UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
PLANTEL ARAGON

LA IMPORTANCIA DE OPENSTEP COMO UNA ALTERNATIVA MAS DE LA PROGRAMACION ORIENTADA A OBJETOS RUMBO AL SIGLO XXI

Tesis que para obtener el título de:

INGENIERO EN COMPUTACION

P R E S E N T A N:

Alejandro René González Ponce
Liliana Hernández Cervantes

DIRECTOR DE TESIS:
Ing. Donaciano Jiménez Vázquez

San Juan de Aragón, Edo. de México 1997

Dedicatorias

A mis Papas:

Sra. Maria Elisa Cervantes R.
Sr. Ramón Hernández Bonilla.

Muchas gracias por todo lo que me han dado, por su amor, apoyo y comprensión, Por ser los mejores papás del mundo. Y sobre todo por haber permitido que esto fuera realidad.

A mis hermanos:
Maribel, Ramón, Ivonne y Paola

Por que son el motivo que me impulsa a seguir luchando.

A mi abuelita:

Ma. Concepción R.

Te quiero muchísimo y aunque quizás no puedas leer esto. Gracias por cuidarme siempre y por enseñarme que hasta el final se tiene que seguir luchando.

A:
Catalina Hernández e Isaac Astorga Flores
Ma. de Jesús Hernández y Miguel Nava Duarte
Ana María Mundo y Bernardino Hernández

Gracias por el amor y apoyo incondicional que durante toda mi vida me han brindado. Este triunfo también es de ustedes.

A mis tíos y primos:

Porque siempre me impulsan a ser mejor cada día.

A Alejandro René González:

Gracias por permitirme compartir contigo esta experiencia, por apoyarme siempre por ser ejemplo de superación y lucha, pero sobre todo por ser lo mejor de mi vida.

A la familia González Ponce:

Por toda la confianza y apoyo que desinteresadamente me han brindado.

Liliana Hernández Cervantes

Dedicatorias

A mis Papas:

Mtro. en P. y C.P. Norberto González Covarrubias
Lic. Bertha Ponce Marquez

Gracias por siempre brindarme todo
su apoyo, lo que ha hecho posible la realización
de todos mis triunfos en la vida, de los cuales
les debo todo.

A mi hermano:
Luis Bernardo González Ponce

Gracias por apoyarme en momentos
gratos y amargos, y fomentar mi
superación.

A mis Abuelitos:

Fernando, Raquel , Santos y Esperanza
Son las personas que siempre creyeron y
han creido en mí, dandome siempre todo
su apoyo, sabiduria y comprensión.

A mis Tios y Primos:

Los cuales siempre me han brindado
sus consejos para permitir que cada dia
sea alguien mejor.

A Liliana Hernández:

Gracias por ser la persona que ha dado
un nuevo rumbo a mi vida y me ha ayudado
a obtener muchos triunfos, más lo que me faltan.

Alejandro René González Ponce

Dedicatorias

Al Ing. Sergio F. Beltrán.

Gracias por creer en nosotros y
brindarnos todo su apoyo para
la realización de este y otros
proyectos.

Al

Ing. Fabio Montoya
Ing. Rafael Prieto M.
Ocean. Gabriel A. López M.
Ing. Francisco de Urquijo

y a todas las personas que permitieron
fuera posible la realización de este trabajo,
les agradecemos enormemente su apoyo
y consejos .

A nuestro Asesor : Ing. Donaciano Jímenez V.

Por habernos apoyado en todo momento
y por todas sus valiosas observaciones.

A nuestros Sinodales:

Ing. Silvia Vega M.
Ing. Ernesto Peñaloza R.
Ing. Lilia Enciso G.
Ing. Yolanda Cuevas S.

Por sus valiosos comentarios y apoyo brindado.

A nuestros amigos y compañeros, los cuales
compartieron con nosotros muchas experiencias

A la Universidad Nacional Autonoma de México:

Por permitirnos haber ingresado a sus aulas y
ser fuente inagotable de sabiduria.

A México.

INDICE

Página

CAPITULO IV APLICACIONES PRACTICAS.

INTRODUCCION

En el campo de la computación se hace cada vez más necesario el desarrollo de herramientas que simplifiquen nuestro trabajo y optimicen nuestros recursos. Así mismo los cambios se dan de manera vertiginosa por lo que nosotros como universitarios y futuros profesionales tenemos la responsabilidad de estar al tanto de los acontencimientos.

La programación orientada a objetos no es nueva, ésta surgió en 1967 con el programa SIMULA 67 como un proyecto para poder hacer simulaciones sencillas de problemas mediante la representación de objetos reales, sin embargo actualmente tiene mucho auge debido a las características y ventajas que ofrece.

OpenStep es un sistema operativo orientado a objetos que nos proporciona todo un ambiente de desarrollo para la creación de aplicaciones basadas en ésta metodología. Dicho sistema cuenta con herramientas poderosas y una biblioteca de clases muy amplia, con la cual se pueden crear otras clases, además de manejar objetos portables que pueden ser utilizados en diferentes arquitecturas.

El presente trabajo tiene como objetivo mostrar las ventajas, características y herramientas de desarrollo de OpenStep como una alternativa diferente de la programación orientada a objetos, así como aplicar y/o reforzar los conocimientos que se tengan del tema.

Nuestro trabajo está estructurado en cuatro capítulos.

Iniciamos en el capítulo uno haciendo una breve reseña de las distintas metodologías de programación que antecedieron a la orientación a objetos, se dan los conceptos de objeto, orientación a objetos, sistema operativo, sistema operativo orientado a objetos para finalmente explicar los conceptos básicos de la orientación a objetos que son necesarios para una programación más dinámica dentro de ést ambiente.

Continuamos en el segundo capítulo con un conocimiento de lo que es el sistema operativo OpenStep, sus características, sus ventajas y perspectivas a futuro; así como una descripción detallada de su ambiente de trabajo.

En el tercer capítulo se explica el uso y características de las herramientas para el desarrollo de aplicaciones en OpenStep. Aquí mismo se describe el funcionamiento y la forma de aplicar las herramientas dentro de este ambiente entre las que destacamos a Objective C que es el lenguaje de programación en el que se desarrollan las aplicaciones, el IconBuilder que permite crear y/o editar imágenes e iconos, el InterfaceBuilder que facilita la realización de las interfaces gráficas y el ProjectBuilder que nos permite crear aplicaciones en las que podemos usar conjuntamente todas las herramientas.

En el cuarto capítulo se ejemplifica el uso de herramientas con aplicaciones prácticas, que aún cuando sencillas tratan de dejar en el lector la curiosidad y el deseo de desarrollar programas más complejos que puedan ser motivo de otras tesis.

Finalizamos nuestro trabajo con una serie de conclusiones.

Es importante destacar que la intención es dar a conocer una alternativa más de programación que permita a lectores e interesados en el tema a profundizar más en el mismo, porque la investigación es otro campo que compete a nuestro ámbito profesional y esperamos crear el deseo de las próximas generaciones por emprender nuevas alternativas dentro de nuestra profesión.

CAPITULO I

" LA PROGRAMACION ORIENTADA A OBJETOS "

1.1 Antecedentes

Con el inicio de la computación se cambió la forma de concebir al mundo, lo que provocó que a la par de las computadoras se fueran desarrollando nuevos lenguajes de programación que satisfacieran esas demandas; de está manera se dasarrollaron diferentes formas de pensamiento, lo que provocó que surgieran las distintas metodologías de programación que conocemos hasta la fecha.

Estas metodologías resolvieron los problemas a los que se enfrentaron; sin embargo, cada vez se han hecho más complejos y hay que resolverlos en un menor tiempo.

Mencionaremos algunas de las características más importantes de las metodologías que antecedieron a la Programación Orientada a Objetos sin profundizar en el tema ya que éste no es el objetivo del presente trabajo; sin embargo, sí nos extenderemos un poco en el tema de nuestro interés que es precisamente la Programación Orientada a Objetos; por lo que consideramos importante mencionarlas para comprender mejor la Programación Orientada a Objetos.

En un principio la programación de las computadoras se hacía por medio de código binario y lenguaje emsamblador, estó era suficiente ya que los problemas a resolver eran simples; conforme se fueron haciendo más complejos surgieron las distintas metodologías de programación que son:

1.- Programación Lineal
2.- Programación Modular

3.- Programación Estructurada
4.- Programación Orientada a Objetos

1.1.1 La Programación Lineal

Los primeros lenguajes de programación, estaban diseñados para realizar programas muy simples de menos de 100 líneas de código; sin embargo conforme fueron creciendo las necesidades, los lenguajes de programación se volvieron inadecuados para esas tareas. Los lenguajes de programación existentes eran las primeras versiones de BASIC; COBOL y FORTRAN; estos no tenían la facilidad para reutilizar el código, ya que los programas se ejecutaban en secuencias lógicas haciendo la lógica difícil de comprender.

El control del programa era difícil y se producían continuos saltos a lo largo del programa. Todos los datos eran globales, y podían ser modificados por cualquier parte del programa, cualquier modificación en un dato afectaba a todo el programa.

Con la evolución de los programas se desarrollo una nueva estructura el procedimiento (subprograma o subrutina) que consistía en secuencias de instrucciones que realizaban una tarea determinada, sin embargo está nueva estructura no resolvió el problema.

1.1.2 La Programación Modular

La Programación Modular proponía romper los programas grandes en componentes más pequeños que pudieran ser construidos independientemente,

para después combinarlos y formar un programa completo; el soporte de la Programación Modular fue la subrutina, ya que ésta proporcionaba una división natural de la tarea, los programadores se encargaban de subrutinas que posteriormente eran ensambladas para formar un solo programa; sin embargo eran muy difíciles de entender y mantener.

1.1.3 La Programación Estructurada

Con la Programación Estructurada un programa se descomponía en procesos individuales o Funciones, cada uno de los cuales se descomponía en subprocedimientos hasta llegar a la programación de un procedimiento sencillo.

La información se pasa entre procedimientos usando parámetros, los procedimientos pueden tener datos locales a los que no se puede tener acceso fuera del procedimiento.

Las variables globales desaparecen y se sustituyen por parámetros y variables locales.

Una debilidad de la Programación Estructurada es en esencia la división conceptual de datos y código, este defecto se agranda en proporción con el tamaño del mismo.

1.1.4 La Programación Orientada a Objetos

La Programación Orientada a Objetos tiene actualmente mucho auge por que nos presenta una nueva visión del mundo, ya que sus orígenes se basaron en la premisa de poder simular problemas del mundo real.

El iniciador de ésta tendencia fue el lenguaje SIMULA 67 (desarrollado en 1967), modificado del lenguaje ALGOL - 60 (programa que marcó el inició de la metodología de la Programación estructurada), el cual nació en Noruega debido al esfuerzo conjunto de varios investigadores de sistemas encabezados por Dahl y Nygaard, este proyecto consistía en realizar simulaciones sencillas de los problemas reales.

Los que se simulaban por medio de objetos, los cuales cooperaban o interactuaban entre sí; así nacieron los conceptos de objetos, clases de objetos y jerarquías de herencias entre clases.

Posteriormente aprovechando el diseño y las ideas sobresalientes de SIMULA 67, nació SMALLTALK; un lenguaje desarrollado por Xerox bajo la dirección de Andrew Kay, en un proyecto conocido como Dynabook, cuyo objetivo era crear una interface que hiciera mucho más sencillo el uso de la computadora personal, este retomó los conceptos de objeto, clase y herencia de SIMULA.

Cuando el lenguaje SMALLTALK se popularizó hizo su aparición el término de "OO" (Object Oriented) que dio origen a un nueva metodología o paradigma que es la Orientación a Objetos.

En la figura 1.1 observamos el avance que ha tenido la programación desde sus inicios hasta nuestros días, notamos que con el tiempo el nivel de abstracción (manera de visualizar los problemas) es mayor, ya que la tendencia es hacia una programación en la que se pueda representar de forma más simple el mundo real, que tenga una interface amigable y sus componentes puedan reutilizarse.

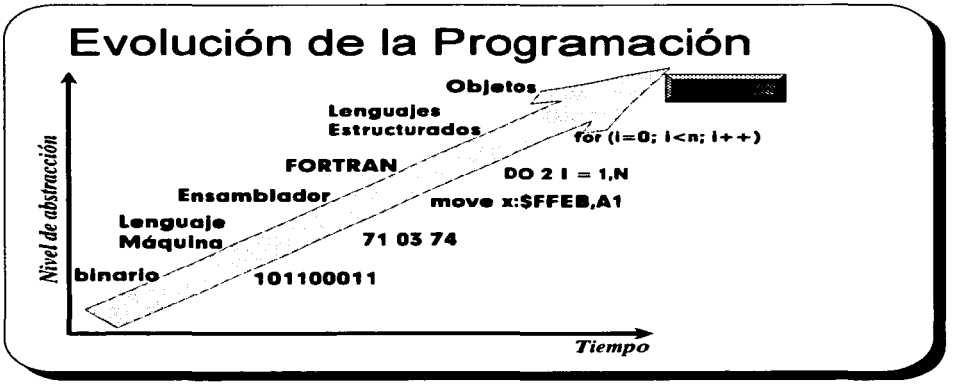


Figura 1.1 Evolución de la Programación

1.2 Concepto de Objeto y Programación Orientada a Objetos.

Para entender la metodología de Orientación a Objetos es necesario comenzar por definir dos conceptos; que es un Objeto y que es Programación Orientada a Objetos.

1.2.1 Concepto de Objeto

Todo lo que nos rodea puede ser considerado como un objeto físico desde un coche, una taza, una mesa, etc. Cada uno de estos objetos tienen características propias como su color, su forma y tamaño; con la introducción de la Orientación a Objetos, los programadores tuvieron que pensar en forma abstracta, es decir tomar en cuenta todas las características que se necesitan del objeto a simular.

Un objeto desde el punto de Grady Booch puede ser:
" Una cosa tangible o visible, algo que pueda ser aprendido intelectualmente, que tenga un estado, comportamiento e identidad propia.[1] "

Un objeto desde el punto de vista de Edward Yourdon es:
" La abstracción de algún problema que nos rodea, que refleja la capacidad del sistema para mantener la información e interactuar con ella, utiliza el encapsulamiento de atributos de los valores y exclusividad de servicios.[2] "

Podemos decir que un objeto es una abstracción del mundo real, caracterizada por contener datos y procedimientos que manipulan los mismos; puede ser concebido como un miniprograma que tiene comportamiento e identidad propia; los objetos contienen datos y funciones que operan sobre los mismos.

---

1 Grady Booch, *Object-Oriented Requirements Analysis and design*, p.29

2 Edward Yourdon, *Object- Oriented Analysis*, 2da. ed. Eglewood Cliffs NJ: Yourdon Press/Pretice Hall , 1990; pag 53).

Podemos ilustrar a los objetos mediante la Figura 1.2, en la cual observamos un reloj-objeto que nos permite observar solamente lo que necesitamos saber de él sin tomar en cuenta lo que lo hace trabajar.

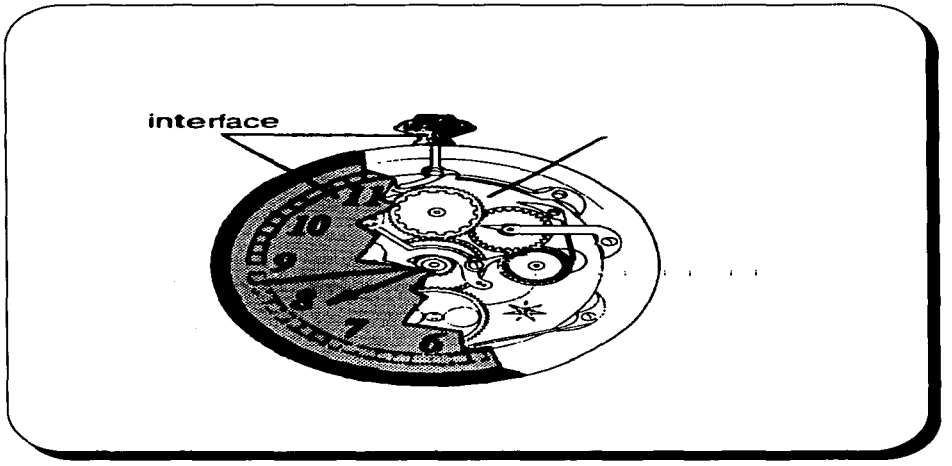


Figura 1.2 Diagrama de un Objeto

A los elementos de un objeto se les conoce como datos y las funciones que operan sobre ellos se denominan métodos.

Los objetos se pueden comunicar entre sí mediante el paso o envío de mensajes (acciones que debe ejecutar el objeto).

1.2.2 Concepto de Programación Orientada a Objetos.

La Programación Orientada a Objetos, es un importante conjunto de técnicas, que promueven la división de un problema en partes mas simples, utilizando como elemento básico a los objetos.

Para que un lenguaje de programación sea considerado como Orientado a Objetos se tiene que apoyar básicamente en: el encapsulamiento, la herencia, el polimorfismo y las ligas dinámicas; y tiene ciertas características como:

1. Que induzca a los programadores a pensar en abstracto, esto es encontrar la forma de modelar los problemas que se van a solucionar.

2. Que el foco de atención de la Programación Orientada a Objetos sean los datos quienes encapsulen la funcionalidad.

3. Que los objetos sean identidades independientes, permitiendo que sea posible la reutilización de componentes (" *component culture* ") y en caso de querer dar extensión no se tengan que realizar cambios en todo el programa sino simplemente en la parte que maneja los datos que se desean modificar.

4. Que sea mucho más sencillo dar mantenimiento a programas que estén construidos a partir de colecciones de objetos.

5. Que la combinación de objetos dé lugar a la creación de otros objetos más completos.

6. Que los objetos sean generados dinámicamente durante la ejecución del programa.

7. Que los objetos oculten la información.

8. Que el tiempo de ejecución sea reducido.

9. Que el control esté distribuido

10. Que los procedimientos se desarrollen a través de metas bien definidas.

11. Que los procedimientos conozcan lo mínimo unos de otros.

12. Que los mensajes sean enviados entre los diversos objetos.

13. Que cada objeto sea lo suficientemente capaz de encontrar sus propios errores.

1.3 Características de la Programación Orientada a Objetos

La Programación Orientada a Objetos, como modeladora del mundo se rige bajo ciertas reglas, las cuales permiten su correcto funcionamiento, éstas características la hacen diferente a las demás metodologías.

Es importante pensar que la Programación Orientada a Objetos tiene como objetivo disminuir el tiempo de mantenimiento de un sistema a futuro, y su lógica se basa en las siguientes características:

1.3.1 Clases

La clase es una colección de objetos que tienen una estructura de estado y un comportamiento común. La agrupación de los objetos dentro de las clases es una manera de ordenar y poder organizar el dominio de los objetos que conforman un sistema. Todos los objetos que pertenecen a una determinada clase también son denominados instancias de clase y sólo contienen los valores particulares para las variables, pero compartiendo el código de los métodos (operaciones empaquetadas en nuestra clase, los cuales son utilizados como procedimientos para modificar el estado de las variables y como funciones para regresar cierta información acerca de él) .

Las clases se utilizan debido a que difícilmente se realizan sistemas que utilizan un sólo objeto para su implementación, sino que se necesitan varios. Este concepto aparece con el sistema SIMULA 67 siendo manejado como un prototipo que define en él los métodos y variables que serán posteriormente incluidas en un objeto en particular.

Todas las descripciones que se realicen tanto de los métodos como de las variables que lo han de controlar se describen sólo una vez, en la definición de la clase, tomando en cuenta que las clases permiten además el poder ocultar información acerca de la implementación, sobre como son manipulados y

administrados los datos de las variables de dicha clase; contiene además las especificaciones de como los objetos de esa clase deben crearse.

Desde otra perspectiva una clase es simplemente la declaración de las propiedades tanto de la estructura de datos como de sus operaciones, las cuales tendrán los objetos pertenecientes a dicha clase, que se generan de manera dinámica. Podemos ejemplificar a las clases con la Figura 1.3 en la cual se puede observar la clase vehículos, todos los vehículos sirven para transportarnos.

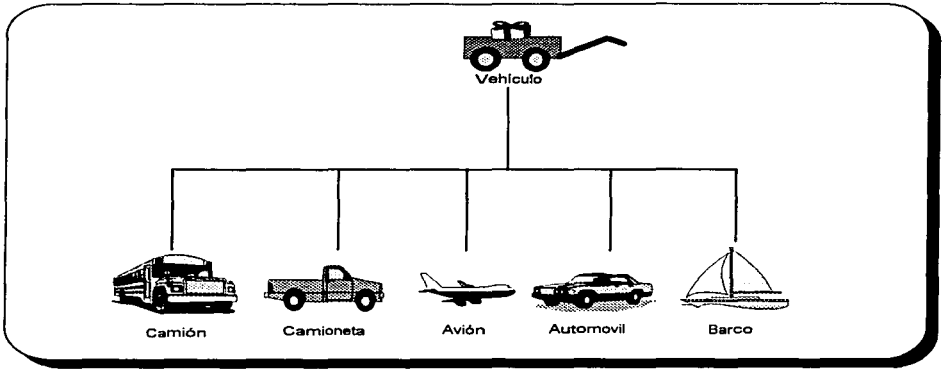


Figura 1.3 Clase Vehículos

1.3.2 Modularidad.

Un módulo es una agrupación de elementos relacionados lógicamente, que consiste en poder ser fácilmente combinados sus elementos con otros módulos, bajo ciertas reglas para producir un sistema complejo. La Programación Orientada

a Objetos crea módulos, los cuales tienen alto grado de cohesión y están ligeramente acoplados entre sí.

Es importante mencionar que cada pieza puede trabajar de manera independiente y ser compilados por separado, pero a la vez también permite ser integrado con otras piezas cuando los programas lo necesiten. Cada módulo es una unidad definida por el sistema que contiene el código fuente. Ejemplificando a la modularidad podemos observar a la Figura 1.4 en la que tenemos las partes que integran a una computadora, los cuales trabajan cada elemento por separado pero al ser unidos siguiendo las normas establecidas forman una computadora.

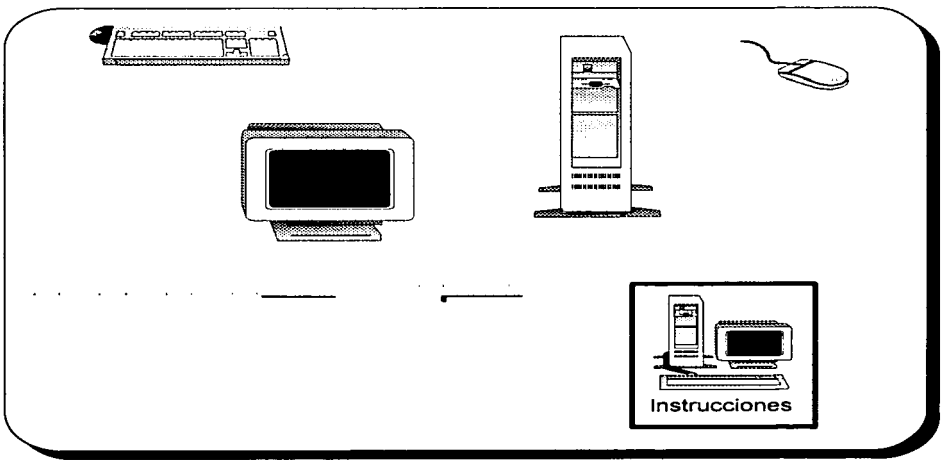


Figura 1.4 Modularidad

1.3.3 Reutilización.

La importancia de la Programación Orientada Objetos se basa en esta característica, que consiste en crear objetos y posteriormente sistemas, los cuales sean posible reutilizarlos posteriormente. En el desarrollo de sistemas es importante saber utilizar los objetos y planear su futuro mantenimiento, ya que se construyen en base a componentes previamente diseñados e implementados. Dichos componentes son seleccionados de una biblioteca de objetos, de acuerdo a los requerimientos que se tengan en concreto.

El objetivo de ello es a largo plazo, debido a que se desea amortizar el costo del desarrollo de cualquier sistema, pero para realizarlo es necesario cambiar las metodologías de análisis y de diseño, el cual es el fin primordial de los Lenguajes Orientados a Objetos. La base para la reutilización consiste en formar una completa biblioteca de componentes u objetos reutilizables, la cual debe de proveer al programador de :

1. Mecanismos que permitan seleccionar y comparar los componentes a utilizar.

2. Componentes de un fácil nivel de abstracción, que permitan saber de una manera sencilla que hacen.

3. Mecanismos que permitan modificar los componentes de acuerdo a las necesidades siguiendo parámetros y restricciones de modificación.

4. Mecanismos que permitan integrar los componentes seleccionados para poder formar sistemas complejos.

Esta característica es la que le augurá un gran futuro a la Programación Orientada a Objetos, debido a que la gran disparidad que existe en cuanto al ritmo de avances del hardware con respecto al software forza a tener herramientas que permitan la reutilización de los sistemas ya creados.

Desde 1968 en la conferencia de la OTAN, sobre la crisis de producción de software, se invita a los desarrolladores a diseñar una forma de poder tener control de los sistemas y con ello lograr que se obtenga el máximo aprovechamiento del hardware actual. Podemos ejemplificar a la reutilización con la Figura 1.5 en la que se observa como un monitor puede ser reutilizado en diferentes equipos de cómputo.

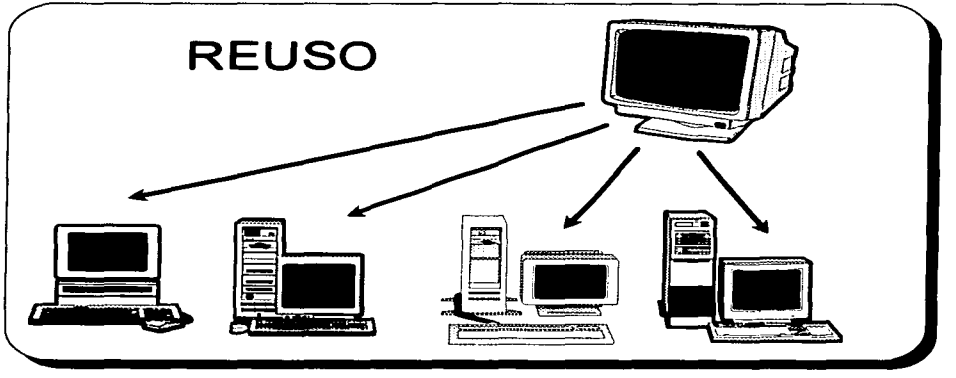


Figura 1.5 Reutilización de componentes

1.3.4 Mecanismos de abstracción de datos

La Programación Orientada a Objetos basa su diseño en la construcción de sistemas de datos abstractos en colecciones estructuradas.

La abstracción consiste en ocultar todos los detalles que resulten irrelevantes para la utilización y manipulación de los datos dándole importancia solamente a los que nos resulten útiles para el fin que deseamos realizar. Cada abstracción se asocia de manera directa con una serie de características tanto en su estructura como en su comportamiento, las cuales son importantes para la representación del modelo.

En la Programación Orientada a Objetos la abstracción de datos se realiza en las ***clases.*** Las interfaces de las clases tienen un trabajo doble, ya que por un lado especifican que operaciones están disponibles para nuestro objeto y por el otro controla el acceso a las operaciones que realizará el mismo.

Los datos abstractos tienen dos propiedades básicas:

1. Unidad lógica de datos y operaciones.

2. Ocultamiento de representación.

La primera propiedad nos da a entender que tanto los datos almacenados como las operaciones que permiten su utilización se encuentra en una sola unidad lógica.

La segunda propiedad tiene dos finalidades básicas, la primera consiste en pretender generalizar los datos que posteriormente sean declarados por el programador, ocultando los detalles que considere inadecuados para ser observados por el usuario; y la segunda como una protección de dichos datos, ya que la única manera de poderlos utilizar es solamente ocupando las operaciones definidas para ello, es decir, con esta característica un usuario que utilice un tipo de dato abstracto no puede modificar su representación interna.

1.3.4.1 Encapsulación

Es un mecanismo que nos permite ocultar los detalles de la representación interna de un componente, presentando al usuario una interface la cual cumple con un doble trabajo, ya que especifica por un lado los servicios que son disponibles al usuario, es decir, los que ofrece un determinado componente así como los requerimientos para su correcta ejecución, y por otro lado oculta todo lo que no se encuentre de una manera explícita en el programa. Los objetos encapsulados permiten simular objetos del mundo real, debido a que tienen como características primordiales un estado y un comportamiento propio que puede ser modificado por agentes externos a él.

La encapsulación nos permite colocar los datos y las funciones de manera conjunta; sirviendo además como una barrera de protección entre la interface y la implementación, es decir, la encapsulación protege a la implementación de ser modificada por el usuario, a su vez al estar empaquetados los datos es posible manejarlos de una manera más adecuada, ya que todas las acciones tienen lugar a través de la interface, la cual la podemos definir como lo que puede utilizar el objeto y permite al usuario observar sólo lo que necesita conocer para utilizarlo.

La encapsulación se ilustra en la figura 1.6 en la cual observamos que los datos privados son importantes para el buen funcionamiento del objeto, pero no siempre es necesario que el usuario observe esos datos.

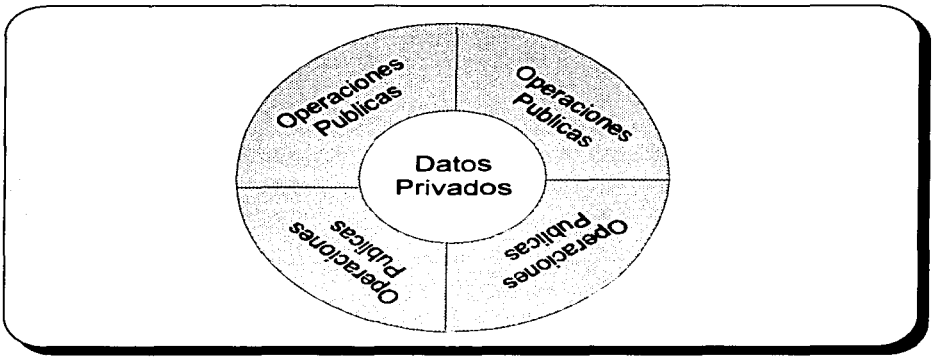


Figura 1.6 La encapsulación

1.3.4.2 Polimorfismo

El polimorfismo es la habilidad de diferenciar los objetos por su respuesta al recibir una misma llamada, se genera al admitir que una expresión o valor tome más de un tipo. Esto permite definir funciones que son aplicables a objetos de diferentes tipos de datos. Resulta de la característica de que cada clase tiene un nombre propio, al cual es asignado para evitar conflictos con cualquier otro nombre asignado fuera de dicha clase. Expresado de otra forma, la misma operación puede tener comportamiento distinto en diferentes objetos.

En la figura 1.7 podemos observarlo ya que el faro emite una señal para todas las embarcaciones, y como respuesta obtendrá una diferente dependiendo de la dirección que tome cada embarcación.

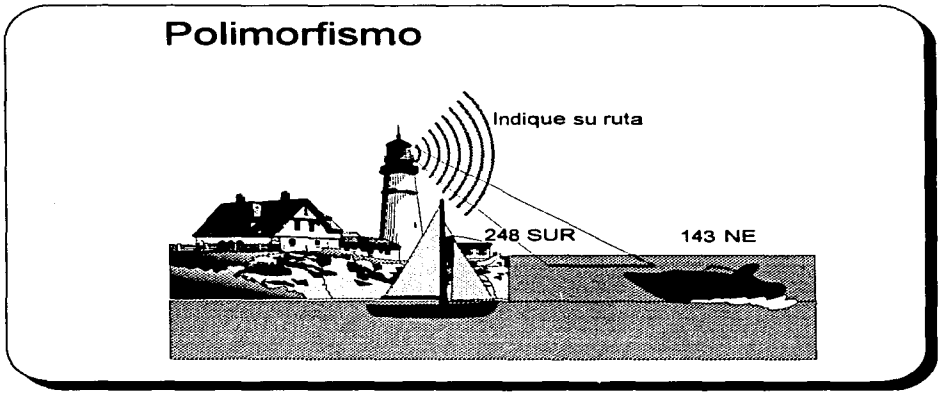


Figura 1.7 Polimorfismo

1.3.5 Herencia

La herencia simplifica la tarea de crear una nueva clase que sea similar a una clase existente, la cual permite a los desarrolladores de sistemas expresar únicamente las diferencias entre la clase nueva y la existente, esto nos permite definir a la herencia como la manera más poderosa de formalizar la extensión y reutilización de módulos de códigos existentes, siendo de esta forma que la clase que hereda utiliza solamente lo que necesita y lo demás no lo toma en cuenta.

La herencia es una relación entre clases que permite declarar subclases como extensión o especialización de otra clase la cual se le conoce como superclase.

Los Lenguajes Orientados a Objetos presentan dos variantes básicas, herencia sencilla y herencia múltiple, siendo posible en ello heredar de más de una clase.

La superclase es entonces la clase que hereda, mientras que a la clase heredera será la subclase. Esta puede añadir o modificar operaciones a su comportamiento, formando así una especialización de la superclase. Por lo tanto, una clase puede ser subclase de una o más superclases, siendo esta última ejemplo de herencia múltiple.

La herencia es ejemplificada gracias a la Figura 1.8 en la cual observamos que las subclases (o los hijos) heredan lo que necesitan de la superclase (o padre) siendo herencia simple debido a que sólo se basan de una superclase o padre; mientras que en la figura 1.9 observamos la herencia múltiple en la que los hijos o subclases heredan lo que necesiten utilizar de las superclases o padres.

1.3.6 Dinamismo.

Las entidades que forman un programa pueden referirse a objetos que están almacenados en más de una clase y las operaciones deben permitir diferentes realizaciones en clases distintas.

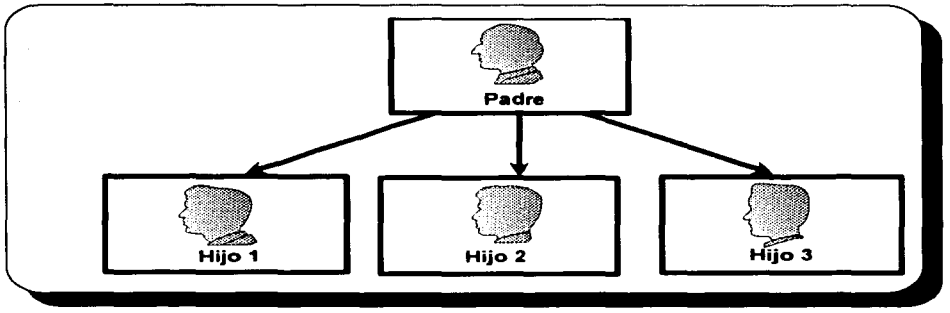


Figura 1.8 Herencia simple

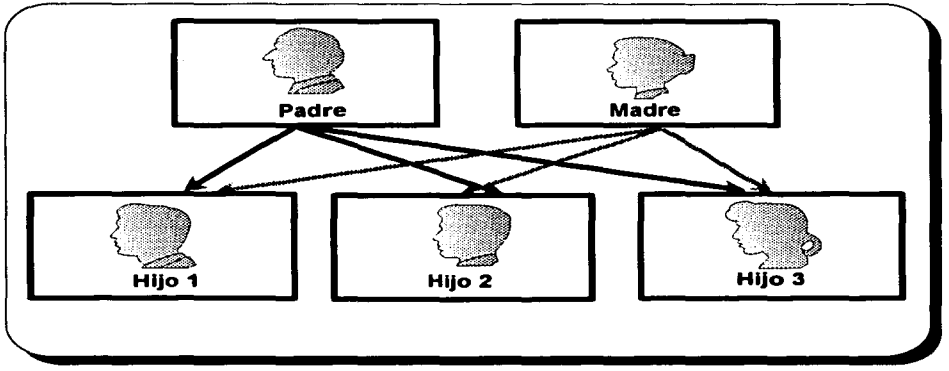


Figura 1.9 Herencia Múltiple

En el diseño Orientado a Objetos existen tres importantes tipos de dinamismo:

1. Escritura dinámica . Espera hasta que el sistema determine la clase del objeto.

2. Ligas dinámicas. Determina el tiempo de ejecución a la llamada a un método.

3. Búsqueda dinámica. Adiciona nuevos componentes a la ejecución de los programas.

En los tres tipos anteriores no podemos tener un control directo, ya que las llamadas las hace el compilador, en el primero determina el tiempo que necesita para saber de cual clase es el objeto y de esa forma reservarle el tiempo preciso.

Con el dinamismo los programadores pueden crear prototipos de una manera más rápida y no tienen que preveer el futuro de cada tipo de objeto. los programas pueden ser extensibles con los módulos de código dinámicos, al igual que permite la flexibilidad del código en un futuro mantenimiento del sistema.

1.4 Concepto de Sistema Operativo y Sistema Operativo Orientado a Objetos.

Antes de comenzar con la descripción de OpenStep es necesario definir dos conceptos; que es un Sistema Operativo y que es un Sistema Operativo Orientado a Objetos.

1.4.1 Concepto de Sistema Operativo

Un Sistema Operativo es un conjunto de programas y rutinas que permiten a los usuarios tener un control de los recursos que maneja una computadora, interpretando instrucciones e incrementando la utilidad del hardware que se utilice, proporcionando la base sobre la cual pueden escribirse los programas de aplicación. Los sistemas operativos controlan todo el flujo de información que se tiene en la computadora, así como el tránsito de la misma por diversas funciones y programas que lo requieran, expresado de otra manera el Sistema Operativo es el conjunto de programas que controlan el hardware y el cúmulo de información que entra, sale y se modifique en la computadora.

1.4.2 Concepto de Sistema Operativo Orientado a Objetos

Podemos definirlo como un pequeño núcleo que se comunica con el hardware de la computadora y el resto del sistema operativo consiste en objetos o módulos reemplazables; teniendo cierta jerarquía el núcleo, ya que no existe

distinción entre el sistema operativo, las rutinas de utilidades y todas las aplicaciones.

El núcleo de un Sistema Operativo Orientado a Objetos tiene la capacidad de ser multitarea, con la finalidad de que muchos objetos puedan ejecutarse de manera simultánea, provee soporte para la recolección automática de objetos muertos evitando con ello la pérdida de memoria.

Los Sistemas Operativos Orientados a Objetos proveen soporte hereditario, lo que ahorra tiempo y optimiza el espacio de almacenamiento. El Sistema Operativo Orientado a Objetos que se conoce hasta el momento es OpenStep, el cual esta construido basado en el micronúcleo (microkernel) MACH.

Podemos decir que en un Sistema Operativo Orientado a Objetos, todo lo que este más allá del núcleo es un objeto desplazable y los archivos pasan a un segundo término con respecto a los objetos persistentes que son transferidos en forma transparente (directa) entre la memoria y el almacenamiento masivo.

CAPITULO II

" ANTECEDENTES DE OPENSTEP Y SUS PERSPECTIVAS A FUTURO "

2.1 Antecedentes de OpenStep

2.1.1 Mach.

Todos los Sistemas Operativos ofrecen características comunes para el control de los datos que manejan, esto es debido a que cada día la cantidad de dicha información va en aumento y con ello se hace necesario tener un control más óptimo de los recursos con los que contamos.

Los sistemas operativos permiten numerosas funciones entre las cuales podemos mencionar las siguientes:

1. Cargar programas en memoria y ejecutarlos.

2. Mantener el control del directorio de archivos y de la cantidad de datos almacenados.

3. Ofrecer un conjunto de rutinas que el programa pueda utilizar para localizar, leer o escribir en un determinado archivo.

4. Administrar el acceso de los programas a los recursos del sistema, como son la memoria, entrada y salida de datos por los puertos y la cantidad de información almacenada.

5. Controlar las prioridades que tenga cada uno de los programas sobre los recursos del sistema, dándole el tiempo necesario a cada operación.

6. Administrar la memoria disponible en el sistema con la finalidad de poder ejecutar los programas que así la soliciten.

7. Permitir el intercambio de información entre programas y dispositivos.

En la actualidad todos los sistemas operativos realizan las anteriores tareas, permitiendo con ello hacer que el manejo de las computadoras sea más fácil y más poderoso. El kernel del sistema operativo de NextStep se llama **MACH** y está diseñado para ser compatible con **UNIX**, pero desde otra perspectiva, es lo que lo hace diferente a los sistemas conocidos.

Cuando se pensó crear NextStep fue necesario tomar en cuenta que la única forma de mejorar los ambientes operativos que había en el mercado era crear su propio sistema operativo o adaptar uno ya existente; después de estudiar las ventajas de cada una de esas opciones llegaron a la conclusión de que no era factible crear su propio sistema, ya que cuando éste estuviera listo la tecnología ya habría cambiado, situación por la cual buscaron un sistema que cumpliera con sus expectativas, pero los existentes en el mercado estaban bastante limitados, ya que estaban diseñados para alguna máquina en especial, un determinado procesador o una determinada arquitectura de hardware.

En el año que empezó el desarrollo del Cubo (Máquina NeXT) se pensó que lo más importante era utilizar el Sistema Operativo **UNIX**, esto debido a que tenía gran aceptación entre las empresas grandes y las universidades, debido a que su creador AT&T incluía en el sistema el código fuente con la finalidad de que pudiera ser mejorado posteriormente.

El kernel de **MACH** es una versión de **UNIX** mejorada, resultado del estudio de la Universidad Carnegie-Mellon de poder utilizar un núcleo muy pequeño y

mejorado para realizar las tareas de manera rápida y concreta; éste maneja ambientes complejos, así como memoria virtual y procesadores múltiples. Los objetivos del desarrollo de MACH, que hicieron que fuera seleccionado por NeXT como el kernel de su Sistema Operativo Orientado a Objetos fueron los siguientes:

1. Proporcionar una base para la creación de otros sistemas operativos.

2. Soporte de un espacio de direcciones de gran tamaño.

3. Permitir un acceso transparente a los recursos de la Red

4. Explotar el paralelismo tanto en aplicaciones como en el sistema.

5. Permitir que sea transportable a otras máquinas.

Su principal concepto de elaboración fue el poder estructurar los sistemas operativos de manera modular, como una colección de procesos que se comuniquen entre sí mediante la transferencia de mensajes.

Al desarrollar MACH se utilizó la idea del micronúcleo (microkernel), éste consistía en poder emular UNIX y otros Sistemas Operativos. La emulación se lleva a cabo mediante una capa del software que se ejecuta fuera del núcleo, en el espacio del usuario; es importante el mencionar que es posible ejecutar varios emuladores al mismo tiempo, por lo que se pueden ejecutar programas 4.3 BSD, UNIX sistema V y MS-DOS al mismo tiempo.

El núcleo de MACH, al igual que otros micronúcleos, proporciona la administración de la memoria, la comunicación y los servicios de Entrada/Salida.

Los archivos, directorios y funciones generales del Sistema Operativo se encuentran en el espacio del usuario.

NeXT se dio cuenta de que el sistema operativo MACH, se acercaba mucho a su idea de utilizar los objetos como elementos del sistema, situación por la cual decidieron contratar a algunos de los desarrolladores de MACH, entre los que destaca el Dr. Avadis Tevanian, el cual ayudo a mejorar MACH con la finalidad de poder ser utilizado de manera comercial.

El núcleo de MACH esta diseñado para manejar principalmente 8 ideas básicas:

1. ***Manejo de Memoria Virtual.*** El sistema distribuye la cantidad de recursos necesarios para cada programa. Cuando un programa necesita más memoria que la que tiene el sistema, éste la simula para poder ejecutarlo; las ventajas de ello son:

   a) Ejecuta varios programas de manera simultánea.
   b) Ejecuta de manera rápida los programas que son grandes distribuyendo espacios en la memoria.

2. ***Tiempo de Procesador.*** El sistema administra el tiempo que le otorgará a cada programa para la ejecución del mismo. MACH es un sistema operativo multitarea, esto es, que se pueden ejecutar al mismo tiempo varios programas, lo cual es determinado por el CPU, dándole el tiempo necesario a cada aplicación para dar la ilusión de que trabajan al mismo tiempo, pero respetando prioridades; MACH esta diseñado para soportar múltiples procesadores, esto es, tener varios CPUs trabajando al mismo

tiempo, esto permitirá obtener un mayor rendimiento en los procesos a ejecutar.

3. ***Comunicación entre procesos.*** El sistema separa la información que va a enviar de un proceso a otro. El núcleo de MACH maneja mensajes que son transmitidos de un hilo a otro, de un proceso a otro, y de un proceso al núcleo. Estas definiciones y los controles de los puertos para cada hilo y proceso; puede mandar mensajes y recibirlos.

4. ***Procesos.*** Este es básicamente donde se lleva acabo la ejecución. Tiene un espacio de direcciones, dentro del mismo se encuentran el texto y los datos del programa y, por lo general, una o más pilas. El proceso podemos decir, que es la unidad básica para la asignación de recursos.

5. ***Hilos***. Un hilo lo podemos definir como una entidad ejecutable. Tiene un contador del programa y un conjunto de registros asociados a él. Cada hilo es parte de exactamente un sólo proceso. Un proceso con un hilo es similar a un proceso tradicional.

6. ***Objetos de la memoria.*** Es una estructura de datos (única de MACH) que se puede asociar con el espacio de direcciones de un proceso. Los objetos de memoria ocupan una o más páginas y forman la base del sistema de memoria virtual de MACH.

7. ***Puerto.*** Las comunicaciones que se realizan entre procesos por medio de mensajes, en MACH las definimos como puertos en donde se reciben los datos, se almacenan dentro del núcleo y pueden formarse en una cola constituída por una lista ordenada de mensajes. Los puertos podemos

definirlos como un objeto separado de los hilos y los procesos, el cual da por entendido los privilegios de cada uno y permite mandar o recibir mensajes.

8. ***Mensajes.*** Los mensajes permiten mandar información de un proceso a otro y con ello permitir la interrelación de los procesos. Esencialmente todas las comunicaciones de tareas internas usan los puertos para mandar y recibir mensajes; MACH tiene que tener una sincronización entre el envió y recepción de los mismos.

La información es organizada en el sistema NeXT por medio de archivos.

Un archivo es una colección de información asociada y guardada en un sistema de almacenamiento (disco duro, CD-ROM, etc.), dichos archivos se pueden clasificar en tres tipos: a) *Archivos de Texto,* b) *Archivos de Datos,* c) *Archivos Objeto,* contiene instrucciones del CPU en formato binario (formato máquina). Los archivos son organizados dentro de Directorios, éstos son una colección de archivos, usualmente relacionados entre sí.

MACH usa una interface standard para el usuario, la cual es llamada *Shell,* éste es un comando en línea, que trabaja como un miniprograma. El sistema NeXT maneja una interface gráfica para el usuario (GUI), la cual permite manipular imágenes por medio del mouse, sin necesitar el shell; por lo tanto NeXT nos ofrece la oportunidad de utilizar tanto comandos en línea a través de la aplicación terminal, así como utilizar el ambiente gráfico.

2.1.2 NeXTStep

NeXTStep es el pionero en la creación de ambientes Orientados a Objetos, debido a que en la actualidad se utiliza el concepto de programa orientado a objetos para indicar que dichos programas pueden utilizar herencias de clases, polimorfismo, encapsular datos y fomentar la reusabilidad de código; pero no son sistemas creados en ambientes orientados a objetos, es decir, su origen no fue en un ambiente que trabajará cada módulo realizando una tarea específica, ni de manera independiente.

La idea de crear NeXTStep nació de la necesidad de desarrollar ambientes diseñados para el desarrollo de software cliente-servidor, el cual permitiera reutilizar el código, ser amigable con el usuario y al mismo tiempo sencillo y rápido para hacer las modificaciones a futuro.

NeXTStep proporciona módulos funcionales fácilmente particularizables y enlazables; lo que permite tener un control total de todos los componentes que se tienen a la mano.

NeXTStep es un ambiente agradable visual, y con elementos gráficos incluídos en el sistema lo que proporciona una facilidad de uso de estos elementos al usuario final; podemos mencionar que es un ambiente parecido a Macintosh dentro del poder que ofrece UNIX.

La interface que presenta NeXTStep esta diseñada de una manera en la cual los desarrolladores que utilicen el ambiente no tienen que modificar de una manera sustancial el código cuando elaboren una modificación a un programa, ahorrando con ello tiempo en la elaboración de las modificaciones, y en el

mantenimiento, siendo además intuitivo, debido a que utiliza iconos y se observa como se van uniendo los elementos del programa; si utilizamos los objetos que ya se encuentran en las bibliotecas podremos ahorrar mucho tiempo en la elaboración de programas, y con ello realizar sistemas cliente-servidor potentes y de manera sencilla.

Son muchas empresas corporativas, financieras, educacionales y de desarrollo las que utilizan NeXTStep para la elaboración de sus programas debido a su facilidad de uso y la ventaja que representa el poder manejar los objetos, y ahorrar tiempo en el mantenimiento de sus programas, cuando se necesita realizar modificaciones.

El tiempo de desarrollo con NeXTStep es mucho menor y más sencillo, por lo que empresas como Chrysler la utilizan para la elaboración de análisis de préstamos y contratos de arrendamiento, Disney para la creación de programas de control de turistas que entren al parque de diversiones, así como análisis de los centros de atracción más importantes que causen mayor interés al turista, para realizar posteriores estudios de factibilidad de nuevas atracciones, Motorola para controlar las ventas que se tienen a nivel mundial; empresas líderes en el mercado de las microcomputadoras, como Lotus Development, han realizado programas bajo NeXTStep, debido a la facilidad y rapidez de elaboración de sus programas, un ejemplo sería Improv, una hoja de cálculo desarrollada a últimas fechas que utiliza la Tecnología Orientada a Objetos, y que la hace más rápida dentro de su género, que próximamente estará incluida para ser utilizada dentro del ambiente UNIX y Windows.

NeXTStep utiliza el kernel MACH, que tiene como características principales, el manejo de memoria, el control de los procesos y la intercomunicación entre procesos, el sistema visual Display PostScript, el cual permite observar una imagen idéntica en la pantalla, al enviar un fax o al imprimir, cuenta con multiprocesos y conexión directa a red Novell e Internet, estas características aunadas con un sistema completamente diseñado a objetos, desde las bases de datos de administración del sistema, los recursos gráficos y hasta el nivel del núcleo, hacen a éste sistema diferente a todos los demás.

NeXTStep fue creado por Steve Jobs y una serie de desarrolladores, con la idea de crear un Sistema Operativo diferente, que le ofreciera a los desarrolladores y al usuario final una interface agradable, fácil de utilizar y de rápido aprendizaje, con la característica fundamental de ser orientada a objetos, lo que la haría más rápida en sus procesos.

La demanda de poder realizar programas a la medida cada día es mayor, por lo que NeXTStep ayuda a realizar dichos sistemas de una manera rápida y sencilla; existen versiones para Intel, SUN, DEC, HP-UX, Motorola, lo que permite que un programa realizado en cualquier plataforma, si es compilado con la finalidad de correr en alguna otra lo logre sin ningún problema, podemos agregar que NeXTStep junta herramientas de desarrollo orientadas a objetos completamente empaquetadas , así como objetos reusables, lo que ayuda al desarrollador a escribir sus aplicaciones; literalmente una aplicación desarrollada en NeXTStep es más rápida de desarrollar que en cualquier otro sistema.

Es importante mencionar que NeXTStep es uno de los mejores Sistemas Operativos debido a su facilidad de conexión a bases de datos como son Informix, Sybase y Oracle; su interface gráfica del usuario (GUI) hace que sea más fácil de

entender sus iconos, es multitarea, hay una gran interoperabilidad entre aplicaciones, la distribución del área de trabajo es muy intuitiva y fácil de entender, hay aplicaciones que nos ayudan a utilizar todos los recursos del sistema, el Editor, la ayuda en línea de la biblioteca digital, el correo multimedia, la interface que es la misma en todas las aplicaciones, el instalador de aplicaciones es estándar en todos los programas y es fácil de entender su lógica de desarrollo.

El ambiente de desarrollo tiene como características que desde su origen utiliza la Programación Orientada a Objetos, las herramientas de desarrollo que provee el sistema son poderosas y fáciles de utilizar, el Interface Builder (creador de interface para el usuario), App Kit (conjunto de herramientas para desarrollar aplicaciones), DBKit (herramientas de conexión a bases de datos), su lenguaje de desarrollo Objetive-C basado en C ANSI, pero obteniendo todo el poder de los objetos, la integración del ambiente del usuario con el de operación del sistema, basado para estar conectado en red, es decir, está pensado para tener varios desarrolladores trabajando al mismo tiempo, ayuda para la elaboración de aplicaciones en línea, muy flexible, no existe límite para la elaboración de aplicaciones.

En la figura 2.1 podemos observar el área de trabajo de NeXTStep, la cual nos presenta un ambiente gráfico agradable con aplicaciones listas para utilizarse, pero que a la vez pueden ser piezas de una aplicación más grande.

NeXTStep permite la administración remota, la utilización de una red virtual, cliente/servidor de acceso remoto, manejo de conexiones a otras redes UNIX o conexiones a Internet, administración remota, permite trabajar con procesadores Intel, los cuales son más baratos que las estaciones de trabajo, manejo de varias plataformas, aplicaciones que pueden crecer con el tiempo, aplicaciones

con gran impacto visual que se ajustan a las necesidades del desarrollador, control absoluto contra virus, alto nivel de seguridad en las estaciones de trabajo, desarrollo de aplicaciones más rápido y fácil que las realizadas en Windows, integración de aplicaciones, utilizando SoftPc es posible ocupar aplicaciones para Windows.

2.2 Características y ventajas de OpenStep.

2.2.1 Historia de OpenStep.

OpenStep es un Sistema Operativo Orientado a Objetos, el cual es el resultado de la evolución de los sistemas operativos, que se desarrollan cada cinco años, tiempo que se calcula es necesario para modificar completamente la tecnología anterior, se puede ilustrar lo anterior con un análisis de Steve Jobs presidente fundador de las computadoras NeXT y pionero de la computación, el cual diseño junto con Steve Wozniac en 1976 la primera computadora personal, la Apple I fundando en ese año la compañía Apple; *" La tecnología en sistemas operativos cambia completamente aproximadamente cada cinco años, tenemos como ejemplo en 1979 CP/M, en 1984 Macintosh, en 1989 NeXTStep y en 1994 OpenStep, ese tiempo es el necesario para que un Sistema Operativo ascienda a ocupar el lugar que el sistema anterior dejo en su descenso llegando a sustituirlo por sus innovaciones e influencia sobre el mercado, necesitando siempre tener una base para su mantenimiento y futura compatibilidad* "[1],

[1] WEBSER, Bruce, Revista BYTE (Edición en Inglés) ; " Whither NeXTStep ? "; Número 11 Noviembre 1994, pag. 289

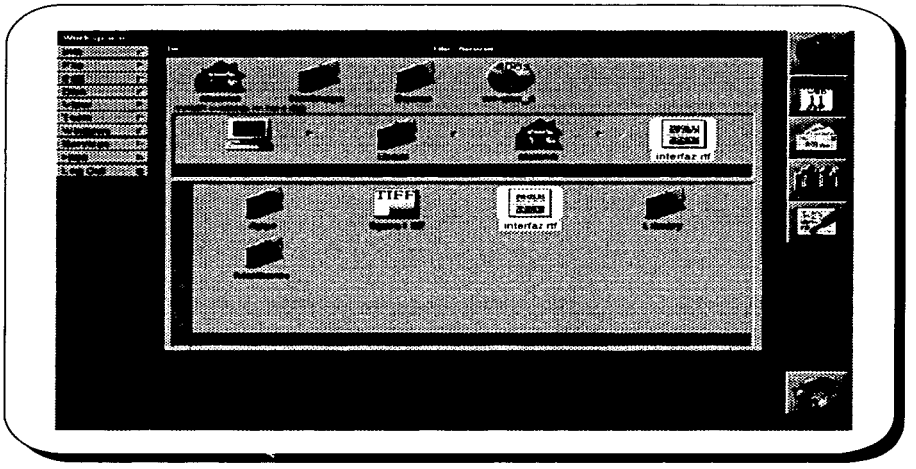


Figura 2.1 Ambiente de trabajo de NeXTStep

Podemos realizar una breve historia del desarrollo de OpenStep desde sus orígenes hasta nuestros días, y con ello entender la razón por la que este sistema haya ascendido tanto en tampoco tiempo. En la tabla 2.1 podemos apreciar el avance que fue necesario para llegar a OpenStep, donde observamos el avance de la tecnología, así como, cada día se acerca más a poder simular el mundo de forma más sencilla y concreta.

OpenStep esta llegando al nivel más alto de ambientes de programación, debido a que presenta muchas ventajas en su utilización, la tecnología avanza a lograr colocar sistemas de información precisos y sencillos en su manejo, los cuales puedan simular la realidad de una manera sencilla, ésta situación la logra

Año	Mes	Descripción
1985	Septiembre	Se funda la compañía NeXT por Steve Jobs
1988	Octubre	Es anunciado el " CUBO " NeXT (computadora UNIX multimedia) que contará con sistema Display PostScript y Software NeXTStep
1989	Septiembre	Se ponen a la venta el en mercado el CUBO y NeXTStep ver 1.0
1990	Septiembre	Es anunciado el sistema NeXTStation (Estación de trabajo NeXT)
1991	Abril	Se ponen a la venta el en mercado la NeXTStation y NeXTStep ver 2.0
1992	Enero	Es anunciado NeXTStep para plataforma Intel
	Septiembre	Se ponen a la venta el en mercado NeXTStep ver 3.0
1993	Febrero	Se deja de producir máquinas NeXT (hardware)
	Mayo	Se ponen a la venta el en mercado NeXTStep ver 3.1 para Intel. Se anuncia que NeXT y HP crearan el software Object Enterprise and PDO (Objetos distribuidos Portables) para HP-UX
	Noviembre	NeXT y SunSoft anuncia la creación de OpenStep, desarrollado a partir de NeXTStep para SPARC, PDO para Solaris y SunOS. NeXT pone a la venta PDO para HP - UX
1994	Marzo	NeXT y DEC (Computadoras Digital) anuncian PDO para DEC OSF/1
	Junio	Las especificaciones preliminares de OpenStep son anunciadas NeXT y DEC anuncian OpenStep para DEC OSF/1 en la plataforma Alpha AXP . Se anuncia OpenStep para Windows NT y Windows 95
	Agosto	Se ponen a la venta el en mercado NeXTStep ver 3.2 para HP-PA RISC NeXT pone a la venta PDO 2.o para HP, SunOS y Solaris
	Septiembre	Se pone a la venta OpenStep
	Octubre	NeXT pone a la venta Enterprise Objects Framework
	Noviembre	Se ponen a la venta NeXTStep ver 3.3 para SPARC
1995	Marzo	Se pone a la venta OpenStep para Solaris
	Mayo	Se anuncia OpenStep para Windows
	Julio	Se pone a la venta OpenStep para Intel, MACH, DEC, HP-UX, Windows NT
	Agosto	Se anuncia WebObjects (Programa para realizar Intranets basadas en la Tecnología Orientada a Objetos)
1996	Marzo	Se pone a la venta WebObjects
	Abril	Se anuncia la unión de desarrollo de Netscape y NeXT para realizar WebObjects 2.0
	Julio	Sale a la venta Solaris NEO
	Diciembre (estimado)	Sale a la venta OpenStep para Windows 95 y WebObjects 2.0

Tabla 2.1 Historia de OpenStep

OpenStep debido a que provee herramientas de desarrollo fáciles de utilizar y aprender, pero a la vez poderosas y utilizando cualquier aplicación lista como elementos de una futura aplicación más completa.

2.2.2 Características y ventajas de OpenStep.

Las compañías desarrolladoras o que necesitan elaborar programas a la medida, se han dado cuenta de la facilidad que representa utilizar OpenStep para sus programas, OpenStep es considerado un Ambiente de Aplicación Orientado a Objetos (OOAE).

Empresas dedicadas a la producción de Tecnología como son HP, SUN, DEC, se han dado cuenta de la ventaja competitiva que representa tener un Sistema Operativo Orientado a Objetos (OOOS) o un Ambiente de Aplicación Orientado a Objetos (OOAE) en sus máquinas, los cuales obtendrán todo el provecho de su hardware y con ello ofrecer al desarrollador un sistema abierto completo.

Debemos tomar en cuenta que OpenStep ofrece entre otras cosas:

1. Ser un ambiente para desarrollo de Aplicaciones, cuenta con una interface gráfica con DisplayPostScript, así como herramientas de desarrollo como son Interface Builder, Icon Builder, Project Builder.

2. Permite la comunicación a través de objetos distribuidos de una plataforma a otra, sin perder su control. (Enterprise Objects Frameworks).

3. Provee el sistema de Objetos Portables Distribuidos (PDO).

4. Interface directa con WebObjects para la elaboración de Intranets.

5. Ofrece además herramientas de control y administración de Red, el protocolo TCP/IP (Protocolo de administración de la transmisión / Protocolo Internet), conectividad a la red Netware, ejecuta todos los programas elaborados para UNIX, herramientas para el control de Bases de Datos Informix, Oracle y Sybase.

El resultado final de desarrollar módulos que sean fácilmente modificables, permiten crear objetos reusables en aplicaciones de negocios.

Existe un gran número de desarrolladores que programan en C, C++, por esa razón Objective C (lenguaje que utiliza directamente OpenStep) permite ejecutar programas realizados en C o C++ indistintamente, pero el código de los dos anteriores es más difícil de escribir que el de Objective C, por lo cual los desarrolladores de OpenStep prefieren este último.

Los Objetos Portable Distribuidos proveen una estructura Cliente / Servidor, heterogénea basada en objetos que extienden la dinámica de OpenStep y distribuyen objetos modelo en un ambiente para servidores UNIX. Con los Objetos Portables Distribuidos, los objetos son accesados en forma local o remota de la misma manera en que se obtiene acceso a las aplicaciones, y permite a los desarrolladores tomar ventaja de los recursos de la red. Para los desarrolladores se cuenta con un soporte para C++ dentro del compilador de Objective C, lo que permite migrar Objective C, C++ y código de C ANSI en una sola aplicación.

Los Objetos Portables Distribuidos proveen una interoperabilidad entre los ambientes de servidores y los clientes de OpenStep; se tiene destinado que esta interoperabilidad utilice los nuevos estándares de distribución informática.

La herramienta Enterprise Objects Frameworks (EOF, Manejo de objetos para infraestructuras empresariales), es utilizado para modificar y manejar bases de datos y se distingue por ser flexible y permitir a los desarrolladores de redes, distribuir componentes personalizados, ésta diseñado para reducir al mínimo el impacto en el cambio de bases de datos o formatos de datos, tiene un control de memoria y es fácil de modificar.

El Enterprise Objects Frameworks, se basa en la idea Modelar-Ver-Controlar, ella ayuda a separar el contenido de los datos de la estructura de presentación mediante una capa de control intermedia. Los componentes adjuntos conocen lo suficiente para comunicarse entre sí, es posible con ello unir dos bases de datos sin necesidad de escribir código, es decir, en vez de escribir código para analizar gramaticalmente una cadena en componentes o para leer información de entrada de un archivo, simplemente se utiliza una llamada a un método que explorara la palabra, todo ello sin utilizar más memoria que la que necesita para buscar ese objeto, debido a su sistema de autoliberación de memoria, podemos entonces decir que la ventaja real que nos ofrece esta herramienta es la confiabilidad en el manejo de los datos, que es extensible y también modular.

Como un beneficio más OpenStep ofrece ser compatible con el sistema que está realizando junto con HP, SUN y DEC, el estándar de UNIX, que es conocido como COSE (Ambiente de Software Abierto Común), lo que permitirá que el

ambiente de implementación y desarrollo orientado a objetos sea común bajo cualquier plataforma que utilice **UNIX** como sistema operativo base.

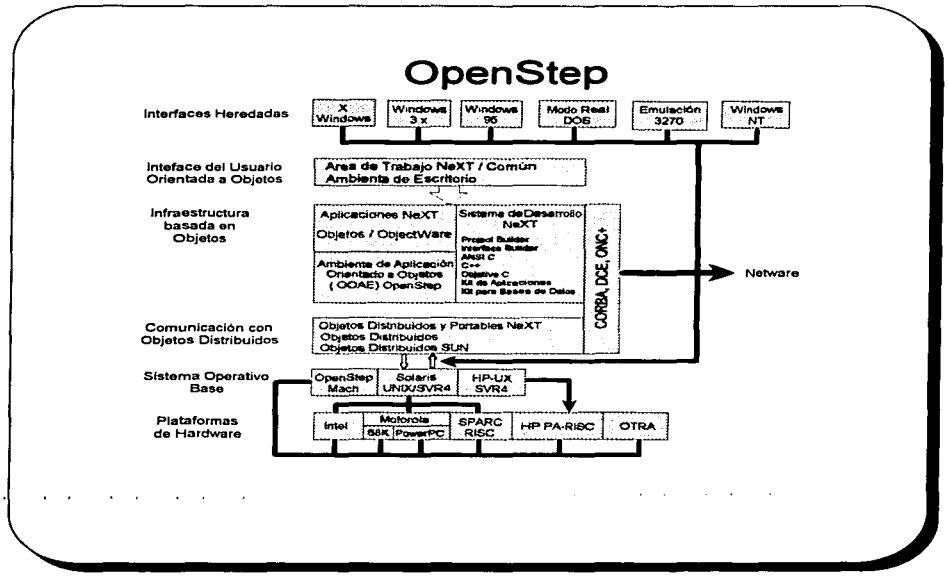


Figura 2.2 Estructura de OpenStep

En la figura 2.2 se puede observar la infraestructura que ofrece OpenStep, dando soporte a las plataformas más importantes en el mercado, así como la interrelación con los sistemas operativos más utilizados, que presentan soporte UNIX; trabajando con esta alternativa de desarrollo se pueden realizar programas para todo tipo de ambientes conocidos hasta la fecha, siendo transparentes sus

interfaces gracias a la tecnología de Objetos Distribuidos Portables, todo ello soportado por redes de computadoras como es Novell, teniendo una conexión directa a Internet, logrando ser un servidor Web.

OpenStep se ha comparado en ocasiones con Windows NT, desde el punto de vista en que cada uno de ellos ofrece soporte a ambientes abiertos, pero teniendo diferencias sustanciales en su desempeño.

La manera de analizar estas diferencias consisten en darnos cuenta de que OpenStep es un Sistema Operativo y al mismo tiempo un ambiente para el desarrollador, lo que permite obtener el máximo provecho de la orientación a objetos, mientras que Windows NT utiliza el paradigma de orientación a objetos para realizar sus funciones, éste último es considerado una copia de OpenStep, debido a la utilización de un Microkernel (micronúcleo), y poder utilizar múltiples procesadores; pero debemos entender desde un principio que Windows NT solamente es un sistema operativo orientado al manejo de redes y OpenStep al desarrollo de sistemas, aplicaciones cliente/servidor y sistemas de comunicaciones empresariales; lo que permite manejar el mercado de transacciones internacionales, mientras que Windows NT esta diseñado para sistemas menos robustos y sin tantos requerimientos, pero ahorra recursos debido a que esta diseñado para el mercado de las PC´s, mientras que OpenStep esta diseñado para el mercado empresarial, de negocios y de desarrollo de sistemas.

Las revistas BYTE y ComputerWorld[2] realizaron un análisis objetivo de los alcances de cada uno de los sistemas teniendo como resultado la tabla 2.2, en la

[2] SMITH, Ben, Revista BYTE (Edición en Inglés); " NeXTStep for Intel"; Volumen 18, Número 9, Agosto 1993, pags. 141-144

cual podemos observar las diferencia y similitudes entre Windows NT y OpenStep desde el punto de vista de sistema operativo.

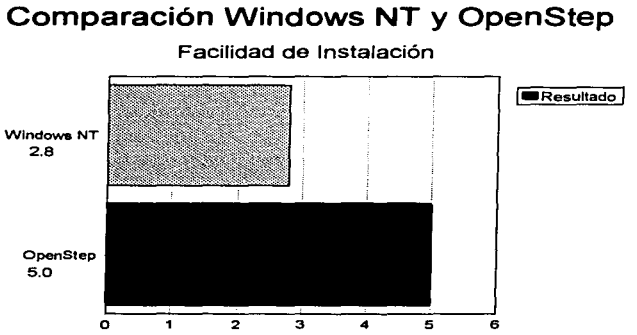
Windows NT	OpenStep
Multiusuario, multitarea	Multiusuario, multitarea
Arquitectura Micronúcleo	Núcleo del sistema operativo
Su ambiente no gráfico es MS-DOS	Su ambiente no gráfico es UNIX
El ambiente gráfico de Windows proporciona una enorme base de usuarios ya familiarizados con esta interface	El ambiente gráfico de OpenStep esta claramente integrado con el sistema operativo siendo más intuitivo
Sistemas de archivos MS-DOS, Windows e interpretación UNIX	Sistema de archivos UNIX, compartido con MS-DOS y Macintosh
Comunicación entre procesos locales y remotos	Comunicación entre procesos locales y remotos
Ejecuta aplicaciones Windows de 16 y 32 bits y aplicaciones MS-DOS	Ejecuta aplicaciones OpenStep, UNIX, MS-DOS, Windows 16 bits, Macintosh
El desarrollo de aplicaciones es difícil	El desarrollo de aplicaciones es sencillo
El sistema operativo emula un sistema orientado a objetos, existen lenguajes para desarrollo orientado a objetos, pero la relación entre ambos aún no esta muy desarrollada	La orientación a objetos inicia desde el sistema operativo hacia las herramientas de desarrollo, teniendo en ellas la relación directa y transparente entre todos los elementos del sistema.

Tabla 2.2 Comparativa entre Windows NT y OpenStep

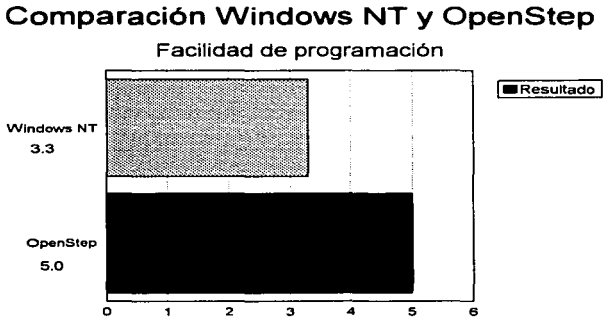
En las siguientes gráficas (2.1, 2.2, 2.3, 2.4, 2.5) podemos observar el análisis realizado por la Revista ComputerWorld [3], en la cual podemos observar las diferencias existentes entre Windows NT y OpenStep; esta clase de

[3] SULLIVAN, Michael, Revista ComputerWorld (Edición en Inglés); " The ComputerWorld's guide to Object-Oriented Programming "; Número 13, Junio 14, 1993, pag. 121

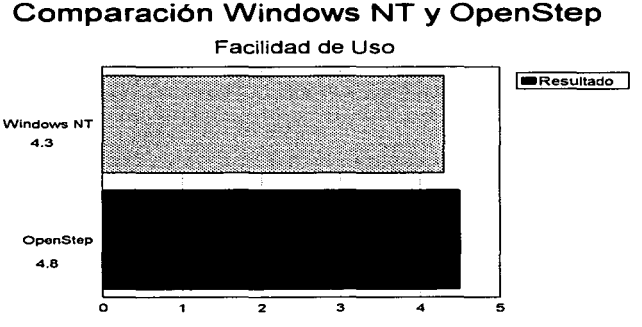
confrontación es realizada con la finalidad de mostrar al usuario las diferencias entre dos sistemas creados para realizar aplicaciones cliente/servidor. OpenStep cuenta con la ventaja de poder trabajar como huésped con varios sistemas operativos entre los que se incluye Windows NT, lo que hace transportable la información sin importar los sistemas operativos (UNIX, MachOS, Solaris, HP-UX, Windows NT, Windows 95) o diversas plataformas que se utilicen en la actualidad.



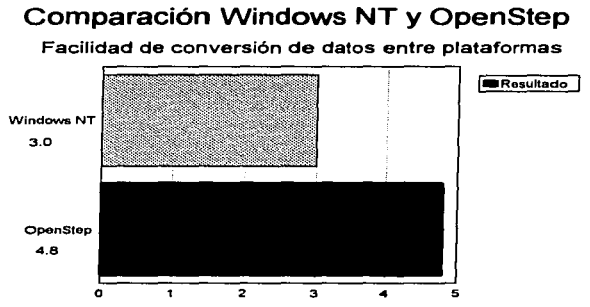
Gráfica 2.1 Facilidad de Instalación



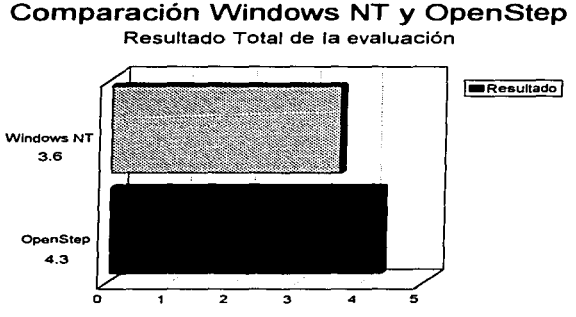
Gráfica 2.2 Facilidad de Programación



Gráfica 2.3 Facilidad de Uso



Gráfica 2.4 Facilidad de Conversión



Gráfica 2.5 Evaluación Final

Con las gráficas anteriores podemos darnos una idea de las diferencias que existen entre cada uno de los sistemas analizados, no tratamos de crear una polémica acerca de estos resultados, ya que no es la finalidad de nuestra tesis observar las diferencias que existen entre estos sistemas, sino presentar otra alternativa diferente a las utilizadas en la programación orientada a objetos, la cual a nuestra consideración podría ayudar a desarrollar sistemas cliente/servidor de una manera sencilla, así como utilizar las ventajas que nos presenta UNIX y prepararnos para lo que consideramos inminente, desarrollar aplicaciones para sistemas abiertos.

2.3 El ambiente de OpenStep

OpenStep maneja una interface gráfica basada en el Display PostScript, lo que lo hace muy potente e intuitivo; además cada aplicación es representada por un objeto que tiene semejanza con el mundo real, por ejemplo, si un archivo es un texto Open Step maneja el objeto como si fuera una hoja de papel escrita, si en el texto está incluida alguna imágen maneja un objeto en forma de una hoja escrita que contiene algún dibujo, si se trata de representar un directorio lo maneja en forma de un folder etc.

OpenStep maneja una poderosa versión de Unix en el sistema operativo; como es un sistema operativo orientado a objetos todo en él es un objeto desde un menú, un directorio, un archivo, las ventanas, etc.; además de ser multiusuario; por lo que para tener acceso al ambiente es necesario contar con un login y un password.

2.3.1 Display Postscript

El lenguaje PostScript fue el primero en implementar el manejo de las impresoras, así se introdujo el concepto de utilizar un dispositivo independiente capaz de obtener los mismos resultados obtenidos en pantalla que en la impresión.

Las dos primeras implementaciones del lenguaje PostScript fueron usadas para crear gráficos de gran calidad en las minicomputadoras de Evans & Sutherland y Xerox.

Adobe y Next comenzaron a desarrollar un Software en PostScript para visualización en la pantalla (el sistema de Display PostScript) en 1985, en ese mismo año también apareció la primera impresora que incorporó el lenguaje PostScript. Introduciéndose en 1988 como parte del software de las computadoras Next. El display PostScript es un sistema para desplegar en la pantalla modo texto y modo gráfico de gran calidad.

Este modelo de imágenes usa el lenguaje PostScript para brindar poderosas capacidades gráficas para el display y es completamente compatible con el modelo encontrado en las impresoras PostScript.

El sistema de Display PostScript fue desarrollado originalmente para la plataforma Next porque Adobe y Next compartían una visión similar del futuro de la computación; ambos tomaron en cuenta la ventaja de tener el mismo modelo de imagen manejado en la pantalla que el obtenido en la impresora.

Estos avances hicieron posible lo que conocemos como What You See Is What You Get (lo que se ve es lo que se obtiene) que es la correspondencia entre las imágenes o texto que vemos desplegado en la pantalla y la forma en como se ven en la impresión.

Los dibujos pueden tener un alto nivel en su manejo y libertad en su desarrollo para poder observar detalles específicos como es resolución de pantalla y color.

Este sistema permite tener un control gráfico total en el desarrollo de sistemas, debido a que es posible modificar arbitrariamente la escala del dibujo sin perder con ello sus características principales; las aplicaciones Display PostScript ofrecen la tecnología necesaria para los requerimientos de alta calidad de impresión y de visualización.

Los sistemas Display PostScript contienen un intérprete PostScript y un traductor de ambiente, el cual es utilizado directamente por el sistema operativo que permite manipular los eventos y manejo de pantallas así como la modificación de textos e imágenes dentro de las aplicaciones

2.3.2 El Ambiente de Openstep

OpenStep maneja una interface gráfica que permite su fácil interrelación con el ambiente, la cual está basada en el Display PostScript (imágenes y texto que tanto en la pantalla como al imprimirse se observan de la misma manera).

La interface gráfica de OpenStep hace que el ambiente sea amigable e intuitivo, ya que maneja al igual que Windows ventanas, menús, íconos etc.

Para tener acceso al ambiente es necesario contar con un login y un password porque OpenStep es un sistema multiusuario y cuenta con la seguridad que da Unix.

Las dos formas de accesar a OpenStep son: como Root o como Usuario; como en Unix el Root es el usuario de mayor jerarquía (también conocido como superusuario) este puede crear o dar de baja usuarios, cambiar el login, otorgar los permisos con los que va a contar el usuario, etc.

El usuario puede hacer uso del ambiente sujeto a los permisos que el Root le concede y puede usar sólo las aplicaciones que se le tienen permitidas.

Una vez que se accesa al sistema aparece una pantalla como la que se muestra en la figura 2.3; está es la interface gráfica de OpenStep.

Al iniciar OpenStep se ejecuta una aplicación que se llama Workspace Manager, ésta es el administrador del sistema.

El uso Workspace Manager es simple e intuitivo ya que contiene elementos comúnes como son: ventanas, íconos, menús, etc.; que son manipulados mediante el mouse que es el elemento principal que nos permite interactuar con el ambiente.

Para conocer el manejo del ambiente de OpenStep es necesario entender el funcionamiento del mouse, las características de las ventanas, los tipos de controles y el Workspace Manager.

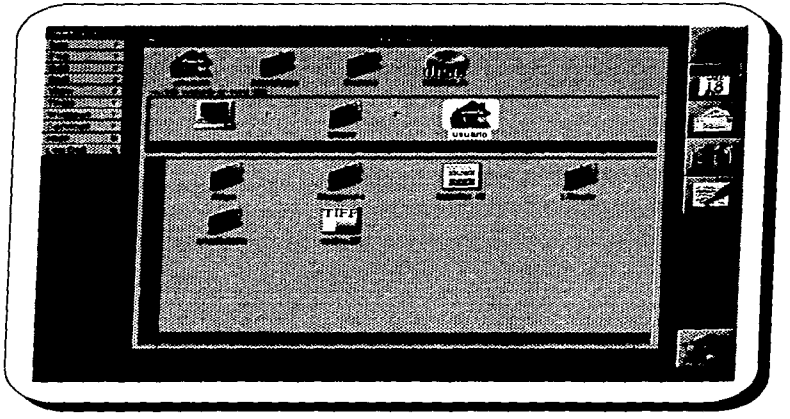


Figura 2.3 Inicio de OpenStep

2.3.2.1 Uso del mouse.

En OpenStep el elemento principal que se usa para interactuar con la computadora es el mouse; las tres acciones básicas que se hacen con el mouse son:

1. Click sencillo
2. Doble click
3. Arrastrar

Click

El click lo utilizamos cuando activamos un menú, nos cambiamos de una ventana a otra, insertamos texto en alguna parte de un archivo, etc.

Doble Click

El doble click nos sirve cuando activamos alguna aplicación, abrimos un directorio, un documento, vemos una imagen o el contenido de algún archivo, etc.

Arrastrar

La función de arrastrar sirve para mover los objetos, cambiar de lugar alguna aplicación o insertarla en application dock , copiar, borrar, ligar, para seleccionar líneas de texto completas, para señalar varios archivos al mismo tiempo, etc.

Cuando arrastramos un objeto se forma un fatasma que desaparece en cuanto colocamos el objeto en otro lugar, como el que se muestra en la figura 2.4.

El cursor del mouse también nos indica las acciones que se están realizando en ese momento:

1. **La Flecha.** Es el cursor más común y es el que nos sirve para seleccionar algún objeto o hacer un click sencillo en alguna aplicación.

2. **Barra en forma de I.** Este nos indica la posición del cursor en el editor de textos.

3. **El disco girando.** Nos indica que se está realizando algún proceso como: abrir un directorio, copiar un archivo, activar alguna aplicación, guardando

un documento etc.; y nos señala que hay que esperar antes de realizar otra acción.

4. **El Lápiz.** Indica que estamos en una aplicación como el Icon Builder, en donde podemos dibujar o modificar alguna imagen etc.

5. **Una cruz.** Sirve para indicarnos que arrastrando el mouse podemos dibujar un rectángulo o un cuadrado del tamaño que deseemos en una aplicación de dibujo.

6. **Dos páginas.** Indica que se está copiando algún archivo dentro de un directorio o en una unidad de disco por ejemplo.

7. **El cursor con doble flecha.** Indica la operación de unión que se va a realizar en el workspace manager u otra aplicación.

2.3.2.2 Características de las ventanas.

La interface gráfica de OpenStep maneja las ventanas como un objeto de uso muy común lo que facilita su uso; en el sistema se manejan varios tipos de ventanas como son:

1. Las ventanas comunes.
2. Los paneles
3. Las miniventanas.
4. Los menús.
5. Las listas POP-UP y PULL-DOWN

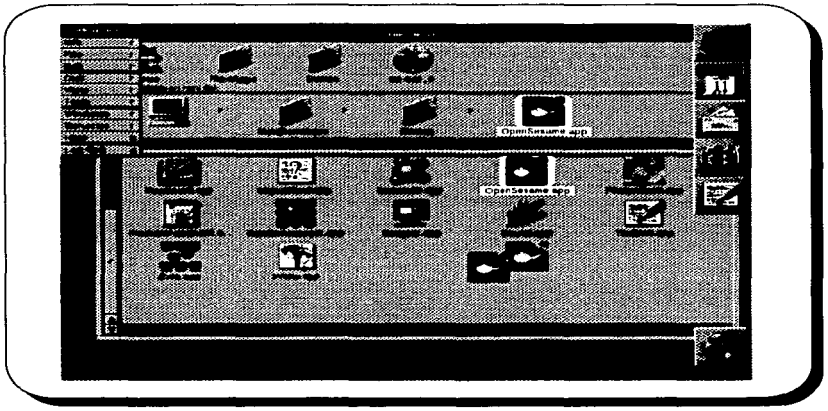


Figura 2.4 Fantasma de una aplicación.

Ventanas comunes

Son las ventanas que aparecen cuando abrimos algún archivo de texto, un dibujo, una imagen, etc., éstas ventanas como se puede observar en la figura 2.5 constan de varios elementos:

a) En la parte superior tienen una barra para el título de la aplicación que incluye su vía de acceso.

b) En la esquina superior izquierda tienen un pequeño botón que sirve para minimizar la ventana.

c) En la esquina superior derecha tienen un botón que indica si el documento está guardado o no; si tiene una cruz completa indica que está guardado en caso contrario el documento no ha sido guardado. Ese mismo botón sirve para cerrar la ventana.

d) En la parte inferior tienen una barra que sirve para modificar el tamaño de la ventana con el mouse.

e) En la parte izquierda cuando el archivo es muy grande y no se puede ver completo aparece una barra y unos botones con flecha que sirven para ir desplegando el documento poco a poco.

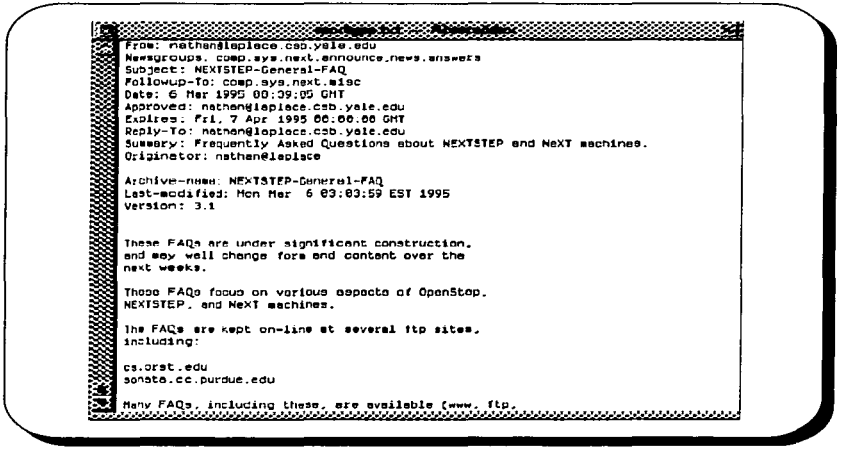


Figura 2.5 Ventana Común

Paneles.

Son las ventanas que aparecen en el caso de que alguna aplicación tenga que brindar alguna información al usuario; hay dos tipos de paneles los de atención y los de control.

El panel de atención requiere una respuesta del usuario antes de continuar con el trabajo que se esté realizando; un panel de atención sería por ejemplo el que aparece cuando se va a imprimir algo y la impresora no se encuentra en línea, o como el ejemplo de la figura 2.6, cuando cerramos algún documento y no se ha salvado nos pregunta si queremos cerrar guardando el archivo o no.

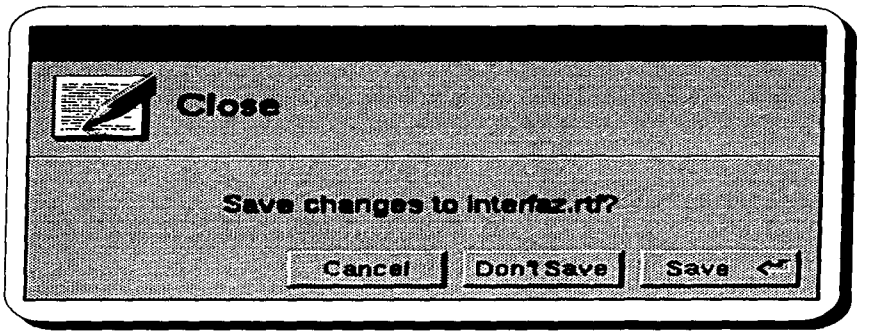


Figura 2.6 Panel de Atención

El panel de control es usado cuando se necesita dar una información extra o configurar alguna aplicación como en el ejemplo de la figura 2.7, cuando se activa la aplicación de preferencias, podemos modificar el color del escritorio, el idioma del teclado, etc.

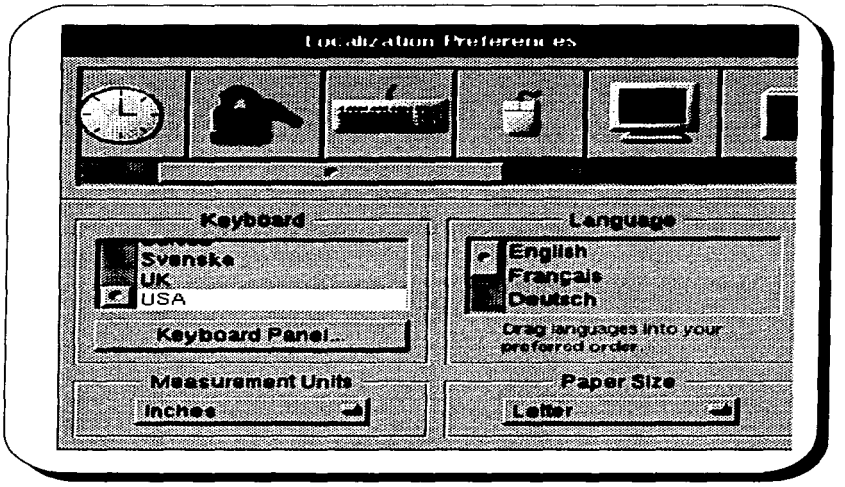


Figura 2.7 Panel de control

Miniventanas.

Es un icono pequeño que representa a una ventana que fue minimizada, para devolver la ventana a su tamaño original basta con dar doble click en la misma; por ejemplo en la figura 2.8 podemos observar una ventana minimizada de un archivo de texto.

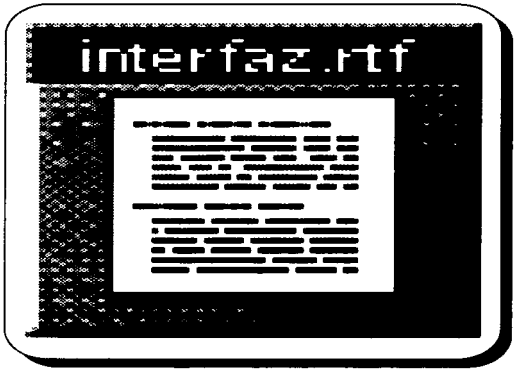


Figura 2.8 Miniventana

Menús.

Un menú puede considerarse como una lista vertical de comandos o acciones a realizar, que se activan mediante un click del mouse. Si alguna de las acciones del menú en la parte derecha tiene una flecha indica que esa acción activa un submenú; siempre que se activa alguna aplicación se activa también su menú correspondiente un ejemplo de los menús y submenús que maneja OpenStep lo podemos observar en la figura 2.9.

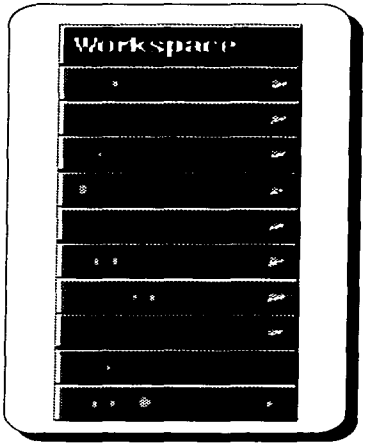


Figura 2.9 Menú de OpenStep

Listas Pop-Up y Pull-Down.

Las listas pop-up y pull-down que utiliza OpenStep son pequeñas ventanas que están incluídas en otra; y están representadas como botones que se activan mediante un click con lo cual despliegan sus acciones; para seleccionar una hay que arrastrar el mouse.

En las listas pop-up una vez que se hace la selección, está aparece en la parte de arriba, la selección de las aciones también la podemos hacer por medio del teclado ya que en la parte izquierda tienen un número, y podemos cambiar entre acciones como en la figura 2.10.

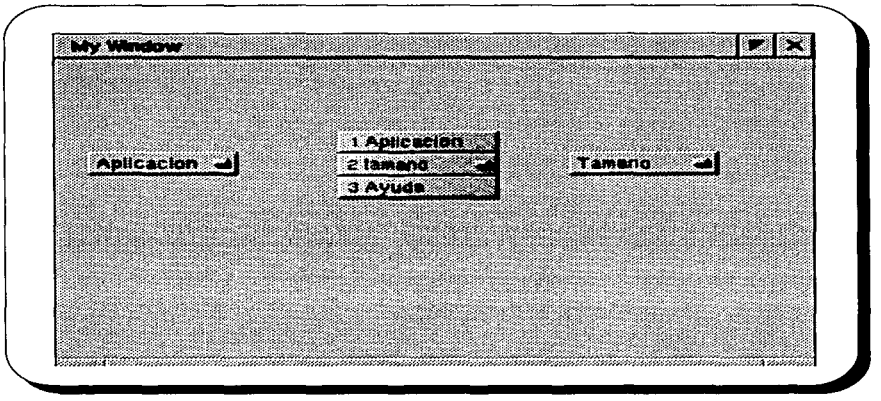


Figura 2.10 Lista Pop-Up

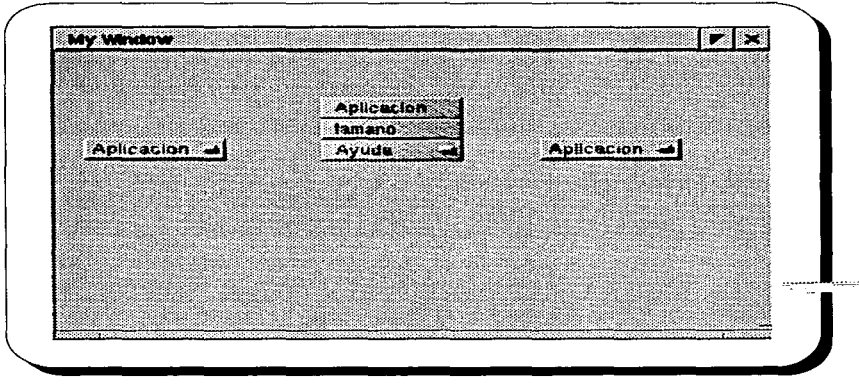


Figura 2.11 Lista Pull-Down

Las listas pull down funcionan como menús, una vez que selecciona una acción está se ejecuta y no se altera la lista como en el caso de las pop-up como se puede ver en la figura 2.11.

2.3.2.3 Tipos de control

Existen 6 tipos de controles que OpenStep maneja en su interface gráfica como podemos ver en la figura 2.12:

1. Botones
2. Menú de comandos.
3. Campos de texto.
4. Barras de desplazamiento y de despliegue.
5. Visores y listas de selección
6. Campos de color

Botones.

En OpenStep los botones se pueden clasificar en dos grupos: los de acción que con un click llevan a cabo una acción determinada y los que tienen 2 estados como por ejemplo con un click se activa y con otro se desactiva alguna aplicación .

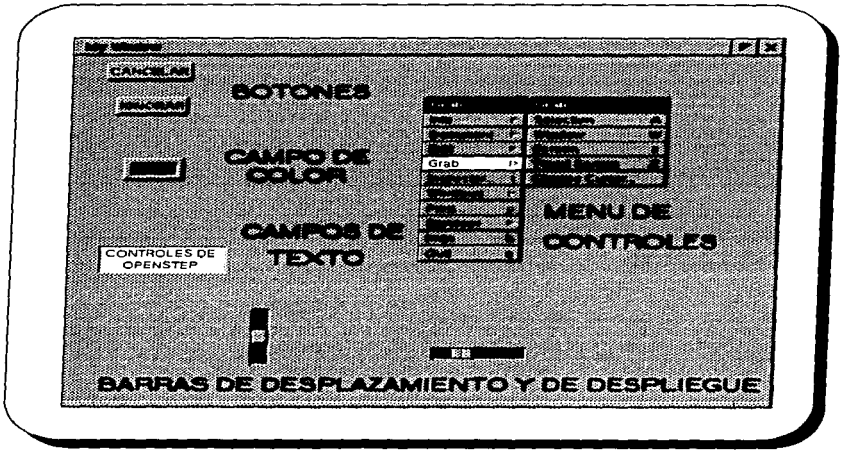


Figura 2.12 Controles de OpenStep

Menú de comandos.

Los menús de comandos en OpenStep se manejan jerárquicamente, ya sea en forma de lista o como un conjunto de botones y se activan mediante un click ya sea en el nombre de la celda del menú o en el botón indicado.

Campos de texto

Un campo de texto es un área rectangular que puede desplegar una línea sencilla de texto. El texto puede ser editado mediante un doble click en esa área y se usan por ejemplo para colocar en una ventana alguna indicación.

Barras de desplazamiento y despliegue.

Las barras de desplazamiento aparecen en forma de botones y se utilizan para modificar poco a poco el estado de algún objeto como por ejemplo el brillo de la pantalla o el contraste.

Las barras de despliegue aparecen en la parte izquierda o inferior de las ventanas y se utilizan para desplegar poco a poco el contenido de un archivo que sea más grande que la ventana.

Visores y listas de selección.

Los visores y listas de selección son usados para elegir uno o más elementos en una lista; la diferencia radica en que los visores utilizan listas múltiples que están organizadas en forma jerárquica para mostrar los datos, como un archivo, un folder, etc.; mientras que la lista de selección contiene opciones simples.

Campos de color.

Los campos de color son utilizados para seleccionar y manipular colores, cuando se da un click con el mouse aparece la paleta en la cual mediante un click se selecciona un color y se arrastra a donde queremos cambiar el color.

2.4 Workspace Manager.

El workspace manager es la aplicación que se ejecuta automáticamente al entrar al ambiente, y la que permite el manejo y acceso a otras aplicaciones; es el administrador de recursos del sistema, por ejemplo se puede crear, duplicar, organizar, buscar, renombrar y borrar archivos y directorios; iniciar y manejar aplicaciones; accesar a los dispositivos de almacenamiento como disco duro, drives, cd-room; configurar y optimizar los recursos de la máquina, etc.

El Workspace Manager está constituido por 4 partes que son:

1. El Visor de archivos (File Viewer)
2. El menú principal
3. La Barra de Herramientas (Aplication Dock)
4. El reciclador (Recycler)

2.4.1 El visor de archivos (File Viewer).

El visor de archivos (File Viewer) es la ventana que aparece en el workspace manager, en ella podemos observar y manipular los archivos, los directorios, aplicaciones y documentos almacenados en la computadora.

El visor de archivos (File Viewer) está conformado por las tres partes que se indican en la figura 2.13.

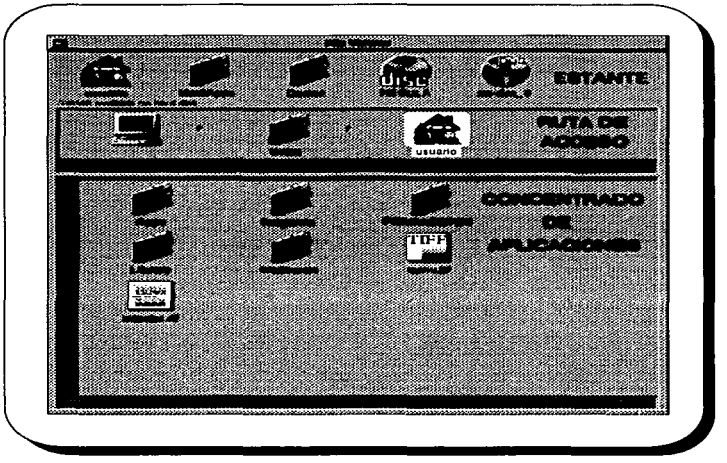


Figura 2.13 Visor de Archivos (File Viewer)

2.4.1.1 El estante (Shelf).

Se usa para colocar los directorios o archivos que se utilizan con mayor frecuencia; esté comienza siempre con nuestro directorio raíz representado por una casa siguida de los directorios que se utilizan con mayor frecuencia, como se indica en la figura 2.14; logrando de está forma que el acceso sea más rápido.

Figura 2.14 Estante de OpenStep

En la parte inferior izquierda nos indica el espacio que tenemos disponible en el disco duro y el parte inferior derecha los procesos que se están llevando a cabo.

El estante se puede personalizar arrastrando los directorios de uso frecuente logrando de está forma que el acceso a los mismos sea más rápido y fácil. Aquí no es recomendable colocar aplicaciones, solamente directorios o archivos, ya que para este fin OpenStep cuenta con la barra de herramientas.

2.4.1.2 La ruta de acceso (Icon Path).

Es la representación gráfica de los directorios del sistema y nos indica en que parte del mismo estamos trabajando, siempre comienza con el directorio del Root representado por una terminal Next. Como se ve en la figura 2.15.

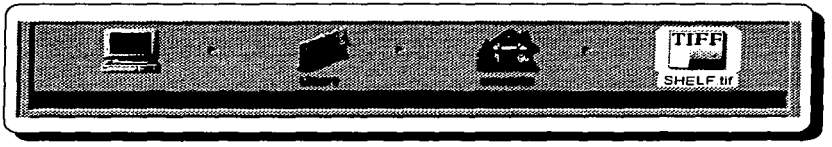


Figura 2.15 Icon Path

2.4.1.3 El concentrado de aplicaciones.

Nos muestra los archivos, directorios y aplicaciones que se encuentran almacenados en el directorio que nos indica la ruta de acceso.

En el concentrado de aplicaciones todos los objetos están representados lo más cercano a la realidad, lo que hace que sea intuitivo su uso. De está forma como podemos observar en la figura 2.16, los directorios quedan representados como folders, los archivos son una carta, etc.

2.4.2 La barra de herramientas (Application Dock).

La barra de herramientas es una colección de las aplicaciones que se utilizan con mayor frecuencia, logrando así que sea más fácil y rápido el acceso a las mismas.

Cuando se entra por primera vez al sistema en la barra de herramientas solamente se encuentran las aplicaciones más comunes como el workspace manager, Las preferencias, el correo (mail), la librería y el editor como se puede observar en la figura 2.17.

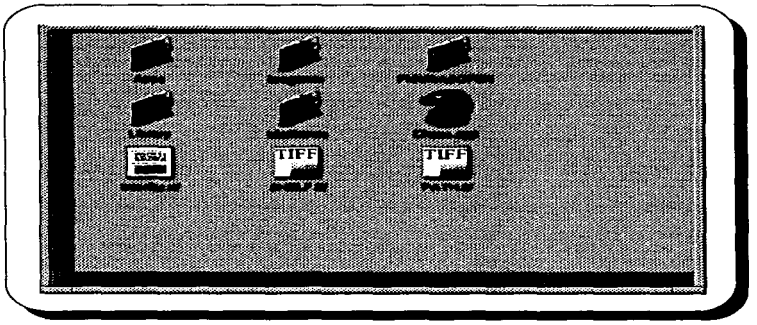


Figura 2.16 Concentrado de aplicaciones

Las aplicaciones de la barra de herramientas se pueden personalizar, de acuerdo a las necesidades de cada usuario de manera sencilla; cuando se quiere incorporar una aplicación, ésta se tiene que arrastrar del visor de archivos (file viewer) a la barra de herramientas.

En el caso de que se quiera borrar alguna aplicación de la barra; se tiene que arrastrar ésta de la barra al visor de archivos (file viewer); conforme se va arrastrando la aplicación, en la barra aparece el fantasma del icono; cuando la aplicación se suelta ese fantasma desaparece, lo que significa que la aplicación ha sido borrada de la barra más no del sistema ya que al soltar la aplicación, el icono desaparece de la barra y regresa a su localización original en el sistema.

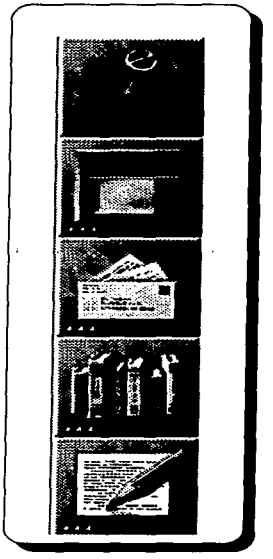


Figura 2.17 Barra de Herramientas de OpenStep

Una característica importante de la barra de herramientas es que cada aplicación en la parte inferior de su icono tiene tres puntos como se observa en la figura 2.18. Si la aplicación tiene los puntos significa que la aplicacion no está activada en caso contrario significa que la misma está en uso.

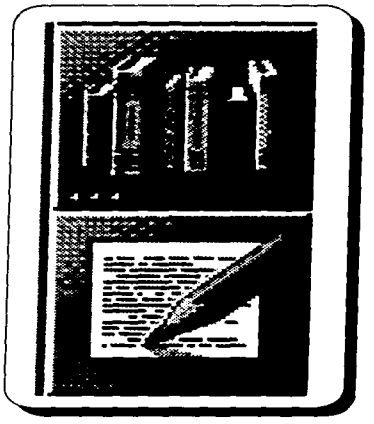


Figura 2.18 Ejemplo de una aplicación activada y otra desactivada

2.4.3 El Reciclador

El Workspace Manager cuenta con una herramienta de reciclaje la cual está representada en la figura 2.19, está sirve para destruir archivos, aplicaciones, directorios etc. arrastrando sus íconos al reciclador; cuando se suelta algún archivo o directorio, esté comienza a girar, con lo cual nos está indicando que está trabajando.

En caso de querer recuperar algún archivo que haya sido arrojado al reciclador basta con dar un doble click en su icono, entonces se despliega una ventana con los íconos de los archivos que han sido arrojados como la que aparece en la figura 2.20; para recuperarlos sólo basta arrastrarlos del reciclador al visor de archivos (File viewer).

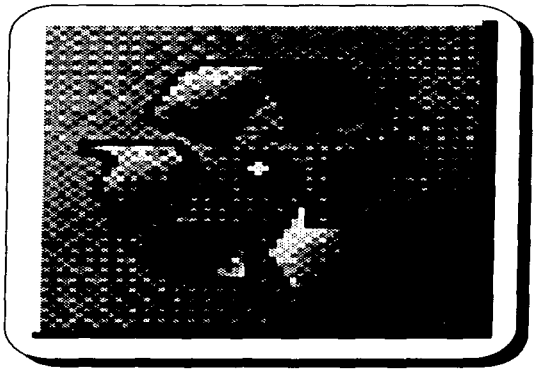


Figura 2.19 Icono del reciclador

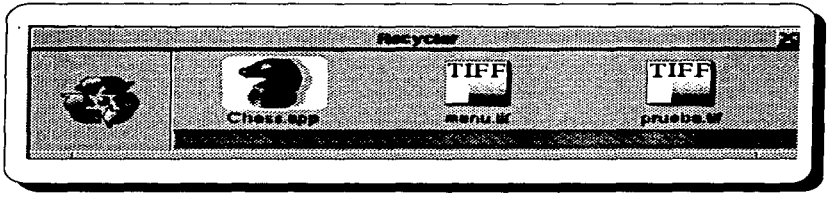


Figura 2.20 Ventana con archivos reciclados

Esta es una herramienta muy útil ya que en Unix una vez que se borra un archivo ya no hay forma para recuperarlo.

2.4.4 El Menú Principal.

El menú principal del workspace manager está estructurado en forma jerárquica como se muestra en la figura 2.21, como podemos observar hay opciones que están marcadas con un pequeño triángulo, esto nos indica que la opción contiene un submenú.

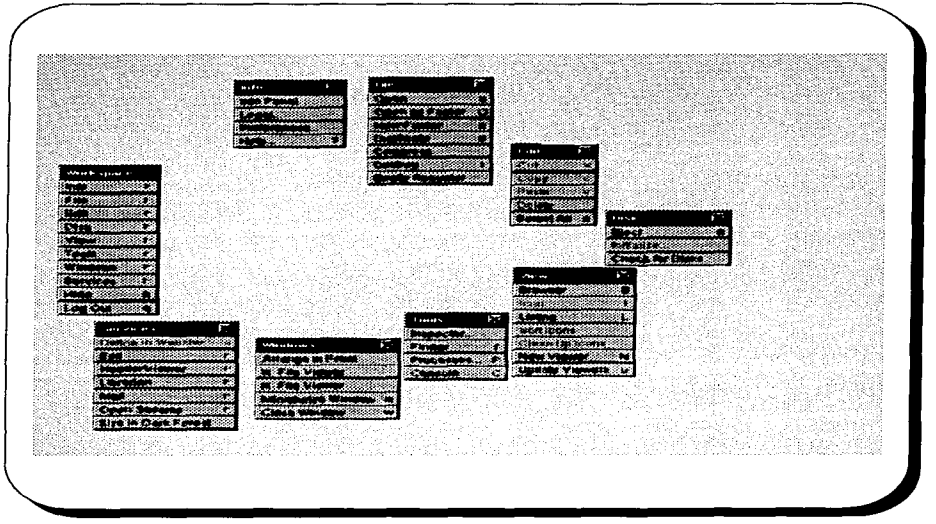


Figura 2.21 Menús y Submenús de OpenStep

En el menú aparecen las opciones de:

1. Info
2. File
3. Edit
4. Disk
5. View
6. Tools
7. Windows
8. Services
9. Hide
10. Log Out

Info

Este submenú cuenta con opciones como la del info panel, la cual nos indica quién hizo la aplicación etc; la opción de preferencias y la de ayuda.

File

Contiene opciones para manipular los archivos y directorios del Workspace Manager, tales como abrir un archivo, abrir alguna aplicación en forma de archivo, crear un directorio, duplicar, comprimir y destruir un archivo o directorio.

Edit

Cuenta con las opciones básicas de edición como son la de cortar, pegar, copiar, deshacer y seleccionar todo.

Disk

Disk cuenta con las opciones necesarias para detectar las unidades de almacenamiento, formatear disquetes.

View

Contiene todas las opciones de visualización, esto es ver los archivos o directorios en forma de lista que nos muestra el tamaño del archivo, su ruta sus atributos, etc; en forma gráfica con sus iconos correspondientes etc.

Tools

Son las herramientas básicas de OpenStep como el inspector de atributos, un browser, que sirve para buscar documentos con contenido idéntico, o algún archivo en especial, una consola que indica lo que se ha realizado desde el principio de la sesión y una ventana de procesos que indica que es lo que se está haciendo en ese momento.

Windows

Contiene las opciones para situar adelante alguna ventana, para ver el visualizador, para miniaturizar la ventana, cerrarla, etc.

Services

Aquí se encuentran algunas de las utilerias más usadas como el omniweb, la librería, el correo, el editor etc.

Hide

Está función sirve para ocultar el workspace manager.

Log out

Sirve para salir del sistema.

2.5 Perspectivas de OpenStep

La red Internet es hasta el día de hoy lo más novedoso dentro de la industria computacional, es importante tomar en cuenta que es el futuro dentro de la industria debido a que cada día se esta tratando de globalizar la información con la finalidad de poder tener acceso a la misma desde cualquier parte del mundo y en cualquier momento.

2.5.1 OpenStep en el Web.

NeXT, creadora de OpenStep es la única empresa de desarrollo que esta siempre a la vanguardia computacional, pudiendo mencionar que en ella surgió la idea de realizar el correo (mail) multimedia, es decir, mandar mensajes no sólo con palabras escritas sino que agregándoles gráficas, música y mensajes hablados, situación que hace a los mensajes más interpersonales y permite humanizar la computación, surgiendo también la idea de instalar programas por medio de CD-ROM y con ello ahorrar tiempo y esfuerzo al usuario al no tener que estar introduciendo disquete por disquete, sistemas que puedan leer todos los formatos conocidos utilizados por sistemas operativos (MS-DOS, Macintosh, Windows, UNIX, OpenStep), con la finalidad de ayudar al usuario a poder conocer que información se encuentra dentro de esos disquetes, conexión inmediata a Internet, así como a la dirección del periódico de noticias de NeXT, lo que ayuda a que la persona que utilice éste sistema siempre se encuentre

informado de lo que sucede en el mundo, dentro del ambiente que el utiliza, la interface gráfica es tan amigable que muchas ideas fueron utilizadas para crear programas que en la actualidad utilizamos en el ambiente Windows, el sistema Plug and Play (Conectalo y listo) el cual ayuda al usuario a no cometer errores a la hora de instalar equipo. A partir del correo multimedia fue desarrollado el formato HTML (Lenguaje Creador de Hipertexto), el cual ha sido pionero en la elaboración de lugares de acceso dentro de Internet.

OpenStep ha permitido junto con otras herramientas de desarrollo directamente creadas para Internet e Intranet, elaborar sitios de interés dentro de la red, lo que permite hacerlos interactivos, y sencillos de mantener.

En la actualidad la compañía NeXT ha dado un giro total al mercado ofreciendo soluciones a las necesidades que existen, debido a ello se crean sistemas abiertos, los cuales puedan trabajar en plataformas tan variadas y pudiéndolas clasificar como soluciones para Windows NT, UNIX el World Wide Web. Los objetos que ofrece al mercado han sido diseñados desde hace más de 10 años, lo que permite que sean ampliamente transportables hacia todos los sistemas; proporciona desarrollos de misión crítica para Windows NT y el World Wide Web.

Resumiendo, se desea crear aplicaciones de misión crítica de manera rápida, que proporcionen una ventaja competitiva en el mercado.

2.5.2 WebObjects

La necesidad de implementar nuevos horizontes ha hecho voltear a lo que en este momento muestra un gran futuro, es decir, la red, dentro de ésta podemos

observar dos aspectos importantes Intranet, e Internet, los cuales son posibles de atacar por medio de objetos dinámicos, teniendo en cuenta que un ambiente dinámico basado en el reuso de componentes y con las herramientas de desarrollo suficientes, permite realizar aplicaciones para todos los ambientes visuales, ya sea OpenStep, Windows NT o el mismo World Wide Web.

Las características de las aplicaciones de Web objects son:

1. Nuevas, prestando servicios agradables y con mucho colorido.
2. Los ambientes desarrollados fácilmente pueden ser cambiados.
3. Desarrollo de sistemas a gran escala.
4. Aplicaciones dentro del Web rápidas de desarrollar.
5. Rápido desarrollo dentro de Internet.

Es importante hacer referencia a que una aplicación desarrollada para Internet puede llevarse a cabo en alrededor de 30 días; una aplicación Intranet de 30 a 60 días de desarrollo, mientras que una aplicación multiplataforma de 4 a 6 meses, pero tomando en cuenta que todo lo desarrollado por nosotros puede posteriormente ser reutilizado. Con esta nueva alternativa de desarrollo dentro del Web un desarrollador puede crear soluciones empresariales para el Web dentro del ambiente de Windows NT, o cualquier otro sistema basado en UNIX.

El desarrollar dentro de Web objects permite que:

1. El Web pueda cambiar de un día a otro su aspecto, como se necesite.

2. La infraestructura que presente reduzca considerablemente el tiempo de mantenimiento.

3. El crear aplicaciones con objetos dinámicos, que faciliten el acceso.

Es importante construir en el Web debido a varios aspectos entre los que se encuentran:

1. Presentar un Sistema de Información Global
2. Lograr el acceso a plataformas cruzadas
3. Navegar fácilmente con múltiples tipos de contenidos.

Es importante como ingenieros observar hacia donde se dirige la tecnología, con la finalidad de estar siempre al día de las necesidades que presenta el mercado y con ello estar un paso adelante de lo que llegaran a necesitar nuestros clientes en un futuro inmediato, y clasificarlas dependiendo de lo que podamos ofrecer, ya sea utilizar el Web de manera personal, como publicidad, trabajo en grupo, servicios al cliente, mercadotecnia, comercio, etc.

En la figura 2.22 podemos observar la tendencia del mercado hacia la creación de Intranets, situación por la cual debemos de prepararnos en este terreno para sacar el máximo provecho de dichas tendencias.

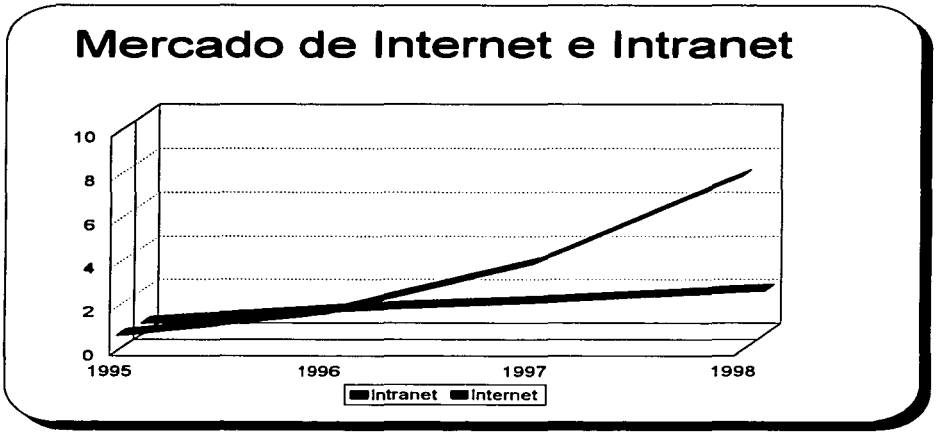


Figura 2.22 Tendencias del Mercado del Web

Es importante mencionar que la tendencia de que las herramientas de desarrollos de NeXT estén pensando en el Web es debido simplemente a que el Web nació gracias a la tecnología de NeXT; en 1990 Tim Berners-Lee desarrollo la primera aplicación cliente y servidor de World Wide Web usando las herramientas de desarrollo de NeXT en sólo 2 meses. La combinación del ambiente orientado a objetos de NeXT, así como las herramientas de desarrollo y las librerías hicieron posible esto.

Consideramos que debemos estar listos a los requerimientos del mercado ya que nuestro campo de trabajo cambia día con día y en caso de no estar preparados perderemos la ventaja tecnológica que nos ofrece manejar las herramientas nuevas de desarrollo.

CAPITULO III

" LAS HERRAMIENTAS DE DESARROLLO DE OPENSTEP "

3.1 Objective C.

Objective C amplía el lenguaje de programación orientado a objetos, éste ha sido descrito por Brad Cox y uno de los conceptos fundamentales del mismo es sinónimo de circuito integrado (Software-IC). Brad Cox supusó que los Ingenieros de software construirían aplicaciones utilizando componentes (**clases**) predefinidos conocidos como Chips de Software que aumentan notablemente la productividad del software.

El lenguaje Objective C se construye sobre la idea de que los circuitos integrados o chips de software; son la base del desarrollo de los programas de OpenStep; esta basado directamente en C ANSI con extensiones a objetos y en los conceptos de **clase, envío de mensajes** y un tipo de datos conocido como **identificador de objetos.**

El compilador de Objective C proporciona al desarrollador la opción de poder ejecutar programas creados en C ANSI (terminación .c), en C++ (terminación .C), así como los archivos de código elaborados en Objective C (terminación .m). Es posible compilar programas desarrollados en C++ debido a que éste es considerado por Objective C como un " ***C mejor*** " , no como un lenguaje orientado a objetos.

El Objective C esta basado principalmente en objetos, datos asociados al mismo y las operaciones particulares que lo afectan o usan. En éste lenguaje los operadores son conocidos como *métodos del objeto* y los datos son afectados por las variables de instancia.

En esencia podemos decir, que un objeto une una estructura de datos (variables de instancia) y un grupo de procedimientos (métodos), dentro de la unidad del programa.

Vamos a utilizar para ejemplificar el uso de comandos un objeto matriz, en el cual podemos introducir datos. En la figura 3.1. podemos observar algunos tipos de matrices que proporciona el sistema.

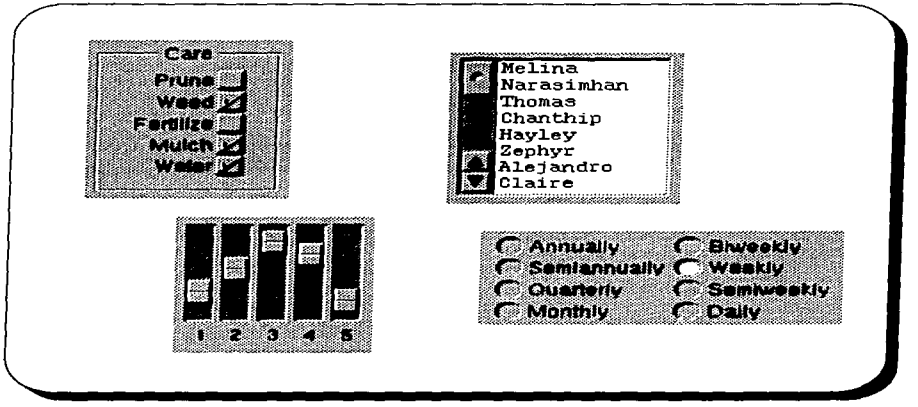


Figura 3.1 Tipos de Matrices

Un objeto puede crearse como una matriz, donde las variables de instancia, están definidas de tal forma que incluye las dimensiones, coordenadas, la fuente de letra con la cual se observará, las celdas dentro de los renglones y columnas.

Cada celda dentro de una matriz es un objeto, las celdas tienen las variables de instancia que contienen las acciones que van a realizar, los métodos

determinan los cambios dentro de las celdas. En Objective C, las variables de instancia están dentro de los objetos.

Al programar con Objective C es importante utilizar los tipos de variables; los identificadores distintivos del tipo de datos son **id**. Este tipo es definido como un apuntador a un objeto, en realidad podemos decir, que es un apuntador a los datos del objeto (variables de instancia). Todos los objetos sin considerar sus métodos o variables de instancia se declaran como **id,** ejemplo:

id Objeto;

Para que los constructores orientados a objetos de Objective C, devuelvan un valor el tipo de datos **id** es reemplazado por **int** por ejemplo:

int Objeto

Para poder enviar un mensaje de un objeto a otro es necesario utilizar una sintaxis especial escrita entre corchetes, ejemplo:

[mensaje recibido];

Al escribir el código fuente, el mensaje es el nombre del método al que se va a dirigir él mismo y los argumentos que se mandarán hacia él. Cuando un mensaje es enviado el sistema de ejecución selecciona el método apropiado para recibir e invocar a éste.

Por ejemplo, para poder observar los datos contenidos en el objeto **miMatriz** se utiliza el método **display** , que permite observar a la matriz y sus celdas en una ventana:

[miMatriz display];

Los métodos también pueden tomar algunos argumentos. Por ejemplo el mensaje cambiará la ubicación dentro de una ventana de nuestro objeto matriz, es decir, **miMatriz** lo desplazaremos a las coordenadas de la ventana (10.0, 24.0):

[miMatriz moveTo: 10.0 : 24.0];

En este ejemplo el método se llama **moveTo::** , el cual tiene dos puntos, uno para cada argumento. Los argumentos se introducen después de cada dos puntos (:). Éstos no tienen que ser agrupados al final del nombre de un método. Comúnmente una palabra clave describe el argumento que antecede a cada dos puntos. Por ejemplo el método **getRow:andColumn:ofCell**: toma tres de argumentos:

```
int renglon, columna;
[miMatriz getRow:&renglon andColumn:&columna ofCell:someCell];
```

Este método encuentra **someCell** en la matriz y pone el renglón y columna donde se ubican las variables .

3.1.1. Mensajes

3.1.1.1. Las Variables de instancia del receptor.

Un método tiene acceso automático a las variables de instancia de un objeto receptor; no es necesario transmitir al método como argumento, esta característica simplifica el código fuente de Objective C , apoyándose en la idea que los programadores de sistemas orientados a objetos piensen sobre objetos y mensajes.

Los mensajes se envían a los receptores como si fueran cartas.

Los argumentos del mensaje traen información desde afuera por lo que no necesitan traer al receptor con ellos

3.1.1.2 Polimorfismo

En particular, un objeto es accesado sólo por los métodos que se definieron para ello. No se puede confundir con métodos definidos para otros tipos de objetos, aún cuando otro objeto tiene un método con el mismo nombre. Esto significa que dos objetos pueden responder de manera diferente al mismo mensaje. Por ejemplo, regresando a nuestro objeto que denominamos **miMatriz**, cada tipo de objeto envió un mensaje **display** que se utiliza de manera única, pero la información contenida en **ButtonCell** y un **TextFieldCell** responderá de manera diferente a instrucciones idénticas .

Este aspecto, juega un papel importante en el diseño de sistemas orientados a objetos, junto con la ligadura dinámica, permite que podamos escribir

código que puede aplicarse a cualquier tipo de objetos, sin tener que escoger qué tipos lo podrán utilizar; logrando desarrollar posteriormente objetos que pueden ser utilizados por otros programadores para otros proyectos.

3.1.1.3 La Ligadura Dinámica

Una diferencia importante entre los mensajes y las llamadas a función es que una función y sus argumentos se unen en el código recopilado, pero un mensaje y un objeto receptor no son unidos hasta que el programa corra y el mensaje se envía.

La selección de un método de implementación sucede en el tiempo de ejecución. Cuando un mensaje se envía, se ubica la implementación del receptor de un método que tenga el mismo nombre.

Esta ligadura dinámica de métodos de mensajes trabajan junto al polimorfismo y le dan a la programación orientada o objetos mucha de su flexibilidad y poder.

Por ejemplo en el AppKit, los usuarios determinan que objetos reciben mensajes desde el menú de control, como es Cortar, Copiar, y Pegar, el mensaje va a cualquier objeto que se este seleccionando.

En el ejemplo que estamos utilizando de nuestra matriz, ésta respondería de manera diferente que una Celda. Debido a que los mensajes no seleccionan métodos (los métodos no son comprometidos a los mensajes) hasta el tiempo de ejecución. Cada aplicación puede inventar sus propios objetos que responderán a su manera para copiar los mensajes.

3.1.2 Las Clases.

Un programa orientado a objetos se construye con una variedad diversa de objetos. En OpenStep se puede utilizar un objeto Matriz, un objeto Ventana, un objeto Lista , un objeto reproductor de música, un objeto Texto, etc.

Un programa frecuentemente usa más de un de objeto del mismo tipo o clase, por ejemplo: Listas o Ventanas

En Objective C, definimos los objetos por la definición de su clase. La clase de definición es un prototipo para un tipo de objeto; declaramos las variables de instancia que llegan a ser parte de cada miembro de la clase y define un conjunto de métodos que todos los objetos en la clase pueden usar.

El compilador crea un objeto accesible para cada clase y una clase de objeto que sabe como construir nuevos objetos perteneciendo a la clase. (también se le conoce como fábrica de objetos)

La clase de un objeto es la versión recompilada de la clase; los objetos construidos son las instancias o ejemplos de la clase. Los objetos que harán el trabajo principal del programa son las instancias creadas por la clase objeto al tiempo de su ejecución.

Todos los ejemplos de una clase tienen acceso al mismo conjunto de métodos, y todos ellos a un conjunto de variables de instancia cortadas desde el mismo molde. Cada objeto consigue sus propias variables de instancia, pero los métodos se comparten.

Por convención, el nombre de una clase comienza con una letra mayúscula tal como nuestro ejemplo: Matriz; los nombres de instancias comienzan con una letra minúscula en nuestro ejemplo: miMatriz.

3.1.2.1 La Herencia.

La definición de una clase es aditiva, es decir, cada nueva clase que definimos es con base en otra clase que hereda variables de instancia y métodos. La nueva clase simplemente agrega o modifica lo que hereda. No necesita duplicar el código heredado.

La herencia víncula todas las clases juntas en un árbol jerárquico con una clase única, la clase Objeto o raíz. Cada clase tiene una superclase que le antecede hasta llegar a la raíz, y cualquier clase (incluyendo a la Objeto) puede ser la superclase para cualquier número de subclases que se encuentren más alejadas de la raíz.

La figura 3.2 ilustra la jerarquía de las clases en el AppKit de OpenStep; ésta figura muestra que la clase Matriz es una subclase de la clase Control, y que ella es una subclase de la clase Vista, y ésta última es una subclase de la clase Respuesta y ésta es una subclase de la clase Objeto.

La herencia es acumulativa; por ello el objeto de la clase Matriz tiene las variables de instancia y métodos definidos para la clase Control, Vista, Respuesta, y Objeto, así como también los que se definieron específicamente para la clase Matriz. Con esto podemos concluir que un objeto de la clase Matriz no es único de la clase Matriz, sino también de la clase Control, Vista, Respuesta, y Objeto.

Cada clase puede ser vista como una especialización o una adaptación de otra clase. Cada clase que creamos debe ser una subclase de otra clase ya existente (a menos de que definamos una nueva clase raíz). OpenStep proporciona una gran cantidad de clases para la elaboración de sistemas, con la finalidad de disminuir el tiempo de desarrollo de los mismos.

Una clase no tiene que declarar variables de instancia para cada método.

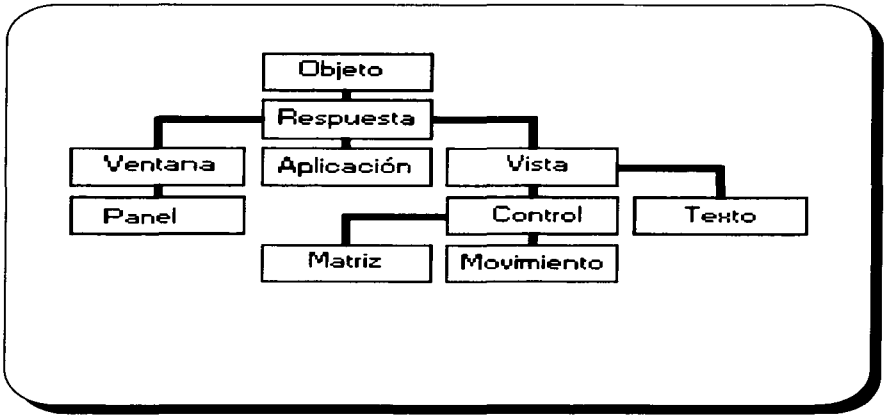


Figura 3.2. Clases del AppKit

3.1.2.1.1 Herencia de Métodos.

Un objeto puede accesar no solamente a los métodos que se definieron para su clase, sino también a los métodos que se definieron para su superclase, por ejemplo un objeto de la clase Matriz puede usar los métodos definidos en las clases: Control, Vista, Respuesta y Objeto así como también los métodos que se definieron en su clase propia.

Este tipo de herencia es un beneficio importante de la programación orientada a objetos.

3.1.2.1.2 Las Clases Abstractas.

Algunas clases son diseñadas para que las otras clases puedan heredar desde ellas. Estas clases abstractas agrupan variables de instancia y métodos que serán usados por un número de diferentes subclases. La clase abstracta es incompleta por ella misma, pero contiene código útil que reduce la carga de implementación de sus subclases.

La clase Objeto es el ejemplo básico de una clase abstracta, debido a que todos los programas directa o indirectamente utilizan instancias de la superclase objeto, pero nunca usan las instancias que pertenecen directamente a dicha clase

.

3.1.3 Elaboración de Clases.

La definición de una clase es una especificación para un tipo de objeto. La clase, define los tipos de datos, la definición de clase se basa no simplemente en la estructura de datos (las variables de instancia), sino también en su comportamiento (métodos).

El nombre de una clase puede aparecer en el código fuente, por ejemplo, como un argumento al operador **sizeof**:

```
int a = sizeof ( Matriz );
```

3.1.3.1 Elaboración de clases estáticas.

Podemos utilizar el nombre de una clase en lugar de **id** para designar tipo de datos de un objeto por ejemplo:

```
Matriz *miMatriz;
```

La declaración de un tipo objeto y su reconocimiento por parte del compilador es conocida como ligadura estática. Así como **id** define a un apuntador o indicador al objeto, los objetos se ligan estáticamente como los apuntadores o indicadores a una clase. Los objetos son escritos siempre por un apuntador. En la ligadura estática la indicación es de manera explícita, es decir, **id** se encuentra oculta.

Un objeto puede ligarse estáticamente en su clase propia o en cualquier clase que hereda. Por ejemplo, desde la herencia hace una Matriz un tipo de Vista, un ejemplo de Matriz podría ligarse estáticamente a la clase Vista :

```
Vista *miMatriz;
```

Esto es posible porque la clase Matriz es una Vista. Es más una Matriz también tiene el método y variables de instancia de una Vista. Para la verificación del código, el compilador considerará miMatriz como una Vista, pero al ejecutarse se tratará como una Matriz.

3.1.4 Clases de Objeto.

La definición de una clase contiene diversos tipos de información, principalmente sobre las instancias de la clase:

1. El nombre de la clase y su superclase
2. La plantilla que describe un conjunto de variables de instancia
3. La declaración de los nombres de los métodos y sus tipos de argumento
4. Los métodos de implementación

Esta información se compila y guarda en las estructuras de datos que están disponibles para el sistema de ejecución. El compilador crea simplemente un objeto, una clase objeto, para representar la clase. La clase objeto tiene el acceso a toda la información sobre la clase, que significa información sobre las instancias

La definición de una clase puede incluir los métodos que se destinaron específicamente a la clase. Una clase objeto hereda clases de métodos desde las clases de mayor jerarquía, así como los instancias heredan métodos de instancia.

En el código fuente, la clase objeto es representado por el nombre de la clase. En el ejemplo siguiente, la clase Matriz devuelve el número de versión que usa el método heredado desde la clase Objeto:

```
int numeroVersion = [ Matriz version];
```

Sin embargo, el nombre de la clase permanece para la clase objeto de manera única como el receptor en una expresión de mensaje. En otra parte del programa, es necesario pedir que una instancia o la clase devuelva la clase id. Ambos responden a una clase mensaje:

```
id unaClase = [unObjeto class];
id matrizClase = [Matriz class];
```

Como estos ejemplos muestran, la clase objeto puede escribirse con id. Pero la clase objeto puede ser más especifica al escribir datos escritos en la Clase :

```
Class unaClase = [unObjeto class];
Class matrizClase = [Matriz class];
```

En el ejemplo anterior todas las clases de objetos son del tipo **Class**. Usar este tipo de nombre para una clase es equivalente a usar el nombre de la clase escrito de manera estática .

En la clase objeto se encuentran los objetos que pueden ligarse dinámicamente, recibir mensajes y heredar métodos de otras clases. Ellos son creados por el compilador, las variables de instancia son propias a excepción de los que se construyeron desde la definición de la clase, y son los encargados de producir instancias en la ejecución.

3.1.4.1 Crear instancias.

La función principal de la clase objeto es la de crear nuevas instancias. Al regresar a nuestro ejemplo la clase Matriz puede crear una nueva instancia Matriz y asignar ésta en la variable miMatriz por ejemplo:

```
id  miMatriz;
miMatriz = [Matriz alloc];
```

El método **alloc** destina memoria de manera dinámica para las nuevas variables de instancia de objetos e inicializa todo como cero (0), exceptuando la variable que conecta la nueva instancia a su clase. Un objeto para ser útil, generalmente necesita ser inicializado. Esto se puede hacer con la función del método **init**. La inicialización generalmente se utiliza después de la distribución por ejemplo:

```
miMatrix = [[Matriz alloc] init];
```

En esta línea de código podemos observar que miMatriz puede recibir cualquier mensaje.

El método **alloc** devuelve una nueva instancia y ésta desempeña un método de inicialización **init** para colocarla en su estado inicial.

Cada clase objeto tiene por lo menos un método (como **alloc**) que permite producir nuevos objetos, y cada instancia tiene por lo menos un método (como **init**) que la prepara para su uso.

Los métodos de inicialización permiten valores particulares que necesitan utilizar instrucciones que indiquen esos argumentos (initFrame: mode: cellClass: numRows: numColumns:), pero todos comienzan con **init**.

3.1.5 Definición de una Clase.

La programación orientada a objetos se basa principalmente en escribir código para nuevos objetos definiendo nuevas clases. En el Objective C, las clases se definen en dos de partes:

1. Una ***interface*** en la que se declaran las variables de instancia y los métodos de las clases y nombre de superclases.

2. Una ***implementación*** que define la clase (contiene el código que ejecutan sus métodos)

Aunque el compilador no requiera de esto, la interface y la implementación se separan comúnmente en dos archivos diferentes. El archivo debe estar disponible para quien usa la clase. Generalmente no se distribuye el archivo de implementación; principalmente debido a que los usuarios no necesitan el código fuente de la implementación.

Un archivo único puede declarar o implementar más de una clase. No obstante, lo más usual consiste en tener un archivo separado de interface y uno de implementación para cada clase, con la finalidad de poder modificar posteriormente sólo alguno de esos archivos.

Al guardar la clase de interfaces de manera separada refleja mejor su comportamiento en entidades independientes.

Los archivos de implementación e interface típicamente se nombran después de la clase. El archivo de implementación tiene una terminación **" .m "** , lo que indica que contiene código fuente de Objective C. Al archivo de interface se le puede asignar cualquier otra extensión. Pero lo más común es utilizar la terminación **".h"** típica de archivos de cabecera. Por ejemplo, la clase Matriz se declara en **Matriz.h** y define en **Matriz.m**.

Al separar la interface de un objeto de su implementación adopta el diseño de los sistemas orientados a objetos.

Un objeto contiene dentro de él su propia personalidad que puede ser observada desde afuera como una "caja negra". Posteriormente debemos analizar como va a interactuar un objeto con otros elementos de un programa, una vez que

hemos declarado nuestra interface podemos alterar libremente su implementación sin afectar cualquier otra parte de la aplicación.

3.1.5.1 La Interface.

La declaración de una clase de interface comienza con la directiva del compilador **@interface** y termina con la directiva **@end**. (Todas las directivas que utilicemos dentro de Objective C comienzan con **"@"**.) por ejemplo:

```
@interface Nombre_de_la_Clase : Su_Superclase

{
        Declaraciones variables de instancia
}
        Declaraciones del método
@end
```

La primera línea de la declaración presenta el nombre de la nueva clase y la vincula con su superclase. La superclase define la posición de la nueva clase en la jerarquía de herencia.

En la declaración de la clase, las llaves juntan declaraciones de variables de instancia, las estructuras de datos que serán parte de cada instancia de la clase. Regresando a nuestro ejemplo de la Matriz de esta forma se declararían las variables de instancia:

```
id      selectedCell;
int     numRows;
```

```
int     numCols;
float   backgroundGray;
id      cellClass;
```

Los métodos para la clase se declaran posteriormente, después de las llaves que juntan variables de instancia y antes del final de la declaración de la clase. Los nombres de los métodos que pueden ser usados por la clase objeto, ***clase de métodos***, son precedidos por un signo más:

+ alloc;

Los métodos que las instancias de una clase pueden usar (los métodos de instancia), se marcan con un signo menos :

- display;

Un método puede tener el mismo nombre que una variable de instancia, esto se utiliza si el método devuelve el valor en la variable. Por ejemplo, la Matriz tiene un método llamado **selectedCell** para igualar su variable de instancia **selectedCell** .

El método devuelve los tipos que son declarados usando la sintaxis de C estándar:

- (int) tag;

Los tipos de argumento se declaran del mismo modo:

- setTag : (int) anInt;

Si un tipo de argumento no es declarado explícitamente, se presume que es del tipo más común; para métodos y mensajes: **id**, **alloc**, **display**, y **setTag:** los métodos ilustran el regreso de tipos de datos **id**.

Cuando hay más de un argumento, se declaran dentro del nombre del método después de los dos puntos. Los argumentos rompen el nombre en la declaración, como en un mensaje. Por ejemplo:

```
- moveTo:(NXCoord)x :(NXCoord)y;
- getRow:(int *)aRow andColumn:(int *)aColumn ofCell:aCell;
```

Los métodos que toman un número variable de argumentos se declaran usando una coma y tres puntos, como una función:

```
- makeGroup:agrupe, ...;
```

3.1.5.1.1 La importación del Interface.

El archivo de interface debe incluirse en cualquier módulo fuente que dependa de la clase interface incluyendo cualquier módulo que cree una instancia de la clase, envíe un mensaje para invocar un método declarado por la clase o mencionar una variable de instancia declarada en la clase. La interface se incluye comúnmente con la directiva de **#import** por ejemplo:

```
#import "Matriz.h"
```

Esta directiva es idéntica a #include, excepto que asegura que el mismo archivo no se incluya nunca más de una vez.

Para reflejar el hecho, la definición de una clase se construye sobre las definiciones de clases heredadas, un archivo de interface comienza importando la interface para su superclase por ejemplo:

```
#import "ItsSuperclass.h"

@interface Nombre_de _la_Clase : ItsSuperclass
{
    las declaraciones variables de instancia
}
    las declaraciones de método
@end
```

Estos medios de convención indican que cada archivo de interface incluye, indirectamente; la interface guardada para todas las clases heredadas. Cuando un módulo fuente importa una interface de clase, consigue interfaces para la jerarquía entera de herencia que la clase construye.

3.1.5.1.2 Referencia de Otras Clases.

Un archivo de interface declara una clase y al importar su superclase, implícitamente contiene declaraciones para todas las clases heredadas, desde la

clase Objeto mediante su superclase. Si la interface menciona clases que no están en su jerarquía, las declara con la directiva **Class** :

@Class Matriz, List;

Esta directiva simplemente informa al compilador que **"Matriz"** y **"List"** son de la clase nombres; y con ello no importa sus archivos de interface.

Una interface llama a la clase nombres cuando sus tipos de variables de instancia son escritas estáticamente, regresa valores, y argumentos. Por ejemplo, esta declaración menciona la clase lista:

```
- getCells:(List *)theCells;
```

La directiva **@Class** minimiza la cantidad de código compilado y es la manera más simple para dar una declaración de una clase de nombre. Siendo simple, evita problemas potenciales que pueden venir al importar archivos que importan todavía otros archivos. Por ejemplo, si una clase declara un variable de instancia escrita estáticamente de otra clase, y sus dos archivos de interface importan el uno al otro, ni la clase puede recopilar correctamente.

3.1.5.2 La Implementación.

La definición de una clase se estructura con su declaración. Comienza con la directiva **@impementation** y termina con la directiva **@end** por ejemplo:

```
@implementation ClassName : ItsSuperclass
{
    las declaraciones variables de instancia
}
    las definiciones de los métodos
@end
```

Sin embargo, cada archivo de implementación debe importar su propia interface. Por ejemplo, Matriz.m importa Matriz.h. Porque la implementación no necesita repetir ninguna de las declaraciones importadas, como:

1. El nombre de la superclase
2. Las declaraciones de las variables de instancia

Simplificando la implementación solamente es necesario hacer las definiciones de los métodos quedando como sigue:

```
#import "ClassName.h"

@ implementation ClassName
    las definiciones de los métodos
@ end
```

Los métodos para una clase se definen, como funciones de C , dentro de un par de llaves. Antes de las llaves, estos se declaran de la misma manera que en el archivo de interface, pero sin el punto y coma. Por ejemplo:

```
+ alloc
{
        . . .
}

- (int)tag
{
        . . .
}
- moveTo:(NXCoord)x :(NXCoord)y
{
        . . .
}
```

Los métodos que toman un número variable de argumentos se manejan como funciones:

```
#import <stdarg.h>

- getGroup:group, ...
    {
      va_list     ap;
      va_start (  ap, agrupar);
          . . .
    }
```

3.1.5.2.1 El Alcance de Variables de Instancia.

Las variables de instancia se declaran en la clase interface. La interface de un objeto esta unida a sus métodos, no a la estructura interna de sus datos.

Frecuentemente existe una correspondencia uno a uno entre los métodos y las variables de instancia, como se observa en el siguiente ejemplo.

```
- (int)tag
{
   return tag;
}
```

Pero algunos métodos pueden devolver la información no almacenada en variables de instancia, y algunas de ellas podrían almacenar información que un objeto no permite dar a conocer.

Como una clase se revisa de vez en cuando, la elección de las variables de instancia pueden cambiar, aunque los métodos declarados sean los mismos. Mientras los mensajes son el vehículo para interactuar con las instancias de clase, estos cambios no afectarán realmente su interface.

Para imponer la capacidad de un objeto para ocultar sus datos, el compilador limita el alcance de las variables de instancia a los límites su visibilidad dentro del programa. Pero para proveer flexibilidad, es posible utilizar tres niveles diferentes de alcance. Cada nivel es marcado por una directiva del compilador, los cuales se pueden observar en la tabla 3.1

En la figura 3.3 es posible observar gráficamente el alcance de las variables de instancia.

Directiva	Significado
@private	La variable de instancia es accesible únicamente dentro de la clase que la declaró
@protected	La variable de instancia es accesible dentro de la clase que la declara y dentro de las clases que la heredan.
@public	La variable de instancia es accesible en cualquier lugar.

Tabla 3.1. Alcance de las variables de instancia

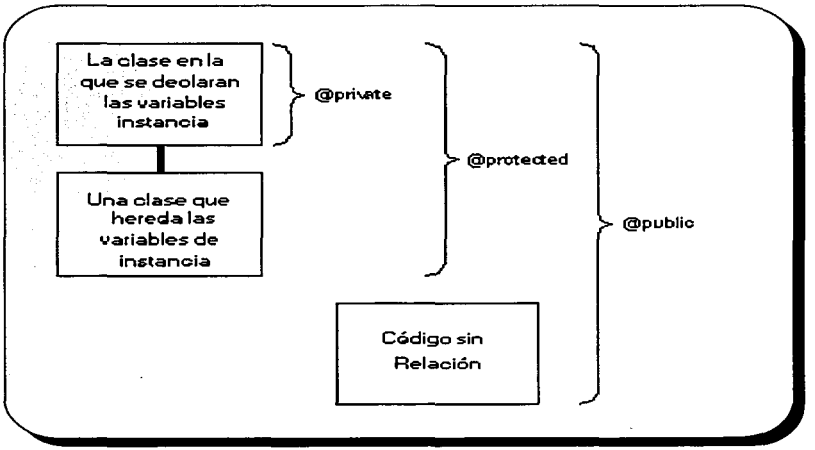


Figura 3.3. Alcance de las variables de Instancia

Una directiva se aplica a todas las variables de instancia que se encuentren después de ella, hasta la próxima directiva o el fin de la lista. En el ejemplo siguiente, las variables de instancia de **evaluación** y **edad** son privados, **nombre**, **trabajo**, son protegidos, y el jefe es público.

```
@interface Trabajador: Object
{
    char *nombre;
@private
    int edad;
    char *evaluacion;
@protected
    id trabajo;
@public
    id jefe;
}
```

Todas las variables de instancia que no sean marcadas son consideradas del tipo **@protected**, es decir, protegidas.

Para limitar el alcance de una variable de instancia simplemente se declara como **@private**. Sin embargo, una variable de instancia **public** puede accesarse donde quiera como si fuera un campo en una estructura de C, pero hay que evitarlo a menos que sea un caso muy excepcional, debido a que se encuentra en contra del principio de la programación orientada a objetos de la encapsulación de los datos, en la que los objetos son protegidos tanto a la vista como ante un error inadvertido.

3.2. ProjectBuilder

El ProjectBuilder es una herramienta para el desarrollo de aplicaciones en OpenStep, esta aplicación es la encargada de administrar los componentes de la aplicación a desarrollar y permite el acceso a otras herramientas de desarrollo.

El ProjectBuilder se encuentra en el Workspace Manager y su icono es el que se muestra en la figura 3.4.

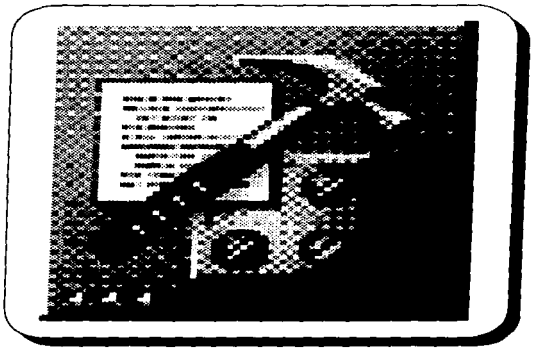


Figura 3.4 Icono del ProjectBuilder

Conceptualmente un proyecto comprende un número de componentes fuente y se destina para hacer una aplicación.

Físicamente, un proyecto es un directorio como el que se muestra en la figura 3.5; este contiene los archivos fuente y los controles de archivos del ProjectBuilder, llamados PB.project; Este archivo guarda los componentes del proyecto, y la aplicación antes de ser terminada.

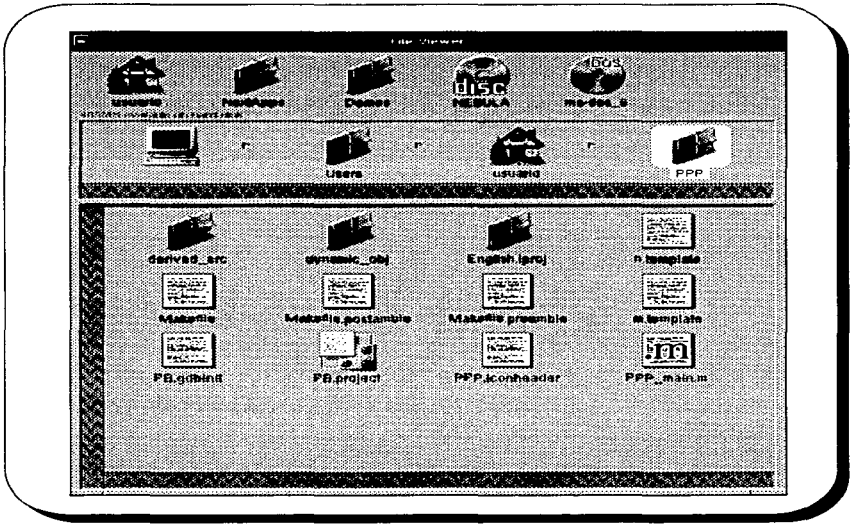


Figura 3.5 Directorio de una aplicación en ProjectBuilder.

Para que un archivo sea parte de un proyecto, debe encontrarse en el directorio de dicho proyecto y ser almacenado en el archivo PB.project del proyecto.

Al crear una aplicación no se puede editar directamente el PB.project, ya que esta labor es realizada dentro del ProjectBuilder de manera automática al modificar el proyecto.

El ProjectBuilder se utiliza para crear y modificar los proyectos, ya sean aplicaciones finales o subproyectos. Un subproyecto se debe de entender como un proyecto que se encuentra dentro de otro proyecto. Al realizar aplicaciones grandes, lo más adecuado es agrupar componentes en subproyectos; estos pueden construirse de forma independiente al proyecto principal.

Al construir un proyecto, el ProjectBuilder construye algunos subproyectos que necesita para realizar el proyecto final, estos archivos tienen la terminación ".o" y los coloca dentro del subdirectorio del proyecto principal o aplicación.

OpenStep permite realizar aplicaciones que pueden ser ejecutadas bajo cualquier arquitectura, ya sea para Motorola, Intel, NeXT, SUN, HP-RISC, etc.

El ProjectBuilder se apoya en el desarrollo de arquitecturas múltiples binarias, conocida como " fat " binaria; lo que hace posible elaborar una aplicación, que sea ejecutada bajo cualquier plataforma, esta aplicación contendrá los ejecutables binarios que proporciona cada plataforma; así cuando se elabora una aplicación el sistema es el encargado de checar que arquitectura y mensajes binarios debe de utilizar, para poder manejar las imágenes y sonidos adecuados para cada plataforma.

El ProjectBuilder, también prepara nuestras aplicaciones para diferentes idiomas, este proceso es llamado " localización". Por lo que su subdirectorio tiene

una terminación ".lproj" , por ejemplo, un proyecto en español, tendrá un subdirectorio Spanish.lproj.

Con la clase NXBundle, la aplicación puede cargar el idioma, dependiendo de los componentes que el usuario maneje.

3.2.1 Creación de Proyectos en ProjectBuilder

3.2.1.1 Crear un Nuevo Proyecto

Para crear un proyecto nuevo, se escoge en el menú del ProjectBuilder el comando New (Nuevo); con lo que aparece un panel como el que se muestra en la figura 3.6; en el cual se debe especificar la ubicación y el nombre del proyecto.

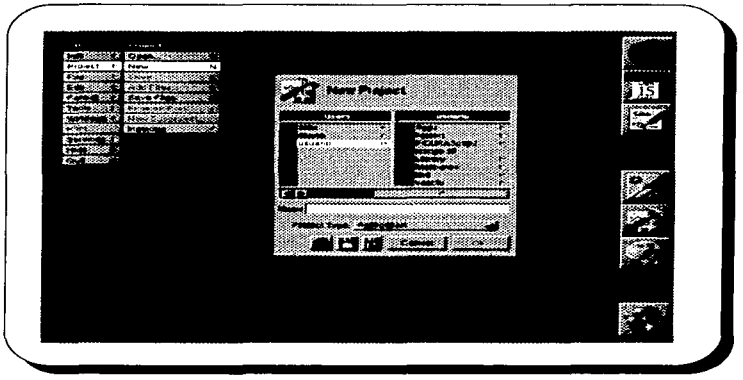


Figura 3.6 Inicio del ProjectBuilder

El sistema inmediatamente se encarga que el nuevo proyecto sea una aplicación autosuficiente; por lo que aparece una ventana para la creación del proyecto como la que se muestra en la figura 3.7, esta nos permite construir, modificar y encontrar errores en el proyecto.

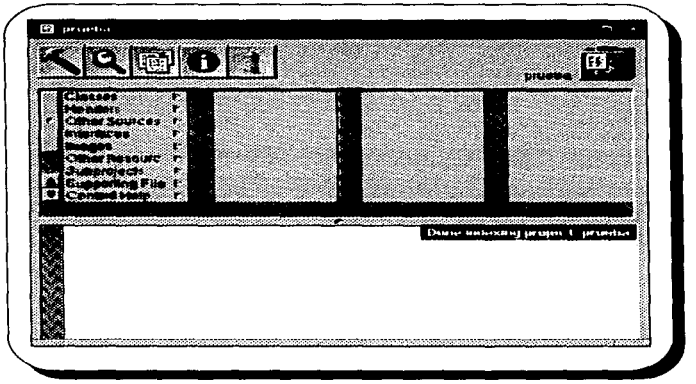


Figura 3.7 Ventana para la creación de proyectos

Existen cinco modos de operación para la elaboración del proyecto, los cuales se encuentran en la parte superior izquierda del panel de la ventana, los cuales son:

1. **Constructor.-** Se utiliza una vez que se terminó el proyecto para compilar.

2. **Localizador.-** Se utiliza para buscar algún archivo en particular.

3. **Archivos.-** Sirve para adicionar, quitar o abrir archivos de proyectos .

4. **Inspector de atributos.-** Sirve para ver los atributos del proyecto como el directorio de origen, el nombre de la interface gráfica, su tamaño, etc.

5. **Depurador.-** Sirve para depurar los programas del proyecto.

En general los modos de aplicación, incluyen los campos para especificar el nombre de proyecto, el idioma primario (que es, el idioma en que el proyecto se esta desarrollado), el directorio destino, los campos para especificar la clase de aplicación y la aplicación principal, una opción que evita volver a generar el archivo principal y la opción de incluir un icono a la aplicación.

3.2.1.2 Visualizador del ProjectBuilder

En el visualizador del ProjectBuilder podemos:

1. Localizar proyectos y observar los archivos que contiene.
2. Aumentar archivos.
3. Remover archivos.
4. Abrir un archivo de proyecto como si fuera una aplicación, es decir por medio de un doble-click sobre su nombre o icono.

Existen 2 formas para agregar archivos a un proyecto:

1. Arrastrar el archivo desde el File Viewer dentro de la ventana de proyectos; al arrastrarlo se ve en el icono de la maleta como esta se abriera. Las clases que adicionamos a la maleta toman la terminación ".m", mientras las cabeceras toman la terminación ".h" .

2. Se selecciona una maleta y se escoge en el menú el comando de agregar (add), con lo que aparece un nuevo panel de información, en el cual se especifica el archivo que se va a agregar.

El directorio de nuestro proyecto nos proporciona con el ProjectBuilder una forma sencilla de organizar los archivos junto a su aplicación; este sistema es conocido como Gerente de Proyecto y se clasifica en un número de categorías.

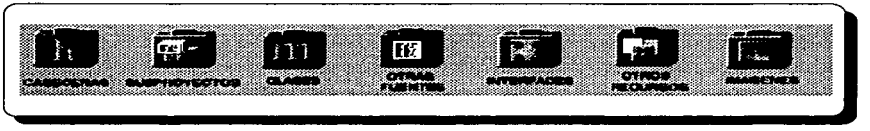


Figura 3.8 Categorías del ProjectBuilder

Estas categorías se representan con un icono en forma de maleta como podemos observar en la figura 3.8 estas categorías son:

1. **Clases**, contiene el código de las clases más usadas por una aplicación.

2. **Cabeceras,** contiene declaraciones de métodos y las funciones usadas por una aplicación

3. **Otras fuentes,** contiene el código auxiliar para una aplicación. Incluyendo archivos ".m" codificados en Objective C; los archivos ".c" codificados en C

ANSI, así como los archivos ".psw" que son codificados en código PostScript. El ProjectBuilder automáticamente agrega estos archivos a el archivo ApplicationName_main.m

4. **Interfaces,** agrega los archivos Nib (los archivos .nib son los archivos de la interface gráfica, si se le da un doble-click automáticamente se activa la aplicación de InterfaceBuilder) a una aplicación.

5. **Imágenes,** agrega las imágenes usadas por una aplicación, incluyendo archivos ".TIFF" y ".EPS" .

6. **Otros Recursos,** incluye los archivos de sonido que se agregan a la aplicación.

7. **Subproyectos,** indica los subdirectorios que contienen los subproyectos usados por la aplicación.

8. **Archivos de soporte,** agrega otro tipo de archivos que pueden ser utilizados para la optimización del proyecto.

9. **Archivos de ayuda,** incluye archivos para facilitar el uso de la aplicación.

3.2.1.3 Compilación de un Proyecto.

Cuando se termina un proyecto se le indica al ProjectBuilder que la aplicación ya se terminó, dando un click en la opción de compilar (buil) que esta representada por el icono de la figura 3.9.

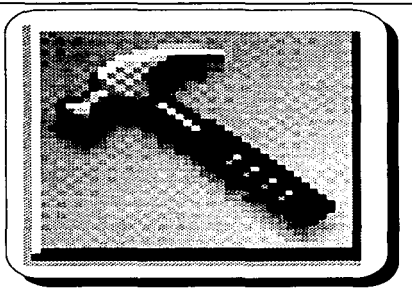


Figura 3.9 Icono para compilar un proyecto.

Al activar la opción de compilar aparece una ventana como la que se muestra en la figura 3.10, la cual tiene 3 botones en la parte superior, que sirven para ver las opciones de compilación, limpiar la memoria y compilar el proyecto.

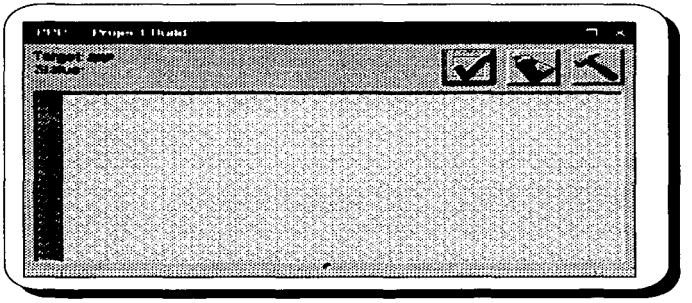


Figura 3.10 Ventana de compilación

Cuando se da click al primer botón (en forma de una paloma), se activan las opciones de compilación con lo que aparece otra ventana como la que se muestra

en la figura 3.11, en la cual se indica para que tipo de arquitectura hay que compilar la aplicación, si se quiere hacer una aplicación, un .profile, etc.

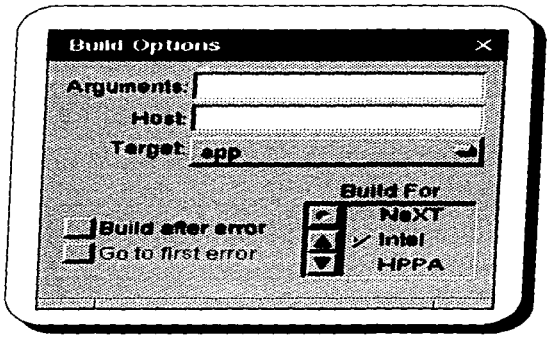


Figura 3.11 Opciones de compilación

Antes de compilar siempre es recomendable limpiar la memoria, lo cual se puede hacer dando un click en el botón con la imagen de una escoba.

Finalmente se hace la compilación del archivo, dando un click en el botón en forma de martillo.

Al compilar el código fuente del proyecto, este se relaciona con un archivo ejecutable; lo que se conoce como makefile y nos provee de información respecto a las necesidades para trabajar la aplicación.

Existen dos ventanas que nos informan los errores o advertencias encontradas durante el periodo de compilación, la parte de arriba, o sea la primera

ventana muestra en forma resumida los errores y su colocación, mientras que la parte inferior, es decir, la segunda ventana detalla un poco mas las características de los errores, así como su colocación en el código y visualización del error, es decir, podemos ver en que nos equivocamos (un punto y coma, en escribir mal una palabra, etc).

El compilador nos indica con una línea roja la ubicación del error y por medio de hipertexto podemos trasladarnos a ese lugar y editar nuestro error. Podemos posteriormente volver a compilar y en caso de no tener errores nuestra aplicación ya puede ser ejecutada.

3.3 InterfaceBuilder

El interfaceBuilder es una herramienta que OpenStep utiliza para crear las interfaces gráficas de una aplicación y hacer la conexión entre los objetos de la misma.

El InterfaceBuilder se encuentra en el Workspace Manager y su icono es el que se muestra en la figura 3.12.

Hay dos formas de activar está aplicación:

1. Cuando se hace un proyecto se crea de manera automática un nuevo .nib, y se agrega a la maleta de interfaces del proyecto como podemos ver en la figura 3.13; este archivo tiene el mismo nombre de su aplicación proyecto, solo con la diferencia de que contiene la terminación .nib cuando se abre este archivo se activa el InterfaceBuilder.

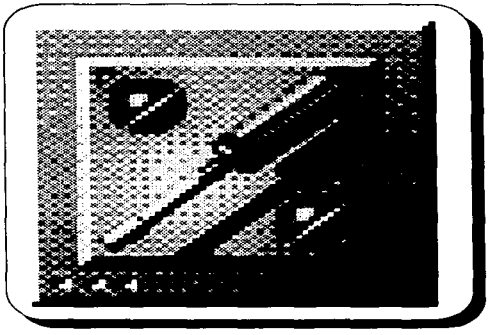


Figura 3.12 Icono del InterfaceBuilder

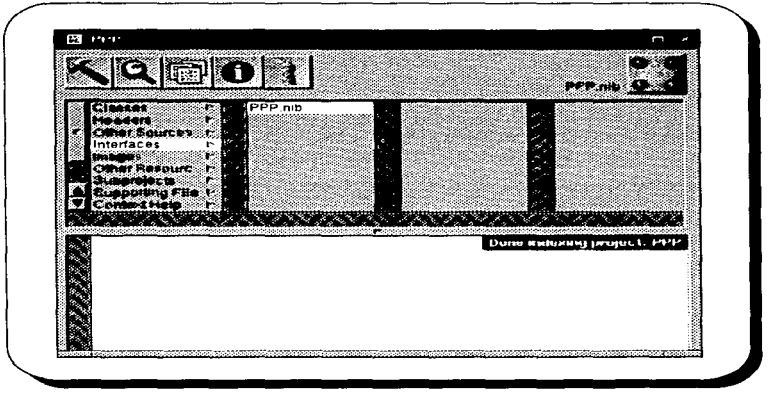


Figura 3.13 Archivo .nib en el ProjectBuilder

2 Cuando se le da un doble-clik en su icono.

En ambos casos aparece una pantalla como la que se muestra en la figura 3.14 que tiene varios elementos como la ventana que contiene las paletas, la de los archivos, un menú y la ventana de aplicación.

La ventana de archivos nib (que se indica en la figura 3.15); contiene múltiples vistas de los contenidos del archivo nib, los cuales podemos seleccionar mediante las lengüetas de folder; estas vistas muestran objetos archivados, las conexiones entre objetos, la jerarquía de la clase que utilizamos, las imágenes y sonidos almacenados en el archivo.

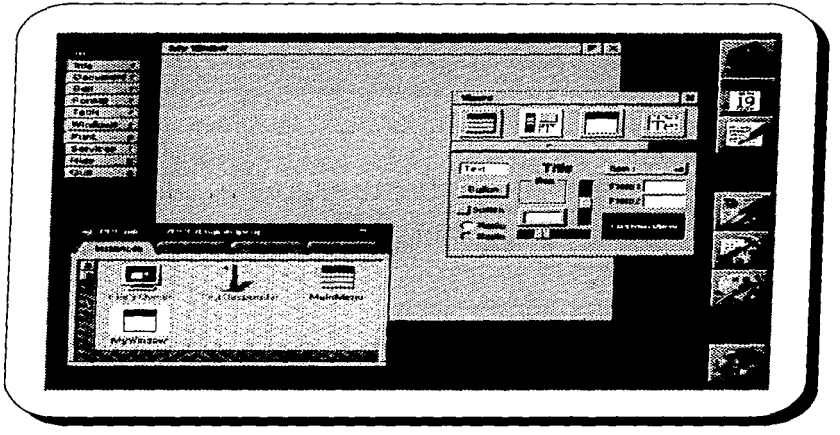


Figura 3.14 Pantalla de inicio del InterfaceBuilder

Cada aplicación tiene un archivo nib, que contiene el menú principal con el que va a trabajar y los objetos que forman dicha interface. Un archivo nib contiene:

1. Los objetos almacenados, así como su jerarquía
2. Sonidos e Imágenes
3. Información acerca de las clases que utilizamos
4. Información acerca de cómo y con quién van a ser conectados los objetos

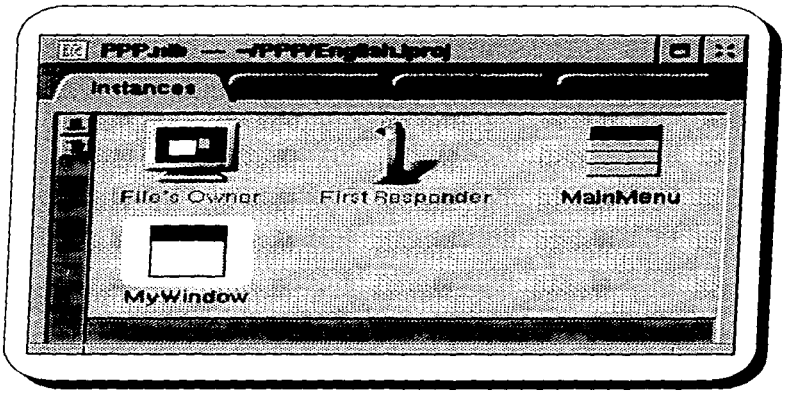


Figura 3.15 Ventana de archivos.

El archivo nib almacena información codificada sobre el conjunto de objetos, incluyendo su tamaño, ubicación, y su jerarquía; la jerarquía mas alta es el objeto del archivo propietario, el sistema le indica quién es el propietario del archivo nib.

Los sonidos e imágenes se representan por medio de archivos, los cuales son almacenados en el archivo nib principal.

El InterfaceBuilder almacena los detalles del conjunto de objetos, pero no sabe como archivar ejemplos de las clases, esto lo maneja el sistema por seguridad de no perder valiosa información, así que el InterfaceBuilder solamente almacena información adjunta que le permite saber en que lugar se encuentran las clases y como llamarlas cuando las necesite.

Un archivo nib contiene también información sobre como los objetos se interconectan. El conector objeto especial que tiene el InterfaceBuilder almacena esta información, así que cuando guardamos el Documento, el conector de los objetos se guarda en el archivo nib conjuntamente con los objetos que interconecto.

3.3.1 Creación de un archivo Nib

Los archivos nib se crean automáticamente cuando se hace una aplicación en el ProjectBuilder, pero a veces es necesario crearlos directamente en el InterfaceBuilder, cuando se quiere agregar paneles y ventanas adicionales a la aplicación.

El interfaceBuilder puede crear archivos de forma independiente y para ello cuenta con 2 opciones:

1. Crear una nueva aplicación .nib diferente a la que nos ofrece el ProjectBuilder.

2. Crear un nuevo módulo.

Cuando seleccionamos un nuevo módulo se nos ofrece la oportunidad de escoger en el submenú las siguientes opciones:

1. Una aplicación vacía.
2. Un panel de información.
3. Un panel de arhivos.
4. Un inspector
5. Una paleta.

Estos nuevos archivos nib contienen un panel especial para trabajar de manera inmediata; las aplicaciones pueden llamar a estos archivos nib en el momento en que se necesiten.

El comando vacío (New Empty) crea un archivo nib en el cual se pueden crear ventanas y paneles arrastrando estos objetos desde la paleta de herramientas.

Para salvar un archivo se indica en el menú de Documento la opción guardar (Save), con lo que aparece una ventana de información como la de figura 3.16, donde hay que indicar el nombre del archivo nib y su localización.

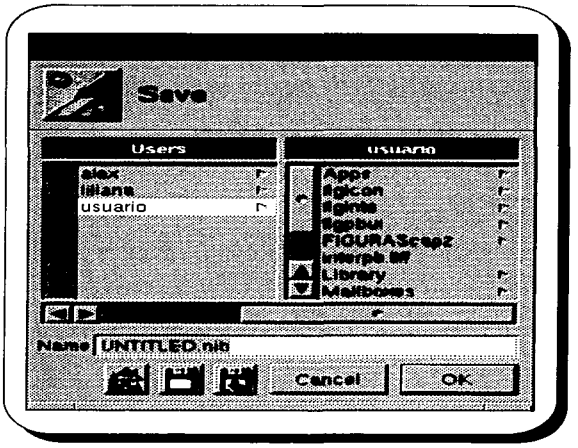


Figura 3.16 Panel para guardar archivos nib

3.3.2 Manejo de paletas

En la ventana de paletas se encuentran todos los objetos necesarios para hacer la interface gráfica de una aplicación, las distintas opciones que nos proporciona el sistema. están representadas por los iconos que se ven en la figura 3.17.

Para poder utilizar los objetos de las paletas solamente hay que señalarlos y arrastrarlos hacia la ventana de trabajo.

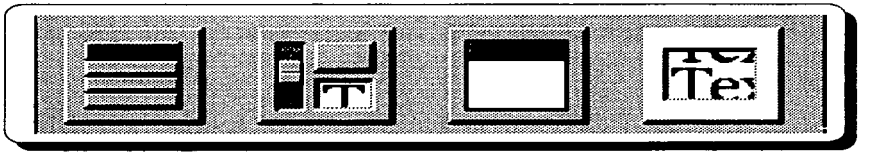


Figura 3.17 Iconos de las paletas

Así tenemos un conjunto de paletas como el que se observa en la figura 3.18, dentro de las cuales encontramos ventanas, paneles, localizadores, inspectores de listas, botones, campos de texto, etc. que podemos utilizar de manera inmediata en la interface; pero si es necesario podemos crear nuestras propias paletas o en su defecto modificar las que nos proporciona el sistema.

Para colocar objetos en la ventana de aplicación hay que seleccionar el objeto de las paletas, arrastrarlo y soltarlo en la ventana en el lugar que se desee.

Los objetos dentro de la ventana de aplicación pueden ser movidos a otra posición o modificar su tamaño ya que al seleccionarlos aparecen unos cuadros guía como los de la figura 3.19 que nos permiten " jalar " desde esa posición hasta obtener el tamaño requerido.

En caso de querer colocar un objeto en una posición y con un tamaño específico lo podemos hacer mediante el Inspector del objeto que se indica en la figura 3.20.

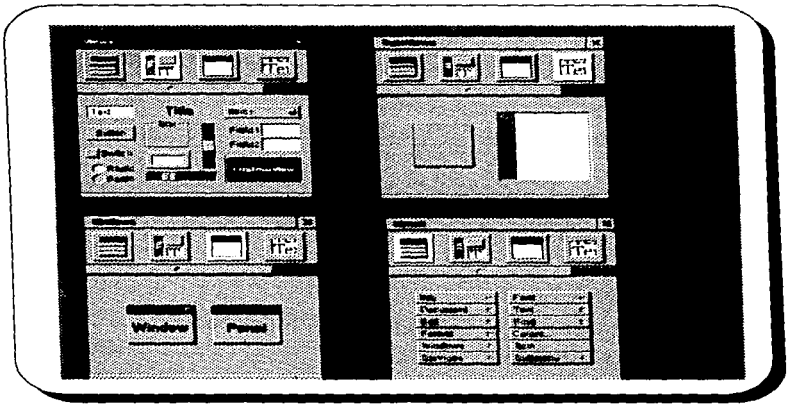


Figura 3.18 Contenido de las paletas del InterfaceBuilder

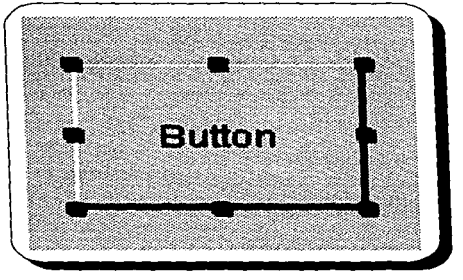


Figura 3.19 Botón marcado con los cuadros guía

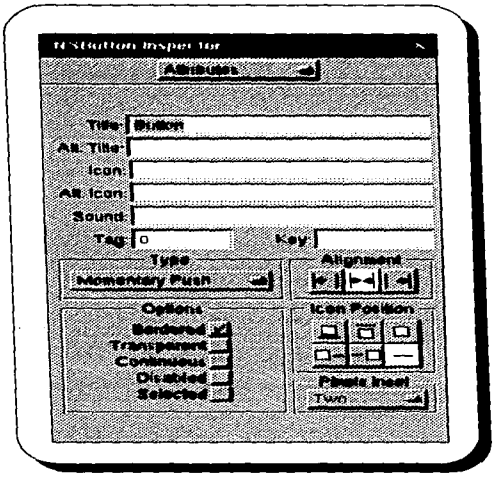


Figura 3.20 Inspector de objetos.

También es posible realizar una selección múltiple de objetos, señalándolos todos y oprimiendo la tecla alt al mismo tiempo; para separarlos, solamente es necesario dar un click en una zona que no contenga a ninguno de los objetos seleccionados.

3.3.3 Tamaño de ventanas y paneles

A las ventanas y paneles se les puede modificar su tamaño mediante una barra de ajuste arrastrando el ratón hacia la izquierda, derecha o verticalmente.

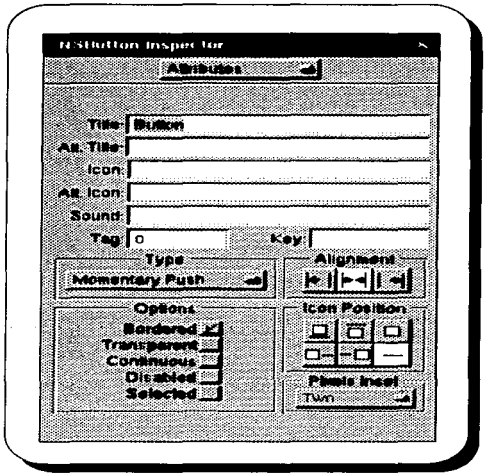


Figura 3.20 Inspector de objetos.

También es posible realizar una selección múltiple de objetos, señalándolos todos y oprimiendo la tecla alt al mismo tiempo; para separarlos, solamente es necesario dar un click en una zona que no contenga a ninguno de los objetos seleccionados.

3.3.3 Tamaño de ventanas y paneles

A las ventanas y paneles se les puede modificar su tamaño mediante una barra de ajuste arrastrando el ratón hacia la izquierda, derecha o verticalmente.

Algunas ventanas no pueden ser ajustadas por medio de la barra de ajuste, porque no cuentan con ella, pero se puede modificar su tamaño y atributos mediante el inspector de la ventana, o haciendo temporalmente visible la barra con la herramienta de ajuste de tamaño, representada por un botón en forma de triángulo en el ángulo superior derecho.

El inspector de ventana de la figura 3.21 nos ofrece la opción de dar un tamaño específico e indicar de que lado la ventana o panel no es posible disminuirlo, para poder controlar la apariencia y su tamaño.

3.3.4 Copia y alineación de los objetos

Cuando se utilizan 2 objetos con una precisión determinada y características similares se puede modificar un solo objeto y posteriormente copiarlo, seleccionando el objeto que se va a copiar, y posteriormente pegarlo en la posición requerida.

Para hacer la copia se escoge la opción de copiar (Copy) en el menú de Edición (Edit), y posteriormente la opción pegar (Paste).

Si se quieren alinear los objetos dentro de la interface, se puede hacer utilizando la rejilla del panel de alineación.

Al desarrollar un sistema es necesario darle una apariencia profesional, procurando que los botones, rejillas y paneles de control se vean alineados y siguiendo unas líneas invisibles tanto vertical como horizontalmente.

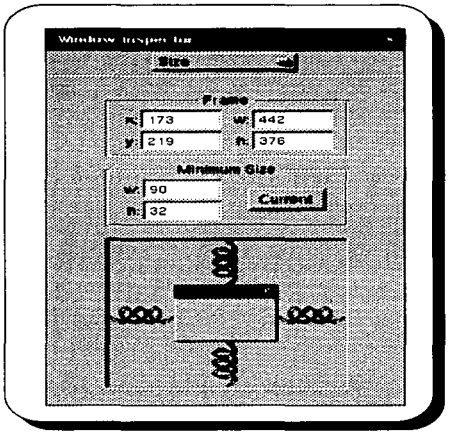


Figura 3.21 Inspector de ventana.

Cada ventana o panel tiene una rejilla asociada, las dimensiones pueden ser modificadas desde el panel de alineación del InterfaceBuilder. Este se activa con la opción de Alineación (Alignment) la cual se encuentra en el submenú de Alineación (Align) dentro del menú de Formato.

En ocasiones es necesario colocar varios objetos alineados en relación unos con otros, que tengan el mismo nivel jerárquico gráficamente o en forma de columna; se necesita marcar primero todos los objetos como grupo; posteriormente en el menú de Formato en el submenu de Alineación se indica si los objetos se quieren en forma de renglón o de columna como se ve en la figura 3.22.

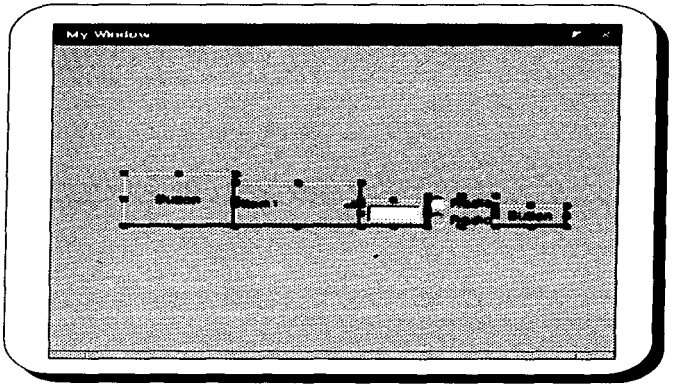


Figura 3.22 Objetos alineados horizontalmente

3.3.5 Creación de Menús

Se pueden crear menús personalizados para las aplicaciones como el que se indica en la figura 3.23, en los cuales se colocan los comandos o listas que consideremos necesarios para manejar en la aplicación.

Primero debemos entrar a la paleta de menú y arrastrar el submenú que consideremos que es útil. Es importante recordar que estamos trabajando con objetos y que al arrastrar un submenú y pegarlo se unen además todas sus aplicaciones a las que está vinculado, a estas partes del menú se les conoce como celdas de menú y se les puede editar su contenido; es decir, en caso de no necesitar el panel de fuentes de letras, no se agrega, pero en caso de necesitar el

mismo panel un submenú es factible colocárselo; además se puede cambiar el nombre a una celda del menú de acuerdo a las necesidades.

Figura 3.23 Menú personalizado

3.3.6 PERSONALIZAR VENTANAS Y PANELES

Las ventanas y paneles se pueden personalizar utilizando el Inspector de atributos, ya que este nos muestra varias opciones, que podemos utilizar dependiendo de lo que se necesite en esa ventana.

Existen 3 opciones de almacenaje como podemos ver en la figura 3.24.

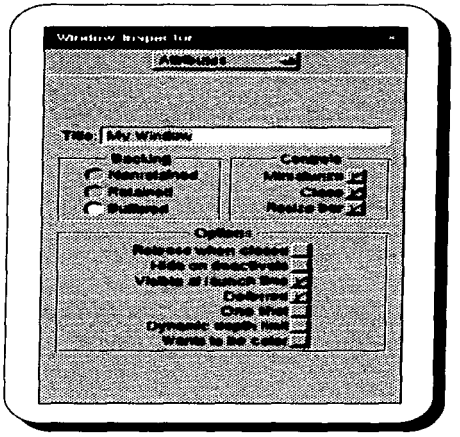


Figura 3.24 Inspctor de ventana

1. **No mantener (Nonretained)**
   Esta opción se responsabiliza de todos los dibujos que se encuentran en la pantalla, y se usa para aplicaciones con imágenes fijas.

2. **Retener (Retained)**
   Esta opción se utiliza cuando se trabaja con muchas ventanas auxiliares y se necesita que la actual sea almacenada por unos instantes mientras no se ocupa.

3. **Memoria temporal (Buffered)**
   Esta opción es usada cuando se necesita cierta memoria en la ventana, se ocupa en el caso de crear animaciones dentro de una ventana.

Hay otras opciones de control como las que se observan en la tabla 3.2.

Opción	Características
Released when closed	Cuando se cierra la ventana el sistema la libera y recupera la memori ocupada por ésta
Hide on desactivate	Cuando se desactiva la ventana esta desaparece automáticamente
Visible at launch time	La ventana aparece cuando se le llama, en el tiempo de inicialización
Deferred	La ventana es delegada, es decir, no se toma en cuenta hasta que est colocada en la pantalla
One shot	La ventana puede ser cerrada por el usuario
Dynamic depth limit	La ventana puede cambiar su tamaño dependiendo de las necesidades características de la pantalla
Wants to be color	La ventana puede presentarse de dos colores, ya sea gris claro o gris oscuro.

Tabla 3.2 Opciones del inspector para el control de la ventana

Es importante saber diferenciar un panel de una ventana, ya que el sistema los maneja de manera diferente, un panel es una ventana que sirve como una función de ayuda dentro de una aplicación, y esta destinado a jugar un papel importante en la elaboración de programas pero en un nivel menor a las ventanas, un panel tiene estos aspectos:

1. Puede tener una ventana auxiliar, pero nunca la ventana principal.

2. Cuando la aplicación se desactiva, el panel también lo hace (no se ven en la pantalla); pero al reactivar la aplicación, el panel aparece nuevamente.

3. Cuando un panel se cierra, éste desaparece de la pantalla.

4. Un panel tiene menos controles que una ventana

3.3.7 Atributos de los botones

Los botones al igual que todos los objetos que se manejan tiene sus propios atributos, los cuales se pueden modificar gracias al inspector de atributos del botón que podemos ver en la figura 3.25, esté nos permite modificar el botón, cambiar el titulo, su icono, etc.

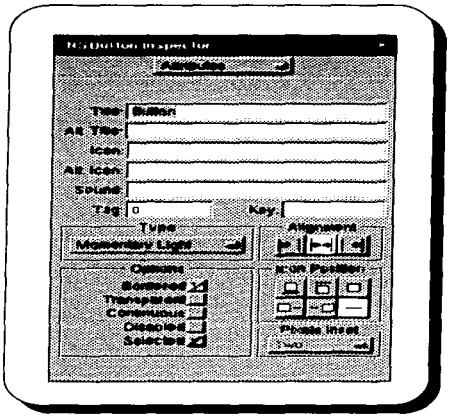


Figura 3.25 Inspector del Botón.

Las opciones con las que cuenta el panel de inspección del botón son :

1. Los títulos e iconos, a estos les podemos indicar el nombre que consideremos mas representativo de la idea que queremos plasmar.

2. El icono con el cual va a ser posible accionarlo.

3. La opción de clave equivalente (Key equivalent), nos da la oportunidad de crear un sistema de " teclas calientes", (instrucciones que pueden realizarse desde el teclado oprimiendo algunas teclas).

Los botones pueden comportarse de distinta manera dependiendo de las necesidades que se tengan, ya sea que se enciendan al ser oprimidos o simplemente que indiquen por un momento que fueron oprimidos; estos pueden tener borde o una colocación especial para el texto y nuestro icono.

La importancia de manejar un sistema operativo orientado a objetos, es que todos sus componentes se manejan de esa forma, así que si deseamos agregar un sonido a un botón simplemente lo arrastramos y lo pegamos desde la ventana de archivos nib sobre un botón; lo mismo sucede con una imagen , ya que sólo hay que arrastrarla y pegarla en el botón como se puede ver en la figura 3.26.

Las imágenes aparecen en botones con o sin el texto y existen algunos iconos que proporciona el sistema que juegan un papel importante puesto que pueden funcionar de manera directa, como si fueran un enter.

Es importante siempre que se utilicen archivos de sonido y de imágenes agregarlos a los objetos y a nuestro archivo nib debido a que sino lo hacemos cuando compilemos nuestra aplicación se pueden perder debido a que el sistema lo busca en el directorio del usuario.

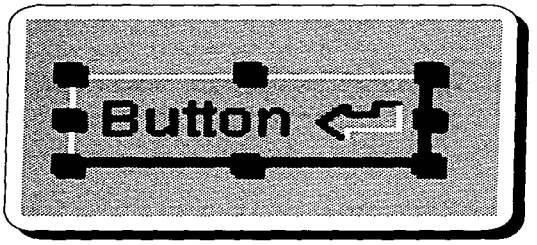


Figura 3.26 Botón con una imagen asociada

Los sonidos y las imágenes tienen sus propios atributos como podemos ven las figuras 3.27 y 3.28.

3.3.8 Comunicación entre los objetos: las salidas y acciones

1. Las Salidas

Una salida es un variable de instancia que apunta a otro objeto. Los objetos usan estas salidas para comunicarse con otros objetos; ellos envían mensajes al otro objeto identificado por la salida. El InterfaceBuilder nos permite declarar y colocar salidas a los objetos que utilicemos en nuestra aplicación. El Appkit define dos tipos de salidas que podemos utilizar para establecer conexiones especializadas con otros objetos: Otorgar responsabilidades (Delegate) y Destinos (Target).

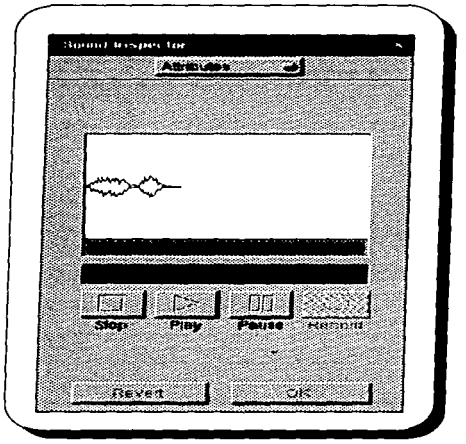


Figura 3.27 Inspector de sonidos

Otorgar responsabilidades (Delegate)

Un objeto delegado actúa en nombre de otro objeto. Existen muchas clases ya predefinidas en el sistema que utilizan esta clase de salida. El conjunto de objetos envía mensajes a sus objetos delegados, dando la oportunidad de que estos participen en el proceso o detengan algún proceso. El ejemplo mas representativo de los objetos delegados es el localizador (browser) el cual llama a sus objetos delegados para abastecer las celdas de búsqueda en la columna; las aplicaciones informan a sus objetos delegados cuando es inicializada se oculta o activa la aplicación.

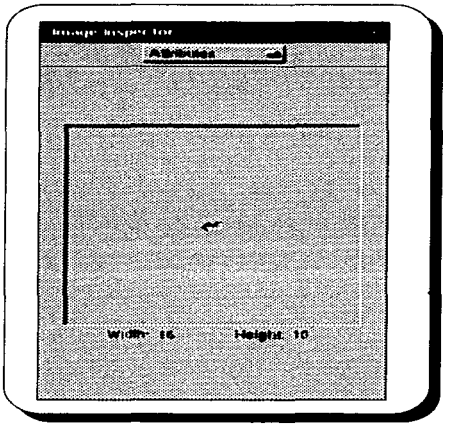


Figura 3.28 Inspector de imágenes

Los Destinos (Targets).

Los destinos son un tipo especial de salida, la cual identifica los objetos que pueden responder a los mensajes de acción. Cuando un usuario activa el control de un objeto ya sea por ejemplo oprimiendo un botón , el objeto envía un mensaje de acción al destino. El mensaje de acción especifica el significado de oprimir con el ratón algún objeto o la acción de una tecla especial.

Como un objeto delegado, cada destino debe implementar métodos para responder a los mensajes que se le envían. Pero la diferencia consiste en que

recibe mensajes limitados por un conjunto predefinido por el sistema, mientras que el destino responde a los mensajes de acción definidos por el programador.

2.- Las Acciones

La acciones consisten en que los objetos de control traduzcan los mensajes de los eventos que se produzcan, estos son recibidos cuando los usuarios manipulan los mensajes dentro de la aplicación, estos envían mensajes a otros objetos con la finalidad de unir acciones.

A los mensajes específicos a la aplicación que son iniciados por un objeto de Control se llaman mensajes de acción, y al método que utilizan se le conoce como método de acción.

Un objeto de Control es simplemente una interface con el usuario que permite al mismo dar instrucciones a la aplicación, este dispositivo se encuentra entre el usuario y la aplicación y sirve de control de respuesta a los eventos que el usuario indique.

Los métodos de acción toman un argumento único, el id del objeto de Control (puntero que indica que es un objeto) que envía el mensaje. Este argumento permite el receptor pedir al control más información si se necesita.

Un Objeto de control puede enviar un mensaje de una acción a destinos diferentes debido a que contiene los controles que distribuyen los mensajes de acción de manera diferente; para ilustrar lo anterior podemos observar que un Botón generalmente envía mensajes de acción sobre un click de ratón, es decir,

que indique la iniciación de un suceso pero un localizador de celdas (browser) envía mensajes de acción continuamente, mientras el botón del ratón se encuentra oprimido.

3.3.9 Conexión de objetos

En una aplicación orientada a objetos, es necesario unir los objetos, por que si se encuentran aislados no ofrecen muchas ventajas; ellos necesitan enviar mensajes de un objeto a otro para ejecutar las instrucciones que forman la aplicación.

El InterfaceBuilder ofrece una manera simple de unir a los objetos; primero se necesita hacer una conexión entre el InterfaceBuilder con el control de la interface, esto se observa gráficamente por medio de una línea que une a un objeto con otro como se ve en la figura 3.29.

Cada vez que se unen los objetos se observa su respectiva línea de unión; casi siempre se realizan entre nuestro menú de variables de instancia y los objetos que se encuentran representados en la interface, la línea se forma seleccionando un objeto (dando un click con el ratón sobre el objeto) y oprimiendo la tecla Control y con el ratón desplazarnos hasta el siguiente objeto.

La línea se observa, hasta que dejamos de oprimir el ratón, automáticamente se abre el panel de inspección que controla el manejo de datos de un objeto a otro; sus conexiones muestran los caminos potenciales de los datos para el objeto de destino.

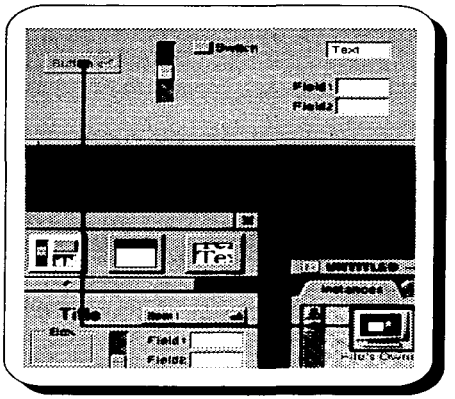


Figura 3.29 Conexión de objetos.

Existen dos clases de conexiones que se observan en el panel de inspección de la figura 3.30, las conexiones de salida y las conexiones de acción.

1. Las Conexiones de Salida

Al realizar las conexiones entre los objetos, en el panel de inspección se observan dos columnas, la primera de las cuales nos indica las posibles salidas mientras que la segunda las acciones que se pueden realizar.

2. Las Conexiones de Acción

Al tener la conexión entre los objetos, ya sea botón, buscador, menú, etc, los objetos seleccionados se convierten en destinos permitiéndonos completar la conexión por medio de los métodos de acción.

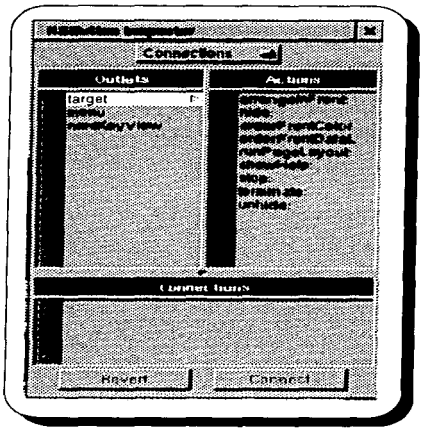


Figura 3.30 Inspector de la conexión de objetos.

El objeto destino en una conexión de acción es frecuentemente un objeto que administra la aplicación o una ventana particular (el objeto de control).

Cuando hacemos una conexión desde un objeto de Control, el panel de Inspector muestra las Conexiones que son posibles para el objeto destino.

Cuando el usuario manipula el objeto de Control, ya sea oprimiendo un botón o mediante la barra de desplazamiento, el mensaje de acción se envía al objeto (destino).

Existen varias modalidades para ver las variables de instancia:

1. La modalidad de Icono

Al abrir un archivo nib en el InterfaceBuilder las variables de instancia del archivo nib permiten observar a los objetos como iconos. Esta modalidad no muestra todos los objetos, simplemente los de alto nivel , es decir, objetos que no son contenidos por otros objetos.

Las ventanas, paneles y la mayoría de los controladores de objetos que son los que administran una aplicación son los objetos de alto nivel; aunque ellos puedan contener otros objetos (por ejemplo, una ventana contiene uno o mas vistas), ningún otro objeto los contiene.

La representación gráfica de objetos en el modo de icono como se puede ver en la figura 3.31, hace a la interfase ideal para muchas operaciones, debido a que es simple, intuitiva, y permite hacer fácil las cosas, tales como conexiones de elaboración entre objetos de alto nivel y objetos de interface.

2. La modalidad de línea de salida.

La ventaja más importante de esta modalidad consiste en que muestra todos las instancias de todos los objetos que se encuentran en el archivo nib, no solamente los objetos de alto nivel; muestra además todas las conexiones; las existentes en un objeto y las conexiones desde un objeto a otros objetos.

La modalidad de línea de salida como se indica en la figura 3.32, enumera los objetos de alto nivel en el archivo nib, al oprimir el botón del ratón sobre esa instancia abre otras instancias, en donde podemos ver los objetos que

contiene; al oprimir nuevamente una conexión se crea un botón en forma de triángulo para observar las conexiones que van dentro o fuera de un objeto.

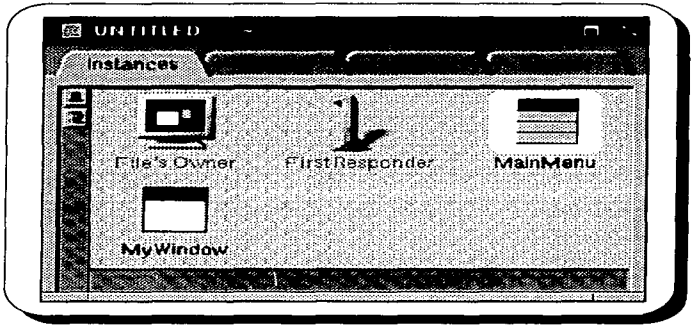


Figura 3.31 Modo de icono

El InterfaceBuilder nos permite examinar las conexiones entre objetos; por ejemplo, observar que salidas y que acciones podrían asociarse con un objeto en el interface o en las instancias en las dos modalidades (icono y línea de salida) y localizar en el panel de Inspector que objeto se conecta a otro.

Las conexiones es posible verlas de una en una en la modalidad de icono, mientras que en la modalidad línea de salida (también conocida como resúmen) las instancias muestran todas las conexiones que un objeto tiene, desde él a otros objetos.

Cuando oprimimos el triángulo en la modalidad de resúmen, las líneas que aparecen muestran las conexiones entre objetos. El nombre y la clase de

cada objeto conectado se destaca en negritas. Cada conexión se etiqueta con el nombre de una salida o la acción.

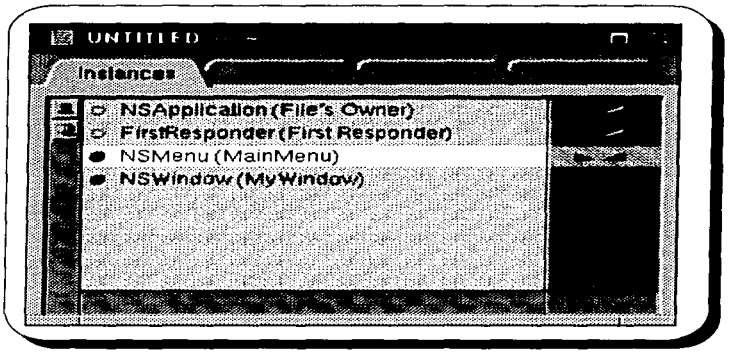


Figura 3.32 Modo de Línea o Resúmen

En los casos especiales en que un objeto tenga conexiones múltiples con otro objeto, ya sea dentro o por fuera ambas salidas y acciones; la modalidad de resúmen nos permite tener un control entre ambas conexiones.

Los objetos en el panel de instancias juegan un papel importante por lo que describiremos 3 de ellos.

1. Propietario del Archivo (File´s Owner).

Cada archivo nib tiene un propietario, representado por el icono de Propietario del Archivo (figura 3.33). El propietario es un objeto externo al archivo

nib, que canaliza mensajes entre los objetos sin archivar desde el archivo nib y los otros objetos en su aplicación.

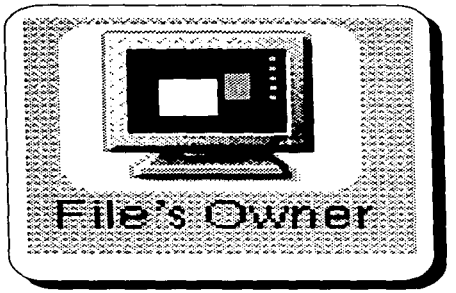


Figura 3.33 Propietario de archivo

El objeto propietario no solamente es externo a su archivo nib, además no debe de estar archivado, esto se debe a que el mensaje de la clase que controla las aplicaciones (conocida como NXApp que llama al archivo nib con la instrucción loadNibSection: el propietario: y sus variantes) también especifica el propietario del archivo. El propietario del archivo es un auxiliar del archivo nib.

2. Archivo de Respuesta Inmediata (First Responder)

El archivo de respuesta inmediata (figura 3.34) es el objeto dentro de una ventana que primero recibe los eventos del teclado, ratón, y mensajes de acción desde los objetos de Control que no tienen un destino especifico (por ejemplo, la acción de cortar y pegar). Este archivo es el encargado de manejar los datos en una ventana activa para futuros sucesos.

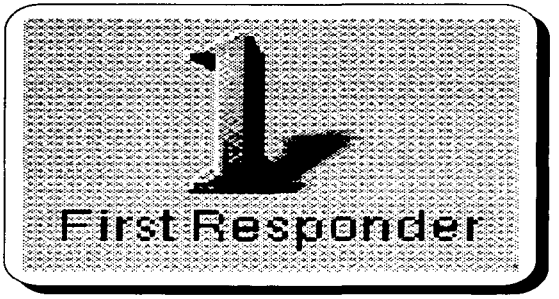


Figura 3.34 Archivo de respuesta inmediata

Podemos ejemplificar su funcionamiento mencionando que por ejemplo, nosotros escribimos en un campo de texto el objeto llega a ser elemento de un archivo de respuesta inmediata, debido a que en ahí es donde se controlan los primeros datos.

La condición cambia cuando estamos utilizando otra ventana, por eso es importante saber controlarla debido a que es muy útil al hacer aplicaciones de múltiples documentos.

3. Controlador de Fuentes de letras (FontManager)

El Control de fuentes de letras es representado por el icono de la figura 3.35, este controla a las fuentes de letras entre los objetos de una aplicación. El InterfaceBuilder automáticamente crea y agrega éste objeto a nuestro proyecto cuando arrastramos el menú de fuente de letras al menú de nuestra aplicación.

El Control de fuentes de letras es el centro de actividad para la conversión de fuentes, acepta mensajes del usuario de conversión desde el panel de control de las mismas y convierte la fuente actual en la seleccionada.

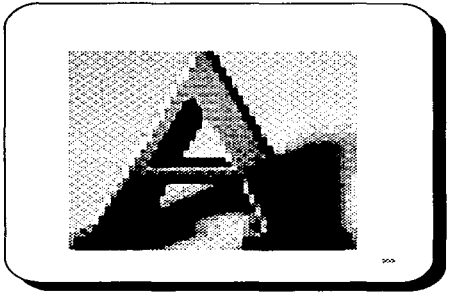


Figura 3.35 Controlador de fuentes de letras

3.3.10 La prueba de la interface

Después de que se termina la interface, el InterfaceBuilder nos permite ver como trabaja desde la perspectiva del usuario, esto lo haremos al escoger la opción de Prueba de Interface (Test Interface) que se encuentra en el menú de Documento (Document). Al seleccionar esta opción los menús de InterfaceBuilder, ventanas, y los paneles desaparecen, solamente se observa la interface que debe ver en la pantalla el usuario y el icono que se muestra en la figura 3.36.

Es importante verificar:

1. Que el cursor se mueva de campo a campo cuando oprimamos cada uno de ellos.
2. Que podamos copiar, cortar, y pegar texto (Acciones de respuesta inmediata).
3. Que se pueda imprimir.

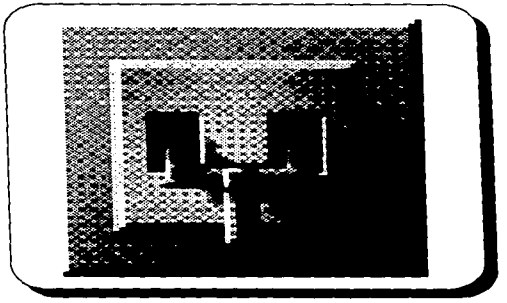


Figura 3.36 Icono del test de la interface

Una vez que se verifican los puntos anteriores la interface queda terminada.

Si es necesario agregar una clase ya existente a nuestro archivo nib, se debe de arrastrar dicha cabecera desde el visor de clases al interior de nuestro archivo nib; al agregarla aparecerá en el visor de clases con las acciones y salidas que se le definieron, pero no podemos todavía utilizar dicha clase, debemos de inicializarla de la manera siguiente:

1. Hacer una instancia para la clase.
2. Conectar las instancias salidas y acciones con otros objetos dentro del archivo nib.

3.4 IconBuilder

3.4.1 Inicio de IconBuilder

El IconBuilder es una herramienta para crear iconos para las aplicaciones, el IconBuilder no es una aplicación solamente para dibujo, ofrece integración con otras aplicaciones, ya que tiene la capacidad para crear y editar múltiples archivos de dibujo.

El IconBuilder se encuentra en el Workspace Manager y su icono es el que se indica en la figura 3.37.

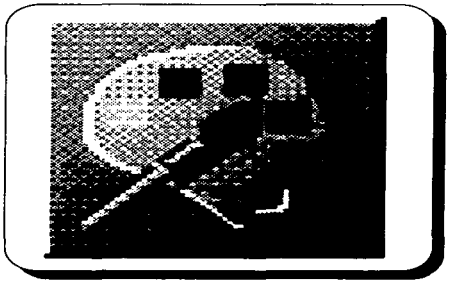


Fig. 3.37 Icono del IconBuilder

Una vez que se activa el IconBuilder aparece una pantalla como la que se muestra en la figura 3.38, la cual tiene una barra de herramientas que sirve para crear o editar iconos e imágenes.

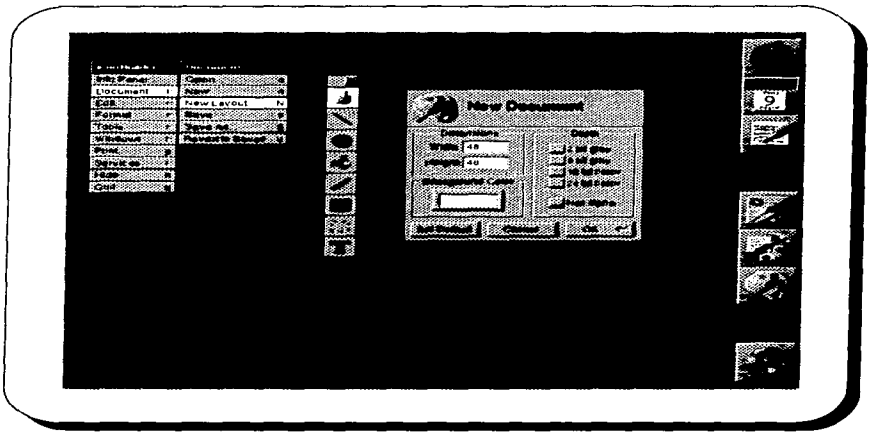


Fig. 3.38 Pantalla inicial del IconBuilder

3.4.2. Creación de iconos e imágenes.

Cuando el IconBuilder se activa, aparece una nueva ventana que utiliza un tamaño por default; este se puede modificar de acuerdo a las necesidades del usuario.

Para crear un nuevo archivo se selecciona el comando de nuevo en el menú de documento, así se crea una ventana por default de 48 x 48 pixeles. Si se desea crear un archivo con atributos diferentes; primero en el menú de documento en lugar de seleccionar la opción de New (Nuevo) hay que elegir la opción de

New Layout (nuevo esquema) con lo que aparecerá un panel como la de la figura 3.39.

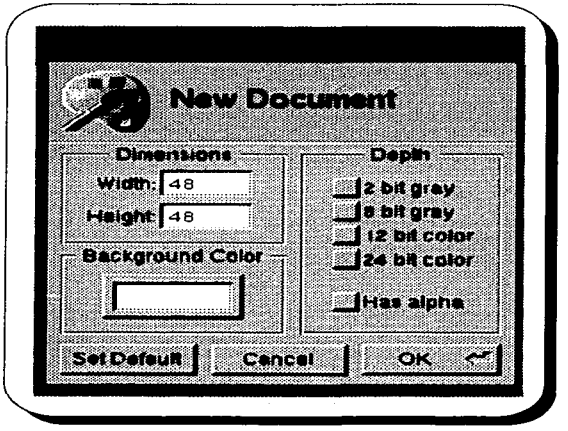


Fig. 3.39 Panel para crear un nuevo documento.

Una vez que aparece el panel de Nuevo Documento, se escogen las nuevas opciones, como son: tamaño, color, tipo, etc.; una vez que se ha determinado como va a ser la nueva ventana se da un click en OK.

Para abrir un archivo existente, hay que activar el comando Open (abrir) en el menú de Documento y usar el panel de abrir para buscar el archivo. Se pueden abrir archivos que contengan la figura de icono o una imagen; en la cual se puede editar, copiar y pegar la imagen o icono en otro documento. Además de abrir archivos .tiff el IconBuilder puede abrir archivos .gif y .eps.

Para guardar un archivo se selecciona el comando de Save (guardar) en el menú de documento. Si el documento no ha sido salvado antes aparece un panel como el de la figura 3.40 en donde se especifica un nombre, formato y ubicación.

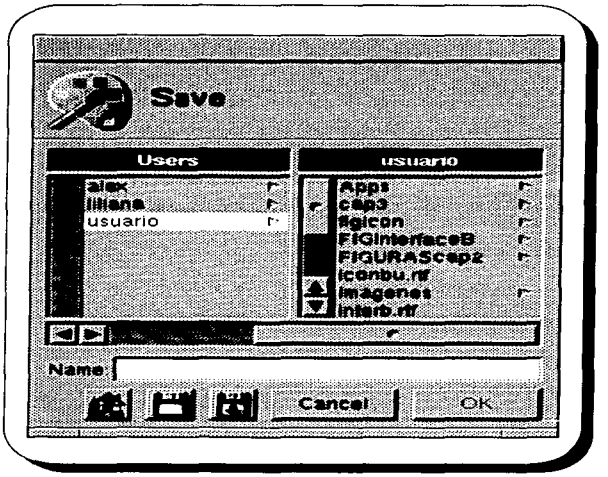


Fig. 3.40 Panel para guardar imágenes

Aunque el documento a salvar no sea .tiff (por ejemplo puede ser .giff) IconBuilder puede salvarlo como un archivo .tiff

El IconBuilder guarda los archivos .tiff en un formato sin comprimir por lo que antes de utilizarlo como parte de un proyecto de aplicación, usted deberá usar la aplicación tiffutil para comprimir el archivo.

3.4.2.1 La barra de herramientas y el inspector de atributos.

La barra de herramientas es como la de la figura 3.41, está aparece automáticamente cuando se activa el IconBuilder, si se cierra la barra, está se puede recobrar mediante el comando de herramientas en el menú.

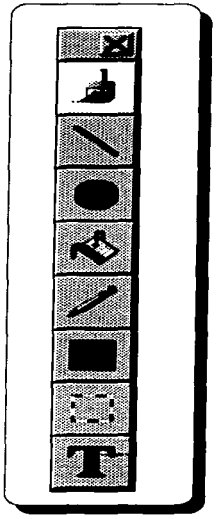


Fig. 3.41 Barra de herramientas del IconBuilder

Para usar cualquier herramienta, está se selecciona primero mediante un click en su icono en la barra, una vez seleccionada, se puede utilizar en la ventana de la imagen.

Cuando se usa el panel de Herramientas, se debe tener abierto el panel de Colores (figura 3.42), ya que todas las herramientas de dibujo inician utilizando el color gris; ahí se especifican los colores que se van a utilizar, tiene varias paletas en donde podemos seleccionar el color.

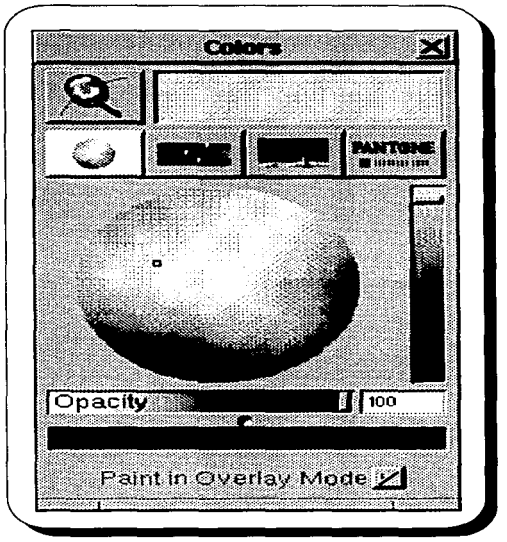


Fig. 3.42 Panel de Color.

Para especificar las características de cada una de las herramientas es necesario utilizar su inspector (este se encuentra en el menú de herramientas), para utilizarlo se da un click en el icono de la herramienta y se abre el inspector.

A continuación daremos una breve descripción de cada una de las herramientas de la barra y de su inspector de atributos:

1. La brocha

Esta es útil para llenar áreas grandes con un color en particular, primero se selecciona el color deseado en el panel de color y posteriormente se da un click en el área a pintar sin soltar el mouse. su inspector es el que se muestra en la figura 3.43, ahí se selecciona la forma la punta con la que desea trabajar mediante un click.

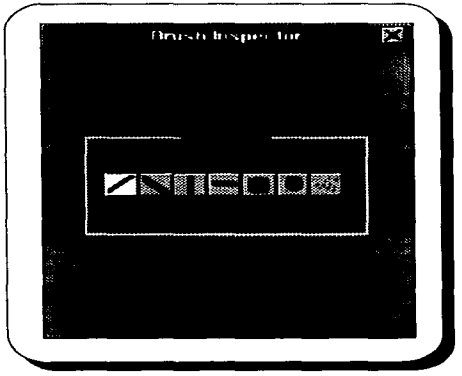


Fig. 3.43 Inspector de la Brocha

2. La Línea recta

Se utiliza para trazar líneas, mediante un click se indica el punto de inicio y sin soltar el botón del mouse se arrastra hasta el punto final; el inspector de Línea se muestra en la figura 3.44, se utiliza para cambiar la forma del principio, fin y grosor de las líneas; mediante la barra de desplazamiento o el campo de texto se indica el grosor con un valor entre 1 y 50 pixels; con un click se escoge alguno de los tres tipos de raya para el principio y final de línea.

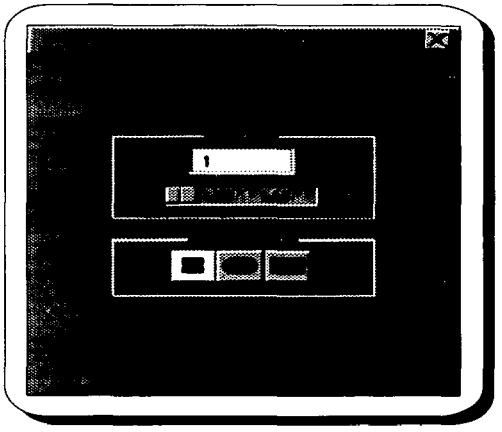


Fig. 3.44 Inspector de la línea recta

3. El Ovalo

Se utiliza para dibujar círculos y óvalos; se da un click en la ventana de aplicación y se arrastra el ratón hasta determinar el tamaño, posición y forma; en

su inspector se puede seleccionar uno de los tipos de círculos y óvalos como los que se muestran en la fig. 3.45 que se pueden hacer.

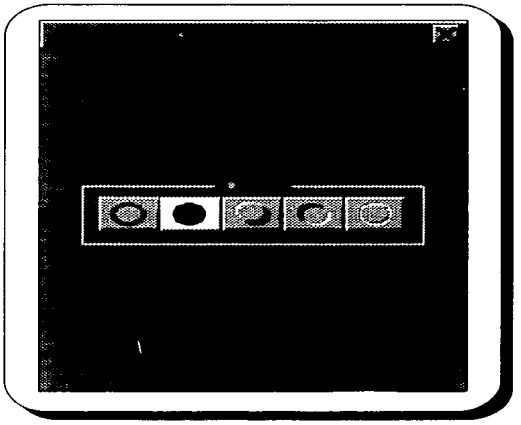


Fig. 3.45 Inspector del Ovalo

4. La cubeta.

Se utiliza para cambiar el color de un área con color idéntico o por pixeles, el color puede ser elegido en el panel de color, está herramienta no tiene inspector.

5. El Lápiz

Se utiliza para dibujar líneas de forma libre, mediante un click se le indica el comienzo y se va arrastrando sin soltar el ratón del mouse hasta conseguir la forma deseada indicándole el fin soltando el botón del mouse; su inspector es el

que se muestra en la figura 3.46; se utiliza para cambiar el grosor de la punta del lápiz; se puede usar la barra de desplazamiento o el campo de texto para indicar el grosor de la línea con un valor entre 1 y 50 pixeles.

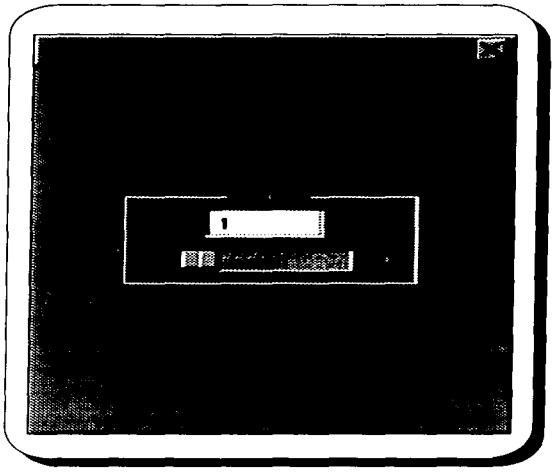


Fig. 3.46 Inspector del lápiz

6. El Rectángulo

Se usa para dibujar cuadrados y rectángulos, de manera similar al óvalo; su inspector es el que se indica en la fig. 3.47, se usa para cambiar el aspecto de los cuadrados y rectángulos que se van a dibujar mediante un click en alguno de los siete botones.

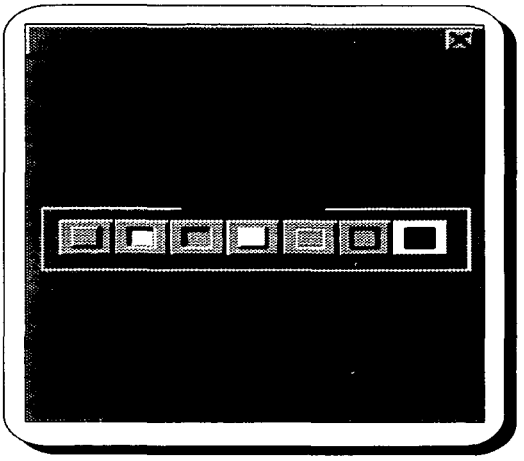


Fig. 3.47 Inspector del rectángulo.

7. Rectángulo punteado

Sirve para seleccionar áreas determinadas de una imagen que posteriormente se pueden copiar, pegar y borrar. El inspector de ésta herramienta sirve para cambiar la orientación de la selección que se hizo y tiene 2 opciones la opción de espejo y de rotación.

Si se utiliza el inspector de espejo se puede hacer la selección de Vertical u Horizontal para indicar en que dirección se va a reflejar la selección que se hizó y su inspector es como el que aparece en la figura 3.48.

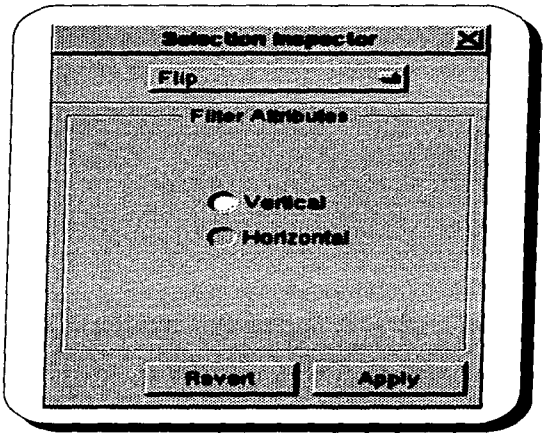


Fig. 3.48 Inspector para obtener el espejo de la selección de una imagen

Si se utiliza el inspector para rotar, aparece el inspector como el que se muestra en la fig. 3.49, en donde se especifican los ángulos en los que se va a rotar la selección entre valores de 0° y 360° usando la barra de desplazamiento o el campo de texto. Este valor representa el número de grados en los que se va a rotar la selección

Una vez que se hace la selección de las opciones ya sea de espejo o de rotación y en caso de que los resultados no sean los deseados, ambos inspectores cuentan con la opciones de revertir los resultados.

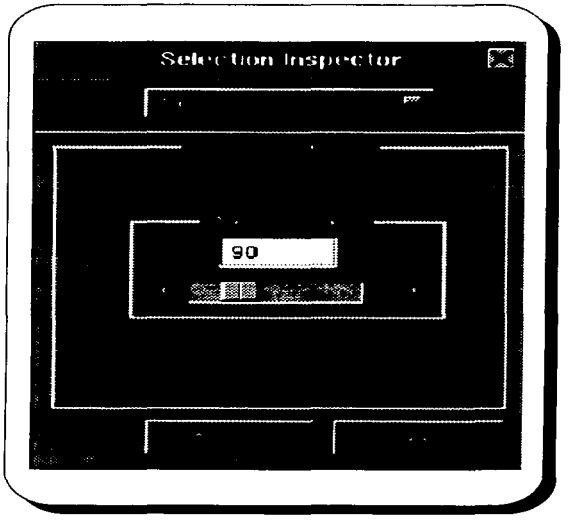


Fig. 3.49 Inspector para rotar la selección

8. El Texto

Se utiliza cuando se quiere agregar texto en una imagen; para poder incluir texto se utiliza el inspector que es el que se muestra en la fig. 3.50, aquí se escribe el texto a incluir, se cambian los atributos de fuente y formato antes de incluirlo en la imagen .

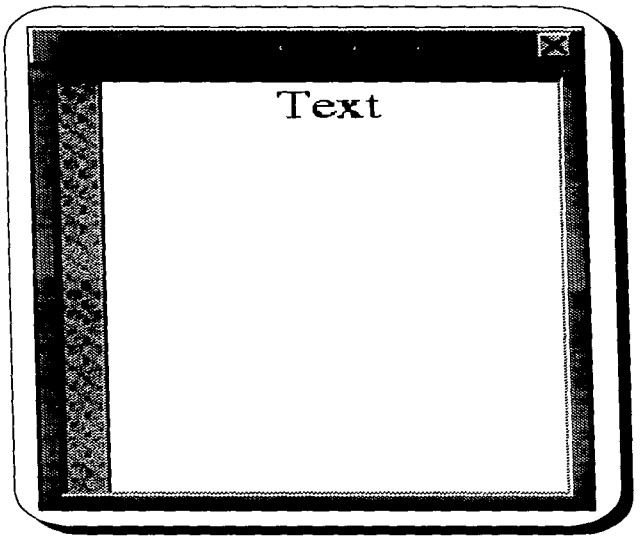


Fig. 3.50 El Inspector de texto

3.4.3 Detallar una imagen.

Para detallar imágenes IconBuilder tiene una herramienta muy útil llamada Obesebites que maneja las imágenes como un mapa de bits, por ejemplo si se quiere hacer un icono de 48 x 48 pixels, está herramienta ofrece una vista en la que las imágenes aparecen como si se estuvieran viendo a través de una lupa.

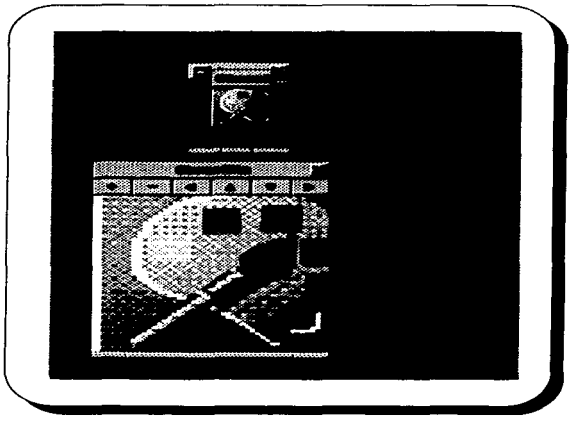


Fig. 3.51 Obesebites

Este aplicación amplia la imagen de la ventana principal, y permite ver un zoom de una parte del dibujo y tiene una barra con botones.

Los botones de + y - reducen o amplían el margen del zoom, y las flechas permiten el movimiento a través de la imagen

Hay realmente dos ObeseBits de paneles como se puede ver en la figura 3.51; el panel grande es el que se utiliza para la edición, el pequeño se mueve sobre la imagen del documento y el grande muestra la parte ampliada de la imagen que se ve en el pequeño.

CAPITULO IV

" APLICACIONES PRACTICAS "

4.1 Aplicación: Agencia de viajes

4.1.1 Elaboración de la aplicación

1. Se crea un nuevo proyecto con el ProjectBuilder (Figura 4.1).

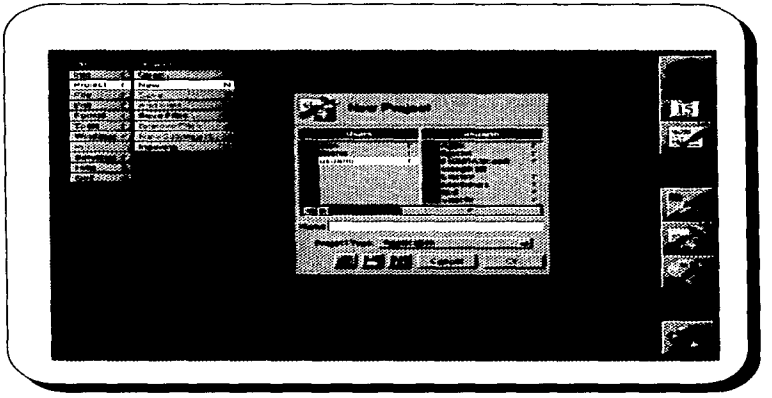


Figura 4.1 Creación de un proyecto nuevo.

2. En la ventana del ProjectBuilder se abre el archivo .nib con lo que se activa el InterfaceBuilder.
3. Una vez que estamos en el InterfaceBuilder se crea la interface gráfica de la aplicación agregando los objetos que sean necesarios de las paletas a la ventana de inicio.
4. Se agregan las imágenes y sonidos que se quieran incluir al proyecto (se arrastran a la ventana de instancias y posteriromente a los botones).

5. La interface de la aplicación debe quedar como se indica en la figura 4.2.

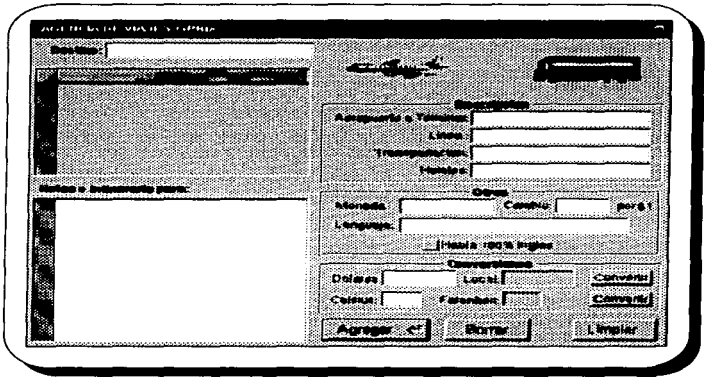


Figura 4.2 Interface de la aplicación.

6. Se crea la clase **conversiones** (subclase de la Clase NSObject) y se agrega el método **convertCantidad.**
7. Se crea la instancia de la clase anterior y los archivos que se van a agregar al ProjectBuilder (archivos .m y .h).
8. Se crea una subclase que se llame **AGControl** en la cual se crean las salidas y acciones siguientes:

Salidas	**Acciones**
vista_de_la_Tabla	*nuevoRegistro*
grados_celsius	*convertCelsius*
etiqueta_Comentarios	*convertMoneda*
campo_Comentarios	*borraRegistro*

Salidas	Acciones
conversion	*limpiaCampos*
nombre_de_Pais	*registroProximo*
moneda_Dolares	*registroAnterior*
nombre_moneda	*switchChecked*
cantidad_dinero	
hablar_Ingles_Switch	
grados_fahrenheit	
idioma	
forma	

9. Se crea una instancia **AGControl**.
10. Se crea una clase que se llame **Pais**, de momento no va a tener ni salidas ni métodos, estos se agregan posteriormente en el ProjectBuider, esta clase va a controlar el flujo de datos que se introduzca dentro de la aplicación.
11. Este paso es uno de los más importantes debido a que en él vamos a realizar las conexiones de la interface con el control del programa; para esto es necesario abrir el inspector y seleccionar conexiones.
12. Se hacen las conexiones como las que que se indican en las figuras del punto 4.1.2 . Recuerde que se oprime la tecla control y se arrastra el mouse hasta el objeto destino y en el inspector indicamos la salida que vamos a utilizar).
13. Se guarda el archivo .nib y se cierra el InterfaceBuilder.
14. En el ProjectBuilder se abre el archivo **AGControl.h** y verificamos que se encuentren los métodos y las acciones que declaramos en el InterfaceBuilder y agregamos los de limpieza y asignación de memoria de

objetos (init, dealloc), recuerde que al crear un método en el archivo .h tiene que tener su correspondiente en el archivo .m.

15. Se edita el archivo **Pais.h** en él cual nuevamente vamos a introducir los métodos de limpieza y asignación de memoria en los cuales indicaremos las variables y métodos a utilizar durante la compilación de la aplicación, después se edita el archivo **Pais.m** en el cual se escribe el código necesario para el manejo de cadenas de datos para el buen funcionamiento de la aplicación. (el código se encuentra en las páginas subsecuentes.)
16. Se edita el archivo **Conversiones.m** en el cual modificamos la salida y la operación que va a realizar con las cantidades que se introduzcan
17. Se edita el archivo **AGControl.m** el cual va a reconocer los datos que se introduzcan en la interface (recuerde que las conexiones se realizaron de la instancia de **AGControl** hacia la interface por lo que los datos que se introduzcan en la interface primero llegan a **AGControl** y posteriormente a **Pais** el cual se encargará de manipular todos los datos que le lleguen).
18. Se hace el icono representativo para la aplicación con el IconBuilder, el tamaño es de 48 x 48 pixeles, para agregarlo se abre el inspector del ProjectBuilder y en el menú de atributo de proyecto se coloca el icono (Figura 4.3).

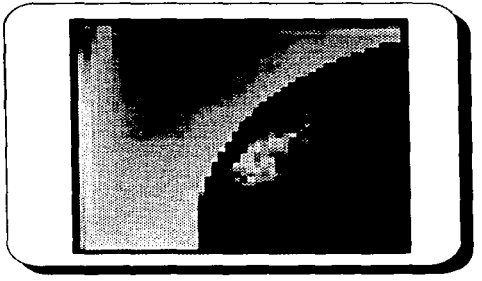


Figura 4.3 Icono de la aplicación Agencia de viajes

19. Se compila el programa y aparece el archivo ejecutable en el directorio de la aplicación el cual ésta identificado por el icono que se le asignó anteriormente.

4.1.2 Figuras de las conexiones de los objetos

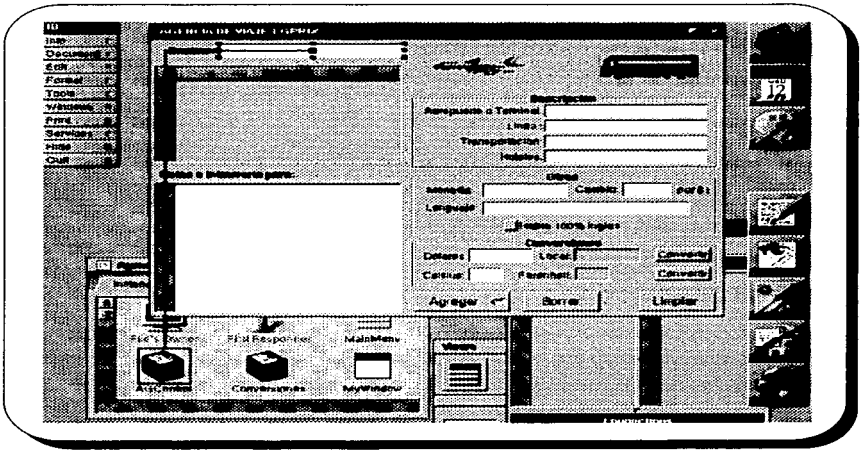


Figura 4.4 Conexión 1

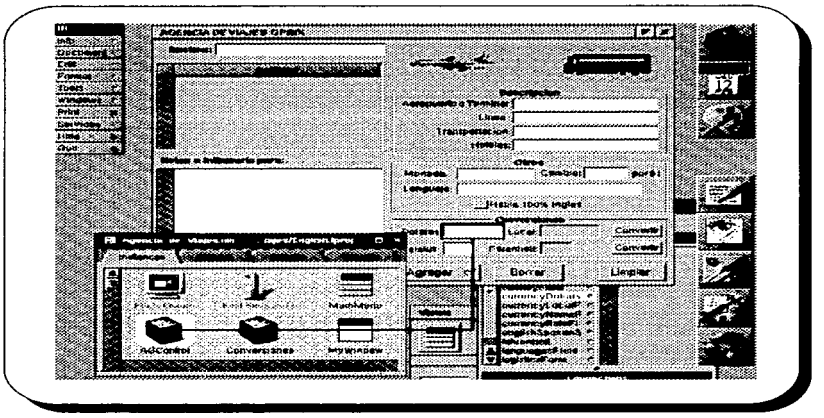


Figura 4.5 Conexión 2

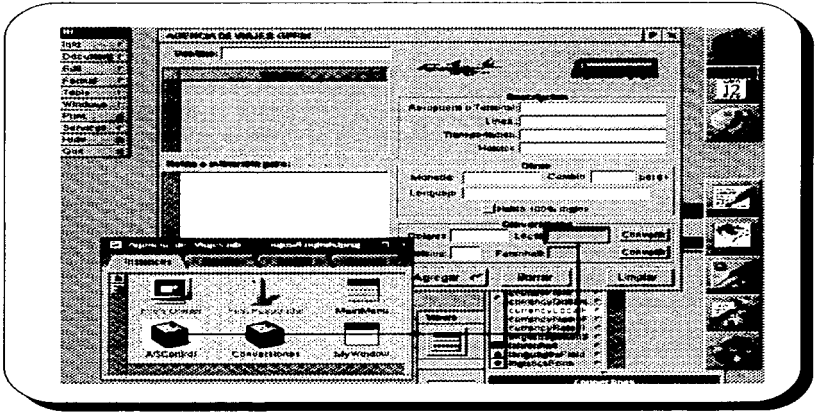


Figura 4.6 Conexión 3

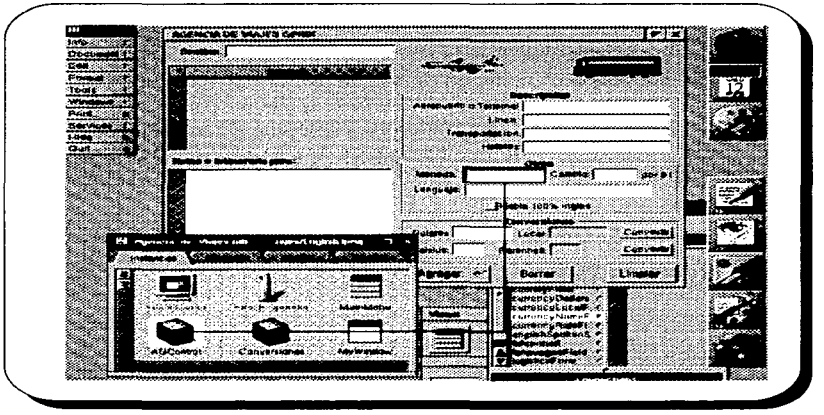


Figura 4.7 Conexión 4

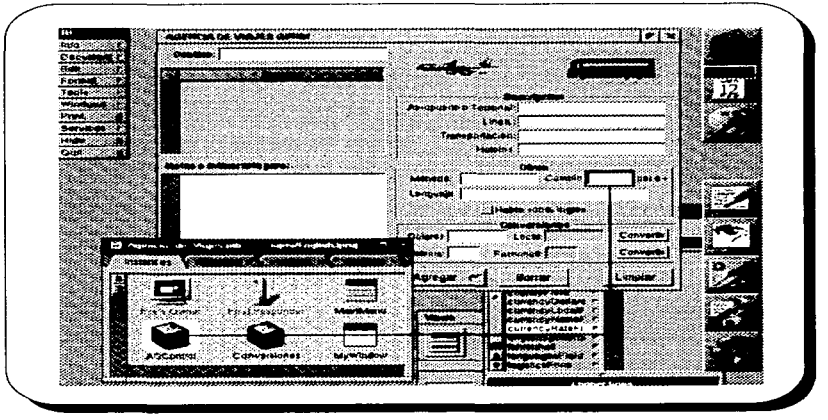


Figura 4.8 Conexión 5

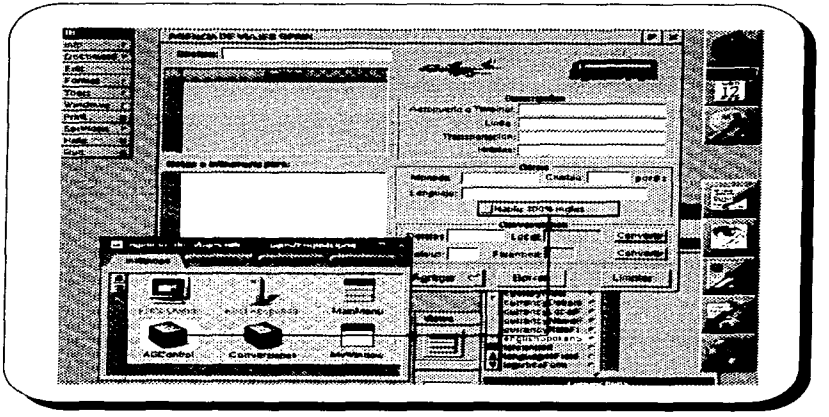


Figura 4.9 Conexión 6

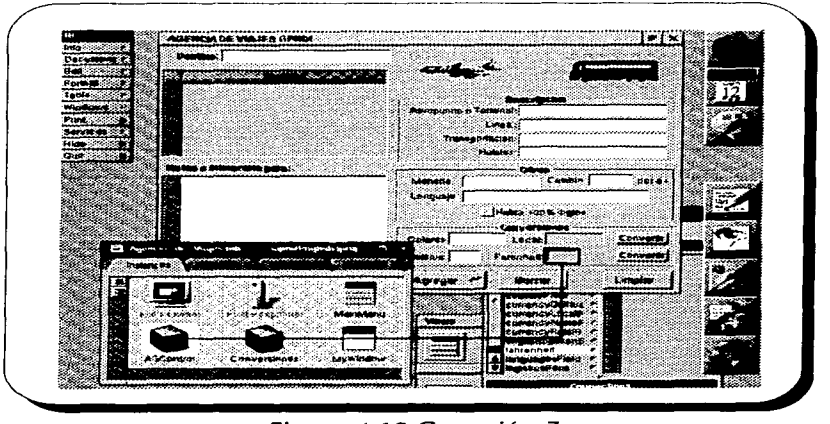


Figura 4.10 Conexión 7

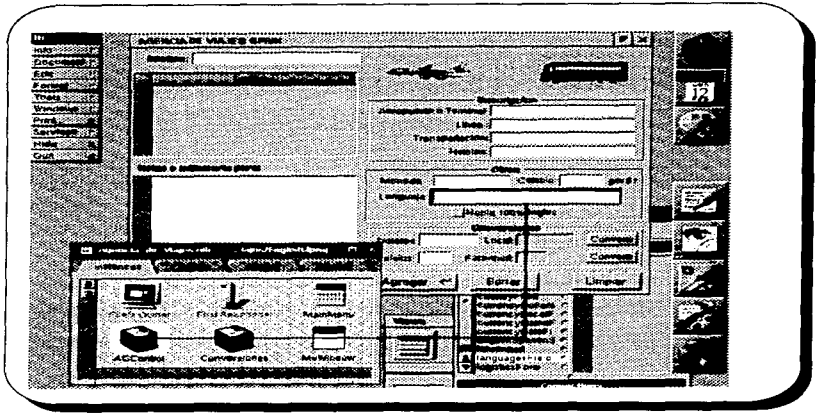


Figura 4.11 Conexión 8

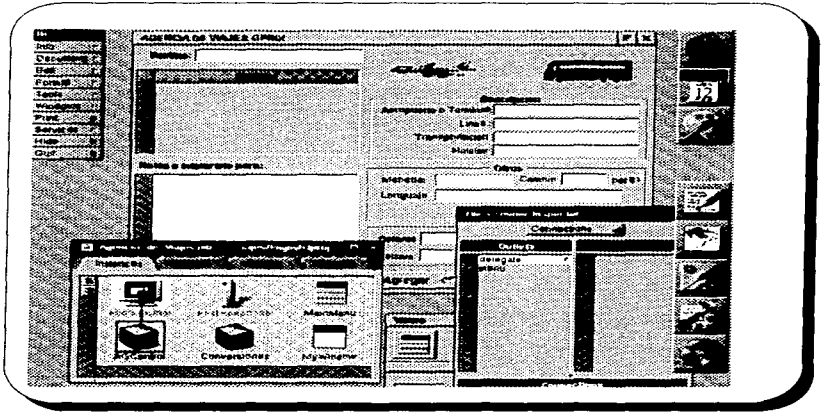


Figura 4.12 Conexión 9

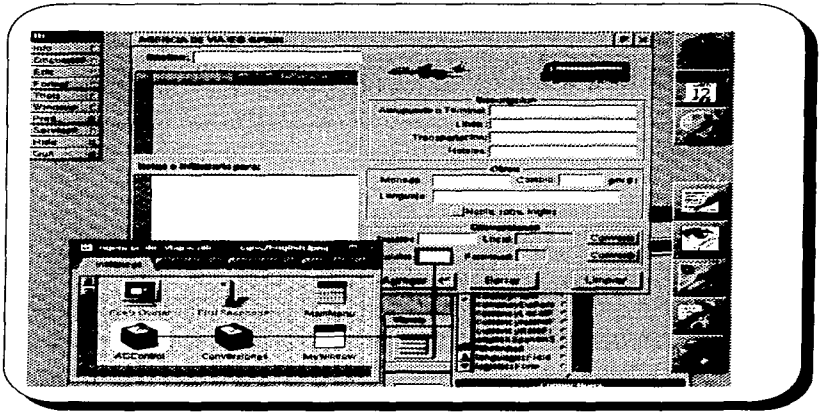


Figura 4.13 Conexión 10

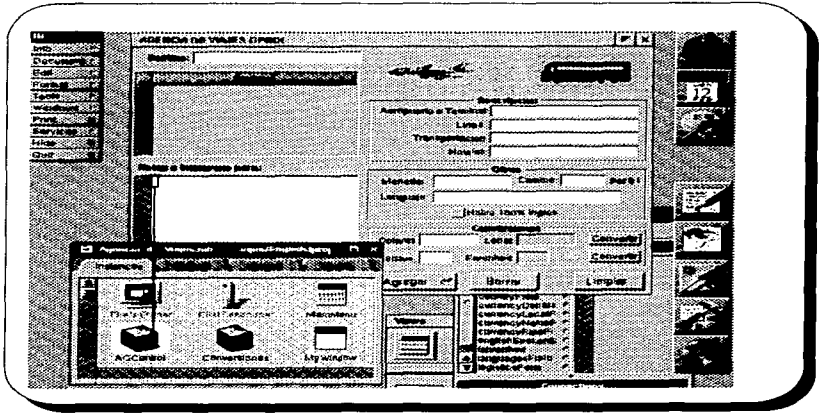


Figura 4.14 Conexión 11

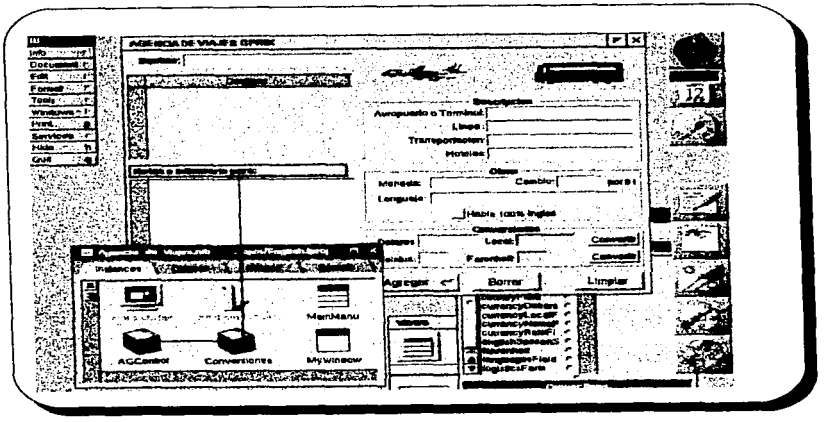


Figura 4.15 Conexión 12

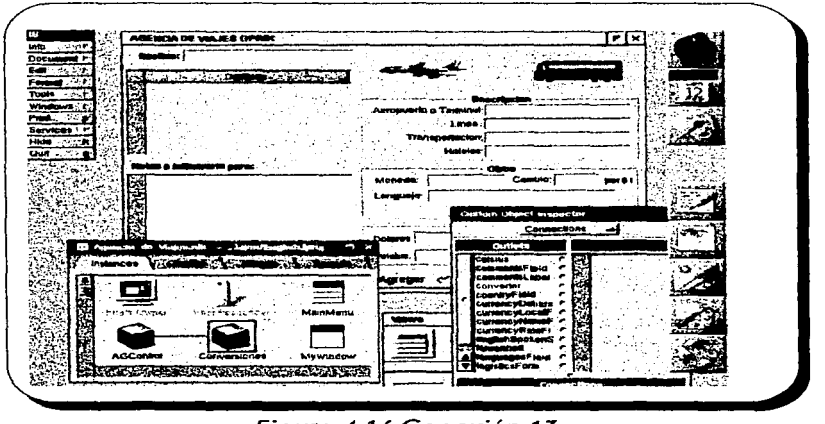


Figura 4.16 Conexión 13

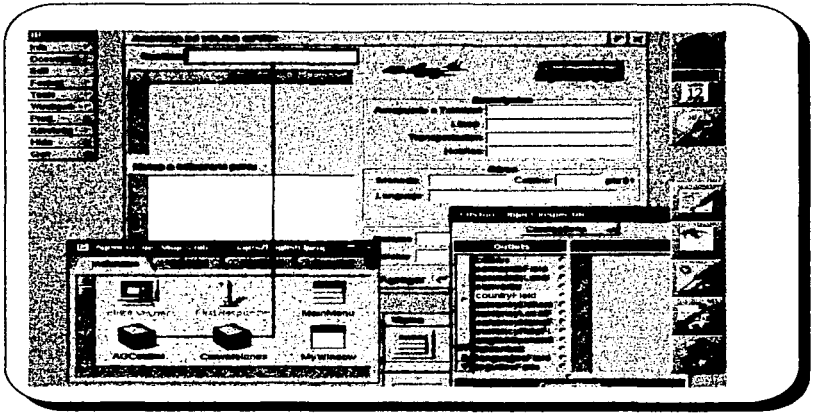


Figura 4.17 Conexión 14

4.1.3 Archivos de la aplicación.

AGControl.h

```
#import <AppKit/AppKit.h>
#import <Foundation/Foundation.h>
#import "Pais.h"

enum DescripcionFormTags {
   LGaeropuerto_terminal=0,
   LGlineas,
   LGtransportacion,
   LGhoteles
};

@interface AGControl:NSObject
{
   id     vista_de_la_Tabla;
   id     grados_celsius;
   id     etiqueta_Comentarios;
   id     campo_Comentarios;
   id     conversion;
   id     nombre_de_Pais;
   id     moneda_Dolares;
   id     moneda_Local;
   id     nombre_moneda;
   id     cantidad_dinero;
   id     hablar_Ingles_Switch;
   id     grados_fahrenheit;
   id     idioma;
   id     forma;

   NSMutableDictionary *paisDict;
   NSMutableArray *paisKeys;
   BOOL recordNeedsSaving;
}
                /*    Metodos que realizan acciones en la interface */

- (void)nuevoRegistro:(id)sender;                  /* Crea un nuevo registro */
- (void)convertCelsius:(id)sender;                 /* Convierte los grados celsius a farenheit */
- (void)convertMoneda:(id)sender;                  /* Convierte en tipo de moneda a dolares */
- (void)borraRegistro:(id)sender;                  /* Borra el registro señalado */
- (void)handleTVClick:(id)sender;                  /* Metodo que busca si existe el destino nuevo */
- (void)limpiaCampos:(id)sender;                   /* Limpia los campos para introducir nuevos campos*/
- (void)registroProximo:(id)sender;                /* Salta al campo siguiente dentro de la ventana
                                                    destinos */
- (void)registroAnterior:(id)sender;               /* Regresa al registro anterior en la ventana de
                                                    destinos*/
- (void)switchChecked:(id)sender;                  /*Método que revisa si se marca la opción de hablar
                                                    100% ingles */
```

```
        /*   Metodos que leen y escriben datos */

- (void)populateFields:(Pais *)aRec;
- (void)extractFields:(Pais *)aRec;

        /*   Métodos de Limpieza, asignación de memoria y busqueda  de los objetos*/

- (id)init;
- (void)awakeFromNib;
- (void)dealloc;

        /* Métodos de notificación */

- (void)textDidChange:(NSNotification *)notification;
- (BOOL)applicationShouldTerminate:(id)sender;
@end
```

AGControl.m

```
#import "AGControl.h"
#import "Conversiones.h"
@implementation AGControl

- (void)addRecord:(id)sender
{
    Pais *unPais;
    NSString *nombrePais = [nombre_de_Pais stringValue];

        /* Verifica nombre en el Diccionario
         En caso de existir borra ese registro, de lo contrario, lo crea */

    if (nombrePais && (![nombrePais isEqualToString:@""])) {
        unPais = [paisDict objectForKey:nombrePais];

        if (unPais && recordNeedsSaving)
        {
          [countryDict removeObjectForKey:[unPais name]];
        }
        else if (!unPais)
            unPais = [[Pais alloc] init];
        else return;
        [self extractFields:unPais];
        [paisDict setObject:unPais forKey:[unPais name]];
        [paisKeys addObject:[unPais name]];

            /* Ordena el arreglo de datos */

        [countryKeys sortUsingSelector:@selector(compare:)];

        recordNeedsSaving=NO;
```

```
        [commentsLabel setStringValue:[NSString stringWithFormat:@"Notas e Intinerario para
  %@", [nombre_de_Pais stringValue]]];
        [nombre_de_Pais selectText:self];
        [vista_de_la_Tabla title];
        [vista_de_la_Tabla selectRow:[paisKeys indexOfObject:[unPais name]]
  byExtendingSelection:NO];

    }
    return;

}

- (void)limpiaCampos:(id)sender
{
    [nombre_de_Pais setStringValue:@""];

    [[forma cellAtIndex:LGaeropuerto_terminal] setStringValue:@""];
    [[forma cellAtIndex:LGlineas] setStringValue:@""];
    [[forma cellAtIndex:LGtransportacion] setStringValue:@""];
    [[forma cellAtIndex:LGhoteles] setStringValue:@""];


    [nombre_moneda setStringValue:@""];
    [cantidad_dinero setFloatValue:0.000];
    [idioma setStringValue:@""];
    [hablar_Ingles_Switch setState:NO];
    [moneda_Dolares setFloatValue:0.00];
    [moneda_Local setFloatValue:0.00];
    [grados_celsius setIntValue:0];
    [grados_fahrenheit setIntValue:0];
    [campo_Comentarios setString:@""];
    [nombre_de_Pais selectText:self];
    return;
}

- (void)populateFields:(Pais *)aRec
{
    [nombre_de_Pais setStringValue:[aRec name]];
    [[forma cellAtIndex:LGaeropuerto_terminal] setStringValue:[aRec aeropuerto_terminal]];
    [[forma cellAtIndex:LGlineas] setStringValue:[aRec lineas]];
    [[forma cellAtIndex:LGtransportacion] setStringValue:[aRec transportacion]];
    [[forma cellAtIndex:LGhoteles] setStringValue:[aRec hoteles]];
    [nombre_moneda setStringValue:[aRec currencyName]];
    [cantidad_dinero setFloatValue:[aRec currencyRate]];
    [idioma setStringValue:[aRec languages]];
    [habla_Ingles_Switch setState:[aRec habla_Ingles]];
    [campo_Comentarios setString:[aRec comentarios]];
    [nombre_de_Pais selectText:self];
    return;
}
```

```
- (void)extractFields:(Pais *)aRec
{
   [aRec setName:[nombre_de_Pais stringValue]];
   [aRec setAeropuerto_terminal:[[forma cellAtIndex:LGaeropuerto_terminal] stringValue]];
   [aRec setLineas:[[forma cellAtIndex:LGlineas] stringValue]];
   [aRec setTransportacion:[[forma cellAtIndex:LGtransportacion] stringValue]];
   [aRec setHoteles:[[forma cellAtIndex:LGhoteles] stringValue]];
   [aRec setNombreMoneda:[nombre_moneda stringValue]];
   [aRec setCantidadDinero:[cantidad_dinero floatValue]];
   [aRec setIdioma:[idioma stringValue]];
   [aRec setHablaIngles:[habla_ingles_Switch state]];
   [aRec setComentarios:[campo_Comentarios string]];
      return;
}

- (void)convertCelsius:(id)sender
{
   return;
}

- (void)convertMoneda:(id)sender
{

   [moneda_Local setFloatValue:[converter convertCantidad:[moneda_Dolares floatValue]
byRate:[cantidad_dinero floatValue]]];

   return;
}

- (void)borraRegistro:(id)sender
{
   Pais *unPais;
   NSString *nombrePais = [nombre_de_Pais stringValue];

   if (nombrePais && (![nombrePais isEqualToString:@""])) {
      unPais = [paisDict objectForKey:nombrePais];

      if (unPais) {
         [paisDict removeObjectForKey:nombrePais];
         [paisKeys removeObject:nombrePais];
         [self limpiaCampos:self];
         [vista_de_laTabla tile];
      }
   }
   return;
}

- (void)switchChecked:(id)sender { recordNeedsSaving = YES; }
- (void)handleTVClick:(id)sender
{
   Pais *aRec, *newRec, *newerRec;
   int index = [sender selectedRow];
```

```
    if (recordNeedsSaving) {
        if ( aRec = [countryDict objectForKey:[nombre_de_Pais stringValue]] ) {
            if (aRec) {
                [paisDict removeObjectForKey:[aRec name]];
                [paisKeys removeObject:[aRec name]];
            }
        }

            /* Crea un objeto pais y lo adiciona al diccionario (nuestra base de datos )*/

        newRec = [[Pais alloc] init];
        [self extractFields:newRec];
        [paisDict setObject:newRec forKey:[nombre_de_Pais stringValue]];
        [paisKeys addObject:[nombre_de_Pais stringValue]];

            /* ordena el arreglo de datos*/

        [countryKeys sortUsingSelector:@selector(compare:)];
    }

    if (index >= 0 && index < [paisKeys count]) {
        newerRec = [paisDict objectForKey:[paisKeys objectAtIndex:index]];
        [self populateFields:newerRec];
        [etiqueta_Comentarios setStringValue:[NSString stringWithFormat:
            @"Notes and Itinerary for %@", [nombre_de_Pais stringValue]]];
        recordNeedsSaving=NO;
        [vista_de_la_Tabla tile];
    }
    return;
}

- (void)registroProximo:(id)sender
{
    int r;
    r = [vista_de_la_Tabla selectedRow];
    if (r == [paisKeys indexOfObject:[paisKeys lastObject]])
        r = 0;
    else
        r++;
    [vista_de_la_Tabla selectRow:r byExtendingSelection:NO];
    [self handleTVClick:vista_de_la_Tabla];
    return;
}

- (void)registroAnterior(id)sender
{

    int r;
    r = [vista_de_la_Tabla selectedRow];
    if (r == 0)
        r = [paisKeys indexOfObject:[paisKeys lastObject]];
```

```
    else
        r--;
    [vista_de_la_Tabla selectRow:r byExtendingSelection:NO];
    [self handleTVClick:vista_de_la_Tabla];

    return;
}

- (BOOL)validateMenuItem:(NSMenuItem *)anItem
{
    int row = [vista_de_la_Tabla selectedRow];
    if ([[anItem title] isEqualToString:@"Proximo registro"] &&
        (row == [paisKeys indexOfObject:[paisKeys lastObject]])) {
        NSBeep();
        return NO;
    }
    if ([[anItem title] isEqualToString:@"Registro anterior"] && row == 0 ) {
        NSBeep();
        return NO;
    }
    return YES;
}

- (id)init
{
    NSString *storePath = [[NSBundle mainBundle] pathForResource:@"TravelData"
ofType:nil];
    [super init];
    paisDict = [NSUnarchiver unarchiveObjectWithFile:storePath];

    if (!paisDict) {
        paisDict = [[NSMutableDictionary alloc] initWithCapacity:10];
        paisKeys = [[NSMutableArray alloc] initWithCapacity:10];
    } else
        paisDict = [paisDict retain];
    recordNeedsSaving=NO;
    return self;
}

- (void)dealloc
{
    [paisDict release];
    [paisKeys release];
    [super dealloc];
}

- (void)awakeFromNib
{
    NSArray *tmpArray = [[paisDict allKeys] /* 1 */
                    sortedArrayUsingSelector:@selector(compare:)];
    paisKeys = [[NSMutableArray alloc] initWithArray:tmpArray];
    [nombre_de_Pais selectText:self];
```

```
    [[NSNotificationCenter defaultCenter] addObserver:self
                selector:@selector(textDidChange:)
                name:@"NSControlTextDidChangeNotification" object:nil];

  [vista_de_la_Tabla setDataSource:self];
  [vista_de_la_Tabla setDelegate:self];
  [vista_de_la_Tabla sizeLastColumnToFit];
  [campo_Comentarios setDelegate:self];
  [cantidad_dinero setDelegate:self];
  [[cantidad_dinero cell] setEntryType:NSFloatType];
  [[cantidad_dinero cell] setFloatingPointFormat:YES left:2 right:1];
  [[moneda_Dolares cell] setEntryType:NSFloatType];
  [[moneda_Dolares cell] setFloatingPointFormat:YES left:6 right:2];
  [[moneda_Local cell] setEntryType:NSFloatType];
  [[moneda_Local cell] setFloatingPointFormat:YES left:6 right:2];
  [[grados_celsius cell] setEntryType:NSFloatType];
  [[grados_celsius cell] setFloatingPointFormat:YES left:2 right:1];
}

- (int)numberOfRowsInTableView:(NSTableView *)theTableView
{
  return [paisKeys count];
}

- (id)vista_de_la_Tabla:(NSTableView *)theTableView
    objectValueForTableColumn:(NSTableColumn *)theColumn
        row:(int)rowindex
{
  if ([[theColumn identifier] intValue]==0)
    return [paisKeys objectAtIndex:rowindex];
  else
    return nil;
}
        /* Metodos de Notificacion     */

- (void)textDidChange:(NSNotification *)notification
{
  recordNeedsSaving=YES;
}

- (void)controlTextDidChange:(NSNotification *)notification
{
  recordNeedsSaving=YES;
}

- (BOOL)textShouldBeginEditing:(NSText *)textObj
{
  recordNeedsSaving=YES;
  return YES;
}
```

```
- (BOOL)applicationShouldTerminate:(id)sender
{
   NSString *storePath = [[[NSBundle mainBundle] bundlePath]
stringByAppendingPathComponent:@"TravelData"];
  [self nuevoRegistro:self];
  if (countryDict && [countryDict count])
      [NSArchiver archiveRootObject:countryDict toFile:storePath];
   return YES;
}

- (BOOL)control:(NSControl *)control isValidObject:(id)obj
{
   if (control == currencyRateField) {
      if ([obj floatValue] < 0.0) {
         NSRunAlertPanel(@"Travel Advisor", @"Rate cannot be negative.", nil, nil, nil);
         return NO;
      }
   }
   return YES;
}
@end
```

Pais.h

```
#import <AppKit/AppKit.h>
#import <Foundation/Foundation.h>

@interface Pais : NSObject <NSCoding>
{
   NSString *name;
   NSString *aeropuerto_terminal;
   NSString *lineas;
   NSString *transportacion;
   NSString *hoteles;
   NSString *idioma;
   BOOL     habla_ingles;
   NSString *nombre_moneda;
   float cantidad_dinero;
   NSString *comentarios;
}

- (id)init;
- (void)dealloc;
- (void)encodeWithCoder:(NSCoder *)coder;
- (id)initWithCoder:(NSCoder *)coder;
- (NSString *)name;
- (void)setName:(NSString *)str;
- (NSString *)aeropuerto_terminal;
- (void)setAeropuerto:(NSString *)str;
- (NSString *)lineas;
```

```
- (void)setLineas:(NSString *)str;
- (NSString *)transportacion;
- (void)setTransportacion:(NSString *)str;
- (NSString *)hoteles;
- (void)setHoteles:(NSString *)str;
- (NSString *)idioma;
- (void)setIdioma:(NSString *)str;
- (BOOL)habla_ingles;
- (void)setHabla_Ingles:(BOOL)flag;
- (NSString *)nombre_moneda;
- (void)setNombreMoneda:(NSString *)str;
- (float)cantidad_dinero;
- (void)setCantidadDinero:(float)val;
- (NSString *)comentarios;
- (void)setComentarios:(NSString *)str;

@end
```

Pais.m

```
#import "Pais.h"

@implementation Pais

- (id)init
{
   [super init];

   name=@"";
   aeropuerto_terminal=@"";
   lineas=@"";
   transportacion=@"";
   hoteles=@"";
   idioma=@"";
   nombre_moneda=@"";
   comentarios=@"";

   return self;
}

- (void)dealloc
{
   [name release];
   [aeropuerto_terminal release];
   [lineas release];
   [transportacion release];
   [hoteles release];
   [idioma release];
   [nombre_moneda release];
   [comentarios release];
```

```
    return [super dealloc];
}

- (void)encodeWithCoder:(NSCoder *)coder
{
    [coder encodeObject:name];
    [coder encodeObject:aeropuerto_terminal];
    [coder encodeObject:lineas];
    [coder encodeObject:transportacion];
    [coder encodeObject:hoteles];
    [coder encodeObject:idioma];
    [coder encodeValueOfObjCType:"s" at:&habla_ingles];
    [coder encodeObject:nombre_moneda];
    [coder encodeValueOfObjCType:"f" at:&cantidad_dinero];
    [coder encodeObject:comentarios];

    return;
}

- (id)initWithCoder:(NSCoder *)coder
{
    name = [[coder decodeObject] copy];
    aeropuerto_terminal = [[coder decodeObject] copy];
    lineas = [[coder decodeObject] copy];
    transportacion = [[coder decodeObject] copy];
    hoteles = [[coder decodeObject] copy];
    idioma = [[coder decodeObject] copy];
    [coder decodeValueOfObjCType:"s" at:&habla_ingles];
    nombre_moneda = [[coder decodeObject] copy];
    [coder decodeValueOfObjCType:"f" at:&cantidad_dinero];
    comentarios = [[coder decodeObject] copy];
    return self;
}

- (NSString *)name
{
    return name;
}

- (void)setName:(NSString *)str
{
    [name autorelease];
    name = [str copy];
}

- (NSString *)aeropuerto_terminal { return aeropuerto_terminal; }

- (void)setAeropuerto_terminal:(NSString *)str
{
    [aeropuerto_terminal autorelease];
    aeropuerto_terminal = [str copy];
}
```

```
- (NSString *)lineas { return lineas; }

- (void)setLineas:(NSString *)str
{
   [lineas autorelease];
   lineas = [str copy];
}

- (NSString *)transportacion { return transportacion; }

- (void)setTransportacion:(NSString *)str
{
   [transportacion autorelease];
   transportacion = [str copy];
}

- (NSString *)hoteles { return hoteles; }

- (void)setHoteles:(NSString *)str
{
   [hoteles autorelease];
   hoteles = [str copy];
}

- (NSString *)idioma { return idioma; }

- (void)setIdioma:(NSString *)str
{
   [idioma autorelease];
   idioma = [str copy];
}

- (BOOL)habla_Ingles { return habla_Ingles; }

- (void)setHabla_Ingles:(BOOL)flag
{
   habla_Ingles = flag;
}

- (NSString *)nombre_moneda { return nombre_moneda; }

- (void)setNombre_moneda:(NSString *)str
{
   [nombre_moneda autorelease];
   nombre_moneda = [str copy];
}

- (float)cantidad_dinero {return cantidad_dinero;}

- (void)setCantidad_dinero:(float)val
{
   cantidad_dinero = val;
```

```
}

- (NSString *)comentarios { return comentarios; }

- (void)setComentarios:(NSString *)str
{
   [comentarios autorelease];
   comentarios = [str copy];
}

@end
```

Conversiones.h

```
#import <AppKit/AppKit.h>
#import <Foundation/Foundation.h>
#import <SoundKit/SoundKit.h>

@interface Conversiones : NSObject
{
}

- (float)convertAmount:(float)amt byRate:(float)rate;

@end
```

Conversiones.m

```
#import "Conversiones.h"

@implementation Conversiones

-(float)convertCantidad:( float )cantidad byRate:( float )factor
{
        return (cantidad *factor );
}

@end
```

4.2 Aplicación: PaintBrush1

4.2.1 Elaboración de la aplicación

1. Se crea un nuevo proyecto con el ProjectBuilder (Figura 4.18).

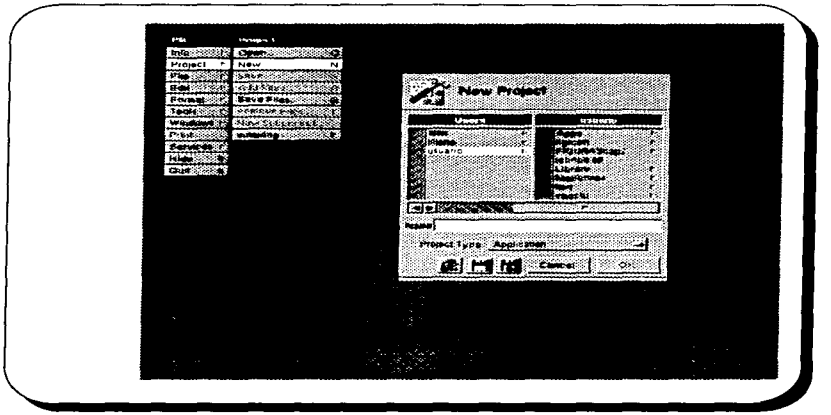


Figura 4. 18 Creación de un proyecto nuevo.

2. En la ventana del ProjectBuilder se abre el archivo .nib con lo que se activa el InterfaceBuilder.
3. Una vez que estamos en el InterfaceBuilder se crea la interface gráfica de la aplicación agregando los objetos que sean necesarios de las paletas a la ventana de inicio.
4. La interface de la aplicación debe quedar como se indica en la figura 4.19.
5. Se crea la clase **PaintBrush1**(subclase de la Clase NSObject) y se agregan las salidas:

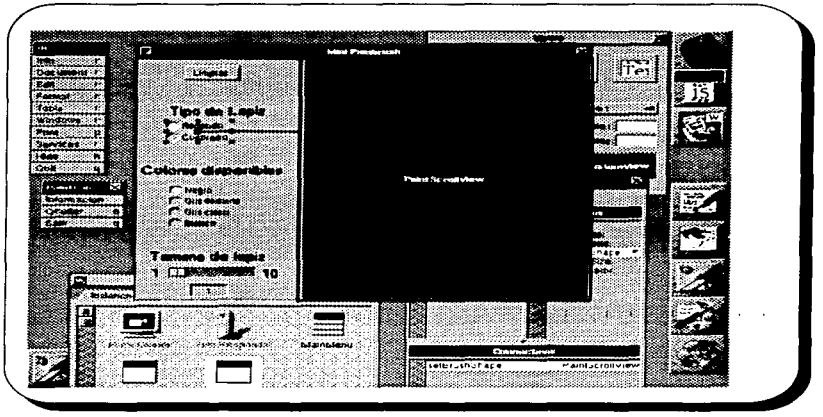


Figura 4.19 Interface de la aplicación.

arrastreBitmap tipoLapiz colores

docBitmap tamanoLapiz

y los métodos:

limpiar setColores

setTipoLapiz

6. Se crea una subclase que se llame **PaintBrushScrollView** en la cual se crean las salidas y acciones siguientes:

salidas	**acciones**
Caja_de_TamanoLapiz	setCaja_de_TamanoLapiz
	limpiar
	setTipoLapiz
	setTamanoLapiz
	setColores

7. Las conexiones se realizan únicamente dentro de la interface, y con ello se controla el programa gráfico, para esto es necesario abrir el inspector y seleccionar conexiones.
8. Se hacen las conexiones que se indican a continuación (Figuras 4.20). Recuerde que se oprime la tecla control y se arrastra el mouse hasta el objeto destino y en el inspector indicamos la salida que vamos a utilizar).

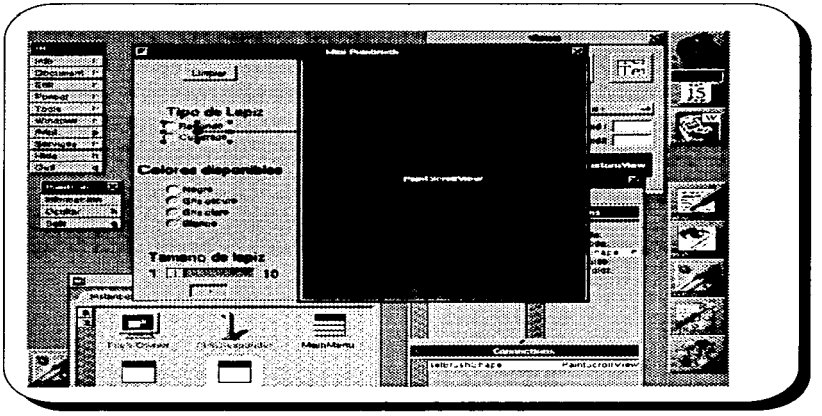
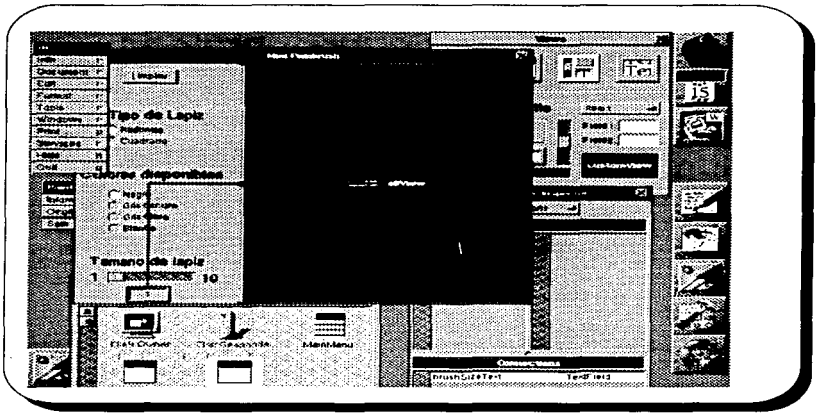


Figura 4.20 Conexiones

9. Se guarda el archivo .nib y se cierra el InterfaceBuilder.
10. En el ProjectBuilder se abre el archivo **PaintBrush1.h** y verificamos que se encuentren los métodos y las acciones que declaramos en el InterfaceBuilder y agregamos métodos de control de gráficos (drawSelf, mouseDown), recuerde que al crear un método en el archivo .h tiene que tener su correspondiente en el archivo .m.

11. Se edita el archivo **PaintBrush1.m** en él cual nuevamente vamos a introducir el código necesario para el control gráfico tanto del mouse como de los tamaños de lapiz y su color (el código se encuentra en las páginas subsecuentes.)



4.21 Conexiones

12. Se edita el archivo **PaintBrushScrollView.h** el cual modificamos la salida y agregaremos métodos.
13. Se edita el archivo **PaintBrushScrollView.m** el cual va a reconocer los movimientos del mouse, el tamaño del lapiz así como el color de la punta con la que vayamos a dibujar en la interface.
14. Se hace el icono representativo para la aplicación con el IconBuilder, el tamaño es de 48 x 48 pixeles, para a gregarlo se abre el inspector del ProjectBuilder y en el menú de atributo de proyecto se coloca el ícono.

15. Se compila el programa y aparece el archivo ejecutable en el directorio de la aplicación el cual ésta identificado por el icono que se le asignó anteriormente

4.2.2 Archivos de la aplicación.

PaintBrush1.h

```
#define        PUNTA_REDONDA          0
#define        PUNTA_CUADRADA         1
#define TAMANO_BITMAP    10.0
#define TAMANO_MITAD     TAMANO_BITMAP / 2.0
#define DOC_ANCHO        500.0
#define DOC_ALTO         500.0
```

PaintBrush1.h

```
#import <Appkit/View.h>

@interface PaintBrush1 : View
{
   id arrastreBitmap;
   id docBitmap;
   int tipoLapiz;
   float tamanoLapiz;
   float colores;
}

- limpiar;
- setTipoLapiz:(int)nuevoTipoLapiz;
- setTamanoLapiz:(float)nuevoTamanoLapiz;
- setColores:(float)nuevoColores;
- nuevoArrastreBitmap;
- mouseDown:(NSEvent*)theEvent;
- paintOneDrop:(NSPoint *)cursorLocation;
- drawSelf:(NSRect *)r:(int)contador;

@end
```

PaintBrush1.m

```
#import "PaintBrush1.h"
#import "PaintBrushParams.h"
#import <Appkit/Application.h>
#import <Appkit/Bitmap.h>
#import <Appkit/Window.h>
#import <DPSClient/wraps.h>

@implementation PaintBrush1

+newFrame:(NSRect *)tF
{
                NSRect docRect;
                self = [super newFrame:tF];
                tamanoLapiz = 1;
                tipoLapiz = PUNTA_REDONDA;
                colores = NX_BLACK;
                arrastreBitmap = [[Bitmap newSize:TAMANO_BITMAP :TAMANO_BITMAP
                                                        type:NS_UNIQUEBITMAP] setFlip:NO];
                [self nuevoarrastreBitmap];
                docBitmap = [[Bitmap newSize:DOC_ANCHO :DOC_ALTO
                                                    type:NS_NOALPHABITMAP] setFlip:NO];
                NSSetRect(&docRect, 0.0, 0.0, DOC_ANCHO, DOC_ALTO);
                [docBitmap lockFocus];
                NSEraseRect(&docRect);
                [docBitmap unlockFocus];
                return self;
}
- limpiar
{
                NXRect rect;
                [self lockFocus];
                NSEraseRect(&bounds);
                [self unlockFocus];
                NSSetRect(&rect, 0.0, 0.0, DOC_ANCHO, DOC_ALTO);
                [docBitmap lockFocus];
                NSEraseRect(&rect);
                [docBitmap unlockFocus];
        return self;
}

- setTipoLapiz:(int)nuevoTipoLapiz
{
                tipoLapiz = nuevoTipoLapiz;
                [self nuevoarrastreBitmap];
                return self;
}
- setTamanoLapiz:(float)nuevoTamanoLapiz
{
                TamanoLapiz = newTamanoLapiz;
                [self nuevoarrastreBitmap];
                return self;

}
```

```
- setColores:(float)nuevoColores
{
        colores = nuevoColores;
        [selfnuevoarrastreBitmap];
        return self;
}
- nuevoarrastreBitmap
{
        [arrastreBitmap lockFocus];
        PSsetstrokeadjust (NO);
    PScompositerect (0.0, 0.0, TAMANO_BITMAP, TAMANO_BITMAP, NS_CLEAR);
        PSsetalpha(1.0);
        PSsetgray(Colores);
        if (tipoLapiz == PUNTA_CUADRADA) {
                PSrectfill (0.0, 0.0, TamanoLapiz, TamanoLapiz);
    } else if (tipoLapiz == PUNTA_REDONDA) {
                PSnewpath();
                PSarc(TamanoLapiz/2.0, TamanoLapiz/2.0,
                    (TamanoLapiz/2.0)-0.5,
                    0.0, 360.0);
                PSclosepath();
                PSfill();
        }
        [arrastreBitmap unlockFocus];
        return self;
}


#define ARRASTRE    NX_MOUSEDRAGGEDMASK|NX_MOUSEUPMASK
- mouseDown:(NSEvent *) theEvent
{
        int antiguo_evento;
        NSEvent *ultimoEvento;

        [self paintOneDrop:&theEvent->location];
        antiguo_evento = [window eventMask];
        [window addToEventMask:ARRASTRE];
        while (YES) {
                ultimoEvento = [NSApp getNextEvent:ARRASTRE];
                if (ultimoEvento->type == NS_MOUSEUP) break;
                [self autoscroll:ultimoEvento];
                [self paintOneDrop:&ultimoEvento->location];
        }
        [window setEventMask:antiguo_evento];
        return self;
}
```

```
- paintOneDrop:(NSPoint *) cursorLocation
{
                NSPoint ventana;

                [self convertPoint:cursorLocation fromView:nil];
                ventana.x = cursorLocation->x - TamanoLapiz/2.0;
                ventana.y = cursorLocation->y - TamanoLapiz/2.0;
                [self lockFocus];
                [arrastreBitmap composite:NS_SOVER toPoint:&ventana];
                [[self window] flushWindow];
                [self unlockFocus];
                [docBitmap lockFocus];
                arrastreBitmap composite:NX_SOVER toPoint:&ventana];
                [docBitmap unlockFocus];
                return self;
}

-drawSelf:(NSRect *)r:(int) contador
{
                NSPoint ventana;
                NSRect fuente;
                int i;
                for (i=((contador == 3) ? 1 : 0); i<contador; i++) {
                                fuente = *r;
                                [docBitmap composite:NS_COPY
                                                fromRect:&fuente
                                                toPoint:&r->origen];
                                r++;
                }

                return self;
}

@end
```

PaintBrushScrollView.h

```
#import <Appkit/ScrollView.h>

@interface PaintBrush1ScrollView : ScrollView
{
   id Caja_de_TamanoLapiz;
}

+ newFrame:(const NSRect *)tF;
- set Caja_de_TamanoLapiz:anObject;
- limpiar:sender;
- setTipoLapiz:sender;
- setTamanoLapiz:sender;
- setColores:sender;
@end
```

PaintBrushScrollView.m

```
#import "PaintBrush1ScrollView.h"
#import "PaintBrush1.h"
#import "PaintBrushParams.h"
#import <Appkit/Control.h>
#import <Appkit/Matrix.h>

@implementation PaintBrush1ScrollView

+newFrame:(const NSRect *)tF
{
                NSRect docRect;
                id miVista;

                self = [super newFrame:tF];
                [[self setVertScrollerRequired:YES]
                setHorizScrollerRequired:YES];

                NSSetRect(&docRect, 0.0, 0.0, DOC_ANCHO, DOC_ALTO);
                miVista = [PaintBrush1 newFrame:&docRect];
                [self setDocView:miVista];
                return self;
}



- setCaja_de_TamanoLapiz:anObject
{
   Caja_de_TamanoLapiz = anObject;
   return self;
}


- limpiar:sender
{
                [[self docView] clear];
     return self;
}

- setTipoLapiz:sender
{
                int lapiz;

                lapiz = [sender selectedRow];
                [[self docView] setTipoLapiz:lapiz];
                return self;
}
```

```
- setTamanoLapiz:sender
{
                float tamanoLapiz;

                tamanoLapiz = (float)[sender intValue];
                [Caja_de_TamanoLapiz setFloatValue:tamanoLapiz ];
                [[self docView] setCaja_de_TamanoLapiz:tamanoLapiz ];
                return self;
}


- setColores:sender
{
                float arreglo_color[] = {NX_BLACK, NS_DKGRAY, NX_LTGRAY, NX_WHITE};
                [[self docView] setColores:arreglo_color[[sender selectedRow]]];
                return self;
}

@end
```

CONCLUSIONES

- La Programación Orientada a Objetos tiene actualmente mucho auge por que nos presenta una nueva visión del mundo debido a que sus orígenes se basaron en la premisa de poder simular problemas del mundo real.

- La Programación Orientada a Objetos es un conjunto de técnicas, que promueven la división de un problema en partes simples, utilizando como elemento básico a los objetos.

- La Programación Orientada a Objetos tiene como características: la herencia, el polimorfismo y la encapsulación.

- La Programación Orientada a Objetos permite la elaboración de clases a partir de otras, la modularidad, la reutilización.

- Un Sistema Operativo Orientado a Objetos ésta formado por un núcleo que tiene cierta jerarquía y módulos reemplazables entre los cuales no existe distinción entre los programas de utilieria y las aplicaciones.

- OpenStep es un Sistema Operativo Orientado a Objetos basado en la versión de UNIX conocida como MACH.

- OpenStep fue creado para el desarrollo de aplicaciones cliente-servidor logrando la optimización del tiempo gracias a que el código puede ser reutilizado, que maneja una interface amigable con el usuario, que es sencillo y rápido de desarrollar , lo que permite que las aplicaciones sean modificables a futuro.

- OpenStep puede ser un sistema operativo multiarquitecturas, como SUN, DEC, HP-UX, Intel, Motorola.

- OpenStep permite la administración remota, la utilización de una red virtual cliente/servidor de acceso remoto, manejo de conexiones a otras redes UNIX, conexiones a Internet o a redes Novell.

- OpenStep permite utilizar bases de datos y sistemas de información precisos y sencillos de manejar, los cuales puedan simular la realidad de una forma sencilla.

- OpenStep es un ambiente para el desarrollo de aplicaciones, consta de una interface gráfica con DisplayPostScript y herramientas de desarrollo como: InterfaceBuilder, IconBuilder y ProjectBuilder.

- Las herramientas de OpenStep permiten desarrollar sistemas de una manera sencilla y con una interface de usuario intuitiva y agradable.

- El Objective C es el lenguaje de Programación de OpenStep y ésta basado en el lenguaje C ANSI, el cual puede utilizar las librerias del lenguaje C así como las propias.

- Para la elaboración de programas cuenta con un inspector, el que muestra todas las caracteristicas que tiene un objeto, sus conexiones con otros Objetos, su color, tamaño, posición, así como sus funciones dentro del programa.

- La herramienta Interface Builder permite hacer de manera gráfica el desarrollo de la interface con la que trabajará el usuario, indicando en ella sus conexiones con otros Objetos, posición y comportamiento.

- El ProjectBuilder se utiliza para crear programas o aplicaciones conjuntamente con otras herramientas (IconBuilder, InterfaceBuilder, Objective C).

- El IconBuilder permite crear y/o modificar imágenes o iconos que pueden ser incluidos en un proyecto.

- OpenStep permite desarrollar sistemas de Intranet e Internet basados en Objetos, con la ayuda de Webobjects, lo que permite que podamos crear sitios Web y realizar modificaciones de una manera sencilla y rápida, utilizando en forma de objeto todos los elementos que la conforman.

BIBLIOGRAFIA

GARFINKEL, Simson L. Mahoney, Michael
" NextStep programming . Step one: Object-Oriented Applications. "
Springer-Verlag New York
Ed. Telos, 1993
631 p.

BACKLIN , Gene
" Developing NextStep Applications. "
Indianapolis, Indiana
ed. SAMS publishing, 1995
604 p.

" NextStep Object-Oriented programming and the Objective C language "
Developers Library, Next Computer.
ed. Addison-Wesley, 1993

BOOCH, Grady
" Object-Oriented Analysis and desing with applications "
Redwood City, CA
ed. Benjamin/Cummings publishing company, 1991

NGHIEM, Alex
" NextStep Programming concepts and applications "
Englewood Cliffs, New Jersey
ed. Prentice Hall, 1993

PINSON, Lewis J., Wiener, Richard S.
" Objective-C: Objective-Oriented programming techniques. "
Massachusetts
ed. Addison-Wesley, 1991

SHEBANEK, Michael
" The complete guide to the NextStep user environment "
New York, NY
ed. Springer-Verlag, 1993

" NextStep programming interface summary "
Developers Library, Next Computer.
ed. Addison-Wesley, 1993

" NextStep Development tools and techniques "
Developers Library, Next Computer.
ed. Addison-Wesley, 1993

" How to build Dynamic Websites "
Next Computer Inc.
1995

WEBSTER, Bruce F
" The Next Book "
ed. Addison-Wesley, 1993

" Sound, Music and signal processing on Next "
Developers Library, Next Computer.
ed. Addison-Wesley, 1993

YOURDON, Edward
" Object-Oriented Systems Desing "
Englewood Cliffs, Ney Jersey.
ed. Prentice Hall, 1994
400 p.

COLEMAN, Derek
" Object-Oriented Development. The fusion method. "
Englewood Cliffs, Ney Jersey.
ed. Prentice Hall, 1994
313 p.

BUDD, Timothy
" An introduction to Object-Oriented Programming "
Massachusetts
ed. Addison-Wesley, 1991
396 p.

POUNTAIN, Dick
" Retrato de un sistema operativo Orientado a Objetos "
Byte México. Año 9 No. 94
Noviembre de 1995.
pp. 34-38

RAMÍREZ, José Antonio
" OpenStep de Next Computer "
Byte México. Año 9 No. 89
Junio de 1995.
pp. 18-19

SEMICH, William J.
" Jobs´ NextStep is onto Intel PC´s "
Data Mation. Vol. 39 No. 10
Mayo 15, 1993
pp. 97-98

SEMICH, William J.
" Uncertain future: What´s the NextStep after Client/Server ? "
Data Mation. Vol. 40 No. 6
Marzo 15, 1994
pp. 26-34

OKTABA, Dra. Hanna
" Analisis y diseño Orientado a Objetos "
Soluciones Avanzadas. Año 1 No. 4
Julio-agosto, 1993
pp. 8-11

PÉREZ, José Raúl Cázares.
" La migración hacia la tecnología Orientada a Objetos "
Soluciones Avanzadas. Año 1 No. 6
Noviembre-diciembre, 1993
pp. 35-36

SÁNCHEZ, Dr. Antonio
" Postulados de los paradigmas de programación "
Soluciones Avanzadas. Año 1 No. 5
Septiembre-octubre, 1993
pp. 42-45

VALDERRAMA, Miguel
" Explorando la relación entre el paradigma OO y el reuso del software:
Un marco de referencia (Parte 1) "
Soluciones Avanzadas. Año 4 No. 28
Diciembre 15, 1995
pp. 39-46

QUINTANILLA, Gloria
" Abstracción de datos en lenguajes Orientados a Objetos "
Soluciones Avanzadas. Año 1 No. 5
Septiembre-octubre, 1993
pp. 25-31

QUINTANILLA, Gloria
"El impacto de la Ingeniería de Software de la Programación Orientada a Objetos"
Soluciones Avanzadas. Año 1 No. 3
Abril-mayo, 1993
pp. 43-46

GREIFF, Warren R.
" Paradigma vs Metodología : El caso de la programación Orientada a Objetos (Parte 1)"
Soluciones Avanzadas. Año 1 No. 6
Noviembre-diciembre, 1993
pp. 10-16

UDELL, Jon
" La POO se ha debilitado y el software de componentes se fortalece "
Byte México. Año 8 No. 77
Mayo, 1994
pp. 45-56

QUINTANILLA, Gloria. Silva, Sergio
" Elementos de la programación Orientada a Objetos "
Soluciones Avanzadas. Año 1 No. 4
Julio-agosto, 1993
pp. 5-7

OROZCO, Octavio
" Nuevas tecnologías para el desarrollo de aplicaciones "
Soluciones Avanzadas. Año 1 No. 6
Noviembre-diciembre, 1993
pp. 4-9

CERVANTES, Ofelia P.
" Bases de datos Orientadas a Objetos "
Soluciones Avanzadas. Año 1 No. 6
Noviembre-diciembre, 1993
pp. 28-31

RAMÍREZ, Miguel
" Orientación a Objetos "
Soluciones Avanzadas. Año 1 No. 6
Noviembre-diciembre, 1993
pp. 57-58

OKTABA, Hanna
" Análisis y diseño Orientado a Objetos: Introducción al método de Booch "
Soluciones Avanzadas. Año 1 No. 6
Noviembre-diciembre, 1993
pp. 18-22

OKTABA, Hanna
" Programación Orientada a Objetos: ¿ moda o realidad ? "
Soluciones Avanzadas. Año 1 No. 3
Marzo-abril, 1993
pp. 39-42

Editorial.
Soluciones Avanzadas. Año 1 No. 6
Noviembre-diciembre, 1993

Editorial
Soluciones Avanzadas. Año 1 No. 3
Marzo-abril, 1993

DERAMAT, Frederic
" Programación Orientada a Objetos desde el punto de vista de un Ingeniero "
Soluciones Avanzadas. Año 1 No. 4
Julio-agosto, 1993
pp. 38-40

RODRÍGUEZ, Guillermo. González, Jorge.
" Diseño Orientado a Objetos "
Soluciones Avanzadas. Año 1 No. 6
Noviembre-diciembre, 1993
pp. 24-27

SVEN, Kevin B.
" EOF: Una infraestructura coherente de sistemas objetos "
Byte México. Año 8 No. 87
Abril, 1995
pp. 54-56

RAMÍREZ, José Antonio
" Next y SunSoft presentaron la especificación OpenStep "
Byte México. Año 8 No. 86
Marzo, 1995
pp. 22

WEBSTER, Bruce F.
" Whither NextStep "
Byte. Vol 19 No. 11
Noviembre, 1994
pp. 289-290

RAMÍREZ, José Antonio
" Next es un Unix light "
Byte México. Año 8 No. 88
Mayo, 1995
pp. 22

SMITH, Ben
" NextStep para Intel "
Byte México. Año 6 No. 67
Agosto, 1993
pp. 26-32