77
259

UNIVERSIDAD NACIONAL AUTONOMA
DE MEXICO

FACULTAD DE CIENCIAS

EL SOFTWARE ORIENTADO A OBJETOS.
PANORAMA GENERAL.

T E S I S

QUE PARA OBTENER EL TITULO DE

A C T U A R I O

P R E S E N T A :

JOSE GUILLERMO PRECIADO LOPEZ

DIVISION DE ESTUDIOS PROFESIONALES
FACULTAD DE CIENCIAS
SECCION ESCOLAR

MEXICO, D. F. 1996

TESIS CON
FALLA DE ORIGEN

TESIS CON
FALLA DE ORIGEN

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

TESIS

COMPLETA

VNIVERSIDAD NACIONAL
AVFNOMA DE
MEXICO

M. en C. Virginia Abrín Batule
Jefe de la División de Estudios Profesionales de la
Facultad de Ciencias
P r e s e n t e

Comunicamos a usted que hemos revisado el trabajo de Tesis:

"EL SOFTWARE ORIENTADO A OBJETOS. PANORAMA GENERAL

realizado por PRECIADO LOPEZ JOSE GUILLERMO

con número de cuenta 8153319-0 , pasante de la carrera de ACTUARIA

Dicho trabajo cuenta con nuestro voto aprobatorio.

Atentamente

Director de Tesis Propietario	M. EN C. GUADALUPE IBARGUENGOITIA GONZALEZ	
Propietario	MAT. ANA LUISA SOLIS GONZALEZ COSIO	
Propietario	M. EN C. GUSTAVO MARQUEZ FLORES	
Suplente	DRA. HANNA OKTABA	
Suplente	ACT. JAIME ANDRES BLANCO CACIQUE	

Consejo Departamental de Matemáticas

M. EN C. ALEJANDRO BRAVO MOJICA

A CARMEN Y A MÓNICA

Quienes son la luz y el combustible de mi vida.

A MIS PADRES Y A LOS ABUELOS

Por permitirme conocer este mundo, colmado de objetos.

A MIS HERMANOS

Por todo lo que hemos compartido.

A Paty, el tio Rody, Laura y Felipe, Rosy y Jano, los Aboñas y especialmente a Andrés, todos ellos instancia de una clase muy abstracta: CRONOPIOS

CONTENIDO

INTRODUCCIÓN

El impacto de la tecnología orientada a objetos en el desarrollo de software durante lo que va de la década de los noventas, podría ser más dramático que los progresos en el análisis y el diseño estructurado en los ochentas. La mayor parte del software desarrollado a finales de los noventas será, muy posiblemente, orientado a objetos. Una cantidad importante de compañías de software están desarrollando y comercializando una amplia variedad de herramientas para producir aplicaciones de toda índole orientadas a objetos: lenguajes, sistemas operativos, bases de datos, interfaces gráficas de usuario, ambientes personales de productividad, etc. Por otro lado tanto las organizaciones como los usuarios finales están viendo los beneficios inmediatos que se derivan de la introducción de las técnicas orientadas a objetos en el desarrollo de software de excelencia.

Aunque las nuevas aplicaciones y los nuevos sistemas operativos basados en objetos ofrecen eventualmente interoperabilidad entre plataformas, la transición hacia la tecnología orientada a objetos puede no ser tan suave ni tan rápida como lo sugieren sus beneficios inherentes. Los fabricantes de hardware y las compañías desarrolladoras de software, podrían resisitirse a abandonar inversiones considerables en sus propias estrategias de interoperabilidad y herramientas de desarrollo basadas en caracteres, y en metodologías estructuradas. Mientras que la "reutilización" de objetos ofrece a los desarrolladores ahorros a largo plazo, el entrenamiento inicial y los gastos en herramientas de desarrollo pueden ser desalentadores. Sin embargo, las tecnologías orientadas a objetos están afectando al espectro del software en su totalidad. Incluyendo áreas como el análisis y el diseño, las interfaces gráficas de usuario (las famosas GUI'S por sus siglas en ingles Graphical User Interface), los sistemas de administración de bases de datos y los ambientes de productividad personal.

En este trabajo se plantean dos objetivos principales:

Exponer los conceptos que subyacen en las técnicas de orientación a objetos; los métodos y los temas técnicos que rodean al software orientado a objetos y la tecnología relacionada.

Ofrecer un panorama general del software orientado a objetos desde la perspectiva de los desarrolladores de software comercial y de los equipos de desarrollo.

El primer capítulo de este trabajo inicia con una breve revisión de la historia del desarrollo de sistemas computacionales, partiendo de la década de los cincuentas hasta la fecha. El hilo conductor de ésta sección es la evolución de las distintas técnicas de programación que se han usado desde entonces. En este capítulo presentamos los conceptos fundamentales de las técnicas de orientación a objetos, incluyendo terminología relacionada con el enfoque de orientación a objetos. También se discuten las ventajas y desventajas de este enfoque.

En el segundo capítulo exponemos los principios, incluyendo una breve descripción del proceso, del diseño y el análisis dentro del enfoque de la orientación a objetos.

En el capítulo tres se expone el panorama general del software orientado a objetos, los temas que aquí se revisan incluyen: lenguajes orientados a objetos, objetos y bases de datos, los objetos y las interfaces de usuario, y los ambientes personales de productividad.

En el capítulo cuatro pretendemos mostrar que la programación basada en el enfoque de orientación a objetos, será la que domine el desarrollo de software a finales de la década de los noventas, a través de un resumen de las más prominentes técnicas de orientación a objetos y examinando su impacto comercial en el futuro.

Finalmente planteamos las conclusiones que se desprenden de la revisión de las metodologías de desarrollo, los productos y las tendencias actuales del software orientado a objetos.

La mayor parte de este trabajo esta basado en la revisión de diversas fuentes bibliográficas que incluyen libros especializados en el tema, artículos publicados en diversas revistas de computación, documentación elaborada por los fabricantes de software, y hasta artículos periodísticos. Por ello hemos decidido colocar las referencias, en forma de notas al pie de página, al final de cada capítulo y la bibliografía completa al final del trabajo.

CAPÍTULO 1

LOS CONCEPTOS DE LA ORIENTACIÓN A OBJETOS

HISTORIA DEL DESARROLLO DE SISTEMAS

Antes de adentrarnos en el estudio de los conceptos básicos del software orientado a objetos haremos un breve recorrido por la historia del desarrollo de sistemas de computación. Es interesante observar cómo, a través de ciertos períodos de tiempo, los desarrolladores cambian del enfoque inicial centrado en los procesos a un enfoque centrado en los datos y recientemente en los objetos. Cada uno de estos enfoques tienen técnicas de programación que representan diferentes maneras de ver el desarrollo de sistemas de computación; todas son *correctas*, pero cada una tienen sus ventajas y desventajas. No pretendemos estudiar en detalle la historia de los lenguajes de programación, ni las diversas metodologías creadas para el desarrollo de sistemas; simplemente queremos esbozar el camino que han seguido los desarrolladores para llegar a la definición de metodologías que sirvan para resolver problemas crecientes en complejidad. Iniciamos nuestro breve recorrido a partir de mediados del presente siglo, cuando surgen las primeras computadoras.

En la década de los cincuentas el desarrollo de sistemas de computación era virtualmente inexistente. Lo poco que se hacia se llevó a cabo usando frecuentemente lenguaje de máquina o ensamblador. En esta etapa las primeras versiones de los lenguajes de programación de alto nivel, tales como FORTRAN (1954) y COBOL (1959),[1] empiezan a aparecer.

Durante la década de los sesentas se inicia la era de los *grandes sacerdotes* de la programación; ésta es considerada un arte. Las grandes corporaciones con posibilidades

de adquirir los costosos equipos de computo, también tenían *"gurús"* quienes producían programas de forma misteriosa. Eran comunes comentarios como: *"Lo encerramos en un closet y más o menos tres meses después sale con un programa"*. Nadie sabía el proceso por medio del cual se desarrollaban los programas. El problema de obtener las estimaciones de tiempo y costo de desarrollo de software era virtualmente irresoluble, dado que no existía una metodología claramente definida. A principios de los sesentas surge el lenguaje de programación ALGOL 60 que influyó enormemente en la creación de la programación estructurada.[2]

Es hasta finales de esta década, cuando se inician verdaderos cambios, que por su impacto fueron realmente significativos. Mejoras importantes en el desempeño del hardware y la adopción de lenguajes de programación de alto nivel permitieron la construcción de sistemas más grandes y complejos; sin embargo, la ausencia de metodologías de análisis y diseño provocó una crisis en la producción de software. Esta crisis impulsó la evolución de las metodologías modernas. En 1968 se publica el famoso ensayo de E. W. Dijkstra: *"GO TO Statement considered Harmful"*, el cual fue la guía teórica para el desarrollo de la programación estructurada.[3] En Noruega en 1967 un grupo de investigadores extienden ALGOL 60 con los conceptos de objetos, clases y jerarquías de herencias entre clases, para programar las simulaciones discretas de los problemas del mundo real, resultando en un nuevo lenguaje: SIMULA 67.[4] Este permitía al programador definir *objetos* en el sentido que hoy se le da a la programación orientada a objetos. SIMULA resultaría ser el ancestro común de la mayoría de los lenguajes basados en objetos y orientados a objetos de la actualidad.[5] En otras áreas del software surgen también innovaciones importantes, Ted Codd publica en 1970 su descripción de una nueva clase de base de datos llamada *relacional*.

Durante la década de los setentas se desarrollaron las técnicas de la programación estructurada. Con el definición del concepto de ciclo de vida del desarrollo de sistemas, por primera vez se describieron una serie de pasos requeridos para desarrollar un sistema

de computación. Las estimaciones de tiempo y costo fueron entonces posibles. Los desarrolladores pudieron comunicarse unos con otros sobre los pasos del proceso de desarrollo. A pesar de los diferentes significados atribuidos a los términos, este fue un gran avance. Las metodologías estructuradas permitieron un análisis más efectivo y un diseño más estable y manejable en términos de mantenimiento. El primero en incursionar en la llamada "revolución estructurada" fue Tom DeMarco a mediados de los setentas por medio de su clásico: *Structured Analysis and System Specification.*[6] DeMarco popularizó los conceptos de diagramas de flujo, diccionarios de datos y otras técnicas para traducir los resultados del análisis de sistemas en diseño de software útil. Con estas metodologías se podía, por fin, describir el sistema en términos que pudieran ser entendidos por los usuarios. Si bien los usuarios no podían entender todo lo que hay que hacer con el sistema podían al menos, con una explicación adecuada, seguir los procedimientos descritos por el analista y estar de acuerdo (o en desacuerdo) con él. El trabajo de DeMarco fue seguido mejorado y aumentado por muchos otros entre los que destacan: Yourdon y Constantine,[7] Mellor y Ward.[8]

A grandes rasgos, el proceso de análisis y diseño estructurado consiste en dividir al problema en sus partes más simples. Estas partes se conectan por medio del flujo y el control de los datos. Dividir los problemas en partes más pequeñas facilita la tarea de escribir el software que dirija esas partes y la comunicación entre ellas.

Una característica de las técnicas estructuradas, la cual resultaría significativa, es que son orientadas a procesos, es decir que están enfocadas a dar mayor peso a los procesos dentro del sistema por sobre los datos. Esto estaba completamente de acuerdo con el desarrollo de las décadas previas, los desarrolladores centraban su atención en los programas a ser desarrollados y los datos a ser procesados eran secundarios. Así en este enfoque se considera a un sistema en términos de *entrada (input), proceso (process) y salida (output)* es decir como una serie de procesos cooperativos, donde la salida de uno

se convierte en la entrada de otro. De esta forma la estructura del sistema se basaba en el proceso involucrado.

Una segunda característica, la cual no fue tanto el resultado de la metodología subyacente como de las limitadas capacidades de las herramientas (i.e. los lenguajes de programación) disponibles en ese tiempo, es que tienden a describir detalladamente el sistema en papel antes de empezar a programar. Esto provocó el desarrollo de herramientas para metodologías estructuradas: diagramas de flujo, diccionarios de datos, diagramas de estados de transición, gráficas de estructuras, etc.

A fines de la década de los setentas un grupo de investigadores del Centro de Investigaciones de Palo Alto de Xerox (PARC por sus siglas en Ingles) estuvieron experimentando con diferentes interfaces del usuario. Una de estas interfaces tenía un ambiente de ventanas (windowing), con un *ratón o mouse*, menús desplegados (drop down menús), iconos y gráficas. Para trabajar en este ambiente los investigadores desarrollaron un nuevo lenguaje llamado Smalltalk. Este lenguaje se basaba en los conceptos de objeto, clase y herencia como los definidos en Simula 67 y en los conceptos del List Processing Lenguage LISP.[9] En LISP todo, incluyendo el programa mismo es una *lista*. En Smalltalk todo es un *objeto*. Con Smalltalk empieza a popularizarse el término programación orientación a objetos (POO).

En los ochentas las bases de datos empezaron tener predominio. IBM apoyo fuertemente las bases de datos relacionales y, como resultado, un número significativo de organizaciones empezaron a cambiar hacia su uso. Esta tendencia hacia el uso de bases de datos relacionales tuvo un impacto significativo porque éstas permiten que los datos y los procesos estén separados, así los datos toman esencialmente una *vida* independiente. Cambios estructurales pueden hacerse en los datos sin tener impacto en los programas; repentinamente los programadores empezaron a ver a los datos como un recurso que necesitaba ser especificado y manejado en lugar de ser un anexo de los programas. Es

entonces que surgen las técnicas conocidas como entidad - relación (entity - relationship) para el modelado de grandes sistemas de bases de datos.

La introducción de micro computadoras (pc's) puso el poder de la computación a disposición de millones de personas. Inicialmente BASIC fue el lenguaje más popular entre los programadores para PC's, quienes gradualmente cambiaron al lenguaje C. Con la definición de Stroustrup del lenguaje C++ , una versión *mejorada* de C (incluyendo casi todas las características propias del enfoque de orientación a objetos) muchos programadores cambiaron hacia su uso de manera natural. La primera versión comercial de Smalltalk aparece en el mercado en 1984 [10] y un número de lenguajes propiamente orientados a objetos fueron desarrollados.

De manera similar a la evolución de los enfoques y técnicas estructuradas de los setentas, el cambio hacia la programación orientada a objetos provocó el desarrollo de técnicas orientadas a objetos en las etapas tempranas del desarrollo de sistemas - primero el diseño y luego el análisis. Los primeros libros describiendo técnicas de diseño orientadas a objetos empezaron a publicarse en 1988; libros sobre análisis orientado a objetos empezaron a publicarse en 1990.[11]

En la década actual (noventas) parece ser que la orientación a objetos se está convirtiendo en la técnica de programación más poderosa. El lenguaje C++ es ya uno de los más populares y versiones comerciales de lenguajes como Smalltalk y Eiffel se están convirtiendo en lenguajes de *moda*.[12] Existen una gran bibliografía sobre técnicas que involucran el enfoque de orientación a objetos para el desarrollo de sistemas; y las ventajas de este enfoque están empezando a realizarse en sistemas operativos y en interfaces gráficas del usuario (GUI, por Graphical User Interface) como es el caso de Windows de Microsoft y la gran cantidad de aplicaciones para este ambiente, que han sido desarrolladas aplicando el enfoque de orientación a objetos. Tanto en el ambiente académico como en el de las casas desarrolladoras de software comercial existe consenso

en cuanto al retraso que tienen las posibilidades del software, en relación al estado actual y el potencial del hardware. El enfoque de orientación a objetos esta demostrando ser una técnica superior para el desarrollo de sistemas porque permite la programación en niveles más altos de abstracción: del objeto a la clase, hasta la biblioteca de clases y a los complejos marcos estructurales de aplicación. Más adelante nos adentraremos en la discusión sobre las ventajas y desventajas del modelo de orientación a objetos..

CONCEPTOS BÁSICOS DE ORIENTACIÓN A OBJETOS

Cuando nos encontramos con la frase: *Programación Orientada a Objetos* (POO) sabemos que ésta puede tomar un número de significados diferentes dependiendo de la *pureza* de las operaciones con objetos y del sistema de software que se está usando. Debido a que Smalltalk es el lenguaje que abarca por completo los conceptos de un sistema de software totalmente consistente con el enfoque de orientación a objetos, es comúnmente usado como el parámetro para medir la *pureza* (en el sentido de orientación a objetos) de las aplicaciones.

El concepto clave del enfoque de orientación a objetos reside en considerar al sistema como una colección de objetos cooperativos, los cuales se comunican unos con otros enviándose mensajes. Los objetos, en el contexto del desarrollo de un sistema de computación, son abstracciones o representaciones de objetos del mundo real: un cliente, una cuenta, etc.; y contienen la información relevante a estos objetos, como la dirección del cliente o el balance de la cuenta. A continuación definimos los conceptos básicos de los términos usados en la POO.

¿Qué es un Objeto?

En contraste con los sistemas de software tradicionales los sistemas de software orientados a objetos combinan los datos y los procedimientos en una sola entidad: el objeto. El objeto es una entidad que agrupa a estructuras de datos con las operaciones que las manipulan (Figura 1.1) o en los términos de orientación a objetos, un objeto es un paquete de *atributos y métodos*. Los atributos son la información (estructuras de datos) contenida por el objeto, conforman el estado interno del objeto; también se les conoce como *variables de instancia*. Durante la vida de un objeto el valor de sus variables puede cambiar dinámicamente a través de las operaciones definidas para él. Los métodos son las

operaciones asociadas con el objeto, proveen la capacidad de manipular el objeto y describen su comportamiento. El concepto de objeto es a la vez simple y poderoso: *cada objeto conoce información y sabe como operar con ella.*

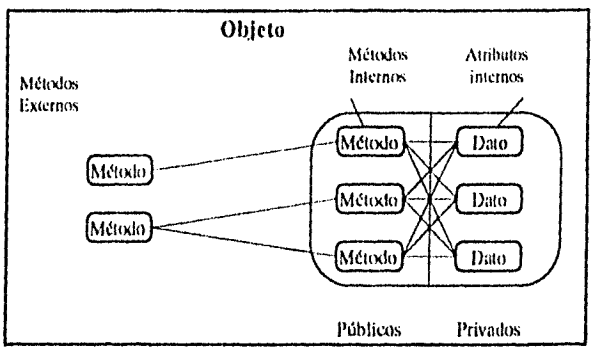

Figura 1.1 Objeto como un módulo que encapsula atributos y métodos en una sola entidad.

Mensajes y Métodos.

A diferencia de los sistemas tradicionales, en el enfoque de la POO los objetos son elementos del software que tienen la posibilidad de actuar unos con otros. Esta interacción entre objetos se logra a través del envío de mensajes pidiendo que se ejecute un método especifico. Así un método describe una secuencia de acciones a realizar por un objeto; el método puede considerarse análogo a un procedimiento o una función en Pascal, la diferencia radica que en lugar de formar parte de un gran programa está incluido en el objeto. El objeto que envía un mensaje se conoce como *transmisor* y a al que lo recibe como *receptor*. Este receptor determina por sí mismo que método (i.e. comportamiento o

manipulación) debe llevar a cabo. La estructura del objeto protege sus descripciones al resto del sistema. El programador sabe que un objeto puede realizar una función particular, pero no los detalles de cómo la realiza. De esta forma la abstracción de datos es natural y hay ocultamiento de la información.

Clases e Instancias.

Los sistemas de software orientados a objetos hacen una distinción entre las descripciones del objeto y el objeto mismo. Las descripciones del objeto (léase los métodos y los datos que resumen las características comunes de un conjunto de objetos) son la *clase*. Una clase es entonces una colección de objetos que tienen los mismos atributos y un comportamiento común. La clase identifica los atributos que cada objeto de la clase tendrá, también identifica los métodos aplicables a tales objetos. Así todos los objetos de una clase dada tendrán los mismos métodos aplicables a ellos. Una clase no es una entidad física sino una abstracción del concepto de objeto. El objeto que como elemento de software el programador manipula es en realidad una *instancia* de una clase.

Herencia.

Las clases de objetos deben definirse de tal forma que forman una estructura jerárquica. Objetos definidos a un nivel más bajo de la jerarquía heredan los atributos de los objetos de niveles superiores. Este es uno de los aspectos más importantes del modelo de orientación a objetos. Esta característica permite definir, en los objetos de más alto nivel en la jerarquía, comportamientos que se aplicarán en niveles más bajos. La clase de mayor nivel es la superclase mientras que la clase inferior es la subclase.[13] Usualmente la subclase hereda los datos y los procedimientos de una superclase, (Figura 1.2); estas

características heredadas pueden modificarse al redefinir los datos o los métodos en las subclases.

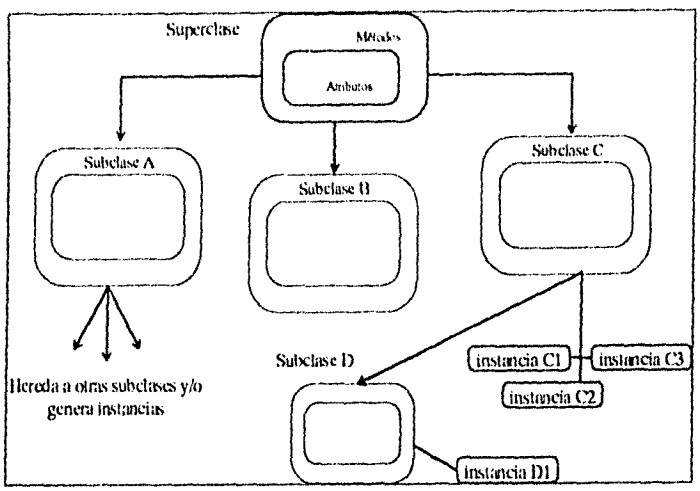

Figura 1.2 Ejemplo de una jerarquía de herencia.

Encapsulación.

La encapsulación es uno de los términos fundamentales en el paradigma de la orientación a objetos. Se refiere ha las cualidades privadas de un objeto y exclusivas para ese objeto, otro objeto aun de la misma clase puede no tener acceso a esos atributos y/o métodos *privados* del primer objeto. De esta forma la comunicación dentro de un conjunto de objetos se da exclusivamente por medio de mensajes explícitos. La encapsulación es un mecanismo que permite ocultar los detalles de la representación interna de un componente. La encapsulación preserva la integridad de un objeto, las implementaciones internas pueden estar cambiando, pero la interfaz visible para los otros objetos permanece

constante. En conjunto con la abstracción, la encapsulación permite definir interfaces entre los aspectos internos y externos del objeto. Proveen un poderoso mecanismo para especificar objetos que sean independientes unos de otros.

Abstracción.

La abstracción es un proceso cognoscitivo por medio del cual se identifican las características relevantes para cierto fin de una entidad dejando de lado las características que no son de interés para tal fin. Las metodologías orientadas a objetos se caracterizan por asociar abstracciones del mundo real con componentes computacionales que reflejan las características de la abstracción. El hecho de que los lenguajes orientados a objetos permitan a los programadores usar objetos definidos previamente como clases abstractas y estructuradas bajo una relación de herencia dentro de la jerarquía de clases, los convierte en un herramienta de abstracción sumamente poderosa. Cada nivel de abstracción (objeto, clase, jerarquía) facilita el trabajo de programación. Sin embargo, el proceso de categorizar objetos dentro de clases que compartan propiedades similares implica realizar una *clasificación* de objetos como modelos abstractos. La abstracción es el concepto clave que permite al programador generar código reutilizable.

Polimorfismo.

Es la habilidad de una operación de tomar formas diferentes cuando es aplicado a objetos diferentes. La idea que subyace en el concepto de polimorfismo es que operaciones semánticamente equivalentes, deben presentar la misma interfaz aun cuando están siendo aplicadas a diferentes objetos.

Como hemos mencionado arriba, los objetos actúan en repuesta a los mensajes que reciben; sin embargo un mismo mensaje puede originar comportamientos completamente diferentes al ser recibido por objetos diferentes. A esto se le conoce como polimorfismo. El concepto de polimorfismo es tal vez el más difícil de entender pero en el radica la clave de la flexibilidad de los sistemas de software orientados a objetos. El polimorfismo establece que un objeto que envía mensajes a otro no necesita saber exactamente la clase a la que pertenece el segundo objeto, simplemente que el objeto puede entender el mensaje que esta recibiendo.

Persistencia.

La mayoría de los lenguajes orientados a objetos van creando modelos de clases cuando un programa se ejecuta. Algunos de estos modelos de clases son necesarios por un breve período de tiempo. Cuando un objeto ya no es necesario es destruido y el espacio de memoria que tenía asignado es recuperado. El concepto de persistencia se refiere al tiempo y/o al espacio durante el cual los objetos permanecen accesibles en la memoria de la computadora, o en algún dispositivo de almacenamiento no volátil. Declarando a un objeto persistente el programador se asegura que éste permanecerá *guardado* (en algún archivo en disco duro, por ejemplo) al salir del programa, y que puede ser recuperado en la siguiente invocación del programa. En el ámbito de las bases de datos la persistencia es una de sus características intrínsecas, sin embargo en el caso de las bases de datos orientadas a objetos se distingue entre los objetos creados únicamente para el momento de la ejecución de la aplicación y los objetos persistentes, es decir aquellos que se almacenan permanentemente.

CONCEPTOS Y TERMINOLOGÍA RELACIONADA

La introducción de las técnicas orientadas a objetos para el desarrollo de software, ha generado la creación de nuevos conceptos y términos que no son fundamentales para el modelo de orientación a objetos pero que son aplicables a ciertas áreas donde se ha utilizado la orientación a objetos, o bien, que sirven de puente entre las técnicas convencionales y las orientadas a objetos. Aquí revisaremos tres de estos términos relacionados: la ligadura dinámica, herencia múltiple y el concepto de BLOB.

Ligadura Dinámica.

La ligadura se define como el proceso de entrelazar un programa de manera que todos sus componentes estén conectados adecuadamente. Esta ligadura puede ser estática o dinámica [14]. La ligadura estática sucede cuando todas las conexiones se dan antes de se empiece a ejecutar el programa; en los lenguajes convencionales la ligadura se efectúa durante la compilación. La ligadura dinámica sucede cuando la ligadura se realiza al ejecutarse el programa. El término se refiere a la habilidad del sistema para seleccionar la operación más apropiada para un objeto al momento de la ejecución. Esto implica que si una operación dentro de un objeto ha sido redefinida después que ha iniciado la ejecución del programa, una invocación a tal operación es capaz de usar la última forma y aplicarla en el objeto que la solicita. Los lenguajes orientados a objetos usan comúnmente la ligadura dinámica, en situaciones donde el ambiente requiere la selección, al momento de la ejecución, entre diferentes versiones de una operación. La ligadura dinámica es una consecuencia de la puesta en práctica del polimorfismo. Para Booch el polimorfismo y la ligadura dinámica van de la mano:

> "Con polimorfismo, la ligadura de un método con un nombre no es determinada sino hasta la ejecución." [15]

Herencia Múltiple.

La herencia múltiple se refiere a la habilidad de una clase para heredar propiedades de más de una superclase. Al especificar que una subclase se derive con herencia múltiple, el programador le esta permitiendo tomar la estructura y el comportamiento de más de una superclase. La necesidad de la herencia múltiple surge en el modelado de situaciones complejas del mundo real, en las que un objeto exhibe las características de más de una superclase. La mayoría de los lenguajes orientados a objetos suportan la herencia múltiple.

BLOB.

La introducción de los conceptos de orientación a objetos al ámbito de las bases de datos ha generado la creación de un tipo de datos llamado BLOB (Binary Large OBject) el cual puede almacenar cualquier forma de datos binarios. Los programadores pueden escribir procedimientos para guardar información en un BLOB y accesarla después. Esto permite manejar bases de datos con tipos de datos no tradicionales como imágenes y voz. El polimorfismo es nuevamente la clave que permite a estas bases de datos manipular (almacenar, encontrar y recuperar) estos objetos, y otros definidos por el programador, sin conocer de antemano todos los tipos de datos posibles.

VENTAJAS Y DESVENTAJAS DEL ENFOQUE DE ORIENTACIÓN A OBJETOS.

En esta parte revisaremos las principales ventajas y desventajas del enfoque de orientación a objetos para el desarrollo de sistemas de computación. El uso de estas técnicas representa ventajas para el usuario, por una parte, y para los desarrolladores en varios aspectos. Una aplicación orientada a objetos es más semejante al modelo del mundo del usuario que una aplicación desarrollada por métodos tradicionales. Usualmente las aplicaciones orientadas a objetos mantienen una correspondencia casi uno a uno con los objetos del mundo real y los objetos de la aplicación. Esto hace que la traducción de las necesidades del usuario a la aplicación completa sea más sencilla y clara. Las compañías desarrolladoras de software están poniendo énfasis en la estrategia basada en el concepto de los prototipos rápidos [16], el cual consiste en hacer a los usuarios parte integral del equipo de desarrollo. Los lenguajes orientados a objetos permiten a los desarrolladores modelar las características esenciales de la aplicación rápidamente porque la funcionalidad está contenida dentro de los mismos objetos. Y más importante aún, después de recibir la retroalimentación del usuario, el trabajo de desarrollo se centra en perfeccionar y mejorar el prototipo para llegar a la aplicación completa. Los lenguajes de programación tradicionales no ofrecen la flexibilidad ni los componentes de software preexistentes necesarios para la implementación efectiva de los prototipos rápidos.

Otra ventaja importante de los lenguajes de programación orientados a objetos es que ofrecen la posibilidad de reutilización del código, gracias en gran parte al polimorfismo y a la herencia. La generación de bibliotecas de clases proporcionan una plataforma para el desarrollo de nuevas aplicaciones de manera rápida y eficiente. Los desarrolladores pueden usar y ampliar las bibliotecas en vez de reescribir el código partiendo de cero. Esto es especialmente cierto para componentes de las GUI, los cuales son el común denominador para las aplicaciones que se ejecutan sobre estos. Los lenguajes orientados a objetos como Smalltalk proveen componentes del sistema que

forman los bloques de construcción para virtualmente toda aplicación futura.[17] La reutilización del código tiene como consecuencia obvia aumentar la productividad y reducir los costos de desarrollo a largo plazo.

En un aplicación orientada a objetos la ampliación y extensión de sus caracterfsticas así como su mantenimiento son más sencillos, ya que la herencia permite definir nuevos objetos a partir de los existentes. Además los métodos al estar en una única locención son más fáciles de modificar.

Sin embargo, en ciertas circunstancias los métodos orientados a objetos no son superiores o siquiera comparables a los métodos estructurados. Tal es el caso de aplicaciones que involucran el manejo intenso de operaciones numéricas. Esto se debe en gran medida al hecho de que existen mucho mejores herramientas de desarrollo de software que tienen un largo historial dentro del enfoque estructurado. Los lenguajes orientados a objetos aún carecen en su mayoría de software para controlar y probar código fuente en gran escala. Otra desventaja de las aplicaciones orientadas a objetos es que corren más lento que sus contrapartes desarrolladas con metodologías estructuradas.

A pesar de sus deficiencias el enfoque de programación orientada a objetos está mostrando ser una herramienta de primera línea para el desarrollo de aplicaciones. La flexibilidad que ofrece radica en gran medida en su capacidad para extender y cambiar rápidamente un sistema. Esto distingue al enfoque de orientación a objetos de las técnicas anteriores.

---

[1] Nuncio Limón, Reynaldo., "Historia y Perspectivas de la Programación", Ed. Trillas, México, 1991, pp 93-96

[2] Nuncio Limón, Reynaldo., *Op Cit*, pp 91-92

[3] Boreham, T.,"Object Oriented Analysis and Design for PowerHouse", NUTAT Technologies Inc., paper # 103 en "Directions 1992, Cognos International User Conference" pp. 3-19

[4] Oktaba, H.,"Programación Orientada a Objetos. ¿Moda o realidad?", Soluciones Avanzadas, No. 3, abril-mayo, 1993, pp.39-42

[5] Booch, Grady., "Objet Oriented Design with aplications", Benjamin/Cunamings, 1991, p 474

[6] DeMarco, Tom., "Structured Analysis and System Specification" , Yourdon Press/Prentice Hall, 1979. En este texto se definen por primera vez los conceptos de la metodología del análisis estructurado.

[7] Yourdon, E., and Constantine, L., "Structured Design: Fundamentals of a Disipline of Computer Programming and Design", Prentice Hall, 1979. Vease también: Yourdon, E., "Modern Structured Analysis", Yourdon Press, 1989 para una actualización del análisis estructurado.

[8] Ward, P.T. and Mellor, S.J., "Structured Development of Real-Time Systems", Yourdon Press, 1985. En esta obra se extienden las ideas del análisis estructurado para soportar mejor el modelado de sistemas en tiempo real.

[9] Oktaba, H., *Op Cit* p.39

[10] Varhol, P.,"Object Oriented Programming: The Software Development Revolution", Computer Technology Research Corp., 1993 p. 64

[11] Boreltam, T., *Op Cit*, pp. 3-19

[12] Oktaba, H.,"Análisis y Diseño Orientedo a Objetos. Introducción al Método de Booch", Soluciones Avanzados, No. 6, noviembre-diciembre, 1993, pp.18-22

[13] En la literatura tambien se le conoce como la clase padre o meta clase y a las subclases se les llama cleses hijas.

[14] En la literatura pueden encontrase estos términos como *early binding* o *static binding* para ligadura estática y *late binding* o *dynamic binding* para ligadura dinámica o tardía.

[15] Booch, Grady., *Op Cit*, p. 104

[16] Un prototipo es una versión "borrador" (llamada comunmente versión beta) de una aplicación o un producto de software comercial, la cual es distribuida entre cierto número de usuarios para que estos la prueben y hagan sugerencias de modificaciones y mejoras.

[17] Varhol, P.,*Op Cit*, p. 55

CAPÍTULO 2

ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS

En esta parte presentaremos los componentes más importantes de los métodos de análisis y diseño de sistemas orientados a objetos.

Cuando se va desarrollar un sistema de software las etapas iniciales son las de análisis y diseño del sistema. Estas etapas son fundamentales ya que sirven para precisar cuál es el problema (su ámbito y dimensiones) y cómo se va a solucionar. Cuando se sigue un proceso de análisis y diseño orientado a objetos se entiende que las representaciones de los dominios del problema (etapa de análisis) y de la solución (etapa de diseño) están expresadas en términos del modelo de objetos.[1]

El análisis orientado a objetos se centra en el espacio del problema (i.e. en definir el problema), en tanto que el diseño orientado a objetos se centra en proveer, al usuario, la funcionalidad que ha sido identificada. Un método de análisis y diseño orientado a objetos debe proponer los pasos a seguir para construir el modelo del sistema en términos de clases, objetos y de las relaciones entre ellos.[2] También debe proporcionar algún tipo de notación que ayude a representar la información que se genere en las distintas fases. En general, se necesitan dos tipos de notación para representar los dos aspectos más importantes de un sistema: su estructura estática y su comportamiento dinámico.[3] A estos aspectos también se les conocen como la vista estática y la vista dinámica de un problema. La notación usada para representar la vista estática debe incluir herramientas que denoten clases, objetos, los atributos y su comportamiento, así como las relaciones entre las clases y los objetos. Para la vista dinámica, la notación debe incluir herramientas para denotar la comunicación entre objetos, sus cambios de estado, la temporización de eventos, etc. En los últimos siete años se han publicado varias propuestas de métodos de análisis y diseño,

entre las más destacadas por su consistencia y utilidad, podemos mencionar: el método de Booch,[4] el de Coad y Yourdon,[5] la técnica de modelado de objetos (OMT por sus siglas en inglés Object Modeling Technique) de Rumbaugh et al.[6] y el de Wirfs-Brock et al.[7]

La utilidad práctica de un método en particular depende de los apoyos y herramientas que aporte para expresar la estructura de un sistema por complejo que este sea. Un método de análisis y diseño orientado a objetos, puede considerarse de calidad si cumple con los siguientes criterios [8]:

1) "Debe proporcionar herramientas para la definición de las vistas estáticas de un sistema. Para ello se necesitan reglas de identificación y colocación de clases semánticas, sus atributos y sus relaciones de generalización, agregación y otras. Para expresar las vistas estáticas se requiere de una notación gráfica que sea entendible y fácil de utilizar. Es deseable la inclusión, en el diseño, de las clases de aplicación, utilerías e interfaz.

2) Debe proporcionar herramientas y notación para la definición del comportamiento dinámico del sistema, es decir de las vistas dinámicas. Esta parte es especialmente importante cuando el sistema que estamos diseñando es distribuido, paralelo y/o tiene requerimientos de sistemas de tiempo real.

3) Debe ayudar a manejar la complejidad del sistema a nivel estructural, es deseable que lo haga también a nivel dinámico."

Los métodos de análisis y diseño orientado a objetos mencionados arriba son los que mejor concuerdan con los criterios anteriores. En particular Grady Booch, sin duda uno de los pioneros en el desarrollo de las metodologías orientadas a objetos, ha desarrollado un método gramatical en el que sugiere que el diseñador inicie con la descripción en prosa del sistema a desarrollar y partiendo de los nombres que identifiquen a los candidatos a ser clases de objetos. Los verbos servirán para identificar los métodos. Este método ha demostrado ser eficiente y se caracteriza por ser evolutivo, es decir, que al irse identificando las clases y sus relaciones y encontrar nuevas relaciones entre las clases, se puede ir cambiando en el momento que sea necesario.

A continuación revisaremos con más detalle el proceso de análisis y diseño orientado a objetos tratando de exponer los conceptos sin comprometernos con algún método particular.

Análisis orientado a objetos.

La cuestión fundamental en el análisis orientado a objetos (AOO) es identificar a los objetos que formarán parte del sistema. En esta fase se descubrirán los objetos semánticos del dominio del problema. Identificar los objetos semánticos es probablemente la tarea más difícil del proceso, porque estamos definiendo abstracciones que representan conceptos que tengan un significado claro en el contexto del problema. Los analistas son generalmente los miembros de los equipos de desarrollo con mayor experiencia porque el proceso de análisis es el proceso de *descubrir* y *resolver* el problema y requiere mucha creatividad. El proceso de AOO puede dividirse en tres actividades principales:

a) Identificar en las clases de objetos semánticos, los atributos y las operaciones que describen el comportamiento de los objetos.

b) Colocación de los atributos y las operaciones dentro de las clases, así como la definición de las relaciones de generalización, agregación y asociación entre las clases.

c) Especificación del comportamiento dinámico de los objetos.

No es necesario que estas actividades se realicen en serie como una receta, sino más bien deben considerarse como una guía, para llegar al objetivo principal del AOO que es identificar el comportamiento del sistema. En esencia, estamos respondiendo a la pregunta: ¿cómo responde el sistema a los eventos?. La idea es determinar que objeto es responsable de manipular y coordinar la respuesta para cada evento.

Una vez que los objetos han sido identificados lo que sigue es buscar las relaciones o asociaciones entre ellos. Una manera de determinar estas relaciones es tratando de clasificarlas con base en estas tres categorías:

- "*Es un*", o "*es una especie de*"
- "*Es una parte de*", o "*forma parte de*"
- Otras

La relación "*es un*" es fundamental en el mundo de la orientación a objetos. Esta relación es usada para construir la jerarquía de herencia. También ayuda a definir las relaciones de generalización y especialización entre las clases. La segunda relación que podríamos llamar de *ensamblado* es también importante porque la estructura de ensamblado debe estar oculta para el mundo exterior. Permite definir las relaciones de agregación. Por ejemplo, si tenemos un automóvil como una reunión de partes, el conducirlo no requiere conocimiento de la estructura de su ensamblado. Finalmente, las relaciones restantes se clasifican como *otras*, permiten definir las asociaciones entre las clases.

El siguiente paso es identificar los eventos que afectan al sistema y determinar la respuesta del sistema a estos eventos. Esto puede lograrse considerando el ciclo de vida para cada objeto. El ciclo de vida determina cómo se va a crear el objeto, cómo va a cambiar durante su existencia y, en su caso, cómo es borrado o eliminado del sistema.

Al final de la última etapa, revisamos nuestro modelo en términos de *responsabilidad* de cada objeto. Este proceso de revisión debe conducirse a través de preguntarnos, para cada *elemento de conocimiento*: ¿Cúal objeto es responsable de mantener la información y asegurar su consistencia?. Y para cada *elemento de comportamiento*: ¿Cúal objeto es responsable de manejar el comportamiento y asegurar

una respuesta completamente satisfactoria a cualquier evento?. En resumen y tomando la definición de Booch:

> "El análisis orientado a objetos es un método de análisis que examina los requerimientos [del sistema] desde la perspectiva de las clases y objetos encontrados en el vocabulario del dominio del problema." [9]

Diseño orientado a objetos.

En el diseño orientado a objetos (DOO) el objetivo es hacer el modelado del dominio de la solución, para ello se descubren otros objetos que son útiles para llegar a la solución del problema. Estos son objetos de interfaz como pantallas y objetos reporte necesarios para soportar la interfaz del usuario. Una vez establecidos se identifica como están relacionados con los objetos del espacio del problema. Estos objetos adicionales pueden clasificarse en tres grupos: [10]

Objetos de interfaz: interfaz con el usuario.

Objetos de aplicación: para la inicialización y control de la aplicación.

Objetos de utilerías: objetos reutilizables e independientes de aplicación como listas, cadenas, arreglos, etc.

El DOO incluye las actividades a), b) y c) mencionadas en el proceso de análisis pero esta vez dedicadas a la identificación, colocación y especificación de las clases adicionales.[11] Una fase adicional es la de optimización de las clases, la cual puede definirse como:

d) Reestructuración de las jerarquías y relaciones entre las clases debido al descubrimiento de ciertas posibilidades de mejorar el diseño como por ejemplo: reutilización de clases ya existentes, la generalización introduciendo clases abstractas, la partición de clases demasiado grandes, etc.[12]

Diseñar el espacio del problema consiste en determinar cómo construir los objetos identificados en la etapa del análisis y cómo implementar su comportamiento. La mayoría de los ambientes de programación orientados a objetos están provistos con objetos que pueden usarse y adaptarse para el espacio del problema particular; en otros casos ya se tiene un trabajo previo. Cuando es posible encontrar tales objetos, los del espacio del problema pueden heredar de ellos, o aquellos pueden extenderse para cubrir los requerimientos del espacio del problema actual. Para las relaciones se determinan las rutas de acceso para ser usadas por los objetos, esto nos permite establecer cómo pueden ser representadas sus relaciones.

También en el diseño se debe considerar el problema del almacenamiento de la información. Es importante determinar cómo representar los objetos en el ambiente de almacenamiento en que se ésta trabajando. Algunos objetos pueden ser temporales y puede no ser necesario mantenerlos durante las invocaciones del programa.

Cuando se ha determinado la interacción entre los objetos de la interfaz del usuario y los objetos del espacio del problema, se puede decir que el diseño esta completo.

El proceso de análisis y diseño orientado a objetos.

El proceso de análisis y diseño orientado a objetos no se realiza como una receta que sigue una serie de pasos, sino más bien consiste en la construcción de un modelo a partir de ciertas preguntas, relacionadas con la estructura lógica del sistema (análisis) y con respecto al mapeo físico del sistema a los módulos de programas y a la arquitectura particular de las máquinas (diseño). El proceso de análisis y diseño orientado a objetos consiste entonces en ir *descubriendo* las clases de objetos que modelarán el espacio del

problema y en ir *inventando* las clases adicionales que ayudaran a proveer la solución; a la vez que ir *adaptando* lo que ya existe en lo modelado previamente y lo que se pueda usar de los ambientes de programación. Así el proceso de análisis y diseño orientado a objetos es iterativo, en el que las diferentes etapas pueden intercalarse y repetirse aleatoriamente con la idea de aportar la mejor solución del problema.

Aunque aquí hemos tratado el análisis y diseño orientado a objetos como dos actividades conceptualmente distintas, en la práctica la línea que divide el análisis del diseño es muy borrosa. En la literatura se encuentran propuestas de metodologías *de análisis* o *de diseño* y otras de *análisis y diseño*. El hecho es que cualquier metodología es un proceso de producción de software que usa un conjunto de técnicas específicas y una notación. En dicho proceso la frontera entre las etapas de análisis y diseño puede ser muy tenue o incluso inexistente, en el caso de las técnicas orientadas a objetos. En general los autores considerados *analistas* tienden a enfocar su trabajo hacia la estructura de los objetos, en tanto que la corriente de los *diseñadores* se concentran más en el comportamiento dinámico de los objetos.

La programación orientada a objetos tiene su marco teórico en el modelo de objetos. Para Booch existen cuatro elementos principales del modelo de objetos [13] a saber: abstracción, encapsulación, modularidad y jerarquía. Sin alguno de estos elementos un modelo no puede llamarse orientado a objetos. El mismo autor considera a su vez tres elementos secundarios, es decir, útiles pero no esenciales para el modelo de objetos: tipificación, concurrencia y persistencia.

Bapat [14] es más estricto al presentar los conceptos del modelo de objetos, de acuerdo a este autor las nociones básicas del modelo son: abstracción, encapsulación, clasificación, herencia, extensión y evolución; y las construcciones que expresan estas nociones son los atributos, las funciones y las clases.

El paradigma de orientación a objetos es un paradigma para la creación de modelos, es decir *una perspectiva para desarrollar metodologías que permitan modelar sistemas de software*. Es la expresión más acabada, desde el punto de vista teórico, en la evolución de las teorías formales para el desarrollo de sistemas. Wegner [15] distingue los siguientes paradigmas para la creación de modelos dentro del enfoque de orientación a objetos:

Modelado Basado en Objetos: Paradigma que soporta las nociones de abstracción y encapsulación.

Modelado Basado en Clases: Paradigma que además de ser basado en objetos, soporta clasificación.

Modelado Orientado a Objetos: Paradigma que además de ser basado en clases soporta herencia.

A pesar de que entre la comunidad de investigadores existe consenso en cuanto a los conceptos del modelo de objetos, es un hecho que el propio modelo aun carece de una especificación formal.[16] Sin embargo, para fines prácticos el modelo funciona, y ha propiciado el surgimiento de metodologías de análisis y diseño basadas en alguno de los paradigmas mencionados arriba. Estas metodologías encuentran su aplicación en la ingeniería de software. El modelo de objetos es importante porque pone en manos de los desarrolladores y de los programadores (con independencia de la metodología OO que apliquen) un poderoso marco teórico para enfrentarse al diseño de aplicaciones que resuelvan problemas complejos.

No basta con conocer un lenguaje de programación orientado a objetos para desarrollar una aplicación orientada a objetos. De hecho, existen diferencias entre la programación orientada a objetos y el modelado orientado a objetos.[17] Este último es más abstracto, menos detallado, y más amplio en su vista del sistema, en tanto que la programación orientada a objetos esta más enfocada con los detalles de diseño de bajo nivel del sistema y con la implementación y la eficiencia. El modelado se concentra más en

el *que*, esto es, en la *especificación* del sistema. En tanto que los programadores se concentran más en el *como*, o sea en la *implementación* del sistema. En la tabla 2.1 se resumen algunas de las diferencias entre la programación orientada a objetos y el modelado orientado a objetos. La programación orientada a objetos y el modelado de objetos se complementan al momento de desarrollar un sistema de software, siendo ambas beneficiosas el alguna de las diferentes fases del ciclo de vida del sistema.

CARACTERÍSTICAS	Programación orientada a objetos	Modelado orientado a objetos
Abstracción	Si	Si
Encapsulación	Si	Si
Atributos	Si	Si
Clases	Si	Si
Herencia	Si	Si
Extensión	Si	Si
Instanciación	Declarativa	Depende del sistema
Agregación	Si	Si
Reusabilidad	Implementaciones	Especificaciones
Herencia múltiple	Si	Si
Polimorfismo	Si	No significativo
Ligadura dinámica	Algunos ambientes	No significativa
Persistencia de objetos	Algunos ambientes	No significativo
Recolección de basura	Algunos ambientes	No significativo
Inspección de tipos	La mayoría de los ambientes	No significativo

Tabla 2.1 Algunas diferencias entre la programación orientada a objetos y el modelado orientado a objetos.

[1] Oktaba, H.,"Análisis y Diseño Orientado a Objetos", Soluciones Avanzadas, No. 4, julio-agosto, 1993, pp.8-11

[2] Oktaba, H.,"Análisis y Diseño Orientado a Objetos. Introducción al Método de Booch", Soluciones Avanzadas, No. 6, noviembre-diciembre, 1993, pp.18-22

[3] Oktaba, H.,"Análisis y Diseño Orientado a Objetos", Soluciones Avanzadas, No. 4, julio-agosto, 1993, p 10

[4] Booch, G., "Object Oriented Design with aplications", Bemjamin/Cummings, 1991. Vease también su artículo: "What Is and What Isn't Object-Oriented Design", Am. Programmer, Vol. 2, No. 7-8, Verano 1989, pp. 14-21

[5] Coad, P., & Yourdon, E., "Object-Oriented Analysis", Prentice Hall, 1991.

[6] Rumbaugh, J. et al. "Object-Oriented Modeling and Design", Prentice Hall, 1991.

[7] Wirfs-Brock, R. R., et al,"Designing Object-Oriented Software", Prentice Hall, 1990.

[8] Oktaba, H.,"Análisis y Diseño Orientado a Objetos", Soluciones Avanzadas, No. 4, julio-agosto, 1993, p 11

[9] Booch, Grady., "Object Oriented Design with aplications", Bemjamin/Cummings, 1991, p 37

[10] Oktaba, H.,*Op Cit*, p 9

[11] *Ibidem*, p 9

[12] *Ibidem*, p 9

[13] Booch, G., "Object Oriented Design with aplications", Bemjamin/Cummings, 1991, pp. 38

[14] Bapat, S.,"Object-Oriented Networks. Models for architecture, operations, and management ", Prentice Hall, 1994.

[15] Bapat, S., *Ibidem*, p 32

[16] Wegner, Peter,"Interactive Foundations of Object-Based Programming", Computer, October 1995, pp. 70-72

[17] Bapat, S., *Ibidem*, pp. 36-49

CAPÍTULO 3

EL PANORAMA DEL SOFTWARE ORIENTADO A OBJETOS.

LENGUAJES ORIENTADOS A OBJETOS.

En un sentido teórico estricto, los lenguajes orientados a objetos son aquellos que soportan los mecanismos y los conceptos propios del modelo de orientación a objetos, a saber: objetos, clases, métodos, mensajes y herencia. En la práctica los lenguajes que tienen tal grado de *pureza* se pueden contar con los dedos de una mano. A pesar de que el modelo empezó a generarse en la década de los sesentas, es hasta los años ochenta que toma fuerza en su desarrollo comercial. En la actualidad existen dos grupos de lenguajes que soportan el modelo. El primero de ellos podemos llamarlo de lenguaje puro orientado a objetos en el que se incluye: Smalltalk, Actor y Eiffel. El otro es el grupo híbrido en el que a ciertos lenguajes basados en procedimientos, se han agregado construcciones que les permiten soportar el modelo de orientación a objetos. En este trabajo se estudian los lenguajes puros mencionados arriba y del grupo de lenguajes híbridos: C++, Objective-C, Pascal orientado a objetos, LISP y XLISP.

Smalltalk.

El lenguaje orientado a objetos Smalltalk tiene sus orígenes en las ideas y en los trabajos de Alan Kay [1] y del Learning Research Group (ahora conocido como Software Concepts Group) de XEROX en Palo Alto California a principios de los setentas. La idea central de sus creadores era: "crear un poderoso sistema de información, en el que el usuario pueda almacenar, accesar y manipular información de tal forma que el sistema

crezca en la medida que las ideas del usuario crezcan".[2] Entre 1970 y 1980 Smalltalk experimentó varias revisiones en ciclos bianuales en PARC. Generalmente el lenguaje no estaba disponible fuera del laboratorio hasta 1984 cuando Digitalk lanzó la primera versión comercial al mercado. Sin embargo, Smalltalk fue incorporado en los ambientes de las computadoras, Alto, Star y Dorado de XEROX.

Smalltalk es más que un lenguaje orientado a objetos. Combina elementos de un sistema operativo, un ambiente de programación y una interfaz de usuario. El ambiente Smalltalk está construido sobre cuatro principios:

- Smalltalk es una visión
- Smalltalk esta basado en unos cuantos conceptos y estos están definidos en una terminología inusual
- Smalltalk es un ambiente de programación gráfico e interactivo
- Smalltalk es un sistema grande

La visión de Smalltalk era desarrollar un sistema que tuviera la capacidad de crecer como crecía la experiencia del usuario. Esto que ahora es la retórica obligada de la mayoría de los productores de computadoras de la actualidad fue un concepto radical hace casi veinticinco años. Los especialistas en hardware y software que describen a los productos como "valiosos para novatos pero mucho más útiles y poderosos en manos de usuarios experimentados" están describiendo el primero de los principios originales de Smalltalk.

El segundo principio se refiere al lenguaje usado para describir objetos y sus actividades. Las bases del lenguaje Smalltalk son objetos, mensajes, clases, instancias y métodos. Esta lista tan corta permite que los usuarios entiendan las bases del sistema casi inmediatamente y exploren el potencial de Smalltalk en su propio espacio. Sin embargo,

debido a que estos términos están estrechamente relacionados, es necesario entender todos con el propósito de entender cualquiera de ellos.

Smalltalk está diseñado de tal forma que cada componente (con excepción del núcleo o kernel) del sistema pueda ser presentado de forma que los usuarios puedan observarlos y modificarlos significativamente. Aunque esto hace a Smalltalk susceptible a "caídas" del sistema tras leves modificaciones, también lo hace muy flexible. La interfaz con el usuario es el resultado del esfuerzo de Smalltalk por crear un lenguaje visual para cada objeto. Fue el primer ambiente en requerir la interacción directa del usuario vía un dispositivo apuntador y aún le da al usuario mayor control sobre el sistema que cualquier otro sistema de software comercial.

Smalltalk incluye objetos que proveen funciones usualmente atribuidas al sistema operativo, incluyendo: administración de almacenamiento, sistema de archivos, manejo del display, entrada del teclado y del dispositivo apuntador, planeación de procesos, compilación, depuración y monitoreo del desempeño.

Smalltalk fue el primer sistema de software orientado a objetos que es completamente consistente con la jerarquía orientada a objetos. La clase Objeto es la superclase para todas los otros objetos y define los protocolos comunes para todos éstos en el sistema. Define los comportamientos por omisión para desplegar, copiar, comparar e inspeccionar objetos. La clase Objeto tiene capacidad para mantener las relaciones entre los objetos y enviar mensajes de un objeto a sus dependientes.

Todas las otras clases de objetos son subclases de la clase Objeto. Esto incluye a las clases para describir ventanas, gráficas, texto y cada aspecto del sistema Smalltalk.

En 1983 Jim Anderson un experimentado desarrollador de software fundó Digitalk la primera compañía de herramientas de software comercial, dedicada a proveer a los programadores con productos de desarrollo de software orientado a objetos. Aunque no

fue uno de los investigadores originales de Smalltalk, cuando Anderson conoció el lenguaje se convenció de que tenía el potencial para transformar el proceso de desarrollo de software. Su versión conocida comercialmente como Smalltalk/V es el lenguaje de programación orientado a objetos puro de mayor uso en la actualidad. Tiene más de 300 clases de objetos predefinidas, construidas en una jerarquía que reensambla un diagrama de árbol, asegurando que las clases de niveles inferiores puedan heredar características de las clases superiores. Smalltalk/V también posee mas de 1,000 métodos predefinidos, lo que genera una biblioteca de subrutinas extremadamente grande; además, los métodos no están compilados por lo que pueden modificarce y adaptarse. Actualmente pueden encontrarse versiones de Smalltalk/V para ambientes DOS, OS/2, MS-Windows y Macintosh. También la versión de IBM comercializada como VisualAge/Smalltalk es una de las más populares.

En la tabla 3.1 presentamos una comparación de cuatro lenguajes de programación orientados a objetos, con base en las principales características del modelo.

CARACTERÍSTICAS	Smalltalk	Eiffel	Objective-C	C++
Abstracción	Si	Si	Si	Si
Encapsulación	Si	Si	Si	Si
Clases	Si	Si	Si	Si
Estructura para Clases	superclass	parent class	factory class	base class
Herencia	Si	Si	Si	Si
Herencia Múltiple	No	Si	No	Si
Polimorfismo	Si	redefinición	Si	función virtual
Sobrecarga de op.	Si	No	No	Si
Ligadura dinámica/estática	dinámica	ambas	ambas	ambas
Validación de tipos	dinámica	estática	estática	estática
Admón. de memoria	automática	automática	programador	programador
Recolección de basura	Si	Si	No	No

Tabla 3.1 Comparación de algunas de las características de cuatro lenguajes de programación orientados a objetos.

Actor.

Actor es un lenguaje de programación y un ambiente gráfico para MS- Windows que fue desarrollado y comercializado por el Whitewater Group (Symantec Corp. compró Whitewater en Junio de 1992).[3] Actor implementa un enfoque de orientación a objetos que describe a los objetos individuales como "actores" que tienen un papel por desempeñar. Aunque el lenguaje usado para describir las construcciones de Actor es de

alguna manera diferente que los lenguajes de objetos convencionales, los conceptos subyacentes y los métodos de desarrollo de aplicaciones son muy similares. Actor es un método altamente factible para desarrollar aplicaciones con uso intensivo de interfaz gráfica de usuario bajo MS-Windows 3.x.

Eiffel.

Eiffel es un lenguaje de programación y un ambiente desarrollado por Bertrand Meyer y comercializado a través de su compañía: Interactive Software Engineering. Eiffel es también un ambiente de desarrollo en que se tiene un número de herramientas integradas alrededor del lenguaje. Fue diseñado específicamente para desarrollar la producción de software y ofrece herramientas y capacidades que agregan valor a los proyectos de desarrollo de software generados con él. La presentación que Meyer hace de Eiffel comienza por lo que él considera como los principios básicos de la ingeniería de software: reusabilidad, extensibilidad, confiabilidad, portabilidad y eficiencia.[4] El lenguaje Eiffel y su biblioteca de clases están optimizados para soportar estos conceptos a su máxima extensión posible.

El ambiente de Eiffel posee muchas de las características de Unix y de Smalltalk. Sin embargo a diferencia de Smalltalk, los archivos ejecutables compilados de Eiffel incluyen únicamente aquellas bibliotecas de clases necesarias para la ejecución, haciendo al programa ejecutable más pequeño y la ejecución más confiable.

Eiffel es una alternativa seria para cualquier organización de software ya sea únicamente porque el compilador de Eiffel primero compila el lenguaje a código en C, haciéndolo potencialmente portable a una gran variedad de plataformas y de sistemas operativos. Esto también significa que la plataforma de desarrollo es teóricamente independiente de la plataforma de liberación. Los programadores pueden usar hardware y sistemas operativos con la mejor interfaz o las mejores herramientas confiados en que su

producción de software puede ser portada relativamente fácil hacia la plataforma del usuario final.

C++

En 1986 Bjarne Stroustrop de los laboratorios Bell de la AT&T diseñó modificaciones para el lenguaje de programación C, originalmente diseñado e implementado por Brian Kernigan y Dennis Ritchie de la misma institución a principios de los años setenta. La meta original de Stroustrop no era agregar extensiones de orientación a objetos a C, sino más bien producir un lenguaje C "mejorado" . Quería corregir y modificar algunas de las desventajas de C a la luz de 15 años de avances en la ingeniería de software. El lenguaje resultante, liberado primero por la AT&T en 1988, fue llamado C++. Además de muchas construcciones orientadas a objetos, posee características que permiten manipulación de tipos de datos variables y complejos.

C++ soporta muchas de las mismas construcciones orientadas a objetos como en Smalltalk, pero dentro del contexto de programación del lenguaje C. Algunas características de C++ no son orientadas a objetos pero se incluyeron para ayudar al a los programadores que aún usan lenguaje C sin los aspectos orientados a objetos. Sin embargo, Stroustrop consideró el paradigma de objetos de gran importancia para la ingeniería de software. C++ puede así, ser usado para programar usando técnicas estructuradas o técnicas orientadas a objetos.

La definición del lenguaje C++ soporta el concepto de clases de objetos a través de la declaración CLASS, ésta es similar a STRUCT, con la diferencia que la construcción Class puede tener tipos de datos y funciones. C++ es un lenguaje de tipos de datos en que se requiere que las variables sean del mismo tipo y tamaño. En consecuencia un programa C correcto no es necesariamente un programa C++ correcto. C++ usa herencia múltiple en su estructura de objeto clase. Esto significa que un objeto de un nivel inferior en la

jerarquía de clases es capaz de heredar código y otras características de más de un objeto arriba de él. La jerarquía no es un árbol, como en Smalltalk, sino una red, pues permite herencia múltiple.

C++ a diferencia de Smalltalk requiere que el programador asigne y desasigne memoria manualmente. Aunque esto significa una pesada carga para los programadores generalmente resulta en que los programas corren más rápido por que no tienen que manejar la memoria mientras se ejecutan.

AT&T licenció C++ en forma de código fuente, el cual los vendedores generalmente portan a un plataforma de hardware o software en particular, agregando valor, y revendiéndolo bajo sus nombres. Tal es la estrategia que han seguido Glokenspiel, Symantec, Borland y Microsoft en sus ofertas del lenguaje C++.

Objective-C

Otro lenguaje de programación basado en C es Objective-C desarrollado por Brad Cox para Productivity Products International, actualmente conocida como Stepstone, Inc.[5] entre muchas de las características de Objective-C se incluyen una extensa biblioteca de clases y herramientas de asistencia en ingeniería de software y en procesos de administración de código.

Objective-C es un ambiente de desarrollo más completo que C++, pero difícil de aprender porque sus conceptos se desvían del tradicional C. Objective-C fue un favorito temprano para convertirse en el estándar para las extensiones de objetos del lenguaje C, pero fue eclipsado por el estándar de C++ impuesto por la AT&T. Objective-C llamó la atención como parte del entorno de desarrollo provisto con la familia de computadoras NeXT. Ahora que NeXTStep y el ambiente Objective-C han sido portados para correr en computadoras basados en procesadores Intel de 32 bits es probable que tengan mayor

impacto en la industria. Objective-C soporta los conceptos básicos de orientación a objetos tales como: objeto, clase, método y mensaje, subclase, herencia y herencia múltiple.

Pascal orientado a objetos.

Pascal orientado a objetos conocido como Object Pascal es un lenguaje híbrido basado en extensiones de objetos del lenguaje Pascal. En el mercado existen varias versiones de Object Pascal entre las más populares destacan: Quick Pascal de Microsoft y Turbo Pascal de Borland en el ámbito de las PC's compatibles IBM; en el entorno de las Macintosh Pascal orientado a objetos es asociado normalmente con MacApp, que proporciona una biblioteca de clases, y con el Macintosh Object Pascal para el desarrollo de aplicaciones para las Macintosh. Object Pascal es un producto específico de Apple Computer originalmente llamado Clascal. Las diferentes versiones de Pascal orientado objetos soportan los conceptos de: objeto, clases, métodos, mensajes y herencia.

LISP y XLISP

En 1983 David Betz intrigado por el potencial de los lenguajes orientados a objetos desarrolló la versión inicial de XLISP, una variante del lenguaje de programación funcional basado en listas LISP con extensiones simples de orientación a objetos.[6] Con la intención de hacerlo un herramienta de enseñanza aprendizaje para usarse en pequeñas computadoras, Betz lo hizo de dominio público y se ha convertido en la versión de mayor distribución en el mundo de LISP.

XLISP es un subconjunto significativo del Common LISP, el cual corre en computadoras PC, Macintosh y Atari. También existen versiones independientes para la Amiga de Commodore, la DEC VAX (Virtual Address Extension) y una variedad de

otras computadoras. XLISP es una elección natural para las extensiones a orientación a objetos debido a su extensibilidad y a sus poderosas herramientas de definición de funciones. XLISP tiene raíces académicas y de investigación similares a las de Smalltalk y fue el primer lenguaje, después de Smalltalk, en recibir extensiones de objetos.[7] El lenguaje XLISP hace fácil desarrollar estructuras de datos en forma de árbol, importante en la herencia. Como Smalltalk, XLISP también soporta el enfoque de "basurero" para reclamar memoria. Más importante tal vez , es que XLISP representa datos y programas en la misma manera. Es posible usar las mismas construcciones para referirse a datos y a funciones LISP, con el resultante comportamiento determinado ya sea si la construcción es un dato o unn función. La mayoría de las implementaciones de XLISP son entornos de programación que se emparentan con la complejidad de Smalltalk y soportan múltiples ventanas, inspección y depuración. El aspecto negativo de la herencia de programación en XLISP es la variedad de interpretaciones en relación a las funciones de objetos. Esto trajo como consecuencia la implementación de versiones incompatibles de LISP, las cuales han sido estandarizadas en un conjunto de extensiones conocido como CLOS (Common LISP Object System) [8] otras implementaciones populares de LISP en el ámbito de la inteligencia artificial sobre todo son Flavors y LOOPS.

En la figura 2.1 ilustramos una genealogía de lenguajes basados en objetos y de lenguajes orientados a objetos según la clasificación de Grady Booch. [9] De acuerdo a este autor existen alrededor de 100 diferentes lenguajes de programación orientados a objetos y basados a objetos. Por lenguaje basado en objetos, según Booch, se entiende aquel que soporta directamente abstracción de datos y clases. Un lenguaje orientado a objetos es aquel que además de ser basado en objetos soporta el concepto de herencia como un significado de expresar jerarquías de clases.

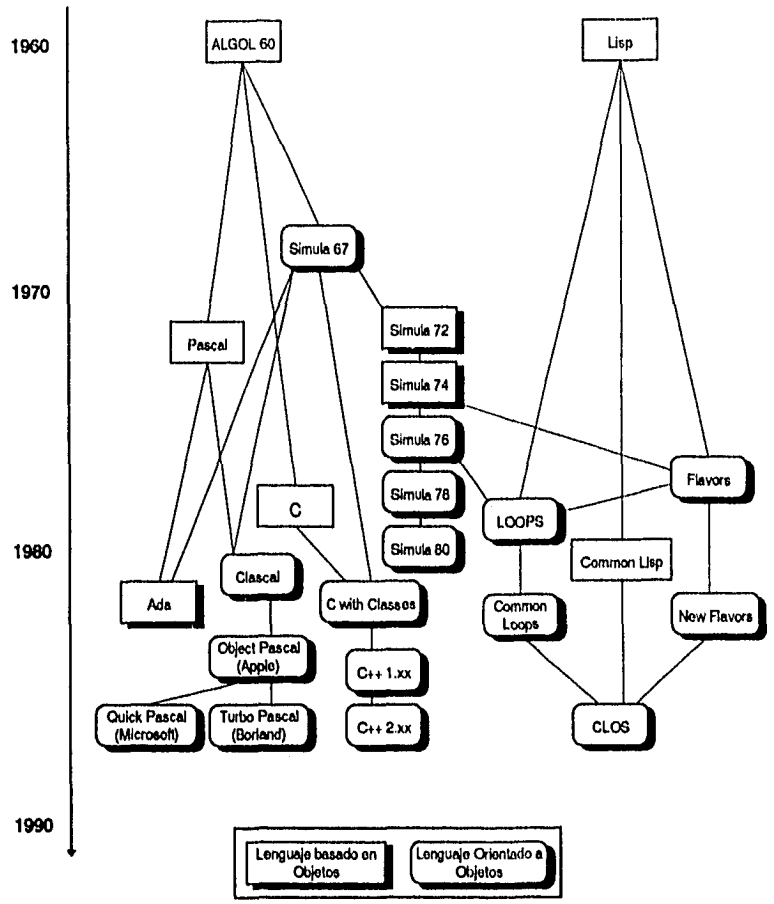

Figura 2.1 Una genealogía de lenguajes basados en objetos y orientados a objetos. Según Booch (1991).

Objetos y Bases de Datos.

Limitaciones de las bases de datos relacionales.

El modelo relacional de administración o manejo de bases de datos fue diseñado y desarrollado en los años setentas; en ese tiempo casi toda la información que se requería almacenar y administrar era textual. Las necesidades inmediatas cubiertas por las bases de datos fueron primeramente el manejo de largas listas de datos similares. El modelo relacional, basado en la representación matemática de datos, es un enfoque eficiente para el manejo de este tipo de información. Sin embargo, la información se ha incrementado de gran manera tanto en variedad como en complejidad. Las expresiones actuales de la información incluyen: dibujos, imágenes, gráficas, hojas de cálculo, sonidos o cualquier combinación de objetos que puedan representarse en las computadoras. Los actuales sistemas de manejo de bases de datos relacionales (SMBDR) no pueden manejar y almacenar esos objetos con sus réplicas textuales. Esta limitación de los SMBDR se magnificará en la medida que las formas no textuales de la información se están convirtiendo en lo común.

Las limitaciones de los SMBDR han creado ya una dicotomía artificial. Para propósitos de almacenamiento los usuarios generalmente no distinguen entre imágenes, dibujos y otras formas de información. Sin embargo los SMBDR deben representar estos objetos de manera diferente a través del software, o aun en diferentes sistemas de computación. Una base de datos que puede representar el precio de lista de cierto artículo pero no una ilustración de éste, refleja las limitaciones de la tecnología actual del software más que las diferencias entre las expresiones de la información. Para los usuarios las limitaciones de las estrategias de las bases de datos actuales son más que teóricas y pueden ser medidas a través de los costos de productividad. Documentos complejos que incorporan varias formas de información requieren que los usuarios manipulen

aplicaciones individuales o, más primitivamente, que "corten y peguen" un copia burda. La dicotomía es aún más pronunciada si los archivos deben almacenarce en diferentes computadoras. En este caso, el tener juntos diversos formatos de información no es tan sencillo como lo era antes del uso amplio de las computadoras de escritorio. Este problema es exacerbado más aun con proyectos con múltiples participantes. Muchos proyectos requieren la participación de profesionales de áreas con funciones muy diversas. Estos profesionales usan computadoras, aplicaciones y hasta sistemas operativos dispares. Cada contribución es distinta - datos financieros, texto, gráficas, diagramas, formas estandarizadas - y toda esa diversidad inherente está compuesta por el uso de diferentes formatos de software.

Evolución hacia las bases de datos orientadas a objetos.

Estas limitaciones del modelo relacional en conjunción con el auge de las herramientas de software basadas en la tecnología de orientación a objetos, han propiciado la evolución hacia los sistemas manejadores de bases de datos orientados a objetos (SMBDOO), las cuales prometen ventajas por sobre los SMBDR. Esta evolución esta apoyada en investigaciones que han partido de dos diferentes filosofías: bases de datos relacionales evolutivas y bases de datos orientadas a objetos con una visión revolucionaria. [10]

Las primeras aplicaciones a gran escala del software orientado a objetos fueron sistemas CAD, los cuales manipulan dibujos complejos como etiquetas de partes, enormes bibliotecas de componentes y otros tipos de datos no tradicionales. Al principio los vendedores del software para CAD usaron sistemas de archivos propios para el manejo de los datos, los cuales eran rápidos pero incompatibles con otras estructuras de software. Por lo que este enfoque fue satisfactorio hasta que los usuarios finales se vieron en la necesidad de tener una interfaz de CAD con otras aplicaciones.[11] Esta situación obligó a

los desarrolladores de software CAD a integrar sus diagramas, planos y otros productos con información meramente textual. Algunos vendedores diseñaron complejos esquemas de traducción, otros usaron tecnologías de sistemas de archivos de SMBDR comerciales dejando de lado sus sistemas propietarios. Otros se tornaron de lleno a las tecnologías orientadas a objetos para manejar todos los aspectos del manejo de sus datos.

Los sistemas de software CAD que utilizan una estrategia de base de datos orientadas a objetos, ofrecen mayor eficiencia en el manejo de su información y capacidades superiores de traducción de los datos, que los productos semejantes que usan otras técnicas de administración de la información.[12] Los SMBDOO manejan eficientemente la variedad de tipos de datos que se pueden encontrar en el entorno de trabajo típico de los sistemas CAD.

Los mismos problemas encontrados por los usuarios de software CAD están siendo encontrados ahora por los usuarios de las aplicaciones de negocios. Literatura de mercadotecnia, documentos de capacitación, presentaciones, manuales de usuario y de especificaciones técnicas, son todos ellos ejemplos de datos diversos que se usan en los ambientes de negocios de la actualidad. Debido a que es frecuente que estos usuarios trabajan con datos provenientes de plataformas de hardware y software diversas, enfrentan problemas más complejos que los usuarios de CAD. Los usuarios de aplicaciones de negocios almacenan y manejan datos de una diversidad y cantidad que se incrementa constantemente. En la propuesta de un proyecto, por ejemplo, se puede tener una parte con información técnica, la estrategia administrativa, presupuestos, una línea de tiempo de la duración del proyecto, dibujos técnicos y una gran variedad de otros tipos de información. Estos componentes rara vez son creados y almacenados en la misma aplicación o en el mismo tipo de computadora. Cuando es necesario integrar estas piezas en un todo, aún las más altamente automatizadas oficinas descansan en técnicas manuales que se han venido usando desde los setentas.

Los SMBDOO prometen la posibilidad de tratar a la información de manera tal que cada componente producido por un usuario pueda almacenarse y manejarse como una parte igual, y cada miembro del grupo de trabajo pueda preparar su contribución en la misma computadora, o en computadoras en red corriendo el mismo administrador de bases de datos. A la hora de la presentación del proyecto, para seguir con el ejemplo, los usuarios simplemente seleccionan el documento final desde una aplicación de la base de datos; la base de datos se encarga de ensamblar los componentes en la forma deseada.

Desafortunadamente no todos los componentes necesarios para hacer realidad esta visión están disponibles aún. Aunque existen varios productos SMBDOO (GemStone, Itasca, O2, Objetivity, ObjectStore, Ontos, OpenOBD, Statice, UniSQL, Versant, entre otros) [13] las aplicaciones que permitan la creación de aquellos componentes y su almacenamiento en una base de datos orientada a objetos común, no existe. Lo más cercano a la realización de un SMBDOO son algunos productos CAD avanzados y muy especializados que manejan dibujos, texto, segmentos individuales de líneas, listas de materiales y otros componentes dentro de la misma base de datos.

Actualmente la industria de los SMBDOO puede considerarse en su segunda generación. La primera generación, con productos de empresas como Ontologics con Vbase, Servio Logic con GemStone y Graphel con G-Base, iniciaron el uso sistemas orientados a objetos para el mercado CAD.[14] La segunda generación representada por empresas como Object Design, Objectivity y Object Sciences se está expandiendo sobre esos esfuerzos tempranos por traer los conceptos de los SMBDOO hacia un mercado más general. Sin embargo los SMBDOO carecen de un modelo formal estandarizado (a diferencia de los SMBDR) por lo que cada fabricante a diseñado su propia metodología, lo que ha provocado una resistencia a su uso en el usuario final.[15] A pesar de ello todos los fabricantes parten de los mismos conceptos. Los SMBDOO, como sus contrapartes relacionales, deben soportar poderosas herramientas para el rápido desarrollo de

aplicaciones. Esto significa soportar al menos un lenguaje de programación de alto nivel, generadores de reportes y herramientas CASE.

Virtualmente todo SMBDOO soporta C++ con extensiones de SQL, haciéndolos similares a los sistemas de desarrollo tradicionales de los SMBDR. Muchas aplicaciones de SMBDR están escritas en C con SQL insertado y son teóricamente independientes de la plataforma. Aunque muchos SMBDOO usan SQL como un lenguaje de consulta, las construcciones SQL deben cambiarse para acomodar el paradigma de objetos. Un problema para la migración hacia los SMBDOO es que SQL es un ineficiente, y en mucho casos, inapropiada manera de accesar datos en el ambiente de objetos. Esto a llevado a que algunos fabricantes como ObjectStore usen lenguajes de neceso totalmente nuevos y diferentes. Estos lenguajes propietarios se conocen como DML (Data Manipulation Language).[16] Este tipo de lenguajes, sin embargo, son incompatibles con los lenguajes de consulta existentes, haciendo que el usuario difícilmente los acepte.

En respuesta a la demanda de los usuarios los fabricantes tradicionales de SMBDR han adoptado la estrategia de ofrecer extensiones a sus productos existentes para incorporar cierta funcionalidad orientada a objetos. Tal es el caso de INGRES Corp. con su INGRES Object Management Intelligent, IBM con sus extensiones a DB2, Hewlett-Packard con IRIS y hasta Oracle y Sybase están trabajando en productos que ofrezcan funcionalidad orientada a objetos.[17]

Para finalizar cabe señalar que los SMBDR sobrevivirán por mucho tiempo por la sencilla razón de que aún muchas cosas las hacen mejor que los SMBDOO. Para aplicaciones que requieren manipular datos completamente textuales no hay nada mejor que el formato de tablas de los SMBDR. Además la tecnología relacional está altamente desarrollada. Los millones de paquetes que existen de dBase, Paradox y FoxPro no van a desaparecer pronto. Los SMBDOO son buenos candidatos para facilitar el manejo y almacenamiento de entidades complejas, heterogéneas que cambian dinámicamente.[18] Sin

embargo los SMBDR coexistirán con los SMBDOO, en un mercado en el que la naturaleza, complejidad y variedad de la información está en constante aumento.

Objetos e Interfaces Gráficas de Usuario.

Ventajas y funcionalidad de las Interfaces gráficas de usuario.

La comercialización de las primeras interfaces gráficas de usuario GUI (por Graphical User Interface) se inicio a principios de los ochentas. Las GUI orientadas a objetos se desarrollaron en los años setentas por XEROX como componentes del Smalltalk. Aunque XEROX ya había iniciado la comercialización de sistemas caros con GUI de escritorio a fines de los setentas, la popularidad de las GUI (orientadas a objetos o no) coincidió con la introducción de la Macintosh de Apple en 1984. Casi un año después Microsoft lanzó al mercado su primera versión de Windows, describiéndola como un entorno o interfaz gráfica de usuario (aunque no orientado a objetos).

Una GUI está asociada a un conjunto común de características disponibles en distintas plataformas y en gran variedad de productos. Estas características comunes pueden resumirse [19] de la siguiente forma:

- Dispositivos secundarios de entrada de usuario, normalmente un dispositivo señalador, generalmente un ratón.
- Funcionalidad del tipo "apuntar y disparar" con menús en pantalla que aparecen o desaparecen bajo control del dispositivo señalador.
- Ventanas que presentan gráficamente lo que la computadora esta haciendo.

- Iconos que representan archivos, directorios, aplicaciones y otras entidades del sistema.

- Cuadros de diálogo, botones, barras de desplazamiento, casillas de verificación y muchas otras metáforas gráficas que permiten al programador y al usuario indicar a la computadora qué hacer y cómo hacerlo.

En la actualidad los GUI han ampliado la funcionalidad básica para soportar no solamente gráficos sino también dimensión, color, luz, video, e interacción altamente dinámica, que permiten simulaciones tridimensionales muy realistas. En cierta medida las GUI están basadas en objetos. Para el usuario, la pantalla contiene una serie de objetos visuales como ventanas, botones, barras de desplazamiento e iconos con los cuales puede interactuar a través del teclado o en la mayoría de los casos con el ratón. Desde la perspectiva del programador la mayoría de las interfaces gráficas no son orientadas a objetos. La relativa proximidad de las GUI con el modelo de objetos es una medida de la utilidad de la interfaz. Con el tiempo, muchas de estas construcciones de objetos simulados en la pantalla serán también implementados internamente.

Entre las principales ventajas de las GUI orientadas a objetos se tiene que su curva de aprendizaje es mucho menor que un entorno de lenguaje de comandos y casi no requiere memorización. Para el usuario final lo interesante radica en la experiencia visual e interactiva, los objetos en la pantalla pueden manipularse para controlar el comportamiento de la computadora; dicho de otro modo la experiencia de *lo que ve y lo que siente* es muy parecida en las computadoras disponibles actualmente. Las GUI orientadas a objetos hacen que las aplicaciones se parezcan mucho más a las funciones del mundo real y cada vez menos a los procesos de programación. Esto permite que los usuarios se centren en las tareas en vez de distraerse con las herramientas; es decir, el

usuario final emplea menos tiempo entrando y saliendo de los programas y más tiempo realizando una tarea inmediata.

La interfaz Macintosh

Reflejo de las investigaciones de las interfaces gráficas orientadas a objetos de XEROX la GUI de Macintosh es para muchos el principal motivo por el cual Apple se ha convertido en una de las compañías de computación más grandes del mundo. El éxito de las Macintosh es aún más sorprendente a la luz de su incompatibilidad con el estándar de la PC's de IBM y sus clones. Sin duda, las GUI orientadas a objetos son una de las mejoras técnicas más importantes en la historia de la computación. Mientras que la interfaz de las Macintosh ha sido el modelo a seguir en los noventas, en 1984 era un concepto radicalmente nuevo. La mayoría de sus construcciones son realmente de objetos, inicialmente usando Object Pascal el cual fue durante mucho tiempo desarrollado por Apple especialmente para la Macintosh (MAC). En la actualidad las últimas versiones de su sistema operativo ha sido en gran medida reescrito en C con extensiones de objetos.

La interfaz de la MAC es única por la relación entre el sistema operativo y la computadora misma: mucho del código de bajo nivel está localizado en hardware ROM. Estos componentes (software y hardware) están tan íntimamente integrados que el sistema operativo y la interfaz comparten el mismo nombre. Todo acceso al sistema es a través de su GUI no hay necesidad de teclear comandos.

La interfaz de la MAC mantiene dos tipos primarios de objetos de interfaz: ventanas e iconos. Varios tipos diferentes de ventanas descienden del objeto genérico ventana. La ventana es usada para "ver" todas las aplicaciones y archivos. Así mismo, las ventanas son usadas para introducir datos al sistema a través de cajas de dialogo. Estas diferentes ventanas tienen comportamientos similares, y funcionan diferente solo en aquellas circunstancias cuando el usuario solicita operaciones únicas para aquel tipo de ventana.

El icono es también un objeto muy flexible. Los iconos representan aplicaciones, archivos de datos y folders de archivos. El folder de archivos es análogo a la estructura de directorio en un sistema operativo más convencional. Abrir un folder de archivos (haciendo doble click sobre su icono) despliega los archivos al siguiente nivel más bajo en la jerarquía de directorios. Los archivos de datos son generalmente representados por iconos de documentos, mientras que las aplicaciones usualmente proveen iconos únicos. Aunque representan diferentes construcciones, los iconos pueden manipularse de la misma manera. Este es un ejemplo de la jerarquía de la clase objeto, la cual concentra funcionalidad similar en la mayoría de las clases generales posibles. Por lo tanto un usuario puede interactuar con todos los iconos en una forma similar y los iconos se comportaran de acuerdo a sus definiciones individuales.

La rígida adherencia de Apple a la uniformidad de las aplicaciones a evitado que la compañía modifique significativamente la apariencia de la interfaz de la MAC. Aunque cada versión nueva del sistema operativo agrega nueva funcionalidad, la apariencia básica ha permanecido intacta.

Windows 3.x de Microsoft.

La interfaz Windows 3.x de Microsoft Corp. está basada en tecnología originalmente desarrollada para el Presentation Manager (PM) de OS/2 de IBM. [20] Las primeras versiones de Windows, liberadas en 1985, corrían sobre el sistema operativo MS-DOS a partir de su versión 2.0 en computadoras personales IBM y compatibles, tenían algunas características de orientación a objetos. Las evidentes deficiencias técnicas contribuyeron a una pobre aceptación en el mercado. Sin embargo con Windows 3.0 se presentó a los usuarios un entorno de computación desde el punto de vista de objetos y se eliminaron muchas de aquellas limitaciones técnicas que habían plagado las primeras

versiones. Como resultado, Windows 3.0 y 3.1 han sido dos de los productos de software mejor vendidos.

Windows 3.x es una GUI que se centra alrededor de la ventana del Administrador de Programas (Program Manager) cuyos bordes contienen todas las ventanas del "escritorio" (desktop). Las aplicaciones son desplegadas en ventanas separadas o grupos dentro del Administrador de Programas. Los usuarios pueden crear grupos y distribuir las aplicaciones entre ellos, presumiblemente en un orden lógico que facilita el acceso. Los grupos aparecen como iconos en la parte baja de la ventana del Administrador de Programas, y se convierten en subventanas (ventanas dentro de ventanas) del Administrador de Programas cuando son abiertos.

Windows, al igual que el PM de OS/2, hacen una marcada distinción entre las aplicaciones ejecutables y los archivos de datos. Solo las aplicaciones que han sido montadas bajo Windows aparecen desplegadas. Las aplicaciones que no están montadas y los archivos de datos pueden accesarce a través del Administrador de Archivos (File Manager) o de la misma aplicación. Esta distinción arbitraria entre aplicaciones y archivos no es consistente con el modelo de orientación a objetos y es una desventaja significativa. En muchos casos el usuario debe montar manualmente los iconos de las aplicaciones en la interfaz, aunque la mayoría de las aplicaciones realizan esta tarea automáticamente cuando se instalan. Las convenciones de DOS respecto al nombrado de archivos se siguen aplicando tanto a los archivos de datos como a los ejecutables, aunque el nombre del icono dentro del ambiente gráfico de estos últimos pueda ser menos restringido que el nombre real de la aplicación.

Las aplicaciones activas pueden "minimizarse", o contraerse, a un icono que aparece en la parte baja y atrás de la ventana del Administrador de Programas. Cuando se corre en el hardware apropiado (Windows puede ejecutarse en el modo 386 mejorado (Enhanced)) las aplicaciones minimizadas pueden estar corriendo y procesando datos aún cuando el

usuario este trabajando en otras tareas. Cuando Windows corre en modo estándar, las aplicaciones minimizadas no pueden realizar ningún proceso, sin embargo permanecen en la memoria principal.

Actualmente coexisten en el mercado las versiones 3.1 y la versión 3.11 conocida como Windows para Trabajo en Grupos (WFW) a la cual se le han añadido capacidades para trabajar en conjunto con otras computadoras en ambientes de red de área local (LAN). Sin embargo, Windows 3.x es un GUI que necesita a DOS para correr.

Esta situación ha llevado Microsoft a reescribir totalmente Windows para hacerlo un sistema operativo. El fruto de este trabajo, y la rápida evolución en el desarrollo de hardware, propició que Microsoft desarrollará una GUI que sea también un sistema operativo, para lo cual ha seguido dos vertientes. La primera vio la luz en 1993 con la liberación de Windows NT (New Technology) un sistema operativo de 32 bits con estructuras internas de objetos, diseñado para correr en una variedad de arquitecturas. Windows NT actualmente ya en su versión 3.5 combina la familiar interfaz del Windows 3.1 pero con la robustez, confiabilidad y seguridad que deben proveer un sistema operativo. El NT es tanto un sistema operativo para redes como para estaciones de trabajo, que tienen incorporadas muchas funciones modulares, se incluyen protocolos TCP/IP y herramientas para conectarse a redes Unix y Macintosh. Los nombres de los archivos no tienen por qué estar limitado a ocho más tres caracteres. Sin embargo, Windows NT tiene cuando menos dos desventajas que han evitado su éxito comercial a gran escala: requiere de una computadora poderosa (y por tanto costosa) para funcionar adecuadamente y no hay muchas aplicaciones "nativas" que aprovechen el acceso a toda la memoria del sistema bajo este sistema operativo.

En la otra vertiente Microsoft recién ha liberado (Agosto de 1995) su largamente esperada nueva versión de Windows. Prometida desde 1993 e inicialmente bautizada como versión Chicago, después versión 4.0 y finalmente liberada con el nombre de

Windows 95. Con este producto Microsoft sustituye al MS-DOS, Windows 3.1 y Windows para trabajo en Grupo 3.11 con lo que supuestamente el usuario podrá ocuparse más de su trabajo en la pantalla y menos de las interrupciones por fallas en el software, debidas en gran medida al anacrónico binomio DOS-Windows.

Windows 95 es un sistema operativo de 32 bits, con capacidades de multitareas y por fin se ve liberado de limitarse a usar únicamente 640 Kb de memoria RAM para correr aplicaciones. Ofrece una nueva interfaz de trabajo en donde los elementos del escritorio ofrecen mayor versatilidad; se supone que es capaz de reconfigurar dinámicamente los componentes (tanto hardware y software) sin necesidad de realizar complicados procesos de instalación. Cuenta con soporte de arquitectura "Plug & Play" (conectar y usar) y con servicios integrados de red y comunicación que permiten compartir recursos fácilmente; esta diseñado para trabajar en conjunto con Windows NT. Cuenta con un nuevo sistema de archivos que ofrece mayor velocidad y permite nombres de archivos más grandes. Se supone que tanto Windows NT como Windows 95 están comprometidos con el modelo de orientación a objetos que les permiten sofisticadas capacidades para manejar todos los componentes del sistema.

X Window, Motif y Open Look.

X Window fue desarrollado en el Instituto Tecnológico de Massachusetts (MIT) en cooperación con la Digital Equipment Corporation (DEC) a finales de la década de los ochentas. [21] X Window es el estándar para los GUI del sistema operativo Unix, pero no es orientado a objetos y no usa objetos de pantalla. Sin embargo, provee la base para mejores productos gráficos como Motif y Open Look. Estas interfaces, aunque no son orientadas a objetos, presentan ante el usuario final varios objetos de pantalla. Ambos proveen sofisticados conjuntos de ventanas, iconos y menús que hacen al subyacente sistema operativo Unix accesible a muchos usuarios.

Open Desktop creado por AT&T y Sun Microsystems, fue la primer adición de objetos a X Window disponible en el mercado. Actualmente esta disponible en las estaciones de trabajo de Sun y AT&T, aunque no ha sido adoptado por el resto de la industria tan rápido como se esperaba.

Motif es un GUI desarrollado por la Open Software Foundation (OSF) y basado en la tecnología de interfaz de usuario de HP, Microsoft y DEC. Derivado de la GUI NewWave de HP y acentuado por su apariencia tridimensional , el realzado de los bordes de las ventanas y menús, permite a los usuarios de Motif reconocer claramente cuando un botón es oprimido o cuando una ventana está seleccionada. Motif también forma la base para el GUI Open Desktop que la Santa Cruz Operation (SCO) implementa en las PC's con el sistema operativo Unix System V de la AT&T. Open Desktop agrega a Motif un manejador de archivos, un emulador de DOS, el SMBDR de INGRES y el X.desktop que es una utilería orientada a objetos para la personalización de la interfaz de usuario. Open Desktop es la versión gráfica de Unix más popular en el mundo de las PC's y forma parte integral de la estrategia de la SCO para promover el cambio de los sistemas multiusuario con interfaces textuales, hacía el mundo de la computación cliente/servidor.

NeXTStep.

Presumiblemente, la mejor interfaz de usuario orientada a objetos disponible es NeXTStep , desarrollada para la implementación de Unix en las computadoras NeXT. La interfaz NeXTStep demuestra que los entornos gráficos pueden ser tan comprehensivos y expresivos como los de líneas de comandos. Mientras que el sistema operativo subyacente definitivamente no es orientado a objetos, virtualmente todo en la pantalla de NeXTStep [22] es un objeto para los usuarios y los programadores.

Aunque el comando Unix está disponible en un icono en la interfaz, NeXTStep actualmente en la versión 3.2 virtualmente no da motivo para usarlo. Casi toda la funcionalidad disponible bajo Unix está también disponible gráficamente desde la interfaz del usuario. Los usuarios pueden realizar tareas complejas y administrar el sistema en una computadora NeXT sin teclear nunca un comando de Unix.

NeXTStep se basa en un explorador o visualizador (Browser): un administrador de archivos que despliega simultáneamente tres niveles del sistema de archivos. Los usuarios pueden seleccionar y correr aplicaciones desde el visualizador o arrastrarlas hacia el Puerto (Dock): una columna de iconos localizada en el lado derecho de la pantalla. Desde el Puerto los usuarios pueden ejecutar las aplicaciones haciendo doble click con el ratón sobre el icono correspondiente. Las aplicaciones se ejecutan sin importar su posición en el sistema de archivos. En el fondo del Puerto esta el equivalente en NeXTStep del bote de basura de la Macintosh: un símbolo de reciclaje. Usado en la misma forma que el bote de basura, el símbolo de reciclaje tiene un nombre más apto ya que el espacio en el disco sea reclamado por el sistema después que son borrados los archivos innecesarios.

En el Puerto también se encuentra el icono de Administración del Sistema (SA por System Administration) el cual da a los usuarios acceso orientado a objetos a una amplia variedad de herramientas Unix para el manejo del sistema operativo. A través del SA, los usuarios pueden ajustar el sonido y la pantalla, respaldar discos duros, configurar una red TCP/IP, modificar los privilegios de acceso al sistema y realizar cualquier otra función administrativa necesaria sobre una base regular.

Como ya lo habíamos señalado arriba el desarrollo de software de NeXTStep, el cual incluye al lenguaje de programación Objective C y el Constructor de Interfaces (IB por Interface Builder), es verdaderamente orientado a objetos. Aunque Objective C es una herramienta para el programador, hace posible para virtualmente cualquier usuario

construir las bases de una aplicación usando el IB. Es posible construir en su totalidad aplicaciones sencillas usando solo el IB.

El IB viene con una paleta de objetos de pantalla como menús, botones y barras de desplazamiento. Para crear pantallas los usuarios seleccionan y dan tamaño a las ventanas de la aplicación, luego colocan aquellos objetos en la ventana que correspondan a la apariencia deseada. Los usuarios pueden agregar cierta funcionalidad básica ligando los objetos de la pantalla de manera que la acción sobre uno de ellos produzca un resultado en otro. Por ejemplo, presionar un botón con el ratón pueda producir un sonido.

Agregar funcionalidad requerirá de programadores y de Objective C. Sin embargo, la disponibilidad inmediata de objetos de pantalla predefinidos y las capacidades de orientación a objetos del entorno de Objective C permiten reducir drásticamente el tiempo de desarrollo de la aplicación. Las ventas de NeXT no han alcanzado las expectativas de su lanzamiento y Steve Jobs [23] ha optado por vender a Cannon, el inversionista de NeXT, la parte de hardware de la compañía, manteniendo el control del sistema operativo y de la interfaz de usuario. En sus esfuerzos por incrementar las ventas, la compañía ha portado el sistema operativo hacia la plataforma Intel, actualmente NeXTStep esta disponible para las PC con procesador 486 en adelante, además en los equipos HP y SPARC. Los analistas de la industria de cómputo están de acuerdo en que el éxito inicial de NeXT giró sobre su novedosa interfaz de usuario y en las herramientas de software orientado a objetos puestas a disposición de los desarrolladores de software comercial.

Herramientas de desarrollo orientadas a objetos.

En esta sección revisaremos algunas herramientas de programación conocidas como entornos propietarios de desarrollo. Un entorno o ambiente de programación orientado a objetos puede definirse como aquel que soporta la creación y ejecución de programas en un lenguaje orientado a objetos, generalmente contiene editores, compiladores (o interpretes) y depuradores; opcionalmente soportan persistencia de objetos, manejo de distintas versiones de objetos, herramientas CASE, herramientas para el manejo de configuración de software y herramientas para documentación. Estas permiten a los programadores desarrollar aplicaciones robustas, con interfaces gráficas de usuario.

EASEL.

Easel es una herramienta de desarrollo de Easel Corp. un socio de IBM, que corre bajo DOS, OS/2 y Windows, tiene un lenguaje orientado a objetos y una biblioteca de clases que contiene gran variedad de objetos de pantalla compilables según los estándares SAA (System Aplication Architecture) y CUA (Common User Acces). Easel es una excelente colección de herramientas que permite crear aplicaciones con interfaces gráficas de usuario a través de herramientas de dibujo. Tales herramientas incluyen un editor de objetos, una biblioteca de clases, un modulo gráfico de dibujo y por separado un lenguaje de desarrollo propietario para agregar funcionalidad a los objetos.

Choreographer.

Choreographer de GUIdance Technologies, es uno de los entornos propietarios de desarrollo orientado a objetos mas completos y robustos. Incluye muchas herramientas interactivas como un Editor de Modulo de Desarrollo, un Editor de Interfaces y un Editor de Bitmap que agilizan el proceso de desarrollo. Sin embargo, la abundancia de herramientas - que incluyen un nuevo lenguaje de programación (una cruza entre C y

Pascal) - aunque útiles agregan complejidad al entorno. Después de un cierto período de aprendizaje y dentro del editor apropiado, los objetos de pantalla pueden crearse al estilo de *apuntar y pulsar* y colocarlos en la pantalla como aparecerán en la aplicación.

Choreographer corre bajo OS/2 y Windows y se trabaja en la versión para correr en la plataforma Unix/X Window. Incluye una extensa biblioteca de clases y es altamente portable entre las plataformas en que corre. Aunque presumiblemente es el más sofisticado entorno de desarrollo orientado a objetos, su precio relativamente alto y su lenguaje de desarrollo propio han restringido su popularización.

VZ Programmer.

A diferencia de Easel y de Choreographer que cuentan con sus propios lenguajes, VZ Programmer de VZ Corporation es una herramienta de desarrollo cuyo lenguaje de desarrollo es un subconjunto de C++. También cuenta con una biblioteca de clases y una paleta de objetos de pantalla, que permite al usuario seleccionar y colocar objetos en una ventana y luego agregarles funcionalidad a través de código C++.

VZ Programmer corre bajo OS/2 y Windows. Su interfaz de objetos y el código subyacente son altamente compatibles con ambas plataformas. La habilidad que provee para fácilmente dibujar la interfaz y agregar funcionalidad usando código C++ hacen a VZ Programmer muy útil para crear prototipos rápidos de aplicaciones para el usuario final para que estos hagan comentarios y retroalimenten el diseño.

Power Builder.

Creado por Powersoft, Power Builder es una herramienta de programación para crear interfaces de usuario para aplicaciones de bases de datos. Power Builder provee una extensa biblioteca de objetos pantalla, además de un lenguaje que permite la manipulación

de tales objetos. Mas aún, Power Builder provee "enlaces" para las bases de datos más populares. Esto le permite ser usado como una herramienta de desarrollo para productos con bases de datos extensas. Así los usuarios obtienen la ventaja de una interfaz de usuario orientada a objetos y la habilidad de accesar bases de datos existentes.

Ambientes personales de productividad.

El mercado de los ambientes personales de productividad.

Entre la amplia variedad de productos de software que existe en la actualidad sobresale un grupo que llamaremos el de los ambientes o entornos personales de productividad. Estos se distinguen por servir a dos propósitos. El primero es permitir a los usuarios más avanzados personalizar su escritorio virtual (computing desktop) con pequeñas utilerías gráficas. El segundo es proveer una GUI estándar que sirva de plantilla o machote para los desarrolladores.

Antes de la introducción del Hypercard de Apple en 1987 algunos usuarios de computadoras reconocieron la necesidad de ambientes personales de desarrollo para aplicaciones sencillas. Hypercard, incluido gratuitamente con la compra de una computadora de la línea Macintosh, creó de inmediato un mercado muy definido para tal producto. Estas herramientas de productividad no podrían existir sin el modelo de orientación a objetos. Aunque estos productos técnicamente no sean orientados a objetos, ni cumplan estrictamente las definiciones del paradigma, son fieles al espíritu del modelo de orientación a objetos. En otras palabras y de igual manera que las interfaces gráficas de usuario estos productos no han sido implementados usando técnicas de orientación a objetos, pero presentan objetos de pantalla fáciles de manipular y que el usuario puede acomodar para crear aplicaciones sencillas. Esta abstracción de objetos del mundo real es una concepción suficientemente simple para permitir que los usuarios que no son

programadores profesionales generen aplicaciones útiles. Aquí revisaremos algunos de los productos más populares.

Hypercard.

Como se mencionó arriba Hypercard se regalaba (en 1987) con la compra de una computadora Macintosh. Se le reconoce como el primer entorno personal de productividad viable. Basado en una metáfora simple: la ficha o tarjeta (index card) Hypercard toma ventaja de la interfaz de la MAC dejando a los usuarios explorar y crear aplicaciones. Texto, gráficas y sonidos son almacenados en tarjetas que aparecen en la pantalla y pueden contener también botones, campos para texto y fondos de diversos tamaños y formas. Los usuarios acomodan y ordenan las cartas y arreglan los objetos dentro de cada carta. Finalmente se agrega funcionalidad a los objetos usando el código HyperScript incluido en el paquete. Estas cartas o fichas son agrupadas en "pilas" (stacks) que son las aplicaciones de Hypercard.

Usar Hypercard es muy fácil e intuitivo. Al correrlo Hypercard pone al usuario dentro de una ficha base (home card), la cual despliega iconos para todas las pilas disponibles. El usuario navega a través del sistema simplemente pulsando sobre botones que seleccionan nuevas pilas o cartas dentro de su pila actual. Sin embargo, también tiene sus deficiencias y limitaciones entre las que destacan: no se puede ver la estructura de fichas completa de una pila, haciendo difícil para los usuarios determinar su posición dentro de la aplicación; la entrada y salida de datos es muy restringida al igual que sus capacidades para manejar información. Dejando de lado sus limitaciones no cabe duda que este producto abrió el espacio para la comercialización de otros productos semejantes para plataformas diversas.

Toolbook.

Siguiendo la tendencia iniciada por Apple con el Hypercard, Asymetrix Corporation creó el primer ambiente personal de productividad para Windows 3.x: Toolbook. Las aplicaciones de Toolbook se llaman "libros" (books). Estos libros, a semejanza de las pilas de tarjetas de Hypercard, están formados de "paginas" (pages). En Toolbook hay dos niveles de usuario: autor y lector. Los libros son usados en el nivel de lector y desarrollados en el nivel de autor. Además de crear nuevos libros al nivel de autor, los usuarios tienen la capacidad de agregar gráficas sencillas e importar aplicaciones más complejas al interior del libro. Cuenta con cinco objetos predefinidos además de los libros y las páginas: fondo, grupo, campo, botón y gráfico. El libro es la aplicación misma. Un grupo de páginas, el nivel al cual se interactua con otra aplicaciones, comprenden un libro. Cada pagina tiene un fondo, el cual puede variar en estilo. Los objetos asociados con cada página son colocados en un fondo.

Al igual que Hypercard, Toolbook no permite la creación de nuevos objetos basado en los objetos predefinidos. Sin Embargo, Toolbook ofrece una variedad de objetos para seleccionar. Es posible usar los objetos de Toolbook para crear sofisticadas interfaces de usuario en Windows. También contiene su propio lenguaje script llamado OpenScript, cuyas caracterfsticas son comparables con el HyperScript de Hypercard, con el añadido de más y mejores instrucciones para el manejo de datos. Toolbook corre bajo Windows 3.x sobre una computadora basada en procesador 386 y con 8 Mb para que se ejecute adecuadamente. Ha sido usado para desarrollar una gran variedad de aplicaciones.

Visual Basic.

En 1991 Microsoft entró al mercado de los ambientes personales de productividad con su Visual Basic. Como Toolbook, Visual Basic corre sobre Windows 3.x y provee de una biblioteca de objetos gráficos para crear una interfaz gráfica de usuario en Windows.

Las poderosas características de Visual Basic lo han convertido en un estándar para programadores aficionados y profesionales: basta echarle una mirada a las ofertas de empleo para profesionales en informática, lo mismo se solicitan programadores en C, COBOL que en Visual Basic.

Visual Basic tiene mucho de *visual* y poco de BASIC. Aunque su lenguaje de programación se parece al tradicional BASIC, debe considerarse como un lenguaje de programación aparte. La organización del código en Visual Basic se basa, como en los ambientes anteriores, en un modelo dirigido por eventos, mientras que los programas hechos en el BASIC estándar son dirigidos por procedimientos.

Visual Basic provee una paleta de objetos de pantalla, tales como botones, texto, cajas de listas, menúes, barras de desplazamiento y otros. Cuenta también con una paleta de propiedades con la cual se puede agregar gran funcionalidad a los objetos. Como en los productos anteriores a los objetos de pantalla de Visual Basic se les puede asociar funciones más complejas con segmentos específicos de código.

El éxito de Visual Basic ha llevado a Microsoft a extenderlo con una versión que corre bajo DOS, y a desarrollar productos semejantes mucho más poderosos y robustos como Visual C++, Access y su complemento: Access Developers Toolkit.

En el mercado pueden encontrarse una cantidad considerable de productos que caben dentro de la categoría de ambientes personales de productividad. La mayoría pretenden ser, y así lo afirman en sus campañas publicitarias, herramientas orientadas a objetos. Esta pretensión es en la mayoría de los casos inapropiada porque estas herramientas carecen cuando menos de alguna de las características importantes del modelo de orientación a objetos. Sin embargo, desde un punto de vista práctico, permiten la manipulación sencilla de objetos de pantalla y, a través del lenguaje que cada uno de ellos provee, establecer interrelaciones entre tales objetos. Así un gran número de

aplicaciones sencillas han podido realizarse por usuarios que no son programadores especializados.

¿Qué hace a un producto Orientado a Objetos?

Como puede comprobarse en cualquier revista especializada en el área, mucho lenguajes de programación y productos de software para el usuario final, reclaman ser o estar basados en técnicas de orientación a objetos. La mercadotecnia obliga a las empresas a publicitar sus productos como totalmente compatibles con las tecnologías de vanguardia. Es común ver productos sin referencia tecnológica previa, reclamar compatibilidad con el modelo de orientación a objetos, sin haber experimentado modificaciones o actualizaciones reales.

Muchos puristas insisten en que únicamente aquellos productos que han sido diseñados y desarrollados de una manera estrictamente orientada a objetos, usando todas y cada una de las características asociadas con la tecnología, puede ser considerados orientados a objetos.[24] Otros, especialmente los vendedores asociados con productos existentes, son mucho menos exactos en sus definiciones. Sin embargo unos cuanto lineamientos sirven para determinar si un producto es, y hasta que punto, orientado a objetos:

¿Puede el producto soportar la creación de nuevos tipos de objetos?

Muchos productos vienen con varios tipos de objetos predefinidos. Si estos objetos predefinidos pueden modificarse o combinarse para crear nuevos tipos de objetos con nuevos comportamientos, el producto cumple una característica fundamental del enfoque de orientación a objetos.

¿Soporta el producto la herencia de características a lo largo de una jerarquía de objetos?

La herencia es una característica crítica de un ambiente orientado a objetos porque le permite aplicar las definiciones y procedimientos de los objetos más generales a los objetos más

específicos en los niveles bajos de la jerarquía. Esto significa un menor esfuerzo de desarrollo y el comportamiento consistente de los objetos de la aplicación.

¿Puede la estructura del objeto encapsular la definición del objeto y el procedimiento que describe los comportamientos permitidos del objeto?

Por ejemplo, la instrucción STRUCT de C permite la definición de una estructura objeto. Sin embargo, esta estructura no puede contener código que defina cómo el objeto deberá ser manipulado. La inhabilidad de crear objetos reutilizables es una falla mayor desde el punto de vista de la ingeniería de software. A menos que se trate de un lenguaje de programación el usuario no podrá ser capaz de detectar esta limitación.

¿El producto se presenta ante el usuario con objetos gráficos de pantalla que son abstracciones de objetos del mundo real?

Aunque el modelo de programación subyacente pueda no ser orientado a objetos, las interfaces gráficas orientadas a objetos han demostrado ser muy útiles para los usuarios novatos y para la productividad de toda clase de usuarios. Un estudio dirigido por el Gartner Group indica que los usuarios se muestran más dispuestos a explorar las características más avanzadas y difíciles de usar de una aplicación, si la interfaz es orientada a objetos.

Estos lineamientos pueden usarse para evaluar la orientación a objetos de productos comerciales. Sin tomar en cuenta estos criterios, el verdadero valor de un producto radica en la utilidad que éste tiene para el usuario. Por ejemplo, Toolbook es un excelente producto para desarrollar pequeñas, pero efectivas aplicaciones personales de uso intenso. Su incapacidad de soportar herencia o de crear nuevos tipos de objetos no disminuyen la utilidad del producto.

En nuestra opinión es importante ignorar la pretendida "orientación a objetos" de la publicidad y centrar la atención en la arquitectura de las estructuras y las posibilidades que ofrezca para el usuario final. Aunque no todos los productos de software orientado a objetos implementan todas estas características, ni en el mismo grado, evaluar un producto

contra tales criterios puede dar una indicación relativa de su virtudes y desventajas contra otros productos.

---

[1] Alan Kay es considerado el visionario detras de la revolución de la orientación a objetos. Recien graduado de la Universidad de UTHA a finales de los sesentas, concibio la idea de la Dynabook, una computadora de alto rendimiento y del tamaño de un cuaderno, con monitor de alta resolución, dispositivos de entrada/salida que soportaran comunicaciones de audio y video y conexiones de red para compartir recursos de información. Su meta era desarrollar la Dynabook como la computadora universal, que cualquier persona sin importar su profesión o entrenamiento en computadoras pudiera llevar a cualquier parte y utilizar como una extensión de sus mentes. Sus ideas influenciaron grandemente el desarrollo de los sistemas Apple, Macintosh; además de la importancia de su trabajo para XEROX en PARC.

[2] Varhol, P.,*Op Cit*, p. 62

[3] Varhol, P.,*Op Cit*, p. 74

[4] Meyer, Bertrand., "Object-Oriented Software Contruction", Prentice Hall, 1988

[5] Cox, Brad, J., "Object-Oriented Programming: An Evolutionary Approach", Addison-Wesley, 1986

[6] Varhol, P.,*Op Cit*, p. 64

[7] Varhol, P.,*Op Cit*, p. 71

[8] Winblad, A. *et al*, "Software Orientado a Objetos", Addison-Wesley/Diaz de Santos, 1993, pp .99-101

[9] Booch, Grady., "Objet Oriented Design with aplications", Benjamin/Cummings, 1991, p 474

[10] Calderón, M. M.,"Una visión general de los Sistemas Manejadores de Bases de Datos Orientadas a Objetos", Soluciones Avanzadas, No. 22, Junio de 1995, pp.37-43

[11] Winblad, A. et al, *Op Cit*, pp .110-141

[12] Varhol, P.,*Op Cit*, p. 84

[13] Calderón, M. M., *Op. Cit.*

[14] Varhol, P.,*Op Cit*, pp. 86-88

[15] Calderón, M. M., *Op. Cit.*

[16] Winblad, A. et al, *Op Cit*. p.120

[17] Vizcaíno, S. C.,"Sybase: Arquitectura para el cambio" Entrevista con Salvador Barra, Director General de Sybase de México, en Soluciones Avanzadas, No. 22, junio de 1995, p. 23. Vease también la sección dedicada a Bases de Datos en ComputerWorld , Marzo 20-24, 1995, p. F-22

[18] Calderón, M. M., *Op. Cit.*

[19] Winblad, A. et al, *Op Cit.* p.144

[20] Winblad, A. et al, *Op Cit.* p.p. 152-154

[21] Winblad, A. et al, *Op Cit.* p.p. 159-165

[22] Martínez René, Victor. ,"NeXStep: una visión a Unix",en RED No. 60, septiembre de 1995, p.p. 44-48.

[23] Steve Jobs junto con Steve Wozniack son considerados los pioneros en el desarrollo del concepto de la computadora personal. Ambos fundaron la compañía Apple, entre 1975 y 1977; Apple fué la primera empresa dedicada a la producción en serie de computadoras personales. En cinco años la empresa pasó de ser un pequeño taller casero a una de las 500 empresas más grándes del mundo. Véase: Nuncio Limón, Reynaldo., "Historia y Perspectivas de la Programación", Ed. Trillas, México, 1991, pp. 149-152

[24] Vease: Greiff, W.,"Pardigma vs. Metodología. El caso de la Programación Orientada a Objetos (I)", Soluciones Avanzadas, No. 6, noviembre-diciembre, 1993, pp.10-16

CAPÍTULO 4

PERSPECTIVAS DEL ENFOQUE DE ORIENTACIÓN A OBJETOS.

LA ESTANDARIZACIÓN DE LOS CONCEPTOS.

Desde 1990 las tecnologías de software orientado a objetos han recibido gran atención por parte de los publicistas, lo cual es un reflejo del surgimiento del interés que éstas han provocado en amplios sectores de la industria del software. Desde los inicios de la comercialización a gran escala de sistemas de computo (hardware y software) los distintos fabricantes han buscado establecer alianzas y determinar estándares con sus competidores, el objetivo de estas alianzas es lograr que sus productos puedan usarse en diferentes equipos y bajo ambientes diversos.

Iniciativas y alianzas entre los fabricantes de software.

La primera organización creada para la estandarización de las conceptos de orientación a objetos fue el Grupo de Administración de Objetos OMG (Object Management Group), asociación internacional formada en 1989 y compuesta por HP, Sun Microsystems, DEC y otros líderes de la industria. [1] Actualmente cuenta con más de 170 fabricantes de hardware y software. El propósito de la OMG es establecer un amplio rango de estándares comunes para permitir que el marco estructural del software orientado a objetos esté disponible para gran variedad de fabricantes. Uno de los estándares ya adoptado por el OMG es la Arquitectura del Operador o Intermediario de Solicitud de Objeto Común: CORBA (Common Object Request Broker Architecture) la

cual permite la comunicación de objetos provenientes de diferentes bases de datos.[2] El componente central del CORBA es el Operador o Intermediario de Solicitud de Objetos ORB (Object Request Broker) el cual garantiza que todos los objetos, independientemente de su base de datos original, se comuniquen en forma estándar a través de una o varias redes.

La estrategia para sistemas de software de Microsoft esta envuelta en su iniciativa de Vinculación e Incrustación de Objetos OLE (Object Linking and Emmbeding). OLE en su versión 1.0 fue incorporado a partir de la versión 3.1 de Windows y esta diseñado para llevar a cabo la vinculación e interacción entre aplicaciones en una sola computadora. Actualmente en su versión 2 incluida en Windows 95 y Windows NT, es un estándar de facto, esto es que si bien no ha sido aceptado oficialmente por el OMG cuenta con el respaldo de líderes en la industria como DEC. OLE 2.0 es compatible con el ORB lo que le da flexibilidad para trabajar en ambientes de red. Todo desarrollador que desee trabajar en aplicaciones que corran sobre los sistemas operativos de Microsoft (excluyendo a DOS por supuesto) deben usar este estándar y la API Win32. Otro estándar de facto que ha implantado Microsoft es el de las especificaciones de Conectividad Abierta con Bases de Datos ODBC (Open Database Connectivity).

IBM ha sido uno de los fabricantes que más comprometidos se han mostrado con las tecnologías de orientación a objetos. En alianza con Borland ha desarrollado sus bibliotecas de objetos para su sistema operativo de 32 bits OS/2.0 usando el C++ de Borland. Además, ha desarrollado las bibliotecas de clases orientadas a objetos conocidas colectivamente como el Modelo de Sistema Objeto o SOM (System Object Model) y el DSOM (Distributed SOM) que permite compartir objetos entre sistemas que trabajan en red. Entre las compañías que desarrollan herramientas de apoyo al SOM se encuentran: Borland, Symantec, Metaware, Easel, ParcPlace Systems, Intelligent Enviroments y Digital.[3] El mismo IBM ha desarrollado productos como: VisualAge (herramienta de

programación visual basada en Smalltalk), SOMobjets Developer Toolkit y C Set ++. Con el uso de las tecnologías orientadas a objetos IBM está logrando que las aplicaciones existentes para mainframes de muchos de sus clientes, puedan funcionar en ambiente cliente/servidor a través de plataformas múltiples. En asociación con Apple, IBM está trabajando en el desarrollo del sistema operativo Taligent, que dará soporte completo al modelo de orientación a objetos.

Además del OMG y de las alianza de IBM con Apple existen otras iniciativas dentro de la arena de la orientación a objetos. Desde 1993 IBM, Sun y HP anunciaron sus planes para competir contra las iniciativas de Microsoft. [4] Estos esfuerzos han visto sus frutos sobretodo en el ámbito de las interfaces comunes orientadas a objetos entre distintas plataformas de hardware: el ya mencionado SOM de IBM, la Instalación Distribuida para el Manejo de Objetos DOMF (Distributed Object Management Facility) de HP y los Objetos Distribuidos por todas partes DOE (Distributed Objects Everywhere) de Sun Microsystems. Otros fabricantes como AT&T, Digital y Apple están ansiosos por que se les considere comprometidos con la tecnología de orientación a objetos,[5] pero no hay que perder de vista que la mayoría de la atención se enfoca en la tecnología de objetos como un paradigma para el usuario final, mas que como una herramienta CASE poderosa para el desarrollo de aplicaciones robustas y flexibles.

Existe también la tendencia por adoptar estándares de objetos para la creación y manejo de documentos. La Arquitectura de Documentos de Oficina ODA (Office Document Architecture) es un estándar que se ha establecido para mejorar la transferencia de documentos entre aplicaciones y plataformas. La ODA está basada en el concepto de ciclo de vida del documento (document lifecycle). El ciclo de vida del documento es una ruta circular y repetitiva dentro de la cual los documentos son creados, almacenados, accesados, distribuidos y modificados. Este concepto recibía poco soporte fuera de los círculos de la orientación a objetos. Otro estándar para documentos es el de

Especificación de Estilos Semánticos y de Lenguaje de Documentos DSSSL (Document Style Semantics and Specification Language) de la ISO (International Standars Organitation). Mientras que la ODA define completamente la especificación del formato de un documento, el DSSSL se concentra únicamente en aquellas características que son necesarias para asegurar las consistencias del formato entre plataformas. En este aspecto es el estándar más flexible. Un tercer estándar a considerar por la industria de la tecnología de objetos es el Standard Generalizado de Marcas del Lenguaje SGML (Standard Generalized Mark-up Language). El SGML contiene los lineamientos para representar texto y puede usarse para definir la estructura de un documento. Este es complementario del DSSSL ya que define la estructura lógica del documento en tanto que DSSSL describe el formato del documento.

Sin embargo, aun existen muchos obstáculos para lograr la madurez necesaria en la tecnología de objetos: no se puede "hojear" las bibliotecas lo suficiente porque hay fronteras entre los lenguajes y entre los ambientes, los estándares del OMG han fallado en cuanto a suministrar sistemas objeto distribuidos entre proveedores, y no hay aún alguna visión del desarrollo de los marcos de trabajo necesarios para obtener una alta tasa de retorno de las inversiones. Tampoco hay notaciones Upper CASE que sean graduales y que permitan que los objetos sean la tecnología de "construcción" que se promete. A pesar de estas carencias, muchos informáticos no dudamos en que la tecnología de objetos será la corriente principal en la ingeniería de software de los años venideros.

LENGUAJES Y BASES DE DATOS.

Tendencias en la administración de bases de datos.

Con la aparición de las PC's de escritorio de alto rendimiento y de bajo costo, los fabricantes de sistemas manejadores de bases de datos SMBD de alta capacidad se han visto de alguna forma obligados a cambiar sus sistemas centralizados de software para equipos grandes y costosos (mainframes y minicomputadoras) para que puedan correr sobre ambientes más flexibles y distribuidos como la arquitectura Cliente/Servidor (C/S) [6] sin sacrificio del desempeño y la confiabilidad de sus sistemas. Así la prioridad para los fabricantes líderes de bases de datos es proporcionar herramientas para el desarrollo de aplicaciones C/S. El concepto C/S puede entenderse como una separación o división de esfuerzos de un proceso de datos entre dos componentes: el "cliente" que debe ser un equipo con un procesador propio, una PC por ejemplo, y el "servidor" que contiene el SMBD y que puede ser una máquina grande o una PC con alta capacidad de almacenamiento. Al software que está corriendo en la PC cliente se le llama front-end y generalmente es una GUI como Windows y es la encargada de todos los procesos de interacción con el usuario como la entrada/salida de datos, las ventanas de captura o el despliegue de gráficas. La aplicación en el servidor se le conoce como back-end y es la responsable del tráfico de datos y del acceso a disco, así como otros servicios propios de un SMBD, por norma estas aplicaciones deben soportar multitareas y multiusuarios. En esta arquitectura el peso de la carga de procesamiento recae en el front-end en el cliente de lo cual se derivan ciertas ventajas: disminución del tráfico en la red, independencia del cliente al poder tener un sistema operativo distinto al del servidor, mejor preservación de la integridad de los datos y un ambiente de trabajo mucho más amigable para el usuario final.

Como parte del front-ends podemos distinguir dos tipos: las herramientas de desarrollo de aplicaciones (enfocadas a programadores en ambiente C/S) y programas para elaborar reportes, formas y gráficas (enfocadas para usuarios finales con cierta experiencia).[7] Muchos de los fabricantes dotan a sus Sistemas Manejadores de Bases de Datos Relacionales SMBDR (sus back-ends) con herramientas para el desarrollo de aplicaciones (front-ends) que adoptan de manera variable el modelo de objetos: ORACLE proporciona el Project X un juego de herramientas que corre en PC cuyo lenguaje es BASIC con funcionalidad de objetos, corre bajo Windows, OS/2 y MAC y se planea liberar el Cooperative Development Envlroment 2 para los desarrolladores UNIX, antes que termine 1995 [8]; Informix provee su NewEra para Windows que se erige como un medio de desarrollo cliente - servidor abierto, gráfico y orientado a objetos, el cual incluye un lenguaje de aplicación de base de datos, herramientas de programación visual, soporte de desarrollo orientado a bibliotecas y conectividad con otras bases de datos relacionales; productos semejantes pueden encontrarse por parte de otros proveedores de SMBD como INGRES, Sybase, SQLBase de Gupta, SQLServer de Microsoft, AS/400 de IBM, etc.

Como vimos en el capítulo 3 los sistemas manejadores de bases de datos orientados a objetos SMBDOO están aun en su etapa de desarrollo. Estos sistemas tienen poca pero creciente participación en el mercado del manejo y almacenamiento de grandes cantidades de información. La migración hacia los SMBDOO será lenta, gradual. La tendencia actual (al menos en el ámbito comercial) nos muestra que la adopción de la arquitectura C/S para la implantación de los SMBD, está llevando la metodología de objetos a ser parte integral de las herramientas front-end para el desarrollo de aplicaciones. La figura 4.1 ilustra las ventas mundiales de bases de datos (en sus diversas plataformas) en millones de dólares.

Ventas Mundiales de Bases de Datos Millones de Dolares			
Tipo de Bases de datos	1994	1995*	1996*
UNIX relacional	1,000	1,250	1,800
Mainframe relacional	1,000	1,000	960
Relacional Grupo de Trabajo **	25	80	225
Orientada a Objetos	80	90	115

* Proyectado
** Base de datos multiusuario ejecutandose con el OS/2, NetWare o Windows NT.
FUENTE: COMPUTER WORLD, Año 15, No. 426, Marzo 20-24, 1995. p. F-22

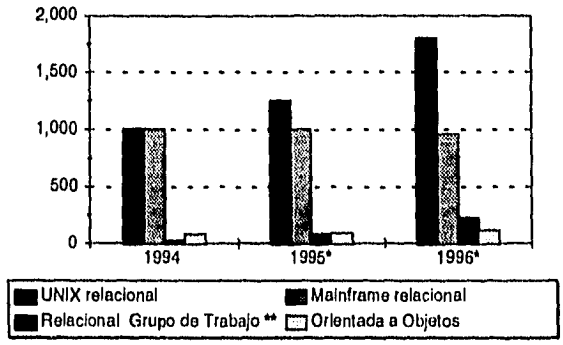

Figura 4.1 Ventas mundiales de bases de datos en millones de dólares.

Programación Orientada a Objetos.

La vasta mayoría del software que se desarrolla en la actualidad está siendo elaborado con técnicas y lenguajes de programación tradicionales, sin el beneficio de los objetos. Aún aquellas organizaciones que están usando C++ o algún producto de Pascal orientado a objetos muy probablemente no están tomando las ventajas de las capacidades de los lenguajes orientados a objetos. Fuera de la comunidad académica y de investigación, son pocos los esfuerzos que se realizan para apoyar a los programadores para que usen los lenguajes orientados a objetos y determinen su factibilidad para

proyectos grandes y complejos. Esta poca disposición a emigrar es alimentada por la existencia de inversiones considerables en tecnologías tradicionales de software, pero sobre todo, por la falta de entendimiento de las técnicas orientadas a objetos. Hará falta una nueva generación de programadores para aceptar completamente el uso de objetos. Es probable que los lenguajes tradicionales con extensiones de objetos dominen el mercado durante los próximos diez años, aunque un mercado significativo para lenguajes no tradicionales y más "puros" en el sentido de orientación a objetos como Smalltalk y Eiffel se desarrollará lentamente. Tal es el caso de las herramientas de desarrollo conocidas como RAD (Rapid Application Development) que están teniendo una amplia aceptación en el mercado de los ambientes PC. Estas herramientas tienen su origen en las herramientas de fabricación de prototipos [9] y en los entornos de programación visual. Prometen dos ventajas importantes: un ciclo de desarrollo más corto y más flexible permitiendo saltar directamente del prototipo a la aplicación terminada; la segunda ventaja consiste en que un usuario final razonablemente sofisticado puede desarrollar aplicaciones. Sin embargo tienen sus limitaciones: requieren de un entendimiento profundo de los requerimientos del negocio o empresa; no eliminan la necesidad de un diseño fuerte y de hábil programación. En gran medida las bondades de las herramientas RAD se derivan del uso de la metodología de objetos.

INTERFACES DE USUARIO.

Interfaces de usuario alternativas.

Las interfaces de usuario forman el área del software orientado a objetos más destacada y de mayor importancia porque en gran medida determina quién usará computadoras y para que propósitos. Probablemente el desarrollo orientado a objetos más significativo en interfaces de usuario es la interfaz basada en pluma. Con varios sistemas

operativos de propósito general los vendedores han estado liberando computadoras portátiles con esta interfaz de usuario (PenPoint de Go Corp., Windows for Pen Computing de Microsoft, Newton de Apple, etc.). Es claro que la tecnología de objetos ha hecho que el enfoque de la computación basada en pluma sea viable comercialmente.

Los sistemas operativos orientados a objetos basados en ventanas o basados en pluma están, obviamente, situados para la orientación a objetos. El sistema PenPoint, por ejemplo, es enteramente orientado a objetos; en tanto que Windows para computación con pluma no está completamente implementado en forma de orientación a objetos debido a los requerimientos de compatibilidad.

Además de funcionar como interfaz de usuario la metáfora de objetos también ayuda a suavizar la transición hacia otras formas de interacción usuario - computadora, incluyendo el trackball, la pantalla sensitiva (touchscreen) y aun las interfaces de voz, que sin duda serán la tendencia en el futuro. Debido a que todos estos dispositivos de entrada alternativos involucran la manipulación de objetos de pantalla, los sistemas operativos y el software que ha integrado objetos ofrecen una mejor interfaz entre el usuario y la computadora.

Sistemas operativos orientados a objetos.

La tecnología de objetos está también en la base para las nuevas generaciones de sistemas operativos. El Windows NT versión 4.0 que se espera este disponible entre 1997 o 1998 será desarrollado para soportar enteramente la arquitectura de objetos [10]. En el OS/2 de IBM todas las funciones del usuario, incluyendo la impresión y el acceso a discos son orientadas a objetos desde su primera versión de 1992. Sun Microsystems con sus sistema Solaris disponible para varias plataformas y derivado del SunOS, es una de las más

exitosas implementaciones de Unix y también fue desarrollado usando enteramente tecnología de orientación a objetos.

En general los principales fabricantes de software están entrando en la carrera de los sistemas operativos orientados a objetos. Esto se explica porque la ventaja clave de estos sistemas operativos radica en su habilidad para proveer un vinculo transparente entre los objetos gráficos en pantalla y las construcciones internas del sistema operativo. Esto simplifica el sistema operativo, el cual crea mejores modelos para la interfaz de usuario. También permite a los desarrolladores agregar extensiones y funcionalidad al sistema operativo, en forma de nuevos objetos.

La nueva generación de sistemas operativos tendrá que dar soporte a las nuevas formas en que la interacción humano-computadora esta constante evolución. Conceptos como computación distribuida, flujo de trabajo, trabajo cooperativo en grupo, computación empresarial, escritorio empresarial (enterprise desktop), colaboración, multimedios, etc. son cada vez más comunes, lo que obliga a los fabricantes de software a ofrecer productos flexibles, confiables, fáciles de portar y que no sean obsoletos en el corto plazo. En estos sistemas los objetos están ya presentes: el ya mencionado NeXTStep, el sistema Pink de Taligent Inc. producto de una colaboración entre IBM, Apple y HP es un sistema operativo totalmente orientado a objetos, la famosa iniciativa Cairo de Microsoft se espera que también sea orientada a objetos.

---

[1] Varhol, P. "Object-Oriented Programming: The Software Development Revolution", Computer Technology Research, Corp. pp. 109-111, 1993.

[2] "Objetos en todas Partes ", COMPUTER WORLD, Año 15, No. 410, Agosto 29, 1994, pp.G25-G26

[3] "Reducción a escala de los Host IBM vía objetos Objetos ", COMPUTER WORLD, Año 15, No. 409, Agosto 22, 1994.

[4] Varhol, P. Op. Cit. pp. 113.

ESTA TESIS NO DEBE
SALIR DE LA BIBLIOTECA

[5] "Sujetando los Objetos", COMPUTER WORLD, Año 15, No. 428, Marzo 20-24, 1995.

[6] Entendiendo Cliente - Servidor como: un paradigma para el proceso de la información en un sistema de computación distribuido, en el cual un objeto actua como el solicitante y otro como el proveedor de ciertos servicios. Bapat, S., " Object-Oriented Networks: models for architecture, operations and management", Prentice Hall, 1994. pp. 698.

[7] Gónzales, Sánchez, J., R., "Entonces, ¿Qué es la tecnología Cliente-Servidor?", PC WORLD, Año I, Número 2, Junio 1995. pp. 95-96.

[8] "Los proveedores de DBMS cumplen con sus promesas", COMPUTER WORLD, Año 15, No. 428, Marzo 20-24, p. F-22, 1995.

[9] Linthicum, S., David, "El fin de la programación", BYTE MÉXICO, Agosto 1995, pp. 66-68.

[10] Halfhill, R., Tom, "Dentro de la mente de Microsoft", BYTE MÉXICO, Agosto 1995, pp. 26-30.

CONCLUSIONES

En este trabajo hemos tratado de dar un panorama general del software orientado a objetos con base, por una parte, en la revisión de los conceptos de las técnicas de orientación a objetos y, por otra, de los principales productos de software disponibles hoy. Hemos tratado se abarcar la mayor parte del vasto espectro del software: técnicas de análisis y diseño, lenguajes de programación, sistemas operativos, interfaces de usuario, ambientes personales de productividad, bases de datos, etc.

Hemos encontrado que la mayoría de los analistas de la industria del software concuerdan en que el software orientado a objetos será la corriente principal del futuro. Por su parte las comunidades académicas especializadas en el tema coinciden en que la orientación a objetos es uno de los avances más importantes para el desarrollo de software. Todos concuerdan en que la programación orientada a objetos mejora no solo el proceso de desarrollo de software sino también la flexibilidad y utilidad del software resultante. Más aun, ***dado el reto de la creciente complejidad del software y los elevados costos de desarrollo, la reutilización de componentes de software es una necesidad imperante. Las bibliotecas de clases y las API's orientadas a objetos han probado ser excelentes plataformas para desarrollar aplicaciones de manera rápida, confiable y económica, en las que reutilizar código es un proceso natural. En otros términos: la reutilización de componentes de software es una necesidad económica en el contexto de las organizaciones modernas.*** No es gratuito que las más preponderantes compañías desarrolladoras de software estén, en mayor o menor grado, comprometiéndose con las técnicas de objetos: establecen alianzas estratégicas para desarrollar productos *ad hoc*, generan iniciativas para establecer estándares que sirvan a sus intereses, promocionan sus productos adjudicándoles "orientación a objetos" aprovechando la ignorancia de los usuarios, y muchas veces sin fundamento alguno.

Otro aspecto relevante que podemos derivar de este estudio es el referente a la necesidad de formar cuadros de programadores capacitados en las diversas metodologías

de análisis y diseño orientado a objetos. Esto requiere un cambio de mentalidad al interior de la comunidad académica y en la forma en que se imparten las materias de computación en las universidades. Si las tendencias en los industriales del software se dirigen claramente hacia la paulatina implantación del enfoque de objetos como estándar para el desarrollo de sistemas en los años venideros, no podemos darnos el lujo de quedar una vez más, rezagados en cuanto a la capacitación y aplicación de estas metodologías. Más aun, existen propuestas muy definidas en cuanto al papel que deben tener los nuevos programadores, que más bien deberían concebirse como integradores de componentes de software ya probados; es decir integradores de objetos.

Otro aspecto que cabe destacar es la innegable influencia que, en los diversos ámbitos de la computación, están teniendo las diferentes interpretaciones del concepto objeto. Desde discusiones filosóficas hasta las más vagas interpretaciones, se habla de orientación a objetos: el paradigma, la metáfora, las técnicas, las metodologías, etc. Sin querer entrar en terrenos que van mas allá de los alcances de este trabajo, podemos establecer que el ***software orientado a objetos es aquel que es elaborado utilizando alguna metodología OO existente y en un lenguaje de programación cuya semántica soporte los conceptos de objetos, clases y herencia.*** Existe sin duda una gran cantidad de aplicaciones, interfaces gráficas de usuario, y hasta ambientes operativos que usan objetos de pantalla, esto es: usan los objetos como una abstracción en la computadora de los objetos del mundo real, pero ello no implica que hallan sido elaborados o que permitan crear software orientado a objetos. Afirmamos que la tendencia actual es precisamente establecer las bases para desarrollar tales aplicaciones siguiendo el enfoque de orientación a objetos.

Finalmente cabe señalar que el enfoque de orientación a objetos para el desarrollo de sistemas de software no es una panacea. Como vimos a lo largo de este trabajo hay software que seguirá aun desarrollándose siguiendo las técnicas tradicionales. Las razones son simples: en primer lugar hay problemas en los que éstas son muy superiores y están

perfectamente estandarizadas y probadas (bases de datos, análisis numérico, interfaces textuales, etc.); en segundo lugar las inversiones en capacitación y en herramientas de desarrollo son considerables. Agregese a lo anterior el hecho de que la resistencia al cambio en las elites de la comunidad de desarrolladores y programadores, que por muy diversas razones existe. Lo que si es un hecho es que los objetos llegaron para quedarse, como alguien escribió por ahí.

BIBLIOGRAFÍA

LIBROS

Apple Computer Inc.,"The Apple Business Solution Directory - Advertising, Communications & Graphics Design", 1991.

Bapat, S.,"Object-Oriented Networks. Models for architecture, operations, and management ", Prentice Hall, 1994.

Booch,G.,"Object Oriented Design with Applications", Benjamin/Cummings, 1991.

Coad, P., Yourdon, E., "Object-Oriented Analysis", Yourdon Press/Prentice-Hall, 1991.

Coad, P., Yourdon, E., "Object-Oriented Design", Yourdon Press/Prentice-Hall, 1991.

Cox, J.B., Novobilski, J.A."Object-Oriented Programming An Evolutionary Approach", Addison-Wesley, 1991.

DeMarco, T, "Structured Analysis and System Specification", Yourdon Press /Pretince Hall, 1979.

Meyer, B., "Object-Oriented Software Construction", Prentice-Hall, 1988.

Microsoft Corporation,"Microsoft Windows 3.1: Manual del Usuario", 1992.

Microsoft Corporation,"Windows Shopping", 1990.

Nuncio Limón, R.,"Historia y Perspectivas de la Programación", Ed. Trillas, México, 1991.

Ravi Sethi, "Programming Languages", Addison-Wesley, 1989.

Rizzo, J., Clark, D.,"How Macs Work", Aiff-Davis Press, 1993.

Rumbaugh,et al., "Object-Oriented Modeling and Design", Prentice-Hall, 1991.

Shlaer,M., "Object-Oriented Systems Analysis", Prentice-Hall, 1991.

Straustrup, B., "The C++ programming lenguage", Addison-Wesley

Taylor, D., "Object-Oriented Information Systems", John Wiley and Sons, 1992.

Taylor, D., "Object-Oriented Technology: A Managers Guide", Servio Corporation/Addison-Wesley, 1990.

Varhol, P.,"Object Oriented Programming: The Software Development Revolution", Computer Technology Research Corp., 1993

Voss, G.,"Object-Oriented Programming: An Introduction",Osborn McGraw-Hill, 1992.

Yourdon, E., "Modern Structured Analysis", Yourdon Press, 1989.

Yourdon, E., & Constantine, L., "Structured Design: Fundamentals of a Disipline of Computer Programming and Design", Prentice Hall, 1979.

Ward, P. T., & Mellor, S.J., "Structured Development of Real-Time Systems", Yourdin Press, 1985.

Winblad,L.A.,Edwards,D.S.,King, R.D.,"Software Orientado a Objetos", Addison-Wesley, 1993.

Wirfs-Brock, B. R., et al,"Designing Object-Oriented Software", Prentice Hall, 1990.

Zdonik,S.,Maier,D., eds.,"Readings in Object Oriented Databases", Morgan Kaufmann Publishers, Inc.1990.

REVISTAS

BYTE

Hayes, F., & Baran, N., "A Guide to GUIs". July 1989. pp. 250-257

Udell, J., "Clash of the Object-Oriented Pascals". July 1989. pp. 104-106

Grehan, R.,"C++ Does Windows". September 1993. pp. 130-136

Linthicum, S., D., "El fin de la programación". Agosto de 1995. pp. 66-68

Halfhill, R., T., "Dentro de la mente de Microsoft". Agosto de 1995. pp. 26-30

COMPUTER

Fichman, G.,R., & Kemerer, F., C., (MIT) "Object-Oriented and Conventional Analysis and Design Methodologies". October 1992. pp. 22-39

Wegner, P., (Brown University) "Interactive Foundations of Object-Based Programming". October 1995. pp. 70-72

COMPUTERWORLD

"Un nuevo enfoque en las bases de datos". Año 14, No. 369 del 17 de mayo de 1993. pp.C16-17

"Reducción a escala de los Host IBM vía Objetos". Año 15, No. 409 del 22 de agosto de 1994.

"Objetos en todas Partes". Año 15, No. 410 del 29 de agosto de 1994. pp. G25-G26

"Cairo ¿Qué tan fácil será la migración". Año 15, No. 410 del 29 de agosto de 1994. pp. E12

"Karat, Nuevo Sistema de Administración Empresarial". Año 15, No. 418, Noviembre de 1994.

"Sujetando los Objetos". Año 15, No. 428. Marzo 20-24 de 1995.

"Los proveedores de DBMS cumplen con sus promesas". Año 15, No. 428. Marzo 20-24 de 1995. p. F-22

"Juicio a la Administración Basada en Objetos". Año 15, No. 429, Marzo 27-316 de 1995. pp. D15

Contreras, M., R., "Complejidad sin llegar al Caos, la Tendencia en PC's". Año 15, No. 420 del 05 de diciembre de 1994. pp. A1

"Enseñando a apreciar la orientación a objetos" partes I y II . Año 16, Números 441 y 442 de Agosto-Septiembre de 1995.

"Smalltalk, primer lenguaje OO para los programadores". En el suplemento Dealer World de septiembre de 1995. p. S-8

"Los Operadores para Solicitud de Objeto", Año 16, No. 446, Octubre 2-6 de 1995. pp. E19-20

PC World

Fox, S., "Windows 4.0. Is this the Windows you've always wanted?". August 1994. pp. 129-135

González, S., I.,R.,"Entonces, ¿Qué es la tecnología Cliente-Servidor?", Año I, No. 22, junio de 1995. pp. 95-96

RED

Martínez, R.V., "NextStep una visión de UNIX". No. 60 de septiembre de 1995. pp. 44-48

Soluciones Avanzadas

Calderón, M. M., "Una visión general de los Sistemas Manejadores de Bases de Datos Orientados a Objetos." No. 22 julio 1995. pp. 37-43

Cervantes. O. "Bases de Datos Orientadas a Objetos: Realidades y Promesas." No. 6 noviembre-diciembre 1993. pp. 28-31

Greiff. R. W. "Paradigma vs. Metodología. El caso de la Programación Orientada a Objetos." No. 6 noviembre-diciembre 1993. pp. 10-16

Greiff. R. W. "Paradigma vs. Metodología. El caso de la Programación Orientada a Objetos. (Parte II)" No. 7 enero-febrero 1994. pp. 31-39

Ibargüengoitia, G. &. López, A. "Análisis Orientado a Objetos mediante la Técnica de Modelado de Objetos OMT. Modelos Dinámico y Funcional." No. 14 diciembre 1994. pp. 25-28

Ibargüengoitia, G., & Olmos, S., A., "El Modelo de objetos Implementado en Bases de Datos Relacionales." No. 25 septiembre 1995. pp. 36-40

Ibarra, E. & Vergara, G., "Evaluación del Sistema Operativo NextStep en el Grupo Financiero InverMéxico." No. 18, febrero 1995. pp. 05-11

Oktaba, H., "Análisis y Diseño Orientado a Objetos." No. 4 julio-agosto 1993. pp. 08-11

Oktaba, H., "Análisis y Diseño Orientado a Objetos. Introducción al Método de Booch"? No. 6 noviembre-diciembre 1993. pp. 18-22

Oktaba, H., "Programación Orientada a Objetos. ¿Moda o Realidad?" No. 3 abril-mayo 1993. pp. 39-42

Quintanilla, G. , "El Impacto en la Ingeniería de Software de la programación Orientada a Objetos." No. 3 abril-mayo 1993. pp. 43-46

Quintanilla, G. , "Elementos de programación Orientada a Objetos." No. 4 julio-agosto 1993. pp. 05-07

Vizcaíno, S., C., "La verdadera Importancia de los Objetos". No. 20. Abril de 1995. Editorial

Índice de Materias

A

B

C

D

E

H

I

L

M

N

O

W

X