptual utiliza los diagramas de flujo de datos, los diagramas de entidad relación y la definición de la Base de Datos en sus diferentes profundidades y alcances.

El sistema de bases de datos se compone de varios elementos que funcionan al unísono, a fin de proporcionar la información de los datos almacenados. El usuario del sistema ejecuta comandos en una base para producir esa información. Antes que el ciclo

empiece, se debe diseñar la base de datos de manera que llene todos los requisitos de información de los usuarios. Los siguientes son los elementos del sistema:

- **Metodología** de la base de datos.
- **Modelo** de la base de datos.
- **Interfaz del** usuario.
- **Sistema de** manejo de la base de datos.
- **Producto de la base de datos.**

Metodología de la base de datos.- El núcleo del sistema de una base de datos es su **metodología, donde se definen los objetivos y metas del sistema y donde la administración acepta el enfoque de esta. De la metodología surge el producto de la base de datos. Una buena base de datos necesita la participación del usuario. Este es el responsable en última instancia de la percepción y uso comunes del producto de base de datos. Se deben establecer objetivos en las siguientes áreas:**

- **Integridad de la base de datos.**
- **Eficacia de las operaciones de la base de datos.**
- **Satisfacción total de las necesidades del usuario.**
- **Control sobre el acceso a la base de datos.**

La metodología, es la forma en que se hacen las cosas, se presenta primero como un programa que muestra todas las tareas y actividades, tanto administrativas como técnicas, necesarias para un producto. Esta serie de procedimientos ayudará a desarrollar el producto. Al mismo tiempo que se planea la metodología, se deberán establecer políticas para la definición, administración, mantenimiento y uso de la base de datos.

Modelo de la base de datos.- El modelo está diseñado para representar las posiciones y manejar las transacciones de la organización. El modelo, como un talonario de cheques, permite a los usuarios representar los datos, en las diferentes formas que describan el mejor estado del producto. En un talonario se registra el importe de los cheques que se emiten y su concepto; se mantiene el balance actualizado restando la cantidad de cada cheque al saldo total y se suma a la de cada depósito. Esta cifra no necesariamente representa el balance real, tal vez porque se olvidó anotar un cheque o reducir un cargo bancario. Se confía en que el banco llevará el saldo real, porque tiene el dinero y la prueba en documentos. Sin embargo, el saldo del banco simplemente es una representación; los empleados del banco no toman el efectivo de la cuenta y lo transfieren a otra. Para asegurar que todos los saldos representen correctamente las cuentas se debe contar con medidas de control, se concilia la representación del estado de cuenta comprobandolo con el del banco. Un talonario es un sistema de manejo de bases de datos. Se registran las transacciones en él, se actualizan y se consultan. El talonario representa tanto las transacciones como el estado del flujo de efectivo.

Interfaz del usuario.- La interfaz está diseñada para ayudar al usuario. Debe responder a sus necesidades, dar manejos adecuados, recuperar errores, ser confiable, útil y amable con él. De hecho, la interfaz debe permitir que el usuario opere clara y fácilmente la base de datos, con comandos y operaciones.

El usuario interactúa con la computadora a través de la interfaz. Esta emplea la pantalla y el teclado, de modo que el usuario pueda introducir, validar y procesar los datos, así como recuperar la información de la base de datos. Es ahí en donde se empieza a trabajar con la base de datos.

Sistema de manejo de base de datos.- Las funciones y comandos de operación manipulan la base de datos mediante una serie de utilerías que trabajan junto con el sistema operativo, el software y el hardware. Estas utilerías son programas internos que efectúan varias tareas en la base de datos. El grupo de utilerías se conoce como sistema de manejo de base de datos. Traduce las necesidades del usuario en operaciones sobre los datos almacenados. Su función es asegurar la integridad y la seguridad de esta.

El sistema proporciona los mecanismos mediante los que se efectuan las operaciones que el usuario solicita a la base de datos. El sistema permite que varios usuarios compartan la misma base.

El sistema incluye los siguientes procesos para la manipulación del archivo y los registros:

- Acceso al registro.
- Apertura de archivos.
- Autorización.
- Validación y expresión.

El acceso recupera los registros por procesar. El mecanismo de apertura de archivos abre los archivos en diferentes modos. La autorización establece la protección del archivo y campo mediante el citado. Los mecanismos de validación y expresión validan los datos que el usuario teclea y calculan las expreiones de los valores de actualización y programas de control.

Producto de bases de datos.- La base de datos se crea de acuerdo a la base modelo explícita y contiene los siguientes elementos:

- Una estructura detallada.
- Las aplicaciones y operaciones.
- Las normas de validación de datos.
- La relación entre archivos.

En primer lugar, se diseña la base de datos de acuerdo a los principios respectivos del diseño. En segundo lugar, se crea un conjunto compartido de datos. Se debe definir perfectamente los valores de los datos en la base, es decir, los que compartirán los usuarios. Para que el sistema funcione, todos los usuarios deben tener una percepción común del producto de la base de datos. En tercer lugar, los datos están interrelacionados, validados y se describen por sí solos. La base se encuentra interrelacionada porque es un conjunto de archivos, de relaciones entre dichos archivos, entre campos dentro de los archivos y relaciones de la organización, y no se trata únicamante de un conjunto de archivos. En cuarto lugar la base de datos se ve sujeta a muchos procesos que efectúa el sistema de manejo de la base en ellos. El sistema almacena, recupera y manipula los datos que se encuentran en la base de datos. Quinto, la base de datos satisface las necesidades de información de los usuarios finales.

Tomando en cuenta que el sistema será realizado en un lenguaje de programación visual, en donde la interface con el usuario será gráfica, entonces estamos hablando de que el sistema será manejado bajo ambiente Windows. Así, el usuario podrá obtener información turística del puerto de Ixtapa-Zihuatanejo, mediente el uso de iconos y de ventanas.

Para elaborar el presente sistema nos basaremos en las características de la Programación Orientada a Objetos, basándonos principalmente en los conceptos de Abstracción, (el cual hace más simple la escritura de programas grandes), la encapsulación

(facilita el mantenimiento de un programa) y finalmente el concepto de jerarquías de clases, (que es una herramienta de clasificación muy poderosa que hace que un programa sea extensible fácilmente). Una vez definido lo anterior, el trabajo de diseño consistirá primeramente en:

1.- Hacer un análisis del problema planteado, definiendo las variables de entrada y de salida de información.

2.- Identificar las clases que el sistema necesita. Las clases deben estar basadas en las actividades centrales del sistema.

3.- Una vez que se identificaron las clases, la siguiente tarea es determinar que responsabilidades presentan, a esto se le llama "Asignación de atributos y comportamientos". Las responsabilidades de una clase caen dentro de dos categorías:

- **La información que un objeto de esa clase debe contener.**
- **Las operaciones que un objeto puede ejecutar o que pueden ser ejecutados. (¿Qué puede hacer este objeto?)**

4.- Encontrar relaciones entre las clases, para poder identificar tales relaciones se debe de considerar como una clase ejecuta el comportamiento que le ha sido asignado.

5.- Organizar las clases en jerarquías. Esto es una extensión del segundo paso, pero requiere la información ganada del tercer y cuarto paso. El proceso de asignar atributos y comportamientos a las clases logra que se tenga una mejor idea de sus similitudes y de sus diferencias; al identificar las relaciones entre clases se observa que clases necesitan ser incorporadas a la funcionalidad de otras.

6.- Recabar y capturar la información turística correspondiente de las bases de datos que el sistema manejará. Las bases de datos contendrán información de todos los servicios que el puerto ofrece, por ejemplo el nombre de aeropuertos, central de autobuses, hoteles, hospitales, restaurantes, discotecas, delegaciones, bancos, aduanas, casas de cambio, escuelas, capitanías, etc.; junto con toda la información concerniente a tales sitios, como su dirección, precio, categoría y teléfono.

También se manejarán bases de datos de imágenes; especialmente en los hoteles, pensando en ayudar al turista en hacer una elección del hotel en donde se desee hospedar.

7.- Una vez hecho lo anterior, se dispondrá a digitalizar los mapas de las ciudades de Ixtapa y Zihuatanejo; esto se hará mediante una tableta digitalizadora, con la cual se obtendrán, una serie de puntos que se manejarán como coordenadas "*x, y*" que serán almacenadas en un arreglo, con un formato "autocad", y de ahí se hará la conversión de formato a archivos PCX para que el lenguaje lo pueda manejar.

8.- Después se realizarán el diseño de las pantallas de consulta y de información del sistema, tales como la pantalla de presentación y las de los menús junto con todas las cajas de diálogo y edición que el sistema manejará.

9.- Entonces se procederá a realizar los módulos de programación para que se puedan relacionar las pantallas que contienen los mapas de las ciudades, con las diferentes bases de datos y entre ellas.

10.- Una vez hecho lo anterior se elaborará el código de programación que se encargará de ligar los módulos de programación anteriores.

11.- Se elaborará en el sistema una opción de ayuda del usuario, en donde se describirán los pasos que se requieren para ejecutar el sistema.

3.1.2 DIAGRAMA DE FLUJO DE DATOS

La estrategia de flujo de datos muestra el empleo de éstos en forma gráfica. Las herramientas utilizadas al seguir esta estrategia muestran todas las características esenciales del sistema y la forma en que se ajustan entre sí. Puede ser difícil comprender en su totalidad un proceso del sistema si se emplea para ello sólo una descripción verbal; las herramientas para el flujo de datos ayudan a ilustrar los componentes esenciales de un sistema junto con sus interacciones.

Un diagrama de flujo de datos se emplea para describir y analizar el movimiento de datos a través de un sistema, y se puede generar de forma manual a automatizada, incluyendo procesos, lugares para almacenar datos y retrasos en el sistema. Los diagramas de flujo de datos son la herramienta más importantes y la base sobre la cual se desarrollan otros componentes. La transformación de datos de entrada y salida por medio de procesos puede describirse en forma lógica e independiente de los componentes físicos asociados al sistema.

El diagrama de flujo de nuestro sistema se muestra en la figura 3.1.2.1 el cual consta de 4 grandes procesos:

1. Proceso de Ubicación.
2. Proceso de Información Turística.
3. Proceso de Quejas.
4. Proceso de Ayuda.

Proceso de Ubicación

En este proceso los usuarios seleccionan el menú de ubicación su opción (Ixtapa o Zihuatanejo). Ubicación necesita esta información para poder obtener de la base de mapas, las coordenadas de la ciudad elegida, para procesar el mapa y desplegar en pantalla el mapa y mandar la ubicación al proceso de información turística.

Proceso de Información Turística

Una vez recibidos los datos de ubicación el proceso de Información Turística recibe por parte del usuario la opción de la cual necesita información; con esto el proceso obtiene de la base de información turística la información del lugar y de la base de imágenes las gráficas disponibles de los bienes y servicios del lugar y las despliega en pantalla..

Proceso de Quejas

Aquí el usuario manda en un archivo tipo texto su queja, la cual será almacenada en la base de almacenamiento de quejas.

Proceso de Ayuda

En todo momento el usuario del sistema podrá solicitar ayuda del funcionamiento del sistema mandando su pregunta y el proceso de Ayuda se encargará de desplegar el tema solicitado.

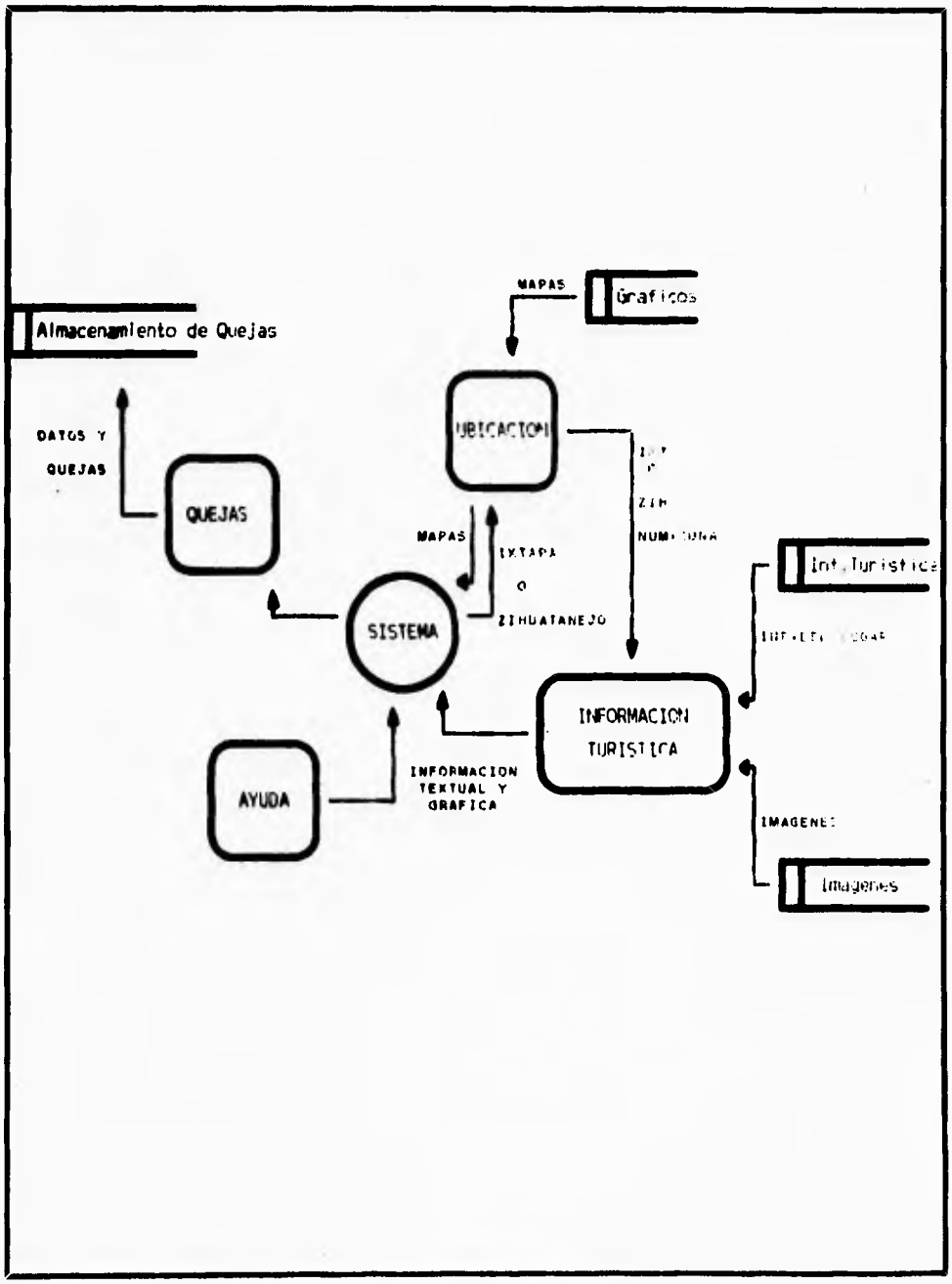


Fig. 3.1.2.1 Diagrama de Flujo de Datos.

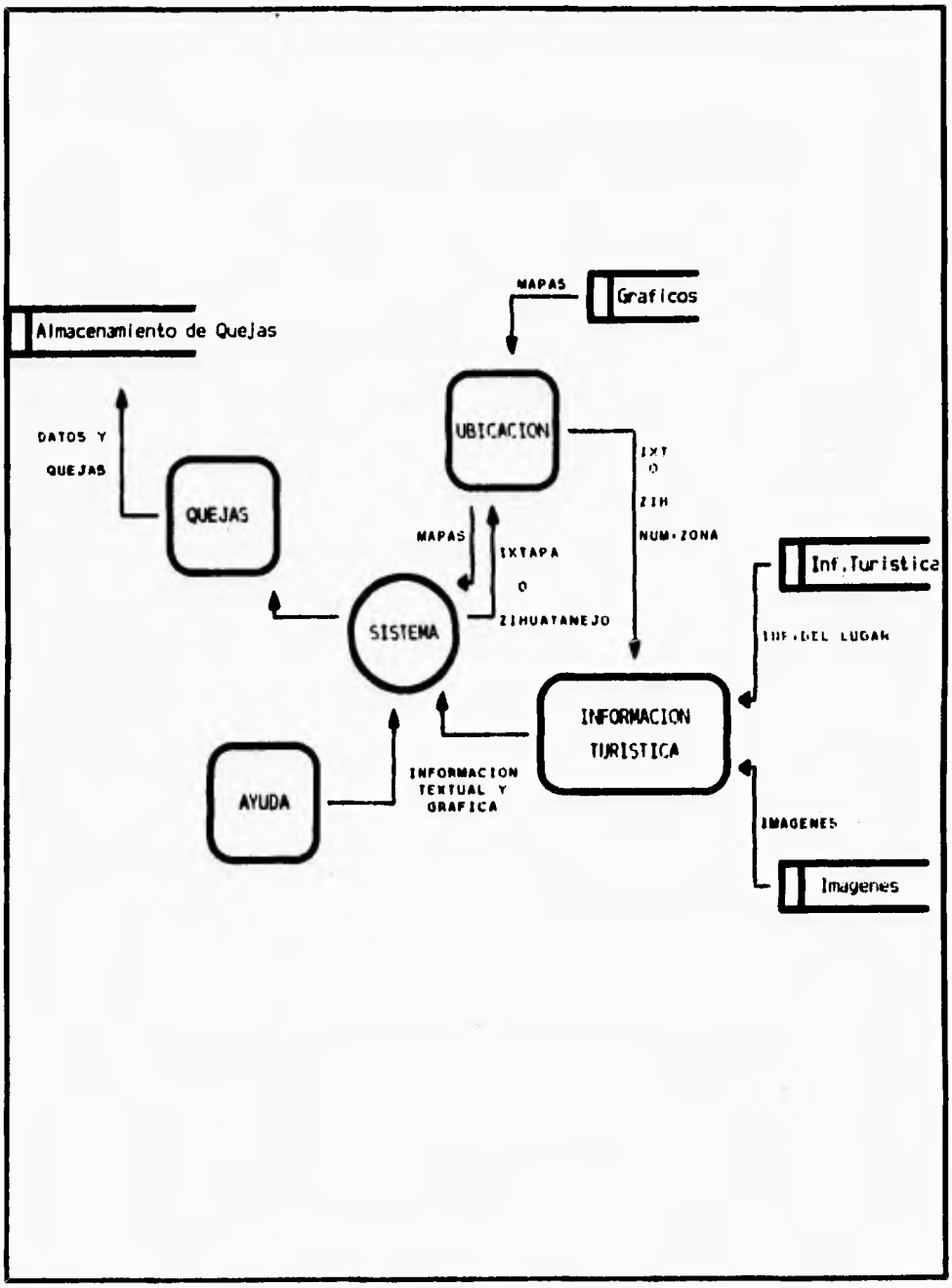


Fig. 3.1.2.1 Diagrama de Flujo de Datos.

3.1.3 DISEÑO FISICO Y CONSTRUCCIÓN DE LA BASE DE DATOS.

En cuanto al diseño físico del sistema podemos decir que este estara integrado por una serie de imagenes; que representaran las pantallas del mismo y figuras tales como mapas de los lugares en estudio, en este caso de Ixtapa-Zihuatanejo. Para tales imágenes se describirá brevemente cuales son los tipos de formatos de imágenes que existen y las características de estos para el manejo de la información. La elección de el formato de las imágenes se integrará finalmente al sistema.

Para la construcción de la base de datos se tomarán en cuenta: como principal requisito, los requerimientos del usuario, fundamentales para la elaboración del diseño del sistema y posteriormente para el desarrollo del mismo. La recopilación y clasificación de la información. El manejo de las bases de datos se realizará mediante el DBMS (Data Base Manager System), Manejador de Base de Datos de Paradox para DOS; el cual nos ayudará a crear y generar las bases de datos, realizar las relaciones entre ellas, generar los indices necesarios y de esta forma conjuntar las imágenes y la información almacenada en bases de datos para desarrollar el sistema.

DESCRIPCION DE LOS FORMATOS "TIFF", "GIF" Y "PCX".

FORMATO TIFF (TAG IMAGE FILE FORMAT).

TIFF es un formato que fue diseñado para promover el intercambio de datos de imágenes digitales. Los campos fueron definidos primeramente pensando en "desktop publishing" y aplicaciones relacionadas.

El escenario general para el cual TIFF fue inventado asume que el software de aplicación para "scanear" o "pintar" crea un archivo TIFF, el cual puede ser leído e incorporado dentro de un documento o publicación tal como un paquete de "desktop publishing".

La intención de TIFF es organizar y codificar practicas existentes con respecto a la definición y uso de datos digitales. TIFF fue diseñado para ser un formato de intercambio muy extensible.

TIFF no es un lenguaje de impresora o lenguaje de descripción de páginas, tampoco es su propósito ser un standard general para el intercambio de documentos. Puede ser de utilidad para algunas aplicaciones de edición de imágenes. El objeto de diseño primario fue el proveer de un medio ambiente dentro del cual la permuta de datos e imágenes entre programas de aplicación pueda ser realizado. TIFF es por lo tanto planeado para ser un super conjunto de formatos de archivos de imágenes existentes para scanners de oficina (y programas de pintura y cualquier otro que produzca imágenes con pixeles).

TIFF esta propuesto para ser independiente de sistemas operativos especificos, sistemas, compiladores y procesadores. La única hipótesis significante es que el medio de almacenamiento soporte algo como un archivo, definido como una secuencia de bytes de 8-bits, donde los bytes son numerados de 0 a N. El archivo TIFF más largo es de 2**32 bytes. Dado que los apuntadores son usados libremente, un archivo TIFF es más fácilmente leído desde un dispositivo de acceso aleatorio, aunque es posible leer y escribir archivos TIFF en un medio secuencial como cintas magnéticas.

La extensión de archivos recomendada por MS-DOS para archivos TIFF es .TIF. El filetype recomendado por Macintosh es TIFF. Convenciones en otros ambientes computacionales no han sido aun establecidos.

ESTRUCTURA

En TIFF, campos individuales son identificados con una etiqueta única. Esto permite que campos particulares puedan estar presentes o ausentes en un archivo según sea requerido por la aplicación.

Algunos archivos TIFF tendrán solo unos pocos campos en ellos; otros tendrán varios. El software que cree archivos TIFF deberá escribir tantos campos como crea que serán necesarios y útiles (y no más). El software que lea archivos TIFF deberá hacer lo mejor que pueda con los campos que encuentre ahí.

Existen muchas formas en las cuales un esquema de formato de archivos orientados a etiquetas puede ser implementado. TIFF usa la siguiente aproximación:

Existen tres partes principales en un archivo TIFF. La primera es un pequeño encabezado de archivo imagen. El siguiente es un directorio de todos los campos que serán encontrados en este archivo. Finalmente, tenemos los datos para los campos.

Un archivo TIFF comienza con una pequeña cantidad de datos posicionales, conteniendo la siguiente información:

Bytes 0-1:

La primera palabra de el archivo sirve para especificar el orden de los bytes usados dentro del archivo. Los valores definidos actualmente son:

LL (hex 4949)

MM (hex 4D4D)

En el formato LL, el orden de los bytes es siempre de el menos significativo al más significativo, para enteros de 16 y 32 bits.

En el formato MM, el orden de los bytes es siempre de el más significativo al menos significativo, para enteros de 16 y 32 bits.

Bytes 2-3:

La segunda palabra de el archivo es el número de versión de TIFF. Este número no debe cambiar. La actual descripción se refiere a la versión 42, por lo cual 42 (2A en hex) debe estar almacenado en esta palabra.

Bytes 4-7:

Esta palabra contiene el offset (en bytes) del primer Directorio del Archivo de Imagen (Image File Directory = IFD). El directorio debe estar en cualquier localidad en el archivo después del encabezado. (El termino byte offset será siempre usado para referirse a una localidad con respecto al principio de el archivo. El primer byte en el archivo tiene un offset de 0).

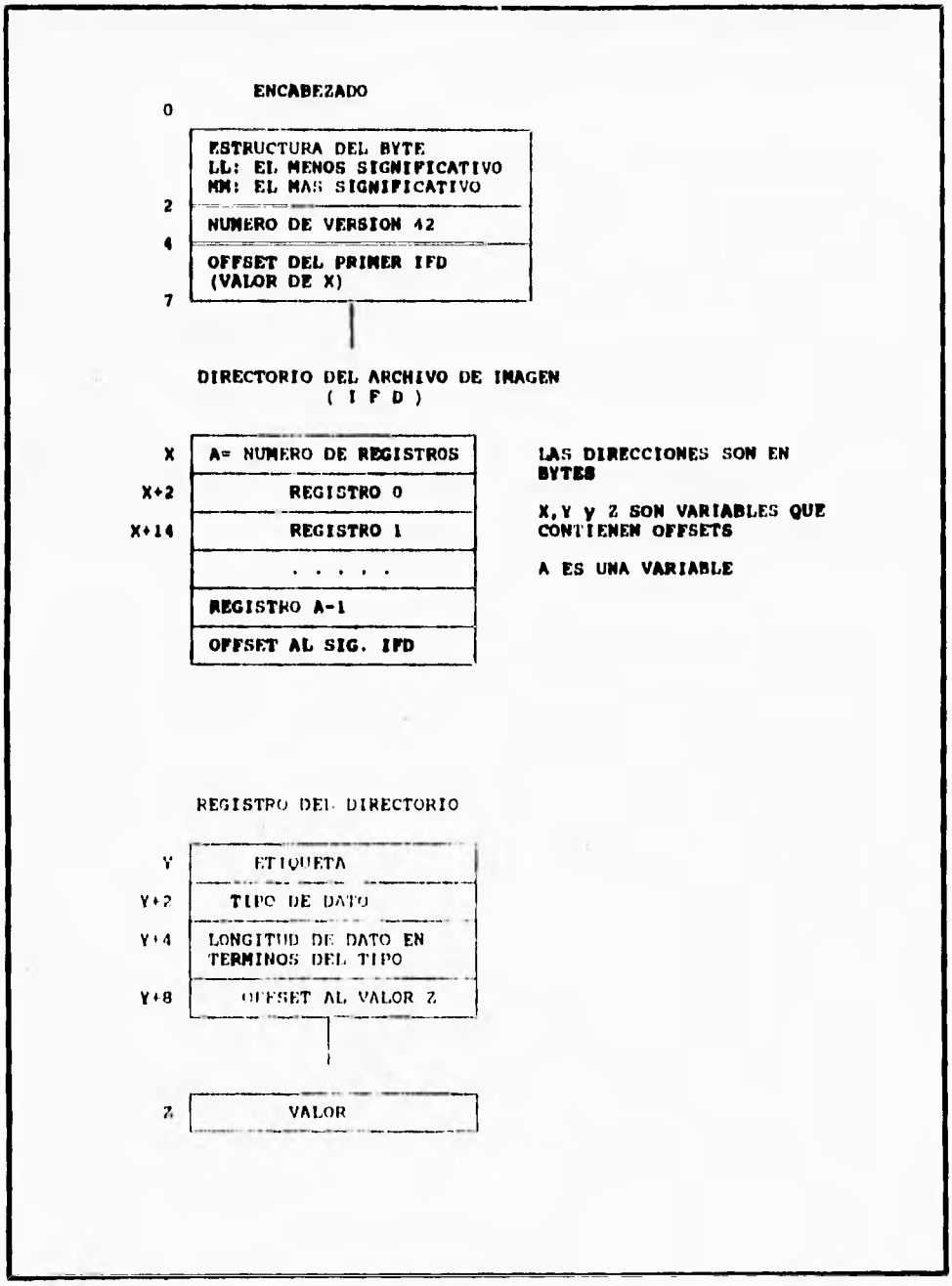


Tabla 3.1.3.1 Formatos tipo TIFF.

Un IFD consiste de 2 bytes para contar el número de entradas (por ejemplo el número de campos), seguidos de una secuencia de campos de 12 bytes, seguidos por un offset de 4 bytes de el siguiente IFD (o 0 si no hay). Cada campo tiene el siguiente formato:

Los bytes 0-1 contienen la Etiqueta para el campo. Los bytes 2-3 contienen el Tipo de el campo. Los bytes 4-7 contienen la longitud de el campo. Los bytes 8-11 contienen el offset (en bytes) del valor para el campo.

Las entradas en un IFD deben estar clasificadas en orden ascendente por etiqueta. Los valores para los cuales las entradas en el directorio apuntan no necesitan estar en algún orden particular en el archivo.

La parte de Longitud es especificada en términos de el tipo de dato. Una palabra simple de 16 bits (SHORT) tiene una longitud de 1, y no de 2, por ejemplo. Los tipos de datos y su longitud se describen a continuación:

1=BYTE	entero de 8 bits sin signo
2=ASCII	byte de 8 bits que almacenan códigos ASCII
3=SHORT	entero de 16 bits (2 bytes) sin signo
4=LONG	entero de 32 bits (4 bytes) sin signo
5=RATIONAL	2 LONGs

Note que puede haber más de un IFD. Cada IFD define un subarchivo. Un uso potencial de archivos subsecuentes es el describir una sub-imágen que esta de algún modo relacionada con la imágen principal. Otro uso es el representar múltiples páginas de imágenes, por ejemplo, un documento de facsímil requiere más de una página.

FORMATO GIF (GRAPHICS INTERCHANGE FORMAT)

GIF es un estandard de CompuServe para la definición de imágenes a color. Este "formato de intercambio gráfico" permite, alta calidad, alta resolución gráfica para ser desplegada en una variedad de hardware gráfico y esta propuesto como un mecanismo de despliegue e intercambio de imágenes gráficas.

FORMATO GENERAL DE UN ARCHIVO

GIF SIGNATURE (FIRMA GIF):

Identifica a los datos como una imagen GIF valida. Consiste de los siguientes seis caracteres :

G I F 8 7 a

Los últimos tres caracteres "87a" pueden ser vistos como un número de versión para una definición particular de GIF.

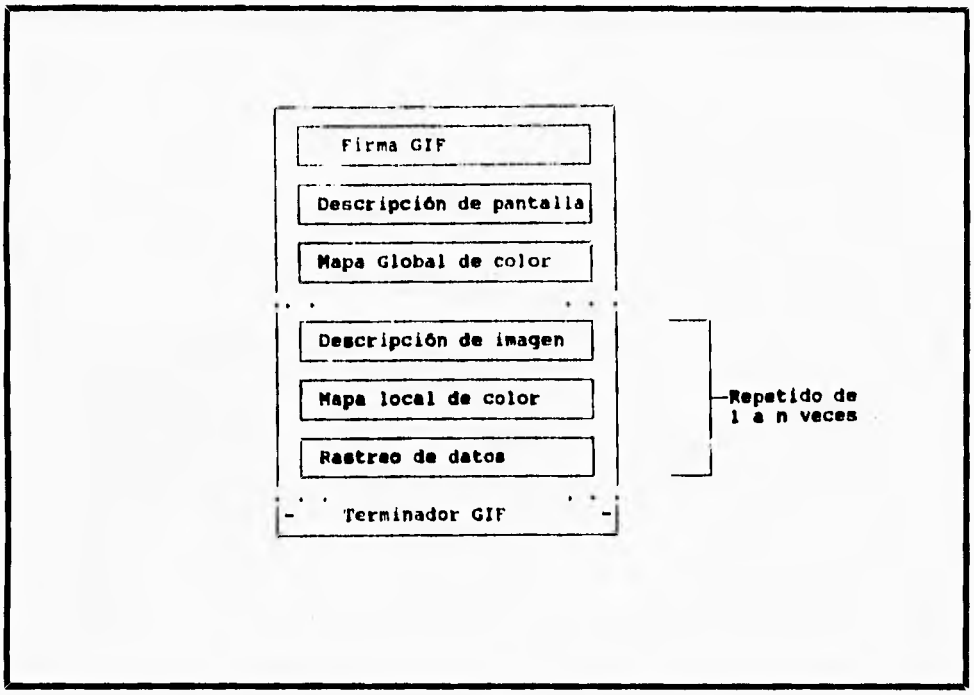


Tabla 3.1.3.2 Formato tipo GIF.

SCREEN DESCRIPTOR (DESCRIPTOR DE PANTALLA):

Describe el total de parámetros para las imágenes GIF. Define todas las dimensiones de un espacio de imágen ó la pantalla lógica requerida, la existencia de información de mapeo de colores, color del fondo de la pantalla, e información de la intensidad del color. Esta información es almacenada en una serie de bytes de 8-bits como sigue:

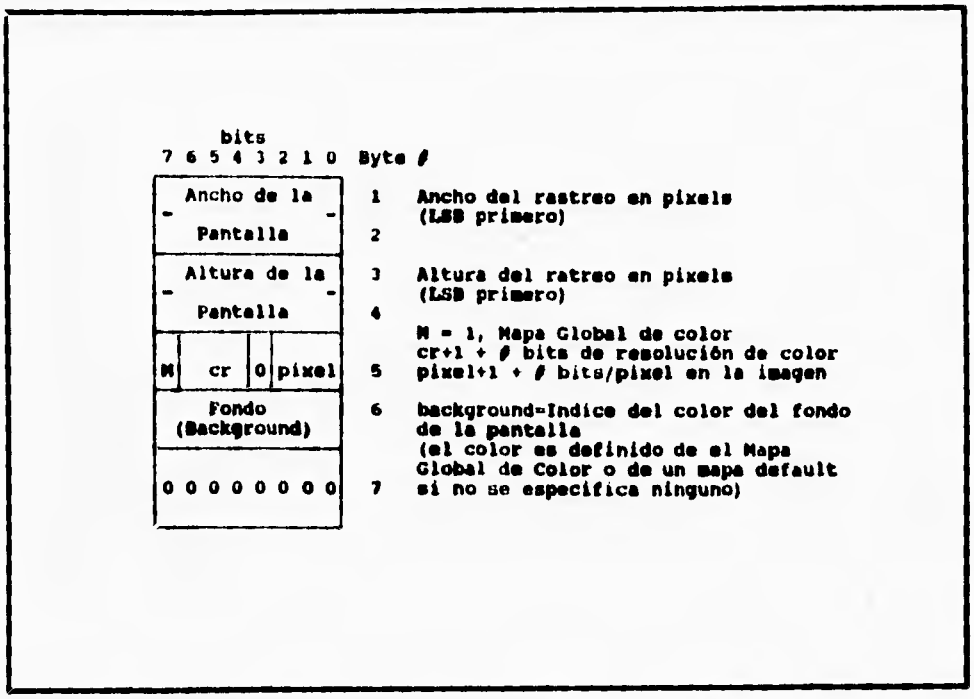


Tabla 3.1.3.3 Descriptor de pantalla.

El ancho y largo de la pantalla lógica pueden ser más largos que el despliegue físico. Como las imágenes más largas que el despliegue físico son manejadas es una implementación que depende y puede tomar ventaja de las características del hardware.

El valor de "pixel" también define el máximo número de colores dentro de una imágen. El rango de valores para "pixel" es de 0 a 7 lo cual representa 1 a b bits. Esto representa un rango de 2 (B & W) a 256 colores. El bit 3 de la palabra 5 esta reservada para futuras definiciones y debe ser cero.

GLOBAL COLOR MAP (MAPA GLOBAL DE COLOR):

Es opcional pero recomendada para imágenes donde la exactitud del color es deseada. La existencia de este mapa de color es indicada en el campo "M" del byte 5 del Descriptor de Pantalla. Un mapa de color puede también ser asociado con cada imágen en un archivo GIF tal como se describirá adelante. Sin embargo este mapa global será normalmente usado dadas las restricciones de hardware en los equipos disponibles hoy. En una imágen individual la bandera "M" será normalmente cero. Si el mapa global esta presente, su definición seguirá inmediatamente después del descriptor de pantalla. El número de entradas en el mapa de color es igual a 2**(bits por pixel), donde cada entrada consiste de tres valores que representan la relativa intensidad de rojo, verde y azul respectivamente. La estructura de el mapa de color es el siguiente :

bits 7 6 5 4 3 2 1 0	Byte #	
Intensidad rojo	1	Valor del rojo para el Indice 0
Intensidad verde	2	Valor del verde para el Indice 0
Intensidad azul	3	Valor del azul para el Indice 0
Intensidad rojo	4	Valor del rojo para el Indice 1
Intensidad verde	5	Valor del verde para el Indice 1
Intensidad azul	6	Valor del azul para el Indice 1
		(Continua para los colores restantes)
:	:	
:	:	

Tabla 3.1.3.4 Mapa Global de Color.

El blanco será representado como (255,255,255), el negro como (0,0,0) y el amarillo como (180,180,0).

bits 7 6 5 4 3 2 1 0	Byte #	
0 0 1 0 1 1 0 0	1	',' - Carácter separador de imagen
- Image Left -	2 3	Comienzo de la imagen en pixels desde la parte izquierda de la pantalla (LSB primero)
- Image Top -	4 5	Comienzo de la imagen en pixels desde el tope de la pantalla (LSB primero)
- Image Width -	6 7	Ancho de la imagen en pixels (LSB primero)
- Image Height-	8 9	Altura de la imagen en pixels (LSB primero)
M | I | 0 | 0 | 0 | pixel	10	

M=0 - Usa Mapa de color global, ignora 'pixel'
M=1 - Mapa de color local, usa 'pixel'
I=0 - Imagen formateada en orden Secuencial
I=1 - Imagen formateada en orden Intercalado
pixel+1 - # bits por pixel para esta imagen

Tabla 3.1.3.5 Descriptor de Imagen.

IMAGE DESCRIPTOR (DESCRIPTOR DE IMAGEN):

El descriptor de imagen define la colocación actual y la extensión de la siguiente imagen dentro de el espacio definido en el descriptor de pantalla. Cada descriptor de imagen es introducido por un carácter separador de imagen. El rol de el separador de

imagen es simplemente proveer con un carácter de sincronización para introducir un descriptor de imagen. Esto es deseable si el archivo GIF contiene más de una imagen. Este carácter es definido como 0x2C hex o ',' (coma). Cuando este carácter es encontrado entre imágenes, el descriptor de imagen seguirá inmediatamente.

LOCAL COLOR MAP (MAPA LOCAL DE COLOR):

Es opcional y se define aquí para uso futuro. Si el bit "M" de el byte 10 de el descriptor de imagen esta prendido, entonces el mapa de color sigue al Descriptor de imagen que aplica solo a la siguiente imagen, note que el campo 'pixel' de el byte 10 de el descriptor de imagen es usado solo si el mapa de color esta indicado.

RASTER DATA (RASTREO DE DATOS):

El formato de la actual imagen esta definido como una serie de valores de pixeles que crean la imagen. Los pixeles son almacenados de derecha a izquierda secuencialmente para un renglón de la imagen. El primer paso escribe cada 8 renglones, empezando con el renglón del tope de la imagen. El segundo paso escribe cada 8 renglones empezando en el quinto renglón desde el tope. El tercer paso escribe cada 4 renglones empezando en el tercer renglón desde el tope. El cuarto paso completa la imagen, escribiendo cada renglón, empezando en el segundo renglón desde el tope. Una descripción gráfica de este proceso es el siguiente:

Image Row	Pass 1	Pass 2	Pass 3	Pass 4	Result
0	**1a**				**1a**
1				**4a**	**4a**
2			**3a**		**3a**
3				**4b**	**4b**
4		**2a**			**2a**
5				**4c**	**4c**
6			**3b**		**3b**
7				**4d**	**4d**
8	**1b**				**1b**
9				**4e**	**4e**
10			**3c**		**3c**
11				**4f**	**4f**
12		**2b**			**2b**
. . .					

Tabla 3.1.3.6 Rastreo de datos.

GIF TERMINATOR (TERMINADOR DE GIF):

Para proveer una sincronización para la terminación de un archivo de imagen GIF, un decodificador GIF deberá procesar el final de el modo GIF cuando el carácter 0x·B hex o ';' sea encontrado después de que una imagen ha sido procesada. Por convención el software decodificador pausara y esperara una acción indicando que el usuario esta listo para continuar. Esta puede ser el introducir el retorno de carro en el teclado o un 'click' del mouse.

FORMATO PCX

Hace algunos años, ZSoft desarrollo el programa para PC Paintbrush como una respuesta al MacDraw de Macintosh. Si bien muchos programas de gráficos rivalizaron a Paintbrush en características y popularidad, su formato para archivos PCX ha trascendido al programa. Con cada versión sucesiva, el formato para archivos PCX se ha adaptado a los nuevos estandares en hardware de despliegue hasta tal punto que muchos programas DOS ahora usan archivos PCX como un medio estandard para intercambiar gráficas.

Los archivos PCX empiezan con un encabezado (header) de 128 byte. El encabezado tiene la siguiente estructura:

CAMPO	TIPO
pcxManufacturer	BYTE
pcxVersion	BYTE
pcxEncoding	BYTE
pcxBitsPixel	BYTE
pcxXmin, pcxYmin	INTEGER
pcxXmax, pcxYmax	INTEGER
pcxHres, pcxVres	INTEGER
pcxPalette[16]	PCXRGB
pcxReserved	BYTE
pcxPlanes	BYTE
pcxBytesLine	INTEGER
pcxPaletteInfo	INTEGER
pcxFiller[58]	BYTE

El campo pcxManufacturer es siempre \xA. Este es un "numero mágico" usado para identificar un archivo PCX. El campo pcxVersion identifica que versión de Paintbrush PC o que programa compatible creó este archivo. Los identificadores de las versiones se muestran en la Tabla 3.1.3.7:

ID	PC	Paintbrush Version
0	2.5	
2	2.8	With palette
3	2.8	Without palette
5	3.0	

Tabla 3.1.3.7 ID de Version PCX.

El campo pcxEncoding debe tener el valor de 1. El campo pcxBitsPerPixel nos indica cuantos bits consecutivos en el archivo representa un pixel en la pantalla, ver Tabla 3.1.3.8.

BITS/PIXEL	Color (Display Mode)
1	Monochorome, EGA/VGA 16 COLOR
2	CGA 4 COLOR
8	MCGA/VGA 256 COLOR

Tabla 3.1.3.8 Bits por Pixel.

Los siguientes cuatro campos (pcxXmin, pcxYmin, pcxXmax y pcxYmax) definen los limites de la imagen. Las dimensiones de la imagen serán pcxXmax - pcxXmin + 1 por pcxYmax - pcxYmin + 1. Los limites mas bajos de pcxXmin y pcxYmin son usualmente 0.

Los campos pcxHres y pcxVres especifican la resolución de la pantalla que será usada para crear el archivo. Estos campos pueden ser ignorados. El campo pcxPalette contiene el color de la paleta. El campo pcxReserved usualmente contiene el modo de video del BIOS pero no esta documentado como tal. Puede ser ignorado.

El campo pcxPlanes dice cuantos planos de color tiene la tarjeta de video. Por ejemplo, el empiezo de una imagen PCX en modo EGA 16-color aparece en la figura 3.1.3.1. La imagen tiene dos pixels azules, tres pixels verdes, dos pixeles rojos, y un pixel magenta (rojo + azul). La imagen tiene de ancho 640 pixels (80 bytes = 640 pixels / 8 bytes/pixel). El número actual de bytes en el archivo por scanline es probablemente menor que 80 bytes debido a la compresión.

El campo pcxBytesLine indica cuantos bytes existen en cada scanline. Este valor dependerá de el numero de bits por pixel y las dimensiones de la imágen. El campo pcxPaletteInfo es para imágenes VGA. El valor es 1 para escalas de gris y 2 para imágenes en color.

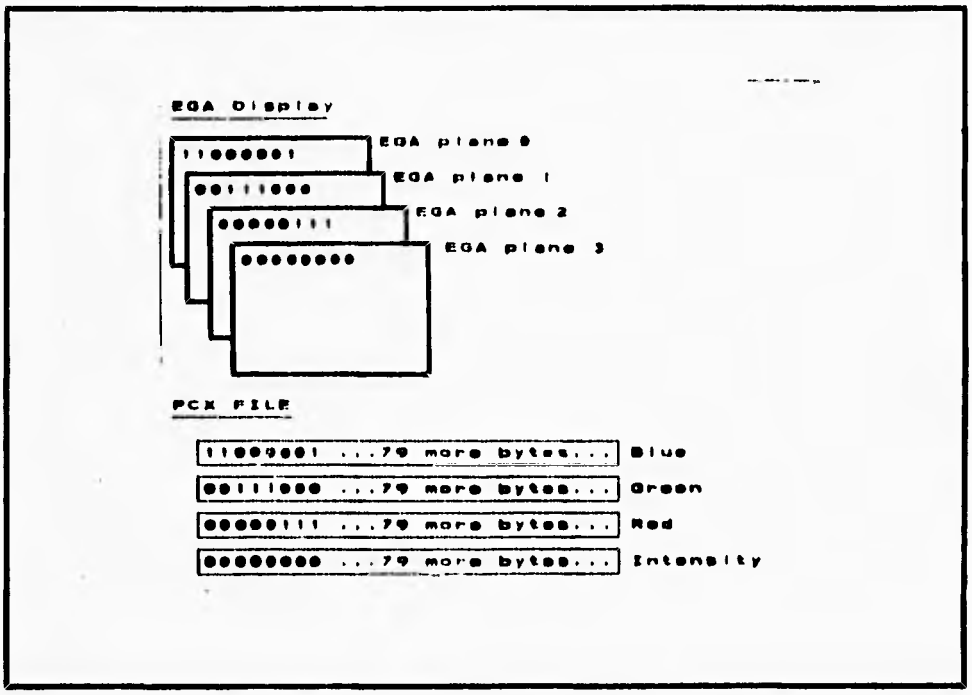


Fig. 3.1.3.1 Representación de una imagen PCX en modo EGA COLOR-16.

El resto de los 128-bytes del encabezado esta reservado para futuras adiciones a el formato **PCX**.

DISEÑO FISICO Y CONSTRUCCION DE LA BASE DE DATOS.

Se ha mencionado en capítulos anteriores, que el manejador de bases de datos (DBMS) que utilizaremos será Paradox para DOS; la elección fue hecha por ser el más apegado a estudios de confiabilidad en cuanto al manejador de bases, a la rápidez de respuesta para un número elevado de registros, a la relación entre las diferentes bases de datos y para fines de nuestro propio sistema.

El diseño físico se generará tomando en cuenta los datos recabados en la elaboración de cuestionarios, técnica utilizada para la obtención de la información en los dos puntos de interés para el desarrollo del sistema; Donde dicha información en forma general para los prestadores de bienes y servicios será :

Sus datos principales como : Nombre, Razón Social, Domicilio, Teléfono (en caso de existir este), Horario de trabajo, Registro Federal de Contribuyentes.

Posterior a estós se recabarán datos particulares dependiendo del tipo de Giro ó Razón Social, como : En el caso de hoteles, precios de habitaciones, categoría de estos, restaurantes dentro del hotel, atractivos turísticos. En cuanto a abarrotes y miscelaneas, los precios de productos de primera necesidad y los precios de productos controlados. Parecido a este tipo de encuestas estara el tipo de farmacias, pero en este caso se solicitará el precios de medicamentos. En cuanto a restaurantes, se clasificarán por el tipo de comida que ofrezcan como pueden ser: comida méxicana, americana, china, italiana, etc. y obteniendo los precios de sus menús.

Toda la información se capturará en diferentes bases de datos clasificandolas en clases distintas para Prestadores y Bienes de Servicio. Estas serán las dos grandes clasificaciones que utilizaremos y de las cuales surgirán las distintas clases y sus respectivos objetos para la manipulación de la información.

Otros puntos importantes que utilizaremos para la construcción de la base de datos serán : Las relaciones entre clases que resultan al generar los Diagramas de Flujo de Datos.

así como el tan necesario Diagrama de Entidad-Relación que nos ayudará a determinar los indices de relación entre clases, para poder accesar a una u otra.

Esto facilitará la construcción de no solo una sino varias bases de datos dependiendo del número de clases que generemos de la información obtenida y del número de prestadores de bienes y servicios.

También, otro dato relevante será la realización del Diccionario de Datos que contendrá cada uno de los campos que utilizaremos : el nombre del campo, el tipo de campo y la longitud de dichos campos. Con tal información se generarán los registros necesarios y las relaciones para cada una de nuestras bases.

No debemos olvidar que todo el manejo de la información en cuanto a bases de datos se refiere será manipulada por el manejador de bases de datos de Paradox para DOS. Y que el desarrollo del sistema a nivel de programas y ambiente visual será mediante Visual C++.

NORMALIZACIÓN DE LA BASE DE DATOS EN PARADOX.

Dentro del módulo de Información Turística, se considerá la información de los diferentes Prestadores de Bienes y Servicios.

Para llevar acabo la normalización utilizaremos los catálogos de giros, información general, productos y de precios.

Para la primera forma normal dentro de los catálogos del sistema y de esté modulo en particular tomaremos los Catálogos de Giros y el de Información General.

Así tomaremos los campos siguientes del catálogo de giros:

CAMPO	DESCRIPCIÓN
cve_giro	La clave que le designaremos a cada uno de los giros ya sea para Bienes o Servicios.
ngiro	Nombre de cada uno de los giros.
what	Campo que generamos para diferenciar un Bien o un Servicio, asignando la letra "B" o "S", respectivamente.

Para el catálogo de información general tomaremos los campos siguientes :

CAMPO	DESCRIPCION
cve_giro	Clave del giro que puede ser un Bien o Servicio.
nombre_com	Nombre comercial del giro.
categoria	Campo particular para el giro de Hoteles y especificar la categoría de éstos.
rfc	Registro Federal de Causantes.
colonia	Colonia del giro (establecimiento: Hotel, Restaurante, Abarrotes, etc.).
calle	Calle del establecimiento.
numero	Número del establecimiento.
c.p.	Código Postal del establecimiento.
tel	Teléfono del establecimiento.
horario	Horario en el cual trabaja el establecimiento.
Prin_Ser_I	Principal(es) servicio(s) que presta el establecimiento en particular.
Razon_soc	Razón Social del establecimiento.

folio	Número del folio para cada una de las encuestas, que corresponde a un establecimiento en particular.

La llave para relacionar a estos dos catálogos será la clave del giro, y la normalización quedaría de la siguiente forma :

Nombre del Giro	**Hoteles**
Bien o Servicio	**S**
Nombre Comercial	**Hotel Dorado Pacífico**
Categoría	**5 Estrellas**
R.F.C.	**OHD800408URO**
Colonia	**Ixtapa**
Calle	**Paseo Ixtapa**
Número	**---**
C.P.	**40880**
Teléfono	**32025**
Horario	**---**
Principal Servicio	**Servicios Complementarios 5 Estrellas**
Razón Social	**Operadora Hotelera Dorado Pacífico**

De esta forma, generamos un nuevo catálogo al cual denominaremos catálogo A. Con este catálogo A y el catálogo de productos generaremos la segunda forma normal.

Dentro del catálogo de productos tomaremos los siguientes campos :

CAMPO	DESCRIPCION
cve_giro	Clave del giro.
cve_produ	Clave del producto, que servirá para diferenciar a cada uno de éstos.
producto	Nombre del tipo de producto.
consec	Número consecutivo del tipo de producto.
marca_prod	Marca del producto.
presen	Presentación del producto.

El campo llave para estos catálogos será nuevamente la clave del giro. La normalización quedará de la siguiente forma :

Nombre del Giro	Hoteles
Bien o Servicio	S
Nombre Comercial	Hotel Dorado Pacífico
Clave del Producto	501
Producto	Habitación
Consecutivo	4
Marca del Producto	J. Suite
Presentación	---

A este catálogo lo denominaremos catálogo B.

Finalmente, para obtener la tercera forma normal utilizaremos el catálogo B y el catálogo de precios. Y los campos que utilizaremos de éste último serán :

CAMPO	DESCRIPCION
cve_giro	Clave del giro.
cve_produ	Clave del producto.
producto	Nombre del tipo de producto.
consec	Número consecutivo del tipo de producto.
marca_prod	Marca del producto.
presen	Presentación del producto.
precio	Precio del Producto.
folio	Número de folio para cada una de las encuestas que corresponde a un giro en particular.

El campo llave en este caso será la clave del giro y el folio, para relacionar ambos catálogos, obteniendo finalmente el tipo de prestador de Bien o Servicio, el nombre del giro, la información general del giro, los productos que maneja y el precio de cada uno de ellos de la siguiente manera :

Nombre del Giro	Hoteles
Bien o Servicio	S
Nombre Comercial	Hotel Dorado Pacífico
Razón Social	Operadora Hotelera Dorado Pacífico
Clave del Producto	501
Producto	Habitación
Consecutivo	4
Marca del Producto	J. Suite
Presentación	---

Precio	580.00
folio	55

Podemos concluir en esta etapa de la normalización que obtenemos información importante de cada una de los prestadores, pero además, de igual forma el precio de cada uno de los productos que se ofrecen al usuario.

El código de la normalización en Paradox para DOS se incluye en el Apéndice B.

3.1.4 DICCIONARIO DE DATOS.

Estructura de los Archivos I_GIRO.DBF y Z_GIRO.DBF

Estos archivos se encargan almacenar todos los bienes y servicios que el sistema ofrecerá al usuario, incluyendo una clave de giro que relacionará estos archivos con los archivos i_info.dbf y z_info.dbf.

Nombre	Nombre del campo	Tipo	Ancho
CLAVE GIRO	CVE_GIRO	N	3
GIRO	GIRO	C	30
TIPO DE_GIRO	TIPO-GIRO	C	1

Clave del giro	Nombre del Bien o Servicio	Tipo
1	Abarrotes	B
6	Farmacias	B
7	Labs. Clínicos	S
9	Electrico Automotríz	S

Estructura de los Archivos I_INFO.DBF y Z_INFO.DBF

Estos archivos se encargan almacenar las coordenadas de los puntos en donde se ubican los bienes y servicios que el sistema ofrecerá al usuario, incluyendo una clave de giro que relacionará estos archivos con los archivos i_giro.dbf y z_giro.dbf.

Nombre	Nombre del campo	Tipo	Ancho
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
CLAVE DEL GIRO	CVE-GIRO	N	2

Registro	X	Y	Clave del giro
10	12	50	1
20	20	48	6
30	25	47	7

Estructura de los Archivos I_DATOST.DBF y Z_DATOST.DBF

Estos archivos se encargan almacenar las coordenadas relacionandolas con los nombres de los bienes, servicios y atracciones turísticas que el sistema ofrecerá al usuario.

Nombre	Nombre del campo	Tipo	Ancho
COORDENADA X	X	N	4
COORDENADA Y	Y	N	4
NOMBRE DEL LUGAR	LUGAR	C	30

Campo X	Campo Y	Nombre del Lugar
36	12	Hotel Ixtapa
50	72	Casa de bolsa
Xn	Yn	Lugar n

Estructura de los Archivos I_LUGT.DBF y Z_LUGT.DBF

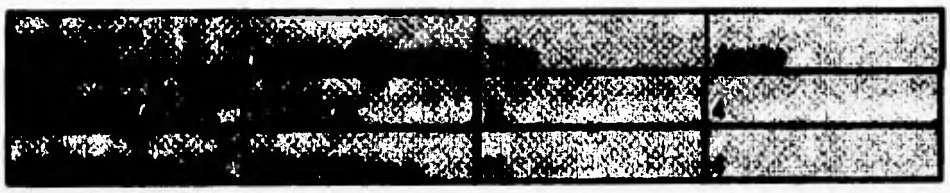
Estos archivos se encargan almacenar los nombres de las atracciones turísticas con los archivos de tipo TXT que almacenan la información cultural de dichas atracciones.

NOMBRE DEL LUGAR	LUGAR	C	20
NOMBRE DEL ARCHIVO	ARCHIVO	C	20

Nombre del lugar	Archivo en que se encuentra la información
Playa del Coral	ICORAL.TXT
Playa Cuachalalate	ICUACHA.TXT
Playa las Cuatas	ICUATAS.TXT

Archivos I_DATOS.DBF y Z_DATOS.DBF

Estos archivos almacenan las coordenadas(x,y) de los puntos que forman los mapas de las ciudades de Ixtapa y zihuatanejo.



Archivos I_SIT.DBF y Z_SIT.DBF

Estos archivos almacenan la información turística de las dos ciudades.

Nombre	Nombre del campo	Tipo	Ancho
Registro	Reg	N	5
Coordenada X	X	N	4
Coordenada Y	Y	N	4
Giro	Giro	N	2
Razón social	Razon_soc	C	40
Colonia	Colonia	C	40
Calle	Calle	C	40
Número	Numero	C	5
Código Postal	CP	C	5
Nombre Completo	Nombre_com	C	40
RFC	RFC	C	13
Propietario	Propiet	C	40
Teléfono	Tel	C	10
Horario 1	Horario1	C	10
Horario 2	Horario2	C	4
Horario 3	Horario3	C	4
Horario 4	Horario4	C	4
Principal servicio	Prin_serv	C	30
Categoria	Categoria	C	20

3.1.5 DIAGRAMA DE ENTIDAD-RELACIÓN

La estructura lógica de una base de datos puede expresarse gráficamente mediante un diagrama Entidad Relación. Una base de datos que se ajusta a un diagrama Entidad-Relación puede representarse por medio de una colección de tablas.

El esquema lógico (Diagrama Entidad-Relación) puede ser definido como el mapeo de entidad en la construcción proporcionada por el manejador de la base de datos. En general el esquema lógico indica como se almacenará y accesará el modelo.

El esquema lógico debe de hacerse de manera que lo único que se deje a los diseñadores de la base física sea la selección de los métodos de acceso y los índices secundarios. Hubband indica que se deben seguir las siguientes reglas durante el diseño físico :

Durante la fase de análisis se determinan las entidades mayores y sus relaciones. Estas entidades y sus relaciones se representan en modelos llamados Modelos de Entidad. El modelo es un diagrama representativo de la relación entre las clases de entidades.

La representación nos permite incluir solo aquellas entidades que se requieren para resolver un problema particular del procesamiento de datos. El modelo de entidad es esencialmente una vista del mundo real de los datos de la organización en términos de entidades, atributos y relaciones.

En nuestro sistema nuestro Diagrama Entidad - Relación estará compuesto por las siguientes entidades:

UBICACION

DATOS

SELECCION

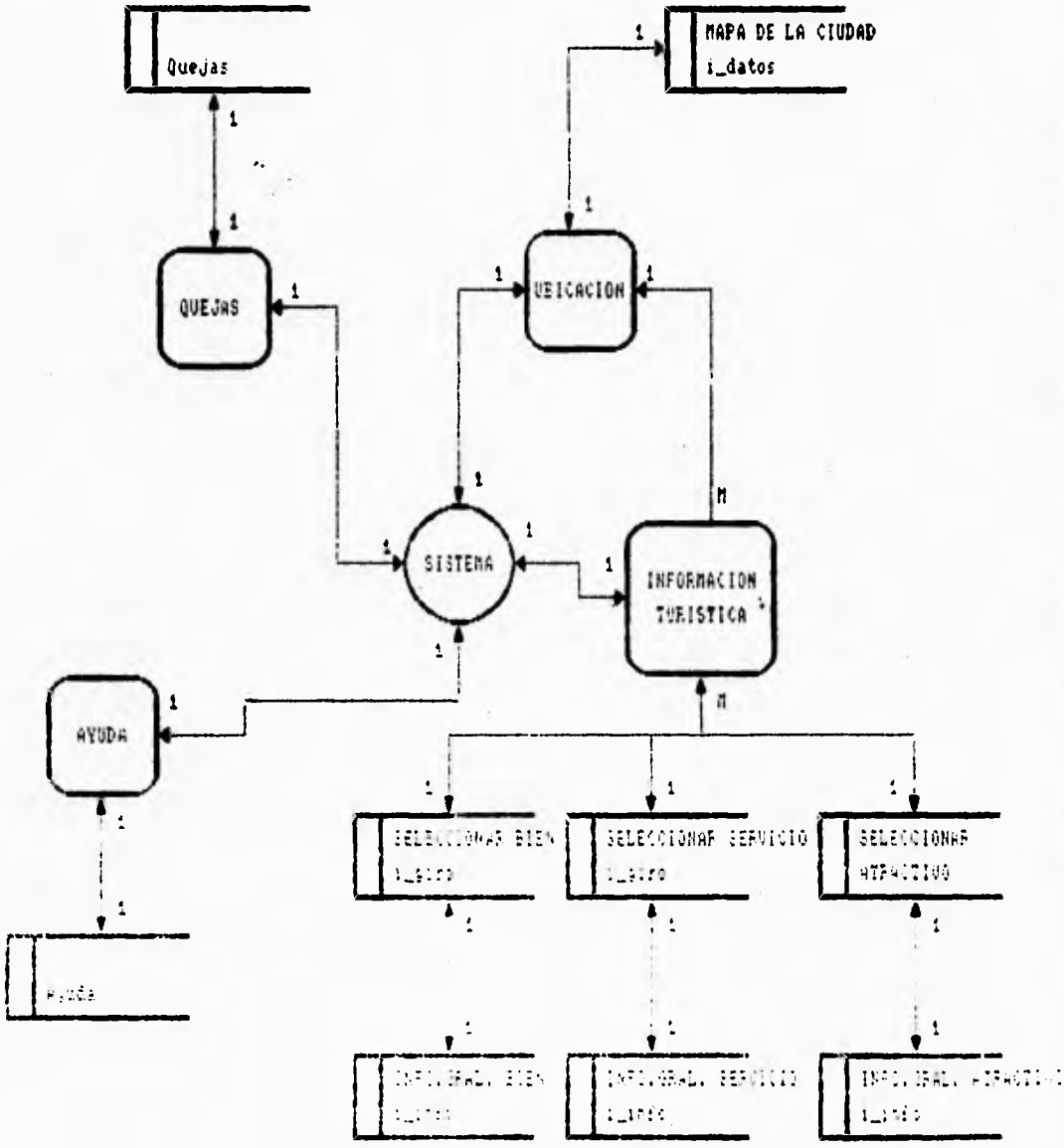
INF_TURISTICA

QUEJAS

AYUDA

Estas entidades a su vez se relacionarán entre sí (algunas no todas).

DIAGRAMA ENTIDAD - RELACION



3.1.6 DISEÑO E IMPLEMENTACIÓN DE LOS DIVERSOS MÓDULOS DE PROGRAMACIÓN.

El desarrollo de este sistema esta basado en los fundamentos de la programación orientada a objetos, es decir; como podemos representar mediante **"objetos"** (figuras o dibujos) diversas acciones o funciones que realiza un programa. Con el propósito de minimizar un poco el uso de la comunicación escrita, en este capítulo se ilustra, como el lenguaje Visual C++ utiliza estos conceptos, en la construcción y en el diseño del presente sistema.

Para comenzar con la realización del proyecto se tomaron en cuenta sus requerimientos y elementos básicos con los que va a contar; tales como; la elaboración de un sistema gráfico; es decir; el ambiente visual, que se le va a presentar al usuario, y los componentes que lo formarán, por ejemplo; dado que este sistema será de tipo informativo y que se ejecutará bajo ambiente Windows; entonces utilizará el concepto de ventanas, ya que mediante ellas se proporcionará la información respectiva que el usuario desee. Ahora bien, de este punto tenemos que considerar que probablemente por el tamaño de la información no se despliegue completamente en la ventana, así que tenemos que implementar un objeto que nos permita visualizar poco a poco la información, y este objeto es la llamada barra de desplazamiento o de Scroll que nos permitirá movernos a través de la ventana.

Después de contemplar como se desplegará la información, consideramos la forma de accesar a esas ventanas y para ello se contará con una barra de menú con todos los comandos necesarios para su manejo, además de que el sistema también tendrá disponible una barra de herramientas la cual es un menú gráfico compuesto de iconos que son dibujos relativos a la función que realizan dentro del sistema.

Para tener acesso tanto a la barra de menú como a la barra de herramientas se utilizarán los dos dispositivos de entrada de la computadora, el teclado y el ratón. Por ejemplo, para utilizar la barra de menú mediante el teclado, el usuario presionará la tecla Alt; para activar la barra; y después con las teclas de navegación; moviéndose de izquierda a derecha; se colocará sobre el nombre del comando deseado para ejecutarlo. Para accesar con el Ratón a la barra de herramientas; la cual realiza la misma función que la barra de menú; se colocará el apuntador del ratón sobre el icono deseado y se presionará el botón izquierdo para ejecutar la acción.

Para desarrollar los requerimientos anteriormente descritos; a cada uno de ellos se les colocará dentro del concepto de **Clases** para implementar sus funciones.

Para saber como podemos definir y formar dichas clases, utilizaremos los siguientes pasos:

- Identificación de las clases básicas
- Asignar atributos y comportamientos
- Encontrar las relaciones entre las clases
- Colocar las clases dentro de Jerarquías

Identificación de las clases

Para la identificación de las **clases básicas** que el sistema manejará; nos tenemos que hacer la siguiente pregunta ¿Qué clases son necesarias para este sistema?, una respuesta inmediata es la **clase ventana** ya que es el objeto que se utilizará con más frecuencia. Entonces la primera **clase** es llamada **Ventana**.

Ahora bien; el sistema maneja las acciones del usuario con los dos dispositivos de entrada, el ratón y el teclado. Estas acciones son significativas para el sistema, porque ellas cambian el estado de una ventana; y de este punto, surge la segunda **clase** llamada **Evento** que describe tales estados de la ventana. Por ejemplo cuando un usuario presiona el botón

del ratón sobre una ventana, una señal del **objeto Evento** es pasada al **objeto Ventana**, y este responde adecuadamente. Esto es un ejemplo de una clase que describe las cosas que le pasan a los objetos. Pero en el caso de que existan varias ventanas a la vez debe de existir una clase que se encargue de mantener la información sobre todas las ventanas y las relaciones entre ellas y el usuario. Esta es la tercera clase y es llamada **Administrador de ventanas.** Dicha clase no modela una entidad mencionada en la descripción de nuestro sistema, es creada porque existe información que debe ser mantenida por el sistema, aunque esa información no será utilizada por otras clases. Es por esta razón que este sistema frecuentemente maneja posiciones, y para representar a estas es conveniente utilizar una cuarta clase llamada **Punto** para que las controle. Análogamente el sistema utiliza ventanas en forma rectangular, entonces se declara, la última clase básica, **la clase Rectángulo** que se encarga de manejar la forma de las ventanas.

Atributos y comportamientos

Para realizar este punto nos hacemos la siguiente pregunta ¿Qué información manejan las clases? por ejemplo la clase Administrador de ventanas controla los eventos generados por el usuario.

La clase Ventana se encarga de conocer el tamaño y la posición de las pantallas del sistema, además de que almacena toda la información que debe ser desplegada por una ventana que se desplace, incluyendo el texto de afuera de la ventana. La ventana conoce su posición, la cual relaciona al texto completo con la barra de desplazamiento.

Por ejemplo consideremos el comportamiento de la barra de desplazamiento más de cerca. Si el usuario se mueve hacia el final del documento, el texto se mueve una linea a la vez en esa dirección.

En el párrafo anterior se describe un conjunto de acciones bastante complejo para ser ejecutadas por una sola ventana. Más aún, si sabemos que una ventana puede tener dos barras de desplazamiento, una vertical y una horizontal, esto trae como consecuencia que

para la realización de este proceso se requiere repetir muchas veces el código para que responda a la señal del ratón. En vez de hacer que una sola ventana sea responsable del comportamiento de la barra de desplazamiento, se puede definir una clase separada que se encargue del funcionamiento de estas barras. Con esto se logra que una ventana sea la responsable de la interpretación de la señal del ratón sobre el texto, y que la barra de desplazamiento sea la responsable de la interpretación de la señal del ratón en su superficie. Y así; los objetos de esas dos clases tanto la barra como la ventana interactuan para ejecutar el desplazamiento en el texto. Una de las funciones principales de la barra es conocer el tamaño de la ventana y la posición sobre la pantalla.

Y de esto encontramos que el propósito de la información que almacena el objeto Evento es determinar el tipo de información que maneja; por ejemplo, si la señal de entrada viene del teclado, entonces la información relevante es la tecla que fue presionada. Si la señal de entrada viene del ratón, la información relevante es la localización del cursor del ratón y de los botones del ratón.

Estos procesos, son dos distintos tipos de eventos, que se pueden implementar en dos clases separadas, llamadas OnLButtonDown y OnRButtonDown

¿Qué hacen las clases OnLButtonDown y OnRButtonDown?. Estas dos clases no hacen mucho, excepto transportar la información que contienen.

Identificación de las relaciones entre las clases

Una ventana puede tener una o dos barras de desplazamiento. Cada barra esta definida como una ventana hija y cada ventana es la madre de las barras de desplazamiento, esto puede ser representado mediante un diagrama de relaciones, donde la clase Ventana contiene una o dos instancias de la barra de desplazamiento.

La interacción entre las ventanas y las barras requieren dos formas de comunicación, una es utilizando las barras de desplazamiento que afectan a la ventana y la otra es

moviendo el cursor de la ventana para que afecten a la barra. Como resultado, la barra debe conocer el tamaño de las ventanas.

Así los eventos son pasados para interactuar con las diferentes ventanas o barras de desplazamiento, entonces el interactor debe tener una función que acepte un objeto evento como un parámetro.

Interfaces

La jerarquía ventana consiste de una clase base que interactua con las clases derivadas, Ventana y Barra de desplazamiento. Vamos a considerar estas clases moviéndonos desde el principio de la jerarquía hacia abajo.

Las características que tienen en común todos los interactores, incluyendo las ventanas de texto y las barras de desplazamiento, son que tienen un tamaño y una posición relativa y esta información debe ser accesible. Ya que ellas deben de saber si un par de coordenadas dado, cae dentro de sus bordes, indicando con esto que deberán responder a una señal del ratón. En resumen todos los interactores deben tener la capacidad para desplegarse ellos mismos y para responder a los eventos.

Colocar las clases dentro de jerarquías

Las jerarquías son muy sencillas en este punto, así que no se necesita reestructurarlas. Este paso se puede realizar hasta que termine el proceso de diseño, sin embargo es apropiado decidir que características pertenecen a la clase base y que características pertenecen a las clases derivadas. Esto requiere de una examinación más profunda de los atributos y del comportamiento de las clases, así que primeramente se tiene que hacer un análisis de las clases con interfaces públicas, que son las siguientes:

moviendo el cursor de la ventana para que afecten a la barra. Como resultado, la barra debe conocer el tamaño de las ventanas.

Así los eventos son pasados para interactuar con las diferentes ventanas o barras de desplazamiento, entonces el interactor debe tener una función que acepte un objeto evento como un parámetro.

Interfaces

La jerarquía ventana consiste de una clase base que interactua con las clases derivadas, Ventana y Barra de desplazamiento. Vamos a considerar estas clases moviéndonos desde el principio de la jerarquía hacia abajo.

Las características que tienen en común todos los interactores, incluyendo las ventanas de texto y las barras de desplazamiento, son que tienen un tamaño y una posición relativa y esta información debe ser accesible. Ya que ellas deben de saber si un par de coordenadas dado, cae dentro de sus bordes, indicando con esto que deberán responder a una señal del ratón. En resumen todos los interactores deben tener la capacidad para desplegarse ellos mismos y para responder a los eventos.

Colocar las clases dentro de jerarquías

Las jerarquías son muy sencillas en este punto, así que no se necesita reestructurarlas. Este paso se puede realizar hasta que termine el proceso de diseño, sin embargo es apropiado decidir que características pertenecen a la clase base y que características pertenecen a las clases derivadas. Esto requiere de una examinación más profunda de los atributos y del comportamiento de las clases, así que primeramente se tiene que hacer un análisis de las clases con interfaces públicas, que son las siguientes:

Clase Ventana

La jerarquía ventana consiste de una clase base que interactua con las clases derivadas ***Ventanas*** y ***Barra de desplazamiento***. Vamos a considerar estas clases comenzando, de arriba hacia abajo.

Las características que tienen en común todos los interactores incluyendo las barras de desplazamiento y las ventanas de texto, es que todas ellas tienen un tamaño y una posición relativa, y esta información debe ser accesible, ya que deberán registrar cuando un par de coordenadas dado, caiga dentro de sus bordes indicando que deben responder a una señal del ratón. En resumen, todos los interactores tienen la capacidad para desplegarse ellos mismos y de responder a los eventos (señales del ratón). Sin embargo como un interactor no es un objeto genérico que pueda desplegarse el mismo o de responder a los eventos, esas funciones deben ser declaradas como funciones virtuales puras.

Ventanas de texto

Una ventana de texto almacena toda la información que va a ser desplegada. Un arreglo puede ser usado para manejar texto, algunas veces de tamaño variable como en las listas ligadas, para hacerlo más flexible, pero en este caso cada ventana tiene una cantidad de texto fija.

Cada ventana puede contener una o dos barras de desplazamiento o ninguna de ellas, dependiendo de la longitud y el ancho del texto que va a ser desplegado.

Barras de Desplazamiento(scroll)

Una barra de desplazamiento conoce su función; ya sea si es vertical u horizontal; y la información que representa. Conoce la posición de la caja de desplazamiento la cual puede ser representada como un número entre 1 y 100.

Una barra de desplazamiento debe tener el ancho de un caracter si es vertical y el alto de un caracter si es horizontal. Y no puede estar separada de una ventana de texto, es más la ventana de texto debe de existir antes que la barra de desplazamiento sea creada.

También existen otras funciones que manejan la barra de desplazamiento por ejemplo si el usuario desplaza una ventana de texto moviendo el cursor con las teclas de navegación, la barra de desplazamiento de la caja debe de moverse también. No es muy apropiado manejar desde el teclado esta barra, ya que esto rompe con su propósito ademas de ser muy incomodo, la ventana de texto es la encargada de manejarla. La barra de desplazamiento debe sin embargo tener una función que fije la posición de la caja la cual se llama ***Ventana.***

Cuando el botón del ratón es presionado sobre la barra de desplazamiento, el texto en la ventana también se desplaza. El objeto ***Barra de desplazamiento*** dice que se asocie con el objeto ***Ventana*** para ejecutar la acción de desplazamiento, puede hacerse esto enviando un nuevo tipo de evento, un ***DesplazamientoEvento***, que contiene la dirección (hacia atrás y hacia adelante) y la cantidad de texto (una línea o una página).

Una barra de desplazamiento puede mover su propia caja de desplazamiento, ¿pero que tanto puede moverla?; primero se tiene que considerar la longitud del documento y el ancho de la ventana. Por ejemplo si el texto es cuatro veces tan largo como la ventana, desplazar una página hacia abajo significa mover la caja de desplazamiento ¼ de la barra de desplazamiento hacia abajo. Sin embargo si el texto es 20 veces tan largo como la ventana, desplazar una pagina hacia abajo significa mover la caja de desplazamiento 1/20 de la barra de desplazamiento hacia abajo. Desplazar una línea a la vez es un proceso similar. Esto es, calcular la distancia que la caja de desplazamiento puede mover la barra entonces, deberá de conocer la longitud del documento y la altura de la ventana (o el porcentaje correspondiente del desplazamiento horizontal). Esto podría duplicar la responsabilidades, porque esos valores son almacenados por la clase Ventana, sin embargo la barra de desplazamiento no mueve su caja de desplazamiento cuando el ratón es accionado,

únicamente ajusta la caja cuando la clase Ventana llama la función setSlider. La clase Ventana es responsable de computar apropiadamente la posición de la caja de desplazamiento.

El administrador de ventanas

La clase administrador de ventanas registra el orden de las ventanas de texto y los eventos hechos por el usuario.

Sobre la pantalla, las ventanas aparecen como un pila de hojas de papel sobre el escritorio, siendo la ventana de la cima la única ventana activa, esto sugiere una clase que implemente una pila como estructura de datos, sin embargo una pila permite accesar únicamente al elemento de la cima. El Administrador de ventanas debe accesar a las otras ventanas sin desactivar la actual ventana. Alguna clase de lista permite accesar a todos los elementos que sean necesarios.

El administrador de ventanas debe accesar estas para ver el orden en que reciben una señal del ratón. Más que una ventana debe registrar la posición donde el ratón fue accionado pero únicamente la ventana expuesta es la que recibe la señal. Para encontrar la ventana expuesta, el administrador de ventanas debe de probar el orden de las ventanas comenzando desde la ventana de la cima de la pila hacia abajo. Sin embargo, la pantalla debe refrescarse sobreponiendo las ventanas otra vez. Esta vez, el administrador debe accesar las ventanas comenzando desde la ventana de abajo de la pila hacia arriba. Así, el administrador de ventanas necesita una clase que permita la iteración de sus elementos en ambas direcciones.

Por consiguiente, una clase Lista que permite las operaciones en ambas direcciones es la apropiada, tal lista deberá ser implementada con una doble lista ligada o en un arreglo. La clase administrador de ventanas puede tener un objeto miembro de tipo lista.

Consideramos como las clases Administrador de ventanas, Ventana, y Barra de desplazamiento trabajan juntas. Cuando el usuario acciona el ratón sobre la barra de

desplazamiento y sobre la ventana de texto. El objeto Evento contiene la locación donde la señal del ratón ocurre la cual es enviada al objeto administrador de ventanas ya que esta clase es la encargada de organizar la ventanas y busca cual de ellas recibió la señal del ratón esta pasa la señal al objeto Ventana para verificar si las barras de desplazamiento están siendo accionadas.

Todo el proceso anterior que describe la comunicación entre las ventanas y el usuario se ilustra en la siguiente figura 3.1.6.1.

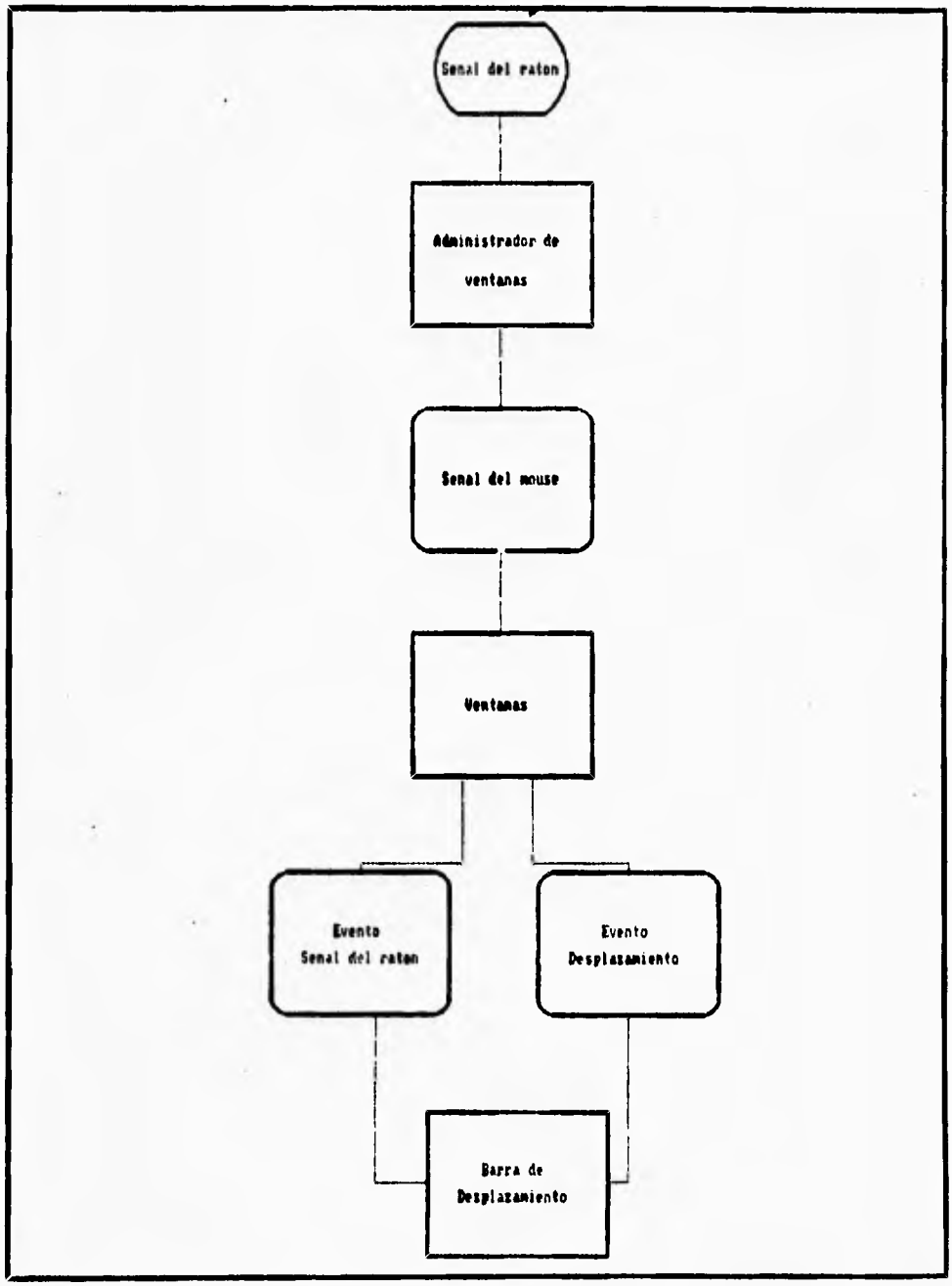


Fig. 3.1.6.1 Administrador de Ventanas.

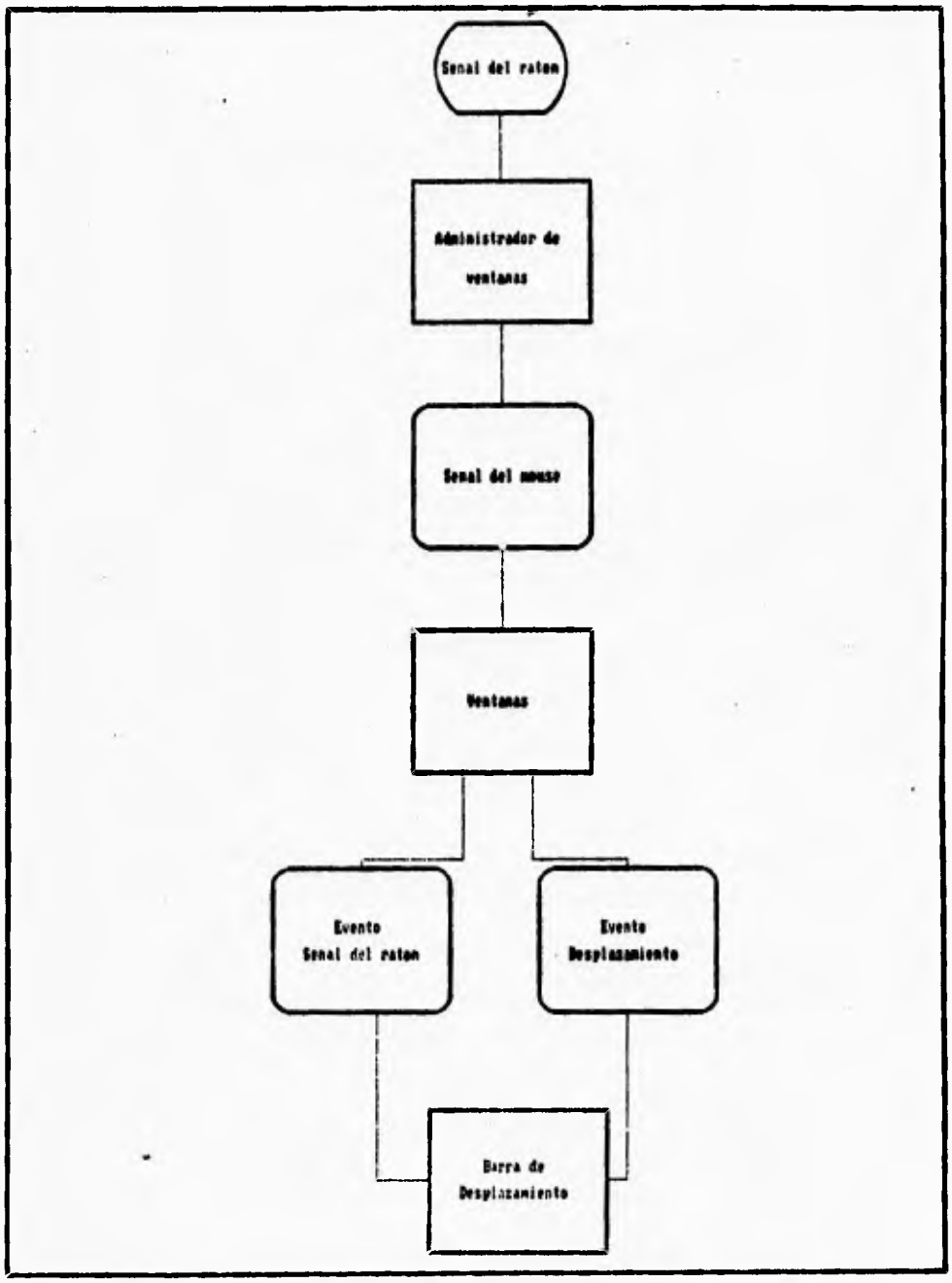


Fig. 3.1.6.1 Administrador de Ventanas.

Uso de App Wizard

Una vez que sabemos como ir identificando las clases y su comportamiento en nuestro sistema, procederemos a hacer uso de una de las herramientas que el lenguaje nos ofrece para definirlas de una manera fácil. El lenguaje Visual C++ utiliza tres poderosas herramientas que nos ayudan en la creación de una aplicación de Windows y una de ellas es la utileria App Wizard (las otras dos son Class Wizard y App Studio) que se emplea para la creación de código fuente de manera automática, con esto se genera el código básico del programa, el cual se puede introducir ya sea manualmente o utilizando las herramientas Class Wizard y App Studio. La utileria App Wizard emite una serie de archivos que contienen varias clases que construyen la aplicación. La utileria Class Wizard nos permite usar el código emitido por App Wizard, para generar otras clases que complementan la aplicación, la utileria App Studio a su vez también nos permite usar varios de los archivos generados por App Wizard para generar otros componentes del sistema .

La rutina que se sigue para generar estos archivos es la siguiente:

1. Se entra a la aplicación del Visual C++ y se invoca la opción **App Wizard** del menú ***project.***

   Se teclea Site como nombre del proyecto y se selecciona el directorio donde se almacenará este último.

2. Se presiona el botón con la etiqueta ***Options***, de este paso aparece la caja de dialogo "Options" en la cual marcaremos la opción de barra de herramientas inicial, que va a contener nuestra aplicación.
3. Se presiona el botón de ***OK*** de la caja de diálogo Options para regresarnos a la pantalla del App Wizard.
4. Se presiona el botón con la etiqueta ***Classes*** para abrir su caja de diálogo la cual lista las siguientes clases:

- La clase CSiteApp, la cual modela la aplicación SITE
- La clase CMainFrame, la cual maneja la estructura de las ventanas
- La clase CSiteDoc que modela el documento de aplicación
- La clase CSiteWiew la cual representa la apariencia de la aplicación

5. Se presiona el botón ***OK*** para regresar a la pantalla de App Wizard
6. Aparece una ventana con el resumen de todas las clases que se van a generar, entonces se oprime el botón con la etiqueta ***Create*** para construir el código fuente del archivo que genera el proyecto.

La generación de archivos termina y se puede listar los archivos abriéndolos seleccionando la opción de abrir ya sea en la barra de menú o en la barra de herramientas de la pantalla principal del Visual C++.

Los archivos que generó App Wizard son los siguientes:

MAINFRM.CPP

MAINFRM.H

RESOURCE.H

SITEDOC.CPP

SITEDOC.H

SITEWIEW.CPP

SITEWIEW.H

SITE.DEF

SITE.H

SITE.MAK

SITE.RC

STDAFX.CPP

STDAFX.H

Los archivos con la extensión .CPP son los archivos de implementación de las clases, los archivos con la extensión .H contienen la definición de las estructuras llamadas ID´s que son las variables que se encargan de ligar las ventanas con los comandos del menú principal. El archivo con la extensión .DEF define el módulo de parámetros que la aplicación maneja declararando el nombre del archivo y la descripción del mismo. El archivo con la extensión .MAK es el archivo que se genera cuando se ligan todos los archivos para construir el sistema. El archivo con la extensión .RC mantiene la información que controla la apariencia visual del sistema.

Para construir el archivo ejecutable se selecciona la opción ***Build Execute*** del menú ***project***. Entonces el archivo que se genera contiene un menú y una barra de herramientas inicial con diferentes opciones, las cuales nosotros modificamos para diseñar nuestra propia aplicación.

Para ello se crearon otras clases para complementar el propósito del sistema y para lograrlo nos valimos de otras de las herramientas del lenguaje que es **Class Wizard**, esta se utiliza para ir declarando el nombre de otras clases en el archivo CSiteView . La implementación de la función de cada una de estas clases se hizo manualmente.

Los siguientes párrafos se basarán en la explicación de las funciones de las clases dentro del algoritmo del sistema, además de que se anexará el código de dichas funciones.

Algoritmo del sistema

Primeramente se hará una declaración de las variables públicas y de los archivos que las clases van a utilizar más adelante y el código es el siguiente:

```
1.  / siteview.cpp :Implementación de la clase CSiteView
2.  #include "stdafx.h"
3.  #include "site.h"
4.  #define MARCA 9999
```

```
5.  #include "sitedoc.h"
6.  #include "siteview.h"
7.  #include "dradio.h"
8.  #include "dcuadro.h"
9.  #include "dselecc.h"
10. #include "dbienser.h"
11. #include "dinfo.h"
12. #ifdef _DEBUG
13. #undef THIS_FILE
14. static char BASED_CODE THIS_FILE[] = __FILE__;
15. #endif
16. typedef struct  {      entero x;
17. entero y;
18. char titulo[31]; }     desc_sit;
19. typedef struct  {      entero reg;
20. entero x;
21. entero y;
22. byte  giro; } lugar_sit;
23. entero *a_apt;                    // Apuntador
24. entero *a_lin;                    // puntos del mapa
25. int ancho = 4;                    // relleno de los puntos rojos
26. double acerca = .05;              // inicialización del aumento del sistema en 5%
27. byte ciudad = 0, limite = 1,      // inicialización de variables
    giro_act = 0, examinar=0, Amplificar=1;
28. long max_x, max_y, min_x, min_y;  //declaración de la variables que controlan el
                                      tamaño de la ventanas y el mapa
29. long *stack, *a_stack;                  // declaración de la pila
```

```
30.  double derecho, abajo, dif_x,dif_y;
31.  char *na_datos = "i_datos.dbf",      //declaración de las bases de datos
     *na_info = "i_info.dbf",
     *na_datost = "i_datost.dbf";
32.  char *na_lugt = "i_lugt.dbf",
     *na_giro = "i_gir.dbf",
     *na_sit = "i_sit.dbf";
33.  CFile fsit;
34.  desc_sit  *a_dat;                  // sitios en el mapa
35.  lugar_sit *a_inf;          // Relación de Bienes y Servicios en el archivo
36.  lugt_sit  *a_lugt;         // lista de lugares turísticos
37.  desc_giro *a_giro;         // Relación de giros
```

1.- En primer lugar el usuario seleccionará la ciudad sobre la cual desea obtener información, ya sea Ixtapa o Zihuatanejo.

Para que el sistema diferencie entre las dos ciudades, se tomará el estado de una variable llamada **ciudad** cuyo valor variará entre 0 y 1, inicilalizada primeramente con el valor de 0, el cual corresponde a la ciudad de Ixtapa y el valor de 1 le corresponde a la ciudad de Zihuatanejo. El estado de esta variable permitirá abrir los archivos de texto (archivos con la extensión TXT) para que se lea la información cultural de las ciudades.

Los siguientes códigos ejecutan el proceso anterior:

```
1.  void CSiteView::OnCiudadIxtapa()
2.  {
3.  CClientDC dc( this );
4.  CRect rect;
5.  GetClientRect( rect );
6.  ciudad = 0;                    //variable ciudad
7.  na_datos  = "i_datos.dbf";
8.  na_info   = "i_info.dbf";
9.  na_datost = "i_datost.dbf";
10. na_lugt   = "i_lugt.dbf";
11. na_giro   = "i_gir.dbf";
12. na_sit    = "i_sit.dbf"; // archivo que contiene la ubicación de los sitios de
                                Ixtapa
13. free(a_inf);
14. free(a_lin);
15. free(a_dat);
16. free(a_lugt);
17. free(a_giro);
18. giro_act = 0, examinar=0, Amplificar=1;
19. leer();
20. fsit.Close();
21. fsit.Open(na_sit,CFile::modeRead);   //abre el archivo i_sit.dbf
22. InvalidateRect( rect ,TRUE);
23. }
```

```
1.  void CSiteView::OnCiudadZihuatanejo()
2.  {
3.  CClientDC dc( this );
4.  CRect rect;
5.  GetClientRect( rect );
6.  ciudad = 1;              //variable ciudad
7.  na_datos  = "z_datos.dbf";
8.  na_info   = "z_info.dbf";
9.  na_datost = "z_datost.dbf";
10. na_lugt   = "z_lugt.dbf";
11. na_giro   = "z_gir.dbf";
12. na_sit    = "z_sit.dbf"; //archivo que contiene la ubicación de los sitios de
                             //Zihuatanejo
13. free(a_inf);
14. free(a_lin);
15. free(a_dat);
16. free(a_lugt);
17. free(a_giro);
18. giro_act = 0, examinar=0, Amplificar=1;
19. leer();
20. fsit.Close();
21. fsit.Open(na_sit,CFile::modeRead); // abre el archivo z_sit.dbf
22. InvalidateRect( rect,TRUE);
23. }
```

Dependiendo del estado de la bandera **ciudad** ya sea si es 0 o 1 se abre el archivo de texto **"zihuata.txt"o"Ixtapa.txt"**, que contiene la información cultural correspondiente. El código que despliega ésta es el siguiente:

```
1.  void CSiteView::OnSiteConocer()
2.  {
3.
4.  DCuadro dlg;          // ventana que despliega la información
5.  CFile temp;
6.  double tam;
7.  char *buffer;
8.  temp.Open((ciudad)?"zihuata.txt":"Ixtapa.txt",CFile::modeRead); // abre
                                                                    archivo de
                                                                    texto
9.  tam = temp.GetLength();
10. buffer = (char*) malloc(tam+1);  // reserva espacio de memoria para los archivos de
                                     //texto
11. temp.Read(buffer,tam);
12. temp.Close();
13. *(buffer+(int)tam) = '\0';
14. dlg.m_titulo = (ciudad)?"ZIHUATANEJO":"IXTAPA"; //despliega el titulo en la
                                                    //ventana que se abre
15. dlg.m_texto = buffer;
16. dlg.DoModal(); // declaración de que tipo de ventana que se abre
17. free(buffer); //cuando termina libera la memoria reservada
18. }
19. }
```

El procedimiento que lee los datos y que es llamado en las dos clases anteriores es el siguiente. El procedimiento se llama **leer()**.

```
1.  void leer() {
2.  CFile temp;
3.  char *buff, *c_apt4, *c_apt5, *c_apt2;
4.  entero *a_apt, x, y, i, n_apt;
5.  desc_sit *p_dat;
6.  lugar_sit *i_apt;
7.  lugt_sit *l_apt;                    //Apuntadores de las bases de datos
8.  desc_giro *g_apt;
9.  buff = (char*) malloc(200);
10. c_apt4 = (char*) malloc(5); *(c_apt4+4) = '\0';     //reserva de memoria
11. c_apt5 = (char*) malloc(6); *(c_apt5+5) = '\0';     //reserva de memoria
12. c_apt2 = (char*) malloc(3); *(c_apt2+2) = '\0';     //reserva de memoria
```

2.- Dependiendo de la ciudad elegida se desplegará el mapa respectivo de la ciudad. Esto se hace mediante un procedimiento de lectura de datos de los archivos "**l_datos.dbf**" o "**z_datos.dbf**" que son los archivos que contienen los puntos en coordenadas X,Y que forman los mapas de las ciudades.

El siguiente código ejecuta el proceso anterior:

```
1.  // Lectura de puntos del mapa
2.  temp.Open(na_datos,CFile::modeRead);     //Esta instrucción abre el archivo de
                                             //datos  i_datos.dbf o z_datos.dbf
3.  temp.Read(buff,4);          // salta 4 bytes
4.  temp.Read(&n_apt,2);  // Pregunta por la cantidad de registros del archivo .DBF
5.  temp.Read(buff,2);       // salta dos bytes
6.  a_lin = (entero*) malloc((n_apt*2+1)*sizeof(entero));     // Esta instrucción
                                                              reserva // memoria para
                                                              los //puntos del mapa
7.  a_apt = a_lin;
8.  temp.Read(buff,90);
9.  for(i=1;i<=n_apt;i++) {                  //Inicio del bloque de lectura de datos
10. temp.Read(c_apt4,1);
11. temp.Read(c_apt4,4); x = atoi(c_apt4);
12. temp.Read(c_apt4,4); y = atoi(c_apt4);
13. *a_apt++ = x;
14. *a_apt++ = y;
15. };
16. *a_apt++ = FINAL;
17. temp.Close();                            //Final del bloque de lectura
```

3.- Una vez que el mapa de la ciudad elegida se visualice sobre la pantalla, el usuario seleccionará el tipo de informacion que desea obtener, ya sea un bien, servicio o atracción turística. Para diferenciar entre estos, el sistema lee dos archivos **"z_giro.dbf"** o **"l_giro.dbf"** ya que estos almacenan toda la información de cada ciudad. En la estructura de estos archivos se encuentra un campo llamado **Tipo** que es el que distingue entre el tipo de información ya sea si es un Bien o un Servicio; y lo que hace es almacenar el valor de 0

cuando se refiere a Bienes y almacena el valor de 1 cuando se trata de Servicios y estos a su vez estan asociados a un nombre del lugar y a una clave que se le asigna a cada lugar. La selección de Atracciones Turísticas se encuentra implementada en otra clase que se explica en el paso 6.

La estructura de este archivo es la siguiente:

Clave del giro	Nombre del Bien o Servicio	Tipo
1	Abarrotes	0
6	Farmacias	0
7	Labs. Clínicos	1
9	Electrico Automotríz	1

La elección del tipo de información la realizará el siguiente código:

```
1.  // Lectura de Relación de giros
2.  temp.Open(na_giro,CFile::modeRead); //distingue entre el tipo de información
3.  temp.Seek(4,CFile::begin);
4.  temp.Read(&n_apt,2);
5.  temp.Seek(2,CFile::current);
6.  a_giro = (desc_giro*) malloc((n_apt+1)*sizeof(desc_giro));
7.  g_apt = a_giro;
8.  temp.Seek(121,CFile::current);
9.  for(i=1;i<=n_apt;i++) {
```

```
10.  temp.Seek(1,CFile::current);
11.  g_apt->marca = 0;
12.  temp.Read(c_apt2,2);              g_apt->giro = atoi(c_apt2);
13.  temp.Read(g_apt->nombre,36);      g_apt->nombre[36] = '\0';
14.  temp.Read(c_apt4,1);              g_apt->tipo = *c_apt4 - '0';
15.  *g_apt++;
16.  };
17.  g_apt->marca = FINAL;
18.  temp.Close();
```

4.- Al seleccionar el tipo de información, se abrirá una ventana que despliega la lista con todos los nombres de los Bienes, Servicios y Atracciones Turísticas que las ciudades ofrecen. De esa lista el usuario seleccionará el nombre de un lugar y al hacerlo, sobre el mapa se visualizará, con puntos rojos, la ubicación del sitio o sitios donde se encuentran dichos lugares. Este proceso lee los archivos "i_info.dbf" o "z_info.dbf", ya que estos almacenan el número de registro, la ubicación en cordenadas X,Y y la clave del giro ya que esta se asocia con el nombre de la ubicación del sitio el cual se encuentra en los archivos "i_giro.dbf" y "z_giro.dbf" anteriormente descritos. La estructura de esos archivos es la siguiente:

Registro	X	Y	Clave del giro
10	12	50	1
20	20	48	6
30	25	47	7

El siguiente código ejecuta el proceso anterior:

```
1.  //Lectura ubicacion en el archivo de Bienes y Servicios
2.  temp.Open(na_info,CFile::modeRead);
3.  temp.Read(buff,4);
4.  temp.Read(&n_apt,2);
5.  temp.Read(buff,2);
6.  a_inf = (lugar_sit*) malloc( (n_apt+1)*sizeof(lugar_sit));
7.  i_apt = a_inf;
8.  temp.Read(buff,153);
9.  for(i=1;i<=n_apt;i++) {
10. temp.Read(c_apt5,1);
11. temp.Read(c_apt5,5); i_apt->reg  = atoi(c_apt5);
12. temp.Read(c_apt4,4); i_apt->x    = atoi(c_apt4);
13. temp.Read(c_apt4,4); i_apt->y    = atoi(c_apt4);
14. temp.Read(c_apt2,2); i_apt->giro = atoi(c_apt2);
15. i_apt++;
16. };
17. i_apt->x = FINAL;
18. i_apt->y = FINAL;
    temp.Close();
```

.5- Sobre los puntos rojos aparece un letrero que indica el nombre de ese sitio. La colocación de esos letreros se hace mediante un proceso de lectura de los archivos **"i_datost.dbf"** y **"z_datost.dbf"**, ya que estos se encargan de relacionar el punto (coordenada (x,y)) con el nombre que corresponde a ese sitio.

La estructura de esos archivos es la siguiente:

Campo X	Campo Y	Nombre del Lugar
36	12	Hotel Ixtapa
50	72	Casa de bolsa
Xn	Yn	Lugar n

El siguiente código ejecuta el proceso anterior:

```
1.  // Lectura sitios en el mapa
2.  temp.Open(na_datost,CFile::modeRead); //abre el archivo datost.dbf
3.  temp.Read(buff,4);
4.  temp.Read(&n_apt,2);
5.  temp.Read(buff,2);
6.  a_dat = (desc_sit*) malloc((n_apt+1)*sizeof(desc_sit));
7.  p_dat = a_dat;
8.  temp.Read(buff,121);
9.  for(i=1;i<=n_apt;i++) {
10. temp.Read(c_apt5,1);
11. temp.Read(c_apt4,4); p_dat->x    = atoi(c_apt4);
12. temp.Read(c_apt4,4); p_dat->y    = atoi(c_apt4);
13. temp.Read(p_dat->titulo,30);
14. p_dat->titulo[30] = '\0';
15. p_dat++;
```

```
16.   );
17.   p_dat->x = FINAL;
18.   p_dat->y = FINAL;
19.   temp.Close();
```

6.- Al elegir uno de los nombres de la lista de Atracciones Turísticas, se desplegará una ventana con la información cultural de esa atracción en especial. Para obtener dicha información, el sistema abrirá los archivo "i_lugt.dbf" o "z_lugt.dbf" ya que estos almacenan todos los nombres de los lugares asociadas con los nombres de los archivos de texto que contienen la información cultural de dichos lugares. La estructura de los archivos es la siguiente:

Nombre del lugar	Archivo en que se encuentra la información
Playa del Coral	ICORAL.TXT
Playa Cuachalalate	ICUACHA.TXT
Playa las Cuatas	ICUATAS.TXT

El siguiente código ejecuta el proceso anterior:

```
1.   // Lectura de lugares turísticos
2.   temp.Open(na_lugt,CFile::modeRead);
3.   temp.Read(buff,4);
4.   temp.Read(&n_apt,2);
5.   temp.Read(buff,2);
6.   a_lugt = (lugt_sit*) malloc((n_apt+1)*sizeof(lugt_sit));
```

```
7.  l_apt = a_lugt;
8.  temp.Read(buff,89);
9.  for(i=1;i<=n_apt;i++) {
10. temp.Read(c_apt5,1);
11. l_apt->marca = 0;
12. temp.Read(l_apt->nombre,36);
13. temp.Read(l_apt->archivo,12); // lectura del archivo.TXT que contiene la
                                  //información
14. l_apt->nombre[36] = '\0';
15. l_apt->archivo[12] = '\0';
16. l_apt++;
17. };
18. l_apt->marca = FINAL;
19. temp.Close();
20. free(buff);
21. free(c_apt5);
22. free(c_apt4);
23. free(c_apt2);
24. }
25. void trans(long x, long y, long *xp, long *yp) {
26. *xp = ((x-min_x)/dif_x) * derecho;
27. *yp = (1 - ((y-min_y)/dif_y)) * abajo;
28. }
```

7.- Se establece si se desea hacer un acercamiento (zoom) teniendo 3 diferentes opciones de aumento 5%, 25% y 50% . Esto se logra colocando el apuntador del ratón sobre la zona que se quiera observar con más detalle y presionado el botón del mismo.

El siguiente código describe como el sistema diferencia entre los tipos de aumento que el usuario selecciona.

```
1.  void CSiteView::OnUbicarRadiodeaccin()
2.  {
3.  DRadio dlg;
4.  if (acerca <= 0.05)  // como el 5% es el aumento inicial. Se hace a partir de ese
                         valor //la comparación
5.  dlg.m_radio1 = 0;
6.  else if (acerca <= 0.25) //Pregunta si se escogió el 25% de aumento
7.  dlg.m_radio1 = 1;
8.  else if (acerca <= 0.50) //Pregunta si se escogio el 50% de aumento
9.  dlg.m_radio1 = 2;
10. if (dlg.DoModal()==1) {
11. if (dlg.m_radio1 == 0)
12. acerca = 0.05;
13. else if (dlg.m_radio1 == 1)
14. acerca = 0.25;
15. else if (dlg.m_radio1 == 2)
16. acerca = 0.50;
17. };
```

Anteriormente se mencionó que una de las principales funciones del sistema; es desplegar los mapas de las respectivas ciudades, los cuales van a responder a una señal del ratón o del teclado. Para lograr lo anterior se manejaran dos archivos con formato DBF (la extensión .DBF siginifica archivo de base de datos) que almacenarán la información con los

puntos que forman los dos mapas en coordenadas (X,Y), de ahí a cada punto del mapa se le asignará la ubicación de algún sitio en particular, pudiendo ser este algún Bien, Servicio o Atracción Turística y a cada uno de estos lugares se almacenarán en dos bases de datos siendo estas, "i_datost.dbf" y "z_datos.dbf."

Los archivos "i_datost.dbf" y "z_datost.dbf" tendrán la siguiente estructura :

Coordenadas Xd	Coordenadas Yd	Tipo	Descripción
Xd1	Yd1	Base de datos servicios	Banco
Xd2	Yd2	Base de datos Bienes	Hotel
Xd3	Yd3	Base de datos Atracciones Turísticas	Playa el Palmar
Xdn	Ydn		Banco n

donde:

n: número de puntos

Tipo: especificar el nombre de la base de datos en que se encuentra ese punto

Descripción : nombre del lugar al que corresponde ese punto

Y así a cada sitio que se seleccione sobre el mapa, inmediatamente el sistema identificará la posición que ocupa en la ventana y buscará en los archivos **"i_datost.dbf"** y **"z_datost.dbf"** a que punto corresponde ese sitio del mapa. Cuando lo haya hecho, buscará el archivo que contiene la base de datos en que se encuentra ese lugar y desplegará una ventana de texto con la información respectiva de ese sitio.

Para realizar el cambio de aumento se hará una transformación de coordenadas para la graficación de los mapas.

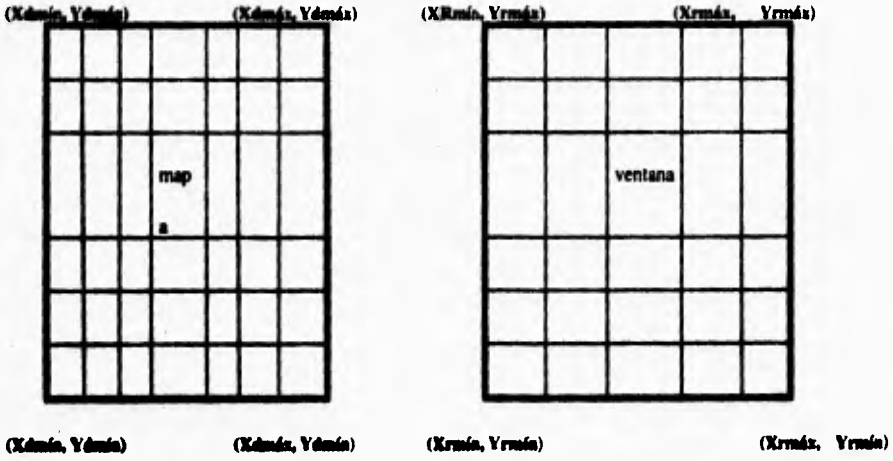
Definiendo el proceso anterior de la siguiente manera:

Se considerarán dos objetos una ventana y un mapa, los cuales estarán representados como dos matrices formadas de renglones llamados (X) y columnas llamadas (Y). Para diferenciar los renglones y las columnas que forman el mapa y la ventana, a los renglones y a las columnas del mapa les llamaremos (Xd,Yd) respectivamente y a los renglones y a las columnas de la ventana les llamaremos (Xr,Yr) también respectivamente.

Los rangos de los renglones y las columnas serán :

Renglones Xd = Xdmínima . . .a Xdmáxima;
Columnas Yd = Ydmínima . . . a Ydmáxima;
Renglones Xr = Xrmínima . . . a Xrmáxima;
Columnas Yr = Yrmínima . . . .a Yrmáxima;

En la siguiente figura se ilustra como quedarían formadas las matrices:



Donde:

(Xd, Yd) : Representan los datos de los puntos del mapa

y

(Xr, Yr) : Representan los puntos de la ventana.

Para poder hacer la transformación de coordenadas entre las ventanas y los mapas se utilizarán las siguientes ecuaciones:

$$\frac{Xr - Xrmin}{Xrmax - Xrmin} = \frac{Xd - Xdmin}{Xdmax - Xdmin} \quad \text{.....(1)}$$

$$Xr = \left(\frac{Xd - Xdmin}{Xdmax - Xdmin}\right)(Xrmax - Xrmin) + Xrmin \quad \text{.....(2)}$$

$$\frac{Yr - Yrmin}{Yrmax - Yrmin} = \frac{Yd - Ydmin}{Ydmax - Ydmin} \quad \text{.....(3)}$$

$$Yr = \left(\frac{Yd - Ydmin}{Ydmax - Ydmin}\right)(Yrmax - Yrmin) + Yrmin \quad \text{.....(4)}$$

En las ecuaciones (1) y (3) se hace una igualdad entre los datos del mapa y la ventana, con el propósito de hacer corresponder el tamaño de la ventana con el tamaño del mapa.

Las ecuaciones (2) y (4) realizan el despeje del punto mínimo o máximo en coordenadas X,Y para obtener su valor.

Un acercamiento se producirá cuando incrementemos los puntos mínimos y decrementemos los puntos máximos del mapa.

Las relaciones dadas por las ecuaciones (1) y (2) se deben de conservar para no deformar la figura del mapa dentro de la ventana, la cual se formará al restar y sumar los puntos máximos y mínimos de las matrices.

Realizando el proceso anterior, entonces se podrá relacionar la clase mapa con las otras clases para obtener la información respectiva de ese punto seleccionado en particular, pudiendo realizar la función de imprimir dicha gráfica del punto con su respectiva información.

El siguiente código es el que se encarga de ejecutar todo el proceso anterior. Para realizar los acercamientos (aumentos) se presionará el botón izquierdo del ratón:

```
1.  void CSiteView::OnLButtonDown(UINT nFlags, CPoint point)
2.  {
3.  CView::OnLButtonDown(nFlags, point);
4.  long xp, yp;                          //, x_inf, y_inf;
5.  long x, y, inc_x, inc_y, mid_x, mid_y;
6.  double fact_m;
8.  unsigned int cont = 0, largo;
9.  lugar_sit *i_inf;
10. DInfo dlg;
11. char *nom_com, *calle, *numero, *colonia, *cp, *ho1, *ho2;
12. char *ho3, *ho4, *pser, *cat, *tel, *rz, *prop, *rfc;
13. CClientDC dc( this );
14. CRect rect;
15. GetClientRect( rect );
16. xp = point.x;
```

```
17.   yp = point.y;
18.   x = (xp/derecho)*dif_x + min_x   ;
19.   y = (1 - yp/abajo)*dif_y + min_y ;
20.   if (Amplificar) {
21.   mid_x = (long) dif_x/2.0;              // implementación de la ecuaciones 1, 2 , 3,y 4
22.   mid_y = (long) dif_y/2.0;
23.   inc_x=mid_x*(acerca/2.0);
24.   inc_y=mid_y*(acerca/2.0);
25.   if (x - (mid_x - inc_x) < min_x) {
26.   max_x = min_x + (dif_x - 2*inc_x);
27.   } else {
28.   if (x + (mid_x - inc_x) > max_x) {
29.   min_x = max_x - (dif_x - 2*inc_x);
30.   } else {
31.   min_x = x - (mid_x - inc_x);
32.   max_x = x + (mid_x - inc_x);
33.   }
34.   };
35.   if (y - (mid_y - inc_y) < min_y) {
36.   max_y = min_y + (dif_y - 2*inc_y);
37.   } else {
38.   if (y + (mid_y - inc_y) > max_y) {
39.   min_y = max_y - (dif_y - 2*inc_y);
40.   } else {
41.   min_y = y - (mid_y - inc_y);
42.   max_y = y + (mid_y - inc_y);
43.   }
```

```
44.   };
45.   dif_x=max_x-min_x;
46.   dif_y=max_y-min_y;
47.   abajo=rect.bottom;
48.   derecho=rect.right;
49.   fact_m=dif_x/dif_y;
50.   if (abajo*fact_m>derecho)
51.   abajo= derecho/fact_m;
52.   else
53.   derecho=abajo*fact_m;
54.   *a_stack++ = (long) min_x;
55.   *a_stack++ = (long) max_x;
56.   *a_stack++ = (long) min_y;
57.   *a_stack++ = (long) max_y;
58.   limite = 0;
59.   InvalidateRect( rect ,TRUE);
60.   } else {
61.   i_inf = a_inf;
62.   while (i_inf->x!=FINAL) {
63.   if ((i_inf->giro==giro_act)   &&
64.   (abs((i_inf->x)-x)<=ancho) &&
65.   (abs((i_inf->y)-y)<=ancho)   ) {
66.   fsit.Seek(579+337*cont,CFile::begin);
67.   fsit.Seek(7,CFile::current);
68.   rz = (char*) malloc((largo=40)+1);
69.   fsit.Read(rz,largo);
70.   *(rz+largo) = '\0';
```

```
71.  dlg.m_rz = rz;
72.  colonia = (char*) malloc((largo=40)+1);
73.  fsit.Read(colonia,largo);
74.  *(colonia+largo) = '\0';
75.  dlg.m_colonia = colonia;
76.  calle = (char*) malloc((largo=40)+1);
77.  fsit.Read(calle,largo);
78.  *(calle+largo) = '\0';
79.  dlg.m_calle = calle;
80.  numero = (char*) malloc((largo=15)+1);
81.  fsit.Read(numero,largo);
82.  *(numero+largo) = '\0';
83.  dlg.m_numero = numero;
84.  cp = (char*) malloc((largo=5)+1);
85.  fsit.Read(cp,largo);
86.  *(cp+largo) = '\0';
87.  dlg.m_cp = cp;
88.  nom_com = (char*) malloc((largo=40)+1);
89.  fsit.Read(nom_com,largo);
90.  *(nom_com+largo) = '\0';
91.  dlg.m_nom_com = nom_com;
92.  rfc = (char*) malloc((largo=13)+1);
93.  fsit.Read(rfc,largo);
94.  *(rfc+largo) = '\0';
95.  dlg.m_rfc = rfc;
96.  prop = (char*) malloc((largo=40)+1);
97.  fsit.Read(prop,largo);
```

```
98.   *(prop+largo) = '\0';
99.   dlg.m_prop = prop;
100.  tel = (char*) malloc((largo=10)+1);
101.  fsit.Read(tel,largo);
102.  *(tel+largo) = '\0';
103.  dlg.m_tel = tel;
104.  ho1 = (char*) malloc((largo=4)+1);
105.  fsit.Read(ho1,largo);
106.  *(ho1+largo) = '\0';
107.  dlg.m_ho1 = ho1;
108.  ho2 = (char*) malloc((largo=4)+1);
109.  fsit.Read(ho2,largo);
110.  *(ho2+largo) = '\0';
111.  dlg.m_ho2 = ho2;
112.  ho3 = (char*) malloc((largo=4)+1);
113.  fsit.Read(ho3,largo);
114.  *(ho3+largo) = '\0';
115.  dlg.m_ho3 = ho3;
116.  ho4 = (char*) malloc((largo=4)+1);
117.  fsit.Read(ho4,largo);
118.  *(ho4+largo) = '\0';
119.  dlg.m_ho4 = ho4;
120.  pser = (char*) malloc((largo=50)+1);
121.  fsit.Read(pser,largo);
122.  *(pser+largo) = '\0';
123.  dlg.m_pser = pser;
124.  cat = (char*) malloc((largo=20)+1);
```

```
125.  fsit.Read(cat,largo);
126.  *(cat+largo) = '\0';
127.  dlg.m_cat = cat;
128.  dlg.DoModal();
   // lectura de las variables que desplegaran las ventanas de información
129.  free(nom_com);    free(calle);  free(numero);  free(colonia);
130.  free(cp);         free(ho1);    free(ho2);     free(ho3);
131.  free(ho4);        free(pser);   free(cat);     free(tel);
132.  free(rz);         free(prop);   free(rfc);
133.  };
134.  i_inf++;
135.  cont++;
136.  };
137.  };
138.  }
```

El siguiente código es el que se encarga de ejecutar las aciones del botón derecho del ratón cuya función es la de realizar los alejamientos y para poderlo hacer; pregunta si esta activada la función de Amplificar (activa los alejamientos) y desactivada la función de Examinar (activa el despliegue de las ventanas de información).

```
1.  void CSiteView::OnRButtonDown(UINT nFlags, CPoint point)
2.  {
3.  double fact_m;
4.  CView::OnRButtonDown(nFlags, point);
```

```
5.  CClientDC dc( this );
6.  CRect rect;
7.  if (examinar) return;   //pregunta si esta activada la opción de examinar
8.  GetClientRect( rect );
   //pregunta por la posición de los puntos máximos y minimos dentro de la pila
9.  if (a_stack-4<=stack) {
10. return;
11. };
   // Colocación de los puntos maximos y minimos dentro de la pila
12. a_stack-=4;
13. min_x = (long) *(a_stack-4);
14. max_x = (long) *(a_stack-3);
15. min_y = (long) *(a_stack-2);
16. max_y = (long) *(a_stack-1);
17. dif_x= (double) max_x-min_x;
18. dif_y= (double) max_y-min_y;
19. abajo= (double) rect.bottom;
20. derecho= (double) rect.right;
21. fact_m=dif_x/dif_y;
22. if (abajo*fact_m>derecho)
23. abajo= derecho/fact_m;
24. else
25. derecho= abajo*fact_m;
26. limite = 0;
27. InvalidateRect( rect ,TRUE);
28. }
```

Para regresar al aumento inicial de los mapas automáticamente, se toma el valor original del Máximo y Mínimo, inicializandolos y volviendo a dibujar. Para ello se utiliza el siguiente código:

```
1.  void CSiteView::OnUbicarCiudad()
2.  {
3.  CClientDC dc( this );
4.  CRect rect;
5.  GetClientRect( rect );
6.  limite = 1;                    // inicializa las variables a su valor original
7.  InvalidateRect( rect ,TRUE);
8.  }
```

8.- Para obtener la información que corresponde a los sitios, se selecciona antes, la opción de **examinar** del menú **Ubicar radio de acción;** esto, para poder desplegar la ventana que contiene dicha información y es cuando entonces se coloca el apuntador del ratón sobre uno de los puntos rojos (sitios) del mapa presionando el botón del mismo. Y si después se quiere hacer un aumento sobre el sitio seleccionado se tiene que seleccionar la opción de **Amplificar** del menú **Ubicar radio de acción.**

El siguiente código efectúa el proceso anterior:

```
1.  void CSiteView::OnUbicarciudadAmplificar()
2.  {
4.  amplificar = 1;  // se activa la opción de amplificar
5.  examinar = 0;  // se desactiva la opcion de examinar
6.
```

```
7.    }
8.    void CSiteView::OnUbicarciudadExaminar()
9.    {
11.   amplificar = 0; // se desactiva la opción de amplificar
12.   examinar = 1;    //se activa la opción de examinar
13.
14.   }
15.   void CSiteView::OnInformacintursticaAtractivostursticos()
   // visualización de la información de atracciones turísticas
16.   {
17.   DSelecc dlg;
18.   DCuadro dlg2;
19.   CFile temp;
20.   double tam;
21.   char *buffer;
22.   lugt_sit *l_apt;
23.   dlg.m_sitio = (ciudad)?"ZIHUATANEJO":"IXTAPA";
24.   dlg.m_lugt = a_lugt;
25.   if (dlg.DoModal()!=1) return;
26.   l_apt = a_lugt;
27.   while ((strcmp(dlg.m_lista,l_apt->nombre)!=0) && (l_apt->marca!=FINAL))
28.   l_apt++;
29.   if (l_apt->marca==FINAL) return;
30.   temp.Open( l_apt->archivo  ,CFile::modeRead);
31.   tam = temp.GetLength();
32.   buffer = (char*) malloc(tam+1);
33.   temp.Read(buffer,tam);
```

```
34.  temp.Close();
35.  *(buffer+(int)tam) = '\0';
36.  dlg2.m_titulo = dlg.m_lista;
37.  dlg2.m_texto = buffer;
38.  dlg2.DoModal();
39.  free(buffer);
40.  }
```

El siguiente código activa la información de Bienes:

```
41.  void CSiteView::OnInformacintursticaBienes()
42.  {
43.
44.  DBienSer dlg;
45.  desc_giro *g_apt;
46.  CClientDC dc( this );
47.  CRect rect;
48.  dlg.m_tipo = 0;
49.  dlg.m_giro = a_giro;
50.  dlg.m_titulo = (ciudad)?"ZIHUATANEJO - BIENES":"IXTAPA - BIENES";
51.  if (dlg.DoModal()!=1) return;
52.  g_apt = a_giro;
53.  while ((strcmp(dlg.m_lista,g_apt->nombre)!=0) && (g_apt->marca!=FINAL))
54.  g_apt++;
55.  if (g_apt->marca==FINAL) return;
56.  giro_act = g_apt->giro;
57.  GetClientRect( rect );
```

```
58.   limite = 0;
59.   InvalidateRect( rect ,TRUE);
60.   }
```

El siguiente código activa la información de Servicios:

```
61.   void CSiteView::OnInformacintursticaServicios()
62.   {
63.
64.   DBienSer dlg;
65.   desc_giro *g_apt;
66.   CClientDC dc( this );
67.   CRect rect;
68.   dlg.m_tipo = 1;
69.   dlg.m_giro = a_giro;
70.   dlg.m_titulo = (ciudad)?"ZIHUATANEJO - SERVICIOS":"IXTAPA -
      SERVICIOS";
71.   if (dlg.DoModal()!=1) return;
72.   g_apt = a_giro;
73.   while ((strcmp(dlg.m_lista,g_apt->nombre)!=0) && (g_apt->marca!=FINAL))
74.   g_apt++;
75.   if (g_apt->marca==FINAL) return;
76.   giro_act = g_apt->giro;
77.   GetClientRect( rect );
78.   limite = 0;
79.   InvalidateRect( rect ,TRUE);
      }
```

9.- Para limpiar la pantalla del mapa se inicializan las variables originales. Para ello se utilizó el siguiente código.

```
1.void CSiteView::OnUbicarLimpiar()
2.{
3.CClientDC dc( this );
4.CRect rect;
5.GetClientRect( rect );
6.giro_act = 0;      // regresa las variables a su valor original
7.GetClientRect( rect );
8.limite = 0;
9.InvalidateRect( rect ,TRUE);
```

10.- Para dibujar el mapa se hace uso de una pila cuya función es la de almacenar los valores máximos y mínimos del mapa, esto con el propósito de ir comparando sí los datos que se están leyendo de los archivos "i_datos.dbf" o "z_datos.dbf" se encuentran dentro de la pila para entonces graficarlos. Para empezar a graficar se coloca en la pila un valor mínimo para X,Y que es la coordenada 10,20.

El método que se sigue para dibujar el mapa es unir una serie de puntos(coordenadas) con rayas, cuando se empieza a dibujar una nueva serie de puntos se pregunta sí el valor de la variable **Marca** es igual **9999,** para entonces comenzar a dibujar una nueva linea, este valor se lee en los archivos "i_datos.dbf" o "z_datos.dbf". El código que se encarga de dibujar el mapa es el siguiente:

```
1.  // CSiteView drawing
2.  void CSiteView::OnDraw(CDC* pDC)
3.  {
4.  CSiteDoc* pDoc = GetDocument();
5.  CClientDC dc( this );
6.  CRect rect, sitio;
7.  CBrush          sit_brush;
8.  entero *a_apt;
9.  long xp, yp, x, y;
10. double fact_m;
11. byte flag = 1;
12. desc_sit *p_dat;
13. lugar_sit *i_inf;
14. GetClientRect( rect );
15. dc.SetTextAlign( TA_BASELINE | TA_CENTER );
16. dc.SetTextColor( ::GetSysColor( COLOR_WINDOWTEXT ) );
17. dc.SetBkMode(TRANSPARENT);
18. a_apt = a_lin;
19. if (limite) {
20. min_x = *a_apt++;
21. max_x = *a_apt++;
22. min_y = *a_apt++;
23. max_y = *a_apt++;
24. dif_x=max_x-min_x;
25. dif_y=max_y-min_y;
26. abajo=rect.bottom;
27. derecho=rect.right;
```

```
28.  fact_m=dif_x/dif_y;
29.  if (abajo*fact_m>derecho)
30.  abajo=derecho/fact_m;
31.  else
32.  derecho=abajo*fact_m;
33.  a_stack = stack;                    // Estas instrucciones guardan los limites en el
                                         //tope de la pila
34.  *a_stack++ = (long) min_x;
35.  *a_stack++ = (long) max_x;
36.  *a_stack++ = (long) min_y;
37.  *a_stack++ = (long) max_y;
38.  } else {
39.  a_apt+=4;
40.  limite = 1;
41.  };
42.  while (*a_apt != FINAL) {
43.  x = *a_apt++;
44.  y = *a_apt++;
45.  if ((x!=MARCA) && (y!=MARCA))  // se pregunta si el valor de X y Y es igual a
                                    // 9999, para comenzar a dibujar una nueva
                                    // serie de puntos
{
46.  flag = 1;
47.  } else {
48.  x = *a_apt++;
49.  y = *a_apt++;
50.  flag = 0;
```

```
51.   };
52.   trans(x,y,&xp,&yp);
53.   if (flag) {dc.LineTo( (int) xp, (int) yp);}
54.   else  {dc.MoveTo( (int) xp, (int) yp);};
55.   };
56.   /*
57.   POINT pt;
58.   pt.x = 10;                    //valor X minímo inicial
59.   pt.y = 20;                    // valor Y minímo inicial
60.   CClientDC dc(this);
61.   dc.SetPixel(pt.x,pt.y,RGB(255,0,0));
62.   */
63.   //dc.SelectStockObject(GRAY_BRUSH);
64.   p_dat = a_dat;
65.   while (p_dat->x!=FINAL) {
66.   x   = p_dat->x;
67.   y   = p_dat->y;
68.   if (min_x<=x && x<=max_x && min_y<=y && y<=max_y)  // pregunta si el punto
                                                        //se encuentra dentro de
                                                        //la pantalla para
                                                        //entonces graficarlo.
{
69.   trans(x,y,&xp,&yp);
70.   dc.TextOut( (int) xp, (int) yp, p_dat->titulo, 30 ); // despliega el titulo del lugar
71.   };
72.   p_dat++;
73.   };
```

```
74.  i_inf = a_inf;
75.  while (i_inf->x!=FINAL) {
76.  x = i_inf->x;
77.  y = i_inf->y;
78.  if (i_inf->giro==giro_act && min_x<=x && x<=max_x &&//pregunta si el
                                                        //punto se
                                                        //encuentra dentro
                                                        //de la pantalla
                                                  //entonces se rellena el punto rojo que
                                                  //indica la ubicación.
79.  min_y<=y && y<=max_y) {
80.  trans(x,y,&xp,&yp);
81.  dc.Arc( (int) xp-ancho, (int) yp-ancho, (int) xp+ancho,
82.  (int) yp+ancho, (int) xp-ancho, (int) yp,
83.  (int) xp-ancho, (int) yp);
84.  };
85.  i_inf++;
86.  };
87.  sit_brush.CreateSolidBrush(RGB(255,60,70));
88.  dc.SelectObject(&sit_brush);
89.  i_inf = a_inf;
90.  while (i_inf->x!=FINAL) {
91.  x = i_inf->x;
92.  y = i_inf->y;
93.  if (i_inf->giro==giro_act && min_x<=x && x<=max_x &&
94.  min_y<=y && y<=max_y) {
95.  trans(x,y,&xp,&yp);
```

```
96.   dc.FloodFill( (int) xp, (int) yp,RGB(0,0,0));
97.   };
98.   i_inf++;
99.   }
100.  /*
101.  dc.SetPixel(xp,abajo-yp,RGB(255,0,0));
102.  dc.TextOut( ( rect.right / 2 ), ( rect.bottom / 2 ),s, s.GetLength() );
103.  CSiteView::OnDraw(&dc);
```

11.- Las instrucciones que se utilizaron para unir los comandos del menú y de los botones de la barra de herramientas con las funciones implementadas en las clases son las siguientes:

```
/////////////////////////////////////////////////////////////////////////////
// CSiteView message handlers
1.  // CSiteView
2.  IMPLEMENT_DYNCREATE(CSiteView, CView)
3.  BEGIN_MESSAGE_MAP(CSiteView, CView)
4.  //{{AFX_MSG_MAP(CSiteView)
5.  ON_COMMAND(ID_UBICAR_CIUDAD, OnUbicarCiudad)
6.  ON_WM_LBUTTONDOWN()
7.  ON_COMMAND(ID_CIUDAD_IXTAPA, OnCiudadIxtapa)
8.  ON_COMMAND(ID_CIUDAD_ZIHUATANEJO, OnCiudadZihuatanejo)
9.  ON_WM_RBUTTONDOWN()
10. ON_COMMAND(ID_UBICAR_RADIODEACCIN, OnUbicarRadiodeaccin)
11. ON_COMMAND(ID_SITE_CONOCER, OnSiteConocer)
```

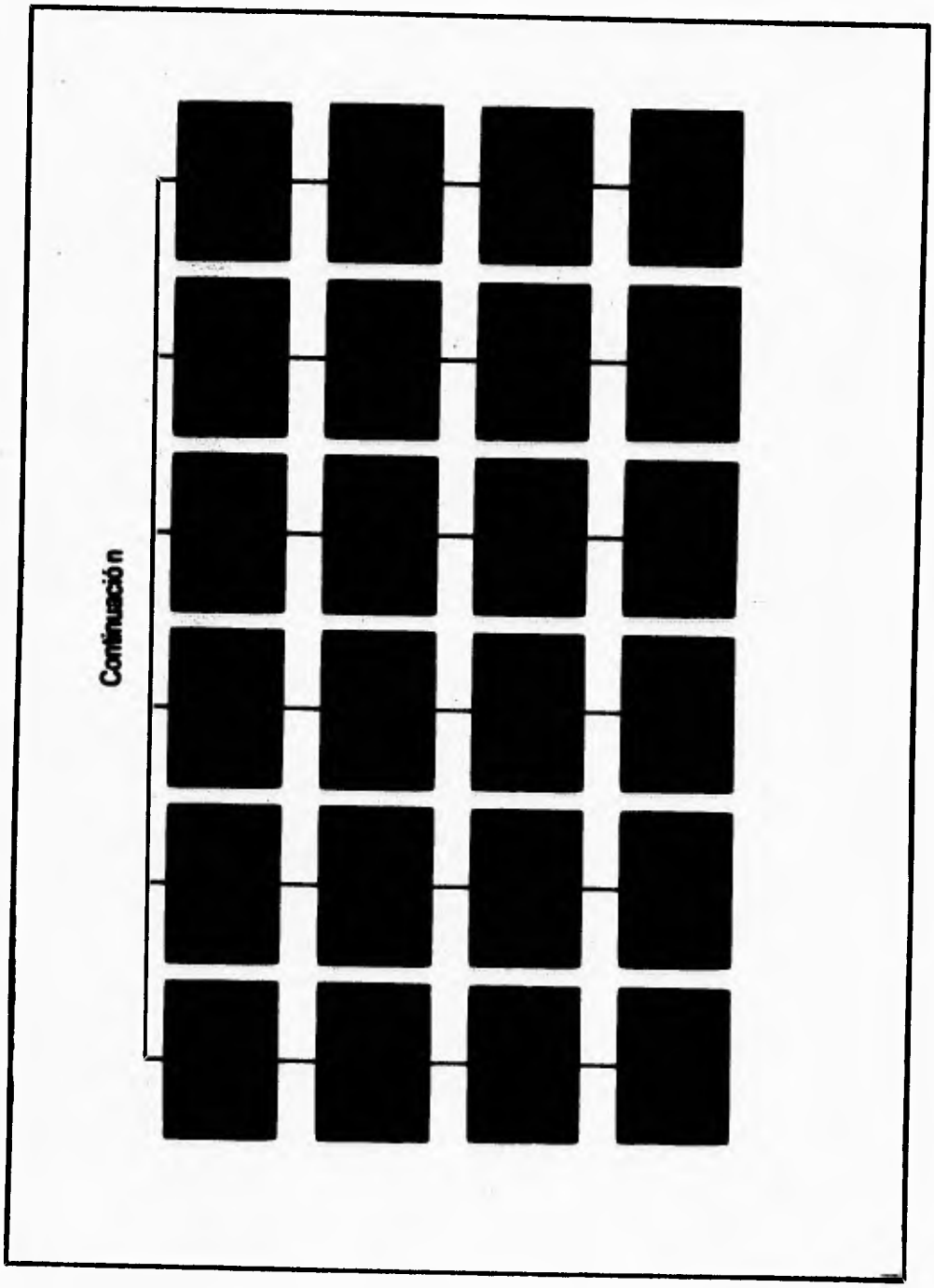
```
12.  ON_COMMAND(ID_INFORMACINTURSTICA_ATRACTIVOSTURSTICOS,
     OnInformacintursticaAtractivostursticos)
13.  ON_COMMAND(ID_INFORMACINTURSTICA_SERVICIOS,
     OnInformacintursticaServicios)
14.  ON_COMMAND(ID_INFORMACINTURSTICA_BIENES,
     OnInformacintursticaBienes)
15.  ON_COMMAND(ID_UBICAR_LIMPIAR, OnUbicarLimpiar)
16.  ON_COMMAND(ID_UBICARCIUDAD_AMPLIFICAR,
     OnUbicarciudadAmplificar)
17.  ON_COMMAND(ID_UBICARCIUDAD_EXAMINAR,
   OnUbicarciudadExaminar)
18.  //}}AFX_MSG_MAP
20.  ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
21.  ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()
```

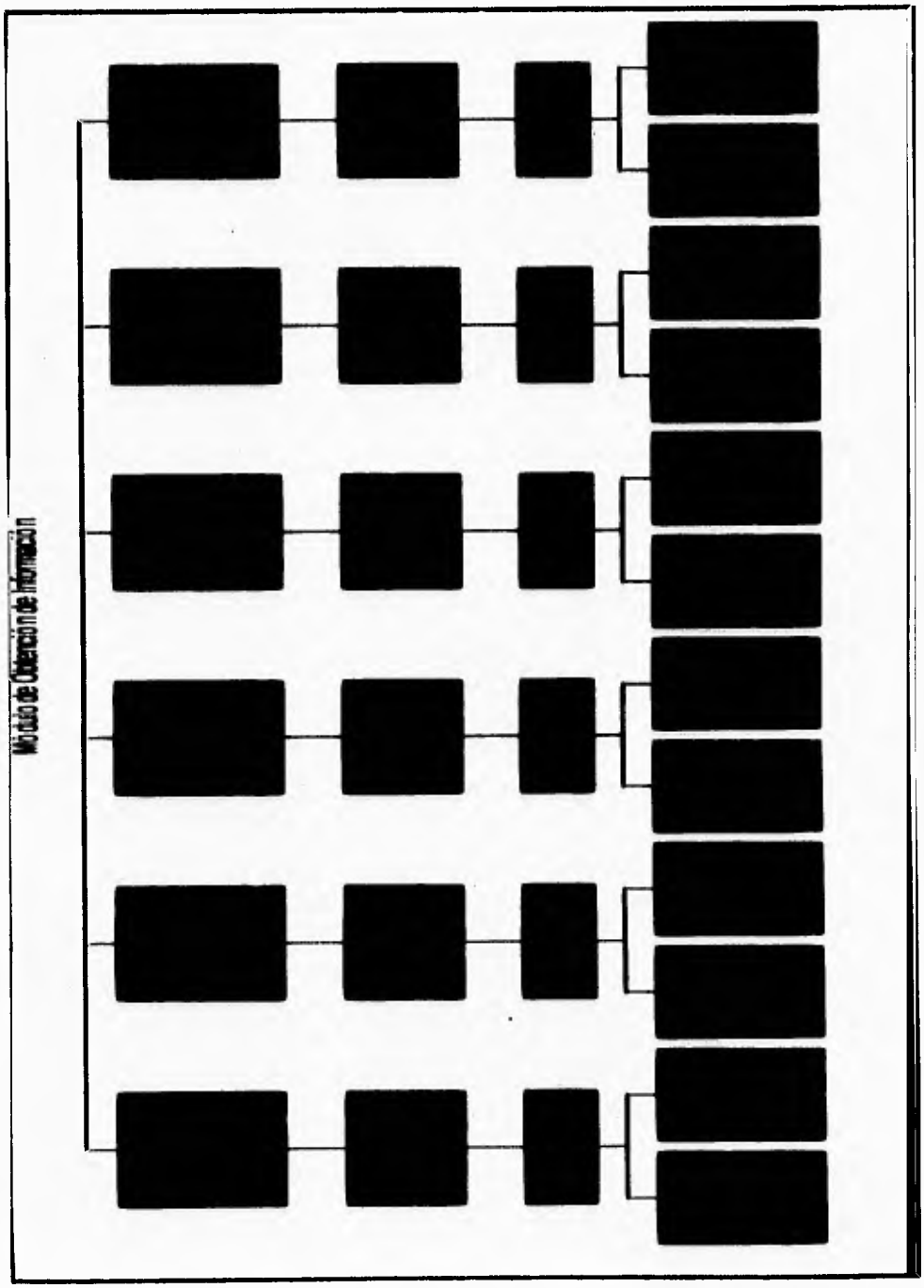
El diseño de las ventanas que despliegan la información se describirá en el siguiente punto de este capítulo.

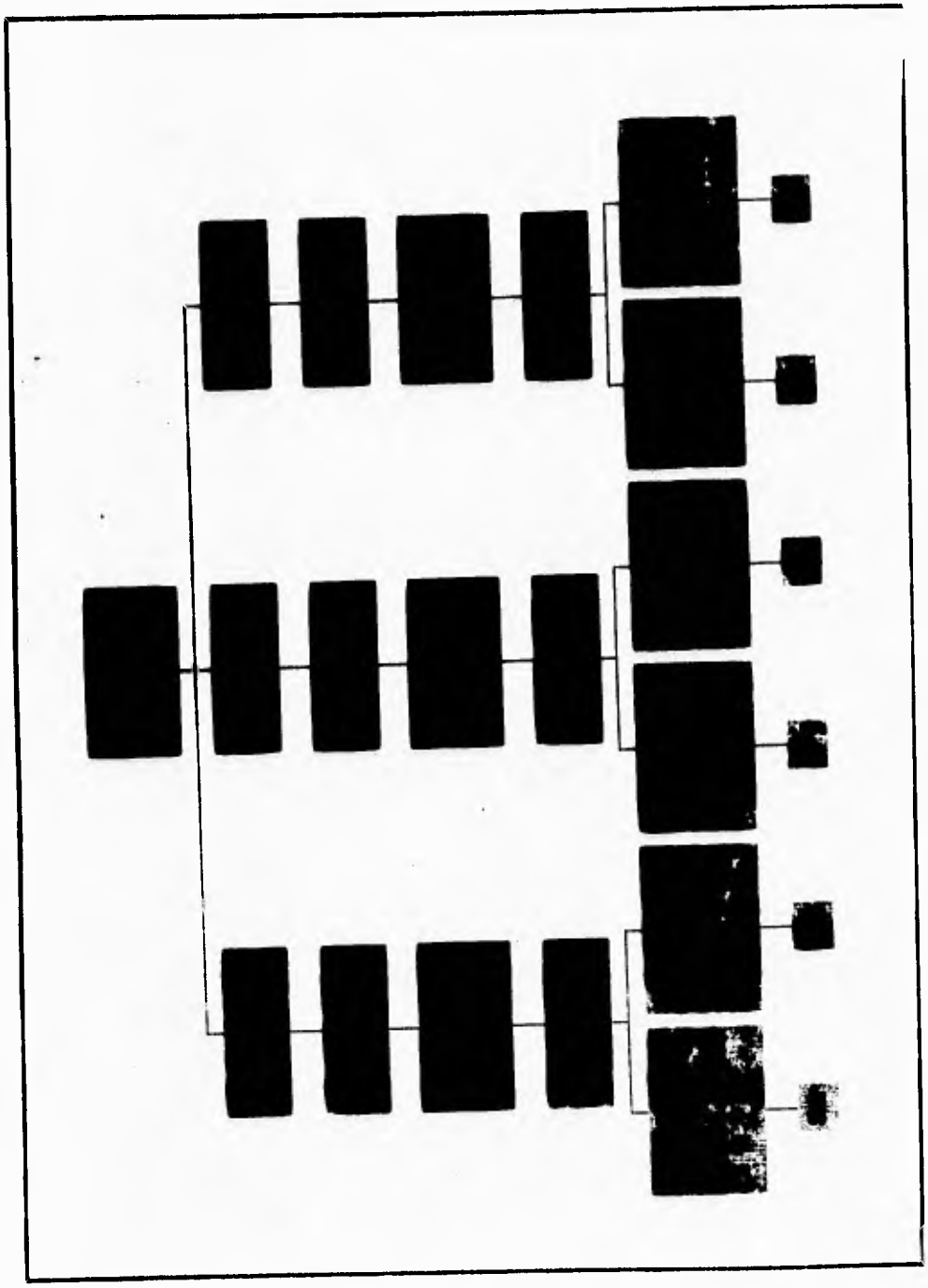
La implementación del archivo CSiteView completo se puede consultar el apéndice que se encuentra en la parte final de esta capítulo.

El siguiente cuadro describe de manera modular la realización de este sistema.


Diagrama Modular


Continuación


Módulo de Obtención de Información



3.1.7 DISEÑO DE LAS PANTALLAS DE CONSULTA.

En este capítulo se ilustra como fueron realizadas las pantallas que el sistema va a presentar al usuario y con las que va a interactuar. Para el diseño de estas, se hizo uso principalmente de una de las herramientas que provee Visual C++, que es **App Studio**. Esta aplicación nos permite desarrollar la presentación del sistema, la cual se compone de una barra de menú, una barra de herramientas compuesta de iconos; que son figuras y dibujos que expresan la función que realizan y las ventanas que van a desplegar la información.

Para el diseño de la pantalla principal, que es la que va a desplegar las barras de menú y herramientas se utilizaron los siguientes procedimientos:

Barra de menú

1.- Primeramente del archivo que genero App Wizard Site.rc (Resource Script) que es el archivo encargado de mantener toda la información de la presentación del sistema, se teclea el código que forma la barra de menú. El símbolo ampersan (&) se coloca antes de una letra del nombre del comando (&Ayuda) para que en la pantalla que se le presente al usuario aparezca el nombre del comando con una de sus letras subrayadas lo que le indica cual es la llave de acceso rápido al comando, por ejemplo en el comando llamado **Ayuda** se encuentra subrayada la letra A lo cual indica que el usuario accesa a ese comando simplemente presionando la tecla ALT y la tecla que contiene la letra A.

El código es el siguiente:

```
IDR_MAINFRAME MENU PRELOAD DISCARDABLE

BEGIN
  POPUP "&Site"
  BEGIN
    MENUITEM "&Acerca",                    ID_APP_ACERCA
    MENUITEM "&Conocer",                   ID_SITE_CONOCER
    MENUITEM SEPARATOR
    MENUITEM "&Salida",                    ID_APP_SALIR
  END
  POPUP "&Ciudad"
  BEGIN
    MENUITEM "&Ixtapa",                ID_CIUDAD_IXTAPA
    MENUITEM "&Zihuatanejo",           ID_CIUDAD_ZIHUATANEJO
  END
  POPUP "&Ubicar"
  BEGIN
    MENUITEM "&Ciudad",                    ID_UBICAR_CIUDAD
    MENUITEM "&Radio de Acción",               ID_UBICAR_RADIODEACCIN
    MENUITEM SEPARATOR
    MENUITEM "&Limpiar",                   ID_UBICAR_LIMPIAR
  END
  POPUP "Información &Turística"
  BEGIN
    MENUITEM "&Bienes",                    ID_INFORMACINTURSTICA_BIENES
    MENUITEM "&Servicios",             ID_INFORMACINTURSTICA_SERVICIOS
```

```
        MENUITEM SEPARATOR
        MENUITEM "&Atractivos Turísticos",
                                ID_INFORMACINTURSTICA_ATRACTIVOSTURSTICOS
    END
    MENUITEM "&Quejas",             65535
    MENUITEM "&Ayuda",              65535
END
endif #
```

2.- Después hay que compilarlo en la opción Compile del menú Project.

3.- Y por último se tiene que ligar con los demás archivos del programa (archivos con las extensiones .cpp , .h, y .def) para elaborar el archivo ejeculable(.Exe).

Barra de Herramientas

La barra de herramientas es también un menú como el anterior, pero en forma gráfica ya que consta de iconos (botones que muestran mediante un dibujo una función específica del sistema), esto con el motivo de que se prevee que el sistema será utilizado tanto por turismo nacional como extranjero, entonces se pretende hacer un sistema que sea entendible a cualquier individuo no importando el idioma que hable. Para realizar la barra se hizo lo siguiente:

1.- Desde la pantalla Workbench del Visual C++ se elige el comando AppStudio que se encuentra en el menú Tools.

2.- En la pantalla que despliega, se elige la opción Open del menú File y se escoge el archivo con la extensión rc. Ahí se presenta una pantalla que despliega dos botones de los cuales elegimos el botón de New para elaborar los iconos que van a formar la barra de herramientas.

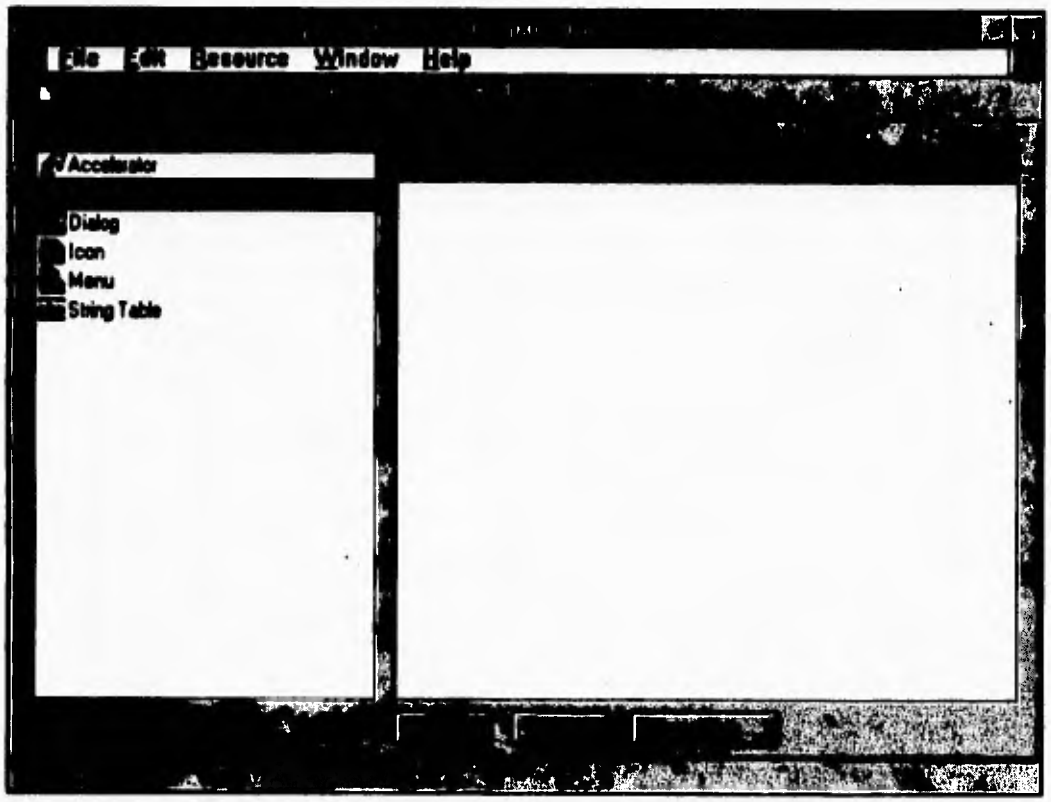


Fig. 3.1.7.1 Pantalla de App Studio.

3.- Se procede a dibujar cada uno de los iconos de la barra, cuidando que la imagen sea alusiva a la función que queramos que realice, para este proceso utilizamos la paleta de herramientas que el visual C + + nos proporciona para ayudarnos a realizar el dibujo.

4.- Una vez terminado el proceso anterior, se salva el archivo y se añade la siguiente estructura en el archivo MainFrame.cpp

```
static UINT BASED_CODE buttons[] =
// lista de las funciones que realiza el sistema
        {
    ID_ABRIR
    ID_CERRAR
    ID_LIMPIAR
    ID_FUNCION_ESPECIFICA
    ...
    ...
    ...
};
```

Tal estructura relaciona los botones de las imágenes con sus respectivos comandos definidos como estructuras ID 's

Nota: Hay que cuidar el orden en que se agregan los comandos porque los va relacionando de acuerdo a como están ordenados los iconos en la barra de herramientas.

5.- Finalmente se compila el archivo y se reconstruye todo el programa en la opción Rebuilt del menú Project, para elaborar el archivo ejecutable.

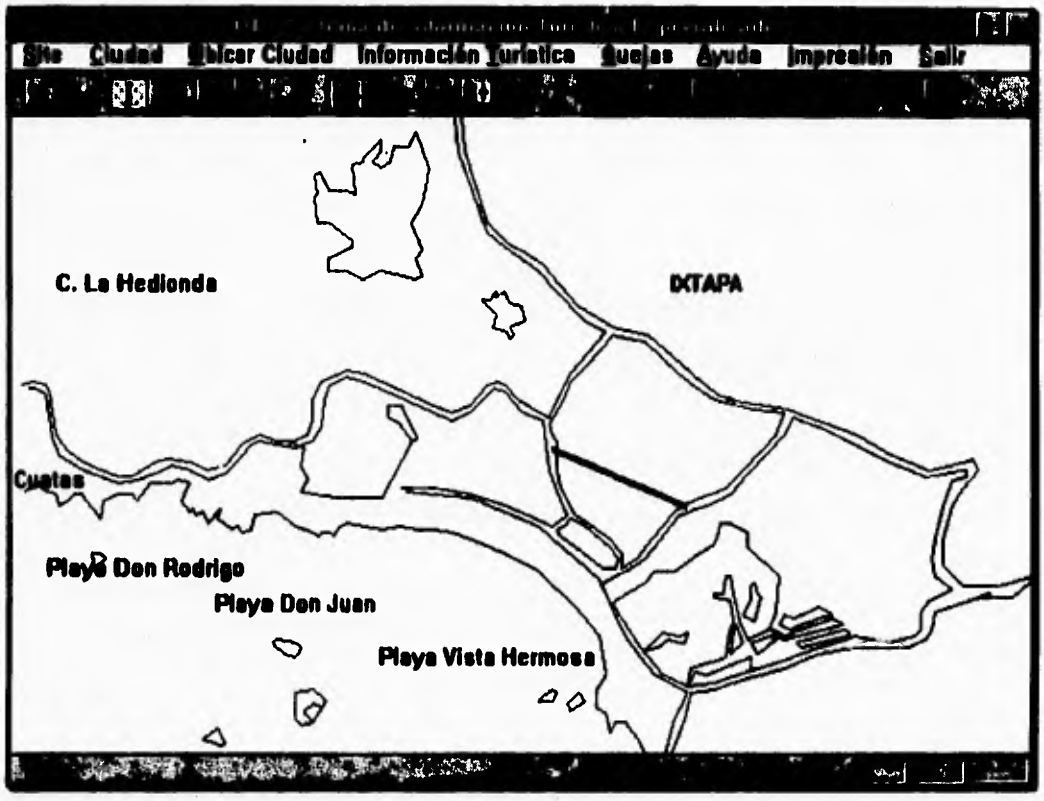


Fig. 3.1.7.2 Visualización de la barra de herramientas

Cajas de dialogo

El término "Cajas de diálogo" es un nombre erróneo ya que estas son ventanas especiales que contienen controles que sirven para desplegar información o entrada de datos. Las aplicaciones de Windows utilizan las cajas de dialogo para intercambiar información con el usuario.

Existen dos tipos de cajas de dialogo las Modal y Modeless, las cajas de dialogo Modal requieren que se cierre la caja antes de continuar con la aplicación, ya que de no hacerlo significaría ejecutar un intercambio de datos critico, de hecho las cajas de dialogo desactivan su ventanas anteriores mientras ellas están abiertas, en

que de no hacerlo significaría ejecutar un intercambio de datos critico, de hecho las cajas de dialogo desactivan su ventanas anteriores mientras ellas están abiertas, en cambio las cajas de dialogo Modeless no necesitan estar cerradas para continuar utilizando la aplicación.

Existe una biblioteca de MFC declarada como clase CDialog que maneja ambas cajas de dialogo la Modeless y Modal. La clase CDialog es una clase derivada de la clase Cventana, tiene un constructor y un número de funciones miembro, incluyendo la función Create La clase CDialog esta declarada como protegida para obligarnos a crear nuestras propias cajas de dialogo de tipo Modeless. Estas clases derivadas pueden declarar ciertas clases para suplementar nuestras cajas de dialogo como son:

- Declarar únicamente la clase constructor como publica ya que esta invoca a la función CDialog::Create, esta característica crea la caja de dialogo en un paso.
- Declarar ambas funciones la constructor y la create. Este esquema crea instancias de la caja de dialogo en dos pasos, primero llamando la clase constructor y después invocando la función Create.

La declaración de la función CDialog::Create es:

BOOL Create (LPCSTR lpTemplateName, CVentana* pParentVentana = NULL);

BOOL Create (UINT nIDTemplate, CVentana* pParentVentana = NULL);

Los parámetros LpTempalte y nIDTemplate son de la caja de dialogo y el comando ID respectivamente. El parámetro pParentVentana es el apuntador a la ventana raíz.

Ejecución de las cajas de dialogo Tipo Modal

Comúnmente, las cajas de dialogo tipo Modal son creadas más frecuentemente que las cajas de dialogo Modeless y mucho más frecuentemente que las ventanas, para la ejecución de las cajas de dialogo modal se involucran los siguientes pasos:

1.- Crear una instancia de la caja de dialogo utilizando la caja de dialogo de la clase constructor.

2.- Llamar a la función DoModal(), declarada en la clase CDialog, para traer la caja de dialogo. Comúnmente, la caja de dialogo contiene dos botones por defecto que son OK y Cancel estos botones tienen predefinido a los comandos ID's IDOK e IDCANCEL, respectivamente. se pueden utilizar los controles de los botones con diferentes mensajes que son OK y CANCEL, sin embargo es mejor mantener IDOK e IDCANCEL con esos nombres de los botones. Utilizar los IDs permite tomar ventaja de las estructuras ya que responde automáticamente a IDOK e IDCANCEL provistas por las funciones OnOK y OnCancel definidas en la clase de la caja de dialogo. Presionando el botón OK comúnmente indica que se acepta el dato corriente en la caja de dialogo. Y en caso contrario presionando el botón de cancel indica que se esta en desacuerdo con el dato corriente y por lo tanto no lo efectúa el proceso con el cual esta relacionado. La declaración de la función DoModal( ) es int DoModal ( ); la función regresa un entero que representa la salida. Comúnmente los valores de salida son IDOK o IDCANCEL.

3.- Se comparan los resultados de la función DoModal con IDOK(o con IDCANCEL). La salida de esta comparación determina los pasos que toma, tales pasos involucra el acceso de datos que se introdujeron en los controles de las caja de dialogo.

Las funciones OnInitDialog, OnOk y OnCancel manejan la ejecución de la caja de dialogo del tipo Modal. Esos parámetros están declarados como funciones virtuales. La función OnInitDialog sirve para inicializar la caja de dialogo y sus controles. La declaración de OnInitDialog es:

virtual BOOL OnInitDialog();

La función regresa el resultado boleano para indicar la inicialización de la caja de dialogo. Comúnmente la función OnInitDialog inicializa los controles de la caja de dialogo. Esta inicialización usualmente copia los datos de los bufffer.

La función OnOk se maneja presionando el botón de Ok. La declaración de la función OnOk es

virtual void OnOk( );

La función OnOk sirve para copiar datos de los controladores de la caja de dialogo a los buffers. Utilizando la función EndDialog, la función OnOk permite proveer la función DoModal regresando su valor. Usualmente la ultima estructura en la función OnOk la definición es EndDialog(IDOK). Después la función llamada origina que la función DoModal regresa a IDOK.

La función OnCancel es manejada presionado el botón cancel. La declaración de la función OnCancel es

virtual void OnCancel ( );

La función OnCancel sirve para limpiar antes de que la caja de dialogo sea cerrada, la cual involucra el cierre de archivos, por ejemplo, Usualmente, la última estructura de la función OnCancel es EndDialog (IDCANCEL). Después la función llamada origina que la función DoModal regrese a IDCANCEL.

Interactuar con ventanas frecuentemente involucra utilizar cajas de dialogo que contienen varios tipos de controles, tales como caja de edición y botones de control(pushbutton). Esos controles pueden ser incluidos en ventanas o más frecuentemente en cajas de dialogo. Los controladores pueden ser implementados como se explica a continuación:

Control de texto

El control de texto provee una ventana o una caja de dialogo con texto estático. La clase Cstatic implementa ese control.

La clase Cstatic es derivada de la clase ventana y ofrece un texto estático que esta definida por un área de display, texto a un display y atributos de texto.

sintaxis:

```
BOOL Create(const char FAR* lpText,          // Control de texto
            DWORD dwStyle,                   // Control de estilo
            const RECT& rect,                // Control de área
            Cventana* pParentVentana,        // Ventana Raíz
            UINT nID = 0Xffff),              // Control ID
```

El parámetro lpText especifica el texto del texto de control estático, el parámetro dwStyle designa el control de estilo. Los argumentos del estilo típicamente incluyen los estilos WS_CHILD y WS_VISIBLE. La creación de esos controles incluyen uno o más de esos estilos, el parámetro Rect especifica el área que se va a ocupar de tamaño rectangular, el parámetro pParentVentana es el apuntador a la ventana raíz, el parámetro nID especifica el control ID. La función Create asigna un parámetro por defecto el 0xffff, porque un control de texto estático comúnmente no envía mensajes a su ventana raíz.

El Control Pushbutton

El control PushButton es quizá sociológicamente el control más poderoso, ya que por costumbre sabemos que cuando presionamos un botón de control intuimos que va suceder algo.

Existen básicamente dos tipos de botones, los botones por defecto y los botones de nodefecto. Para manejar estos dentro de las ventanas hay que colocarlos primero dentro de una clase llamada Cbutton que es una clase derivada de la clase ventana, y en su implementación se hace uso de dos funciones que son de una relevante importancia.

La función Create

```
BOOL Create(const char FAR* lpCaption,        //Etiqueta del botón
            DWORD dwStyle,                    // Control de estilo
            const RECT& rect,                 // Control de área
            Cventana* pParentVentana,         // Ventana Raíz
            UINT nID );                       // Control ID
```

El parámetro lpCaption especifica la etiqueta del botón es decir, el nombre de la función que ejecuta, el parámetro dwStyle designa el tipo de botón. Los estilos comúnmente utilizados son WS_CHILD, WS_VISIBLE., BS_PUSHBTTON Y WS_TABSTOP, el parámetro Rect especifica el área que se va a ocupar de tamaño rectangular y la posición, el parámetro pParentVentana es el apuntador a la propia ventana, el cual puede estar junto a una ventana o a una caja de dialogo, el parámetro nID especifica el control ID.

El visual C++ incluye la aplicación App Studio que permite crear cajas de dialogo y los botones de control de estas, pero es conveniente primero aprender como funciona el archivo con la extensión: RC para hacer más fácil el trabajo con AppStudio.

La implementación del archivo CMainframe se encuentra en el apéndice B al finalizar este capítulo.

3.2 Documentacion.

3.2.1 MANUAL DEL USUARIO

SITE Sistema de Información Turística Especializado este sistema nos proporciona Información Turística del puerto de Ixtapa Zihuatanejo.

Para empezar con Windows y SITE

Para comenzar a trabajar con **SITE**, siga estos pasos:

1.- Escriba WIN en el indicador de comandos de DOS (Por lo general C: >) y oprima Enter.

2.-Active la ventana del grupo donde se encuentra **SITE** Para Windows. Esta ventana es la del grupo en donde se haya creado el elemento del programa de **SITE**.

La figura 3.2.1 muestra la ventana del grupo **SITE** con el elemento del programa **SITE** para Windows seleccionados. La ventana del grupo puede estar abierta en su pantalla, como la del grupo Main (Principal) y la del grupo **SITE** de la figura 3.2.1, o bien cerrada, como los iconos de los cuatro grupos en la esquina inferior de la ventana del Program Manager (administrador de programas) de la figura 3.2.1.

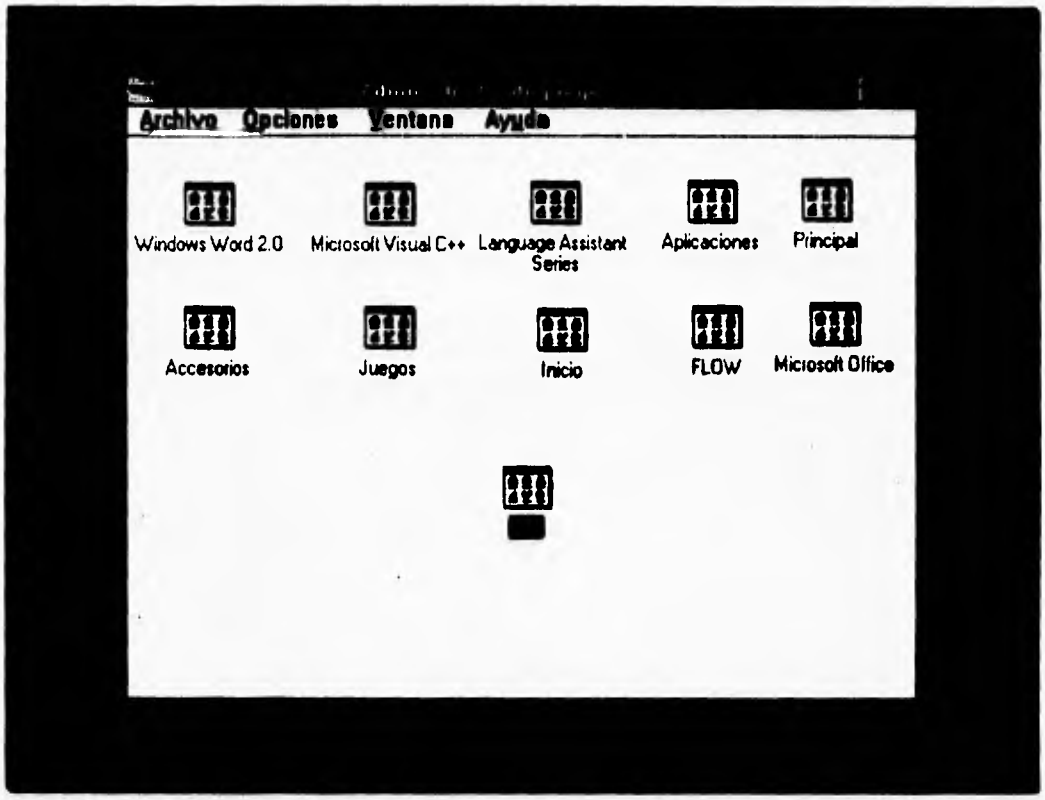


Fig. 3.2.1 Ventana del grupo SITE

Ratón: Pulse sobre la ventana abierta del grupo o pulse dos veces sobre el grupo cerrado de iconos que contenga el elemento del programa de **SITE** para Windows.

Teclado: Oprima Ctrl +Tab hasta que se active la ventana abierta del grupo que se desea o hasta que resulte el título del icono del grupo cerrado. Presione Enter para llevar el icono del grupo cerrado. Presione para llevar el icono del grupo a la ventana.

3.- Inicie **SITE** para Windows.

Ratón: Pulse dos veces sobre el icono del elemento del programa **SITE** para Windows.

Teclado: Oprima las teclas de dirección hasta que resalte el icono del elemento del programa **SITE** para Windows y en seguida oprima Enter.

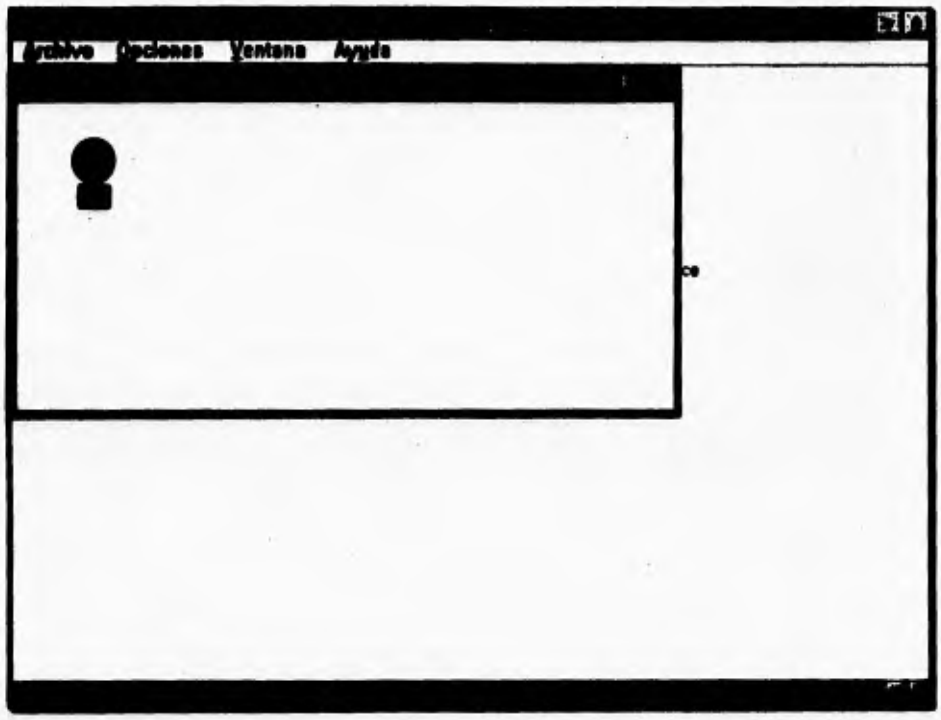


Fig. 3.2.2 Iniciación de SITE.

Si está en el indicador de comandos de DOS (por lo general C:>) y desea iniciar Windows y **SITE** para Windows al mismo tiempo, haga lo siguiente:

1.- Escriba WIN C:\SITE\SITE.EXE

2.- Oprima Enter

Para comprender la pantalla de SITE

Si utiliza por primera vez las aplicaciones de Windows, debe aprenderse las partes que forman la pantalla. Estudie la figura 3.2.3 y la tabla 3.2.1. para conocer las partes de aplicación y de documento de la pantalla .

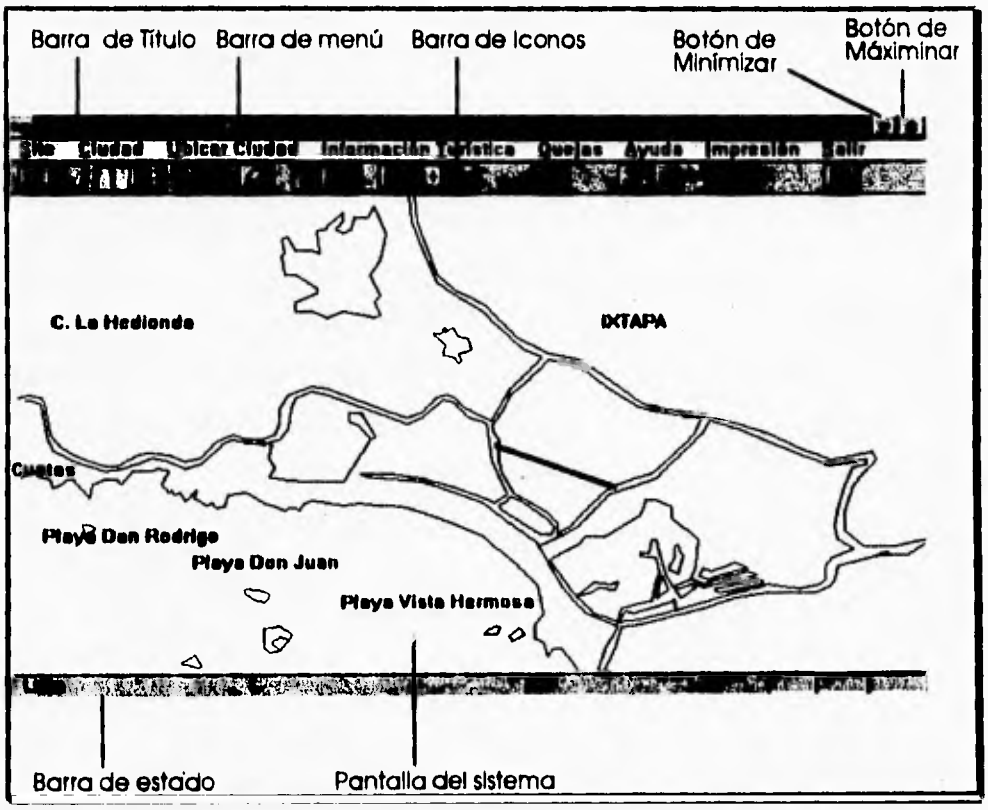


Fig. 3.2.2 Ventana Principal.

Tabla 3.2.1 Componentes de la pantalla de SITE

Término	Aplicación
Ventana de Aplicación	La Ventana dentro de la que corre **SITE**. Si esta utilizando Windows, puede ejecutar varias aplicaciones de Windows, cada una en su propia ventana. Una ventana de aplicación puede ocupar toda la ventana.
Ventana de documento	La ventana en la que aparece cada documento.
Menú de control de la aplicación	El menú utilizado para el manejo de la ventana de aplicación. Para activar su menú, oprima Alt, barra espaciadora o sobre el icono de control.
Menú del control del documento	El menú que se utiliza para el manejo de la ventana del documento activo. Para activar su menú, oprima Alt, guión (-) u oprima el botón del ratón sobre el icono de control.
Barra de Título	La barra de la parte superior de una ventana, que contiene el título de la aplicación o del documento.

Ventana Activa	La ventana que acepta entradas y comandos. La ventana cuya barra de título aparece sólida es por lo general la ventana del frente.
Ventana Inactiva	Cualquier ventana con la que no se este trabajando, cuya barra de título aparece sombreada o en tono gris.
Apuntador del ratón	Apuntador en la pantalla que muestra la posición del ratón.
Punto de Inserción	Punto donde aparece el texto cuando se escribe.
Barra de menúes	Lista horizontal de nombres de menúes, exhibida detrás de la barra del título de la aplicación
Barra de iconos	Barra de contiene botones y cajas de texto, que indican la dos ciudades a escoger, los acercamientos, las opciones de bienes, servcios y atracciones turísticas.
Icono para minimizar	El equivalente simbolico del comando Minimize, que reduce una ventana a un icono.
Icono para Maximizar	El equivalente simbolico del comando Maximize, que aumenta el tamaño de una ventana a toda la pantalla.

Barra de recorrido	Barra horizontal o vertical con flechas, se utiliza para recorrer el documento con el control del apuntador del ratón.
Barra de estado	Barra en la parte inferior de la pantalla, que muestra los icono o indicadores de comandos

Cómo usar la barra de iconos

La barra de iconos le da acceso directo a la mayoría de los comandos de uso más frecuente con el ratón sin pasar por los menúes.

La tabla 3.2.2 resume la funciones de cada uno de los botones que aparecen en la barra de iconos.

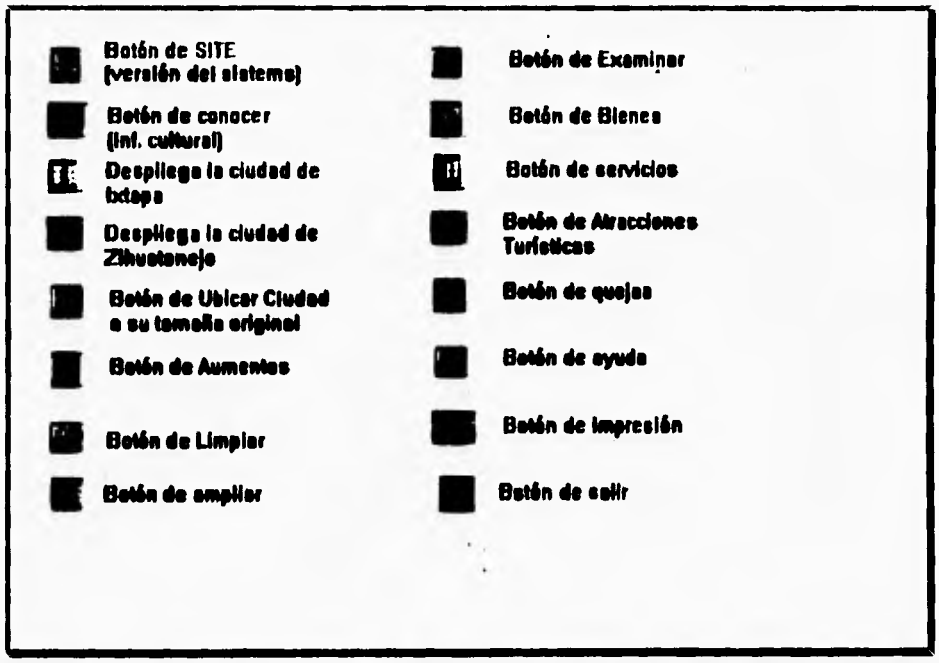


Tabla 3.2.2.

Cómo usar el ratón

El ratón o mouse es un dispositivo manual que, al moverse desplaza un apuntador en la pantalla. los botones permiten hacer indicaciones a **SITE** cuando el ratón esta bien colocado en un menú o comando. Cuando oprime uno de los botones del ratón, se produce alguna acción, correspondiente a la posición del apuntador.

Cómo mover el ratón

El ratón que se muestra en la figura 3.2.3 es un pequeño dispositivo que acomoda a la palma de su mano. Al mover el ratón en su escritorio, el apuntador del ratón, que por lo general es una flecha, se mueve en la misma dirección relativa en la pantalla. Notará que el uso del ratón para "apuntar" a un elemento de la pantalla resulta un proceso natural.

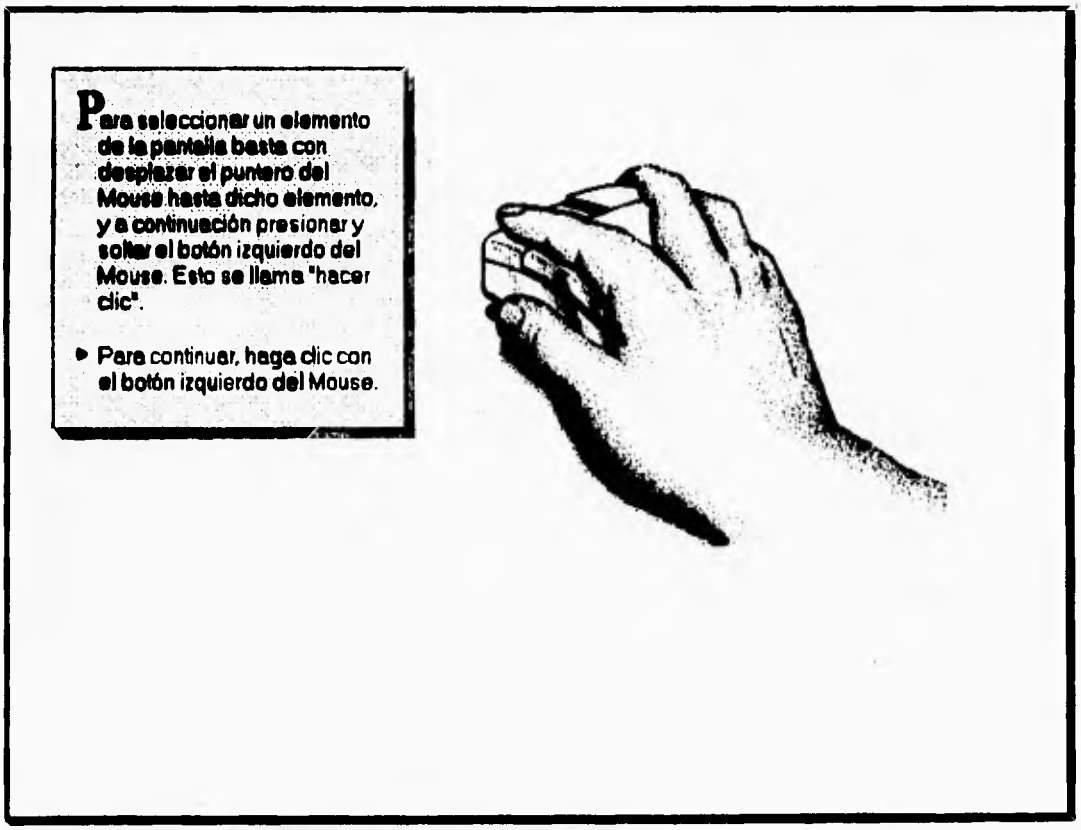


Fig 3.2.3 El ratón.

Cuando utilice el ratón, el cable debe quedar hacia adelante, lejos de su brazo, de modo que los botones estén debajo de sus dedos. Asegurese de mantener el aparato orientado, de manera que al alejar el ratón de usted, el apuntador se mueva hacia arriba en la pantalla. Este método hace que el ratón sea fácil de usar. Un ratón tiene dos o tres botones; sin embargo **SITE** sólo utiliza dos. El botón izquierdo será el de uso más frecuente.

Cómo hacer una selección con el ratón

Se utilizan dos técnicas básicas para la selección de elementos con el ratón:

- Se oprime el botón una vez para seleccionar un elemento, como un menú, comando, celda u objeto gráfico. Al oprimir el botón en alguna posición de la pantalla, como una palabra, se obtiene el acceso rápido a un comando.
- Oprima dos veces el botón para seleccionar una opción de una caja de diálogo y elegir al mismo tiempo la opción de aprobación. Este procedimiento, realizado en algunas regiones de la pantalla, genera un acceso rápido a una selección. Para esto, mueva el apuntador del ratón sobre el elemento y oprima de manera rápida dos veces el botón izquierdo.

Para elegir un comando con el ratón, oprima el botón en el nombre del menú en la barra de menúes y después sobre el nombre del comando en el menú descendente.

Cómo usar el teclado

SITE para Windows no se limita al uso del ratón. Con el teclado puede ejecutar comandos o hacer selecciones de dos maneras:

- Oprimir la teclas de dirección para elegir menúes y comandos.
- Oprimir las teclas de función.

Cómo elegir comandos de menú con el teclado

En la barra de menués, el nombre de cada menú tiene una letra subrayada que se utiliza para activarlo desde el teclado. Cada comando en un menú también tiene sólo una letra subrayada, que se utiliza para activarlo.

Para elegir un comando desde un menú con el método de las combinaciones de teclas, siga estos pasos:

1. Oprima la tecla Alt para activar la barra de menúes.
2. Oprima la letra subrayada del menú que desee abrir.
3. Oprima la letra subrayada del comando en el menu descendente.

Después de activar la barra también se puede pasar de un menú o comando a otro al oprimir las teclas de dirección. Para elegir un comando nuevo se debe oprimir las teclas de dirección. Para elegir un comando con este método, seleccione el menú que desee pulsando Alt y la letra apropiada, oprima la tecla de dirección hacia abajo para resaltar el comando que piense elegir y presione Enter.

Sugerencia

Oprima Esc o el botón del ratón en el nombre del menú para salir de él sin tener que hacer una elección. Presione Esc o el botón del ratón en el botón de cancelar para regresar de una caja de diálogo sin hacer una elección.

Cómo usar SITE

Una vez que tenemos abierta la ventana principal de **SITE**, en la cual se nos muestra la barra de título, la barra de menú, la barra de iconos y la barra de estado en la parte inferior de la ventana; notamos que sobre la pantalla se encuentra desplegado el mapa de la ciudad de **IXTAPA**. Si observamos la barra de iconos, notamos, que el icono que contiene la letra **I** se encuentra sumido. Y esto nos indica que tenemos seleccionada la ciudad de Ixtapa y que podemos obtener la información Turística de ella.

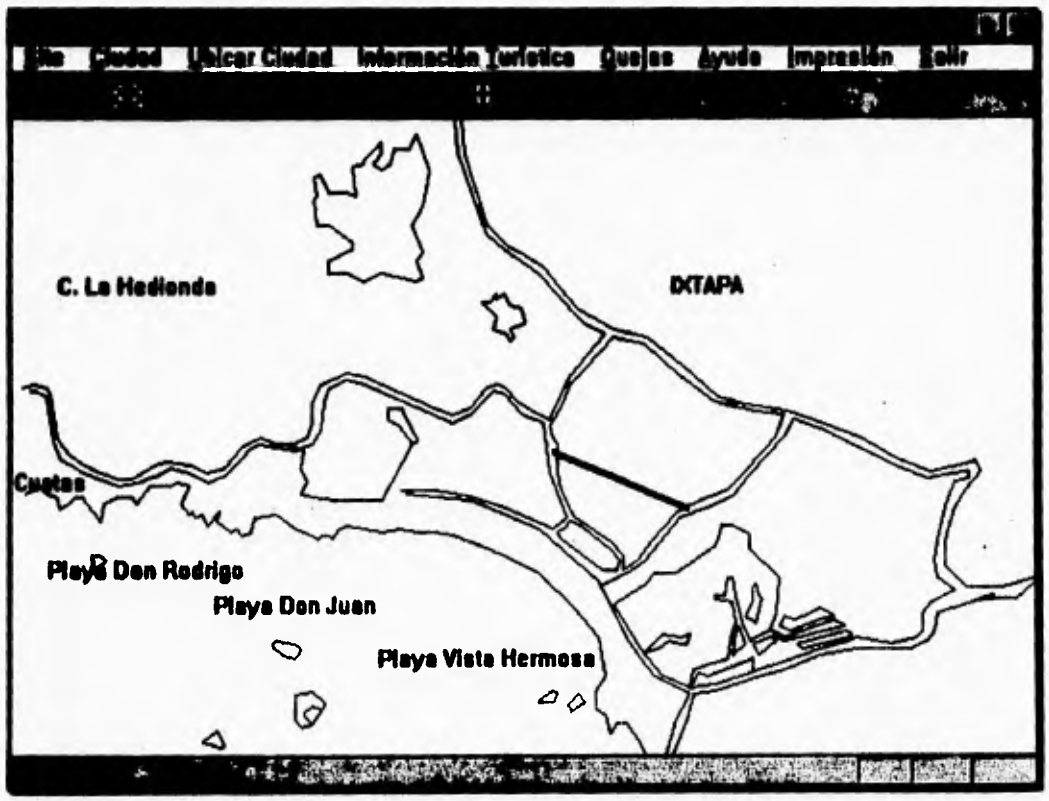


Fig 3.2.4 Ventana de SITE.

Cómo cambiar de Ciudad

Si queremos cambiar a la ciudad de **ZIHUATANEJO,** utilizando el ratón o el teclado hacemos lo siguiente:

Ratón: Colocando el apuntador del ratón sobre el botón de la barra de iconos que contiene la letra **Z** (se sume el botón), se presiona este, para activar la ventana que contiene el mapa de la ciudad de Zihuatanejo para poder obtener la información turística correspondiente.

Teclado: Presionamos la tecla **Alt** para activar la barra de menú, se oprime la letra **C** subrayada del menu **Ciudad** y se presiona la letra subrayada **Z** del menú descendente **Zihuatanejo**, para activar la ventana que contiene el mapa de la ciudad de Zihuatanejo para poder obtener la información turística correspondiente.

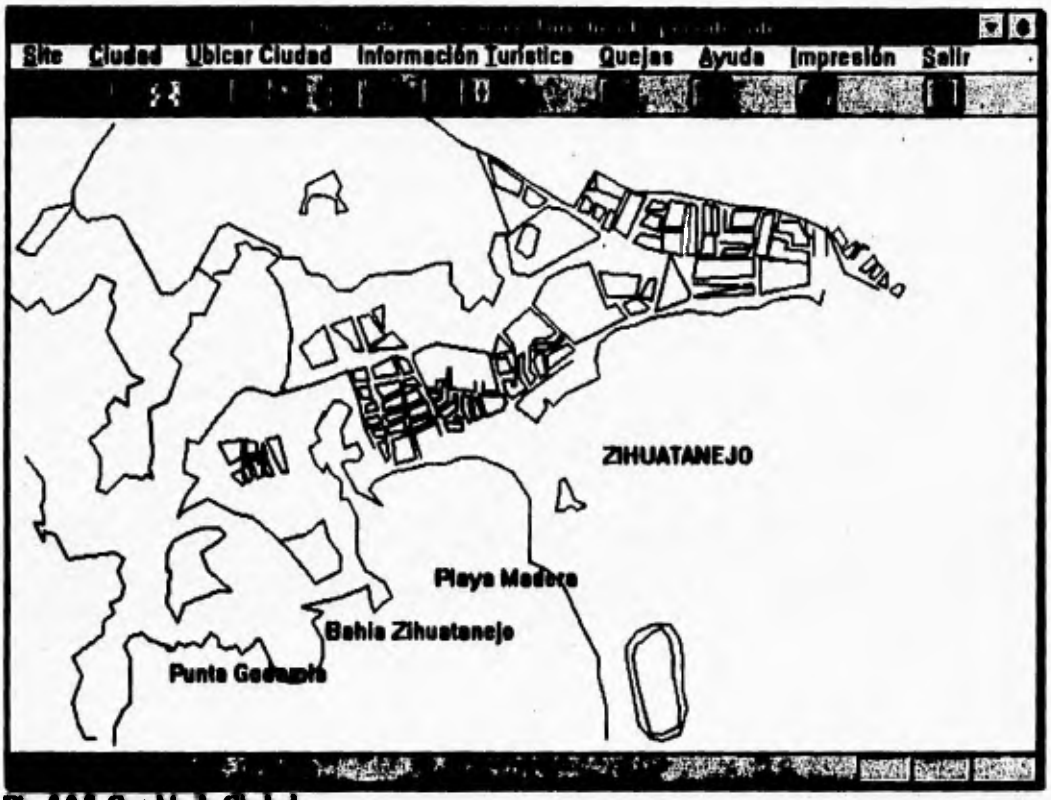


Fig 3.2.5 Cambio de Ciudad.

Si queremos cambiar a la ciudad de **IXTAPA,** utilizando el ratón o el teclado hacemos lo siguiente:

Ratón: Oprimimos el botón de la barra de iconos que contiene la letra **I** (se sume el botón), y con esto se activa la ventana que contiene el mapa de la ciudad de Ixtapa para poder obtener la información turística correspondiente.

Teclado: Presionamos la tecla **Alt** para activar la barra de menú, se oprime la letra **C** subrayada del menu **Ciudad** y se presiona la letra subrayada **I** del menú descendente **Ixtapa**, para activar la ventana que contiene el mapa de la ciudad de Zihuatanejo para poder obtener la información turística correspondiente.

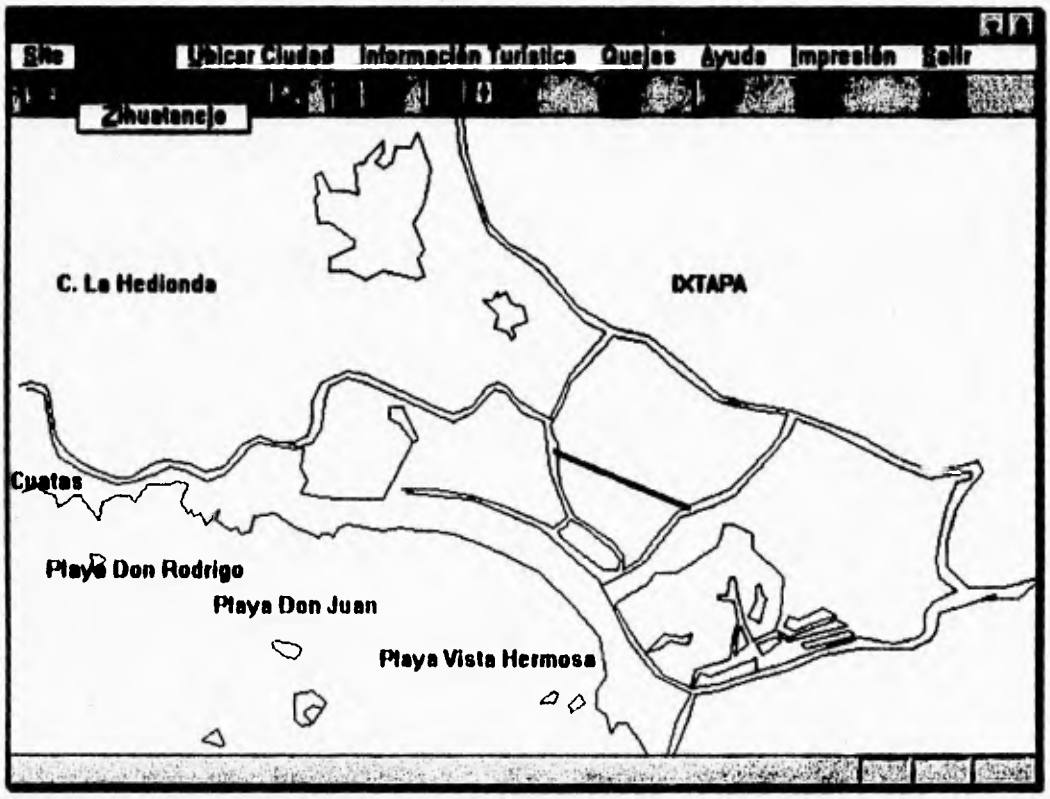


Fig. 3.2.6 Selección de la Ciudad Ixtapa.

Cómo utilizar el comando Conocer

Dependiendo de la ciudad que tengamos seleccionada ya sea Ixtapa o Zihuatanejo podemos obtener información cultural de la ciudad, y para ello se selecciona el comando **Conocer** que se encuentra en el menu **Site** ya sea de la barra de iconos o de la barra de menú.

Ratón: Oprimimos el botón de la barra de iconos que contiene el dibujo de una antorcha encendida (se sume el botón), y con esto se activa la ventana que contiene la información cultural de esa ciudad.

Teclado: Presionamos la tecla **Alt** para activar la barra de menú, se oprime la letra **S** subrayada del menu **Site** y se presiona la letra subrayada **C** del menú descendente **Conocer**, para activar la ventana que contiene la información cultural de esa ciudad.

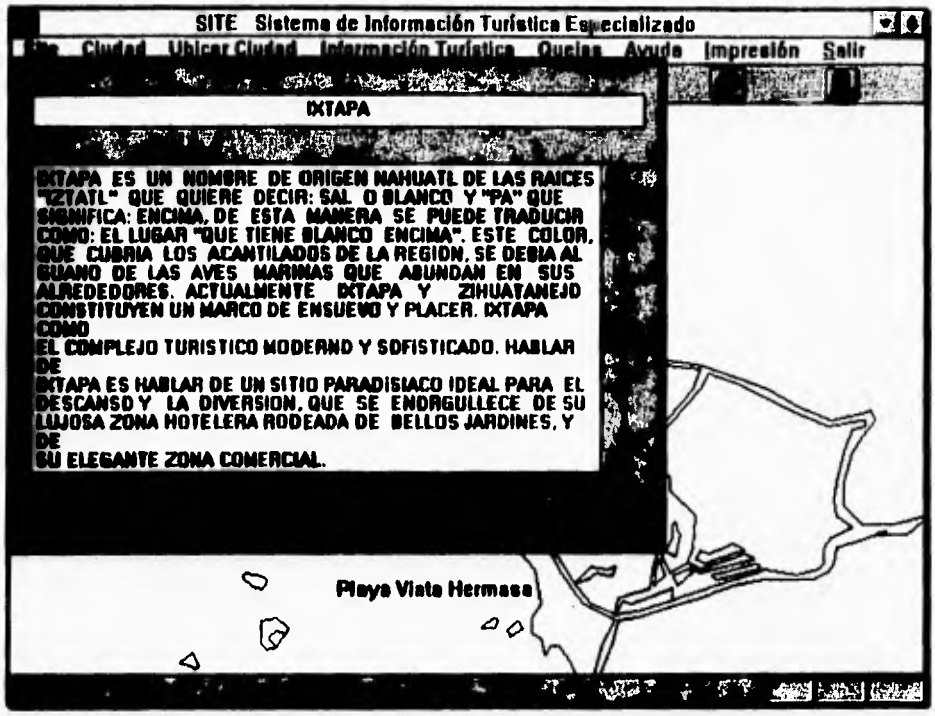


Fig. 3.2.7 Selección del comando Conocer.

Cómo utilizar el menu Ubicar Ciudad

El menú **Ubicar Ciudad** nos permite seleccionar los comandos **Radio de acción** y **Limpiar.**

Comando Radio de Acción

El comando Radio de Acción nos permite efectuar los acercamientos que queremos efectuar sobre el mapa respectivo de la ciudad seleccionada, teniendo tres opciones de acercamiento 5%, 25% y 50%.

Ratón: Oprimimos el botón de la barra de iconos que contiene el dibujo de una lupa (se sume el botón), y con esto se activa la ventana que contiene la caja de diálogo que nos permite seleccionar una de tres opciones de aumento, ya sea 5%, 25% y 50%. A lado de cada una de las opciones se encuentra un circulo vacio. Para activar la opción, por ejemplo del 25%, colocamos el apuntador del ratón sobre el circulo vacio del letrero que dice 25% y se presiona el botón izquierdo del ratón, al hacer esto se rellena el circulo de la opción de 25% después se presiona el botón **OK** que se encuentra en la parte derecha de la caja de dialogo para indicar que estamos de acuerdo con este 25% de aumento.

Teclado: Presionamos la tecla **Alt** para activar la barra de menú, se oprime la letra **U** subrayada del menu **Ubicar Ciudad** y se presiona la letra subrayada **R** del menú descendente **Radio de acción**, y con esto se activa la ventana que contiene la caja de diálogo que nos permite seleccionar una de tres opciones de aumento, ya sea 5%, 25% y 50%. A lado de cada una de las opciones se encuentra un circulo vacio. Para activar la opción, por ejemplo del 25%, pulsamos la tecla **TAB** hasta colocarnos sobre el circulo vacio del letrero que dice 25%, al hacer esto se rellena el circulo de la opción de 25% y se presiona **Enter** para indicar que estamos de acuerdo con este 25% de aumento.

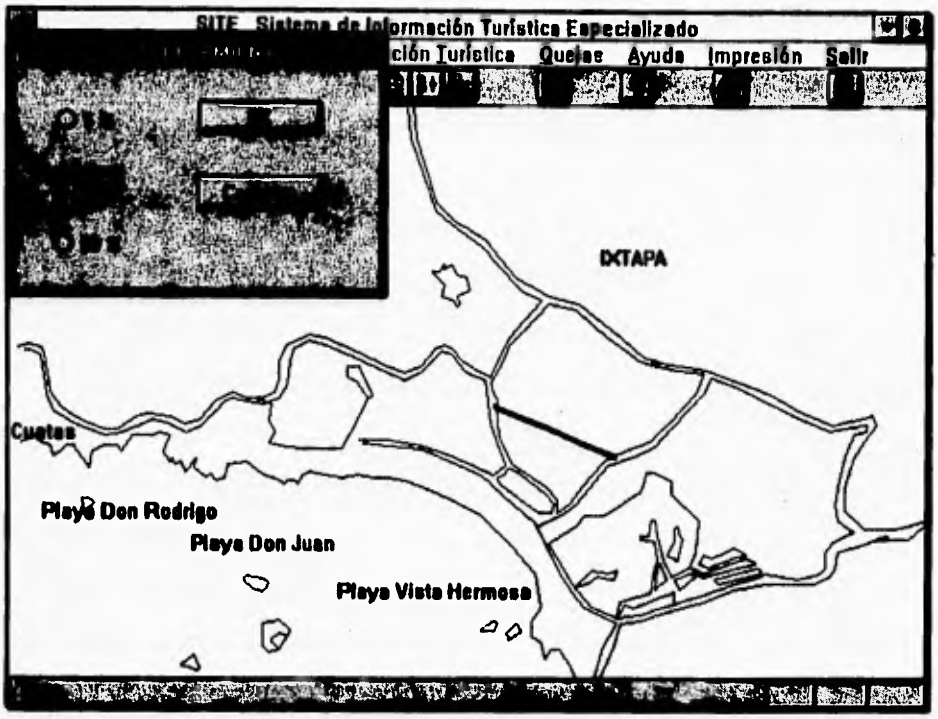


Fig. 3.2.8 Selección del Menú Ubicar Ciudad.

Para realizar el aumento sobre los mapas, basta con colocar el cursor del Ratón sobre el área del mapa que queremos visualizar mejor y pulsamos el botón izquierdo del ratón. Con esto se observa un aumento del 25% cada vez que pulsamos el ratón sobre el área respectiva.

Comando Ubicar

El comando Ubicar nos permite regresar al tamaño inicial de los mapas.

Ratón: Oprimimos el botón de la barra de iconos que contiene el dibujo de cuatro flechas que se expanden (se sume el botón), y con esto se regresa al tamaño inicial de la ventana.

Teclado: Presionamos la tecla **Alt** para activar la barra de menú, se oprime la letra **U** subrayada del menu **Ubicar Ciudad** y se presiona la letra subrayada **C** del menú descendente **Ciudad**, y con esto se regresa al tamaño inicial de la ventana.

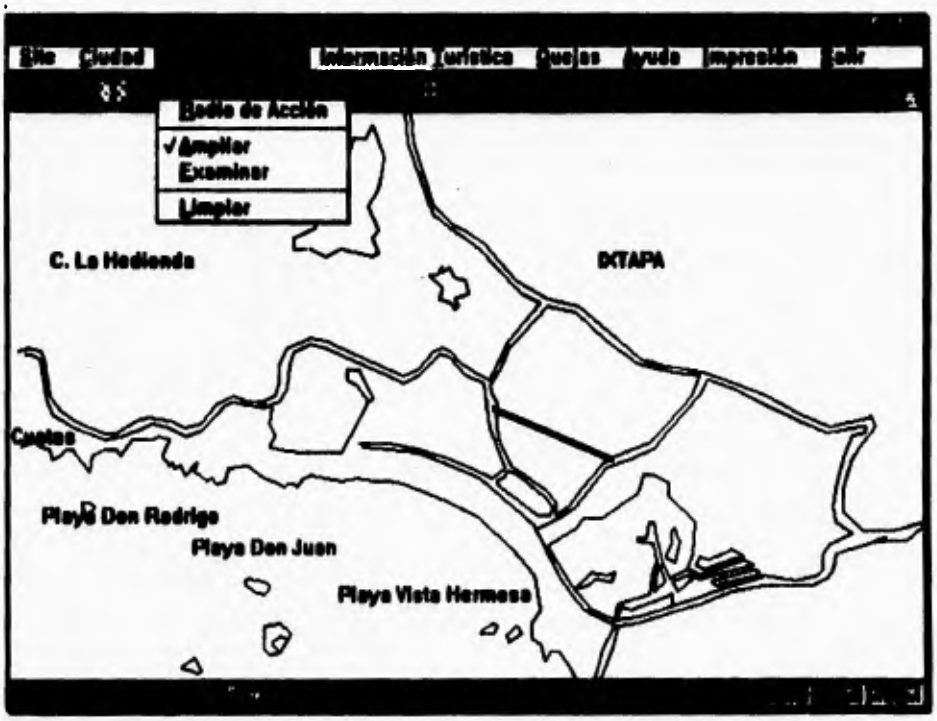


Fig. 3.2.9 Selección del comando Ubicar.

Comando Limpiar

El comando Limpiar nos permite limpiar los mapas de las selecciones que hayamos realizado sobre ellos.

Ratón: Oprimimos el botón de la barra de iconos que contiene el dibujo de un borrador o goma de borrar (se sume el botón), y con esto se borran todas las selecciones anteriores.

Teclado: Presionamos la tecla **Alt** para activar la barra de menú, se oprime la letra **L** subrayada del menu **Limpiar** y se presiona la letra subrayada **L** del menú descendente **Limpiar**, y con esto se borran todas las selecciones anteriores.

Cómo usar el menu Información Turística

Dependiendo de la ciudad seleccionada, este menu nos permite acesar a los comandos de Bienes, Servicios y Atracciones Turísticas que son los que nos proporcionan la Información Turística del puerto de Ixtapa - Zihuatanejo.

Comando Bienes

Este comando nos permite escoger un nombre de un Bien entre la lista de **Bienes** de las ciudades de Ixtapa o Zihuatanejo.

Ratón: Colocando el apuntador del ratón sobre el botón de la barra de iconos que contiene la letra **A** (se sume el botón), se presiona este, para activar la ventana que contiene la lista de todos los bienes de cada ciudad.

Teclado: Presionamos la tecla **Alt** para activar la barra de menú, se oprime la letra **f** subrayada del menu **Información Turística** y se presiona la letra subrayada **B** del menú descendente **Bienes**, para activar la ventana que contiene la lista de todos los bienes que ofrece cada ciudad.

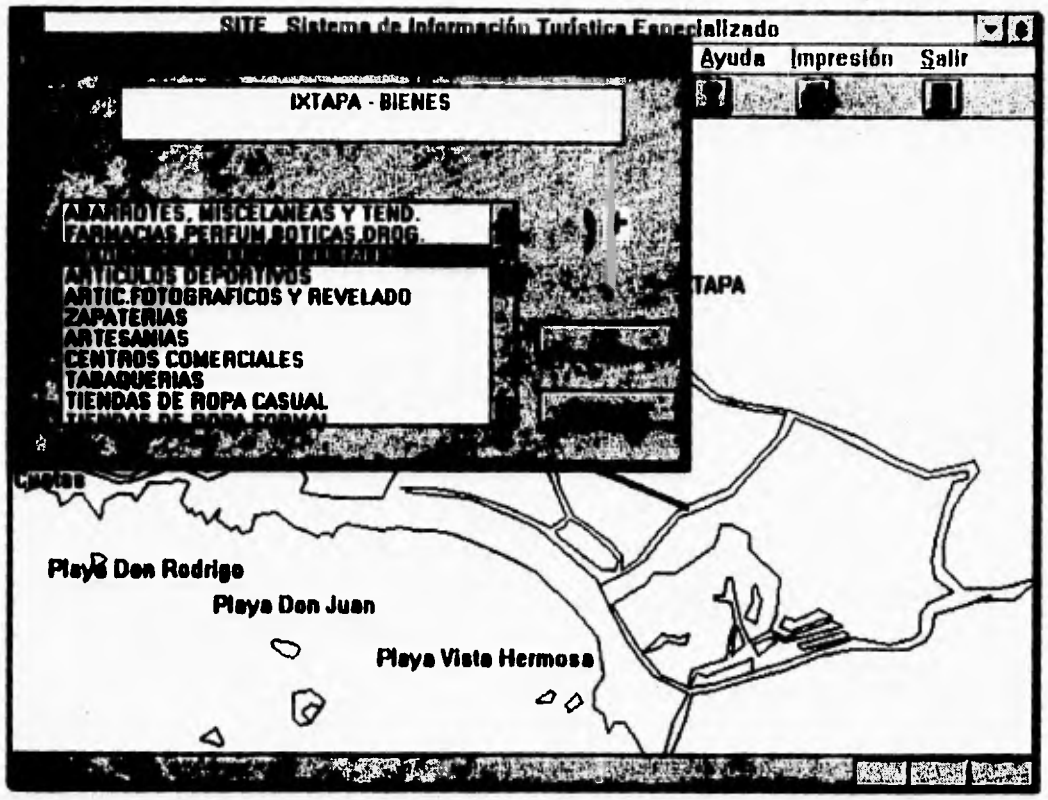


Fig. 3.2.10 Selección del comando Bienes.

En la ventana que se abre se despliega una lista con todos los nombres de los Bienes que la ciudad ofrece. Use esta lista para seleccionar el nombre del Bien del que desee obtener información. Este tipo de ventana permanece abierta siempre de forma que pueda ver la lista. Puede recorrer la lista al oprimir el botón del ratón en las flechas hacia arriba y hacia abajo, en la barra de recorrido que se ubica a la derecha de la caja de lista o bien al oprimir las teclas de dirección hacia arriba o hacia abajo. Se puede desplazar a una alternativa con rápidez al oprimir una vez el botón en la caja y escribir la primera letra de la alternativa deseada. La lista se recorre

hasta los nombres que comienzan con esa letra. Cuando vea el elemento que desea seleccionar, oprima el botón sobre él, o bien oprima la tecla de dirección hacia arriba o hacia abajo hasta que el elemento quede resaltado.

Comando Servicios

Este comando nos permite escoger entre la lista de **Servicios** de las ciudades de Ixtapa o Zihuatanejo.

Ratón: Colocando el apuntador del ratón sobre el botón de la barra de iconos que contiene los dibujos de un telefono y cruz roja (se sume el botón), se presiona este, para activar la ventana que contiene la lista de todos los servicios de cada ciudad.

Teclado: Presionamos la tecla **Alt** para activar la barra de menú, se oprime la letra **I** subrayada del menu **Información Turística** y se presiona la letra subrayada **S** del menú descendente **Servicios**, para activar la ventana que contiene la lista de todos los bienes que ofrece cada ciudad.

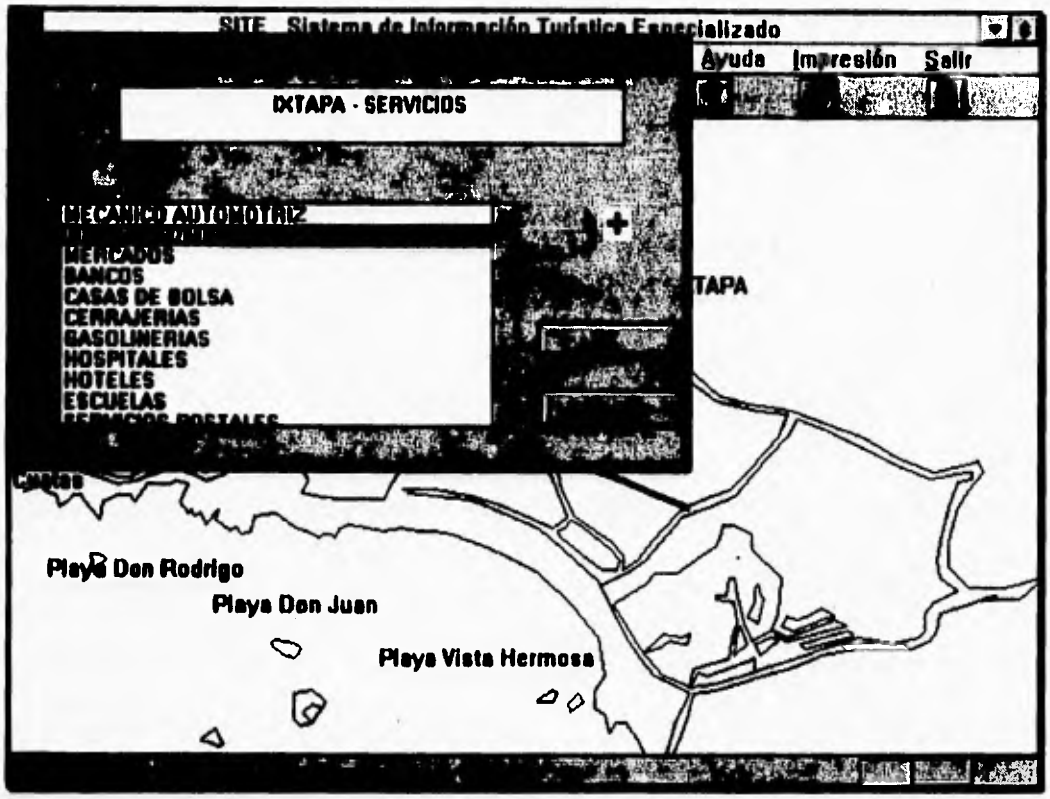


Fig. 3.2.11. Selección de comando Servicos.

En la ventana que se abre, se despliega una lista con todos los nombres de los Servicios que la ciudad ofrece. Use esta lista para seleccionar el nombre del servicio del que desee obtener información. Este tipo de ventana permanece abierta siempre de forma que pueda ver la lista. Puede recorrer la lista al oprimir el botón del ratón en las flechas hacia arriba y hacia abajo en la barra de recorrido que se ubica a la derecha de la caja de lista o bien al oprimir las teclas de dirección hacia arriba o hacia abajo. Se puede desplazar a una alternativa con rápidez al oprimir una vez el botón en la caja y escribir la primera letra de la alternativa deseada. La lista se recorre hasta los nombres que comienzan con esa letra. Cuando vea el elemento que desea seleccionar,

oprima el botón sobre él, o bien oprima la tecla de dirección hacia arriba o hacia abajo hasta que el elemento quede resaltado.

Comando Atracciones Turísticas

Este comando nos permite escoger entre la lista de **Atracciones Turísticas** de las ciudades de Ixtapa o Zihuatanejo.

Ratón: Colocando el apuntador del ratón sobre el botón de la barra de iconos que contiene el dibujo de dos playas (se sume el botón), se presiona este, para activar la ventana que contiene la lista de todas las Atracciones Turísticas de cada ciudad.

Teclado: Presionamos la tecla **Alt** para activar la barra de menú, se oprime la letra **T** subrayada del menu **Información Turística** y se presiona la letra subrayada **A** del menú descendente **Atracciones Turísticas**, para activar la ventana que contiene la lista de todas las Atracciones Turísticas que ofrece cada ciudad.

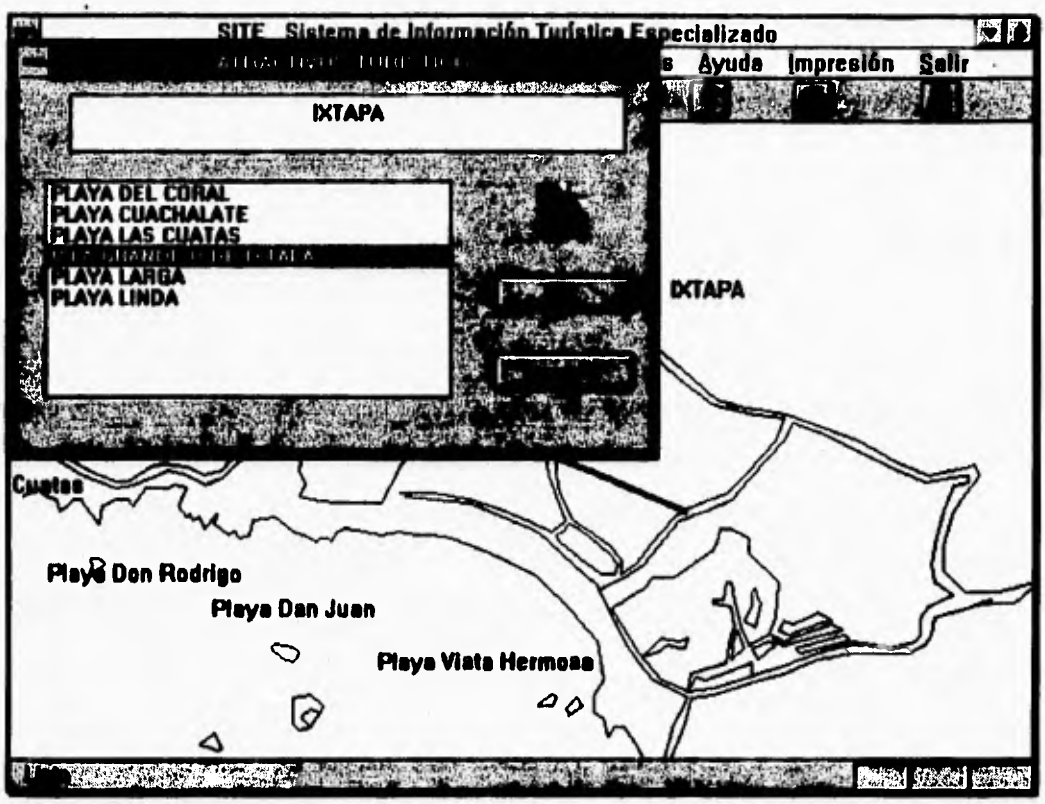


Fig 3.2.12 Selección del comando Atracciones Turísticas.

En la ventana que se abre, se despliega una lista con todos los nombres de las Atracciones Turísticas que la ciudad ofrece. Use esta lista para seleccionar el nombre de la Atracción Turística de la que desee obtener información. Este tipo de ventana permanece abierta siempre de forma que pueda ver la lista. Puede recorrer la lista al oprimir el botón del ratón en las flechas hacia arriba y hacia abajo en la barra de recorrido que se ubica a la derecha de la caja de lista o bien al oprimir las teclas de dirección hacia arriba o hacia abajo. Se puede desplazar a una alternativa con rápidez al oprimir una vez el botón en la caja y escribir la primera letra de la alternativa deseada. La lista se recorre hasta los nombres que comienzan con esa letra. Cuando vea el elemento que

desea seleccionar, oprima el botón sobre él , o bien oprima la tecla de dirección hacia arriba o hacia abajo hasta que el elemento quede resaltado.

Cómo Obtener Información

Una vez que se haya seleccionado el nombre del Bien, Servicio o Atracción Turística, sobre el mapa de la ciudad se dibujaran una serie de puntos rojos, cada punto indica la ubicación de un sitio de la ciudad. Para obtener información sobre ese sitio, hay que seleccionar primeramente de la barra de iconos el botón verde que contiene la letra **E** que corresponde al comando de examinar o de la barra de menu hay que oprimir la tecla **Alt** para activar la barra de menú, se oprime la letra **U** subrayada del menu **Ubicar Ciudad** y se presiona la letra subrayada **E** del menú descendente **Examinar.**

Una vez hecho esto ya se puede obtener la información de ese sitio al unicamente tener que colocar el apuntador del ratón sobre un punto rojo y presionar su botón izquierdo. Al hacer esto se desplegará una ventana con la información correspondiente de ese sitio, como es la dirección del lugar, el tipo de servicio que ofrece, su horario de atención, su teléfono, su categoría o la información cultural de ese lugar.

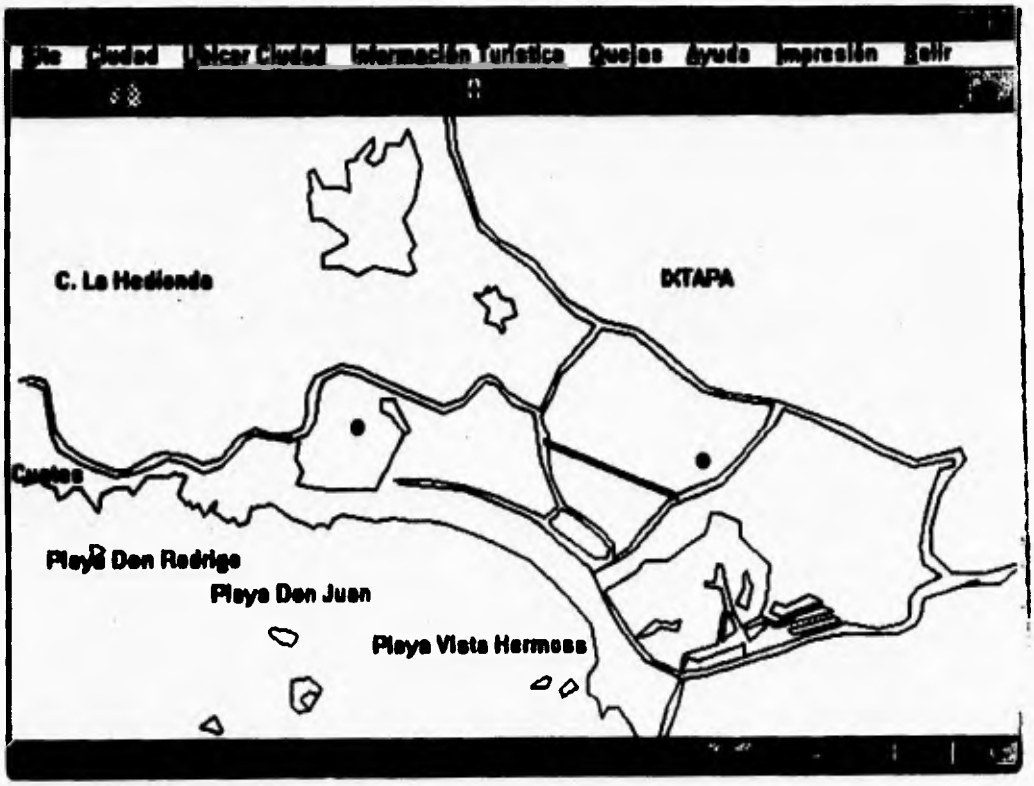


Fig 3.2.13 Localización de los Sitios (Tiendas de Autoservicio y Departamentales).

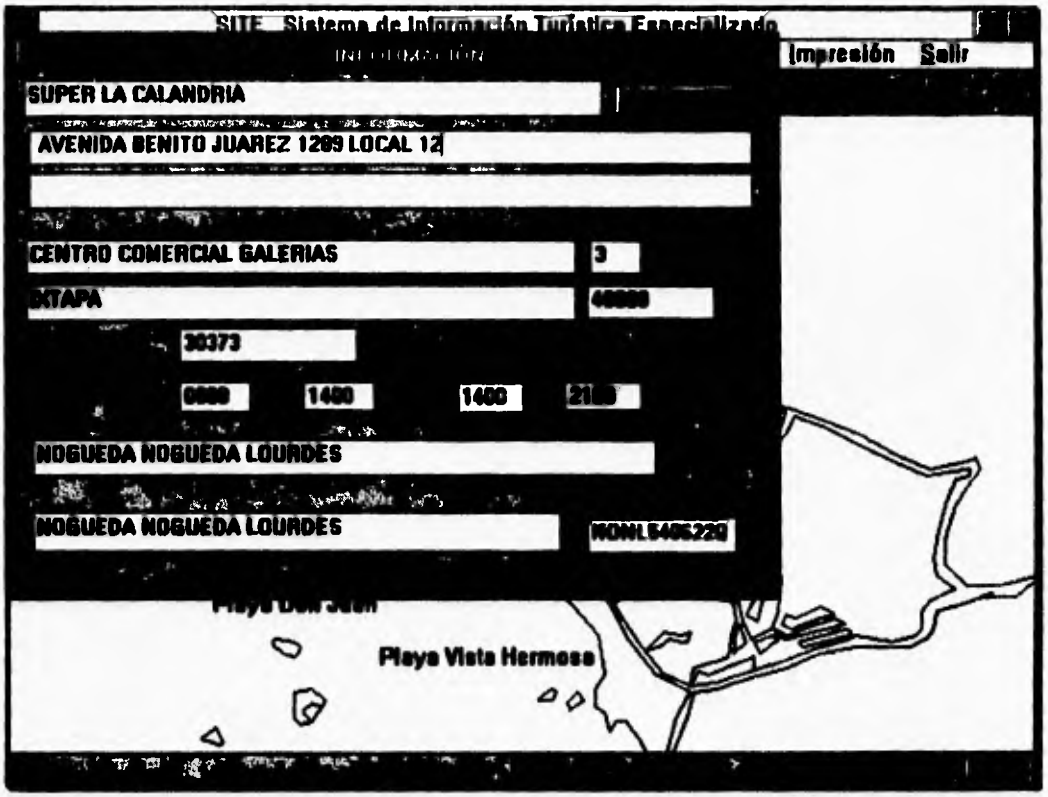


Fig 3.2.14 Despliegue de información.

Si se desea hacer un acercamiento sobre algún sitio en particular se tendrá que activar el comando Amplificar de la barra de iconos, presionando el botón verde que contiene la tecla A o de la barra de menú oprimiendo la tecla **Alt** para activar la barra de menú, se oprime la letra **U** subrayada del menu **Ubicar Ciudad** y se presiona la letra subrayada **A** del menú descendente **Amplificar.** Después se realiza el procedimiento de **Cómo usar el menú Ubicar Ciudad** anteriormente explicado.

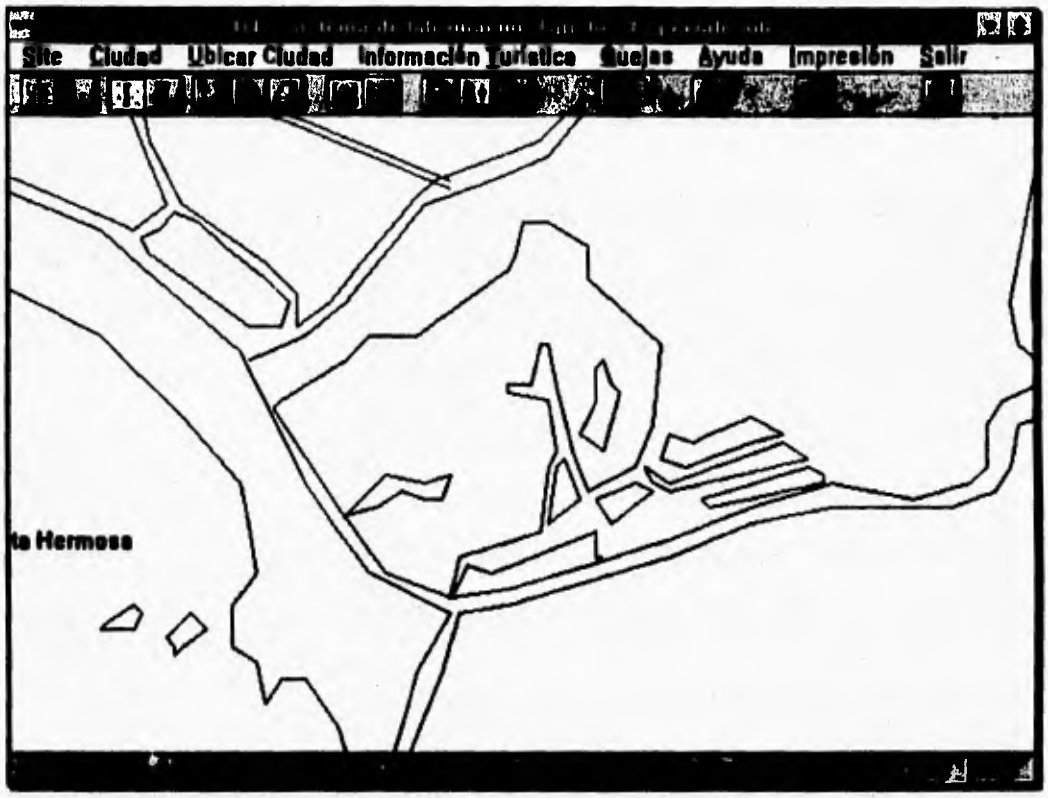


Fig. 3.2.15 Selección del comando Amplificar.

Cómo usar el Comando Quejas

Este comando nos permite accesar a un formato donde el usuario puede manifestar alguna sugerencia o queja respecto a Site. Se puede accesar a este comando ya sea usando la barra de iconos o desde la barra de menú.

Ratón: Oprimimos el botón de la barra de iconos que contiene el dibujo de una cara enojada (se sume el botón), y con esto se activa la ventana que contiene el formato de quejas de Site.

Teclado: Presionamos la tecla **Alt** para activar la barra de menú, se oprime la letra **Q** subrayada del menu **Quejas** y con esto se activa la ventana que contiene el formato de quejas de Site.

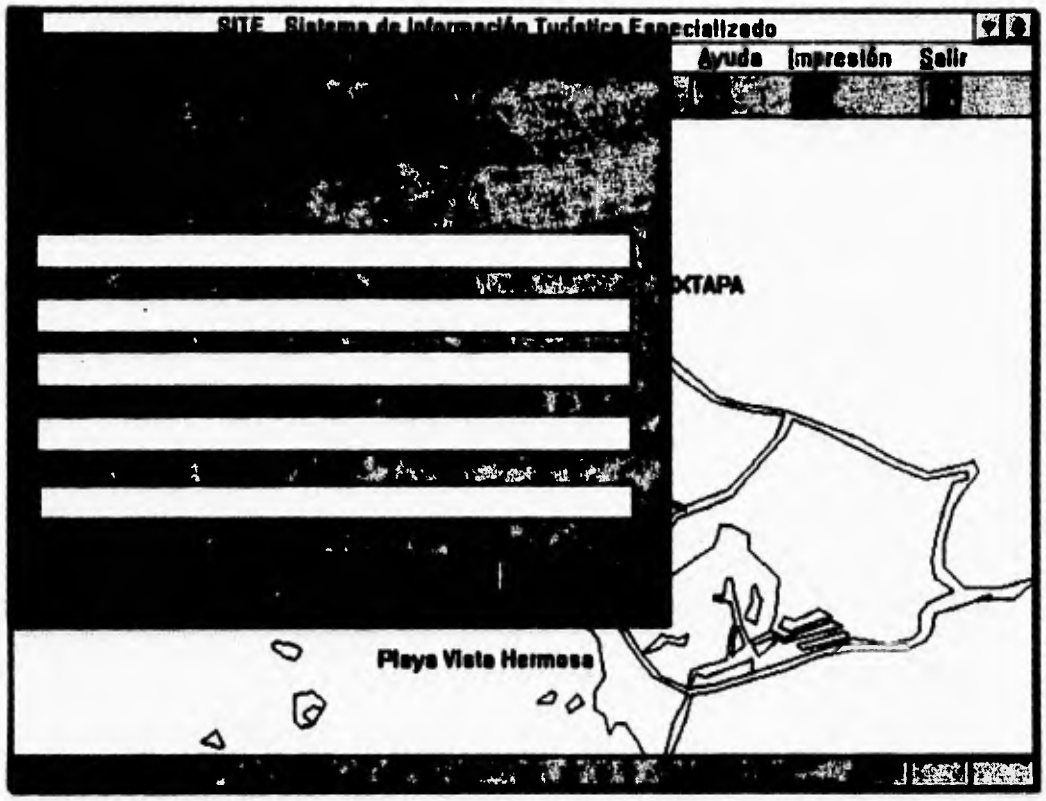


Fig. 3.2.16 Selección del Formato de Quejas.

Cómo usar el comando Ayuda

Este comando nos permite accesar a un formato donde el usuario puede obtener Ayuda respecto al funcionamiento del sistema Site. Se puede accesar a este comando ya sea usando la barra de iconos o desde la barra de menú.

Ratón: Oprimimos el botón de la barra de iconos que contiene el dibujo de un signo de interrogación (se sume el botón), y con esto se activa la ventana que contiene la información de Site.

Teclado: Presionamos la tecla **Alt** para activar la barra de menú, se oprime la letra **A** subrayada del menu **Ayuda** y con esto se activa la ventana que contiene la información de Site.

Cómo usar el menu de Impresión

El menú de impresión nos permite obtener una gráfica del sitio impresa con la información respectiva del lugar. Para usar el menu de impresión siga estos pasos:

Elija el comando Impresión de la barra de iconos o de la barra de menú.

Ratón: Oprimimos el botón de la barra de iconos que contiene el dibujo de una Impresora.

Teclado: Presionamos la tecla **Alt** para activar la barra de menú, se oprime la letra **I** subrayada del menu **Impresión.**

Cómo salir de Site Para Windows

Para salir de Site para Windows, siga estos pasos:

Elija el comando salir de la barra de iconos o de la barra de menú.

Ratón: Oprimimos el botón de la barra de iconos que contiene el dibujo de una puerta.

Teclado: Presionamos la tecla **Alt** para activar la barra de menú, se oprime la letra **S** subrayada del menu **Salir**

3.3 Mantenimiento

Actualización y depuración de la infromación

En cuanto al mantenimiento del sistema, éste se realizará unicamente para las bases de datos, Para lo cual, se actualizará la información dentro de un módulo diseñado para cumplir con tal función, dicho módulo contará con las funciones básicas de actualización tales como: altas, bajas, cambios y consultas. De igual forma se pretende depurar toda la información dependiendo de los tiempos que se establezcan para realizar ésta. El código que genera Paradox para DOS, que lleva acabo el mantenimiento de la información ésta contenido en el Apéndice B.

CONCLUSIONES

Conclusiones

La terminación del sistema se logró principalmente al empleo de las técnicas Orientadas a objetos, implementadas mediante el lenguaje de programación Visual C + +. Gracias a la utilización de tales técnicas se obtuvieron las siguientes conclusiones:

- Se logró optimizar el proceso de información turística en el puerto de Ixtapa-Zihuatanejo.

- El software que se realizo tuvo el propósito de que su uso con una interfaz usuario gráfica fuera lo más sencillo posible.

- El sistema resulta ser amigable para cualquier persona, mexicana o extranjera que no posea ningún conocimiento de computación, ya que proporciona acceso mediante símbolos gáficos de tipo universal, con los cuales no es necesario leer las opciones del menú principal, sino únicamente presionar un botón y así obtener la información.

- El sistema permite al usuario conocer la gran variedad de Bienes, Servicios y Atracciones Turísticas que el puerto de Ixtapa Zihuatanejo ofrece, además de los lugares donde puede adquirirlos, los precios de venta, sin tener que moverse del sitio donde esta efectuando la consulta, para su mayor comodidad

- Se logró crear una pantalla sugestiva al usuario de manera que este, sólo tiene que apuntar a iconos o elementos de un menú desplegado, que están relacionados con los objetos. Tomando en cuenta que el proceso de ver y apuntar es más fácil que recordar y escribir. El software de los futuros sistemas se debe de diseñar, teniendo presente que

contarán con una interfaz usuario gráfica, para así poder trabajar con las diversas interfaces dominantes de este tipo (como la Macintosh).

- El diseño de software se generó, a partir de reglas, formatos y clases basadas en depósitos previamente establecidos.

- El código no se generó de forma manual, en la medida de los posible. La aplicación se realizó a partir de componentes ya existentes. Muchos de los cuales están construidos de modo que se pueden adaptar a un diseño particular. Las nuevas herramientas de programación deben proporcionar la máxima capacidad posible para la generación de código.

- En el diseño de las pantallas se logró de crearlas de buena calidad, puesto que se integrarón a partir de componentes que han sido probados y verificados varias veces.

- La programación se volvió más sencilla ya que los programas se conjuntaron a partir de piezas pequeñas, cada una de las cuales, en general se pueden crear varias veces.

- Este sistema ayudará a incrementar el nivel turístico del sitio de interés, ya que al contemplar diversos giros comerciales, el usuario tendrá mayores opciones a elegir y por lo tanto, mayor número de negocios aumentarán sus ingresos.

Mientras más pronta sea la difusión del análisis y el diseño orientado a objetos, más rápidamente pasará la creación del software de ser una industria de campo a una disciplina de la ingeniería.

BIBLIOGRAFIA

BIBLIOGRAFIA

Eckel, Bruce
-Aplique C++

Gurewich. Ori & Gurewich, Nathan
-Paradox 4.5 for Windows Unleashed

James Martin, Jame J. Odell
-Analisis y Diseñó Orientado a Objetos

Kryuglinski David J.
-Inside Visual C++

Kryuglinski David J.
-Progrese con Visual C++.

Murray, Willian H.

Microsoft C/C++7 Manual de Referencia.

Padwick, Gordon

-Paradox For Windows.

Saison, Ted
-Programación Orientada a Objetos con Borland C++

Setrag Khoshafian, Razmik Abnous
-Object Orientation
Concepts, Languages, Databases, User Interfaces

Stroustrup, Bjarne
-Turbo C/C++ The Complete Reference

Roetxheim, William
-Programming Windows with Borland C++

Wiener, Richard S., Pinson, Lewis J.
-An Introduction to Objets Oriented Programming and C++

Timothy Budd
-An Introduction to Object Oriented Programming

Mark Mullin
-Object Oriented Program Design with examples in C ++

James A. Senn.
-Análisis y Diseño de Sistemas de Información

Henry F. Korth.
-Fundamentos de Bases de Datos

REVISTAS

-PC Magazine en Español
Volumen 3-Número 12
Diciembre 1992. Pag. 96,100
"Base de Datos"

-PC Magazine en Español
Volumen 5-Número 2
Febrero de 1994. Pag. 10
"Visual C++"

-PC World
Junio 1994. Pags, 11,12,14,15,16,17,18,20,21,22
"La Base de Datos Correcta para Usted"

-PC World
Mayo 1994. Pag. 10
"Biblioteca gráfica C++ para Windows 3.1"

-PC Actual

"Programación Visual."

Madrid, Marzo 1994, Año V, Nº 51.

pag. 102 - 130.

-PC Actual.

"Compiladores de C++"

Madrid, Mayo 1994, Año V, Nº 53.

pag. 138 - 148.

Apendice

A

ARCHIVO MAINFRM.CPP

```
// mainfrm.cpp : implementation of the CMainFrame
class
//
#include "stdafx.h"
#include "site.h"
#include "mainfrm.h"
#include <stdlib.h>
#include <malloc.h>
#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////////
// CMainFrame
IMPLEMENT_DYNCREATE(CMainFrame,
CFrameWnd)
BEGIN_MESSAGE_MAP(CMainFrame,
CFrameWnd)
        //{{AFX_MSG_MAP(CMainFrame)
        ON_WM_CREATE()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
/////////////////////////////////////////////////////////////////////////
// arrays of IDs used to initialize control bars

// toolbar buttons - IDs are command buttons
static UINT BASED_CODE buttons[] =
{
        // same order as in the bitmap 'toolbar.bmp'
        ID_APP_ABOUT,
        ID_SITE_CONOCER,
                ID_SEPARATOR,
        ID_CIUDAD_IXTAPA,
        ID_CIUDAD_ZIHUATANEJO,
                ID_SEPARATOR,
        ID_UBICAR_CIUDAD,
        ID_UBICAR_RADIODEACCIN,
        ID_UBICAR_LIMPIAR,
                ID_SEPARATOR,
                ID_SEPARATOR,
        ID_UBICARCIUDAD_BUSCAR,
        ID_UBICARCIUDAD_EXAMINAR,
                ID_SEPARATOR,
                ID_SEPARATOR,
ID_INFORMACINTURSTICA_BIENES,
ID_INFORMACINTURSTICA_SERVICIOS,
ID_INFORMACINTURSTICA_ATRACTIVOS
        ID_SEPARATOR,
                ID_SEPARATOR,
                ID_SEPARATOR,
                ID_SEPARATOR,
                ID_SEPARATOR,
                ID_SEPARATOR,
        ID_APP_QUEJAS,
                ID_SEPARATOR,
                ID_SEPARATOR,
                ID_SEPARATOR,
                ID_SEPARATOR,
                ID_SEPARATOR,
        ID_APP_AYUDA,
                ID_SEPARATOR,
                ID_SEPARATOR,
                ID_SEPARATOR,
                ID_SEPARATOR,
                ID_SEPARATOR,
                ID_SEPARATOR,
        ID_APP_AYUDA,
                ID_SEPARATOR,
                ID_SEPARATOR,
                ID_SEPARATOR,
                ID_SEPARATOR,
                ID_SEPARATOR,
                ID_SEPARATOR,
                ID_SEPARATOR,
                ID_SEPARATOR,
        ID_APP_EXIT,
};
static UINT BASED_CODE indicators[] =
{
        ID_SEPARATOR,                   //
status line indicator
        ID_INDICATOR_CAPS,
        ID_INDICATOR_NUM,
        ID_INDICATOR_SCRL,
};
/////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
        // TODO: add member initialization code
here
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT
lpCreateStruct)
{
        if (CFrameWnd::OnCreate(lpCreateStruct)
== -1)
                return -1;

        if (!m_wndToolBar.Create(this) ||
        !m_wndToolBar.LoadBitmap(IDR_MAINFR
AME) ||

        !m_wndToolBar.SetButtons(buttons,
         sizeof(buttons)/sizeof(UINT)))
        {
                TRACE("Failed to create
toolbar\n");
                return -1;              // fail to create
        }

        if (!m_wndStatusBar.Create(this) ||
```

```
        !m_wndStatusBar.SetIndicators(indicators,
                sizeof(indicators)/sizeof(UINT)))
        {
                TRACE("Failed to create status
bar\n");
                return -1;              // fail to create
        }

        return 0;
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
        CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
        CFrameWnd::Dump(dc);
}

#endif //_DEBUG
```

ARCHIVO SITEDOC.CPP

```
// sitedoc.cpp : implementation of the CSiteDoc class
//
#include "stdafx.h"
#include "site.h"
#include "sitedoc.h"
#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CSiteDoc
IMPLEMENT_DYNCREATE(CSiteDoc, CDocument)

BEGIN_MESSAGE_MAP(CSiteDoc, CDocument)
        //{{AFX_MSG_MAP(CSiteDoc)
                // NOTE - the ClassWizard will add
and remove mapping macros here.
                //    DO NOT EDIT what you see in
these blocks of generated code!
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////


BOOL CSiteDoc::OnNewDocument()
{
        if (!CDocument::OnNewDocument())
                return FALSE;
        // TODO: add reinitialization code here
        // (SDI documents will reuse this document)
        return TRUE;
}
/////////////////////////////////////////////////////////////////////////////
// CSiteDoc serialization
void CSiteDoc::Serialize(CArchive& ar)
{
        if (ar.IsStoring())
        {
                // TODO: add storing code here
        }
        else
        {
                // TODO: add loading code here
        }
}


/////////////////////////////////////////////////////////////////////////////
// CSiteDoc diagnostics

#ifdef _DEBUG
void CSiteDoc::AssertValid() const
{
        CDocument::AssertValid();
}

void CSiteDoc::Dump(CDumpContext& dc) const
{
        CDocument::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CSiteDoc commands
```

ARCHIVO SITEVIEW.CPP

```
// siteview.cpp : implementation of the CSiteView class
//
#include "stdafx.h"
#include "site.h"
#define MARCA 9999
#include "sitedoc.h"
#include "siteview.h"
#include "dradio.h"
#include "dcuadro.h"
#include "dselecc.h"
#include "dbienser.h"
#include "dinfo.h"
#include "dquejas.h"
#include "dayuda.h"
#ifdef _DEBUG
#undef THIS_FILE
```

```
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////////////
// CSiteView
IMPLEMENT_DYNCREATE(CSiteView, CView)
BEGIN_MESSAGE_MAP(CSiteView, CView)
        //{{AFX_MSG_MAP(CSiteView)
        ON_COMMAND(ID_UBICAR_CIUDAD,
OnUbicarCiudad)
        ON_WM_LBUTTONDOWN()
        ON_COMMAND(ID_CIUDAD_IXTAPA,
OnCiudadIxtapa)
        ON_COMMAND(ID_CIUDAD_ZIHUATAN
EJO, OnCiudadZihuatanejo)
        ON_WM_RBUTTONDOWN()
        ON_COMMAND(ID_UBICAR_RADIODEA
CCION, OnUbicarRadiodeaccin)
        ON_COMMAND(ID_SITE_CONOCER,
OnSiteConocer)
        ON_COMMAND(ID_INFORMACINTURST
ICA_ATRACTIVOSTURSTICOS,
OnInformacintursticaAtractivosturisticos)
        ON_COMMAND(ID_INFORMACINTURST
ICA_SERVICIOS, OnInformacinturisticaServicios)
        ON_COMMAND(ID_INFORMACINTURST
ICA_BIENES, OnInformacinturisticaBienes)
        ON_COMMAND(ID_UBICAR_LIMPIAR,
OnUbicarLimpiar)
        ON_COMMAND(ID_UBICARCIUDAD_BU
SCAR, OnUbicarciudadBuscar)
        ON_COMMAND(ID_UBICARCIUDAD_EX
AMINAR, OnUbicarciudadExaminar)
        ON_UPDATE_COMMAND_UI(ID_CIUDA
D_IXTAPA, OnUpdateCiudadIxtapa)
        ON_UPDATE_COMMAND_UI(ID_CIUDA
D_ZIHUATANEJO, OnUpdateCiudadZihuatanejo)
        ON_UPDATE_COMMAND_UI(ID_UBICA
RCIUDAD_BUSCAR, OnUpdateUbicarciudadBuscar)
        ON_UPDATE_COMMAND_UI(ID_UBICA
RCIUDAD_EXAMINAR,
OnUpdateUbicarciudadExaminar)
        ON_COMMAND(ID_APP_QUEJAS,
OnAppQuejas)
        ON_COMMAND(ID_APP_AYUDA,
OnAppAyuda)
        //}}AFX_MSG_MAP
        // Standard printing commands
        ON_COMMAND(ID_FILE_PRINT,
CView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_PREVIE
W, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

typedef struct      {       entero x;
                            entero y;
                            char titulo[31]; }
        desc_sit;

typedef struct      {       entero reg;
                            entero x;
                            entero y;
                            byte   giro; } lugar_sit;

entero *a_apt;
entero *a_lin; // puntos del mapa
int ancho = 4;
double acerca = .05;
byte ciudad = 0, limite = 1, giro_act = 0, examinar=0,
buscar=1;
long max_x, max_y, min_x, min_y;
long *stack, *a_stack;
double derecho, abajo, dif_x,dif_y;
char *na_datos = "i_datos.dbf", *na_infosi =
"i_infosi.dbf", *na_datost = "i_datost.dbf";
char *na_lugt = "i_lugt.dbf", *na_giro = "i_gir.dbf";
CFile fsit;
desc_sit *a_dat;  // sitios en el mapa
lugar_sit *a_inf;  // Relación de Bienes y Servicios en
el archivo
lugt_sit  *a_lugt; // lista de lugares turísticos
desc_giro *a_giro; // Relación de giros

/////////////////////////////////////////////////////////////////////////////
void leer() {
        CFile temp;
        char *buff, *c_apt4, *c_apt5, *c_apt2;
        entero *a_apt, x, y, i, n_apt;
        desc_sit *p_dat;
        lugar_sit *i_apt;
        lugt_sit *l_apt;
        desc_giro *g_apt;

        buff  = (char*) malloc(200);

c_apt4 = (char*) malloc(5); *(c_apt4+4) = '\0';
c_apt5 = (char*) malloc(6); *(c_apt5+5) = '\0';
c_apt2 = (char*) malloc(3); *(c_apt2+2) = '\0';


        // Lectura de puntos del mapa

        temp.Open(na_datos,CFile::modeRead);

        temp.Read(buff,4);
        temp.Read(&n_apt,2);
        temp.Read(buff,2);
        a_lin = (entero*)
malloc((n_apt*2+1)*sizeof(entero));
        a_apt = a_lin;
        temp.Read(buff,90);

  for(i=1;i<=n_apt;i++) {
  temp.Read(c_apt4,1);
temp.Read(c_apt4,4); x = atoi(c_apt4);
temp.Read(c_apt4,4); y = atoi(c_apt4);

        *a_apt++ = x;
        *a_apt++ = y;
  };
  *a_apt++ = FINAL;
```

```
    temp.Close();

        // Lectura de Relación de giros

        temp.Open(na_giro,CFile::modeRead);

        temp.Seek(4,CFile::begin);
        temp.Read(&n_apt,2);
        temp.Seek(2,CFile::current);
        a_giro = (desc_giro*)
malloc((n_apt+1)*sizeof(desc_giro));
        g_apt = a_giro;
        temp.Seek(121,CFile::current);

    for(i=1;i<=n_apt;i++) {
                temp.Seek(1,CFile::current);
                g_apt->marca = 0;
                temp.Read(c_apt2,2);
                g_apt->giro = atoi(c_apt2);
                temp.Read(g_apt->nombre,36);
                g_apt->nombre[36] = '\0';
                temp.Read(c_apt4,1);
                g_apt->tipo = *c_apt4 - '0';

                *g_apt++;
    };
    g_apt->marca = FINAL;
    temp.Close();


//Lectura ubicacion en el archivo de Bienes y Servicios

        temp.Open(na_infosi,CFile::modeRead);
        temp.Seek(4,CFile::begin);
        temp.Read(&n_apt,2);
        temp.Seek(2,CFile::current);
        a_inf = (lugar_sit*) malloc(
        (n_apt+1)*sizeof(lugar_sit));
        i_apt = a_inf;
        temp.Seek(634,CFile::current);

    for(i=1;i<=n_apt;i++) {
    temp.Read(c_apt5,1);
    temp.Read(c_apt5,5); i_apt->reg  = atoi(c_apt5);
temp.Read(c_apt4,4); i_apt->x    = atoi(c_apt4);
temp.Read(c_apt4,4); i_apt->y    = atoi(c_apt4);
temp.Read(c_apt2,2); i_apt->giro = atoi(c_apt2);
temp.Seek(329,CFile::current);
        i_apt++;
    };
    i_apt->x = FINAL;
    i_apt->y = FINAL;

    temp.Close();

// Lectura sitios en el mapa

        temp.Open(na_datost,CFile::modeRead);
        temp.Read(buff,4);
        temp.Read(&n_apt,2);
        temp.Read(buff,2);
        a_dat = (desc_sit*)
malloc((n_apt+1)*sizeof(desc_sit));
        p_dat = a_dat;
        temp.Read(buff,121);

    for(i=1;i<=n_apt;i++) {
    temp.Read(c_apt5,1);
temp.Read(c_apt4,4); p_dat->x    = atoi(c_apt4);
temp.Read(c_apt4,4); p_dat->y    = atoi(c_apt4);

temp.Read(p_dat->titulo,30);
        p_dat->titulo[30] = '\0';
        p_dat++;
    };

    p_dat->x = FINAL;
    p_dat->y = FINAL;
    temp.Close();


// Lectura de lugares turisticos

        temp.Open(na_lugt,CFile::modeRead);
        temp.Read(buff,4);
        temp.Read(&n_apt,2);
        temp.Read(buff,2);
        a_lugt = (lugt_sit*)
malloc((n_apt+1)*sizeof(lugt_sit));
        l_apt = a_lugt;
        temp.Read(buff,89);

    for(i=1;i<=n_apt;i++) {
                temp.Read(c_apt5,1);
                l_apt->marca = 0;

                temp.Read(l_apt->nombre,36);
                temp.Read(l_apt->archivo,12);

                l_apt->nombre[36] = '\0';
                l_apt->archivo[12] = '\0';
                l_apt++;
    };

    l_apt->marca = FINAL;
    temp.Close();
        free(buff);
        free(c_apt5);
        free(c_apt4);
        free(c_apt2);
}

void trans(long x, long y, long *xp, long *yp) {
        *xp = ((x-min_x)/dif_x) * derecho;
        *yp = (1 - ((y-min_y)/dif_y)) * abajo;
}
//////////////////////////////////////////////////////////////////////
// CSiteView construction/destruction

CSiteView::CSiteView()
```

```
{
        // TODO: add construction code here
        stack = (long*) malloc( 500*sizeof(long));
        leer();
        fsit.Open(na_infosit,CFile::modeRead);

}

CSiteView::~CSiteView()
{
        fsit.Close();
}

/////////////////////////////////////////////////////////////////////////////
// CSiteView drawing

void CSiteView::OnDraw(CDC* pDC)
{
        CSiteDoc* pDoc = GetDocument();

        // TODO: add draw code here
        CClientDC dc( this );
        CRect rect, sitio;
        CBrush  sit_brush;
        entero *a_apt;
        long xp, yp, x, y;
        double fact_m;
        byte flag = 1;
        desc_sit *p_dat;
        lugar_sit *i_inf;

        GetClientRect( rect );

        dc.SetTextAlign( TA_BASELINE |
TA_CENTER );
        dc.SetTextColor( ::GetSysColor(
COLOR_WINDOWTEXT ) );
        dc.SetBkMode(TRANSPARENT);

        a_apt = a_lin;

        if (limite) {
                min_x = *a_apt++;
                max_x = *a_apt++;
                min_y = *a_apt++;
                max_y = *a_apt++;

                dif_x=max_x-min_x;
                dif_y=max_y-min_y;
                abajo=rect.bottom;
                derecho=rect.right;

                fact_m=dif_x/dif_y;
          if (abajo*fact_m>derecho)
                abajo=derecho/fact_m;
          else
                derecho=abajo*fact_m;

                a_stack = stack;
                *a_stack++ = (long) min_x;
                *a_stack++ = (long) max_x;
                *a_stack++ = (long) min_y;
                *a_stack++ = (long) max_y;

        } else {
                a_apt+=4;
                limite = 1;
  };

        while (*a_apt != FINAL) {

                x = *a_apt++;
                y = *a_apt++;

                if ((x!=MARCA) &&
(y!=MARCA)) {
                        flag = 1;
                } else {
                        x = *a_apt++;
                        y = *a_apt++;
                        flag = 0;
                };
                trans(x,y,&xp,&yp);

                if (flag) {dc.LineTo( (int) xp, (int)
yp);}
                        else  {dc.MoveTo( (int)
xp, (int) yp);};

        };

/*
        POINT pt;
        pt.x = 10;
        pt.y = 20;
        CClientDC dc(this);
        dc.SetPixel(pt.x,pt.y,RGB(255,0,0));
*/
        //dc.SelectStockObject(GRAY_BRUSH);

        p_dat = a_dat;
        while (p_dat->x!=FINAL) {
                        x   = p_dat->x;
                        y   = p_dat->y;

                        if (min_x<=x &&
x<=max_x && min_y<=y && y<=max_y) {

        trans(x,y,&xp,&yp);
                                dc.TextOut(
(int) xp, (int) yp, p_dat->titulo, 30 );
                        };

                        p_dat++;
        };
```

```
        i_inf = a_inf;
        while (i_inf->x!=FINAL) {
                x = i_inf->x;
                y = i_inf->y;

        if (i_inf->giro==giro_act && min_x<=x &&
x<=max_x &&
         min_y<=y && y<=max_y) {
        trans(x,y,&xp,&yp);
dc.Arc( (int) xp-ancho, (int) yp-ancho, (int) xp+ancho,
 (int) yp+ancho, (int) xp-ancho, (int) yp,
 (int) xp-ancho, (int) yp);
                        };

        i_inf++;
        };

        sit_brush.CreateSolidBrush(RGB(255,60,70)
);
        dc.SelectObject(&sit_brush);

        i_inf = a_inf;
        while (i_inf->x!=FINAL) {
                        x = i_inf->x;
                        y = i_inf->y;

        if (i_inf->giro==giro_act && min_x<=x &&
        x<=max_x &&
        min_y<=y && y<=max_y) {

        trans(x,y,&xp,&yp);
        dc.FloodFill( (int) xp, (int) yp,RGB(0,0,0));
       };

        i_inf++;
        }

/*
dc.SetPixel(xp,abajo-yp,RGB(255,0,0));
dc.TextOut( ( rect.right / 2 ), ( rect.bottom / 2 ),s,
s.GetLength() );
   CSiteView::OnDraw(&dc);
*/


}

/////////////////////////////////////////////////////////////////////////////
// CSiteView printing

BOOL CSiteView::OnPreparePrinting(CPrintInfo*
pInfo)
{
        // default preparation
        return DoPreparePrinting(pInfo);
}

void CSiteView::OnBeginPrinting(CDC* /*pDC*/,
CPrintInfo* /*pInfo*/)
{
        // TODO: add extra initialization before
printing
}

void CSiteView::OnEndPrinting(CDC* /*pDC*/,
CPrintInfo* /*pInfo*/)
{
        // TODO: add cleanup after printing
}



/////////////////////////////////////////////////////////////////////////////
// CSiteView diagnostics

#ifdef _DEBUG
void CSiteView::AssertValid() const
{
        CView::AssertValid();
}

void CSiteView::Dump(CDumpContext& dc) const
{
        CView::Dump(dc);
}

CSiteDoc* CSiteView::GetDocument() // non-debug
version is inline
{
        ASSERT(m_pDocument-
>IsKindOf(RUNTIME_CLASS(CSiteDoc)));
        return (CSiteDoc*) m_pDocument;
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CSiteView message handlers

void CSiteView::OnUbicarCiudad()
{

        // TODO: Add your command handler code
here
        CClientDC dc( this );
        CRect rect;

        GetClientRect( rect );

        limite = 1;
        InvalidateRect( rect ,TRUE);

}

void CSiteView::OnLButtonDown(UINT nFlags,
CPoint point)
{
        // TODO: Add your message handler code
here and/or call default
```

```
        CView::OnLButtonDown(nFlags, point);

        long xc, yc, xp, yp; //, x_inf, y_inf;
        long x, y, inc_x, inc_y, mid_x, mid_y;
        //unsigned short int *apt;
        double fact_m;
        unsigned int cont = 0, largo;
        lugar_sit *i_inf;
        Dinfo dlg;
        char *nom_com, *calle, *numero, *colonia,
*cp, *ho1, *ho2;
        char *ho3, *ho4, *pser, *cat, *tel, *rz, *prop,
*rfc;

        CClientDC dc( this );
        CRect rect;

        GetClientRect( rect );

        xc = point.x;
        yc = point.y;

        if (buscar) {
        x = (xc/derecho)*dif_x + min_x  ;
                y = (1 - yc/abajo)*dif_y + min_y ;

                mid_x = (long) dif_x/2.0;
                mid_y = (long) dif_y/2.0;
                inc_x=mid_x*(acerca/2.0);
                inc_y=mid_y*(acerca/2.0);

                if (x - (mid_x - inc_x) < min_x) {
                max_x = min_x + (dif_x - 2*inc_x);
                } else {
                if (x + (mid_x - inc_x) > max_x) {
                min_x = max_x - (dif_x - 2*inc_x);
                        } else {
                min_x = x - (mid_x - inc_x);
                max_x = x + (mid_x - inc_x);
                        }
                };

                if (y - (mid_y - inc_y) < min_y) {
                max_y = min_y + (dif_y - 2*inc_y);
                } else {
                if (y + (mid_y - inc_y) > max_y) {
                min_y = max_y - (dif_y - 2*inc_y);
                        } else {
                min_y = y - (mid_y - inc_y);
                max_y = y + (mid_y - inc_y);
                        }
                };

                dif_x=max_x-min_x;
                dif_y=max_y-min_y;
                abajo=rect.bottom;
                derecho=rect.right;

                fact_m=dif_x/dif_y;

                if (abajo*fact_m>derecho)
                        abajo= derecho/fact_m;
                else
                        derecho=abajo*fact_m;

                        *a_stack++ = (long) min_x;
                        *a_stack++ = (long) max_x;
                        *a_stack++ = (long) min_y;
                        *a_stack++ = (long) max_y;

                        limite = 0;
                        InvalidateRect( rect ,TRUE);

                } else {
                        i_inf = a_inf;
                        while (i_inf->x!=FINAL) {
                        x = i_inf->x;
                        y = i_inf->y;
                        trans(x,y,&xp,&yp);
                        if ((i_inf->giro==giro_act)    &&
                        (abs(xp-xc)<=ancho) &&
                        (abs(yp-yc)<=ancho)   ) {

fsit.Seek(643+345*cont,CFile::begin);

fsit.Seek(15,CFile::current);

                        rz = (char*) malloc((largo=40)+1);
                        fsit.Read(rz,largo);
                        *(rz+largo) = '\0';
                        dlg.m_rz = rz;

                        colonia = (char*) malloc((largo=40)+1);

fsit.Read(colonia,largo);
                        *(colonia+largo) = '\0';
                        dlg.m_colonia = colonia;

                        calle = (char*) malloc((largo=40)+1);
                        fsit.Read(calle,largo);
                        *(calle+largo) = '\0';
                        dlg.m_calle = calle;

                        numero = (char*) malloc((largo=15)+1);

fsit.Read(numero,largo);
                        *(numero+largo) = '\0';
                        dlg.m_numero = numero;

                        cp = (char*) malloc((largo=5)+1);
                        fsit.Read(cp,largo);
                        *(cp+largo) = '\0';
                        dlg.m_cp = cp;

                        nom_com = (char*) malloc((largo=40)+1);

                        fsit.Read(nom_com,largo);
                        *(nom_com+largo) = '\0';
                        dlg.m_nom_com = nom_com;
```

```
rfc = (char*) malloc((largo=13)+1);
fsit.Read(rfc,largo);
*(rfc+largo) = '\0';
dlg.m_rfc = rfc;

prop = (char*) malloc((largo=40)+1);
fsit.Read(prop,largo);
*(prop+largo) = '\0';
dlg.m_prop = prop;

tel = (char*) malloc((largo=10)+1);
fsit.Read(tel,largo);
*(tel+largo) = '\0';
dlg.m_tel = tel;

ho1 = (char*) malloc((largo=4)+1);
fsit.Read(ho1,largo);
*(ho1+largo) = '\0';
dlg.m_ho1 = ho1;

ho2 = (char*) malloc((largo=4)+1);
fsit.Read(ho2,largo);
*(ho2+largo) = '\0';
dlg.m_ho2 = ho2;

ho3 = (char*) malloc((largo=4)+1);
fsit.Read(ho3,largo);
*(ho3+largo) = '\0';
dlg.m_ho3 = ho3;

ho4 = (char*) malloc((largo=4)+1);
fsit.Read(ho4,largo);
*(ho4+largo) = '\0';
dlg.m_ho4 = ho4;

peer = (char*) malloc((largo=50)+1);
fsit.Read(peer,largo);
*(peer+largo) = '\0';
dlg.m_peer = peer;

cat = (char*) malloc((largo=20)+1);
fsit.Read(cat,largo);
*(cat+largo) = '\0';
dlg.m_cat = cat;

dlg.DoModal();
free(num_com);     free(calle);
free(numero); free(colonia);
free(cp); free(ho1); free(ho2);
free(ho3);  free(ho4);
free(peer); free(cat);
free(tel); free(rz); free(prop);
free(rfc);
                        };

i_inf++;
cont++;
            };
};
}
```

```
void CSiteView::OnCiudadIxtapa()
{
        // TODO: Add your command handler code
here
        CClientDC dc( this );
        CRect rect;

        GetClientRect( rect );

        ciudad = 0;
        na_datos  = "i_datos.dbf";
        na_info = "i_info.dbf";
        na_sit = "i_sit.dbf";
        na_datost = "i_datost.dbf";
        na_lugt   = "i_lugt.dbf";
        na_giro   = "i_gir.dbf";

        free(a_inf);
        free(a_lin);
        free(a_dat);
        free(a_lugt);
        free(a_giro);
        giro_act = 0, examinar=0, buscar=1;
        leer();
        fsit.Close();
        fsit.Open(na_info,CFile::modeRead);
        InvalidateRect( rect ,TRUE);
}

void CSiteView::OnCiudadZihuatanejo()
{
        // TODO: Add your command handler code
here
        CClientDC dc( this );
        CRect rect;

        GetClientRect( rect );

        ciudad = 1;
        na_datos  = "z_datos.dbf";
        na_infosi = "z_info.dbf";
        na_sit = "z_sit.dbf";
        na_datost = "z_datost.dbf";
        na_lugt   = "z_lugt.dbf";
        na_giro   = "z_gir.dbf";
        free(a_inf);
        free(a_lin);
        free(a_dat);
        free(a_lugt);
        free(a_giro);
        giro_act = 0, examinar=0, buscar=1;
        leer();
        fsit.Close();
        fsit.Open(na_info,CFile::modeRead);
        InvalidateRect( rect,TRUE);
}

void CSiteView::OnRButtonDown(UINT nFlags,
CPoint point)
```

```
{
        // TODO: Add your message handler code
here and/or call default

        double fact_m;
        CView::OnRButtonDown(nFlags, point);
        CClientDC dc( this );
        CRect rect;
        if (examinar) return;
        GetClientRect( rect );

        if (a_stack-4<=stack) {
                return;
        };

        a_stack-=4;
        min_x = (long) *(a_stack-4);
        max_x = (long) *(a_stack-3);
        min_y = (long) *(a_stack-2);
        max_y = (long) *(a_stack-1);

        dif_x= (double) max_x-min_x;
        dif_y= (double) max_y-min_y;
        abajo= (double) rect.bottom;
        derecho= (double) rect.right;

        fact_m=dif_x/dif_y;
  if (abajo*fact_m>derecho)
        abajo= derecho/fact_m;
  else
        derecho= abajo*fact_m;

        limite = 0;
        InvalidateRect( rect ,TRUE);

}

void CSiteView::OnUbicarRadiodeaccin()
{
        // TODO: Add your command handler code
here

        DRadio dlg;

        if (acerca <= 0.05)
                dlg.m_radio1 = 0;
        else if (acerca <= 0.25)
                dlg.m_radio1 = 1;
        else if (acerca <= 0.50)
                dlg.m_radio1 = 2;

        if (dlg.DoModal()==1) {
                if (dlg.m_radio1 == 0)
                        acerca = 0.05;
                else if (dlg.m_radio1 == 1)
                        acerca = 0.25;
                else if (dlg.m_radio1 == 2)
                        acerca = 0.50;
        }.

}

void CSiteView::OnSiteConocer()
{
        // TODO: Add your command handler code
here
        DCuadro dlg;
        CFile temp;
        double tam;
        char *buffer;

        temp.Open((ciudad)?"zihuata.txt":"ixtapa.txt"
,CFile::modeRead);
        tam = temp.GetLength();
        buffer = (char*) malloc(tam+1);
        temp.Read(buffer,tam);
        temp.Close();
        *(buffer+(int)tam) = '\0';

        dlg.m_titulo =
(ciudad)?"ZIHUATANEJO":"IXTAPA";
        dlg.m_texto = buffer;
        dlg.DoModal();
        free(buffer);
}

void
CSiteView::OnInformacinturisticaAtractivosturisticos()
{
        // TODO: Add your command handler code
here
        DSelecc dlg;
        DCuadro dlg2;
        CFile temp;
        double tam;
        char *buffer;
        lugi_sit *l_apt;

        dlg.m_sitio =
(ciudad)?"ZIHUATANEJO":"IXTAPA";
        dlg.m_lugi = a_lugi;

        if (dlg.DoModal()!=1) return;

        l_apt = a_lugi;

        while ((strcmp(dlg.m_lista,l_apt-
>nombre)!=0) && (l_apt->marca!=FINAL))
                l_apt++;

  if (l_apt->marca==FINAL) return;

        temp.Open( l_apt->archivo
,CFile::modeRead);
        tam = temp.GetLength();
        buffer = (char*) malloc(tam+1);
        temp.Read(buffer,tam);
        temp.Close();
        *(buffer+(int)tam) = '\0';
```

```
        dlg2.m_titulo = dlg.m_lista;
        dlg2.m_texto = buffer;
        dlg2.DoModal();
        free(buffer);

}


void CSiteView::OnInformacinturisticaBienes()
{
        // TODO: Add your command handler code
here
        DBienSer dlg;
        desc_giro *g_apt;
        CClientDC dc( this );
        CRect rect;

        dlg.m_tipo = 0;
        dlg.m_giro = a_giro;
        dlg.m_titulo = (ciudad)?"ZIHUATANEJO -
BIENES":"IXTAPA - BIENES";

        if (dlg.DoModal()!=1) return;

        g_apt = a_giro;

        while ((strcmp(dlg.m_lista,g_apt-
>nombre)!=0) && (g_apt->marca!=FINAL))
                g_apt++;

  if (g_apt->marca==FINAL) return;

        giro_act = g_apt->giro;
        GetClientRect( rect );
        limite = 0;
        InvalidateRect( rect ,TRUE);
}

void CSiteView::OnInformacinturisticaServicios()
{
        // TODO: Add your command handler code
here
        DBienSer dlg;
        desc_giro *g_apt;
        CClientDC dc( this );
        CRect rect;

        dlg.m_tipo = 1;
        dlg.m_giro = a_giro;
        dlg.m_titulo = (ciudad)?"ZIHUATANEJO -
SERVICIOS":"IXTAPA - SERVICIOS";

        if (dlg.DoModal()!=1) return;

        g_apt = a_giro;

        while ((strcmp(dlg.m_lista,g_apt-
>nombre)!=0) && (g_apt->marca!=FINAL))
                g_apt++;

  if (g_apt->marca==FINAL) return;

        giro_act = g_apt->giro;
        GetClientRect( rect );
        limite = 0;
        InvalidateRect( rect ,TRUE);

}

void CSiteView::OnUbicarLimpiar()
{
        // TODO: Add your command handler code
here
        CClientDC dc( this );
        CRect rect;
        GetClientRect( rect );

        giro_act = 0;
        GetClientRect( rect );
        limite = 0;
        InvalidateRect( rect ,TRUE);


}

void CSiteView::OnUbicarciudadBuscar()
{
        // TODO: Add your command handler code
here
        buscar = 1;
        examinar = 0;

}

void CSiteView::OnUbicarciudadExaminar()
{
        // TODO: Add your command handler code
here
        buscar = 0;
        examinar = 1;

}

void CSiteView::OnUpdateCiudadIxtapa(CCmdUI*
pCmdUI)
{
        // TODO: Add your command update UI
handler code here
        pCmdUI->SetCheck(!ciudad);

}

void
CSiteView::OnUpdateCiudadZihuatanejo(CCmdUI*
pCmdUI)
{
        // TODO: Add your command update UI
handler code here
        pCmdUI->SetCheck(ciudad);
}
```

```
void
CSiteView::OnUpdateUbicarciudadBuscar(CCmdUI*
pCmdUI)
{
        // TODO: Add your command update UI
handler code here
        pCmdUI->SetCheck(buscar);
}

void
CSiteView::OnUpdateUbicarciudadExaminar(CCmdUI
* pCmdUI)
{
        // TODO: Add your command update UI
handler code here
        pCmdUI->SetCheck(examinar);

}

void CSiteView::OnAppQuejas()
{
        // TODO: Add your command handler code
here
        DQuejas dlg;

        dlg.DoModal();
}

void CSiteView::OnAppAyuda()
{
        // TODO: Add your command handler code
here
        DAyuda dlg;

        dlg.DoModal();

}
```

ARCHIVO DE BIENES Y SERVICIOS
DBIENSER.CPP

```
// dbienser.cpp : implementation file
#include "stdafx.h"
#include "site.h"
#include "dbienser.h"
#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////////
// DBienSer dialog
DBienSer::DBienSer(CWnd* pParent /*=NULL*/)
        : CDialog(DBienSer::IDD, pParent)
{
        //{{AFX_DATA_INIT(DBienSer)
        m_titulo = "";
        m_lista = "";
        //}}AFX_DATA_INIT
}

void DBienSer::DoDataExchange(CDataExchange*
pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(DBienSer)
        DDX_Text(pDX, IDC_EDIT1, m_titulo);
        DDX_LBString(pDX, IDC_LIST1, m_lista);
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(DBienSer, CDialog)
        //{{AFX_MSG_MAP(DBienSer)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
/////////////////////////////////////////////////////////////////////////
```

Apendice

B

MODULO DE MANTENIMIENTO EN PARADOX.

Para este módulo; la aplicación se correrá en el ambiente de DOS, en el cual se deberá tener acceso al subdirectorio de Paradox, tecleando el siguiente comando desde la raíz del disco duro :

C:\PDOX35\PPROG\PPROG [ENTER]

El archivo PPROG.EXE, es el ejecutable que permite que la aplicación se lleve acabo en Paradox. Una vez tecleado este comando, Paradox abrirá un menú de opciones en la pantalla del monitor; dentro de ese menú principal se deberá escoger la opción :

INICIAR [ENTER]

Al tomar esta opción el paquete preguntará por el nombre de la aplicación a correr, se deberá teclear el nombre de :

MANTE [ENTER]

Este nombre corresponde a la aplicación del módulo de mantenimiento. Una vez que se teclea éste, aparecerá en la pantalla el menú principal del módulo de mantenimiento de las bases de datos del sistema de información turística.

Dentro del menú principal del módulo, aparecerán las siguientes opciones :

Altas **Bajas** **Modificaciones** **Consultas** **Salir**

Al elegir la opción de ***ALTAS***, el sistema asume que se trata de un proceso de altas de información en cada una de las bases de datos del sistema, que en este caso, serán aquéllas en las cuales dicha información necesite actualizarse y darse de alta. El programa desplegará otro submenú dentro de esta opción, el cual contendrá las siguientes opciones:

Giros **Información-Gral.** **Productos** **Precios**

En cada una de estas opciones, los catálogos de *GIROS*, *INFORMACION-GRAL.*, ***PRODUCTOS*** y ***PRECIOS***, se generará un registro dependiendo de la opción que se elija, y para dicho registro se deberá introducir la información que solicite cada una de éstas.

Para salir de cada una de las opciones y no realizar la operación se debe hacer con la tecla de **[ESC]**.

Al elegir la opción de ***BAJAS*** del menú principal, el sistema asume que se trata de un proceso de baja de información en cada una de las bases de datos del sistema, que en este caso, serán aquéllas en las cuales dicha información necesite actualizarse y darse de baja. El programa desplegará otro submenú dentro de esta opción, el cual contendrá las siguientes opciones:

Giros **Información-Gral.** **Productos** **Precios**

En cada una de estas opciones, los catálogos de *GIROS*, *INFORMACION-GRAL.*, ***PRODUCTOS*** y ***PRECIOS***, se borrará un registro dependiendo de la opción que se elija.

Para salir de cada una de las opciones y no realizar la operación se debe hacer con la tecla de **[ESC]**.

Al elegir la opción de ***MODIFICACIONES*** del menú principal, el sistema asume que se trata de un proceso de modificación de información en cada una de las bases de datos del sistema, que en este caso, serán aquéllas en las cuales dicha información necesite actualizarse y modificarse. El programa desplegará otro submenú dentro de esta opción, el cual contendrá las siguientes opciones:

Giros **Información-Gral.** **Productos** **Precios**

En cada una de estas opciones, los catálogos de *GIROS*, *INFORMACION-GRAL.*, ***PRODUCTOS*** y ***PRECIOS***, se modificarán los campos de un registro dependiendo de la opción que se elija.

Para salir de cada una de las opciones y no realizar la operación se debe hacer con la tecla de **[ESC]**.

Al elegir la opción de ***CONSULTAS*** del menú principal, el sistema asume que se trata de un proceso de consulta de información en cada una de las bases de datos del sistema, que en este caso, serán aquéllas en las cuales dicha información necesite consultarse. El programa desplegará otro submenú dentro de esta opción, el cual contendrá las siguientes opciones:

Giros **Información-Gral.** **Productos** **Precios**

En cada una de estas opciones, los catálogos de *GIROS*, *INFORMACION-GRAL.*, ***PRODUCTOS*** y ***PRECIOS***, se consultará un registro dependiendo de la opción que se elija.

Para salir de cada una de las opciones y no realizar la operación se debe hacer con la tecla de **[ESC]**.

Al elegir la opción de ***SALIR*** del menú principal, el programa desplegará otro submenú dentro de esta opción, el cual contendrá las siguientes opciones:

NO **SI**

Para la opción de ***NO***, el sistema regresará a la opción de salir y se mantendrá dentro de éste.

Para la opción de ***SI***, el sistema saldra de la aplicación y regresará al Sistema Operativo o a la aplicación de Paradox, dependiendo de la ruta de salida que le demos.

MODULO DE MANTENIMIENTO.
CODIGO GENERADO POR PARADOX.

```
; Mante

if (sysmode() <> "MenúPr") then
  Message "La aplicación sólo puede arrancar desde el modo principal de Paradox"
  Sleep 3000
  return
endif

Echo Off
Clear
Reset
Cursor Off



; preguntar la contraseña de la aplicación; esta contraseña determina
;   el acceso a las tablas en la aplicación permitido para
;   usuario activo de la aplicación.

@ 0, 0
Style Attribute SysColor(0)
?? fill(" ",160)
@ 1, 0
?? "Introduzca la ctraseña. de la aplicación; [Esc] cancelar; [Intro] sin contraseña"
@ 0, 0
?? "Contraseña: "
Cursor Normal
zzzcolor = int(SysColor(0) / 16)
Style Attribute ((zzzcolor * 16) + zzzcolor)
Accept "a50" To pword
Style
EscEnter = not retval
Cursor Off

if (EscEnter) then
  Message "Cancelando la aplicación"
  Sleep 2000
  Clear
  return
endif
```

```
if (pword <> "") then
  Password pword
endif


; fijar el procedimiento de error para la aplicación

ReadLib "Manteutl" ApplicErrorProc
ErrorProc = "ApplicErrorProc"
ApplicErrorRetVal = FALSE


; Arrancar la aplicación

ReadLib "Mantel" MantelMenu
MantelMenu()
Release Procs MantelMenu

Clearall
if (pword <> "") then
  UnPassword pword
endif


 ; Mantel


AppLib = "Mantel"
if (not isfile(AppLib + ".lib")) then
  Createlib AppLib
endif


proc MantelMenu()
private x, escape, zzzmexit, zzzzexit, pword

  zzzzexit = FALSE
  x = "Altas"
  while (TRUE)
    Clear

    ShowMenu
       "Altas": "Alta de registros en los Catalogos",
       "Bajas": "Bajas de Información de registros en Catalogos",
```

```
  "Modificaciones": "Modificaciones a registros de Catalogos",
  "Consultas": "Consultas a los Catalogos",
  "Salir": "Salir de la aplicación"
 Default x
 To x

switch
 case x = "Altas":
  ReadLib "Mante1" Mante2Menu
  escape = Mante2Menu()
  escape = not escape
  Release Procs Mante2Menu

 case x = "Bajas":
  ReadLib "Mante2" Mante4Menu
  escape = Mante4Menu()
  escape = not escape
  Release Procs Mante4Menu

 case x = "Modificaciones":
  ReadLib "Mante2" Mante5Menu
  escape = Mante5Menu()
  escape = not escape
  Release Procs Mante5Menu

 case x = "Consultas":
  ReadLib "Mante1" Mante3Menu
  escape = Mante3Menu()
  escape = not escape
  Release Procs Mante3Menu

 case x = "Salir":
ShowMenu
  "No": "No salir de la aplicación",
  "Sí": "Salir de la aplicación"
 To zzzmexit

zzzzexit = (zzzmexit = "Sí")
escape = (zzzmexit = "Esc")

 case x = "Esc":
  escape = FALSE
endswitch

Reset
```

```
; reincializar el valor de ErrorProc
   ErrorProc = "ApplicErrorProc"
   ApplicErrorRetVal = FALSE

   if (zzzzexit) then
     return TRUE
   endif

   if (not escape) then
     x = "Altas"
   endif
 endwhile
endproc

Writelib AppLib Mante1Menu
Release Procs Mante1Menu


 ; Mante2


AppLib = "Mante1"
if (not isfile(AppLib + ".lib")) then
  Createlib AppLib
endif


proc Mante2S1()
private opResult

  Readlib "Manteutl" EntryTable, KECheck, ToggleForm,
       EdFldView, HelpKey, EntryCancel, EntryDolt,
       RenamePre, RenameSet, SaveList, CreateList,
       PrintList

  opResult = EntryTable("Catgirm", "", "1", FALSE)

  Release Procs EntryTable, KECheck, ToggleForm,
       EdFldView, HelpKey, EntryCancel, EntryDolt,
       RenamePre, RenameSet, SaveList, CreateList,
       PrintList

  return opResult
endproc
```

```
Writelib AppLib Mante2S1
Release Procs Mante2S1

proc Mante2S2()
private opResult

 Readlib "Manteutl" EntryTable, KECheck, ToggleForm,
     EdFldView, HelpKey, EntryCancel, EntryDoIt,
     RenamePre, RenameSet, SaveList, CreateList,
     PrintList

 opResult = EntryTable("Inform", "", "F", FALSE)

 Release Procs EntryTable, KECheck, ToggleForm,
     EdFldView, HelpKey, EntryCancel, EntryDoIt,
     RenamePre, RenameSet, SaveList, CreateList,
     PrintList

 return opResult
endproc

Writelib AppLib Mante2S2
Release Procs Mante2S2

proc Mante2S3()
private opResult

 Readlib "Manteutl" EntryTable, KECheck, ToggleForm,
     EdFldView, HelpKey, EntryCancel, EntryDoIt,
     RenamePre, RenameSet, SaveList, CreateList,
     PrintList

 opResult = EntryTable("Cprodum", "", "F", FALSE)

 Release Procs EntryTable, KECheck, ToggleForm,
     EdFldView, HelpKey, EntryCancel, EntryDoIt,
     RenamePre, RenameSet, SaveList, CreateList,
     PrintList

 return opResult
endproc

Writelib AppLib Mante2S3
Release Procs Mante2S3
```

```
proc Mante2S4()
private opResult

 Readlib "Manteutl" EntryTable, KECheck, ToggleForm,
      EdFldView, HelpKey, EntryCancel, EntryDolt,
      RenamePre, RenameSet, SaveList, CreateList,
      PrintList

 opResult = EntryTable("Precim", "", "F", FALSE)

 Release Procs EntryTable, KECheck, ToggleForm,
      EdFldView, HelpKey, EntryCancel, EntryDolt,
      RenamePre, RenameSet, SaveList, CreateList,
      PrintList

 return opResult
endproc

Writelib AppLib Mante2S4
Release Procs Mante2S4

proc Mante2Menu()
private x, escape

 x = "Giros"
 while (TRUE)
  Clear

  ShowMenu
     "Giros": "Alta de registros del Catalogo de Giros",
     "Información_Gral.": "Alta de Información General de Prestadores Bienes-Servicios",
     "Productos": "Alta de registros del catálogo de productos",
     "Precios": "Alta de registros del catálogo de precios"
    Default x
    To x

  switch
   case x = "Giros":
    ReadLib "Mante1" Mante2S1
    escape = Mante2S1()
    escape = not escape
    Release Procs Mante2S1

   case x = "Información_Gral.":
```

```
      ReadLib "Mante1" Mante2S2
      escape = Mante2S2()
      escape = not escape
      Release Procs Mante2S2

    case x = "Productos":
      ReadLib "Mante1" Mante2S3
      escape = Mante2S3()
      escape = not escape
      Release Procs Mante2S3

    case x = "Precios":
      ReadLib "Mante1" Mante2S4
      escape = Mante2S4()
      escape = not escape
      Release Procs Mante2S4

    case x = "Esc":
      return FALSE
   endswitch

; reincializar el valor de ErrorProc
   ErrorProc = "ApplicErrorProc"
   ApplicErrorRetVal = FALSE

   if (not escape) then
    return TRUE
   endif
 endwhile
endproc

Writelib AppLib Mante2Menu
Release Procs Mante2Menu


; Mante3


AppLib = "Mante1"
if (not isfile(AppLib + ".lib")) then
 Createlib AppLib
endif
```

```
proc Mante3S1()
private opResult, tbl

 Play "Manteq1"    ; poner la consulta en el área de trabajo
 if (ApplicErrorRetVal) then
  ClearAll
  return FALSE
 endif

 Readlib "Manteutl" QueryDolt
 rt = QueryDolt()
 Release Procs QueryDolt

 if (not rt) then
  return FALSE
 endif

 Readlib "Manteutl"  ViewTable, ToggleForm, VwFldView,
     HelpKey

 opResult = ViewTable("Solucion", "Mantes1", "F", FALSE)

 Release Procs ViewTable, ToggleForm, VwFldView,
     HelpKey

 return opResult
endproc

Writelib AppLib Mante3S1
Release Procs Mante3S1

proc Mante3S2()
private opResult, tbl

 Play "Manteq2"    ; poner la consulta en el área de trabajo
 if (ApplicErrorRetVal) then
  ClearAll
  return FALSE
 endif

 Readlib "Manteutl" QueryDolt
 rt = QueryDolt()
 Release Procs QueryDolt

 if (not rt) then
```

```
  return FALSE
 endif

 Readlib "Manteutl"  ViewTable, ToggleForm, VwFldView,
     HelpKey

 opResult = ViewTable("Solucion", "Mantes2", "F", FALSE)

 Release Procs ViewTable, ToggleForm, VwFldView,
     HelpKey

 return opResult
endproc

Writelib AppLib Mante3S2
Release Procs Mante3S2

proc Mante3S3()
private opResult

 Readlib "Manteutl"  ViewTable, ToggleForm, VwFldView,
     HelpKey

 opResult = ViewTable("Cprodum", "Cprodum", "1", FALSE)

 Release Procs ViewTable, ToggleForm, VwFldView,
     HelpKey

 return opResult
endproc

Writelib AppLib Mante3S3
Release Procs Mante3S3

proc Mante3S4()
private opResult

 Readlib "Manteutl"  ViewTable, ToggleForm, VwFldView,
     HelpKey

 opResult = ViewTable("Precim", "Precim", "1", FALSE)

 Release Procs ViewTable, ToggleForm, VwFldView,
     HelpKey
```

```
 return opResult
endproc

Writelib AppLib Mante3S4
Release Procs Mante3S4

proc Mante3Menu()
private x, escape

 x = "Giros"
 while (TRUE)
  Clear

  ShowMenu
     "Giros": "Consultas al catálogo de giros",
     "Información_Gral.": "Consultas al catálogo de información gral. de prestadores".
     "Productos": "Consultas al catálogo de productos",
     "Precios": "Consultas al catálogo de precios"
    Default x
    To x

  switch
   case x = "Giros":
    ReadLib "Mante1" Mante3S1
    escape = Mante3S1()
    escape = not escape
    Release Procs Mante3S1

   case x = "Información_Gral.":
    ReadLib "Mante1" Mante3S2
    escape = Mante3S2()
    escape = not escape
    Release Procs Mante3S2

   case x = "Productos":
    ReadLib "Mante1" Mante3S3
    escape = Mante3S3()
    escape = not escape
    Release Procs Mante3S3

   case x = "Precios":
    ReadLib "Mante1" Mante3S4
    escape = Mante3S4()
    escape = not escape
    Release Procs Mante3S4
```

```
    case x = "Esc":
      return FALSE
    endswitch

; reincializar el valor de ErrorProc
    ErrorProc = "ApplicErrorProc"
    ApplicErrorRetVal = FALSE

    if (not escape) then
      return TRUE
    endif
  endwhile
endproc

Writelib AppLib Mante3Menu
Release Procs Mante3Menu


 ; Mante4


AppLib = "Mante2"
if (not isfile(AppLib + ".lib")) then
  Createlib AppLib
endif


proc Mante4S1()
private opResult

  if (isempty("Catgirm")) then
    Message "No hay registros que editar"
    Sleep 3000
    return FALSE
  endif

  if (ApplicErrorRetVal) then
    return FALSE
  endif

  Readlib "Manteutl" EditTable, ToggleForm, EdFldView,
      HelpKey, EditCancel, SEditDoIt, SEditDelNoIns
```

```
 opResult = EditTable("Catgirm", "Catgirm", "", "3", FALSE,
        "SEditDoIt", "SEditDelNoIns",
        "[Supr] Borrar registro",
        TRUE, FALSE, FALSE)

 Release Procs EditTable, ToggleForm, EdFldView,
     HelpKey, EditCancel, SEditDoIt, SEditDelNoIns


 return opResult
endproc

Writelib AppLib Mante4S1
Release Procs Mante4S1

proc Mante4S2()
private opResult

 if (isempty("Inform")) then
  Message "No hay registros que editar"
  Sleep 3000
  return FALSE
 endif

 if (ApplicErrorRetVal) then
  return FALSE
 endif

 Readlib "Manteutl" EditTable, ToggleForm, EdFldView,
     HelpKey, EditCancel, SEditDoIt, SEditDelNoIns

 opResult = EditTable("Inform", "Inform", "", "2", FALSE,
        "SEditDoIt", "SEditDelNoIns",
        "[Supr] Borrar registro",
        TRUE, FALSE, FALSE)

 Release Procs EditTable, ToggleForm, EdFldView,
     HelpKey, EditCancel, SEditDoIt, SEditDelNoIns


 return opResult
endproc

Writelib AppLib Mante4S2
Release Procs Mante4S2
```

```
proc Mante4S3()
private opResult

 if (isempty("Cprodum")) then
  Message "No hay registros que editar"
  Sleep 3000
  return FALSE
 endif

 if (ApplicErrorRetVal) then
  return FALSE
 endif

 Readlib "Manteutl" EditTable, ToggleForm, EdFldView,
     HelpKey, EditCancel, SEditDoIt, SEditDelNoIns

 opResult = EditTable("Cprodum", "Cprodum", "", "3", FALSE,
        "SEditDoIt", "SEditDelNoIns",
        "[Supr] Borrar registro",
        TRUE, FALSE, FALSE)

 Release Procs EditTable, ToggleForm, EdFldView,
     HelpKey, EditCancel, SEditDoIt, SEditDelNoIns


 return opResult
endproc

Writelib AppLib Mante4S3
Release Procs Mante4S3

proc Mante4S4()
private opResult

 if (isempty("Precim")) then
  Message "No hay registros que editar"
  Sleep 3000
  return FALSE
 endif

 if (ApplicErrorRetVal) then
  return FALSE
 endif
```

```
 Readlib "Manteutl" EditTable, ToggleForm, EdFldView,
     HelpKey, EditCancel, SEditDoIt, SEditDelNoIns

 opResult = EditTable("Precim", "Precim", "", "3", FALSE,
         "SEditDoIt", "SEditDelNoIns",
         "[Supr] Borrar registro",
         TRUE, FALSE, FALSE)

 Release Procs EditTable, ToggleForm, EdFldView,
     HelpKey, EditCancel, SEditDoIt, SEditDelNoIns


 return opResult
endproc

Writelib AppLib Mante4S4
Release Procs Mante4S4

proc Mante4Menu()
private x, escape

 x = "Giros"
 while (TRUE)
  Clear

  ShowMenu
     "Giros": "Bajas a registros del catalogo de giros",
     "Información_Gral.": "Bajas a registros de información general",
     "Productos": "Bajas a registros de catálogo de productos",
     "Precios": "Bajas a registros de catálogo de precios"
    Default x
    To x

  switch
   case x = "Giros":
    ReadLib "Mante2" Mante4S1
    escape = Mante4S1()
    escape = not escape
    Release Procs Mante4S1

   case x = "Información_Gral.":
    ReadLib "Mante2" Mante4S2
    escape = Mante4S2()
    escape = not escape
    Release Procs Mante4S2
```

```
    case x = "Productos":
      ReadLib "Mante2" Mante4S3
      escape = Mante4S3()
      escape = not escape
      Release Procs Mante4S3

    case x = "Precios":
      ReadLib "Mante2" Mante4S4
      escape = Mante4S4()
      escape = not escape
      Release Procs Mante4S4

    case x = "Esc":
      return FALSE
   endswitch

; reincializar el valor de ErrorProc
   ErrorProc = "ApplicErrorProc"
   ApplicErrorRetVal = FALSE

   if (not escape) then
     return TRUE
   endif
  endwhile
endproc

Writelib AppLib Mante4Menu
Release Procs Mante4Menu


 ; Mante5


AppLib = "Mante2"
if (not isfile(AppLib + ".lib")) then
  Createlib AppLib
endif


proc Mante5S1()
private opResult

  if (isempty("Catgirm")) then
```

GLOSARIO

GLOSARIO

ANSI SQL.- Standard Scalable and Portable.

ANSI++.- Librería Estandar de C.

ANSI-SPARC.- (American National Standards Institute/Systems Planning and Requirements Committee). Se le conoce principalmente por su propuesta de crear una arquitectura de bases de datos, según la cual éstas se definen en tres niveles: esquema conceptual, esquema externo y esquema interno.

ANSI.- (American National Standards Institute) Intituto Nacional Americano de Normalización. Organización que determina las normas relativas al hardware, en puntos tales como: protocolos de nivel de enlace, posiciones y significado de las pastillas en los chips, registro en cinta y disco, y algunas normas para el software.

ARGC.- Argumento de C y C++.

ARGV.- Argumento por valor de C y C++.

BENCHMARK.- Prueba de características; evaluación de rendimiento; punto de referencia en comparar la medida del rendimiento con la de otros (hardware y software) que hayan sido sometidos a la misma prueba de características.

CASE.- Computer Aided Software Engineering. Ingeniería de Software Asistida por Computadora. Software que se utiliza en cualquiera o en todas las fases del desarrollo de un sistema de información, incluyendo análisis, diseño y programación. Por ejemplo, los diccionarios de datos y herramientas de diagramación ayudan en las fases de análisis y diseño, mientras que los generadores de aplicaciones aceleran la fase de programación.

CHARACTER (CHAR).- Elemento de un conjunto dado de caracteres, subdivisión de una palabra de máquina, que comprende, generalmente, seis, siete u ocho bits.

CADASYL.- (Conference on Data Systems Languages). Conferencia sobre lenguajes de sistemas de datos. Se fundó en una reunión convocada en el pentagonó en 1959. Su objetivo era, primeramente, el desarrollo de un lenguaje normalizado de programación para el proceso de datos.

DAG.- Gráfica Aciclica Dirigida.

DBMS.- (Data Base Management System). Sistema Manejador de Base de Datos.

DBSCHEMA.- Esquema de la base de datos.

ENTIDAD.- Es un objeto que existe y es distinguible de otros objetos.

GIF (Formato).- Graphics Interchange Format. Estandard de compuserve para la definición de imágenes a color.

GLOBAL COLOR MAP.- Mapa Global de Color. Recomendada para imágenes donde la exactitud del color es deseada.

HFS.- Estructura de árbol jerárquico de directorios y archivos.

ICONO.- Una diminuta representación pictórica de un objeto, tal como una aplicación, archivo o unidad de disco, que se utiliza en interfaces gráficas de usuario (GUI).

IDA.- Intelligent Drive Array.

IMAGE DESCRIPTOR.- Descriptor de Imágen. Define la colocación actual y la extensión de la siguiente imágen dentro del espacio definido en el descriptor de pantalla.

INFORMIX-SQL.- Manejador de base de datos Secuential Query Language.

INTERFACE.- Una conexión e interacción entre hardware, software y usuario.

MAIN( ).- Función de C y C++.

OBJETO.- Unidad única en la que se encapsulan código y datos.

OLE.- Objetos Ligados y Empotrados.

ORDEN.- De un árbol, es el número de descendientes que puede tener un nodo.

OUTLETS.- Técnica de corolario para lograr la comunicación.

PATH.- Ruta que tiene un directorio o archivo dentro de la memoria de la computadora.

PIXEL.- Término derivado de picture element: elemento de imagen. Uno de los elementos en un orden (o matriz) grande que contiene información gráfica. Guarda datos que representan el brillo y, posiblemente, el color de una pequeña región de la imagen.

RANDOM READ.- Lectura Aleatoria.

RASTER DATA.- Rastreo de Datos.

RELACION.- Es una asociación entre varias entidades.

SCREEN DESCRIPTOR.- Descriptor de pantalla. Describe el total de parámetros para las imágenes GIF.

SQL.- Standard Query Language: Lenguaje Normalizado de Consulta

TIFF (Formato).- Tag Image File Format.

TObject.- Clase abstracta de un lenguaje de programación.