UNIVERSIDAD APEC
UNAPEC
Facultad de Humanidades y Ciencias
Escuela de Informática

INTEGRACION DE LA PROGRAMACION ORIENTADA A OBJECTOS (POO) Y LA PROGRAMACION EN LOGICA MEDIANTE O-PROLOG.

Trabajo Monografico Final Para optar por el Titulo de
Ingenieria de Sistemas de Informacion

SUSTENTADA POR

Br. Rosalia Cristo Feliz

Mat. 86-0947

Santo Domingo, D. N.
AGOSTO, 1993

UNIVERSIDAD APEC

-UNAPEC-

FACULTAD DE CIENCIAS Y HUMANIDADES

ESCUELA DE INFORMATICA

PROGRAMACION ORIENTADA AL OBJETO COMO UNA HERRAMIENTA DE PORTABILIDAD ENTRE SISTEMAS OPERATIVOS

Curso Monográfico de Evaluación Final

PARA OPTAR POR EL TITULO DE

Ingeniero de Sistemas en Computación

SUSTENTADO POR:

Br. Dywer Santos Cruz

Mat. 89-0517

SANTO DOMINGO, D. N.

1993

Universidad APEC

Facultad de Humanidades y Ciencias
Escuela de Informática

Programación Orientada al Objeto como una Herramienta de Portabilidad entre Sistemas Operativos

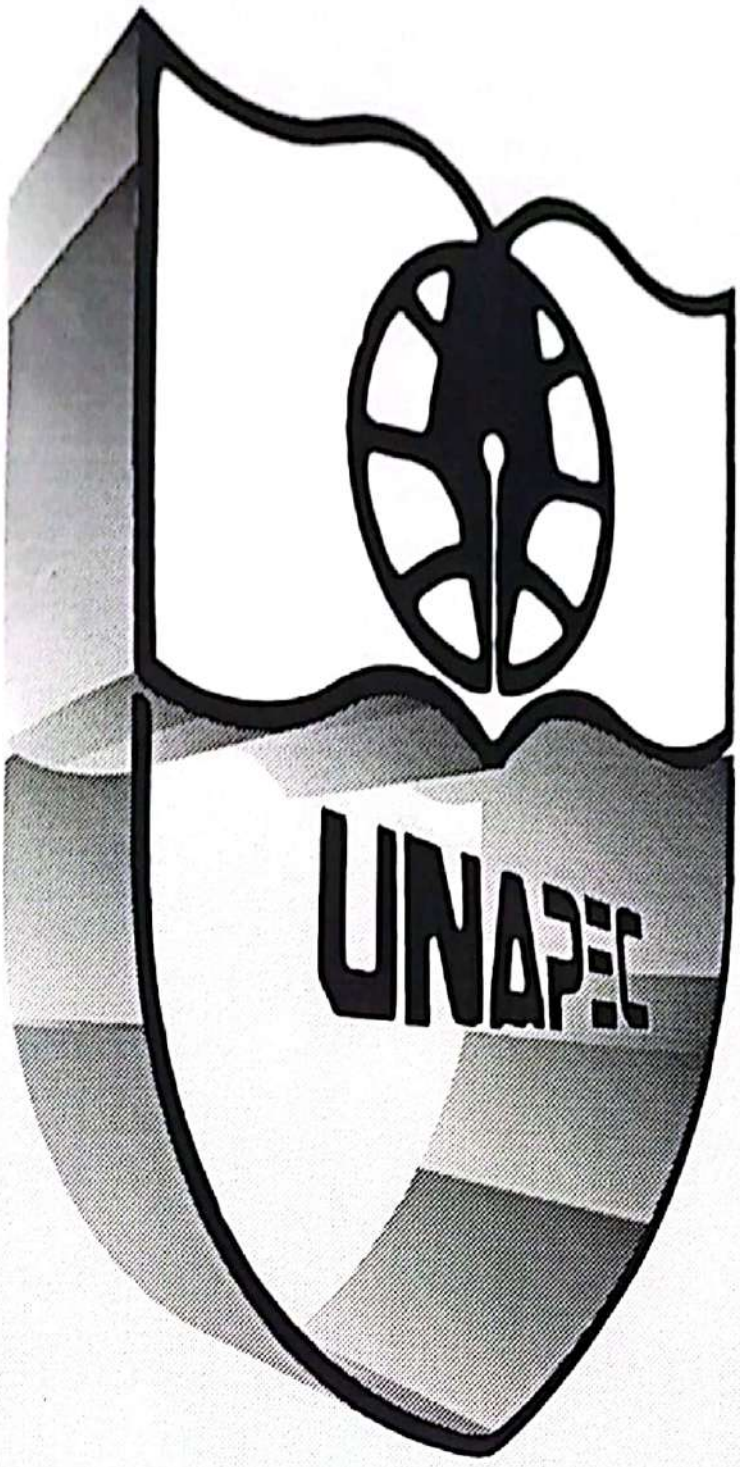
Curso Monográfico de Evaluación final
para optar por el título de:

Ing. de Sistemas de Computación

SUSTENTADO POR:

Br. Dywer Santos Cruz
Mat. 89-0517

Santo Domingo, D.N.
1993



PROGRAMACION ORIENTADA AL OBJETO COMO HERRAMIENTA DE PORTABILIDAD ENTRE SISTEMAS OPERATIVOS

INDICE

CAPITULO III
LENGUAJES DE PROGRAMACIÓN ORIENTADOS AL OBJETO

CAPITULO IV
PORTABILIDAD DE LOS SISTEMAS OPERATIVOS

CAPITULO V
LOS INTERFASES GRÁFICOS DE USUARIO Y LA PROGRAMACIÓN ORIENTADA AL OBJETO

DEDICATORIAS

DEDICATORIAS

A mis padres:
Feromina e Hipolito

El camino ha sido largo,
pero ustedes me han guiado para no desviarme.

He encontrado muchos obstáculos en la ruta,
pero ustedes me han ayudado a saltarlos.

He caído muchas veces,
pero siempre han estado cuando más los he necesitado.

Creo que no hace falta decir gracias,
porque cada cosa que hago es por ustedes
porque cada suspiro de mi vida se lo debo a ustedes.

Mamí, Papí los quiero mucho.

A mis hermanos:
Bynees y Emmy

Todos los días quisiera tener la oportunidad de tenerlos a mi lado. Les doy gracias por comprenderme, por ayudarme y por soportar mar de dudas que a veces que soy.
Espero que juntos sigamos adente para lograr nuestros mas anciados sueños

A mis compañeros de estudios:

A ustedes que juntos compartimos tantos momentos alegres y tristes, que nunca olvidaré, muchos ya somos como hermanos y espero que llegar a la culminación de nuestros sueños al terminar la Universidad no signifique que no nos volvamos a ver.

INTRODUCCION

INTRODUCCIÓN

Si tenemos equipos de arquitecturas diferentes, necesitaremos de herramientas que nos permitan salvar el problema de transferencia de datos. Las redes locales han sido un paliativo a esta situación. Sin embargo, no siempre compartir datos es la mejor solución. Esos datos deben ser interpretados por programas en otras máquinas que pueden no cumplir con las mismas normas. Lo ideas sería, que los datos sean interpretados por una aplicación parecida a la que los generó. Pero, surge la interrogante ¿como llevar mi programa de la máquina A a la B? La repuesta a esta pregunta puede estar en el concepto que encierra esta palabra "Portabilidad".

Portabilidad significa la última meta de la estandarización. Y no será una sorpresa que en un futuro no muy lejanos compremos las aplicaciones sin preocuparnos por el sistema operativo, ya que, compilando o simplemente corriendola en otra máquina esta dará los resultados que deseamos.

Aún hay mucho camino por recorrer. Hace apenas unos años que el término "portable" está sobre el tapete. En esto ha influido en gran medida, los interfases gráficos de usuarios que han llevado la computación a nuevas fronteras, y la competencia por el dominio del mercado de los sistemas operativos. De esta última, quienes han resultado más beneficiados son los usuarios; los cuales tienen a su

disposición aplicaciones de excelente calidad gracias a esa competencia. En la otra parte, están los programadores y fabricantes de software que ahora enfrentan el problema de darle soporte a más de una plataforma de computación.

Manejarse con varios sistemas operativos e interfases gráficos, no es una tarea fácil. Se requiere de elementos que permitan rehusar el código utilizado en una plataforma y que la labor de programación se vuelva más productiva.

Un concepto que se acopla perfectamente a lo antes dicho es el *diseño orientado al objeto*, porque permite acoplarse perfectamente a la programación bajo diferentes ambientes.

Ya podemos ver como aplicaciones utilizando los conceptos de portabilidad y diseño orientado al objeto comienzan a hacer su aparición en le mercado. La primera de estas no es nueva en el mundo de la computación, pero si en los micros, es Smalltalk, pero este sólo fue el concepto en el cuál Steve Jobs se basó al diseñar el poderoso ambiente donde se combina el diseño orientado al objeto y los interfases gráfico, NeXTStep. Con estas nuevas tecnologías al alcance de los usuarios, estamos seguros que la computación dará otro paso gigantesco en su desarrollo.

CAPITULO I
DESARROLLO HISTÓRICO DE LOS LENGUAJES DE PROGRAMACIÓN

CAPITULO I

DESARROLLO HISTÓRICO DE LOS LENGUAJES DE PROGRAMACIÓN

1.1 Generaciones de los lenguajes de Programación

Desde que el hombre comprendió su necesidad de contar, calcular o medir las cosas a su alrededor; nació una nueva ciencia, la cual a través del tiempo se convertiría en una necesidad.

Hoy la ciencia de la informática es considerada como un elemento imprescindible dentro de la vida moderna.

El objetivo de esta ciencia radica en facilitar actividades y procesos que antes o tenían un alto grado de errores o eran muy tediosos. Sin embargo, el nivel de perfección y eficiencia alcanzado por la informática no hubiese sido posible sin los aportes de miles de personas algunas de los cuales, jamas conoceremos sus nombres.

Como bien es sabido esta ciencia es el balance perfecto entre sus dos partes: Hardware y Software. De las dos la segunda, es a la opinión de muchos expertos, la que ha sufrido un desarrollo mas lento, ya que la parte física ha evolucionado de una

forma que nunca hubiesen, siquiera soñado, los creadores del primer computador eletrónico.

En nuestros días, la revolución esta guiada por el software, los usuarios quieren maquina fáciles de manejar; los programadores necesitan herramientas que le permitan sacar el máximo del equipo sin el elevado consumo de tiempo que esto requería anteriormente. Los interfases gráficos, las redes locales, los sistemas operativos portables, la programación orientada al objeto; han contribuido a un amplio desarrollo de la parte intangible de la maquina. Estas tecnologías han llenado las expectativas de los usuarios, programadores y fabricante. Toda esta revolución parecería una película de ciencia ficción en los años 40's y 50's cuando las computadora eran tan grande como un edificio y se programaban por hardware.

Lo beneficios que la ciencia de la informática ha traído al mundo son innumerables y estos seguirán creciendo en la medida que los usuarios acepten los nuevos cambios.

1.1.1 Características de los lenguajes de la 1ra. Generación

La primera generación de lenguajes de programación se puede referenciar del año 1954 al 1958.

Las características mas relevantes de estos lenguajes de programación fueron:

* Debido a que las aplicaciones que se desarrollaron eran puramente científicas, estos lenguajes incluían un alto grado de expresiones matemáticas en su sintaxis. Fortran I y Algol 58 son dos ejemplos claros de esta generación. Su sintaxis era puramente símbolos matemáticos, tanto así, que estas primeras generaciones de lenguajes no incluían saltos condicionales ni instrucciones de repetición.

* Otra característica de estos lenguajes era el ser imperativos. Estos solo se limitaban a decirle a la maquina "que hacer" en vez de como hacerlo, o sea, se cetraban menos en el análisis del problema y más en los resultados que se podían obtener.

* Eran limitados en las operaciones de E/S, porque el tiempo de proceso era consumido mayormente en el procesador por el uso de operaciones numéricas.

Su diseño e Implementación estaban dirigidos hacia el procesamiento de grandes cantidades de datos numéricos y no se pensó, en esa época, en la aplicación comercial

de las computadores. Sin embargo, lenguajes tales como Fortran I, hicieron posible que más persona tuvieran acceso a las computadoras, porque si bien su aplicación era limitada, estos ayudaron a disfrazar las complejidades del Hardware. Para esta época las computadoras eran máquinas más amigables y fáciles de programar. Ya no era necesario programar vía Hardware o utilizar un lenguaje ensamblador con el cual se estaba hablando casi al mismo nivel que el lenguaje de máquina.

* Un elemento que caracterizó notablemente a los lenguaje de esta época fue el mapeo finito (definiciones de arreglos) los cuales son imprescindibles para el cálculo de ciertas operaciones.

* Las variables eran globales y podemos ver en lenguajes como Fortran una ligazón implícita de tipo (enteras empiezan I-N y las reales con cualquier otra letra).

Solo al final de la década de los 50's, y como consecuencia de la naciente necesidad de utilizar el computador en el procesamiento de datos, es que se ve la aparición de lenguajes de programación como Cobol, cuyo diseño está orientado a la solución de los problemas que no podían realizarse con lenguajes puramente científicos.

1.1.2 Características de los lenguajes de la 2da. y 3ra. Generación

El exito alcanzado por el computador en el área científica, utilizado como un gigantesco calculador, pronto comenzó a despertar el interés de personas que veían sus grandes potencialidades. Fue así como nuevas aplicaciones de los lenguajes de programación y la necesidad de procesar informaciones que no eran científicas, produjo una nueva generación de lenguajes que podían ser utilizados, tanto para aplicaciones científicas como comerciales.

Para esta época las computadoras habían reducido significativamente su volumen y habían aumentado su poder de procesamiento y almacenamiento.

Las características principales de los lenguajes de programación de la segunda generación (1959-1961) son:

* La implementación de sub-rutinas y estructura repetitivas.
* Ligazón explcita del tipo de dato-variable.
* Manejo de archivos y grande volúmenes de información almacenada.
* Mejoras en las operaciones de E/S.
* Se utilizaba mas tiempo de proceso en los dispositivos periféricos que en cálculos.

Aunque el procesamiento de los datos era puramente Batch, estos lenguaje de programación representaron un paso de avance gigantesco, puesto que probaron que el computador podía ser utilizado en los negocios.

Un factor que ayudó notablemente a la difusión de los lenguajes de programación en estos años fue el uso de los sistemas operativos.

Los antecesores de esta generación muchas veces se veían en la necesidad de interactuar en lenguaje máquina por la carencia de un S.O. Esta facilidad alejaba, aún más, al programador de las complejidades del Hardware pues la creación de las estructuras de datos y la interacción con este parecían estar "Escondidas" al programador.

Para esta época estuvieron los puntos principales eran los lenguajes declarativos y la compilación externa al código ejecutable.

En la tercera generación de lenguajes podemos notar como característica sobresaliente el uso de la programación estructurada.

La programación estructurada tiene como finalidad la escritura de programas fáciles de leer y modificar además del uso restringido del (Goto).

Otro concepto importante es el de "modularidad". Debido a que las aplicaciones que se desarrollaban sobrepasan, en muchas ocasiones, el limite de codificación por programa, y el uso de equipos separados de desarrollo, los diseñadores implementaron la facilidad de llamar sub-programas (módulos) externos al programa principal. Para esto se añadieron ciertas facilidades tales como:

* Chequeo de tipo tanto dinámico como estático.
* Pasada de parámetro entre módulos.

Todas estas facilidades crearon una disciplina de programación estándar (Programación Estructurada) y fue posible la creación de grandes aplicaciones y grupos de trabajos. Estos lenguajes tuvieron su desarrollo entre el 1962 y el 1975.

1.1.3 Características de los lenguajes de la 4ta. generación (4GL)

La búsqueda por nuevas herramienta que simplifira el proceso de programación, haciendo más productiva esta labor, fue el punto central de los fabricantes de computadoras a finales de los 70's.

Reusar código, y crear prototipos para demostración fueron, el mayor problema para los desarrolladores de aplicaciones comerciales. Si bien, la programación estructurada facilitó "tener equipos de desarrollo grandes, no brindaba beneficios sustanciales términos de reusabilidad y ahorro de tiempo. A esto se suman las exigencias de los usuarios por aplicaciones más fáciles de usar. La respuesta a todos estas exigencia s encontró en los llamados (4GL).

Entre es las características más sobresalientes de los lenguajes de cuarta generación (citados por Dimitris N. Charafas, en su libro Forth and Fifth Generation lenguajes pagina 59) son:

* Creación rápida de prototipos.
* Eliminación del manejo, de archivos gracias al uso de los DBMS.
* Integración de los componentes del Software en un solo ejecutable.
* Más rápida codificación que los lenguajes tradicionales.
* El uso de sintaxis no procedural.
* El tiempo de aprendizaje es rápido tanto para el programador como para el principiante.
* Herramientas de trabajo para usuarios finales más sencillas de manejar.

Los lenguajes de 4ta. generación introdujeron el termino de "pensamiento visual", en el cual lo que usted ve es lo que tendrá (WYSIWYG).

Esto se logra, por ejemplo, en un DBMS (Data Base Management System) "Dibujando" los elementos que tendrá la pantalla (pop-up, menus, scroll bars, iconos, etc.) y a estos se le relacionan las tablas o archivos que contendrán los datos. Un 4GL traducirá esto elementos en "Código" listo para su ejecución o compilación.

La programación visual se ha descrito como colocar bloques y al final de la colocación la aplicación estará lista.

El ahorro en tiempo que esto significa es tal que muchas compañías utilizan sistemas de base de datos (DBMS) para el procesamiento de sus operaciones, y han dejado de usar lenguaje tradicionales como el COBOL; porque con los primeros, la petición de datos que cumplen con cierto criterio puede ser extraída escribiendo una línea, mientras que en cobol, hay que escribir, compilar, depurar y probar todo un programa.

Con lo anteriormente dicho podemos ver como en la evolución de los lenguajes hasta este punto se ha llegado a un nivel en el cual la persona que utiliza un

computador (usuario) se encuentra muy lejos de comprender lo que ocurre dentro de la máquina. En como si tuviéramos una venda que en vez de obstruirnos la visión nos ayuda a ver mejor.

1.2 Características y diferencia de la programación orientada objeto y los lenguajes tradicionales

Tradicionalmente, los programas son codificados teniendo en cuenta las siguientes reglas: Procedimientos, y algoritmos.

El concepto de objeto envuelve un nivel de abstracción y encapsulación más amplio. En vez de utilizar procedimientos esto se basan en el poder expresivo de las clases y los objetos.

La necesidad de programar bajo objeto es más notable cuando tenemos que crear aplicaciones muy grandes o complejas. Mientras que mediante el uso de los lenguajes tradicionales es posible alcanzar cierto nivel de complejidad; existen limites para estos.

En lenguajes orientados al objeto podemos ver como a cualquier nivel de abstracción, los objetos trabajan en forma cooperativa para alcanzar cierto comportamiento.

El diseño orientado al objeto representa una evolución no una revolución en el mundo de la programación; no deja atrás los avances del pasado, pero si crea nuevos basado en estos, y a medida que las técnicas de programación avanzan, se logra el objetivo principal que dio inicio a la era de la programación "Esconder las complejidades del Hardware sin sacrificar, el poder hacer lo que nos propongamos con este".

En conclusión, Grady Booch explica (en su libro Object Oriented Oriented Design with Aplications) que las diferencias entre un lenguaje orientado al objeto y los tradicionales es que, los primero utilizan los módulos como su elemento básico para crear los construcciones, mientras que los orientados al objeto representan una colección lógica de clases y objetos en vez de utilizar sub-programas. También, en los OOP. existe muy poca o ninguna globalizacion de datos, en vez de esto las operaciones y los datos están unidos en una forma lógica de hacer las construcciones y no en algoritmos.

"Si los procedimientos y funciones son verbos y los priezas de datos son nombres, un programa orientado a los procedimientos está organizado alrededor de los verbos, mientras que el lenguaje orientado al objeto está organizado al rededor de los nombres" (Courtis, P. On Tine and Space Decomposition of Complex Structures. citado por Grady Booch en su libro Object Oriented Design with Aplications).

CAPITULO II
ELEMENTOS DEL DISEÑO ORIENTADO AL OBJETO

CAPITULO II

ELEMENTOS DEL DISEÑO ORIENTADO AL OBJETO

2.1 Concepto de objeto

Los seres humanos comprendemos el significado de la palabra objeto y los aprendemos a identificarlos desde que tenemos uso de razón.

Partiendo de la conceptualización que tenemos de lo que son los objeto podemos decir que un objeto es:

* Algo tangible o abstracto.
* Algo que se puede aprender o comprender.
* Algo que tiene una acción bien definida.

Grady Booch, en su libro Object Oriented Design with Aplications nos dice "los objetos que vemos en el mundo real no son los únicos de interés para el diseño de Software. Otra clase importante de objetos son aquellos que son el producto de la invención de un proceso en los cuales existe un alto nivel de comportamiento". Smith y Tockey, sugieren que "un objeto representa una unidad identificable pertenece o no al mundo real con un rol bien definido dentro de cierto problema".

Grady Booch define los objetos como "algo que tiene un estado, un comportamiento y una identidad".

Tomando estas dos definiciones hemos construido una definición de objeto:

"Un objeto es una entidad con limites y característica propias, mostrando un comportamiento estable y puede ser un ente abstracto al mundo real".

Cuando hablamos de proceso nos refierimos a que no necesariamente tiene que ser una entidad real. Por ejemplo un consumidor o un usuario es una entidad real, pero abrir un archivo, guardar ciertos datos en memoria o sumar dos registro son acciones abstractas.

Los objetos tienen limites definidos. Por ejemplo automóvil consta de un motor, carrocería, rueda, etc. Todos esos elementos son objetos que trabajan en común, pero tienen comportamientos y limites bien definidos nunca se observara a las ruedas haciendo la función de motor; y mucho menos al motor haciendo de carrocería.

2.2 Orígenes

El concepto "Objeto" emergió de forma casi independiente a principio de los 70 para referirse a elementos diferentes en su apariencia; pero relacionados mutuamente. Estas nociones fueron inventadas para manejar las complejidades del software en los cuales los objetos representan componentes de modularidad.

Levy añade los siguiente elementos como influencias en el diseño orientado al objeto y su implementación en software.

* Avances en las arquitectura de las computadora, incluyendo el soporte del hardware para sistemas operativos.
* Los avances en los lenguajes de programación como SIMULA, SMALLTALK, CLU, ADA.
* Avances en los sistemas relaciones de base de datos.
* Los investigaciones en inteligencia artificial.

El concepto de "Objeto" tiene sus orígenes hace mas de veinte años atrás (en Hardware) para cambiar el modelo VON-NEWMAN y para cerrar las distancias entre los lenguajes de programación y los bajos niveles de abstracción de la maquina.

Tal vez la mas importante contribución al modelo orientado al objeto proviene de los llamados lenguajes orientados al objeto o basado en objetos. El primero de estos fue en el lenguaje de programación Simula-67 a este le siguió SmallTalk-72, 74 y 76 y la más reciente versión el SmallTalk-80.

La primera persona que formalmente identifico la importancia de los compones los sistemas en objeto que Dijikstra en 1970. Pero mas luego introdujo la idea de encapsulacion. En los años 70's un numero de investigadores, mas notablemente Liskov y Zilles quienes fueron los pioneros en el desarrollo de la teoría de abstracción de datos.

En el campo de la inteligencia artificial, los desarrollos hechos en la "teoría del conocimiento" han contribuido al entendimiento del diseño orientado al objeto. en 1975 Miskey (citado por Grady Booch en el libro Object Oriented Design with Aplications). Propuso la teoría de los "Marcos" (Frames) para representar objetos de la vida real. Desde entonces, esta teoría ha sido utilizada como la base de la arquitectura de representación de los sistema de inteligencia artificial.

2.3 Clases

El concepto de clases y objeto están tan ligados uno del otro que seria imposible hablar de ellos independientemente. Un objeto es una entidad con característica propias y que existe en tiempo y espacio; la clase es la "esencia" de un objeto y como es.

Una clase se puede definir como "un grupo de objeto que tienen características similares".

Partiendo de esta definición, un "objeto es una instancia de una clase".

Grady Booch define las clase como "un conjunto de un objetos que comparten estructura y comportamiento".

Mayer y Snider (citados por Booch) sugieren que la programación consiste en la descomposición de un problema en unidades mas pequeñas y fáciles de manejar. Esta es la idea fundamental del diseño de la clases. Mientras que los objetos realizan acciones de forma individual; la clase captura el comportamiento y la estructura de todos los objetos relacionados entre si. La clase sigue como la ligazón entre los

objetos y su abstracción. Un lenguaje de programación que implementa el Strong Typing puede detectar violaciones del interfaz de la clase.

El interfaz de la clase, es la vista externa de esta y enfatiza en la abstracción mientras esconde la estructura y comportamiento de esta. Este interfaz consiste en la declaración de todos las operaciones aplicables a la instancia de la clase, pero puede incluir la declaración de otras clases, constantes variables y excepciones necesitadas para completar la abstracción. El otro interfaz es el de implementación que es la vista interior de esta, la cual pone de manifiesto los secretos del comportamiento de la clase.

La implementación de la clase consiste en la implementación de todos las operaciones definidas dentro de esta.

El interfaz de clase se puede dividir en 3 partes:

* Publico: La declaración que forma parte del interfaz de clase a todos los objetos visibles a esta.

* Protegida: La declaración del interfaz de la clase no es visible a otras clases solo a su sub-clase.

* Privado: La declaración que forma parte del interfaz de clase no es visible a ninguna otra clase.

2.4 Abstracción

La abstracción proviene del reconocimiento de características similares entre ciertos objetos, situaciones o procesos en el mundo real y la decisión de concentrarse en esas características e ignorar sus diferencias. La definición que Shaw nos da es la siguiente "una descripción simplificada, o especificación de un sistema enfatizando solo en algunos detalles y dejando a un lado otros."

Booch no da la siguiente definición: La abstracción denota las características esenciales de un objeto que lo distinguen de los demás, y provee una idea clara de sus delimitaciones, relativas a la perspectiva del observador". Partiendo de estas definiciones podemos decir que la abstracción consiste en la identificación de las características de un objeto, que en determinado momento, tiene cierto interés para el observador. La forma como un comprador de un automóvil lo ve muy distinto a

la forma como lo es el mecánico. Mientras uno se centra en la belleza estética; el otro centrará su atención a la forma como fue diseñado, si tiene un motor potente, etc.

Sidewitz y Stark sugieren que hay varios tipos de abstracción, desde objetos que tienen muy ligados sus entidades al dominio del problema, hasta de algunos que no tienen razón de existir". Desde el mas útil hasta el menos útil hay ciertos tipos de abstracción que son la siguientes:

* Abstracción de entidad: Un objeto que representa el dominio total del problema.
* Acción de abstracción: Un objeto que provee un conjunto generalizado de operaciones y todos realizan la misma función.
* Abstracción por una maquina virtual: Un objeto que agrupa juntos todas las operaciones utilizadas por un nivel superior de control.
* Abstracción concidencial: Un objeto que empaqueta un grupo de operaciones que no tienen ningún tipo de relación.

En el vocabulario de la abstracción tenemos los cliente, que es un objeto que usa los recursos de otro.

Este concepto forza al programador a concentrarse en la vista exterior del objeto. El conjunto de operaciones que realiza un objeto es llamado protocolo. El protocolo denota la forma en que un objeto actua y reacciona; y el mismo puede constituirse como una entidad dinámica o estática desde el punto de vista exterior de la abstracción.

La abstracción puede tener propiedades estáticas o dinámicas dependiendo de la ligazón del objeto y sus atributos. Por ejemplo una variable tiene un nombre que puede ser siempre el mismo (un valor estático); pero a su vez tiene atributos como la dirección que pueden ser cambiado (valor dinámico).

Lo esencial en la abstracción es tomar las característica relevantes de un objeto para lograr los fines que nos proponemos.

2.5 Encapsulacion

Luego de la abstracción de un objeto precede la forma como serán implementados las características abstraídas del mismo. Ingals, citado por Booch, no dice "Ninguna parte de los sistema complejos debe depender de otra". Mientras la abstracción "ayuda a las persona a pensar acerca de que están haciendo", la encapsulacion

"permite que los cambios en un programa puede ser cambiado de una forma rápida y confiable".

La abstracción se encuentra en la vista exterior de lo objetos y la encapsulación prevee que otros clientes puedan ver dentro de un objeto. La encapsulación provee barreras entre diferentes abstracciones por ejemplo: Los lenguajes de programación tienen un alto nivel de abstracción entre el usuario y la máquina. Los usuarios no le interesa conocer la forma como físicamente se actualiza un registro; pero si la forma como deben pedirle a la maquina que lo haga (Vista lógica versus vista física). En este caso, los objetos a un alto nivel de abstracción están protegidos contra los que tienen un bajo nivel de implementación en sus detalles.

Liskou, citado por Booch, sugiere que "para que la abstracción trabaje, debe ser implementada la encapsulacion". En la practica esto significa que, cada clase debe tener dos partes un interfaz y una implementación. El interfaz de una clase captura solo su vista exterior, teniendo relación con el comportamiento común a todas las instancias de la clase (a todos los objeto pertenecientes a esta clase).

La encapsulación es el proceso mediante el cual se esconden todos los detalles que no contribuyen a la correcta identificación de un objeto.

Booch afirma que esta puede esconder la representación de un objeto así como sus métodos de implementación.

2.6 Modularidad

Los módulos no son un concepto nuevo en programación. Esto causaron un gran revuelo cuando en los años 70's, la programación estructurada proporcionaba este mecanismo para la programación de sistemas complejos. Algunos autores afirman que esta es una de las mejores vías para disminuir el grado de complejidad.

Modularidad es "La propiedad que tienen los sistemas de descomponerse en partes libres y con cierta coordinación entre estas"

En algunos lenguajes como SmallTalk no existe el concepto de modulo teniendo como único mecanismo de descomposición la clases. En otras como Object Pascal, C++ y CLOS el módulo es una construcción separada del lenguaje y esto ofrece la garantía de tener libertad de un grupo de decisiones y operaciones totalmente independientes del programa que llama el módulo. En estos lenguajes los objetos y las clases forman parte de la estructura lógica del sistema.

La modularidad ha sido implementada de diferentes maneras por los lenguajes de programación. Por ejemplo en C++ los módulos son archivos precompilados. La práctica usual en C++ es incluirlos archivos como header files nombrados con la extensión .h. En Object Pascal los módulos son llamados unidades. En ADA son llamados paquetes y tienen dos partes que son la de especificación y el cuerpo del paquete.

En el diseño estructurado tradicional, la modularidad consiste en agrupar sub-programas utilizando cierto criterio de unidad. En los lenguajes orientados es diferente, la tarea es decidir como empaquetar físicamente los objetos y las clases que son muy diferentes.

El objetivo es implementar módulos (clases y objetos) sin afectar el comportamiento de otros. Con esto se logra que el tiempo de modificación y recompilación sea menor.

2.7 Jerarquía

En un programa pueden aparecer varias clases anidadas entre sí. Se pueden ligar diferentes abstracciones a un tiempo y hay que determinar cuál de ellas lleva el control sobre la otra. La jerarquía es el orden correcto de las abstracciones. La

herencia es el tipo más importante de jerarquía. Se define como la relación entre dos clases, en las cuales una comparte la estructura o comportamiento de la otra o de otras clases y puede ser simple o múltiple.

Sin la implementación de la jerarquía, cada clase podría ser una entidad independiente y ninguna de ellas tendría relación con otra. La consistencia de la jerarquía forma parte de la disciplina de cada programador. La herencia hace posible que se definan nuevas partes del sistema de forma modular.

Dansforth, nos explica que "la abstracción de datos provee una barrera entre como los métodos son ocultados. Para cada clase hay dos tipos de clientes: los objetos que invocan las operaciones a instancias de la clase y sub-clases que son herencias de una clase. La sub-clase debe accesar una instancia variable de su superclase ó llamar una operación de esta. En estos casos es que la jerarquía entre una y otra clase toma un papel crucial, pués se debe determinar cual es la subclase y la superclase para que al momento de compartir variables no se provoque un cruce de información. El lenguaje de programación debe proveer una protección precisa entre ambas clases.

CAPITULO III
LENGUAJES DE PROGRAMACIÓN ORIENTADOS AL OBJETO

CAPITULO III

LENGUAJES DE PROGRAMACIÓN ORIENTADOS AL OBJETO

3.1 Smalltalk

Smalltalk es un lenguaje y un ambiente de programación al mismo tiempo. Es el resultado de más de una década de desarrollo y investigaciones hecho por Software Concepts Group del la Xerox en los laboratororios de Palo Alto Research (PARC). Smalltalk es un poderoso sistema orientado al objeto. Este no solo ha afectado a la familia de productos de la Xerox sino que ha tenido gran incidencia en las áreas académicas y comerciales. Su influencia en la Apple Lisa y en el Apple Machintosh es ampliamente reconocida, así como en el diseño de herramientas CAD, de Autoedición y automatización de oficinas. Muchas de las hoy entusiastas tecnologías de alta resolución en gráficos, interfases gráficos de usuarios y computación personal han sido inspiradas por este ambiente de programación.

Hasta recientemente, el valor Smalltalk había sido provado en conceptos explotados de otros sistemas, no como un importante sistema de programación comercial. Con la introducción de este producto en las computadoras personales, el

costo de su uso se redujo sustancialmente y el mercado comenzó a descubrir como Smalltalk podía competir en el tradicional rango de soluciones de programación.

Desde sus inicios, los diseñadores de Smalltalk enfatizaron en un número de innovativas y controversiales ideas acerca de lo que la computación podría ser, como son:

1. Computación Personal: Los computadores son baratos, las personas son caras. Las computadoras podrían no compartir informaciones entre sí. Dándole a cada persona su propia máquina multiplicaría el poder de computación disponible permitiendo un alto grado de eficiencia.

2. Integración: El ambiente de computación debería ir tan lejos que provea inmediata retroalimentación a todas las acciones de los usuarios.

3. Gráficos: La gente se adapta más rápidamente a las cosas que puede identificar de forma visual.

4. Programación Orientada al Objeto: La gente trabaja con conceptos del dominio del problema, mientras que el hardware trabaja diferentes

conceptos. Hay que transmitir el dominio del problema al dominio del computador.

Smalltalk es el sinónimo de varias capas de abstracción las cuales juntas pueden implementar estos puntos. Desde sus inicios, los diseñadores de Smalltalk fueron inspirados por la visión futurista de Alan Kay, lo que el llamaba el Dynabook. Esto es un poderoso computador persona, capaz de manejarse como la más potente estanción de trabajo existente, y tan pequeña como un notebook. Smalltalk fue diseñado para ser el lenguaje de programación de computadoras con ese poder.

Máquina Virtual

En Smalltalk, un objeto es un dato privado y una colección de procedimientos necesarios para accesar dicho dato. El dato es privado al objeto, y no puede ser accesado sin la ayuda de los procedimientos de dicho objeto. Los procedimientos son públicos a todos los clientes, quienes tienen acceso al objeto escribiendo mensajes. Un mensaje es una expresión que le dice al objeto que debe hacer. El objeto responde al mensaje seleccionando y entonces ejecutando un procedimiento en donde todos los objetos de la misma clase ejecutan el mensaje. Este procedimiento es llamdado método, y el comando que lleva el mensaje es llamado selector.

Smalltalk adhiere este principio de forma exclusiva. Literalmente, todo es un objeto - no solamanete las ventanas o los íconos.

La mayor ventaja de Smalltalk es su consistencia, la cual es observada a través de todo el sistema. El sistema completo esta organizado e integrado por un sólo principio, de esta manera, el usuario necesita solamente comprender este principio para explorar cualquier parte de Smalltalk. Una vez se ha aprendido el uso del debugger (el inspector en Smalltalk) para examinar un objeto, se conoce todo acerca de como inspeccionar enteros, reales, stacks, heaps -literalmente todo el sistema.

Smalltalk provee automáticamente la facilidad del *Garbage Collection*, que significa que la vida de todos los objetos es determinada, no por el programador, sino por el sistema. Cuando un objeto no es referenciado más, su espacio de memoria queda libre.

Para lograr esta eficiencia, y para permitir la compactación del espacio libre de memoria de forma fácil cuando se vuelve fragmentado, Smalltalk usa un poderoso identificador. Con este, los objetos son identificados directamente, por su dirección de memoria. Los objetos que son identificados por un offset se encuentra dentro de la tabla de descriptores del objeto. Esta implementación esconde el manejo de

memoria al programador, dejando que el sistema la maneje de la forma más conveniente.

Todo es un objeto, inclusive pequeñas estructuras de datos como los enteros simples. Los beneficios del Garbage Collection automático no son pequeños, pues este elimina una clase completa de instrucciones que podrían crear conflictos en nuestros programas.

Pero el precio por todos estos beneficios es alto. En combinación con el extensivo uso de mensajes dinámicos, puede causar costos monumentales en cuanto a recursos de hardware se refiere.

Lenguaje de programación

La simplicidad de Smalltalk y su máquina virtual permite que su lenguaje de programación también sea inusual. La complejidad de definir estructuras de datos es totalmente innecesaria porque todo es de un solo tipo, un objeto. Tampoco es necesario el uso de estructuras de control, porque estas son logradas vía mensajes. En general, Smalltalk solo define sintaxis para:

1. declarar los nombre de los objetos y asignarles valores

2. Enviar mensajes, y

3. Definir nuevas clases y métodos.

Ambiente de Programación.

El lenguaje de programación, los editores, linkers, debuggers y la operación del sistema en sí están integradas, en el mismo espacio virtual.

Este ambiente enfatiza en la creatividad personal. Todo está al alcance del programador, inclusive cosas que nunca soñarian cambiar, como el sistema o la representación de los numeros reales. Ninguna de estas cosas están protegidas. La facilidad de cambiar cualquier elemento de Smalltalk es en principio para ofrecer un ambiente custumizado a cada usuario.

Un usuario de Smalltalk trabaja en una pantalla de alta resolución, utilizando una herramienta llamada el browser para explorar y posiblemente modificar cualquier parte del sistema. El browser es parecido a un editor de texto, en el cual se pueden desplegar líneas de código para edición o lectura. Es también la vía principal para correr el compilador.

El browser también provee facilidades para que los miembros de un equipo de trabajo puedan comunicarse entre todos los miembros.

3.2 ADA

Desarrollado para el departamento de defensa de los Estados Unidos de América. ADA fue diseñado para ser el reemplazo de una colección diversa de lenguajes antes utilizados por dicha agencia. Ada seria el lenguaje mandatorio para sistemas computacionales integrados y de relativamente, pocos componentes. Sus principales aplicaciones para el departamento de defensa de los Estados Unidos eran: control de superficies, sistemas de defensa, sistemas de navegación, comunicaciones, etc.

Ada ha sido siempre un lenguaje controversial. Sus diseñadores han sido acusados de crear un lenguaje de programación extemadamente complicado. Pero Ada ha demostrado ser tan poderoso como complicado. Cualidades tales como multitarea y manejo de excepciones como parte del lenguaje y los "paquetes" prueban el poder de este.

Cualidades Orientadas al Objeto de ADA

Las complejidades de Ada por lo regular, oscurecen el hecho de que Ada es básicamente un lenguaje de programación convencional. Ada no contiene nada que pueda ser una sorpresa para los usuarios de C o Pascal quienes están acostumbrados a las cualidades de multitarea y manejo de excepciones mediante el uso de librerías. Ada implementa el concepto de Strong Typing (chequeo de tipos estáticamente) y realiza todos los tipos de ligazón al momento de la compilación del programa. Los tipos pueden ser declarados explícitamente, y las decisiones son hechas en tiempo de compilación no en el run-time.

En Ada, la encapsulación es hecha por mecanismos proveídos totalmente por el compilador. Esta es la mayor debilidad y fortaleza de Ada. Es su mayor fortaleza porque el propósito para el cual Ada fue diseñado; proveer un lenguaje de programación en el cual se eliminara el uso de múltiples lenguajes para crear aplicaciones convencionales. El tipo de datos en cada componente pude ser conocido en un determinado punto, porque el compilador puede chequear que cada componente cumple con el tipo de datos especificado. Es su mayor debilidad porque hay problemas en los cuales los cambios no puedan hacerse editando el código fuente y recompilando. El factor que más distingue a Ada es que el programa debe estar bien diseñado antes que este alcance el tiempo de compilación.

Paquetes

Es la reprepresentación que Ada provee a la encapsulación. El paquete contiene partes que son privadas al suplidor, y otras que pueden ser accesadas al consumidor del código. Típicamente, el código y la vista precisa de los campos en un objeto son guardados como privados, pero el suplidor puede decidir que hacer publico cuando lo crea necesario.

Otra característica de Ada es la sobrecarga de operadores. Por ejemplo el signo de + puede ser utilizado tanto en datos del lenguaje (built-in data types) como en datos creados por el usuario.

Una de las limitaciones de Ada es la definición de nuevos tipos de datos reusando algunos de los existentes. Ada no provee sub-tipos de datos y datos derivados, sólo hay tipos generales de datos, como son los enteros; aunque si es posible de extender los datos internos del lenguaje añadiendo campos a las operaciones.

El concepto de paquete si es un mejoramiento de las definiciones hechas por los lenguajes tradicionales como C. Se puede comparar esta implementación con el movimiento de un lenguaje no estructurado como FORTRAN a uno estructurado

como C, pero menos que los beneficios que pueda brindar, el moverse de assembler a FORTRAN.

Ligazón estática de tipos, manejo de excepciones y una complicada sintaxis rica en tipos predefinidos son los elementos característicos de Ada.

3.3 C++

C++ es el producto de mejoras hechas al C, desarrollado por los laboratorios Bell. C++ se diferencia del Objetive-C, porque el primero se le añadieron elementos que contrastan con el lenguaje, mientras que el segundo nació siendo lenguaje orientado al objeto.

C es el lenguaje de programación estándar de la compañía AT&T y es utilizado principalmente para crear sistemas telefónicos; sistemas que deben ser reparados y que tienen un tiempo de vida relativamente alto (40 años o más).

Debido a su gran aceptación, los laboratorios Bell han retenido a casi todas las personas que han tenido influencias en la definición e implementaciones de C. Entre este selecto grupo tenemos a Ken Tompson (mejor conocido como el inspirador del sistema operativo UNIX y el inventor de Belle, un computador campeón de ajedrez,

también el inventor del lenguaje B, antecesor del C). Dennis Ritchie (inventor del primer lenguaje C), Steve Johnson (creador del compilador portable de C, en el cual se basan todos los modernos compiladores de este lenguaje), Brian Kernigahn (co-autor junto a Ritchie, del libro que durante años fue la única referencia del lenguaje C), y muchos otros incluyendo a Bjarne Stroustrup, la fuerza primaria detrás del C++.

Este grupo de talentos hacen de los laboratorios Bell los únicos capases de asegurar a C como el soporte vital de la programación comercial moderna. Con C++ se ha querido alcanzar ciertos puntos cruciales como:

1. Retener el alto grado de eficiencia y portabilidad que ha hecho famoso a C.

2. Retener la compatibilidad entre C y C++

3. Reparar las quejas acerca del tratamiento de tipos en C. C ha sido siempre criticado por su pobre chequeo de tipo inclusive dentro de una función dada, y no chequeo de tipo entre funciones aunque estén dentro del mismo archivo.

4. Actualizar a C con conceptos modernos de ocultación de datos.

C++ es un superset del lenguaje C, con unas pocas modificaciones que son incompatibles como son: la habilidad de definir nuevos tipos de datos (clases), operaciones en esos tipos de datos (operadores y funciones), y la facilidad de tener varias vías de control sobre esos tipos, incluyendo sobrecarga de operadores, constructores y destructores.

Clases

Todos los tipos nuevos de datos en C++ son definidos por las clases, y solo se mantiene la estructura como forma de definición de datos para compatibilidad (una estructura es tratada como una caso especial de la clase en la cual todos sus miembros son públicos).

La única diferencia entre una clase y una estructura es que en la estructura todos los miembros son públicos por defecto, mientras que los miembros de la clase son privados por defecto si no se especifica lo contrario. Mientras que las estructuras solo pueden tener miembros que son datos, las clases pueden tener miembros que sean procedimientos, o funciones. Estas funciones pueden ser de dos tipos: friend y member.

Friend:

Las funciones friend son convencionales en C. Ellas no tienen ninguna conección particular con las clases, excepto que permiten referenciar cualquier miembro de dato que haya sido declarado como privado.

Member:

Las funciones tipo member proveen la orientación al objeto de C++. Ellas implementan operaciones sobre las instancias de la clase. Mientras que los miembros son estándar funciones de C que han sido permitidas para referenciar miembros privados por su nombre. Las funciones tipo member su declaración está dentro de la declaración de la clase junto con la declaración de los miembros variables.

Manejo de Mejoría

En C++ nuevos objetos pueden se localizados en el heap y referciarlos por su dirección. Los objetos pueden ser llevados de forma estática y pueden referenciarse por su nombre.

Compilación Separada

Cada compilación ejecutada en C++ es hecha de forma aislada, el compilador no retiene en la memoria ningún elemento de esta, así que todas las declaraciones

acerca de clases externas deben ser manipuladas dentro de cada programa fuente. C++ no provee otra asistencia para hacer esto que el #include.

En la práctica dos archivos deben ser preparados para cada clase. El primero es el archivo de las declaraciones que debe estar incluido en cada archivo que utiliza esa clase. El segundo es el archivo de definiciones, el cual es compilado, archivado y combinado dentro de la imagen ejecutable por el linker.

3.4 Objetive-C

Es una herramienta para escribir programas que envuelven objetos y mensajes. También es una herramienta para expresar el tipo de programas que los usuarios de C habían estado esperando por años. Es un híbrido que contiene todo lo que es C y las mejores partes de Smalltalk.

Objetive-C trabaja igual que un compilador de C. El programador desarrolla el código fuente con un editor, lo compila a binario con un compilador de Objetive-C, linkea con un linker estándar, y lo depura y prueba igual que cualquier otro programa en C. Es permitido, el uso de archivos convencionales de C y la compilación es hecha como si no existiera un compilador Objetive-C dentro de este. Otros archivos deben tener un alto uso de las construcciones orientadas al objeto.

Una de las características principales de Objetive-C es que se puede obviar el mecanismo orientado al objeto para accesar la información privada de un objeto directamente. En esta característica es que se encuentran todas las debilidades de este lenguaje.

Es un lenguaje con ligazón estática para obtener un alto grado de eficiencia en la máquina, además de implementar strong typing.

3.5 LISP

El LISP es un lenguaje de programación diseñado para trabar en inteligencia artificial. Hasta hace poco estaba restringido a un pequeño grupo de investigadores, pero actualmente cuando los proyectos de computadores de quinta generación son una realidad, el LISP se está empleando de forma mucho más amplia.

Desarrollo

El LISP se desarrolló en 1959 en el grupo de inteligencia del MIT bajo la dirección de John McCarthy. Está diseñado para resolver problemas de manipulación de símbolos recursivos. La mayor parte del trabajo que se desarrolla en inteligencia artificial se centra en estos aspectos. De ahí la popularidad del LISP en esta área.

Objetivos

El espíritu del LISP es el de proporcionar un medio para escribir programas basados en la manipulación de símbolos. Las tareas más comunes cuando se desarrollan programas en inteligencia artificial son de generar y comprobar alternativas y las búsquedas de determinada secuencia en grandes conjuntos de datos. El LISP es un lenguaje pequeño y conciso, ideal para estos propósitos.

CAPITULO IV
PORTABILIDAD DE LOS SISTEMAS OPERATIVOS

CAPITULO IV

PORTABILIDAD DE LOS SISTEMAS OPERATIVOS

4.1 ¿Qué es Portabilidad? y ¿Cuándo es necesaria?

El desarrollo de aplicaciones solía ser relativamente fácil en los tiempos en que las plataformas eran propietarias y pocas. En esa época computación era sinónimo de IBM o Borroughs y los programadores no tenían la necesidad de soportar múltiples plataformas de computación.

Hoy en día la situación ha cambiado radicalmente. Las organizaciones típicamente utilizan más de una plataforma de computación para llevar a cabo sus operaciones diarias.

Desde los principios de esta década, portabilidad ha sido el tema sobre el tapete. La ruptura IBM/MicroSoft, en sus esfuerzos por mejorar OS/2; el éxito logrado por Windows 3.0 y como consecuencia de esto el anuncio hecho por MicroSoft de la creación de un nuevo sistema operativo Windows NT; la alianza IBM/Apple para crear un sistema operativo portable, Talligent, y un nuevo tipo de Pc, el PowerOpen; la reducción de precios en las estaciones de trabajo RISC como la Sun SparcStation y la introducción de su intefaz gráfico de usuario (Solaris) en los

Pc's; la venta de la división de hardware a Xerox por parte de Steve Jobs, creador de la máquina NeXT, y como consecuencia la incursión de esta compañía en los computadores basados en procesadores Intel; el éxito de los compiladores Borlan, los cuales están basados en lenguajes orientados al objeto inclusive la última versión de su poderoso sistema de bases de datos relacional Paradox; son sólo algunos de los acontecimientos que a principios de esta década han marcado el gran revuelo de la portabilidad, la programación orientada al objeto y los interfases gráficos de usuario.

Portabilidad es una colección de ideas que no son fáciles de definir. Pero en esencia significa, escribir programas que ejecuten las mismas operaciones en diferentes máquinas y sistemas operativos. Gracias al incremento de las redes locales, las compañías comienzan a estar más conectadas entre sus departamentos, la necesidad de que diferentes aplicaciones compartan datos en común o inclusive recursos comunes de hardware seguirá creciendo. Es en estos momentos que la portabilidad toma más importancia como parte integral de toda organización.

El mayor problema que enfrenta la portabilidad es crear aplicaciones que corran de forma idéntica en diferentes plataformas. Este problema no podría ser salvado si los sistemas operativos modernos no hubiesen implementado interfases de desarrollo y aplicación (API's) como parte integral de sus sistemas.

Cuando se desarrollo para múltiples plataformas, hay que tomar en cuenta que los usuarios quieren que su aplicación que migra de Windows a Macintosh debe parecer una aplicación de Macintosh con todos sus componentes y no una de Windows en un ambiente Mac.

Aunque es bien sabido, que las redes de área local (LAN) pueden, mediante el uso de protocolos, pasarse información entre sistemas operativos incompatibles. Pero en este proceso pueden suceder dos cosas:

Primero: que no sólo se necesite transmitir datos en forma alfanúmerica, sino que se requiera traspasar información gráfica, de multimedia, de autoedición, etc. y que el programa que va a recibir dicha información en la otra máquina pueda convertirla a su formato nativo. Lo que obliga a utilizar dos copias del mismo programa trabajando bajo ambientes diferentes.

Segundo: que se pierdan elementos importantes en la transmisión de datos.

Los casos que las redes locales pueden salvar este abismo es cuando se transfiere información que puede ser homogénea como datos números, etc. que

pueden transmitirse con facilidad entre dos equipos incompatibles y que el protocolo de la red puede ayudar a transmitir.

Son innumerables los esfuerzos que se han hecho para hacer que una aplicación corra en un ambiente que no fue en el que originalmente se programó. Como citaba en párrafos anteriores, los usuarios quieren que sus aplicaciones de Windows bajo Mac parezcan aplicaciones de Mac y no de Windows. Pero comprender correctamente un interfaz gráfico de usuario no es una tarea fácil, imaginese un programador que deba comprender más de un interfaz gráfico. Estos son puntos que ni siquiera grandes compañías se habían atrevido a tocar.

Puedo concluir diciendo que, la portabilidad entre sistemas operativos es la última barrera que los programación deben salvar. El día en que los fabricantes de softwares y sistemas operativos puedan lograr este objetivo, le habrán dado la facultad a los computadores de hablar en un mismo idioma que beneficiará no sólo a las grandes corporaciones sino, a los programadores que necesitan hacer portables sus aplicaciones a varios sistemas operativos.

4.2 Sistemas Operativos Portables

4.2.1 OS/2

Aunque el anuncio de hacer de OS/2 un sistema operativo portable fue hecho en este año, su historia como uno de los sistemas operativos más poderosos para computadores IBM-Compatibles es bien conocida.

OS/2 ha tenido una tumultuosa historia que ha sido intervenida por el éxito de Windows 3.0 que lo dejó a un lado en cuanto a la preferencia de los usuarios. OS/2 fue originalmente diseñado como el sucesor de DOS en un esfuerzo conjunto entre Microsoft e IBM. OS/2 fue concebido para chips 80286 que pueden accesar hasta 16Mb. de memoria real. El sucesor de este chip, 80386, trajo características que no se pensaron en el diseño original de OS/2. Una de esas era la protección de memoria interna del micro que permite que un sistema operativo pueda utilizar lo que se llama preemptive multitasking.

Desafortunadamente, las primeras versiones de OS/2 eran lentas y ofrecían poca compatibilidad con DOS. El procesador bajo el cual fue diseñado (80286) tenían implementado primitivas para ofrecer un "modo de compatibilidad" para correr aplicaciones pertenecientes a chips 8086 y 8088. Esto no permitía un cambio rápido

entre aplicaciones que corren en modo protegido y aplicaciones que corren en modo real.

OS/2 también carecía de soporte a printers y periféricos estandares y fue mercadeado exclusivamente por IBM para su línea de computadores PS/2. Muchos usuarios vieron en OS/2 algo hecho por IBM que pretendía llevarlos hacia una arquitectura de sistemas cerrados, como los costos MainFrames de esta compañía, opinión esta que contrasta con la mentalidad abierta y de multivendedor que existe en el mundo de los Pc's.

Como resultado OS/2 perdió el mercado con poco soporte a los desarrolladores o los usuarios y Microsoft junto a IBM solo fueron capases de vender no mas de un millón de copias de OS/2.

Para contrarrestar este problema, IBM y Microsoft comenzaron a trabajar en un nuevo, 32-bit OS/2. Pero, para el momento en que los planes finales de OS/2 2.0 eran finalizados las dos compañías rompieron sus acuerdos de cooperación. IBM tomo la responsabilidad de crear las futuras versiones de OS/2.

En contraste con las versiones anteriores, la versión 2.0 sería diseñada para chips de 32-bits explotando el poder de procesadores como el 80386 y posteriores. Proveyendo el acceso a 4GB de memoria real y la implementación del MVDM (Multiple Virtual DOS Machine) para soportar aplicaciones de DOS. El resultado es un sistema operativo que puede competir con cualquier Mainframe en opciones y potencialidades al alcance de todos los usuarios de computadoras Pc.

Fortalezas de OS/2

OS/2 ofrece un gran número de ventajas, utiliza modo protegido y preemptive multitarea. La protección de memoria significa que ninguna aplicación bajo OS/2 puede hacer que otra o el sistema operativo se caiga. Las ventanas utilizan multitarea colectiva, y estas pueden compartir sus recursos.

Multitarea Preemptive significa que el sistema operativo OS/2 localiza el tiempo para las aplicaciones, en vez de que las aplicaciones localicen tiempo entre sí. Esto significa que programas no sobrepasen la capacidad de multitarea del sistema operativo con largas colas de procesos, porque el sistema operativo siempre mantiene el control de estas.

el Adobe Type Manager dentro del sistema para el manejo de tipos escalables de letras.

Sus debilidades

Después de haber leído como trabaja este poderoso sistema operativo es muy difícil hallar debilidades en su estructura. Su mayor debilidad la conserva desde los tiempos de su primera versión: "no hay tantas aplicaciones de OS/2 como para competir con otra plataforma". Aunque existen poderosos ambientes de programación y lenguajes orientados al objeto como C++ y Smalltalk en OS/2 la comunidad de fabricantes de aplicaciones no han ofrecido la migración completa de sus aplicaciones a este sistema operativo. Hasta que esas aplicaciones se hagan nativas en OS/2, los usuarios de este sistema operativo estarán limitados a usar aplicaciones de DOS y Windows (debido a que OS/2 es capaz de correr aplicaciones de ambos ambientes).

El nuevo paso en la evolución de OS/2 es de darle un nuevo Kernel, el cual es la máquina esencial de todo sistema operativo. IBM está trabajan en nuevo kernel portable, parecido al de Windows NT, porque planea que las aplicaciones de OS/2 puedan corren en el sistema operativo PowerOpen y Taligent.

4.2.2 NeXTStep

NeXT nació después que Steve Jobs renunció de su posición con Vice-presidente de Apple Computer. Antes de irse, Jobs estaba trabajando en lo que seria la nueva generación de computadores Apple, y llevó su idea creando la máquina NeXT.

La idea de lo fácil que es utilizar una Mac, emergió junto con un poderoso sistema operativo que fuera fácil de programar. El sistema operativo NeXTStep fue el resultado. Ha tenido más de tres generación desde que fue introducido y hoy se ha consagrado como uno de los sistemas operativos más poderosos para cualquier computador.

Aunque la comunidad de usuarios de máquinas NeXT es relativamente pequeña, cerca de 60,000 usuarios, estos representan un grupo muy peculiar que van desde educadores, científicos, gobierno federal y sistemas financieros. NeXT es muy famosa por su facilidad de programación.

Fortalezas y Debilidades

En estos momentos, NeXTStep está disponible sólo con computadores NeXT, los cuales son sumamente caros. Pero NeXT recientemente demostró su sistema

operativo corriendo en una máquina 486 y ha anunciado que este tendrá capacidad de correr tanto aplicaciones de Windows como de DOS. Pero sus requerimientos de Hardware son un poco alto para los actuales usuarios de Pc: 8Mb. de Ram para correr el sistema en blanco y negro y 16Mb. para el sistema en color.

En el lado positivo, el sistema operativo NeXT ofrece una plataforma de computación muy poderosa basado en protocolos UNIX. El corazón del sistema NeXTStep es Mach 3, un Kernel similar al de UNIX pero más pequeño y más rápido. Mach 3 provee las facilidades de multitarea en hilos para el sistema además de que por su notable herencia de UNIX, este puede ser portable a casi cualquier máquina.

NeXTStep es actualmente el primero y el único sistema operativo orientado al objeto. Esto significa que muchos de los componentes encontrados en aplicaciones NeXTStep residen en el sistema operativo. Las aplicaciones no necesitan duplicar estos componentes sino, comunicarse con ellos usando un mecanismo de pasada-de mensajes. Esto se traduce en que mejoras al sistemas operativo, se reflejan en todas las aplicaciones.

Mientras que la programación en otras arquitectura, es el complejo proceso de codificar, depurar, compilar, etc., en NeXTStep solo existe el problema de conectar

objetos con un programa especial llamado Interface Builder. Para añadir texto a un programa, solo basta con unir al programa el objeto de texto del sistema sin siquiera escribir una sola línea de código. De acuerdo con los ejecutivos de la NeXT, reciclar objetos para nuevos usos hace que el tiempo de programación se reduzca un 66%.

Uno de los nuevos objetos que NeXTStep provee es un objeto de Base de Datos. Este permite a los programadores de NeXTStep poder juntas complejas bases de datos relacionases con facilidad. Permite unir dos bases de datos aún sin campos comunes.

Display PostScrip es otra de las ventajas de NeXTStep. Con este los documentos que se ven en la pantalla se ven exactamente como serán impresos (WYSIWYG), porque PostScrip es usado para describir la apariencia del documento en la pantalla como la apariencia de este en la impresora. También permite calibración de periféricos y que la reescritura de la imagen en la pantalla sea muy rápida.

Otro componente dentro de NeXTStep, es una poderosa librería de gráficos tridimensionales utilizando los comandos interactivos de RenderMan.

Aunque NeXTStep es el sistema operativo más fácil de programar que existe, el número de aplicaciones disponibles para este es muy reducida comparadas con los miles de paquetes existentes para plataformas como Windows, DOS o Macintosh. Esto se debe en gran medida, a la pequeña cantidad de máquinas NeXT instaladas, lo cual las hacen poco atractivas para los fabricantes de softwares.

Debido a que NeXTStep está modelado a imagen de UNIX, el protocolo de red predominante es TCP/IP. NeXTStep ofrece un gran número de aplicaciones para inicializar y manejar una red basada en TCP/IP. Para esto solo basta unos cuantos clicks con el mouse. NeXTStep también tiene un software cliente de NetWare para conectar con redes Ethernet lo cual le permite compartir datos con un sin número de plataformas.

4.2.3 X Windows

Al igual que UNIX, X Windows nació como un proyecto de desarrollo. El Instituto de Tecnología de Massachusetts tenía cientos de estaciones y servidores diferentes instalados en su red y tenían el problema de encontrar un interfaz común entre todos ellos. El instituto quería correr aplicaciones gráficas, y que se pudiera implementar multitarea en estos en diferentes máquinas en la red. Para llenar estas

necesidades fue diseñado X para ser la red y que fuera independiente del sistema operativo.

X no es en sí un interfaz gráfico, es un protocolo para dibujar y manipular objetos en la pantalla. El servidor X es el programa de gráficos que maneja el interfaz de usuario, aceptando las entradas de los usuarios y llevando el control de lo que se dibuja en la pantalla, etc. El cliente X es el programa de aplicación que maneja los datos. El cliente y el servidor se hablan entre si usando el protocolo X, y pueden hablar a través de la red o en la misma máquina.

Las implementaciones más conocidas de X Windows son: OSF Motif y OpenLook.

El poder de X Windows proviene de su habilidad de separar el proceso de aplicación del proceso de desplegar datos en la pantalla. Y estas dos partes no necesariamente tienen que residir en la misma máquina.

4.2.4 Windows NT

Se puede contar la historia de este sistema operativo como el punto en que los dos grandes de la computación IBM y Microsoft decidieron dividir sus senderos.

Para ese tiempo, las dos compañías estaban trabajando en conjunto para crear un sistema operativo que reemplazaría a DOS. Este nuevo sistema operativo estaba supuesto a ser OS/2 2.0. Entonces algo inesperado sucedió. De acuerdo con Microsoft, las ventas de Windows 3.0 sobrepasaron los tres millones de copias en los primeros siete meses después de su salida al mercado.

Microsoft, rápidamente se dio cuenta de lo lucrativo que podía ser el negocio de Windows y las dos compañías acordaron que IBM se encargaría de desarrollar OS/2 2.0; y Microsoft OS/2 3.0. Pero microsoft abandonó la idea de trabajar en un sistema operativo que tuviera por nombre OS/2. La compañía comenzó a referirse a este nuevo sistema operativo como Windows Portable, y luego como Windows New Technology.

Windows NT es una representación muy elaborada del ambiente de Windows 3.0. Su interfaz gráfico es muy familiar, pero el sistema operativo ya no es DOS. NT es un sistema operativo de 32-bits con soporte para multitarea preemptive, multihilos y soporte simétrico de multiprocesadores. Sus especificaciones incluyen seguridad nivel C-2 y un sistema de caracteres llamado Unicode en el cual se pueden incluir alfabetos de letras no romanas.

El corazón de Windows NT es su microkernel, o el NT Executive, como es llamado por Microsoft. Este consiste en 50 KB de código compilado, que contiene todas las funciones que son dependientes del hardware. El kernel es rescrito para diferentes procesadores y sirve como la parte nativa del sistema, aislando las dependencias del hardware.

Sobre el kernel, esta las extensiones del modo de privilegios, que procesa los drivers de dispositivos, el sistema de archivos y las redes que son accesadas directamente por hardware.

El siguiente es la capa del subsistema, que es el componente que hace posible que NT sea capaz de accesar Interfases de Aplicación y Desarrollo múltiples. Este crea una barrera, de forma tal que ningún programa o librería de programas sea capaz de hacer que se caiga el sistema.

Microsoft Planea proveer tres subsistemas dentro de NT: Windows, OS/2 y Posix. El subsistema de Windows soportará programas de 32-bits y tendrá compatibilidad con programas de 16 bits y programas de DOS vía emulación. El subsistema de OS/2 correrá aplicaciones de OS/2 1.3 y 2.0. El subsistema Posix correra programas estandares de Unix. Debido a que los sistemas están

implementados como tareas independientes y NT es un sistema operativo multitarea, los tres subsistemas pueden correr a un mismo tiempo.

La última capa del sistema operativo es la de aplicación. Las aplicaciones son vistas como clientes de los subsistemas protegidos. Los clientes requieren servicios del sistema operativo y estos son pasados al subsistema correspondiente del sistema operativo para su manejo.

4.2.5 Apple/IBM Taligent

La pieza más ambiciosa de la nueva amistad entre Apple e IBM es su aventura para crear un nuevo sistema operativo.

En septiembre de 1990, IBM anunció la formación de Patriot Partners con Methaphor Computer Systems. Pratriot estaba construyendo un software de desarrollo para portabilidad. Para aquel entonces, IBM tenía dos aliados triunfadores bajo su manga. Patriot tenía una licencia de NextStep pero no había hecho nada con esta.

Patriot Partners se convirtió en parte de lo que es la nueva compañía llamada Taligent. Después del anuncio de la alianza IBM/Apple.

Taligent ha traído dos contribuciones básicas: Costellation y Pink.

Constellation fue diseñado para ser la plataforma softwrare. Así como las piezas de una máquina pueden ser utilizadas en otra, los componentes individuales de software son creados de forma individual que pueden potencialmente ser combinados para crear una nueva aplicación.

Los desarrolladores sólo tienen que crear pequeños y bien orientados programas que serán los componentes. Constellation proveerá los mecanismos para ensamblar los componentes, protegiendo las uniones contra una posible separación, y dándole la libertad a los usuarios y desarrolladores de recombinar libremente algunos aspectos de estos para construir nuevas aplicaciones.

La plataforma completa de Constellation está supuesta a ser portable. Constellation podrá correr en OS/2, AIX, OSF/Unix, PenPoint, System Mac y 32-bit Windows.

Pink será el sistema operativo orientado al objeto. Como Taligent representa una combinación de ambas tecnología, se puede esperar que este sea un sistema operativo que use un modelo orientado al objeto en todas sus dimensiones.

4.3 Beneficios de la Portabilidad

Los beneficios que pueden ser resultados de la portabilidad son inmesurables. Pero trataré de recoger los más importantes:

1.- Consistencia de las aplicaciones que corren en diferentes plataformas. Si tenemos una amalgama de plataformas (UNIX, MAC, PC, ect.) y necesitamos que en la red se corran programas comunes a todas estas; la mejor solución es crear una aplicación bajo uno de estos interfases gráficos con la menor cantidad de instrucciones dependientes del hardware y recompilarla en la otra o las otras plataformas.

2.- Estandarización. Los interfases gráficos de usuario que son característicos en los sistemas operativos portables, requieren que el programador se lleve de ciertas normas. Esto se traduce en menos tiempo de modificación en el momento que sea necesario y por lo tanto una reducción sustancial de costos.

3.- Facilidad de programación. Como en el caso de los computadores NeXT, en el que solo hay que hacer conexiones entre objetos predefinidos en el sistema operativo para crear nuevas aplicaciones.

Esta tendencia de crear ambientes que no solo sean fáciles de utilizar para los usuarios finales, sino para los programadores se ha hecho más notable desde la revolución de las herramientas visuales de programación (de 4ta. y 5ta.) como los case.

4.- Integración de los recursos del sistema operativo. En este punto, los programadores no tienen que preocuparse por programar elementos que son comunes a todos los programas como son: drivers, ventanas, tipos, etc. en los cuales se puede desperdiciar recursos que serían muy útil canalizarlos a otros aspectos del software.

5.- Los recursos de hardware pueden ser aprovechados al máximo con el mínimo de esfuerzo por parte del usuario y del programador.

CAPITULO V

LOS INTERFASES GRAFICOS DE USUARIO Y LA PROGRAMACIÓN ORIENTADA AL OBJETO

5.1 El Impacto de los Interfases Gráficos de Usuario

Una de las razones por las cuales la portabilidad se ha vuelto más compleja en los últimos años es la gran demanda por aplicaciones que corran bajo interfases gráficos de usuario (GUI). Estas son un poco más complicadas de convertirlas en portables que aquellas que corren bajo un interfaz de carácter. Dominar los aspectos de un GUI es una tarea difícil para una sola persona e inclusive para una compañía en particular.

Como consecuencia, un gran número de compañías están dedicadas a la tarea de crear herramientas de programación para el desarrollo de aplicaciones portables. Estas herramientas son la contraparte de los compiladores y emsambladores que eran la única herramienta para el desarrollo de aplicaciones portables. En vez de utilizar el API (Aplication Progran Interface) de una determinada plataforma lo que se hace es que se escriben las aplicaciones utilizando el API de la herramienta de desarrollo.[1]

[1] Revista Byte, A Moving Tarjet, Febrero 1992.

Sin embargo, crear aplicaciones utilizando estas plataformas significa que sólo es posible utilizar los elementos que son comunes en todas las plataformas en que se programa. Por ejemplo: en Windows existe una función que permite que un programa coloque un objeto dentro de otro llamada OLE (Object Linking and Embeding) que es el equivalente del Publish/Subscribe en Macintosh.

Pero es indudable que el impacto causado por estos programas es tal, que no se concibe un sistema operativo que no lo incluya, por lo menos como opción. Podemos ver como el gran revuelo que ha causado Windows 3.0 en los Pc y antes de este la larga trayectoria de Macintosh con su metáfora de fácil manejo convirtiéndose en el símbolo de las gráficas en la computación; así como las Sun SparcStations que dominan el mercado de la ingieneria CAD y la investigación científica. Los interfases gráficos son elementos imprescindibles en cualquier estación de trabajo moderna.

5.2 Programación Bajo Interfases Gráficos de Usuarios

Aquellos que hemos tenido la oportunidad de trabajar en máquinas como los 386 y 486, hemos notado que en DOS solo podemos realizar un trabajo a la vez. Y no en más de una ocasión nos hemos preguntado, porque se desperdicia tanto poder en un solo proceso si el procesador de la máquina puede dar más. Pero cuando

vemos al mismo equipo trabajar en UNIX, OS/2 e inclusive en Windows es cuando realmente nos damos cuenta de las potencialidades del equipo. Un mismo computador puede ser mas eficiente en UNIX que en Windows por el tipo de multitatera que el primero implementa. Pero las deficiencias de Windows provienen de la base que es el sistema operativo. A continuación haré reffencia a dos tipos de importantes de ambientes bajo los cuales se programa: Event-Driven y Multithreding.

5.2.1 Programación Event-Driven

Existen ambientes gráficos (como Microsoft Windows 3.0 y 3.1) en los cuales todo es un evento. Un evento puede ser una entrada por el teclado, un click del mouse, etc. y para que un nuevo evento tenga lugar, el que se está ejecutando debe terminar. Por ejemplo: si estamos trabajando en un procesador de palabras y le indicamos que imprima este tomará todo el tiempo del procesador hasta que el evento de impresión termine, al menos que mediante la implementación de multitarea cooperativa en el sistema este pueda salir a otra aplicación, pero en procesador de palabras no se podrá ejecutar otra acción hasta que el "evento impresión" llegue a su final o sea cancelado. Supongamos que necesitamos agregar unas cuantas páginas a nuestro texto mientras se imprime, o que necesitamos abrir otro documento para trabajar en el. En la mayoría de los procesadores de palabras de Windows y Mac se da un caso un particular: hasta que la caja de dialogo indicándonos la impresión no

se retire tenemos que limitarnos a esperar a que esta se retire. En ambientes manejados por eventos, ciertos procesos como el anterior se hacen tediosos y muy lentos, pues una aplicación de esperar terminar un proceso para ejecutar el siguiente.

5.2.2 Programación Multithreding

Contrario al caso anterior, un ambiente multi-hilo divide la ejecución del programa en pequeños objetos (threads -hilos-) que pueden ejecutarse de forma casi simultanea y con el mismo tiempo requerido para cada uno. Por ejemplo: Si una aplicación es dividida en 20 threads y cada uno de estos requiere un segundo de ejecución, el sistema operativo aún si hay otra aplicación corriendo, le da el tiempo exacto de un segundo a cada thread. Retomando el caso anterior: Si el procesador de palabras sube el thread de impresión el sistema operativo será capaz de localizar el tiempo que necesita para los threads de impresión y puede devolver el control del programa al usuario porque ejecuta más rápidamente cada thread, este es el caso de ambientes como OS/2 2.0. En este tipo de ambientes el corte donde va cada thread es escogido por el sistema operativo dependiendo de la cantidad de objetos que estos contengan.

5.3 Programación Orientada al Objeto bajo Interfases Gráficos de Usuario

El rol de los objetos bajo interfases gráficos de usuarios, fue iniciado por ambientes de programación como Smalltalk. En estos todos los componentes del sistema operativo están contenidos dentro de los objetos predefinidos dentro del lenguaje. Mediante esta implementación las complejidades del hardware son escondidas, y el programador se centra más en pensar que debe hacer la aplicación y no como hacer la aplicación. Pero el ejemplo más claro del poder que representa el modelo orientado al objeto lo es la máquina NeXT y su sistema operativo NeXTStep. A esta máquina (conocida como el cubo por su semejanza a una figura de esta forma) muchas revistas la han bautizado como "el sueño de los programadores". En ella todo es un objeto y como describí anteriormente, solo basta conectar líneas entre los objetos que se muestran en el Interface Builder para crear una nueva aplicación o para añadirle nuevas características a una aplicación creando un alto grado de customización.

Sin embargo, toda esta facilidad no hubiese sido posible si no se hubiera implementado el concepto de objetos y su modularidad. Todos los componentes en el listados en el Interface Builder de la Next son objetos y añadir nuevas características al sistema operativo o a cualquier programa no significa tener que reescribirlo, compliarlo, depurarlo, probarlo y luego llevarlo al terreno de la práctica.

Todo lo que se necesita para crear nuevas aplicaciones, en menos tiempo y mucho más poderosas es saber que es lo que se quiere y como se quiere. Este es el más claro ejemplo de la combinación de los interfases gráficos de usuario y la programación orientada al objeto.

5.4 Program Aplication Interface

En los lenguajes de progamación, es bien conocido el uso de librerias que tienen ciertas caracteristacas especiales. El lenguaje de progración C, es el ejemplo más claro del uso de librerias; como por ejemplo: si deseamos accesar los datos de un sistema gestor de base de datos como FoxPro, no es necesario programar el acceso de los archivos, lo que significa cada byte en la cabecera, etc., solo basta con tomar una libreria comercial que tiene predefinidos todo lo que es el manejo de archivos y queries en Fox, incluirla en nuestro código, compilar y tendremos las mismas facilidades. Así tambien los interfases gráficos de usuarios se valen del uso de los Interfases de Programación y Desarrollo (Aplication Program Interface -API-) con los cuales es posible accesar todas las funciones del interfase gráfico. En Windows el API está implementado en forma de Dinamic Link Libraries (DLL). Mediante llamadas a las rutinas del API se pueden accesar: manejo de gráficas (si está implementado en ellas geometría Bézier, si hay acceso a gráficas de alta resolución a 24-bit, etc.), manejo de componentes del sistema operativo (como OLE, DDE -en

Windows-; Publish/Suscribe, cut and paste -en Mac-), y nuevos componentes pueden ser añadidos porque cada uno es un objeto individual del sistema operativo.

5.5 El futuro de los interfases gráficos de usuario y la Programación Orientada al Objeto

En cuanto a la programación orientada al objeto se refiere, muchos autores coinciden en que todavía no se ha tenido la suficiente experiencia en este campo para decir que será el punto de la tecnología a principios del próximo milenio. También afirman que esta no es aplicable a todos los ámbitos de la computación, por ejemplo en las redes locales. Pero en lo que todos coinciden es que este concepto es tan revolucionario que las aplicaciones que comienzan a surgir totalmente orientadas al objeto prueban que por lo menos en el campo de la ingienia y el software sus beneficios son notables. Hay una frase muy famosa entre los textos que leí acerca de este concepto: "para aprender a programar utilizando el modelo orientado al objeto, solo basta olvidar todo lo que conocíamos de programación y comenzar desde cero", y esta afirmación es hecha debido a que la abstracción hecha por los lenguajes orientados al objeto se centra más en las características esenciales que queremos ver en nuestros sistemas obviando aquellas que no son de nuestro interés.

En el aréa de la informática, los conceptos cambian tan rápidamente que muchas veces no nos damos cuenta siquiera que existieron una vez. Hace tres años hablar de cuatro MBytes de ram en un Pc era algo exagerado, hoy existen computadores cuya configuración mínima es de 16MB y este número se ira incrementado. El fracaso de OS/2 fue debido a eso, los usuarios no estaban preparados para ese sistema operativo, ni en hardware ni en conceptos. Así como el los 70's la programación estructurada cambió todos los esquemas, en los 90's y más allá, el modelo orientado al objeto puede ser (porque en informática no hay reglas absolutas) la luz que guíe la programación.

La programación orientada al objeto será el elemento principal de programación bajo interfases gráficos de usuario, debido a que esta ajusta perfectamente su estructruas y conceptos (clases, objetos, herecias, etc.) a la forma de como trabajan estos. Pasar mensajes de un objeto a otro puede ser equivalente de pasar una información entre dos aplicaciones. Además que en sistemas de multihilo (multithreding) los objetos pueden convertirse en cada thread de ejecución, porque tienen límites propios bien definidos.

En cuanto a los interfases gráficos, estos seguirán atrayendo a más usuarios a medida que la cantidad de hardware instalada pueda soportar los requerimientos de estos. Pero el futuro de estos estará muy ligado a:

1.- Nuevos sistemas gráficos de 32-bits de color calibrado. De forma que lo que se vea en la pantalla se imprima tal como es, no importa el printer que se utilice.

2.- La inclusión del manejo efectivo de tipos de letras (como TypeManger, TrueType, ect.).

3.- Soporte del sistema para objetos de 3D.

4.- Video y audio digital.

5.- Los avances del hardware en cuanto microprocesadores, adaptadores gráficos y soporte a multimedia se refiere.

6.- El intercambio y actulización de objetos dentro de aplicaciones ya sean locales o via LAN.

Estos seis puntos unidos a la programación orientada al objeto serán los puntos de discusión para los próximos años. Talvez nos suceda lo que a aquellos que hace más de 40 años en un sótano programaban una máquina a base de cables, aquellos que nunca soñaron siquiera con la décima parte de los avances con que contamos hoy en día en el campo de la informática.

CONCLUSION

CONCLUSIÓN

La ciencia de la informática es tan joven, que hablar del pasado es mencionar sucesos de hace cinco, diez, treinta o cincuenta años. Más en su corta existencia como ciencia formal, ha evolucionado de una forma acelerada.

Viendo hacia el pasado, podemos observar como los antiguos computadores eran programados vía hardware; luego con tarjetas perforadas y así continua una cadena de acontecimientos que han transformado totalmente la palabra "programación".

El concepto de programa almacenado de Von Neuman, la programación estructurada, el diseño orientado al objeto, los ambientes portables y los interfases gráficos de usuarios son partes de las tecnologías que han guíado en el pasado y guiaran en el futuro la computación.

En nuestros días: Down Sizing, Modelamiento Tridimensional, Telecomunicaciones, Realidad Virtual, Multimedia, Autoedición, etc. se han hecho los ecos de la computación en los grandes y pequeños ambientes. Pero enfrentar el reto de estas tecnología requiere de nuevos elementos que aceleren el proceso de creación de las aplicaciones que representarán estas tecnologías.

La programación orientada al objeto jugará un papel determinante el mundo del software porque con esta, podemos rehusar código y transportar programas entre plataformas.

Con la noticia de que los grandes fabricantes de sistemas operativos e interfases gráficos, han decidido implementar en sus productos el modelo orientado al objeto y la portabilidad entre sistemas operativos, estamos seguros de que en los próximos años la programación será tan diferente que recordaremos la codificación manual como algo que hacíamos hace siglos.

BIBLIOGRAFIA

BIBLIOGRAFÍA

Borland C++ and Aplications Frame Work
User Manual and Getting Started
Borland Internacional, 1992

Charafas, N. Dimitris.
Forth and Fith Generation Lenguajes.

Ingeniería del Software
Pressman, Roger
McGraw Hill México, 1993

Inside Windows NT
The Microsoft Windows NT Research Group
Microsoft Press USA, 1992

Object Oriented Design With Applications
Booch, Grady
The Benjamin/Comming Publishing Company, Inc. USA, 1991

Pc-Magazine Anatomy of a utility:
Writing Apps. With C++
Ray Duncan Frebrero 1992

Pc-Magazine C++ Makes OOP Benefits Practical
William F. Zachmann
Diciembre 17, 1991

Pc-Magazine OOP Will Make our Lives Easier too.
Jim Seymour
Enero 14, 1992

Pc-Magazine C++ Gets Mainstream
William F. Zachmann
Abril 14, 1992

Pc-Magazine Mastering The Complexity of the Windows API
Ray Duncan
Junio 11, 1991

Revista Byte
A Moving Tarjet
Tributaries and Deltas
Let the System do the Porting
Marzo 1992

Revista Byte
NeXT: The Programmer's Dream Mchine
The World of Objects Marzo, 1990

UNIX and XENIX demistified
Clukey, Lee Paul
Tab Book, Inc. USA, 1985