



Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

**Implementación del marco de trabajo Cocoa
para el desarrollo de aplicaciones en Mac OS X**

PAUL MICHAEL SOBERANIS LETONA
ASESORADO POR INGA. ELIZABETH DOMÍNGUEZ

Guatemala, enero de 2005

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA



FACULTAD DE INGENIERÍA

**IMPLEMENTACIÓN DEL MARCO DE TRABAJO COCOA
PARA EL DESARROLLO DE APLICACIONES EN MAC OS X**

TRABAJO DE GRADUACIÓN

PRESENTADO A JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

PAUL MICHAEL SOBERANIS LETONA

ASESORADO POR INGA. ELIZABETH DOMÍNGUEZ

AL CONFERÍRSELE EL TÍTULO DE
INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA ENERO DE 2005

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA



NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Sydney Alexander Samuels Milson
VOCAL I	Ing. Murphy Olympto Paiz Recinos
VOCAL II	Lic. Amahán Sánchez Álvarez
VOCAL III	Ing. Julio David Galicia Celada
VOCAL IV	Br. Kenneth Issur Estrada Ruiz
VOCAL V	Br. Elisa Yazminda Vides Leiva
SECRETARIO	Ing. Pedro Antonio Aguilar Polanco

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Sydney Alexander Samuels Milson
EXAMINADOR	Ing. Claudia Liceth Rojas Morales
EXAMINADOR	Ing. Manuel Fernando López Fernández
EXAMINADOR	Ing. Jorge Luis Álvarez Mejía
SECRETARIO	Ing. Pedro Antonio Aguilar Polanco

HONORABLE TRIBUNAL EXAMINADOR

Cumpliendo con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

IMPLEMENTACIÓN DEL MARCO DE TRABAJO COCOA PARA EL DESARROLLO DE APLICACIONES EN MAC OS X

Tema que me fuera asignado por la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería con fecha 10 de enero de 2004.

Paul Soberanis Letona



Guatemala, septiembre de 2004.

Ing. Carlos Alfredo Azurdía Morales
Coordinador Comisión de Trabajos de Graduación
Escuela de Ciencias y Sistemas
Facultad de Ingeniería USAC

Estimado Ingeniero:

Por medio de la presente hago de su conocimiento, que he procedido a revisar el trabajo final de graduación titulado: **Implementación del marco de trabajo Cocoa para el desarrollo de aplicaciones en Mac OS X**, elaborado por el estudiante Paul Soberanis Letona, y de acuerdo a mi criterio, se encuentra concluido y cumple con los objetivos propuestos para su desarrollo.

Agradeciendo de antemano la atención que le preste a la presente, me suscribo a usted,

Atentamente.

Inga. Elizabeth Domínguez
Asesor



Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Carrera de Ciencias y Sistemas

Guatemala septiembre de 2004

Ingeniero
Luis Alberto Vettorazzi España
Director de la Escuela de Ingeniería
En Ciencias y Sistemas

Respetable Ingeniero Vettorazzi:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **PAUL SOBERANIS LETONA**, titulado: “**IMPLEMENTACIÓN DEL MARCO DE TRABAJO COCOA PARA EL DESARROLLO DE APLICACIONES EN MAC OS X**”, y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

Ing. Carlos Alfredo Azurdia
Coordinador de Privados
Y Revisión de Trabajos de Graduación

El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de San Carlos de Guatemala, luego de conocer el dictamen del asesor con el visto bueno del revisor, del trabajo de graduación titulado “**IMPLEMENTACIÓN DEL MARCO DE TRABAJO COCOA PARA EL DESARROLLO DE APLICACIONES EN MAC OS X**”, presentado por el estudiante **Paul Soberanis Letona**, aprueba el presente trabajo y solicita la autorización del mismo.

ID Y ENSEÑAD A TODOS

Ing. Luis Alberto Vettorazzi España

DIRECTOR

INGENIERIA EN CIENCIAS Y SISTEMAS



El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas, al trabajo de graduación titulado **“IMPLEMENTACIÓN DEL MARCO DE TRABAJO COCOA PARA EL DESARROLLO DE APLICACIONES EN MAC OS X”** presentado por el estudiante universitario Paul Soberanis Letona, procede a la autorización para la impresión del mismo.

IMPRÍMASE.

Ing. Sydney Alexander Samuels Milson
DECANO

Guatemala, enero de 2005

AGRADECIMIENTOS

A:

Mis padres José Soberanis y Nora Letona, mi padrastro Anibal Martínez y mis hermanos Alejandro, Guery y Jorge por el apoyo incondicional que me dieron a lo largo de la carrera.

Mis amigos Rubén, Eric, José, Kenneth, Nydia, Oscar, Evelyn y demás compañeros que me apoyaron en este proceso.

Mi asesor Inga. Elizabeth Domínguez y a los catedráticos de la carrera por su apoyo brindado en la ejecución del presente trabajo.

La persona más especial para mi y por la que tengo un gran sentimiento.

Y a todas aquellas personas que de una u otra forma, colaboraron o participaron en la realización de esta investigación, hago extensivo mi más sincero reconocimiento. Gracias totales.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES.....	IV
GLOSARIO.....	VI
RESUMEN.....	X
OBJETIVOS	XII
INTRODUCCIÓN.....	XIII

1. MARCO TEÓRICO	1
1.1 ¿Qué es Cocoa?	1
1.2 El sistema operativo Mac	2
1.2.1 AQUA	3
1.2.2 DARWIN o el propio núcleo del OS X	4
1.2.3 MACH.....	5
1.2.4 BSD.....	9
1.3 Sistema de archivos	9
1.3.1 Diferencias entre HFS+ y UFS	11
1.3.2 Enlaces simbólicos y alias	12
1.3.3 Organización del sistema de archivos.....	13
1.3.4 Dominio del sistema de archivos	14
1.4 Arquitectura del sistema	19
1.5 Uso de memoria.....	22
1.5.1 Mac OS clásico	23
1.5.2 Mac OS X	24
2. EL LENGUAJE DE PROGRAMACIÓN OBJECTIVE C.....	27
2.1 Historia del lenguaje.....	26
2.2 Comparaciones	30
2.3 Tipos definidos de datos.....	31

2.4	Características del lenguaje	33
2.4.1	Objetos	33
2.4.2	Paso de mensajes en objetos	34
2.4.3	Clases y herencia	35
2.4.4	Dinamismo	36
2.5	Otras características	37
2.5.1	Categorías	37
2.5.2	Protocolos	37
2.5.3	Comportamientos	38
2.5.4	Reemplazo	38
2.5.5	Tiempo de ejecución (<i>Runtime</i>)	38
2.6	Manejo de memoria	39
2.6.1	Inicialización y liberación de un objeto	40
2.6.2	Contador de referencia	40
2.6.3	Colas de auto-liberación	42
2.6.4	Retención de objetos en el método accesor	43
2.6.5	Reglas importantes	44
3.	JAVA PARA MAC OS X	47
3.1	Características de Java	48
3.2	Otras características	54
3.2.1	Interfaces	54
3.2.2	Multihilos	55
3.2.3	El puente de Java	55
3.2.4	Java puro	56
3.3	Applets	56
3.3.1	Applets con firma	57
3.4	<i>JavaBeans</i>	58
3.5	La Máquina virtual de Java	58
3.6	Recolección de basura generacional	59

3.7 Manejo de error y excepción	61
4. DISEÑO DE APLICACIONES PARA MAC OS X.....	63
4.1 Herramientas de diseño y desarrollo	63
4.1.1 Project builder	63
4.1.2 Interface builder	64
4.2 Planteamiento de la aplicación	65
4.3 Configuración e instalación de MySQL	66
4.3.1 Requerimientos	66
4.3.2 Configuración del servidor	66
4.3.3 Instalación y ejecución del servidor.....	67
4.4 Diseño de la base de datos.....	69
4.5 Desarrollo de la aplicación	70
CONCLUSIONES.....	73
RECOMENDACIONES.....	74
BIBLIOGRAFÍA.....	75
ANEXO	77

ÍNDICE DE ILUSTRACIONES

FIGURAS

1. Una vista funcional del Mac OS X.....	3
2. Arquitectura del sistema en capas en Mac OS X.....	20
3. Objeto utilizado por tres arreglos diferentes.....	41
4. Liberación progresiva de un objeto.....	41
5. Componentes individuales del JRE y SDK construidos en el Mac OS X Nativo.....	48
6. Ambientes en capas de aplicación de Java.....	53
7. Método de recolección de basura por generaciones.....	60
8. Ventana de Project Builder en Jaguar.....	64
9. Elementos usables en Interface Builder para una aplicación.....	65
10. Pantalla del cliente de MySQL.....	70
11. Ventana de conexión a base de datos en CocoaMySQL.....	70
12. Ventana inicial del administrador de contactos.....	78
13. Ventana con la conexión realizada sin errores.....	78
14. Ventana contactos ingresados	79
15. Ventana de ingreso de contactos nuevos.	80
16. Ventana de edición de contactos.	81
17. Menú de la aplicación.	82
18. Ventana de información acerca de la aplicación	82

TABLAS

I. Sistemas de archivo soportados en el sistema operativo Mac OS X	10
II. Comparación entre Objective C y otros lenguajes.	
30	
III. Tipos de datos definidos en Objective-C.	
31	

GLOSARIO

- API** Acrónimo del inglés *Application Program Interface* (Interfaz para programas de aplicación). Conjunto de convenciones de programación que definen cómo se invoca un servicio desde un programa.
- ASCII** (*American Standard Code of Information Exchange*). Estándar aceptado casi mundialmente que recoge 128 caracteres, letras, números y símbolos utilizados en procesadores de textos y algunos programas de comunicaciones. Su principal ventaja es su amplia difusión y aceptación. De hecho, la mayoría de los procesadores de texto presentes en el mercado pueden importar y exportar ficheros a formato ASCII, lo que facilita el intercambio de información entre personas o empresas que no trabajan con la misma aplicación. El más utilizado es el ASCII extendido (de 8 bits) que permite representar 256 caracteres, como la ñ, vocales acentuadas, etc., frente al ASCII de 7 bits que solo permite representar 127 caracteres.
- Base de datos** Se abrevia como BDD. Es una colección de información organizada de tal manera que un programa de computadora pueda obtener, manejar y actualizar datos requeridos de tal información. Se puede pensar en una base de datos como un sistema de archivos electrónicos.
- Bit** Unidad de datos para representar números binarios. El bit puede poseer los valores 0 ó 1.

Byte	Unidad de datos formada por ocho bits, sin importar los valores que puedan tomar la combinación de bits en ese conjunto. En los sistemas de computadoras los bytes se usan para representar caracteres alfanuméricos o números.
Cocoa	Es una API totalmente orientada a objetos a la que podemos acceder desde dos lenguajes de programación Objective-C y Java.
Controlador	Programa necesario para que un cierto programa o sistema operativo sea capaz de utilizar un dispositivo (por ejemplo, una impresora).
CPU	<i>(Central Processor Unit)</i> Unidad Central de Procesamiento. Contiene la unidad aritmético-lógica del procesador.
Depurar	<i>(Debug)</i> . En el trabajo con herramientas de programación, consiste en la revisión de la aplicación generada con el fin de eliminar los posibles errores que puedan existir en ésta. También persigue la optimización del programa para que su funcionalidad y velocidad sean las máximas. Esta operación pueden realizarla programas especializados, lo que facilita el trabajo del programador y acorta el tiempo empleado en la fase de depuración.
Firewire	Es un estándar para la transmisión de periféricos rápidos el cual lo hace ideal para el uso con dispositivos multimedia.
GCC	GCC es el compilador gnu (gratis) de C, C++, Fortran, Java.

- GNU** Licencia Pública General. Software desarrollado para distribución sin fines de lucro. El proyecto GNU (GNU es un acrónimo recursivo para "Gnu No es Unix") comenzó en 1984 para desarrollar un sistema operativo tipo Unix completo, que fuera software libre.
- GUI** (*Graphical User Interface*). Sistema de interacción entre el ordenador y el usuario, caracterizado por la utilización de iconos y elementos gráficos en su concepción. De esta manera, el usuario recibe una interfaz más amigable. La mayor parte de los GUIs proporcionan soporte para el ratón y/o las ventanas para gestionar programas.
- Kernel** Núcleo. Parte fundamental de un programa, por lo general de un sistema operativo, que reside en memoria todo el tiempo y que provee los servicios básicos. Es la parte del sistema operativo que está más cerca de la máquina y puede activar el hardware directamente o unirse a otra capa de software que maneja el hardware.
- POSIX** (*Portable Operating System Interface for X*). Es un conjunto de normas definidas por el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) con el fin de crear interfaces que permitan la portabilidad del software entre diferentes sistemas operativos abiertos.
- SDK** (*Software Development Kit*) Conjunto de librerías y utilidades de compilación para programar en Java (J2SE), junto con entornos de programación gratuitos.

Servidor	Programa de computación que da servicios a otros programas de computación, ya sea en la misma máquina (computadora) o en otra diferente. 2) La computadora que activa un programa servidor es también llamada servidor (aunque pueda alojar en su interior programas servidor y programas cliente).
Smalltalk	Fue el primer sistema puro de objetos. Todo en Smalltalk es un objeto y toda la computación es desarrollada mediante mensajes que son enviados entre los objetos.
USB	Bus serie universal (<i>universal serial bus</i>), un nuevo tipo de conexión serie que se está imponiendo rápidamente por ciertas características como: conexión de varios dispositivos a un mismo puerto (hasta 127), conexión con el ordenador encendido, y reconocimiento automático por parte del ordenador.
Unicode	Superconjunto del conjunto de caracteres ASCII que utiliza dos bytes en lugar de uno para cada carácter. Unicode es capaz de manejar 65,536 combinaciones de caracteres en lugar de 256, y puede contener los alfabetos de la mayor parte de los lenguajes a nivel mundial. ISO define un conjunto de caracteres de cuatro bytes para alfabetos mundiales, pero también utiliza el Unicode como un subconjunto.
Web	Servidor de información www. Se utiliza también para definir el universo WWW en su conjunto.
Web Service	Servicio público en internet el cual genera datos en formato xml.

RESUMEN

La demanda de aplicaciones para el sistema Mac OS X de Apple va en aumento. Dicho sistema integra cinco ambientes de aplicación, de los cuales uno de ellos es Cocoa, una tecnología importante de desarrollo utilizado para la construcción de aplicaciones de desarrollo rápido, eficaz, productivo y altamente eficiente. Cocoa se constituye entonces en una nueva manera, altamente productiva, de crear nuevos programas por parte de los programadores bajo un ambiente de desarrollo muy diferente de Microsoft Windows y comparado con el sistema Linux y Unix.

Cocoa es un marco de trabajo de aplicación orientado a objetos: un conjunto de componentes de software utilizado para el desarrollo e implementación de aplicaciones para ser ejecutadas en el ambiente del sistema operativo Mac OS X. Se puede pensar en Cocoa como un gran *set* de librerías que se pueden reutilizar de acuerdo con necesidades específicas de programación.

Cocoa está fundamentalmente basado en OpenStep, una tecnología orientada a objetos que fue introducida originalmente en 1987 como NextStep y se ha redefinido continuamente en los últimos años. Consecuentemente, Cocoa es una tecnología que se está convirtiendo en un estándar para la programación orientada a objetos en Mac OS X basada en años de diseño de otros marcos de trabajo.

El marco de trabajo Cocoa puede ayudar en el desarrollo de productos de software rápidamente, utilizando el verdadero potencial de las herramientas de trabajo provistas por el sistema Mac OS X y la metodología orientada a objetos para facilitar el trabajo.

Cocoa está implementada por completo en el lenguaje Objective C, un superconjunto de C ANSI con una sintaxis especial que permite el uso completo de las

técnicas de programación orientadas a objetos por los programadores de hoy en día. Las extensiones del lenguaje son compactas y fáciles de aprender lo que facilita y reduce enormemente su uso y la curva de aprendizaje para nuevos programadores.

El segundo lenguaje de implementación que soporta Cocoa es Java, un lenguaje que se ha convertido en los últimos años en un lenguaje “por defecto” en la programación multiplataforma. La funcionalidad libre de ambos lenguajes permite el desarrollo de aplicaciones robustas y complejas. Esto provee una importante ventaja de desarrollo a favor del marco de trabajo Cocoa.

OBJETIVOS

General

Proporcionar una guía práctica para el diseño, desarrollo e implementación de aplicaciones para el ambiente Mac OS X utilizando el marco de trabajo Cocoa.

Específicos

1. Proporcionar a estudiantes y profesionales en el área de sistemas y ciencias de la computación el conocimiento necesario para que apliquen el marco de trabajo Cocoa como herramienta de desarrollo de aplicaciones para el ambiente Mac OS X.
2. Proporcionar al estudiante y profesional en el área de sistemas y ciencias de la computación una opción eficaz, estandarizada y robusta para el desarrollo rápido y de alta productividad de aplicaciones para el ambiente Mac OS X.
3. Listar las características que hacen del marco de trabajo Cocoa una herramienta eficiente para el desarrollo de aplicaciones para el ambiente Mac OS X.

INTRODUCCIÓN

El trabajo propuesto se basa en el estudio y aplicación del marco de trabajo Cocoa como una herramienta que soporta un desarrollo rápido y una alta productividad en aplicaciones en un ambiente Mac OS X. El marco de trabajo Cocoa incluye una librería completa de clases diseñada para crear aplicaciones robustas y poderosas para Mac OS X. El diseño orientado a objetos de Cocoa simplifica el desarrollo y mantenimiento de aplicaciones.

Aunque Mac OS X ofrece soporte a una extensa variedad de lenguajes de programación, los lenguajes más usados para trabajar directamente con el marco de trabajo Cocoa son Objective C y Java.

Cocoa está implementado en Objective C, un lenguaje compatible con C ANSI con extensiones orientadas a objetos dinámicos modelados en Smalltalk. Las aplicaciones desarrolladas en Cocoa pueden hacer uso de la funcionalidad contenida en las librerías de C tradicional y C++. Además provee interfaces en Java para permitir un alto rendimiento.

El primer capítulo inicia con una vista preliminar de los conceptos esenciales del ambiente de programación del sistema operativo Mac OS X para asegurarse de la comprensión general del marco de trabajo, haciendo énfasis en los temas que es necesario conocer para describir la metodología más efectiva para el análisis y diseño de aplicaciones robustas y poderosas desarrolladas por Cocoa con base en el ambiente Mac OS X.

Con respecto a Objective C, el cual amplía al superconjunto de C para convertirlo en un lenguaje orientado a objetos y es un tema reciente que no ha sido

tratado anteriormente, ofrece conceptos más avanzados que C++ y brinda al programador mucha más flexibilidad en la creación de estructuras y objetos que serán presentados en el segundo capítulo del trabajo. Además, desde Java es posible acceder a todas las clases de interfaz de programación del marco de trabajo Cocoa, lo cual permite realizar aplicaciones Java tan rápidas como sus correspondientes aplicaciones Objective C.

Para incrementar el aporte de este trabajo a la población en general se plantea el análisis, diseño y desarrollo de una aplicación de base de datos, la cual identifica la práctica del conocimiento adquirido. Lo referente a la aplicación puede ser consultado en el capítulo 4 y en el apéndice.

1. MARCO TEÓRICO

El sistema operativo Mac OS X de las computadoras Apple es el sistema operativo más revolucionario que ha aparecido en escena durante muchos años. Con el Mac OS X, Apple ha reafirmado su liderazgo y dirección en los sistemas operativos empleando tecnología sofisticada y diseño sensible, que son la base de la compañía.

Mientras se conserva su afamado lema de “facilidad de uso” y personalidad de sus predecesores, el Mac OS X es un sistema operativo moderno, con una fuerza industrial diseñado para mantener la fiabilidad, estabilidad, escalabilidad y desempeño. Como tal, inicia la fundación para otra década de innovación.

1.1 ¿Qué es Cocoa?

Mac OS X integra, como ambiente de desarrollo de aplicaciones, el marco de trabajo Cocoa, una tecnología importante de desarrollo empleada para desarrollar aplicaciones funcionales en el sistema Mac OS X. De esta forma, Cocoa constituye una eficiente alternativa para crear y desarrollar nuevos productos o trabajar antiguos programas en la tecnología Macintosh de Apple.

Los lenguajes utilizados en Cocoa se pueden clasificar por lo general en dos categorías: lenguajes basados en C y lenguajes de tipo *script*.

Objective C y Java son los dos lenguajes de programación más comúnmente usados con Cocoa, debido a su amplio conocimiento extendido por los desarrolladores, así como por su facilidad de aprendizaje. Pero muchos otros lenguajes funcionan con el marco de trabajo también.

Cocoa es, entonces, un marco de trabajo orientado a objetos con un conjunto de componentes de software reusables como librerías usados para la construcción de aplicaciones ejecutables en el sistema Mac OS X.

1.2 El sistema operativo Mac

El sistema operativo Mac OS X es una evolución radical de los sistemas operativos anteriores de Macintosh.

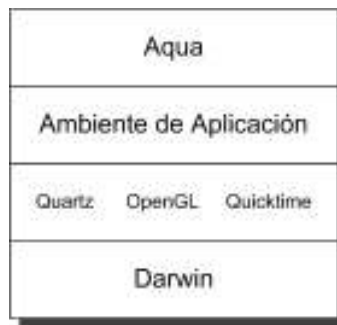
Esta siguiente generación de sistemas operativos es una síntesis de tecnologías; algunas son nuevas y la mayoría forman ya un estándar en la industria de la computación. Está firmemente ajustado sobre una sólida fundación del núcleo del sistema operativo, brindando beneficios tales como memoria protegida y multitarea preventiva a las computadoras Macintosh. El Mac OS X tiene una interfaz de usuario con capacidad de efectos visuales como la translucidez y las sombras de gota. Estos efectos, así como las gráficas finas vistas en una computadora personal, fueron posibles gracias a la tecnología en gráficos que Apple desarrollo específicamente para el sistema operativo Mac OS X.

Pero el sistema operativo Mac OS X es más que un núcleo sofisticado y una interfaz estilizada. Con su ambiente de múltiples aplicaciones, virtualmente todas las aplicaciones de Macintosh pueden ser ejecutadas sobre él. Y con su soporte para varios protocolos de red y servicios, Mac OS X es la plataforma más idónea para usar y disfrutar de internet. Además ofrece un alto grado de interoperabilidad con otros sistemas operativos debido a sus múltiples formatos de volumen y su conformidad con normas y estándares ya establecidos y en desarrollo.

Desde una perspectiva funcional, los componentes más importantes del sistema operativo Mac OS X son:

- Aqua, la interfaz humana diseñada detrás de la experiencia del usuario.
- Los ambientes de aplicación Carbón, Cocoa, Java y Classic.
- El sistema de gráficos y ventanas implementado (con soporte para Quicktime y OpenGL).
- Darwin, el núcleo base de UNIX avanzado para el sistema operativo.

Figura 1. Una vista funcional del MAC OS X



Fuente: <http://developer.apple.com/macosx/systemoverview.pdf>, julio 2003.

El ambiente del Mac OS X es similar al de las versiones previas del Mac OS. La diferencia del diseño de la interfaz del usuario, la infraestructura para localizar dicha interfaz, la estabilidad que se mantiene aun cuando una aplicación no responde, la ejecución concurrente de las aplicaciones sin monopolizar recursos de procesamiento son parte de la experiencia que el Mac OS X ofrece a los usuarios y los rasgos que hacen de esa una experiencia productiva y agradable.

1.2.1 AQUA

La interfaz grafica del sistema operativo Mac OS X da la apariencia de un sistema moderno, fácil de usar y sofisticado a la vez. Las propiedades brindadas por dicha interfaz incluyen profundidad del color, claridad, translucidez y movimiento.

El sistema operativo Mac OS X provee una transición sencilla y fácil para los usuarios y desarrolladores. Mac OS X soporta cuatro ambientes de aplicación, cada uno para un tipo particular de aplicación:

- El ambiente Classic puede ejecutar la mayoría de aplicaciones Mac OS 9. Debido a que el ambiente Classic es un ambiente de compatibilidad, no soporta algunas de las ventajas que ofrece el sistema Mac OS X, tal como Aqua o mejoras arquitectónicas proporcionadas por “Darwin”.
- El ambiente de Carbón ejecuta todas las aplicaciones del Mac OS 9 cuyo código ha sido perfeccionado y optimizado para Mac OS X. Convirtiendo su código para usar los API del Carbón, los desarrolladores de aplicaciones pueden asegurar que las aplicaciones toman ventajas de protección de memoria, multitarea preventiva y otros rasgos de Darwin.
- El ambiente de Cocoa ofrece un avanzado marco de trabajo orientado a objetos para la creación de aplicaciones mejoradas de la siguiente generación.
- El ambiente de Java ejecuta 100% Java puro y aplicaciones de Java y applets API-mezcladas.

1.2.2 DARWIN o el propio núcleo del OS X

El sistema Mac OS X con su interfaz fácil de usar es una “piedra sólida” basada en la ingeniería de UNIX que se diseña para la estabilidad, fiabilidad y desempeño. Esta base es el núcleo del sistema operativo conocido como Darwin que es una tecnología de código abierto.

Darwin es el *kernel* del sistema operativo y sobre el que se centran las más importantes interacciones del software con el hardware. Darwin integra varias tecnologías como por ejemplo el Mach 3.0, originalmente desarrollado en la universidad de Carnegie-Mellon, servicios basados en el sistema operativo 4.4BSD (*Berkeley Software Distribution*), medios de gestión de redes de alto rendimiento y soporte para múltiples sistemas de archivos.

Debido al diseño altamente modular de Darwin es posible agregar dinámicamente controladores de dispositivos, extensiones de red y nuevos sistemas de archivo.

1.2.3 MACH

El trabajo del Mach 3.0 es dotar al procesador y la memoria de la capacidad de abstracción del resto de los componentes del equipo. Se encarga de gestionar los tiempos de trabajo del procesador, así como de facilitar la protección de memoria entre las distintas aplicaciones que se estén ejecutando en un momento dentro del ordenador. Todo ello sin olvidar la gestión de todos los mecanismos de entrada y salida de que se disponga para un proceso en ejecución

Mach es el corazón de Darwin porque desempeña una cantidad considerable de funciones críticas en un sistema operativo. Mucho de lo que Mach provee es la transparencia a las aplicaciones. Mach administra e implementa una infraestructura de mensajería centrada para la comunicación de interprocesos sin tipo, de forma local y remota. Mach trae muchas ventajas importantes a la ingeniería de computación de las Macintosh.

1. Soporte multiprocesador

El *kernel* de Darwin, el Mach 3.0, ahora permite el uso de varios procesadores trabajando en paralelo. Aunque hoy por hoy, el OS X no está capacitado al 100% para el uso de esta funcionalidad, en breve se cree que se podrá implementar del todo en el OS, ya que su *kernel* está preparado para ello.

2. Memoria protegida

La estabilidad de un sistema operativo no debe depender del comportamiento adecuado de las aplicaciones ni de la necesidad de ir escribiendo los datos a cada aplicación en su correspondiente espacio de dirección, lo que puede provocar pérdida o corrupción de información y precipitar la caída del sistema incluso. Mach asegura que las aplicaciones no puedan intervenir en los bloques de memoria de otras aplicaciones o en el bloque de memoria del sistema operativo. Limitando de esta forma las aplicaciones y los procesos del sistema, Mach inhabilita casi virtualmente, a una aplicación con un comportamiento pobre, para que hiera el resto del sistema; y aun si la aplicación afecta parte del sistema, no es necesario reiniciar la computadora.

Ahora el sistema operativo Mac OS X asignará una única dirección de memoria por cada proceso que se esté ejecutando. Las aplicaciones quedan totalmente aisladas del resto de procesos que se estén llevando a cabo en la memoria de la máquina, gracias a lo cual si una aplicación se bloquea, ésta no podrá interferir en el resto de memoria usada por otros procesos, con lo que ésta será la única en quedarse bloqueada, y no nos veremos obligados a reiniciar el ordenador.

3. Multitarea apropiativa

En un sistema operativo moderno, los procesos comparten el CPU de una forma eficaz. Mach va mas allá del procesador de la computadora, priorizando tareas, haciendo que los niveles de actividad estables estén al máximo y asegurándose de que cada tarea consiga la cantidad de recursos necesarios. Mach utiliza cierto criterio para decidir la importancia de una tarea para asignarle la cantidad de tiempo de procesamiento necesario. Los procesos no son dependientes de otros procesos que rinden su tiempo de procesamiento.

Gracias a la multitarea, se podrán tener distintos procesos ejecutándose a un tiempo, y será esta función la encargada de asignar los tiempos de uso de CPU para cada proceso, y ya no van a ser éstos los que liberen la CPU para dar paso a otros procesos. Esto hará que el reparto de tiempo sea mucho más razonable, con lo que el comportamiento general del ordenador será más estable.

4. Avanzada gestión de la memoria virtual

Se ha dotado al sistema operativo de una avanzada gestión de la memoria virtual. Esto permite despreocuparse de asignar memoria a los programas, ya que serán éstos los que irán reservando más o menos memoria en función de los trabajos que realicen y la cantidad de la misma que requieran. Posteriormente, y cuando sus necesidades de memoria bajen, liberarán la parte usada que ya no necesiten.

5. Soporte en tiempo real

Esta característica garantiza el acceso de baja latencia a los recursos del procesador para las aplicaciones sensibles al tiempo. Darwin también habilita la multitarea cooperativa e hilos (*threads*) preventivos y cooperativos.

6. Soporte dispositivo- controlador

Para el desarrollo de controladores (*drivers*) de dispositivos, Darwin ofrece un marco de trabajo orientado a objetos llamado el “Kit I/O”. Este marco de trabajo no sólo facilita la creación de *drivers* para el sistema operativo Mac OS X, sino que además proporciona mucha de la infraestructura que dichos controladores necesitan. Está escrito en un subconjunto restringido de C++.

El marco de trabajo, el cual está diseñado para soportar un rango amplio de familias de dispositivos, es modular y extensible. Los controladores de dispositivos creados empleando el Kit I/O adquieren varias características importantes:

- “*plug and play*” verdadero
- administración dinámica de dispositivos (“*hot plugging*”)
- administración de energía (para escritorios y portátiles)

7. Extensiones de red

Darwin ofrece a los desarrolladores de *kernel* una nueva tecnología para agregar la gestión de redes a las capacidades del sistema operativo: Extensiones de *Kernel* de Red (*Network Kernel Extensions* o NKE). La facilidad del NKE permite crear módulos de gestión de redes e incluso las pilas o *stacks* de protocolo enteros que pueden ser cargados y descargados dinámicamente en el *kernel* internamente. También es posible configurar las pilas protocolares automáticamente.

Los módulos de NKE tienen ciertas capacidades incorporadas para supervisar o monitorizar y modificar el tráfico de la red. En el enlace de datos y capas de la red, es permisible recibir también las notificaciones de los eventos asíncronos de los *drivers* de los dispositivos, como cuando hay un cambio en el estado de una interfaz de la red.

1.2.4 BSD

Integrada con Mach, existe una versión personalizada del sistema operativo BSD (actualmente 4.4BSD). La implementación del BSD en Darwin incluye muchas de las API de POSIX y las exporta a las capas de aplicación del sistema. BSD se emplea como base para el sistema de archivos y utilidades de redes del OS X. Además, provee interfaces de programación abundantes y servicios que incluyen:

- modelación de procesos (identificador de proceso, señales, banderas, etc.)
- políticas básicas de seguridad tales como identificación de usuarios y permisos
- *threading support* (hilos POSIX)
- BSD *sockets*

1.3 Sistema de archivos

El componente del sistema de archivos de Darwin está basado en las extensiones del BSD y en un diseño aumentado de un sistema de archivos virtual (VFS). El sistema de archivos introduce varias características generales.

1. Permisos en medios removibles. Esta característica está basada en un identificador único global registrado en un sistema para cada dispositivo removible conectado (inclusive USB y dispositivos de FireWire).
2. Montaje de volúmenes basado en URL, que habilita a los usuarios vía comando del buscador (semejante al explorador de Windows) montar volúmenes como AppleShare y servidores de Web.
3. *Caché de buffer* unificado, el cual consolida e integra el *caché* del *buffer*, con el *caché* de memoria virtual.

4. Nombres de archivos extensos (255 caracteres o 755 bytes, basado en UTF-8).
5. Soporte para extensiones de nombre de archivos ocultos basados por archivo.

Desde el punto de vista de la arquitectura, el Mac OS X implementa múltiples sistemas de archivos. Desde una perspectiva del usuario, los sistemas de archivos son monolíticos; cuando el usuario copia, mueve o arrastra cualquier archivo o fóldeo pareciera un solo sistema de archivos.

Tabla I. Sistemas de archivo soportados en el sistema operativo Mac OS X

Formato extendido Mac OS X	Denominado también sistema de archivos jerárquico HFS+. Éste es el volumen raíz por omisión del sistema Mac OS X. Esta versión extendida del HFS optimiza la capacidad de almacenamiento de discos duros decrementando el tamaño mínimo de un sector de archivo. Es además el formato estándar del sistema Mac OS 9.
Formato estándar Mac OS X	Éste es el sistema de archivos en los sistemas anteriores al Mac Os 8.1. HFS (así como el HFS+) almacena recursos y datos en distintos sectores de un archivo y hace uso de varios atributos de archivo, incluyendo códigos de tipo y de creador.
UFS	Un formato de disco “plano”, basado en el 4.4BSD FFS (<i>File Fast System</i>) que es muy similar al formato de la mayoría de computadores basados en UNIX; soporta la semántica del sistema de archivos de POSIX, el cual es muy importante para numerosas aplicaciones servidor.
UDF	El formato de disco universal (<i>Universal Disk Format</i>) para volúmenes DVD.
ISO 9660	El formato estándar para volúmenes CD-ROM.

Fuente: <http://developer.apple.com/macosx/systemoverview.pdf>, agosto 2003.

Debido a sus ambientes de aplicación múltiples y a los variados tipos de dispositivos que soporta, el sistema operativo Mac OS X debe ser capaz de ocuparse de los datos de archivo en variados formatos de volúmenes estándar. La tabla 1 muestra los formatos soportados.

Los formatos HFS y HFS+ soportan seudónimos (alias), y el formato UFS soporta enlaces simbólicos (ambos HFS y UFS soportan enlaces duros). Aunque un

alias y un enlace simbólico son referencias ligeras de un archivo o directorio en parte del sistema son semánticamente diferentes en forma significativa.

1.3.1 Diferencias entre HFS+ y UFS

Existen muchas diferencias significativas entre los dos más importantes sistemas de archivos del Mac OS X: HFS+ y UFS. La siguiente lista resume algunas de estas diferencias entre ambos sistemas de archivos:

1. Sensibilidad al contexto: UFS es sensitivo al contexto, en cambio HFS+ no lo es, preserva el contexto.
2. Múltiples procesos hijos: FORK (bifurcar) es la única forma de crear un proceso nuevo. HFS+ soporta múltiples *forks* (metadatos adicionales) mientras que UFS únicamente soporta un *fork* simple.
3. Separadores de ruta: HFS+ utiliza el símbolo ; como separador en donde UFS utiliza el estándar de /. El sistema reconoce y diferencia entre ambos separadores.
4. Modificación de fechas: HFS+ soporta creación y modificación de fechas como metadatos de archivos; UFS soporta modificación pero no la creación de fechas. Si un archivo se copia utilizando un comando que permite modificación pero no creación de fechas, el comando podría *resetear* la fecha de modificación y crear una nueva para la copia. Debido a este comportamiento, es posible tener un archivo con una fecha de creación mayor que su fecha de modificación.
5. Archivos esparcidos y relleno de ceros. UFS permite archivos esparcidos, una alternativa del sistema de archivos de almacenamiento de datos sin almacenar el

espacio vacío en esos archivos. HFS+ no soporta archivos esparcidos y realiza un relleno de ceros en todos los bytes ubicados en el archivo hasta alcanzar un EOF.

1.3.2 Enlaces simbólicos y alias

Los enlaces simbólicos y alias son referencias ligeras hacia carpetas y archivos. Los alias están asociados con el sistema de archivos HFS y el HFS+; los enlaces simbólicos son una característica especial del formato UFS. Ambos tipos de enlaces permiten referencias múltiples hacia carpetas y archivos sin requerir copias múltiples de los objetos.

Originalmente, los alias localizados en un archivo o carpeta usan la llave única primero y su nombre completo después para realizar la búsqueda de un archivo determinado. Si un archivo es movido en el mismo volumen, varios alias del archivo se mantendrán apuntando al archivo original. Si un archivo es eliminado y reemplazado por otro con un nombre de archivo similar, los alias localizarán al archivo por su nombre. En el sistema operativo Mac OS X 10.2, los alias revierten su orden de búsqueda usando el nombre del archivo primero y su llave única después.

Debido a que los alias y enlaces simbólicos utilizan la dirección del archivo para ubicarlos, ambos ofrecen un comportamiento similar. Si un archivo es reemplazado con un nombre de archivo idéntico, moviendo el archivo antiguo hacia una nueva ubicación, los enlaces simbólicos y alias apuntarán hacia el nuevo archivo. Sin embargo, si se mueve un archivo sin reemplazarlo, los enlaces simbólicos del archivo se rompen mientras que los alias no.

En los sistemas de archivos HFS y HFS+, cada archivo y carpeta tiene una llave única y persistente. Los alias almacenan esta llave única además de la dirección del archivo. Si el archivo no puede encontrarse por su nombre, se intenta localizar por su

llave única. Si el archivo es encontrado, se actualiza el registro interno con la información de la nueva ubicación. De forma similar, si la dirección del archivo es correcta, pero la llave primaria es incorrecta, se actualiza el registro interno con la llave única del nuevo archivo.

1.3.3 Organización del sistema de archivos

En Mac OS X cada archivo en el sistema tiene su propia ubicación, una ubicación de directorio estándar para cada tipo de archivo. Para los usuarios, esto no significa que deban colocar sus aplicaciones y recursos en las ubicaciones recomendadas. Las aplicaciones, después de todo, son paquetes autosuficientes sin importar donde sean instalados. Pero si el usuario no coloca ciertos objetos en las ubicaciones esperadas del sistema, se pierden algunas de las ventajas de Mac OS X.

El diseño del sistema de archivos es usualmente representado como una estructura de árbol jerárquico que empieza en la raíz. En la raíz de un sistema de archivos típico de Mac OS X (indicado por una /) se encuentran los siguientes elementos:

/Mac OS X/ - El volumen en el cual el sistema operativo inicia y el software de sistema y recursos son instalados. Este volumen es típicamente un disco duro formateado para ser un Mac OS extendido (HFS+). El nombre "Mac OS X" es el nombre del volumen por omisión, el cual puede ser cambiado.

/Network/ -El directorio raíz para el área de red local montada en el sistema de usuario. Este directorio siempre aparece aunque el usuario no este conectado a una red.

/Otros volúmenes/ - Representa a uno o mas dispositivos conectados internos o externos que no son dispositivos de arranque. Esto incluye dispositivos como *Zip Drives*, *Cd-*

Rom Drives, cámaras digitales así como discos duros y sus respectivas particiones. (El nombre del directorio solo es representativo; el nombre actual de cada volumen conectado será diferente). Todos los volúmenes que no son de arranque del sistema aparecen conforme son montados y desaparecen si estos son desmontados.

La organización física de los volúmenes es en cierta forma diferente a la representación mostrada en el explorador. Es posible observar en una ventana del terminal el volumen de arranque montado en la raíz del sistema (/) y los demás volúmenes localizados en la carpeta /Volumes/. Esto es para proveer una abstracción más tradicional con la interfaz del Mac OS. Además, en el nivel raíz, oculto para los usuarios, se encuentran los directorios estándar de BSD como /usr, /bin, y /etc.

1.3.4 Dominio del sistema de archivos

En un sistema multiusuario, el control del acceso a los recursos del sistema es importante para el mantenimiento de la estabilidad del sistema. Mac OS X define varios dominios de sistemas de archivos, cada cual provee almacenamiento para los recursos en un *set* establecido de directorios. El acceso a los recursos en cada dominio es determinado por los permisos del usuario actual.

Existen cuatro dominios del sistema de archivos, descritos en la siguiente lista:

- **Usuario:** el dominio de usuario contiene recursos específicos del usuario que está conectado en el sistema. Este dominio está definido por el directorio corriente del usuario, el cual puede o no ser el volumen de arranque. El usuario tiene el control completo del contenido de ese dominio.
- **Local:** el dominio local contiene recursos como aplicaciones y documentos compartidos para todos los usuarios del sistema en particular, pero que no son necesarios para ejecutar el sistema. El dominio local no corresponde a un directorio

físico simple, sino que consiste de múltiples directorios en el volumen de arranque (y raíz). Los usuarios con privilegios de administrador pueden agregar, remover y modificar los objetos contenidos en este dominio.

- Red: el dominio de red, al igual que el dominio local, contiene las aplicaciones y documentos compartidos por los usuarios en un entorno de red de área local. Los objetos en este dominio están, por lo general, ubicados en servidores de archivos y bajo el control de un administrador de red.
- Sistema: el dominio de sistema contiene todo el software de sistema instalado. Los recursos en el dominio de sistema son requeridos por él mismo para ejecutarse. Los objetos en este dominio están localizados en el volumen local de arranque y los usuarios no poseen privilegios para agregar, eliminar o modificar los objetos en este dominio.

El dominio para un recurso dado determina su aplicabilidad o accesibilidad para los usuarios en el sistema. Por ejemplo, una fuente instalada en el directorio raíz para un usuario está únicamente disponible para ese usuario en particular. Si el administrador instala la misma fuente en el dominio de red, todos los usuarios de red tendrán acceso a ella.

En cada dominio, Mac OS X provee un *set* de directorios iniciales para la organización de los recursos contenidos. El sistema utiliza nombres de directorios idénticos entre dominios para el almacenamiento de recursos del mismo tipo. Esta consistencia simplifica el proceso de búsqueda de recursos para el usuario y para el sistema. Las búsquedas se inician en el dominio de usuario, seguidas del dominio local, de red y del sistema.

1.3.4.1 El dominio de usuario

El dominio de usuario contiene recursos que son específicos para un solo usuario. El dominio de usuario está representado por el directorio raíz del usuario actualmente conectado. Cada usuario del sistema debe tener una cuenta en la computadora o en el área de red local a la cual la computadora está conectada físicamente. Cada cuenta de usuario involucra un área del sistema de archivos, llamado directorio de usuario. En este directorio residen los programas, recursos y documentos del usuario. El nombre de cada directorio de usuario está designado conforme el nombre de usuario, el cual debe de ser único.

El dominio de usuario crea un ambiente de trabajo específico posible para cada usuario. Cuando un usuario se conecta, el explorador restablece las preferencias y ambiente de trabajo de la sesión anterior de acuerdo con las preferencias de cada dominio de usuario. De forma similar, la información de las aplicaciones y otro software del sistema son restablecidas en el dominio de usuario.

La ubicación de cada directorio de usuario (dominio de usuario) depende de la cuenta de usuario. Si la cuenta de usuario es local respecto a la computadora, el directorio de usuario se encontrará ubicado en el directorio de usuarios (*/users*) del volumen de arranque. Si la cuenta de usuario es una cuenta de acceso de red, el directorio de usuario se ubica en un servidor de red. Independientemente de la ubicación física del directorio de usuario, el sistema Mac OS X utiliza la convención de UNIX de un carácter de tilde (~) en algunas situaciones para indicar el directorio raíz del usuario. El carácter de tilde puede ser usado en combinación con otros directorios o nombres de usuario para especificar directorios de usuario.

Cuando una cuenta de usuario es creada, un directorio de aplicaciones no es creado automáticamente en el directorio raíz. Sin embargo, los usuarios pueden crear un

directorio de aplicaciones y colocar todas sus aplicaciones en él. El sistema automáticamente busca las aplicaciones en esta ubicación.

El sistema protege los archivos y directorios del directorio raíz del usuario por medio de un *set* de permisos por omisión, el cual el usuario puede cambiar en cualquier momento. Cualquier nuevo folder creado heredará los privilegios del directorio padre.

Además, con los directorios individuales del usuario, el directorio de usuario contiene una carpeta de compartidos. Este directorio es accesible para cualquier otro usuario local del sistema. Cualquier usuario tiene privilegios de escritura, lectura y modificación de documentos en este directorio. Esta carpeta no está directamente vinculada con el dominio de usuario pero provee un uso conveniente para el intercambio de documentos y archivos entre usuarios.

1.3.4.2 El dominio local

El dominio local contiene recursos que están disponibles en la computadora a nivel local, pero que no son necesarios para la ejecución del sistema operativo. Los recursos en el dominio local usualmente incluyen aplicaciones, utilidades, fuentes personalizadas, elementos de arranque y preferencias globales de aplicación. Las carpetas de aplicaciones y de biblioteca en el volumen raíz contienen los recursos para el dominio local. Estos recursos están disponibles para el usuario actual del sistema pero no lo están para los usuarios de red. Los administradores de una computadora pueden instalar recursos en el dominio local si desean que los recursos sean compartidos por todos los usuarios del sistema. Otros recursos del sistema como fuentes, perfiles de color, preferencias y plug-ins deben estar en el subdirectorio adecuado de la carpeta de biblioteca.

1.3.4.3 El dominio de red

El dominio de red contiene los recursos disponibles para todos los usuarios registrados en un área local de red. Los usuarios tienen acceso a las aplicaciones, documentos y otros recursos del dominio, incluyendo servidores *web*. La composición del dominio de red depende de las políticas, y su implementación es responsabilidad del administrador de red.

/Network/Applications Contiene las aplicaciones que pueden ser ejecutadas por todos los usuarios en el área local de red.

/Network/Library Contiene recursos como *plug-ins*, archivos de sonido, documentación, marcos de trabajo, colores y fuentes- disponibles para todos los usuarios de la red.

/Network/Servers Contiene los puntos de montaje para servidores de archivos NFS que componen el área local de red.

/Network/Users Contiene las carpetas raíz para todos los usuarios del área local de red. Esta es la ubicación por omisión para las carpetas raíz que pueden ser almacenadas en otros servidores.

1.3.4.4 El dominio de sistema

El dominio de sistema contiene los recursos requeridos por Mac OS X para ejecutarse. Todos los recursos en el dominio de sistema están localizados en el directorio */System* en el volumen raíz. El contenido de estos recursos solamente pueden ser modificados por el usuario raíz. Los usuarios con permisos administrativos y las aplicaciones no pueden instalar los recursos en el dominio de sistema en forma directa.

Por omisión, la carpeta de sistema solamente contiene el directorio de biblioteca. Este subdirectorio contiene una variedad de tipos de recursos como otras carpetas de biblioteca en el sistema. Sin embargo, en el dominio de sistema, este directorio contiene además los servicios del núcleo, marcos de trabajo y aplicaciones propias del Mac OS X.

El ambiente de compatibilidad Classic contiene recursos relacionados con el sistema que no son considerados parte del dominio de sistema.

1.4 Arquitectura del sistema

La necesidad de integrar una diversa colección de tecnologías fue la llave que se consideró para el diseño del sistema operativo Mac OS X. La característica principal de la arquitectura del sistema son las capas o niveles del sistema de software, una capa que posee las dependencias e interfaces de la capa inferior a ella. El sistema Mac OS X tiene cuatro capas distintas en su sistema de software (en orden de dependencia):

Ambiente de aplicación

Incluye los cinco ambientes de aplicación (o ejecución): Carbón, Cocoa, Java, Classic y BSD.

Servicios de aplicación

Incorpora los servicios disponibles del sistema para todos los ambientes de aplicación que tienen algún impacto en la interfaz gráfica del usuario.

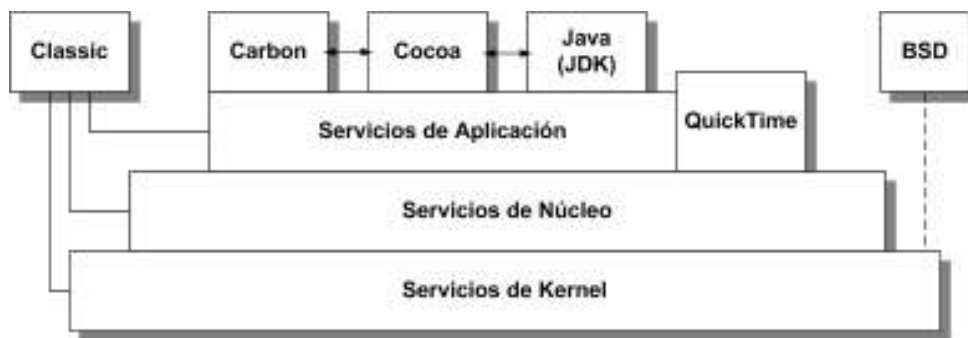
Servicios del núcleo

Involucra aquellos servicios del sistema que no tienen efecto en la interfaz gráfica del usuario.

Ambiente del *kernel*

Provee la capa base del sistema operativo. Sus componentes principales son Mach y BSD, pero también incluye los protocolos de red, sistema de archivos y controladores de dispositivos.

Figura 2. Arquitectura del sistema en capas en Mac OS X



Fuente: <http://developer.apple.com/macosx/systemoverview.pdf>, agosto 2003.

El ambiente Classic representado en la capa superior ofrece cierta compatibilidad a los usuarios para poder ejecutar aplicaciones basadas en Mac OS 8 y Mac OS 9. En lugar de colocarse por encima de todas las capas, el ambiente Classic tiene enlaces directos hacia cada capa. Estos enlaces o conexiones indican que el ambiente Classic no es un ambiente para el cual los desarrolladores puedan específicamente compilar código en el sistema Mac OS X.

El ambiente de comandos BSD ofrece una consola en el cual es posible ejecutar programas BSD desde la línea de comando. Las herramientas, utilidades y *scripts* estándares de BSD están disponibles en este ambiente. El ambiente de comandos BSD está directamente conectado con la capa del ambiente del *kernel*.

El ambiente del *kernel* exporta servicios BSD a las capas superiores del sistema por medio del sistema de librerías. Carbón, Cocoa y Java son los tres ambientes de aplicación principales para desarrolladores:

- Carbón es una adaptación de las librerías y APIs de Mac OS 9 para el sistema Mac OS X. Trae implementada la mayoría de las funciones anteriores (70%) e incluye algunos servicios específicos para el Mac OS X.
- Cocoa es una colección avanzada de APIs orientadas a objetos para el desarrollo de aplicaciones en Java y Objective-C.
- El ambiente de Java es para el desarrollo y despliegue de aplicaciones 100% Java puro y *applets* con API mixto.

Dando soporte directo a los ambientes de Carbón, Cocoa y Java están las capas del sistema que brindan servicios a todos los ambientes de aplicación. Estas capas están apiladas decrecientemente indicando que el código interno de la aplicación pueda “accesar” las capas inferiores directamente, sin la intervención de capas intermedias.

La primera de estas capas es la capa de servicios de aplicación. Esta capa contiene los gráficos y el ambiente en ventanas del sistema Mac OS X. Este ambiente es responsable de la salida y “renderización” a pantalla, impresión, manejo de eventos y dirección del cursor. También incorpora las bibliotecas y marcos de trabajo necesarios para la implementación de interfaces gráficas de usuario.

La capa de servicios de aplicación está por encima de la capa de servicios del núcleo. En esta capa se encuentran los servicios más comunes que no forman parte directamente de la interfaz gráfica del usuario. Aquí se hallan implementaciones de

abstracciones básicas de programación así como las APIs para el manejo de procesos, conexiones, recursos y memoria virtual para la interacción con el sistema operativo.

El ambiente del *kernel* es el último estrato del sistema, por debajo de la capa de servicios del núcleo. El ambiente del *kernel* provee de funcionalidad esencial para todas las capas que se encuentran por encima de él, tales como:

- multitarea apropiativa
- memoria virtual avanzada con protección de memoria
- multiproceso simétrico
- acceso multi-usuario
- sistema de archivos basado en el sistema de archivos virtual (VFS)
- controladores de dispositivos
- conectividad en red

1.5 Uso de memoria

El manejo de memoria difiere tanto en el sistema clásico de Mac como en el sistema X. A continuación se describen algunas características del manejo de la memoria en ambos sistemas operativos.

1.5.1 Mac OS clásico

En el Mac OS clásico, el manejo de la memoria era menos que ideal. Primero que nada, el usuario debía ajustar manualmente los tamaños mínimos y predilectos para cada aplicación específica. Si bien no era necesario que el usuario cambiara estos ajustes, a veces algunas aplicaciones se caerían al efectuar operaciones intensivas, requiriendo al usuario aumentar estos valores. Además, si el usuario no tenía suficiente

memoria física, las aplicaciones se rehusarían a abrir, forzándolos a activar manualmente la memoria virtual.

La fragmentación de la memoria era abundante. Cuando un computador era dejado encendido por un largo tiempo en el Mac OS clásico, y muchas aplicaciones eran abiertas y cerradas, el sistema a veces no podía utilizar toda la memoria disponible para una aplicación que iba a ser abierta. Esto era porque el sistema operativo necesitaba un segmento contiguo de memoria disponible, y luego de abrir y cerrar muchas aplicaciones, la memoria disponible podía quedar severamente fragmentada. El resultado neto era que una aplicación que era abierta a menudo tenía menos memoria total disponible de la que verdaderamente existía, debido a que solo podía utilizar el bloque contiguo más grande de la memoria disponible.

Otro problema con el manejo de la memoria en el Mac OS clásico era que la caída del segmento de memoria de una aplicación podía desbordar en el segmento de memoria de otra aplicación, logrando que la aplicación malograda desplomara a otra.

Este efecto continuaría rápidamente, a menudo comprometiendo la estabilidad de todo el sistema operativo con la caída de solo una aplicación, forzando al usuario a reiniciar inadecuadamente la computadora.

1.5.2 Mac OS X

Afortunadamente, con el advenimiento de Mac OS X, todo eso ha cambiado. Con Unix por debajo de la interfaz de usuario, ahora tenemos un sistema operativo Macintosh que tiene un sistema de manejo de la memoria muy moderno y muy eficiente.

Ya no es necesario asignar la cantidad de memoria deseada a una aplicación

específica. El sistema de manejo de la memoria del Mac OS X asigna en forma dinámica la memoria a las aplicaciones, así que si una aplicación necesita más memoria para realizar tareas intensivas, el Mac OS X le asignará automáticamente más memoria (sin tener que reabrir la aplicación). ¿El efecto? No más aplicaciones que se quejan por la carencia de memoria, y no más errores por falta de memoria.

Además, si una computadora no tiene suficiente memoria física para una aplicación, el Mac OS X cambiará automáticamente a memoria virtual, permitiendo que la aplicación dependa en espacio del disco para su memoria. Esto, por supuesto, hace que la aplicación funcione mucho más lento, pero esta característica del Mac OS X elimina la necesidad de activar manualmente la memoria virtual. La cantidad de espacio de disco usada para la memoria virtual es también asignada dinámicamente según las necesidades de las aplicaciones abiertas.

Otra ventaja del Mac OS X es el hecho de que no hay un requisito para un bloque contiguo de la memoria. El Mac OS X por lo tanto puede utilizar toda la memoria física disponible para una aplicación antes de recurrir a la memoria virtual. Además, la memoria asignada a cada aplicación se protege contra otros segmentos de la memoria, previniendo que la caída de aplicaciones lleven abajo a todo el sistema.

Con estas características, el Mac OS X provee un sistema operativo mucho más estable y eficiente.

Sin embargo, hay que notar que el sistema de uso de memoria de OS X también tiene sus problemas. El mayor cambio de Mac OS 9 a Mac OS X es que el sistema y sus aplicaciones utilizan mucha más memoria. Los requisitos del sistema de Mac OS X dicen que requiere un mínimo de 128MB de RAM, pero se recomienda un mínimo de 512MB para que Mac OS X funcione aceptablemente. Esta cantidad es subjetiva, pero una cosa es segura: de cuanta más RAM dispone Mac OS X, más rápido funcionará.

También Mac OS X tiende a usar toda la memoria disponible, incluso si hay demasiado de ella. Esto es porque Mac OS X guarda toda la información que puede en memoria, de manera que potencialmente puede volver a utilizar esa información sin tener que volverla a poner en memoria (el término de UNIX para guardar en memoria es paginación de memoria). El rendimiento de Mac OS X cae cuando toda la memoria está siendo usada, porque tiene que empezar a borrar cosas de la memoria, proceso que tiene un impacto en el rendimiento. Este problema es mucho más significativo en Mac OS X por que no se limita a las aplicaciones el uso de la memoria, simplemente toman tanta como necesiten, de manera que la memoria libre decae rápidamente.

2. EL LENGUAJE DE PROGRAMACIÓN OBJECTIVE C

El lenguaje de programación Objective C es un lenguaje simple de computadora diseñado para habilitar la programación orientada a objetos en forma sofisticada.

Los lenguajes de programación se estructuran tradicionalmente en dos partes: los datos y las operaciones sobre los datos. La mayoría de los ambientes de desarrollo orientados a objetos consisten de al menos tres partes:

- una biblioteca de objetos
- un *set* de herramientas de desarrollo
- un lenguaje de programación orientado a objetos y una biblioteca de soporte

Cocoa es una biblioteca extensa. Incluye una gran cantidad de marcos de trabajo con definiciones de objetos que pueden utilizar o adaptar según las necesidades del programa. Esto incluye el marco de trabajo fundación, el *kit* de aplicación (para la construcción de una interfaz gráfica del usuario) y otros.

2.1 Historia del lenguaje

Cuando Mac OS fue introducido allá en 1984 era una joya tecnológica fruto de la programación más avanzada. Los sistemas operativos que vinieron en los años siguientes empezaron a implementar características tales como la protección de memoria y la multitarea preemptiva, las cuales se han ido convirtiendo en algo imprescindible para el usuario de hoy. Apple no incluyó tales características en Mac OS simplemente porque entonces los microprocesadores no proporcionaban la potencia de proceso suficiente. Añadir características como la protección de memoria y la multitarea

preemptiva a Mac OS era una tarea muy larga y compleja porque habían de ser implementadas a nivel del núcleo del sistema operativo. Apple lo intentó varias veces pero nunca funcionó. Finalmente Apple decidió que comprar un sistema operativo moderno y hacerlo funcionar en el Mac era lo más razonable. Primero consideraron comprar BeOS de Be, Inc pero finalmente se decantaron por NEXTSTEP de NeXT, Inc.

NEXTSTEP proporcionó a Apple el sistema operativo moderno que necesitaba pero con un interfaz de usuario radicalmente diferente al de Mac OS. Apple empezó a cambiar NEXTSTEP para hacerlo más parecido a Mac OS. El resultado de todo esto fue entonces Rhapsody. Rhapsody era un novedoso y moderno sistema operativo para Mac con el típico interfaz Macintosh. El siguiente paso fue adaptar las aplicaciones existentes como Microsoft Office y Adobe Photoshop y hacerlas disponibles para Rhapsody con que los usuarios tendrían una buena razón para actualizarse.

Desafortunadamente, sistemas operativos diferentes son como idiomas diferentes. Pueden llevar a cabo las mismas tareas pero de manera muy distinta. Hacer que una aplicación Mac OS funcione en Rhapsody significa cambiar todo el código que comunica con el sistema a un nuevo lenguaje que Rhapsody entienda. Para empeorar aún más las cosas, las aplicaciones Mac están escritas en lenguajes como Pascal, C y C++ cuando el único lenguaje que puede comunicar con Rhapsody es Objective-C (y otros lenguajes desarrollados por NeXT). Los desarrolladores Mac que querían comercializar sus aplicaciones para el nuevo sistema operativo no solo tenían que enfrentarse al aprendizaje de un lenguaje totalmente nuevo sino que también tenían que re-escribir sus programas desde cero. El problema es que las aplicaciones suelen estar hechas de miles e incluso millones de líneas de código. No es sorprendente que los desarrolladores no respaldaran la idea en absoluto, por lo cual Apple tuvo que trabajar algo más con Rhapsody y proporcionar un mecanismo para facilitar el trabajo a los desarrolladores.

Objective C es un lenguaje orientado a objetos creado como un superconjunto de C con un estilo muy parecido al de Smalltalk. Originalmente fue escrito por Brad J. Cox y la corporación StepStone en 1980. En 1988 fue adoptado como lenguaje de programación de NeXTStep y en 1992 fue liberado bajo licencia GNU para el compilador GCC.

Según Esteban Manchado Velásquez, Objective C es un lenguaje "de moda", parecido al C, mayormente por la sintaxis, orientado a objetos pero mucho menos flexible que C++, pensado para hacer aplicaciones interactivas más que controladores de dispositivos y sistemas operativos.

Otra evolución del C, que también añade orientación a objetos. Objective C es más parecido a C que el C++. Es más dado a errores, aunque más flexible, y tiene mucha menos aceptación. Está relativamente ligado a las estaciones NeXTStep, y se dice que dado un entorno cualquiera que cumpla la especificación OpenStep, se puede compilar un programa escrito en Objective C sin modificación. El entorno NeXTStep, con Objective C tiene fama de dar la posibilidad de escribir aplicaciones grandes en muy poco tiempo.

Debido a que Objective C es un lenguaje orientado a objetos y es una extensión estándar de C ANSI, los programas desarrollados en C pueden ser adaptados para utilizar el marco de trabajo sin ninguna pérdida del trabajo de diseño realizado en su desarrollo.

La sintaxis del lenguaje de Objective C es pequeña, simple, sin ambigüedades y de rápido aprendizaje. Comparado con otros lenguajes orientados a objetos basados en C, Objective C es muy dinámico. El compilador mantiene una gran cantidad de información sobre el uso de los objetos en tiempo de ejecución. Esto le da a los programas desarrollados en Objective C un poder y flexibilidad inusual.

Los compiladores de Apple están basados en la colección de compiladores GNU. El código fuente de Objective C es reconocido por el compilador por la extensión .m, así como reconoce los archivos de código fuente con la sintaxis de C ANSI por la extensión .c. De forma similar, el compilador reconoce los archivos con extensión .mm como archivos de código fuente de C++ que utilizan Objective C.

2.2 Comparaciones

Objective C puede ser considerado como un híbrido entre Smalltalk y C. Mientras que una cadena puede ser declarada como un *char ** o un objeto, en donde en Smalltalk todo es un objeto, así como en Java esto permite un mejor desempeño. Aunque cabe destacar que Smalltalk es un lenguaje compilado y de mayor flexibilidad.

En contraste, C++ es asociado tradicionalmente con el lenguaje de programación orientado a objetos de Simula 67. En C++, los mensajes de los objetos de tipo estático son fijos. En Objective C los mensajes de objetos de tipo dinámico pueden ser determinados. El formato de Simula 67 permite la detección de problemas en tiempo de compilación.

Tabla II. Comparación entre Objective C y otros lenguajes

	Objective-C	Smalltalk-80	C++	Eiffel
Escritura	dinámico	dinámico	estático	estático
<i>dynamic binding</i>	explícito	implícito	virtual	SÍ
Acceso en tiempo de ejecución a nombres de	SÍ	SÍ	NO	NO
Continuación				
Acceso en tiempo de ejecución a nombres de clase.	SÍ	SÍ	NO	NO

Acceso en tiempo de ejecución a nombres de variables instanciadas.	SÍ	SÍ	NO	NO
forwarding	SÍ	SÍ	NO	NO
metaclasses	SÍ	SÍ	NO	NO
Herencia	simple	simple	múltiple	múltiple
Instanciacion	explícito	explícito	<i>new</i>	<i>create</i>
acceso a super métodos	super	super	::	Renombre
clase raiz	objeto	objeto	múltiple	múltiple
datos privados	SÍ	SÍ	SÍ	SÍ
métodos privados	NO	NO	SÍ	SÍ
Variables de clase	NO	SÍ	SÍ	NO
recolección de basura	NO*	SÍ	NO	SÍ
clases son objetos de 1era clase	NO	SÍ	NO	NO

* Implementado por un método.

Fuente: <http://www.dekorte.com/Objective-C/Comparisons.html>, septiembre 2003.

2.3 Tipos definidos de datos

En términos de tipos de datos, los objetos en Objective C son en su totalidad del tipo id (identificador). Las variables de tipo id no son nada más que identificadores de objetos, punteros a la estructura de datos del objeto. Es posible declarar y crear variables que hacen referencia a un objeto como cualquier otra variable.

En Objective C, el valor de retorno de un tipo de dato es de tipo id por omisión. Si se tuviera un método que carezca de valor de retorno, automáticamente el tipo dado de retorno será de tipo id.

Tabla III. Tipos de datos definidos en Objective-C

Tipo	Definición
Id	Referencia a un objeto (puntero a su estructura de datos)
Class	Referencia a una clase del objeto (puntero a su estructura de datos)
SEL	Un selector (un código asignado por el compilador para la identificación de los métodos)
IMP	Puntero a la implementación de un método que retorna un id
BOOL	Un valor booleano posible entre Sí/No
nil	Un puntero nulo a un objeto
Nil	Un puntero nulo a una clase

Fuente: <http://developer.apple.com/documentation/cocoa/objc.pdf>, agosto 2003.

El tipo identificador (id) puede ser usado con cualquier tipo de objeto, clase o instancia. De la misma forma, los nombres de clase pueden ser usados como nombres de tipo para instanciar clases de forma estática. Una instancia de tipo estático es declarado como un puntero a una instancia de su propia clase o como una instancia de cualquier clase heredada.

También es posible especificar una clase para un tipo de datos variable, conocido como teclado estático. Todas las variables que hacen referencia a un objeto son en realidad punteros hacia su ubicación de memoria y usualmente se realiza de forma transparente para el programador.

En la declaración de variables de tipo id, la creación de un puntero a un objeto se realiza de forma implícita. Id es por definición un puntero de tipo identificador de objeto. Únicamente para la declaración de nuevas variables de tipo puntero es necesario utilizar el constructor de punteros de C.

```
NSString *aString;
```


El asterisco al frente del nombre de la variable no es parte del nombre, sino que indica que la variable `aString` es un puntero de algún objeto `NSString`. De esta forma se declara en forma explícita la naturaleza del identificador de objetos de tipo puntero.

2.4 Características del lenguaje

Existen un sinnúmero de características propias del lenguaje, de los cuales destacan los siguientes aspectos más representativos.

2.4.1 Objetos

Como el nombre lo indica, los programas orientados a objetos están desarrollados utilizando objetos. Los objetos son la raíz del árbol de familia de Objective C.

El identificador de objetos, o sea el `id`, es distintivo de los tipos de datos. El `id` es un puntero hacia los datos del objeto y sus variables instanciadas. La clase actual del objeto no es de importancia, ya que Objective C realiza las uniones o enlaces en forma dinámica.

Los objetos nulos son representados con `nil`. El `id` de `nil` es 0. Otros tipos básicos de Objective C están definidos en el archivo de cabecera, `objc/Objc.h`

Cada objeto lleva también una variable de instancia que identifica la clase del objeto (¿Qué tipo de objeto es?). Esto permite a los objetos ser tecleados dinámicamente en tiempo de ejecución. La instrucción `isa` permite a los objetos inspeccionarse para averiguar que métodos contienen.

2.4.1.1 Creación de un objeto

La creación de un objeto es separada en dos pasos en Objective C: asignación de memoria e inicialización de campos. La asignación de memoria retorna un puntero hacia el bloque de memoria libre en donde el objeto será alojado o almacenado. La inicialización de un objeto significa predefinir los campos a algunos valores específicos. Estas operaciones son para distintos propósitos, son realizados por distintos objetos (asignación por clase e inicialización por instancia) y pueden ser escritos y llamados por cada uno en forma explícita.

2.4.2 Paso de mensajes en objetos

La principal función determinada en la programación orientada a objetos es la capacidad de enviar mensajes entre objetos.

Para que un objeto realice algo, se le envía un mensaje al objeto para que realice un método. El objeto que recibe el contenido del mensaje es llamado el “receptor” y éste determinará la acción a tomar de su comportamiento en concreto.

En Objective C, las expresiones de mensajes van declaradas entre corchetes:

```
[receiver message];  
[receiver message: arg1];
```

El receptor es un objeto, y el mensaje le indica qué hacer. En el código fuente, el mensaje es simplemente el nombre del método y los argumentos que son pasados. Cuando el mensaje es enviado, el sistema en tiempo de ejecución selecciona el método apropiado del repertorio del receptor y lo invoca.

Por ejemplo, este mensaje le indica al objeto myRect que ejecute el método display, el cual despliega un rectángulo en pantalla:

```
[myRect display];
```

Los métodos pueden además tener argumentos. El siguiente mensaje de ejemplo le indica al objeto iniciar su ubicación en las coordenadas de la ventana 30.0, 50.0:

```
[myRect setOrigin:30.0 :50.0];
```

La implementación del método -doesNotUnderstand le permite el paso de mensajes de un objeto a otro objeto (delegación de mensajes), incluyendo mensajes que retornan estructuras, variables de tipo doble, de tipo carácter, etc.

```
-doesNotUnderstand: msg { [msg sentTo:delegate]; }
```

2.4.3 Clases y herencia

Una clase se utiliza para producir objetos similares, o sea, instancias de una clase. Las clases son usadas para el encapsulamiento de datos y métodos que pertenecen a esa clase. Los métodos son operaciones que Objective C aplica a los datos y que son identificados por los mensajes.

Una característica importante de Objective C es el polimorfismo entre clases que posee, es decir, varias clases que contienen un método con el mismo nombre.

La herencia es utilizada muy a menudo para la reutilización de código. Las clases heredan variables y métodos de una clase de nivel superior llamada también super clase. Una clase que hereda algunos o todos los métodos y variables es entonces una subclase.

En Objective C, todas las clases son subclases de una super clase primaria llamada objeto.

2.4.4 Dinamismo

Usualmente el tiempo de compilación y el tiempo de enlace en los lenguajes de programación son limitados porque obligan a decidir los problemas de información encontrados en el código fuente del programador, en lugar de información obtenida por el programa en ejecución. Los lenguajes estáticos normalmente se niegan a introducir nuevos módulos o los nuevos tipos durante el tiempo de ejecución. Objective C es capaz de tomar tantas decisiones como sea posible en el tiempo de ejecución:

- Escritura dinámica: se realiza una espera breve hasta que en tiempo de ejecución se determine la clase del objeto.
- Ligadura o enlace dinámico: determinación en tiempo de ejecución de que método se debe invocar, sin necesidad de interrumpir la compilación de la aplicación sobre lo que se hace. Esto permite cambiar los componentes de un programa de forma incrementada.
- Carga dinámica: los segmentos del programa no están cargados en la memoria hasta que realmente se usen, mientras se minimizan los recursos del sistema requeridos para una instancia del programa.

Si un programa ofrece un marco de trabajo para utilizar con otros programas en tiempo de ejecución, es posible implementarlo y cargarlo en tiempo de ejecución conforme sea necesario.

Los programas en Objective C están estructurados al igual que en C a través de la herencia (cómo los objetos relacionan por el tipo) y el patrón del paso de mensajes (explica cómo el programa trabaja).

2.5 Otras características

Los siguientes aspectos son características propias del lenguaje que mejoran el manejo y la definición de Objective C.

2.5.1 Categorías

Las categorías son una característica de Objective C que activa la especialización de clases sin el uso de la herencia. Las categorías pueden ser usadas para agregar comportamientos a clases existentes sin necesidad de recompilarlos. Las instancias de las clases especializadas adquieren entonces nuevos comportamientos. Aun las instancias pre-existentes y las nuevas instancias creadas, usadas dentro de las clases implementadas de Cocoa, adquieren los comportamientos nuevos.

2.5.2 Protocolos

Los protocolos en Objective C son muy similares a las interfaces en Java. Los protocolos determinan el comportamiento de los objetos en forma independiente de la clase de los objetos. El marco de trabajo Cocoa contiene una extensa variedad de protocolos para implementarlos.

2.5.3 Comportamientos

Un objeto en Objective C puede “invocar” comportamientos en una forma dinámica. Por ejemplo, un programa puede aceptar entradas de usuario que especifican

un comportamiento a invocar en una aplicación en ejecución. La capacidad de preguntar a un objeto a invocar un comportamiento sin la ayuda del compilador contribuye a la integración del marco de trabajo Cocoa con otros lenguajes de tipo *script*.

2.5.4 Reemplazo

Reemplazo (*posing* en inglés) es la capacidad de sustituir universalmente una clase por otra. Cada vez que se realiza un intento de instanciar una clase, se crea una instancia de la clase “*posing*”.

Estas clases trabajan con las librerías compiladas del marco de trabajo Cocoa. Si una aplicación desarrollada en Cocoa incluye un objeto de estas librerías, la clase *posing* es utilizada en lugar de la clase original en todos los casos. *Posing* es una característica distintiva de Objective C que interfiere con la seguridad que Java ofrece y no es recomendable utilizarla en aplicaciones del marco de trabajo Cocoa desarrolladas en Java.

2.5.5 Tiempo de ejecución (*runtime*)

Objective C incluye un sistema de tiempo de ejecución muy similar en muchas formas con la máquina virtual de Java (JVM). Este sistema permite la carga dinámica de los objetos de Objective C. A diferencia del JVM, Objective C es pequeño y no provee soporte multi-plataforma o rasgos de seguridad. El sistema de ejecución de Objective C está escrito en C estándar y puede ser utilizado por programas con código fuente de C o C++ aun sin haber sido compilados de forma previa por el compilador de Objective C.

2.6 Manejo de memoria

Una de las más grandes barreras en el aprendizaje de la programación con Objective C es el manejo y administración de la memoria. Si se tiene experiencia en Java o Perl, donde la memoria es administrada por cada programador, el pensamiento de tener que manejar las cosas puede ser difícil. En cambio, si se tiene un conocimiento previo de C, encontrará que el manejo de memoria en Objective C es un gran paso hacia adelante con una simple llamada de instrucción malloc/free. (James Duncan Davidson).

La administración de la memoria es un tema importante en los lenguajes de programación. Algunos de los problemas encontrados en aplicaciones desarrolladas por programadores novatos son causados por una deficiente administración de la memoria. Cuando un objeto es creado y pasado a una cantidad considerable de objetos “consumidores” dentro de la aplicación, ¿qué objeto es responsable de disponer de él y cuándo? Si un objeto no es liberado cuando ya no es necesario, la memoria se agota. Si el objeto es liberado demasiado pronto ocurren problemas con los demás objetos que asumen todavía su existencia, trabando por lo general la aplicación.

El marco de trabajo Fundación define un mecanismo y una política que se asegura que los objetos son liberados únicamente cuando ya no son necesarios.

La política es muy simple: solamente el programador es responsable de disponer de todos los objetos de su propiedad. Todos los objetos que son creados son propios del programador, ya sea copiándolos o reteniéndolos. La regla indica entonces que un objeto nunca podrá ser liberado si no ha sido creado o retenido; como resultado de la liberación del objeto en forma prematura se producen errores difíciles de rastrear aun si es de fácil solución.

2.6.1 Inicialización y liberación de un objeto

Como fue mencionado anteriormente, un objeto es creado usualmente usando el método malloc e inicializado con el método init (o una variante de dicho método). Cuando un arreglo es inicializado con el método INIT, el método inicializa la instancia del arreglo de variables hacia los valores de omisión y completa otras tareas de inicio. Por ejemplo:

```
NSArray * array = [[NSArray alloc] init];
```

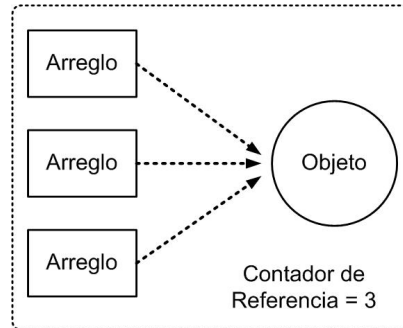
Para la liberación de un objeto creado, se envía un mensaje release hacia el objeto. Si no se encuentran otros objetos registrados de interés para el objeto, serán entonces desalojados y removidos de la memoria.

Cuando un objeto es desalojado, el método invocado por Objective C es dealloc, dando la oportunidad al objeto de liberar los objetos que ha creado, liberando la memoria utilizada.

2.6.2 Contador de referencia

Cada objeto en Cocoa tiene asociado un contador de referencia para permitir a los múltiples objetos registrar su dependencia e interés en algún otro objeto y removerlo de la memoria cuando ningún otro objeto presente algún interés directo sobre él. Cuando un objeto es ubicado o copiado, su contador de referencia es automáticamente puesto en 1. Esto indica que dicho objeto está actualmente en uso en un lugar. Si el objeto es pasado a otros objetos, en espera de asegurarse que el objeto se encuentra activo para su uso, se puede utilizar el método retain para incrementar el contador.

Figura 3. Objeto utilizado por tres arreglos diferentes

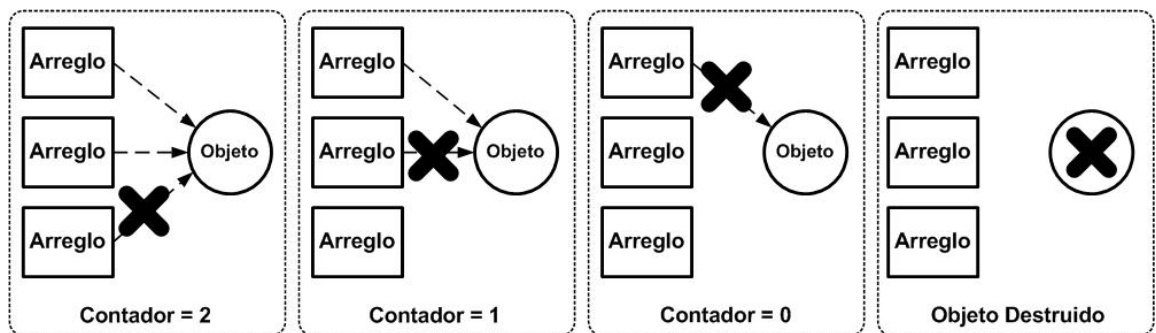


Fuente: http://www.macdevcenter.com/pub/a/mac/excerpt/Cocoa_ch04/index.html,
 noviembre 2003.

Imagine que se tiene un objeto que está retenido por tres arreglos diferentes. Cada arreglo retiene al objeto para asegurarse de que se mantiene disponible para su uso posterior; por consiguiente, el objeto tiene un contador de referencia con valor de 3.

Cuando el objeto deja de ser útil, un mensaje de liberación decrece el contador de referencia. Cuando el contador de referencia alcanza el valor de 0 para la liberación del objeto, se invocará el método dealloc para destruir el objeto.

Figura 4. Liberación progresiva de un objeto



Fuente: http://www.macdevcenter.com/pub/a/mac/excerpt/Cocoa_ch04/index.html,
 noviembre 2003.

2.6.3 Colas de auto-liberación

Según la política de disponer de todos los objetos creados, si el propietario de un objeto debe liberar el objeto dentro de su esquema programático, ¿cómo es posible darle ese objeto a los otros objetos? O dicho de otra manera, ¿cómo se puede liberar un objeto que debe devolver una llamada de un método? Una vez que se está utilizando el método, no hay forma de regresar y liberar dicho objeto.

La respuesta al dilema es el método autorelease definido en la clase NSObject en conjunto con la funcionalidad de la clase NSAutoreleasePool. El método de autorelease marca al receptor para una liberación posterior en una cola de auto liberación NSAutoreleasePool.

Esto le permite a un objeto mantenerse por un tiempo mayor que el objeto propietario, así otros objetos tienen acceso a él. Ejemplo:

```
NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
...code...
[pool release];
```

En cada aplicación se coloca al menos una vez una cola de auto liberación (por cada hilo de control que se está ejecutando en la aplicación), pero puede haber muchos más. Se coloca un objeto en la cola enviando un mensaje de autorelease. En el caso de que la aplicación se encuentre en un ciclo, cuando el código se finalice de ejecutar y el control vuelva al objeto de la aplicación, el objeto envía un mensaje de release a la cola de auto liberación, y la cola envía un mensaje de liberación a cada objeto que contenga. Cualquier objeto que alcance el valor de 0 en el contador de referencia automáticamente es desubicado y desalojado de la memoria.

Cuando un objeto se usa solamente dentro del alcance del método que lo crea, es posible desalojarlo inmediatamente enviándole un mensaje de liberación. Por otra parte, se utiliza el mensaje de autoliberación para todos los objetos que se crean y que otros objetos puedan escoger si deben o no retenerlos.

No es conveniente la liberación de un objeto que se recibe de otros objetos, a menos que sea retenido por alguna razón. Al hacer esto, causará que el contador de referencia alcance el valor de 0 de forma prematura, y el sistema destruirá el objeto, obligando a que ningún otro objeto dependa de él. Cuando un objeto dependiente del objeto destruido intenta acceder a él, la aplicación generará un error no esperado. Este tipo de errores programáticos son difíciles de encontrar.

Se puede asumir que un objeto permanece válido en el método en que fue recibido y se mantendrá válido para el ciclo que lo maneja. Si se desea mantenerlo como una variable de instancia, se debe mandar un mensaje de retain y cuando deje de usarse un mensaje de autorelease.

2.6.4 Retención de objetos en el método accesor

Uno de los lugares primordiales donde es necesario estar alerta en el manejo de la memoria son los métodos accesoros de las clases. A primera vista, es obvio que se desee liberar una referencia hacia un objeto viejo y retener uno nuevo. Sin embargo, las llamadas a código pueden realizarse una gran cantidad de veces hacia el mismo objetos como argumento, entonces el orden de la liberación y retención de las referencias de un objeto es un aspecto muy importante.

Como regla, se debe retener el nuevo objeto antes de liberar el objeto antiguo. Esto asegura que todo trabaje de manera anticipada, aun si el nuevo y antiguo objeto es el mismo. Si en dado caso se hiciera a la inversa, y el objeto nuevo y antiguo fueran en

realidad el mismo, el objeto podría ser removido permanentemente de la memoria antes de ser retenido.

Expresado en código la regla de retención y liberación sería:

```
- (void)setProperty:(id)newProperty
{
    [newProperty retain];
    [property release];
    property = newProperty;
}
```

Existen otras alternativas para asegurar las conexiones entre los métodos, muchas de las cuales son apropiadas y válidas dependiendo de la situación. Sin embargo, el código anterior es la manera más usual y sencilla posible que funciona en Objective C

2.6.5 Reglas importantes

En el manejo de memoria en Cocoa es importante tener siempre presente las siguientes reglas:

- Los objetos creados por alloc o copy tienen un contador de 1.
- Los objetos obtenidos por cualquier otro método tienen un contador de 1 y residen en la cola de autoliberación. Si se desea mantenerlo por un tiempo prolongado mayor conviene retenerlo.
- Cuando un objeto es agregado a una colección, entonces está retenido. Si se remueve de la colección, es liberado. La liberación de un objeto de una colección (como por ejemplo NSArray) libera todos los objetos contenidos en esa colección.

- Asegurarse de que existe una cantidad considerable de mensajes tanto de liberación o de autoliberación a los objetos como de mensajes de alloc, copy, mutableCopy o retain enviados. En otras palabras, asegurarse de que el código escrito esté bien balanceado.
- Retener y luego liberar los objetos en los métodos.
- Los objetos NSString creados utilizando el constructor @" . . ." son constantes efectivas en el programa. El envío de mensajes de retención o de liberación no tienen ningún efecto sobre dichos objetos. Es por ello que no se liberan las cadenas creadas con el constructor @" . . ."

Si las reglas anteriores son aplicadas de manera constante y se mantiene el contador de retención de los objetos de manera balanceada, el manejo de memoria en las aplicaciones se realiza de forma efectiva.

3. JAVA PARA MAC OS X

Java 1.4.1 es el siguiente nivel de la liberación certificada de la plataforma Java 2, edición estándar, para el Mac OS X de Apple. Esta versión contiene centenares de nuevos rasgos, mejoras de desempeño y beneficios únicos integrando las tecnologías más importantes del sistema operativo OS de Mac, surgiendo y brindando soluciones de desarrollo y entrega de aplicaciones de escritorio de alto volumen.

Las mejoras en la versión 1.4.1 con respecto a la versión anterior, la 1.3.1, incluyen soporte para nuevas I/O nativas, tecnología XML y Web Services, más APIs de seguridad, soporte Unicode 3.0 y más.

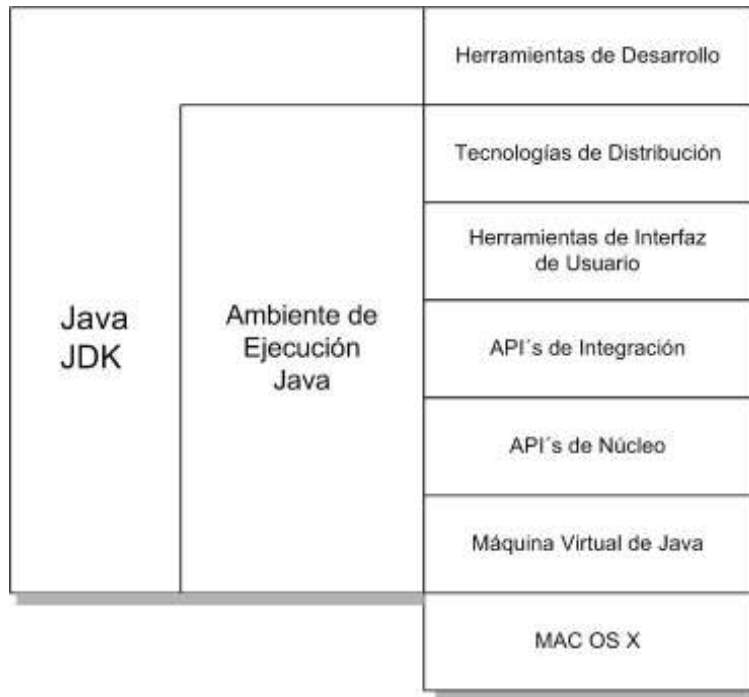
Java 1.4.1 para Mac OS X se encuentra mucho más integrado al sistema operativo que cualquier otra versión anterior de Java en dos aspectos:

- Integración con la plataforma Java 2, segunda edición
- Integración interna del sistema operativo nativo

Todas las aplicaciones de Java y *applets* usan Aqua, el GUI del Mac OS X. Además existe una amplia variedad de herramientas para desarrollar las aplicaciones de Java en Mac OS X, incluyendo IDEA de IntelliJ, JBuilder de Borland, JDeveloper de Oracle, Eclipse y NetBeans de Sun Microsystems por nombrar algunos.

La completa implementación de Java en el Mac OS X incluye los componentes que normalmente están asociados con el ambiente de ejecución de Java (JRE), así como el Java Kit de Desarrollo de Software (SDK).

Figura 5. Componentes individuales del JRE y SDK construidos en el Mac OS X nativo



Fuente:

<http://developer.apple.com/documentation/Java/Conceptual/Java141Development/Overview/index.html>, septiembre 2003.

3.1 Características de Java

Hay muchas razones por las que Java es tan popular y útil. Aquí se resumen algunas características importantes:

- Java integrado: la máxima de Sun, “Escribir una vez, ejecutar en cualquier lugar”, sólo es verdad si Java está por todas partes. Por lo menos, en el sistema Mac OS X, el ambiente de ejecución de Java JRE se encuentra integrado en el sistema operativo. Esto significa que para el desarrollo de aplicaciones en Mac

OS X, Java estará instalado y configurado para trabajar con los clientes de la aplicación. Java es una de las tres API de alto nivel para el desarrollo de aplicaciones (los otros dos son Cocoa y Carbón).

- **Orientación a objetos:** Java está completamente orientado a objetos. No hay funciones sueltas en un programa de Java. Todos los métodos se encuentran dentro de clases. Los tipos de datos primitivos, como los enteros o dobles, tienen empaquetadores de clases y estos objetos por sí mismos son los que permite que el programa las manipule.
- **Simplicidad:** la sintaxis de Java es similar a ANSI C y C++ y, por tanto, fácil de aprender; aunque es mucho más simple y pequeño que C++. Elimina encabezados de archivos, preprocesador, aritmética de apuntadores, herencia múltiple, sobrecarga de operadores, instrucciones struct y union y, además, plantillas.
- **Compactibilidad:** Java está diseñado para ser pequeño. La versión más compacta puede utilizarse para controlar pequeñas aplicaciones. El intérprete de Java y el soporte básico de clases se mantienen pequeños al empaquetar por separado otras bibliotecas.
- **Portabilidad:** los programas desarrollados en Java se compilan en el código de bytes de arquitectura neutra y se ejecutan en cualquier plataforma con un intérprete de Java. Su compilador y otras herramientas están escritas en Java. Su intérprete está escrito en ANSI C. De cualquier modo, la especificación del lenguaje Java no tiene características dependientes de la implantación.

- Amigable para el trabajo en red: Java tiene elementos integrados para comunicación en red, *applets*, sitios web, aplicaciones cliente-servidor, además de acceso remoto a bases de datos, métodos y programas.
- Soporte a GUI: la caja de herramientas para la creación de ventanas abstractas (*Abstract Windowing Toolkit*) de Java simplifica y facilita la escritura de programas con interfaz gráfica de usuario orientados a eventos con muchos componentes de ventana.
- Carga y vinculación incremental dinámica: las clases de Java se vinculan dinámicamente al momento de la carga. Por tanto, la adición de nuevos métodos y campos de datos a clases no requieren de recompilación de clases del cliente. En C++, por ejemplo, un archivo de encabezado modificado necesita reunir todos los archivos del cliente. Además, las aplicaciones pueden ejecutar instrucciones para buscar campos y métodos y luego utilizarlos de manera correspondiente.
- Uso de la memoria: en otras plataformas, cada aplicación de Java consume algo de la memoria del sistema. Así que es posible terminar usando más memoria de la que comúnmente se necesita al ejecutar múltiples aplicaciones de Java. Otros lenguajes, como C o C++, resuelven este problema usando lo que se llama las bibliotecas compartidas. Apple desarrolló una nueva tecnología innovadora que permite compartir el código de Java a través de múltiples aplicaciones. Esto reduce la cantidad de memoria que las aplicaciones de Java normalmente usan. Y encaja perfectamente con la idea principal de la máquina virtual VM de Java, permitiendo el Mac OS X permanecer compatible con Java estándar.
- Internacionalización: los programas de Java están escritos en Unicode, un código de carácter de 16 bits que incluye alfabetos de los lenguajes más utilizados en el

mundo. La manipulación de los caracteres de Unicode y el soporte para fecha/hora local, etc. hacen que Java sea bienvenido en todo el mundo.

- Hilos (*threads*): Java proporciona múltiples flujos de control que se ejecutan de manera concurrente dentro de uno de sus programas. Los hilos permiten que los programas emprendan varias tareas de cómputo al mismo tiempo, una característica que da soporte a programas orientados a eventos, para trabajo en red y animación.
- Seguridad: entre las medidas de seguridad de Java se incluyen restricciones en sus *applets*, implantación redefinible de *sockets* y objetos de administrador de seguridad definidos por el usuario. Hacen que las *applets* sean confiables y permiten que las aplicaciones implanten y se apeguen a reglas de seguridad personalizadas.

Java está completamente orientado a objetos. Un programa consta de una o más clases, éstas pueden organizarse en paquetes. Las clases de Java definen objetos de software encerrando los miembros de los datos (campos) y a los miembros de la función (métodos). Los miembros pueden designarse como privados, protegidos, de paquete o públicos, lo que proporciona una manera conveniente de definir la interfaz pública y el dominio privado de un objeto.

El mecanismo de Java para la herencia es la extensión de clase, que permite las relaciones de tipo y la reutilización de código. Sólo se permite la herencia simple. La sobreescritura dinámica de métodos soporta polimorfismo y permite la construcción de objetos intercambiables que se adecuan a una interfaz uniforme. La superclase abstracta de Java le ayuda a planear interfaces uniformes para un conjunto de objetos compatibles de conexión, además de proporcionar código común para éstos.

El ambiente de aplicación de Java tiene tres componentes importantes en Mac OS X:

1. Un ambiente de desarrollo, incluyendo el compilador de Java (javac) y un depurador de código (jdb), así como otras herramientas, tales como javap, javadoc y el visor de applets.
2. Un ambiente de ejecución en tiempo real que consiste en el alto rendimiento de la máquina virtual de Java Hotspot de Sun, el compilador bytecode y los paquetes básicos de Java.

La máquina virtual de Java está localizada en / System/Library/Frameworks / JavaVM.framework/Libraries. Los paquetes básicos incluyen java.lang, java.util, java.io, y java.net localizados en el archivo de classes.jar en el directorio Clases del mismo marco de trabajo.

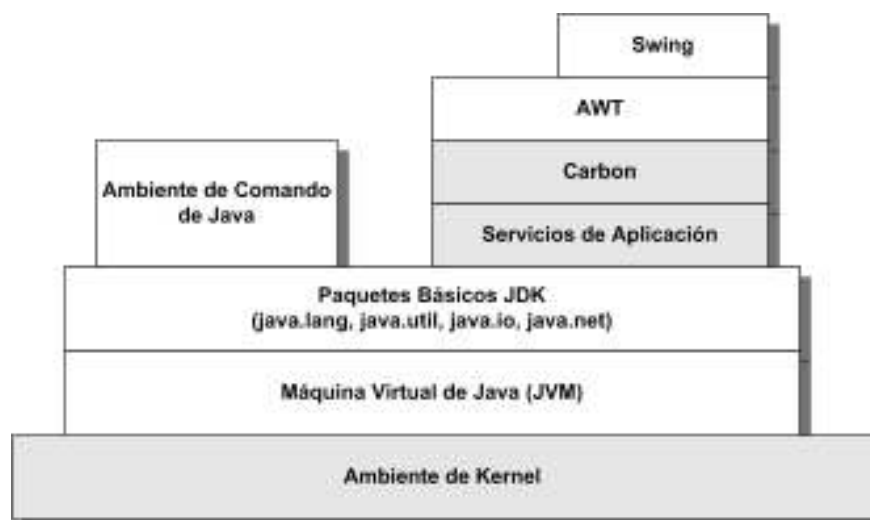
3. Un marco de trabajo de aplicación que contenga las clases necesarias para la construcción de una aplicación en Java.

Los más importantes de estos paquetes son java.awt y javax.swing, comúnmente conocidos como AWT (*Abstract Windowing Toolkit*) y swing. El paquete de AWT implementa componentes estándar de interfaz de usuario comunes (como los botones y los campos texto), capacidades de dibujo básicas, administrador de diseño y un mecanismo de manejo de eventos. El paquete de swing proporciona un juego muy extendido de componentes de interfaz del usuario. Swing incluye las versiones del componente de AWT existente más un juego rico de componentes de nivel superior, como la vista en árbol, cuadros de lista y paneles con pestañas. El paquete AWT y swing están en un archivo jar localizado en JavaVM.framework/Classes/classes.jar.

La máquina virtual de Java conjuntamente con los paquetes básicos de Java (`java.lang`, `java.util` y `java.io`) son equivalentes a la capa de servicios del núcleo del sistema para los ambientes de Carbon y Cocoa. Ellos utilizan los recursos del ambiente del *kernel* para llevar a cabo e implementar servicios de bajo nivel como manejo de procesos, conexiones y entrada/salida. No necesitan “accesar” directamente a la capa de servicios del núcleo del sistema de software.

Todos los demás componentes de Java en Mac OS X están en una capa superior por encima del VM y los paquetes básicos. Si un programa de Java no tiene una interfaz del usuario (por ejemplo, una herramienta o un servidor de aplicación), todo lo que necesita es esta base para ejecutarse. Pero una aplicación 100% Java pura o *applet* (que, por la definición, tiene una interfaz del usuario gráfica) debe usar AWT o *swing* de los cuales ambos tienen enlaces con otros marcos de trabajo o bibliotecas en la capa de servicios de aplicación. *Swing* se encuentra en una parte de la capa primitiva del paquete AWT. Juntos AWT y *swing* son arquitectónicamente equivalentes a una caja de herramientas orientada a la interfaz gráfica del usuario.

Figura 6. Ambientes en capas de aplicación de Java



Fuente: <http://developer.apple.com/java/java141sysproperties.pdf>, septiembre 2003.

3.2 Otras características

Existen otras características que deben ser tomadas en cuenta, como por ejemplo: las interfaces, multihilos, el puente de Java, Java puro, los *applets*, *javabeans* y la máquina virtual de Java.

3.2.1 Interfaces

En Java se pueden especificar interfaces. Una interfaz especifica los métodos requeridos para que cualquier clase funcione para ciertos propósitos. Por ejemplo, una interfaz que permita la ordenación puede pormenorizar el comportamiento de cualquier objeto que pueda ordenarse. Una clase puede implantar cualquier número de interfaces para que sus objetos soporten ciertos comportamientos prescritos logrando que una clase esté de acuerdo con una interfaz al implantar los métodos requeridos.

Los objetos pueden contener múltiples interfaces. Para promover la reutilización de los objetos, es importante que cada objeto dependa lo menor posible de otros objetos. Las interfaces de Java pueden ser utilizadas para minimizar las dependencias entre los objetos. Un objeto en Java se construye para que trabaje con cualquier objeto que implemente una interfaz en particular sin la necesidad de conocer la clase o cualquier otra información relacionada con el objeto. Entre menos conozca un objeto sobre otros es menos común la dependencia sobre otros objetos.

Los elementos de una interfaz son públicos por definición: no es necesario poner la palabra `public`, ni se pueden cambiar sus derechos de acceso.

Las interfaces en Java son muy similares a los protocolos de Objective C. Cocoa utiliza de forma muy amplia y extensa los protocolos. Cuando Java se utiliza de forma

combinada con Cocoa, muchos de los protocolos de Cocoa son accesados con sus equivalentes en interfaces de Java.

3.2.2 Multihilos

Los programas escritos en lenguajes como Fortran, C o C++ ejecutan únicamente un flujo de control. A éstos se les llama programas de un solo hilo. Java permite la coexistencia de varios hilos y su ejecución independiente dentro de un programa. La opción de multihilos arroja muchos beneficios, incluyendo el desacoplamiento de los cálculos que consumen tiempo y el manejo de eventos que se producen a partir de una respuesta.

Un proceso de Java puede crear y manejar, dentro de sí mismo, varias secuencias de ejecución concurrentes. Cada una de estas es un hilo independiente y todos ellos comparten el espacio de dirección como los recursos del sistema operativo. Por tanto, cada hilo puede “accesar” a todos los datos y procedimientos del proceso, pero tiene su propio contador y su pila de llamado a procedimientos.

Con múltiples hilos, es posible organizar la ejecución de las tareas en forma concurrente o paralela dentro de un programa.

3.2.3 El puente de Java

Apple provee una tecnología llamada “El puente de Java” (*Java Bridge*). El puente de Java habilita la interacción transparente entre los objetos de Java y los objetos de Objective C basados en Cocoa. Los objetos de Java pueden especializar los objetos de Objective C. El puente de Java puede manejar problemas tales como convenciones de manejo dinámico de memoria de forma diferente entre Java y Objective C. Los

programas desarrollados en Java que utilizan objetos de Cocoa solamente pueden ser ejecutados en Mac OS X.

3.2.4 Java puro

Los programas de Java que únicamente utilizan las librerías estándar de Java son llamados programas 100% Java puros. Mac OS X es una excelente plataforma para la codificación de aplicaciones 100% Java puros. Sin embargo, si un programa de Java utiliza de cierta forma Cocoa, no funcionará en otras plataformas ajenas a aquellas que provean el marco de trabajo de Cocoa.

3.3 Applets

Los *applets* de Java agregan contenido ejecutable a páginas web. Un *applet* es un programa de Java que puede recuperarse con un explorador web y ejecutarse en la computadora local del mismo.

Un *applet* se distingue de un programa de aplicación Java común en varios aspectos:

- Un *applet* no es un programa aislado. No necesita un método `main` y está estructurado para ejecutarse dentro de otra aplicación, por lo general un explorador web.
- Una aplicación común es un programa independiente que tiene un método `main` inicial y, por lo general, no puede ejecutarse dentro de un explorador web.
- La máquina virtual de Java ejecuta aplicaciones pero no *applets*. Un *applet* puede ejecutarse como una aplicación común, si contiene un método `main`.

- Puede utilizarse un programa como un visor de *applets* para probar y depurar el código de un *applet*.
- Para que un explorador lo cargue con mayor rapidez, un *applet* suele ser de tamaño pequeño.
- Un *applet* hace E/S a través de la interfaz gráfica del usuario del explorador y sólo despliega mensajes de error con flujos de salida estándar.
- Un *applet* es una subclase de la clase *Applet* del paquete `java.applet`.

Los visores de *applets* y los exploradores web imponen las siguientes restricciones de seguridad en *applets* cargados de anfitriones remotos:

- Un *applet* se excluye de ciertas operaciones como lectura y escritura de la máquina cliente.
- Un *applet* no puede hacer conexiones de red, con excepción de las realizadas de regreso a la máquina servidor.
- Un *applet* no puede iniciar otro programa de la máquina cliente y no tiene acceso al portapapeles.
- Un *applet* no puede cargar bibliotecas, ni utilizar funciones escritas en otro lenguaje. Los *applets* dependen únicamente la API de Java.

3.3.1 *Applets* con firma

Desde JDK 1.1 es posible firmar digitalmente el archivo JAR de un *applet*. Un navegador puede usar la información almacenada en una base de identidad para otorgar/denegar el acceso a operaciones críticas por parte de *applets* firmados. La base

de datos almacena certificados de identidad e información de control de acceso, especificados por el usuario para el código que han firmado las entidades identificadas.

JDK 1.2 introdujo un esquema de depuración de control de acceso. Cuando se carga el código, se asignan permisos con base en la política de seguridad activa. Estos permiten el acceso a un recurso en particular; por ejemplo, lectura de un archivo específico, etc.

3.4 *JavaBeans*

Java incluye dentro de su lenguaje los JavaBeans, los cuales son componentes portátiles y reusables de Java que se están convirtiendo en un estándar “de facto” para la carga de objetos en programas en ejecución. Los JavaBeans tienen muchos rasgos en común con la tecnología de Cocoa. Las librerías estándar de Java incluyen propiedades para la carga de JavaBeans así como la identificación de interfaces y comportamientos que los componentes cargados de JavaBeans soportan.

3.5 La máquina virtual de Java

En otros lenguajes de programación un compilador traduce el lenguaje de programación a un lenguaje que el microprocesador de la computadora es capaz de reconocer y ejecutar. O bien se utiliza un intérprete que va ejecutando directamente lo que indica el código, mostrando según el caso los posibles errores que encuentra.

En Java, primero se compila el programa a un lenguaje, códigos de bytes, muy parecido al lenguaje de la máquina, pero es un lenguaje que ningún microprocesador entiende hasta el momento. Por ello se necesita de una máquina virtual de Java que sea capaz de ejecutar lo que indica ese lenguaje de códigos de bytes. A la vez, si se dispone de una máquina virtual en distintas plataformas, el mismo programa se puede ejecutar en

tantas máquinas distintas como se desee (de allí el nombre), sin necesidad de volver a compilar el programa. Por eso su gran ventaja y desventaja: se puede ejecutar en cualquier máquina y sistema operativo, pero para ello se necesita una máquina virtual de Java.

De esta forma, con una máquina virtual de Java para distintos equipos, todo el mundo puede ejecutar ese programa. Cuando se indican distintos equipos se puede estar hablando de ordenadores personales de distintas marcas y fabricantes, con distinto sistema operativo.

3.6 Recolección de basura generacional

Al momento de ejecución, Java soporta la recolección de basura: recopilación automática de objetos que ya no se necesitan. Cuando es bajo el espacio de almacenamiento en la memoria y se necesita más (por medio de `new`), se activa el mecanismo de recolección de basura. Los objetos no necesarios, llamados basura, son reunidos otra vez para su uso. En las aplicaciones, los objetos que ya no son alcanzados por referencias vivas se vuelven basura.

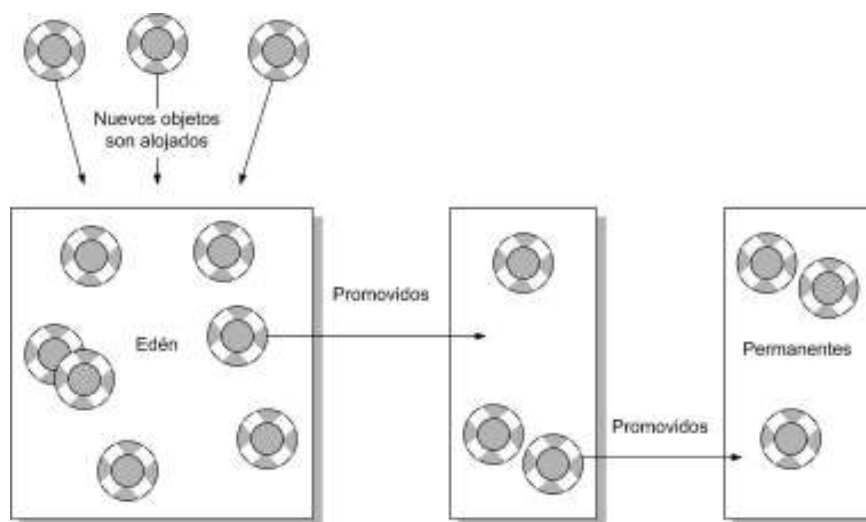
La basura se recoge sin ninguna intervención, pero a pesar de ello lleva trabajo. Crear y recoger grandes cantidades de objetos puede interferir en aplicaciones en las que el tiempo es un factor crítico. Se debe diseñar aplicaciones que sean prudentes en cuanto al número de objetos que se crean.

El recolector de basura no es garantía de que siempre vaya a haber memoria disponible para los nuevos objetos. Se podrían crear objetos indefinidamente, colocarlos en listas o cualquier otra estructura de datos y continuar haciendo esto hasta que no hubiera más espacio para luego reclamar los objetos no referenciados. La recogida de basura resuelve muchos problemas de asignación de memoria, pero no todos.

El recolector de basura generacional parte del hecho de que la mayoría de objetos mueren jóvenes. La mayoría de los objetos únicamente están en uso por un breve período de tiempo. En la recolección de basura generacional, una sección de memoria es apartada cercanamente donde los objetos son creados. Cuando este espacio es llenado, los objetos son copiados a otra sección de memoria. Cada una de estas secciones es llamada entonces generaciones.

Se mantienen los rastros de los objetos para encontrar más tarde las conexiones de regreso a la raíz cuando la recolección se da lugar.

Figura 7. Método de recolección de basura por generaciones



Fuente: <http://developer.apple.com/java/java141development.pdf>, septiembre 2003.

Sólo los objetos que todavía están en uso son desplazados hacia una generación mas reciente. La mayoría de generaciones tienen su propia estrategia para seleccionar los objetos que ya no permanecen en uso sin tener que realizar una búsqueda por la memoria. La generación más vieja no utiliza tal estrategia.

Un problema con esto es que cuando la recolección de basura ha ocurrido sobre una generación permanente, toma un tiempo atravesar por todo el espacio de memoria. Con el pasar del tiempo, múltiples instancias de aplicaciones acabarán con muchos de los mismos objetos en su respectiva generación permanente. Si un usuario está ejecutando múltiples aplicaciones Java, entonces la generación permanente de una usualmente tiene los mismos recursos que las otras lo que resulta en desperdicio de memoria.

3.7 Manejo de error y excepción

Un aspecto importante de Java es el manejo apropiado de errores durante la ejecución del programa. Entre las posibles fuentes de error se incluyen: la división entre cero, un argumento fuera del dominio legal, resultados fuera del rango permitido (demasiado grandes o demasiado pequeños), argumentos no esperados, referencias ilegales, índice de arreglo fuera de rango, archivo no existente o inaccesible y error de E/S.

Si un método encuentra una equivocación durante la ejecución, puede devolver un valor de error predefinido. Además de simples valores de error Java soporta excepciones, una manera sistemática de representar, transmitir, capturar y manejar errores bien definidos durante la ejecución del programa.

Las excepciones se representan en Java como objetos de clases que extienden, directa o indirectamente, la clase `java.lang.Exception`. Las palabras clave de Java `try`, `catch`, `throw` y `finally` se usan para manejo de excepciones y permiten capturarlas, manejarlas, detectarlas y lanzarlas.

Además de tratar con las excepciones existentes, también se pueden crear nuevas para el manejo de errores específicos.

4. DISEÑO DE APLICACIONES PARA MAC OS X

4.1 Herramientas de diseño y desarrollo

Mac OS X provee varios programas y herramientas de diseño y desarrollo de aplicaciones de los cuales Project Builder e Interface Builder son los 2 programas más utilizados por los desarrolladores.

4.1.1 Project builder

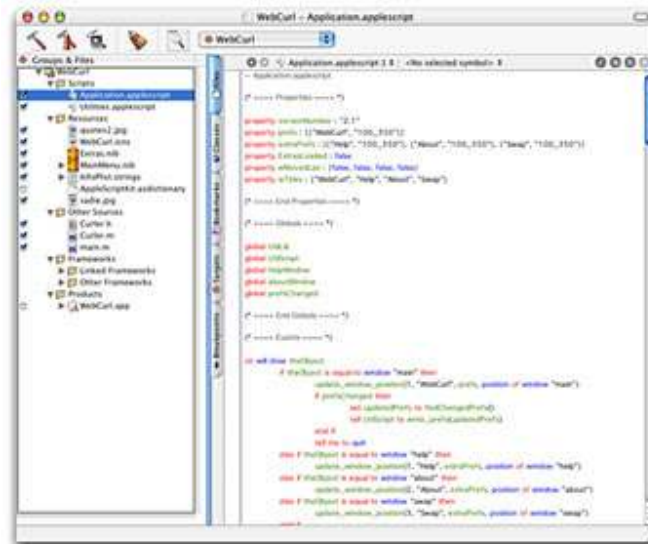
Project Builder es una herramienta de desarrollo integrada para el sistema Mac OS X que provee un ambiente de desarrollo avanzado en el cual se puede editar proyectos, búsqueda, navegación y edición de archivos, así como facilidades para “debugear” todo tipo de proyectos de software para el sistema Mac OS X.

Project Builder es un editor de código poderoso y flexible que facilita las tareas complejas involucradas en el desarrollo y distribución de programas, herramientas, librerías, extensiones, etc. Además, Project Builder posee plantillas que facilitan el desarrollo de aplicaciones que utilizan el ambiente nativo del Mac OS X, o sea, aplicaciones con código usando C, C++ , Objective C, Applescript y Java.

Las plantillas de Project Builder son pequeños proyectos predefinidos que dan un comienzo en el desarrollo de una aplicación en particular.

Project Builder trabaja con una extensa variedad de herramientas adicionales disponibles como por ejemplo Interface Builder para el entorno gráfico, compiladores como gcc, javac y jikes, y depuradores como gdb.

Figura 8. Ventana de Project Builder en Jaguar



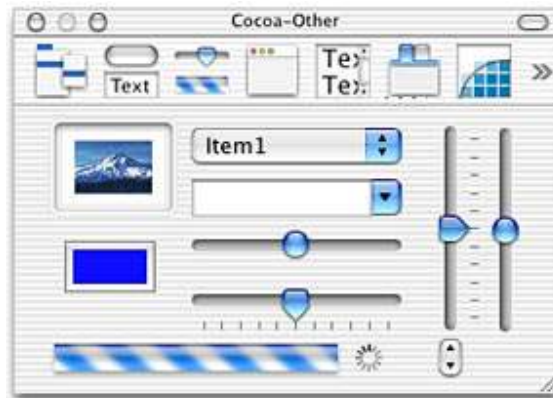
Fuente: <http://www.apple.com/macosx/features/projectbuilder.html>, noviembre 2003.

4.1.2 Interface Builder

Interface Builder es el editor gráfico para el desarrollo de componentes de interfaces de usuario en las aplicaciones de Cocoa. Interface Builder maneja una interfaz intuitiva en un ambiente de edición gráfico que maneja virtualmente cada aspecto del diseño de una interfaz adecuada para el entorno gráfico del sistema operativo.

Interface Builder almacena sus recursos en archivos “nib”. Dichos archivos son una representación estática almacenada del conjunto de objetos utilizados por la interfase de alguna aplicación.

Figura 9. Elementos usables en Interface Builder para una aplicación



Fuente: <http://www.apple.com/macosx/features/projectbuilder.html>, noviembre 2003.

Interface Builder mantiene una estrecha relación con Project Builder ya que al guardar el proyecto inserta automáticamente las clases y objetos diseñados en el código de Project Builder, reduciendo tiempo en el desarrollo de aplicaciones. Además es posible realizar pruebas de la interfaz diseñada en un modo de test que permite probar botones de comando, menús desplegables, ventanas, etc.

4.2 Planteamiento de la aplicación

Se desea contar con una ventana que permita la gestión y administración de una lista de contactos almacenando información relevante del contacto como el nombre, dirección de correo electrónico, domicilio, teléfono, información laboral del contacto y cualquier información adicional sobre ese contacto e ingresarlo a una base de datos.

La aplicación permitirá agregar, modificar y eliminar contactos además de visualizar los contactos ingresados ordenados alfabéticamente.

4.3 Configuración e instalación de MySQL

A continuación se definen los pasos generales para poder realizar la instalación y configuración de un servidor de bases de datos como MySQL sobre el sistema operativo de Macintosh. Todos estos pasos se indican de forma general dado que existe una amplia variedad de versiones de MySQL, los cuales tienen detalles específicos provistos por el constructor.

4.3.1 Requerimientos

MySQL es un servidor de base de datos relacional, rápido, flexible, confiable y fácil de usar, robusto y de libre distribución bajo los términos de el GNU. La última versión disponible actualmente en internet es la versión 4.0 de MySQL para Mac aunque se puede instalar una versión anterior (hasta la 3.23), pero el proceso de instalación es mucho más exhaustivo. Está disponible en el sitio <http://www.mysql.com/downloads>.

4.3.2 Configuración del servidor

Si se tiene un servidor de MySQL actualmente en uso o instalado y se planea reemplazar por una versión mejorada y nueva del servidor, es recomendable realizar una copia de seguridad de los datos existentes en la BDD.

En la versión 3.23 es necesario descomprimir el archivo instalador (extensión .dmg o .tar) tomando en cuenta que:

- Los archivos binarios deben de ir en la carpeta `usr/local`.
- La información de la base de datos se almacenará en la carpeta `/Users/mysql/DB_data`.
- Se habilitará la generación de bibliotecas dinámicas y estáticas.

- Se deben definir los caracteres que se utilizarán y que pueden ser soportados en el servidor.

Con la siguiente instrucción es posible descomprimir el archivo con las consideraciones anteriores y algunas opciones adicionales:

```
Shell> setenv LD_PREBIND_ALLOW_OVERLAP
Shell> ./configure --prefix=/usr/local --
localstatedir=/Users/mysql/DB_data --enable-shared --enable-static --
with-mysqld-user=mysql --with-embedded-server --with-vio --with-openssl
--with-charset=latin1 --with-extra-charsets=all --enable-thread-safe-
client
```

4.3.3 Instalación y ejecución del servidor

Primero se debe preparar el directorio que contendrá las bases de datos del usuario. En la consola de comandos escribiremos:

```
Shell> su - mysql
Password: escribe aquí la contraseña de usuario mysql
Shell> mv DB_data OLD_DB_data
Shell> mkdir DB_data
Shell> exit
```

La línea **mv DB_data OLD_DB_data** tiene el propósito de realizar una copia del antiguo directorio de la base de datos que más tarde se puede reemplazar y eliminar si la instalación se efectuó con éxito.

Después se debe de proceder con la instalación de los archivos binarios, bibliotecas, archivos de cabecera, etc.

```
Shell> sudo make install
```

```
Password: escribe aquí tu contraseña del sistema (debe pertenecer al grupo sudoer)
Shell>
```

Finalmente, y de forma opcional, es posible instalar *scripts* de bases de datos vacías como ejemplo.

```
Shell> su mysql
Password: escribe la contraseña de usuario mysql
Shell> scripts/mysql_install_db
Shell> exit
Shell>
```

Para iniciar el servidor manualmente es necesario escribir en la ventana de Terminal los siguientes comandos:

```
Shell> su
Password: escribe la contraseña del root (sistema)
# /usr/local/bin/mysqld_safe --user=mysql &
# exit
Shell>
```

Además es posible iniciar el servidor automáticamente en el arranque del sistema. Solamente es necesario copiar la carpeta de MySQL en la carpeta de elementos de arranque del sistema Mac (*/Library/StartupItems*) y agregar la siguiente línea `MYSQL=YES-` en el archivo */etc/hostconfig* con permisos de administrador.

Desde la versión 4.0 y posteriores del instalador de MySQL se incluye un archivo de paquete que realiza todas las instrucciones anteriores ejecutando un *script* de instalación genérico facilitando el proceso de instalación y ejecución del servidor.

Además de crear las bases de datos necesarias para MySQL, inicializa el servidor ejecutando el proceso como demonio en el sistema. Solamente es necesario asignar la contraseña al usuario “root” de la base de datos (super usuario) que no es el mismo usuario del sistema. Esto se logra con la siguiente instrucción desde la línea de comando:

```
Shell>./bin/mysqladmin -u root password 'nueva_contraseña'  
Shell>./bin/mysqladmin -u root -h 127.0.0.1 password 'nueva_contraseña'
```

4.4 Diseño de la base de datos

El diseño de la base de datos es del tipo entidad-relación (ER). La entidad contacto tiene como atributos los campos de identificador como llave primaria de tipo entero, nombre, dirección, teléfono, correo electrónico y descripción de tipo cadena con longitud de 50 caracteres.

Para el proceso de las eliminaciones se agregó el campo de estado colocando el valor en 1 si una tupla de la tabla se ha eliminado (borrado lógico) y en 0 si el valor de la fila se mantiene todavía activo.

La creación de la base de datos, creación de tablas y definición de campos se realizó con ayuda de un cliente gráfico de MySQL llamado CocoaMySQ versión 1.0 disponible en internet en <http://www.cocoaproject.com/cocoamysql>.

Es necesario iniciar sesión como usuario con privilegios sobre el servidor en ejecución usualmente localhost. Como entradas opcionales se encuentran el *socket* de conexión, el puerto y la base de datos a la que se desea conectar.

Figura 10. Pantalla del cliente de MySQL

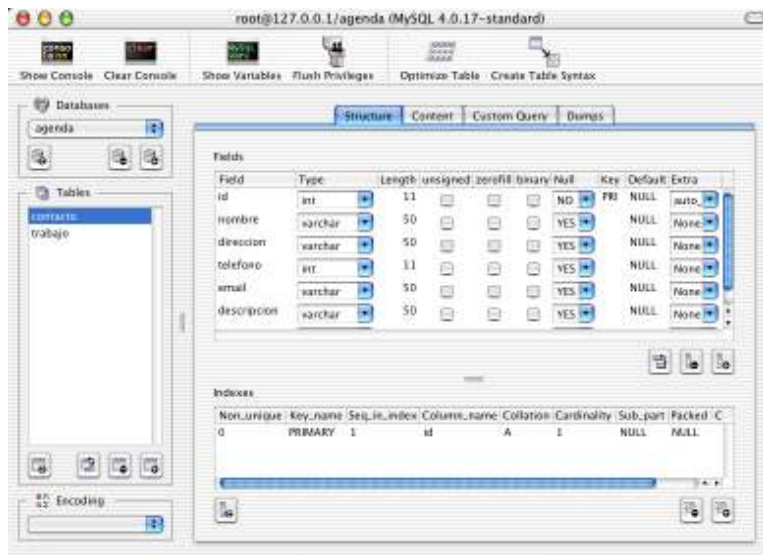


Figura 11. Ventana de conexión a base de datos en CocoaMySQL



4.5 Desarrollo de la aplicación

La declaración de las variables puede ser de tipo dinámico y estático. Las variables de tipo dinámico adquieren su tipo en tiempo de ejecución. Las variables de tipo dinámico van anteceditas por la palabra id.

```
id nombre_de_objeto;
```

Los objetos de tipo estático se declaran como apuntadores a clases.

```
NSString * nombre_de_cadena;
```

De la misma manera que en C estándar, toda la sintaxis del lenguaje finaliza con un signo de punto y coma.

Los métodos son estructurados como funciones. Después de la declaración formal de un método, el cuerpo del método va entre llaves. La declaración de instancias de los métodos se anteceden con un signo menos.

```
- (NSString *) nombre_de_metodo_o_variable.
```

El tipo de argumentos de un método se definen entre paréntesis y van entre la llave del argumento y el argumento en sí:

```
- (id) metodo: (NSString *)variable1 argumento:(int)variable2;
```

Para la definición de un objeto nulo se utiliza la palabra reservada nil, que es análogo a un puntero de tipo nulo.

Los mensajes son expresiones que consisten de una variable identificando el objeto receptor seguido del nombre del método que se desea invocar entre corchetes.

```
[objeto nombre_metodo:nombre_argumento];
```

La asignación de valores de retorno de un método a una variable es igual a la asignación normal de variables. El tipo de la variable debe de ser apropiada para recibir el valor. No es posible declarar la variable dinámicamente.

```
int result = [anObj calcTotal];
```

La conexión a la base de datos se realiza a través de la variable de tipo `iConnection` en la siguiente cadena de conexión donde `MySQLConnection.h` implementa el protocolo `IConnection`:

```
id <IConnection, NSObject> m_ICon;  
...  
m_ICon= [[MySQLConnection alloc]init];
```

El llenado automático de una tabla con valores de retorno de la consulta en MySQL se realiza obteniendo los encabezados del *recordset* por medio de un ciclo y actualizando la vista de la tabla con los valores obtenidos.

CONCLUSIONES

1. El sistema operativo Mac OS X de Apple es una nueva plataforma de desarrollo de aplicaciones al igual que los sistemas operativos Linux y Windows cuya demanda va en crecimiento.
2. Objective C es un lenguaje orientado a objetos derivado del C. En realidad un superconjunto de C ANSI con sintaxis y extensiones especiales en tiempo de ejecución que hace posible la programación orientada a objetos.
3. Objective C es el lenguaje ideal para el desarrollo de aplicaciones funcionales y productivas en un ambiente de desarrollo basado en el sistema operativo X de Macintosh ofreciendo una alternativa para crear y desarrollar proyectos de software debido al amplio conocimiento extendido por los desarrolladores.
4. El código que genera la plataforma de Java no es específico de una máquina física en particular, sino de una máquina virtual. Mac OS X integra una versión de la máquina virtual de Java.
5. Java habilita la interacción transparente entre los objetos de Java y los objetos de Objective C basados en Cocoa.
6. El desarrollo y construcción de aplicaciones para Macintosh en el sistema Mac OS X es sencillo gracias a la simplicidad de programación del lenguaje (código fuente) y a las herramientas integradas de diseño, compilación y depuración que proveen las herramientas Project e Interface Builder.

RECOMENDACIONES

1. Un aspecto importante a considerar es el poco conocimiento que se tiene de Objective C en relación con C++ lo cual hace pensar que no es un lenguaje apropiado para la programación habitual en Windows, pero existen versiones en desarrollo que ofrecen soporte para la iniciación de este lenguaje en el ambiente de Microsoft que merecen ser investigados con profundidad.
2. Existen otras soluciones disponibles aparte de MySQL como sistemas de administración de bases de datos, como por ejemplo Oracle 9i para Mac, disponible en <http://www.oracle.com/downloads>; OpenBase versión 8.0 y por supuesto SQL Server para Mac OS X Server, cada uno con sus características propias y debilidades que se ajustan a las necesidades del programa que se va a desarrollar.
3. Apple provee una extensa documentación así como software de desarrollo gratuito disponible desde su sitio de internet con el propósito de fomentar e incentivar a los desarrolladores de aplicaciones a iniciarse en la programación en Macintosh.

BIBLIOGRAFÍA

1. Apple Computer, Inc. <http://developer.apple.com>. septiembre 2003.
2. Arnold Ken y Gosling James. **El lenguaje de programación Java**. españa: Editorial Addison-Wesley/Domo. 1997. 335pp.
3. Cocoa Dev Central. <http://www.cocoadevcentral.com>. agosto 2003.
4. Duncan Davidson, James. **Learning Cocoa with Objective-C**. O'Reilly.
5. InformIT. <http://www.informit.com>. septiembre 2003.
6. Linux Journal. <http://www.linuxjournal.com>. octubre 2003.
7. López Hernández, Fernando. **Introducción al entorno de programación de Mac OS X**. <http://www.macprogramadores.org/tutoriales/introprog-macosx.pdf>, abril 2003. 10pp.
8. Mac Dev Center. <http://www.macdevcenter.com>. octubre 2003.
9. MacProgramadores. <http://www.macprogramadores.org>. septiembre 2003.
10. Macuarium. <http://www.macuarium.com>. octubre 2003.
11. O'Reilly Network. <http://www.macdevcenter.com>. noviembre 2003.
12. S. Wang, Paul. **Java. Con programación orientada a objetos y aplicaciones en la www**. México. Editorial Thomson. 2000. 444pp.
13. Sánchez Allende, Jesús, Fernández-Toribio Gabriel y Otros. **Java 2. Iniciación y referencia**. españa: Editorial McGraw-Hill. 2001. 368pp.
14. Sun Microsystems, Inc. <http://java.sun.com>. septiembre 2003.
15. Tanenbaum Andrew y Woodhill Albert. **Sistemas operativos. Diseño e implementación**. 2da. Edición. México. Editorial Prentice-Hall. 1997. 940pp.

ANEXO - MANUAL DE USUARIO

El administrador de contactos es una aplicación diseñada para ser ejecutada en el sistema Mac OS X que permite al usuario gestionar y administrar información de importancia sobre sus contactos personales. El programa permite ingresar, modificar, eliminar, buscar y visualizar contactos.

Requerimientos del sistema

Para que el programa funcione correctamente es necesario tener una computadora con el sistema Mac OS X 10.2 o superior instalada, MySQL Server versión 4.0 o superior con la base de datos “agenda” iniciada y un espacio de disco duro de aprox. 10 MB (sólo el ejecutable ocupa 1 MB).

Uso del administrador de contactos

Para iniciar el uso del programa, localice y ejecute el archivo `mysqlagenda.app`. A continuación se muestra en pantalla una ventana con el título de Administrador de contactos. Esta ventana tiene 4 pestañas: Conexión, Contactos, Agregar, y Editar que ofrecen las funciones de la aplicación las cuales se explicarán más detalladamente a continuación.

Conexión

La pestaña de Conexión tiene como fin permitir el uso de la aplicación sólo a aquellos usuarios que tienen privilegios para utilizar la base de datos de MySQL. En el primer cuadro de texto se solicita que ingrese el nombre del URL del servidor de MySQL, el cual es usualmente “localhost” y que ya está ingresado por omisión.

Figura 12. Ventana inicial del administrador de contactos



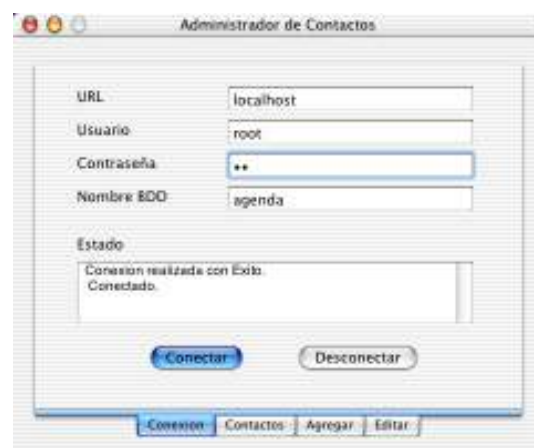
The screenshot shows a window titled "Administrador de Contactos". It contains a form with the following fields and values:

URL	localhost
Usuario	root
Contraseña	
Nombre BDD	agenda
Estado	

Below the form are two buttons: "Conectar" and "Desconectar". At the bottom of the window is a menu bar with the items: "Conexion", "Contactos", "Agregar", and "Editar".

Se solicita también el ingreso del usuario y contraseña, así como también el nombre de la base de datos para realizar la autenticación. Luego de ingresar todos los datos correctamente, pulsar sobre el botón de Conectar para acceder a las demás operaciones de la aplicación.

Figura 13. Ventana con la conexión realizada sin errores



The screenshot shows the same window as Figure 12, but with the "Estado" field containing the text: "Conexion realizada con Exito. Conectado." The "Conectar" button is now disabled (greyed out), and the "Desconectar" button is active. The menu bar remains the same.

La opción de desconectar permite cerrar la sesión y la conexión a la base de datos de forma segura. En el cuadro de Estado se muestran los mensajes del estado de la operación de conexión y de desconexión de la base de datos.

Contactos

En la pestaña de Contactos se puede visualizar la lista de contactos ingresados hasta el momento a la base de datos por el usuario y que no han sido eliminados. Pulse en el botón de Listar para desplegar la información en cuadro de texto de la ventana. El botón de Limpiar permite limpiar el contenido del cuadro de texto sin borrar ningún contacto de la lista de contactos.

Figura 14. Ventana contactos ingresados



Agregar

El proceso de ingreso de nuevos contactos se realiza en la pestaña de Agregar. Debe de proveer la información necesaria disponible del contacto e ingresarla en los

cuadros de texto correspondientes. Después pulsar sobre el botón de Agregar contacto para finalizar el proceso de inserción.

Figura 15. Ventana de ingreso de contactos nuevos



The image shows a window titled "Administrador de Contactos". It contains a form with five text input fields, each with a label to its left: "Nombre:" (with placeholder "Nombre Contacto"), "Direccion:" (with placeholder "Direccion Contacto"), "Telefono:" (with placeholder "Telefono del Contacto"), "E-mail:" (with placeholder "Correo Electronico del Contacto"), and "Descripcion:" (with placeholder "Descripcion del Contacto"). Below the fields is a blue button labeled "Agregar Contacto". At the bottom of the window, there is a navigation bar with four tabs: "Conexion", "Contactos", "Agregar", and "Editar". The "Agregar" tab is currently selected and highlighted.

Puede observar la lista de contactos ingresados en la pestaña de contactos.

Editar

Es posible realizar cambios sobre un determinado contacto previamente ingresado como por ejemplo modificar información actualizada del contacto o eliminar el contacto de su lista de contactos personales.

Para hacerlo se debe realizar una búsqueda sobre el nombre del contacto. Si el contacto se encuentra en la base de datos y no se ha eliminado, entonces se muestran los datos del contacto para su modificación.

Si se desea modificar la información del contacto, entonces solamente es necesario introducir los datos actualizados en los cuadros de texto respectivos. Los

datos que no desea que sean actualizados, no necesita modificarlos ni alterarlos en los cuadros de texto.

Figura 16. Ventana de edición de contactos



Con pulsar sobre el botón de Eliminar se procede a retirar de la lista de contactos al contacto buscado.

Existen otras opciones que la aplicación dispone y que se encuentran en el menú del programa que se ubica en la parte superior de la pantalla. Solamente las opciones que se encuentran activas se pueden realizar en la aplicación. Algunas de ellas son imprimir, copiar, pegar, minimizar ventana, mostrar información acerca de la aplicación, etc.

Figura 17. Menú de la aplicación



Algunas de las funciones mostradas en el menú no son significativas para la aplicación y, por lo tanto, no representan ningún cambio ni alteración en el uso de la misma, pero otras pueden agilizar el manejo de los datos e incluso imprimirlos si se desea.

En la opción de Acerca de MySQLagenda del menú contextual se muestra información extra de la aplicación como por ejemplo la versión del programa, el año de desarrollo y el responsable del producto.

Figura 18. Ventana de información acerca de la aplicación

