UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES
ACATLÁN

MODELO PARA EL ALMACENAMIENTO Y LECTURA DE PAQUETES DE DATOS

TESIS

QUE PARA OBTENER EL TÍTULO DE LICENCIATURA EN MATEMÁTICAS APLICADAS Y COMPUTACION

PRESENTA

Fernando Vélez Saldaña

Asesor: Oscar Gabriel Caballero Martínez

Naucalpan Estado de México Marzo de 2009

Agradecimientos

Agradezco a mis hermanos por haberme inspirado y ayudado para la realización de este trabajo, a mis compañeros y amigos de trabajo y de generación por el apoyo moral y especialmente a mis padres que me apoyaron en todo momento de mi carrera hasta el término de este trabajo

Dedicatoria

Este trabajo y trayecto de la carrera lo dedico a todos los profesores que me asesoraron, alentaron y apoyaron en el transcurso de la carrera y en especial a mi hijo Isaac y a Bety que a pesar del tiempo que dedique a la carrera y a la tesis siempre estuvieron conmigo.

Pensamientos

El presente trabajo es motivación para mi, de seguir investigando y tratando de innovar con nuevas ideas para mejorar nuestra calidad de vida, enriquecer a nuestra Universidad con trabajos de investigación y que nos dejen algo en que pensar y seguir superándonos como mexicanos.

Espero también que este trabajo sea de inspiración para que el lector no titulado, opte por este modo de titulación la cual te dejará un mejor sabor de boca que un diplomado, tesina u otro tipo de titulación ya que la idea es aportar algo a quien nos siguen además de ser una experiencia que no se debe dejar pasar.

Tabla de Contenido

Introducción

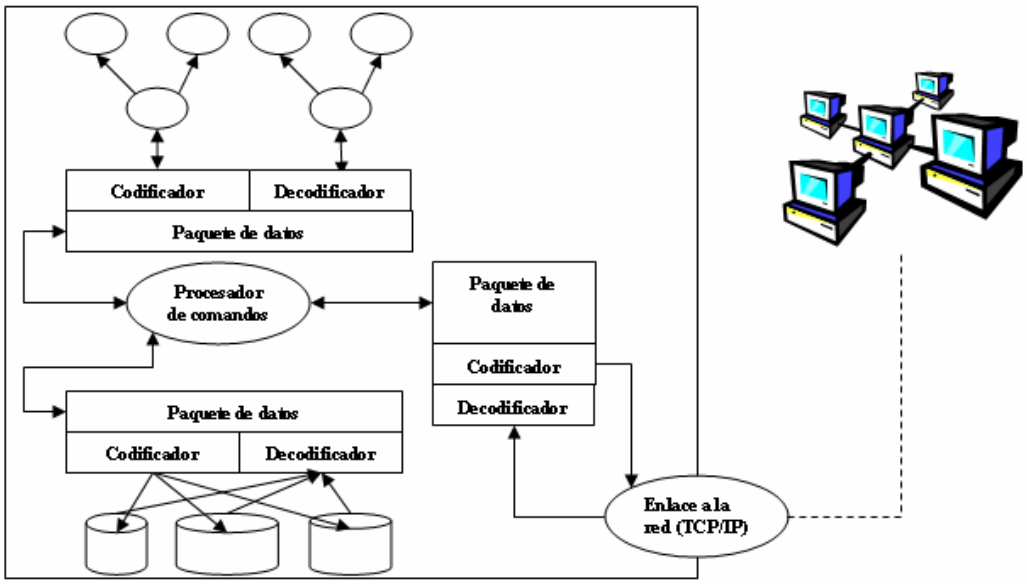
En el campo laboral informático, encontramos que, la información, es llevada por dispositivos o artefactos electrónicos de diferentes formas, debido a que cada programador, plataforma o protocolo, define la información de manera específica a sus necesidades.

Un ejemplo de ello es cuando deseamos enviar información de una base de datos por una red, a una terminal o dispositivo de pocos recursos: se obtiene la información de la base de datos, en una colección de registros, para poderla enviar por la red, es transformada a una cadena continua de caracteres y finalmente para poderlo despegar en la pantalla de la terminal o dispositivo es vaciada a una estructura datos para ser leída por la terminal.

Esto nos hace a buscar nuevas opciones, como el homologar la información para ser transportada, almacenada y leída de forma directa en diferentes plataformas, logrando incluso la reducción de costos de tiempo, recursos y código en la reutilización de procesos. Para esto se debe tener en cuenta que los métodos de lectura y escritura deben ser portables y limitados a los recursos del dispositivo de menor capacidad.

A lo que el *objetivo* de este trabajo es el encontrar una solución simple a la homologación de información, de modo de que pueda ser leída, almacenada y transmitida de forma eficiente y directa.

Y dada la *hipótesis* de definir un modelo de almacenamiento y lectura directo de paquetes de datos en base a una estructura estándar capaz de ser leída en la mayoría de las plataformas. Esta estructura se define como un paquete de datos que describe su contenido de manera compacta, y es capaz de ser almacenada, transmitida y leída de forma eficiente y directa, además de identificar un lenguaje de programación portable para poder reutilizar los métodos tanto de almacenamiento y lectura, como se muestra en la siguiente figura.



Un paquete de datos, se define dependiendo de la necesidad del usuario y a las reglas del negocio, ya sea para almacenar una imagen, un video, una base de datos, una carta, un archivo binario, un comando, etc. Por lo que el problema es definir un universo de elementos necesarios para almacenar, y leer en un paquete que se defina por si mismo y adecuarlos a las capacidades de almacenamiento y procesamiento del dispositivo de menor recursos. Con la finalidad de homogeneizar y optimizar el almacenamiento, se sugieren las siguientes estructuras para la definición del modelo:

- **Bloque de dato sencillo**. Este se puede describir como el elemento mínimo de nuestro modelo, el cual contiene un tipo de dato básico como es el texto, byte, número y fecha y hora.
- **Paquete de datos**. Conjunto de bloques de datos sencillos que conforman un todo, el cual consiste principalmente en una cabecera y un buffer que puede contener cualquier tipo de información previamente definido

Con la definición de estos elementos, es posible construir estructuras o paquetes complejos como son: cursores, textos y bloques de longitud dinámica, comandos, mensajes de error o bien un valor Nulo.

Esto es aplicable, como antes se mencionó, a aplicaciones con diferentes fines, por lo que cada uno de estos paquetes tiene un significado y función específica, en un dispositivo de almacenamiento o en la transferencia de datos de algún protocolo, a continuación se describen los paquetes básicos sugeridos en este trabajo:

- **Paquete tipo texto**. Este es muy común en aplicación cliente - servidor en donde es enviada una descripción, reporte mensaje por parte de cliente. Otros ejemplos son el envío de mensajes cortos vía e-mail, vía Web, comandos SQL, etc.
- **Paquete tipo cursor**. Este es común cuando se trabaja directamente con un manejador de Base de datos y se requiere que se transfiera un conjunto de datos tipo “tabla”, definida por varias columnas y n renglones.
- **Paquete tipo error**. Este paquete es pensado para el caso en el que una aplicación, tipo servidor, envía como resultado tipo error o excepción a la aplicación tipo cliente, o en todo caso para el almacenamiento de una bitácora en donde podemos encontrar el origen y momento del error, tanto en la aplicación tipo cliente, como en la aplicación tipo servidor.
- **Paquete tipo función**. Este es aplicado cuando la aplicación tipo cliente, solicita a la aplicación servidor que ejecute un proceso. Una función puede regresar el tipo de paquete definido en la propia función o un paquete tipo error, esto es que cada función se define como el nombre de la función, el tipo de paquete que regresa, y la lista de parámetros que se esperan dentro de la misma.
- **Paquete tipo Binario**. Este es utilizado principalmente para archivos con un formato o estructura particular, en esta es posible el almacenamiento de estructuras, imágenes, videos, etc.
- **Paquete Nulo**. Es posible que existan funciones que no regresen ningún tipo de valor, es decir, que solo se requiere que se ejecute alguna acción sin esperar ningún valor en particular, solo talvez, alguna excepción.

Una vez definidos el tipo de aplicaciones, y descrito los posibles paquetes que se pudieran construir, es tarea del analista seleccionar un lenguaje de programación que preferentemente sea soportado por los dispositivos que interactúen con estos paquetes, realizar un modelo de comunicación y asignar a un programador la tarea de construir una colección de clases que soporten las estructuras antes mencionadas.

Un ejemplo de esto podría ser la utilización del lenguaje Java como plataforma base del modelo encontramos que:

- El lenguaje java es aceptado por la mayoría de las plataformas conocidas, por lo que es uno de los lenguajes mas portables existentes.
- Al realizar una clase estándar del lenguaje Java como es el InputStream y OutputStream como base del paquete e es posible pensar que este es capaz de ser leído de forma estándar dentro del ambiente, además de que es almacenado, leído y transportado de forma directa
- Al definir un paquete como clase abstracta, es posible definir subclases capaces de tener un comportamiento diferente para cada tipo dato o paquete.
- En la implementación de máquinas de estado para la lectura y escritura de datos, se optimiza el rendimiento de los equipos para aquellos de pocos recursos y aquellos con gran demanda.

Una vez definida la estructura del paquete, los diagramas de estado correspondientes al mecanismo de lectura y escritura de este, así como el sugerir un lenguaje portable como JAVA para el desarrollo del mismo. Se *concluye* que el modelo sería una herramienta eficaz para la implementación de aplicaciones cliente servidor, gracias a su portabilidad y manejo de información tanto local como remota así como su adaptabilidad a las necesidades básicas de comunicación.

Prefacio

El siguiente trabajo esta enfocado a especialistas, profesionales o aquellos que trabajan o han tendido contacto con la informática, el cual pretende ser un trabajo de investigación y principalmente la definición de un modelo para el manejo de información.

En el primer capítulo se da a conocer el campo de investigación y entorno del cual se hablará posteriormente, donde se habla brevemente de los medios almacenamiento y comunicación, así como sus semejanzas y forma de manejar la información, mostrando las importancia de la información en la actualidad.

Una vez mostrado el marco teórico en el capítulo 2 se dan a conocer los usos mas comunes del almacenamiento y envío de información, mostrando así la necesidad de homologar para poder ser almacenada enviada y codificada de forma sencilla y directa algún dispositivo de almacenamiento o medio de transferencia. Una ves planteado el problema que existe en tener diferentes tipos de información se plantea la hipótesis de que al plantear un modelo en el cual se homologa la información en un paquete de datos o datagrama y es posible la lectura y almacenamiento de datos de una forma única dando pie a la transferencia y almacenamiento de paquetes de forma directa y sencilla.

Una vez planteada la hipótesis se describe cada uno de los componentes del modelo y como debe leerse, enviarse almacenarse y algunos ejemplos de datos con los que pueden ser utilizados.

Finalmente en el último capítulo se mencionan algunas aplicaciones del modelo y demostrar mediante código java como es posible homologar la información y manejar la información en código de forma sencilla.

Este trabajo pretende ser lo mas gráfico y entendible posible para todo interesado, no obstante es recomendable tener conocimientos básicos de programación, bases de datos y modelado UML para el mejor entendimiento.

Resumen

El presente trabajo describe un modelo de paquetes o datagramas capaces de identificar la información contenida en su interior mediante un cabezal el cual describe tanto el tipo de información como tamaño y características generales. Logrando manejar la información de manera abstracta y en consecuencia mas sencilla.
Con esto también es posible asegurar que la recepción o lectura de información este integra y en base a sus características podemos tomar decisiones como en donde almacenarla como mostrarla o solo tomar lo que nos interesa de esta.

También se ilustra como es posible la implementación de este además de ejemplificar con varios ejemplos de paquetes, comportamientos y aplicaciones.

Objetivos

Objetivo General

Una vez observado en el campo laboral las dificultades y complicaciones que se tiene en el manejo de información en diferentes plataformas, este trabajo tiene como objetivo general encontrar una solución simple a la homologación de información, de modo de que pueda ser leída, almacenada y transmitida de forma eficiente y directa.

Objetivos específicos

Además de buscar solución a pequeños problemas que se han presentado en la implementación de soluciones en este tipo de aplicaciones (multiplataforma).

1. Encontrar una forma simple para la compactar la información de forma que esta se identifique por si misma
2. Almacenar de manera directa y eficiente cualquier tipo de información
3. Transferir de manera eficiente y directa la información, sin necesidad de transformarla a diferentes formas: código, estructuras de datos, cadenas limitadas por caracteres, etc.
4. Tener la capacidad de navegar o leer directamente la información sin necesidad de cargar a una estructura de datos.
5. Reutilizar métodos de lectura y escritura de información en las diferentes plataformas
6. Encontrar un lenguaje de programación portable de modo que muchos de los métodos y bloques de código sean reutilizados en diferentes plataformas.

Capítulo I

Protocolos y tipos de almacenamiento conocidos

I. Protocolos y tipos de almacenamiento conocidos

I.I. Almacenamiento de datos (marco teórico)

Este trabajo presenta un modelo que pretende agilizar la lectura y escritura de datos en dispositivos de almacenamiento, así como el envío y recepción de datos en un medio de transferencia. Por lo que este primer capítulo, presenta una breve definición de los diferentes dispositivos de almacenamiento y los mecanismos básicos para la transferencia de datos para dar mayor claridad al problema presentado en el siguiente capítulo.

A partir de los años sesentas, las administraciones de gobierno, negocios, educación e incluso, actividades de ocio y entretenimiento, han recurrido a la invención de las computadoras, revolucionando los métodos tradicionales del procesamiento de la información grabando sus datos en medios electrónicos en lugar de guardarlos en papel **[1]**. Dando origen a una revolución tecnológica por definir y presentar la tecnología con mayor desempeño para el almacenamiento de datos, tanto en procesamiento como capacidad de almacenamiento.

En estos tiempos, el uso de dispositivos de almacenamiento, es muy cotidiano incluso indispensable para el funcionamiento básico de muchos aparatos electrónico, desde un televisor que almacena las preferencias del usuario, como el color, canales preferidos, hora, fecha, etc., hasta un banco de datos en una **súper computadora*** con información detallada de todo un negocio: clientes, inventario, facturas, ventas, etc. Esta necesidad es tan demandada, que hay un gran número de proveedores que han solventado soluciones, tanto en tecnología electrónica, como en **programas*** de **software*** para su operación y administración. Dando finalmente una gran diversidad de dispositivos y medios de guardar y recuperar la información.

Por lo que marco teórico se basa en las diferentes tipos de aplicaciones de software con las que he trabajado en campo laboral informático, como por ejemplo:

- Calani: Sistema recolector de datos en dispositivos celulares para el envío de información vía GPRS a un servidor de datos
- SIPAAM: Aplicación cliente servidor con acceso a bases de datos para la generación de apoyos económicos a los adultos mayores (Sistema Informático del Programa de Atención a los Adultos Mayores)

En donde muchos de los problemas encontrados es la interacción entre plataformas, como por ejemplo en calan se debía transformar la información para extraerla de la base de datos, enviarla por la red y finalmente transformarla para guardarla y mostrarla en el dispositivo celular (Palm Tungsten w). Otro problema con SIPAAM era el recibir la información en diferentes formatos de los diferentes puntos de la república y ser interpretados y vaciados a la base de datos para finalmente enviarla en diferentes formatos a otras instancias para su distribución y divulgación.

Por lo que si la información fuese homologada y auto descriptiva, tendríamos varias ventajas: en Calani, se podría vaciar el contenido de una consulta de bases de datos a una estructura capaz de ser leída por el dispositivo móvil, esta no tendría por que ser transformada mas de una vez para su envío recepción y lectura.

En el caso de SIPAAM, si cada paquete de información describiera su contenido, se podría tener una recepción estándar de información donde cada paquete fuera interpretado y vaciado a la base de datos de forma automática.

Por lo que a continuación se hace una breve referencia de los medios de dispositivos básicos de almacenamiento como punto de referencia de cómo es transformada la información de manera física a lógica y posteriormente se interpretará de una forma estándar (archivo) a un paquete de datos.

I.II. Dispositivos electrónicos para almacenar datos

La palabra **datos***, significa simplemente hechos, entidades independientes sin evaluar, por lo que puede pueden ser o no numéricos. Por otro lado **información*** es un conjunto ordenado de datos los cuales pueden recuperarse de acuerdo a las necesidades del usuario **[1].** Dada esta definición, deducimos que la diversidad de formas para definir, organizar y sobre todo la forma de almacenar **datos*** en algún medio físico, es increíblemente inmensa y variada.

Desde el nacimiento de las **computadoras***, se han manejado y evolucionado diferentes formas de almacenar información, en donde cada una funciona con diferentes tecnologías y un número ilimitado de proveedores. Teniendo como resultado la generación de **software*** exclusivo para cada dispositivo y por consecuencia, un mecanismo de la lectura redundante. Otro ejemplo de esta redundancia, son los medios físicos más conocidos para el almacenamiento masivo de **datos*** donde los más sobresalientes son los siguientes.

1. Discos magnéticos.

Este se conforma de un paquete de discos (disck pack), consiste en una pila de discos metálicos giratorios montados en un eje, como se muestra en la ***Figura 1.1****. Cada disco proporciona dos superficies de grabación, excepto por las superficies protectoras superior e inferior. Dos cabezas de lectura y escritura están unidas a un brazo de acceso, una para cada superficie de grabado. El disco está cubierto con óxido de hierro que pueda codificarse con datos, por medio de magnetización selectiva* [1].

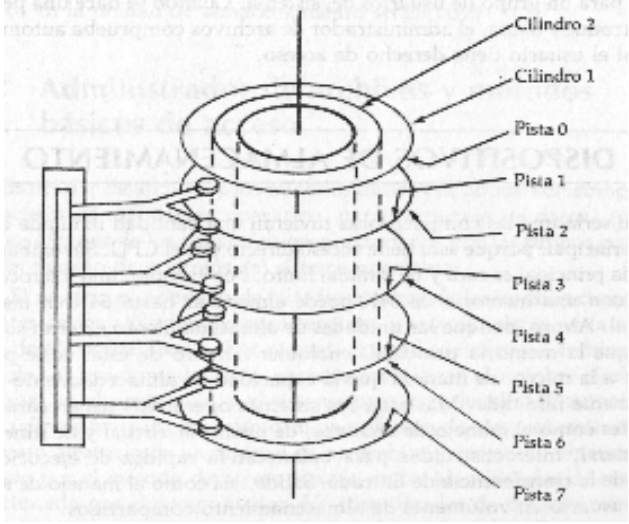


Figura 1.1

De esta tecnología, destacan dos dispositivos, que bien, vale la pena nombrarlos:

Disco Flexible*. *A partir de finales de 1970 y hasta principios de 1980 el **disco flexible*** era el principal dispositivo de almacenamiento utilizado por las **microcomputadoras***. Los **programas*** y la información se guardaban en discos flexibles. Una unidad de disco flexible, es un dispositivo que lee y escribe información de y hacia discos flexibles* **[U2]**.

Estos dispositivos, aunque son de muy bajo costo y fáciles de transportar, se han vuelto obsoletos, debido a que su almacenamiento máximo es de 1.44 **megabytes***, su lectura es lenta y deficiente, y suele averiarse rápidamente, aunque en su tiempo fue uno de los mejores dispositivos de almacenamiento.

Disco Duro*. *Actualmente, es el principal dispositivo de almacenamiento para todas las **computadoras***. Debido a que almacena mucha información, algunas veces se le llama dispositivo de almacenamiento masivo, al igual que a la cinta, discos ópticos y otros medios que pueden almacenar una gran cantidad de **información*** **[U2].**

Estos dispositivos almacenan a evolucionado de tal forma que, gracias a su rápido acceso y a su gran capacidad de almacenamiento de **datos***, que se le ha dado la funcionalidad de ser parte de un **Sistema Operativo*** y el medio de almacenamiento principal de toda **aplicación*** que requiera disponer de información permanentemente. Sin embargo por ser un medio tan solicitado, que suele tener un tiempo finito de vida, por lo que es recomendable, utilizarlo solo para propósitos funcionales, y no como medio de respaldo.

2. Cinta magnética. *Las cintas magnéticas están hechas de bandas plásticas cubiertas con óxido de hierro magnetizable. Su apariencia física es similar a la de las cintas **casettes***. La cinta está divida horizontalmente en siete o nueve canales, como muestra la **Figura 1.2**, los datos están codificados a lo largo de canales o pistas carácter por carácter* [1]

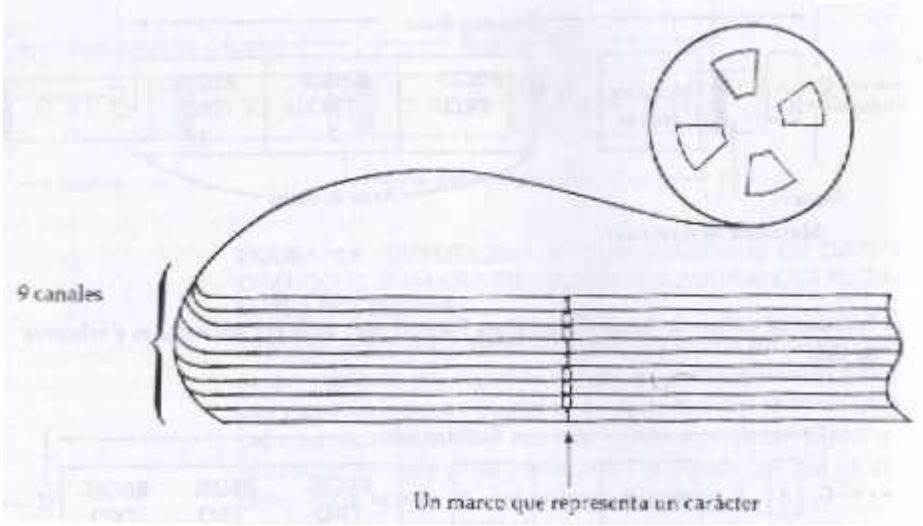


Figura 1.2

Este medio de almacenamiento, es el más lento de leer, sin embargo es uno de los dispositivos mas confiables y puede almacenar grandes cantidades de **bytes***, y actualmente, es utilizada para generar respaldos masivos de **datos***.

3. Disco compacto (CD-ROM). *Un disco compacto (Compact Disk o CD) es un disco óptico usado para el almacenamiento de datos digitales, desarrollado originalmente por digital audio.*

Un disco compacto estándar, después conocido como "CD de audio" para diferenciar este de las variantes posteriores, almacena datos de audio en un simple formato del estándar del libro rojo (nombre dado al libro de este color que contiene la especificación técnica de todos los formatos de CD y CD ROM). Un CD de audio consiste de varias pistas etéreo almacenadas usando el código 16-bit PCM (esta es una representación digital de una señal análoga donde la magnitud de la señal es definida regularmente por intervalos uniformes) a una velocidad de 44.1 kHz. La mayoría de los discos compactos tienen un diámetro de 120 mm, diseñados para definir 74 minutos de audio, y en la práctica un poco más **[U8]**.

Esta tecnología fue adaptada para otros usos, como dispositivo de almacenamiento de datos, conocido como CD-ROM.

*El **CD-ROM** (abreviatura de "Compact Disc Read-Only Memory" (ROM), o memoria de solo lectura) es un medio de almacenamiento de datos óptico no volátil, del mismo formato físico del disco compacto de audio, legible por computadoras con un dispositivo manejador de CD-ROM. Un CD-ROM es un disco de plástico plano, con información digital codificada sobre un espiral desde el centro al límite del borde exterior. La especificación técnica estándar del CD-ROM es definida en el libro amarillo, establecido en 1985 por Sony y Philips. Microsoft y Apple Computer fueron los primeros entusiastas y promotores del CD-ROM* **[U9]**.

En la **Figura 1.3**, se ejemplifica como es posible guardar digitalmente la información, en un medio de este tipo:

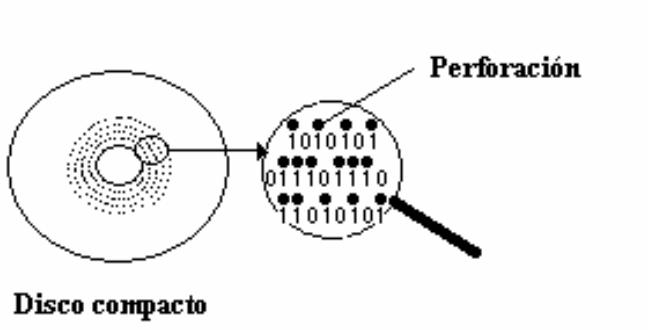


Figura 1.3

Este medio de almacenamiento, como ya se mencionó, es de solo lectura y por lo tanto de una sola escritura, por lo que es utilizado principalmente para establecer programas terminados para su distribución o respaldos históricos donde no es necesario ni debido modificar información, este puede almacenar hasta 650MB o 700MB, según el fabricante. Sin embargo gracias a la evolución de

la tecnología, actualmente existen discos compactos regrabables, conocidos como CD-RW (Compact Disk Read Write o disco compacto de lectura y escritura), en donde es posible guardar y leer la información grabada con un número limitado de escrituras en cada pista.

4. Tarjetas SIM. *Un módulo suscriptor de identidad (**subscriber identity module)** es una elegante tarjeta con seguridad que almacena la llave de identificación de un teléfono móvil. La mayoría de tarjetas SIM, son usadas en sistemas **GSM***, con módulos compatibles para teléfonos **UMTS*, UEs* (USIM*)** y **IDEN***. Esta tarjeta también contiene espacio para almacenar mensajes de texto y una libreta telefónica.*

*Desde el punto de vista técnico, el SIM, es una **microcomputadora*** que ejecuta todas las operaciones necesarias basadas en datos almacenados en esta. Este mecanismo consta de circuitos como el **CPU*, ROM*, RAM*, EEPROM*** y métodos de entrada y salida (I/O)* **[U4].**

Esta tecnología de vanguardia, es una de las tecnologías mas aceptadas para dispositivos móviles gracias a que es muy pequeña y es posible utilizara como memoria principal de algún **Sistema operativo***.

5. **Memoria Flash**. *Es un tipo de **EEPROM*** que permite que se borren o escriban múltiples localizaciones de memoria en una operación de programación. Las **EEPROM*** normales, sólo permiten que se borre o escriba una localización cada vez.*

Las aplicaciones más habituales son:

- ***El llavero USB**. Pequeño dispositivo de almacenamiento que utiliza la memoria flash para guardar la información sin necesidad de pilas. Los llaveros son impermeables a los rasguños y al polvo que han plagado las formas previas de almacenamiento portable.*
- *Las tarjetas de memoria flash que son el sustituto del carrete en la fotografía digital* **[U5]**.

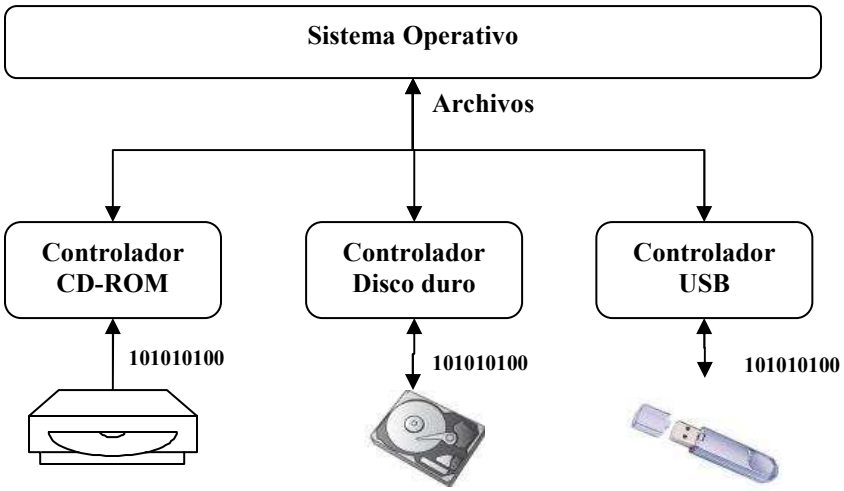
Actualmente el dispositivo mas socorrido con esta tecnología, es ya el mencionado llavero USB, ya que existe un programa (controlador) que interactúa con la memoria para que el **Sistema Operativo***, lo considere como una unidad de almacenamiento definida, como el caso del **disco duro***, haciendo las rutinas de lectura y escritura básicas.

I.III. Almacenamiento lógico de datos

Como ya se mencionó, toda esta variedad de tecnología implica un complejo y variado mecanismo para la lectura y escritura de **datos*.** Por lo que desde el nacimiento de las **computadoras*,** se ha requerido una **Interfaz*,** entre los dispositivos físicos de una computadora (**Hardware***) y los **programas***, que sea capaz de unificar el protocolo de comunicación entre estos.

Gracias a los **Sistema Operativos*** se ha logrado establecer lineamientos y estándares, precisos para la lectura y escritura de **datos*** en un dispositivos de almacenamiento. Cada **Sistema Operativo*,** ha establecido su propio **Sistema de Archivos***, para simbolizar el contenido general

de un medio de almacenamiento, mediante una interfaz, conocida como controlador o decodificador del dispositivo. Comúnmente en forma de **directorios*** y **archivos***.

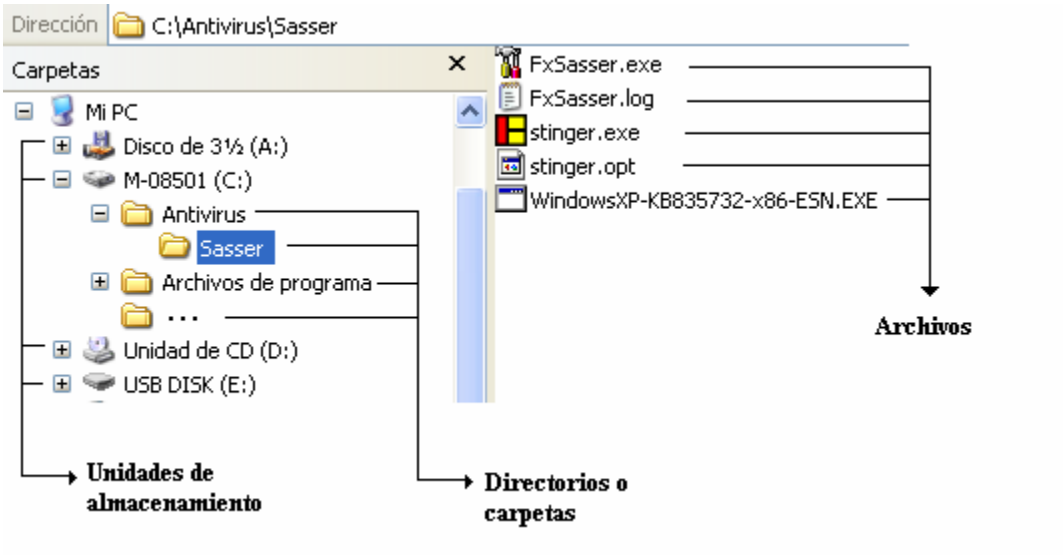


*Un **sistema de archivos*** consta de tipos de **datos*** abstractos, que son necesarios para el almacenamiento, organización jerárquica, manipulación, navegación, acceso y consulta de **datos***.*

*La mayoría de los **sistemas operativos*** poseen su propio sistema de archivos. Los sistemas de archivos son representados ya sea textual o gráficamente utilizando gestores de archivos o **shells***. En modo gráfico a menudo son utilizadas las metáforas de carpetas (directorios) conteniendo documentos, **archivos*** y otras **carpetas***. Un sistema de archivos es parte integral de un sistema operativo moderno* **[U3]**.

I.III.I. Almacenamiento en archivos

Para poder guardar la información en algún medio físico, el sistema operativo emplea una entidad lógica denominada **archivo*,** la cual es la representación de uno a serie de **bytes** que representan un carácter, dígito o símbolo. A su vez, estas entidades lógicas son ordenadas en otra entidad conocida como directorio o carpeta, asemejándolas a un archivero con los cajones llenos de carpetas, y cada carpeta con algún documento, en este ejemplo el archivero se refiere a cualquier dispositivo de almacenamiento, una carpeta a un contenedor de **archivos*** y un documento es visto como un **archivo***. Un ejemplo de ello es la representación gráfica que le da el actual sistema operativo Windows XP:



En un archivo es posible almacenar una secuencia de símbolos cualquiera, por lo que es posible plasmar cualquier tipo de información en un **archivo*,** sin embargo se han definido dos tipos de archivos importantes:

Archivo* de Texto. Utilizado para almacenar cualquier escrito, como una carta, documento, forma, etc. Almacenando una secuencia de **bytes*** en un **archivo***, en donde cada **byte*** representa un carácter o símbolo del escrito.

Archivo* Binario. Este archivo es el más socorrido por **aplicaciones*,** que buscan definir una **estructura de datos*** específica, como son imágenes, videos, secuencias de **bytes*** comprimidas, etc.

Un **byte*** es la también conocida como el octeto, ya que está formada por ocho **bits*** (Binary Digit, mínima unidad de información y puede ser un cero o un uno), y gracias al sistema numérico binario, es posible representar como máximo 256 números.

1	1	1	1	1	1	1	1

$2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 =$
$128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 256$

Actualmente los símbolos son representados por un número (byte), por el **Código ASCII*(American Standar Code for Information Interchange)** para la representación de símbolos en algún **archivos*.**

Sin embargo, Debido a que cada vez más países de diferentes lenguas se sumaron a la tecnología digital, fue necesario definir una serie de símbolos más completo que cubriera todos los símbolos conocidos, este estándar es conocido como **Unicode***, el cual utiliza dos **bytes*** para definir los diferentes símbolo, esto es 2^{16} ó 65,536 símbolos.

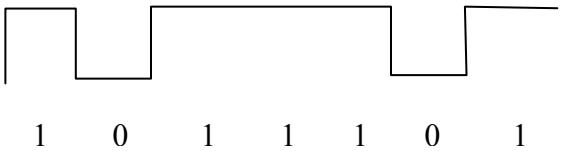
I.III.I. Protocolos de comunicación

La necesidad de guardar y enviar la información, es y ha sido siempre necesaria, como es el caso ya mencionado del almacenamiento de datos en dispositivos físicos, revolucionando así la manera de cargar, o guardar cualquier tipo de información, también se ha hablado de dispositivos diseñados para la carga de datos como es el caso del disco flexible, CD-ROM o llavero USB, sin embargo existen mecanismos para la transferencia de datos por medio de algún periférico de salida o enlace.

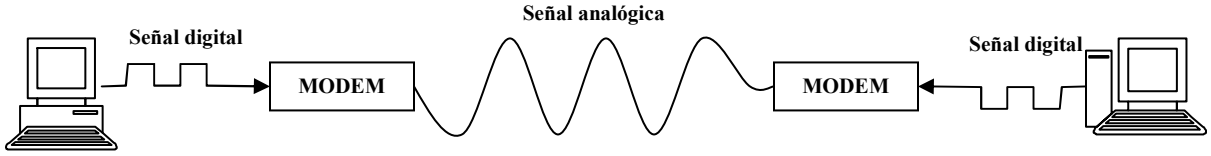
Se entiende por transmisión de datos al proceso de transporte de la información codificada de un punto a otro.

En toda transmisión de datos se ha de aceptar la información, convertirla a un formato que se pueda enviar rápidamente y de forma fiable, transmitir los datos a un determinado lugar y a su vez recibidos de forma correcta, volverlos a convertir al formato que el receptor pueda reconocer y comprender **[2].**

La forma en que se comunican las computadoras es similar a una persona a base de señas o bien, a las señales de humo utilizadas desde hace mucho tiempo para establecer comunicación entre personas de un logar a otra. La comunicación interna en una computadora, al igual que muchos dispositivos de almacenamiento, es digital, es decir se basa en la idea de que para representar un **bit*** (un cero o uno), utiliza la espera de carga en un tiempo determinado, es decir si en n tiempo de espera llega una cierta carga, indica o representa un 1, pero si en ese tiempo de espera no se tiene indicio de carga, entonces se está representando un cero.



El MODEM, es un equipo que convierte las señales digitales de la computadora a señales analógicas y por medio de una línea telefónica, es enviada a otra computadora y cuando esta las recibe, las vuelve a convertir de analógicas a digitales:



Este tipo de comunicación, es utilizad para la transferencia de datos en puntos distantes, ya que el proceso de codificar y decodificar las señales produce un costo económico y de rendimiento, por lo que se han optado otros medios para la transferencia de datos mas económicos para un área local,

es decir n computadoras que interactúan en una misma localidad, como en un edificio o habitación. A continuación, se dan a conocer algunos dispositivos de corto alcance que utilizan señal digital para establecer una comunicación cercana y económica.

*Los **multiplexores** son equipos que permiten mantener más de una conexión simultánea por una sola línea. Cada una de las comunicaciones opera como su tuviera la línea de forma exclusiva pudiendo utilizar diferentes velocidades y **protocolos*** en cada una de ellas.*

*Los **concentradores** (HUBS) son equipos que permiten compartir el uso de una línea entre varios **ordenadores***. Todos los **ordenadores*** conectados a los concentradores pueden usar la línea, aunque no de forma simultánea, ni utilizando distintos **protocolos***, ni distintas velocidades de transmisión.*

*Los **conmutadores** (SWTCHES) se caracterizan por no enviar los paquetes a todos los puertos, sino únicamente al puerto correspondiente al destinatario. La diferencia entre un conmutador y un puente (bridge) es que el puente debe recibir todo el paquete antes de dirigirlo al puerto correspondiente y un conmutador dirige el paquete una vez recibido el encabezado (en ella se encuentra la dirección IP del destinatario)* **[2].**

Estos son ejemplos de algunos dispositivos especializados para enlazar varios equipos de cómputo logrando una comunicación bidireccional, estableciendo una **red*** de computadoras, ya sea de forma análoga o digital.

Para lograr una comunicación entre computadoras o de una computadora a algún dispositivo o periférico, al igual que las personas, es necesario establecer una serie de normas que regulen la comunicación entre dos dispositivos o computadoras. En el contexto de la red de computadoras se ha establecido el término **protocolo*,** como aquellas reglas que hacen posible el intercambio fiable de comunicación entre dos equipos informáticos **[2].**

Actualmente los protocolos, en el entorno de redes computacionales, tienen la función de controlar la comunicación entre dos computadoras, la norma más destacada es el modelo OSI (Open Systems Interconnection), propuesta por la Organización Nacional de Normalización (ISO). Estas en tareas pueden reducirse a:

- ***Establecimiento de la comunicación.*** *En esta fase se establece la conexión física entre los ordenadores y se establece el procedimiento empleado para el intercambio de la información.*
- ***Transferencia de la información.*** *Ambos sistemas intercambian datos a través del enlace establecido. En caso de producirse un error en la recepción de datos, se detecta y se solicita su reenvío.*
- ***Terminación.*** *En esta fase se da por finalizada la comunicación* **[2].**

Entre los más conocidos y utilizados son:

IPX/SPX

Los protocolos de comunicación y transporte IPX/SPX fueron desarrollados por Novell a principios de los años ochenta inspirándose en los protocolos del Sistema de Red de Xerox (XNS).

Sirven como interfaz entre el sistema operativo de red NetWare y las distintas arquitecturas de red **[2].**

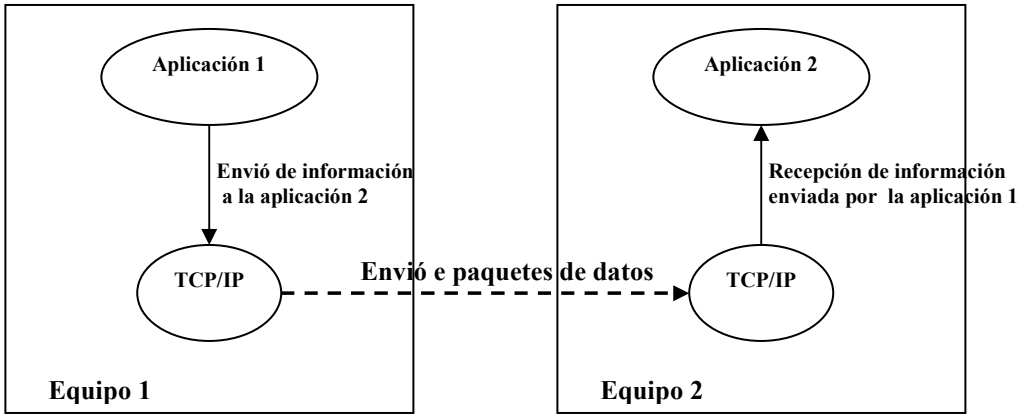
TCP/IP

El nombre de TCP/IP proviene de dos de los protocolos mas importantes de la familia de protocolos ***Internet****, el Transmisión Control Protocol (TCP) y el Internet Protocol (IP).*

La principal virtud de TCP/IP estriba en que está diseñado para enlazar ordenadores de diferentes tipos, incluyendo PCs, Minis y ***mainframes****, que ejecutan sistemas operativos distintos, sobre redes de área local y redes de área extensa y, por tanto permite la conexión de equipos distantes geográficamente.*

La arquitectura TCP/IP transfiere datos mediante el ensamblaje de datos en paquetes, cada paquete empieza con una cabecera que contiene información de control seguida de los datos **[2].**

Finalmente, existen aplicaciones que trabajan directamente con estos protocolos, estableciendo un nuevo protocolo a nivel aplicación, como se muestra en el siguiente ejemplo:



En este ejemplo, el proceso TCP/IP, establecido en el quipo 1, nos representa aquella entidad que establece la comunicación con el equipo 2, empaqueta y transfiere la información otorgada por la aplicación 1 hacia la red. Del lado del equipo 2, el proceso TCP/IP descifra los paquetes de datos recibidos de la red y envía la información contenida a la aplicación solicitante en este caso a la aplicación2. Entiéndase que al estar enviado información de una aplicación a otra, es necesario establecer otro protocolo entre aplicaciones.

Los protocolos de computadoras se asemejan al lenguaje de las personas, un ejemplo burdo de estos es el siguiente dialogo:

Equipo 1 - *¿Quién eres?*
Equipo 2 **–** *Juan*
Equipo 1 - *¿Cuál es tu contraseña?*
Equipo 2 **–** *perez*
Equipo 1 – *¡Que tal!, ¿Qué deseas?*
Equipo 2 **–** *Necesito el archivo guardado en mi carpeta*
Equipo 1 – *Enviando: …*
Equipo 2 – *Ok, Hsata luego*
Equipo 1 – *Hasta luego*

Actualmente existen aplicaciones de **software*** que facilitan la comunicación entre computadoras de forma grafica, sin embargo el protocolo es exactamente el mismo, por ejemplo el programa FTP, más que un programa de computadora, es un **protocolo*** específicamente para enviar y recibir archivos de una computadora a otra, similar al ejemplo anterior. Esto es que existen programas que utilizan este **protocolo*** o reglas de comunicación para compartir información.

I.III.III. Aplicaciones de Software que requieren la lectura y escritura de datos

Dados los dispositivos de almacenamiento y los medios de transferencia de datos, es necesario un **software*** para manipular y definir cada una de los **datos*** manejados. Cuando nos referimos a la palabra **software***, nos referimos a conceptos ambiguos, por lo que para su ubicación y definición del mismo, se emplearán los siguientes conceptos.

El **Software*(**parte blande de un dispositivo o computadora**)** físicamente es un bloque de código enviado al **procesador*** para obtener un resultado. Este bloque reside en la memoria volátil conocida también como Memoria **RAM***.

Memorias de acceso aleatorio. *Acrónimo de Random Access Memory, (Memoria de Acceso Aleatorio) es donde el ordenador guarda los datos que está utilizando en el momento presente. Se llama de acceso aleatorio porque el procesador accede a la información que está en la memoria en cualquier punto sin tener que acceder a la información anterior y posterior. Es la memoria que se actualiza constantemente mientras el* ***ordenador**** *está en uso y que pierde sus datos cuando el* ***ordenador**** *se apaga.*

Cuando las ***aplicaciones**** *se* ***ejecutan***, primeramente deben ser cargadas en memoria RAM. El procesador entonces efectúa accesos a dicha memoria para cargar instrucciones y enviar o recoger* ***datos**** **[2].**

La RAM o memoria volátil, es utilizada principalmente para la ejecución del **software*** y mantener datos en una localidad de fácil acceso temporalmente, por lo que se recurre en muchas ocasiones, a vaciar **datos*** almacenados en algún dispositivo físico a esta memoria. Una vez vaciada la información a la memoria RAM, es posible acceder a ella fácilmente por medio de algún programa de computadora (**software***), y manipularla como se requiera.

El **software*** consta de un segmento de datos y otro de código, conocido también como **proceso*** o **programa***. Para usos prácticos, definiremos a un **proceso*,** como el **software*** elemental de una computadora, el **programa*** es la coordinación de **procesos*** para un fin específico, y finalmente una **aplicación*** como la coordinación de **procesos*** y **programas*** para producir un mecanismo o sistema robusto y completo de información.

Finalmente existen las aplicaciones conocidas como **Sistemas Operativos*** que fungen como administrador y coordinador de todo **proceso***, **programa*** o **aplicación*** que se ejecuta o procesa en una **computadora***, como se muestra en la siguiente **Figura 1.4**.

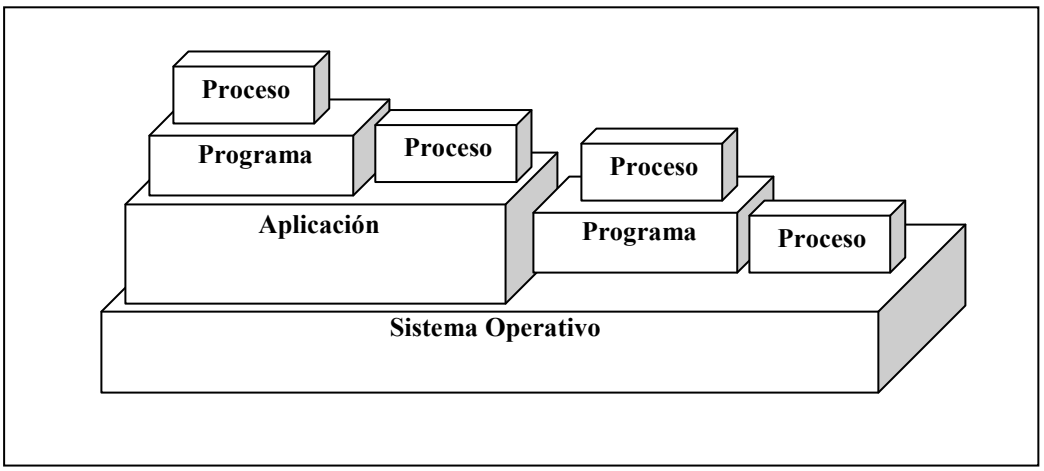


Figura 1.4

A continuación se ejemplifican algunas de las formas de almacenar la información de algunos aplicativos de **software***.

a Aplicaciones que generan información

Hoy en día la variedad de **aplicaciones*** es inmensa, debido a la necesidad de generar o tener un **archivo*,** que respalde todo tipo de trabajo o esfuerzo, un ejemplo de esto es un editor de imágenes que requiere guardar la descripción exacta de una imagen, un editor de texto que guarda un escrito, o bien, un juego de video que registro el seguimiento de un jugador, como el nivel de dificultad, preferencias, puntuación final, etc. Para lograr esta tarea, es necesario definir cualquier **dato*** en forma de **bytes***, para finalmente almacenarlo en un dispositivo de almacenamiento permanentemente y ser recordado en otra sesión de trabajo. Comúnmente, toda esta información, es almacenada en la entidad lógica conocida como **archivo*** antes mencionada.

Se ha observado que la mayoría de las **aplicaciones*** que generan **archivos***, tienen un comportamiento similar en cuanto a el manejo en la memoria volátil y memoria permanente. Por lo que a continuación se ejemplifica el uso de estas memorias con funcionamiento básico de un **software*** denominado editor de texto.

Editores de texto. Es una **aplicación*** que almacena en un **archivo*** la representación digital de un escrito realizado por el usuario de una **computadora***, así como los **datos*** necesarios para definir medidas, formatos y estilos del mismo. Este programa normalmente recibe la información otorgada por el usuario y es vaciada a alguna **estructura de datos*** en la Memoria volátil (**RAM***) en donde es posible manipular la información y finalmente vaciarla hacia un **archivos***. Los procesos básicos de esta aplicación, son:

1. **Carga del aplicativo**. Al iniciar un aplicativo de este tipo, se inicializa en la memoria volátil, una **estructura de datos*** utilizada para la manipulación directa de los datos recibidos por el usuario. En caso de que se le haya especificado la localidad de un **archivo*,** y esta cumpla con el formato establecido, se vacía el contenido de este a la **estructura de datos*** y finalmente es mostrado el contenido de la memoria a pantalla para que el usuario, inicie o continúe el escrito. Este proceso es ilustrado en la **Figura 1.5**.

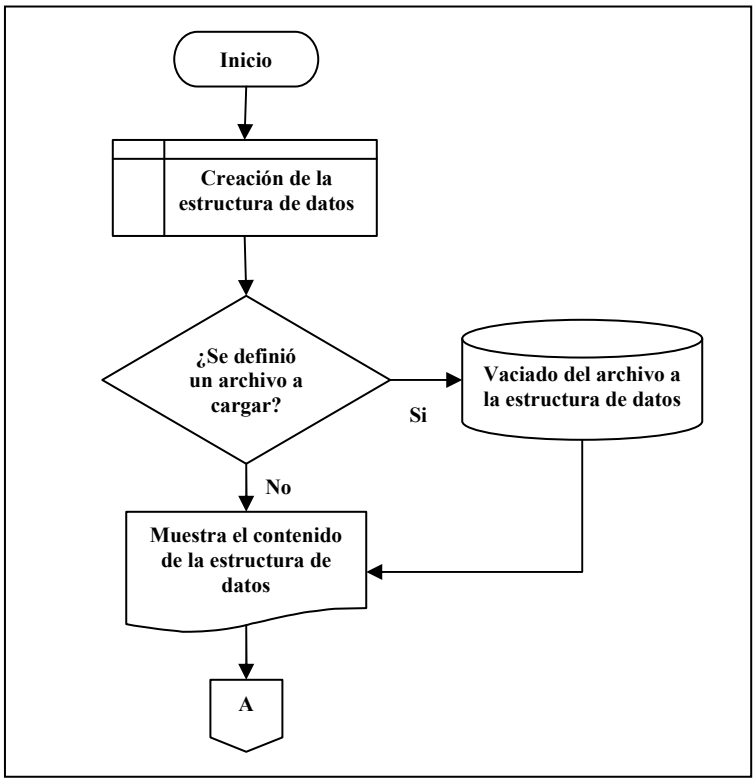


Figura 1.5

2. **Captura de datos.** El proceso de captura se realiza mediante la espera de uno o varios **bytes***, enviado por algún **periférico*** de entrada, como es el caso del **teclado***. Posteriormente se actualiza la **estructura de datos*** y nuevamente se muestra el contenido de la estructura para hacer visible el cambio realizado. El conjunto de **byte*** o **bytes*** recibidos puede ser un símbolo a agregar en el escrito, o bien, puede ser un comando dirigido al editor de texto, la instrucción para eliminar algún carácter, imprimir el documento, cambiar de posición dentro de la pantalla, etc. Vea la **Figura 1.6.**

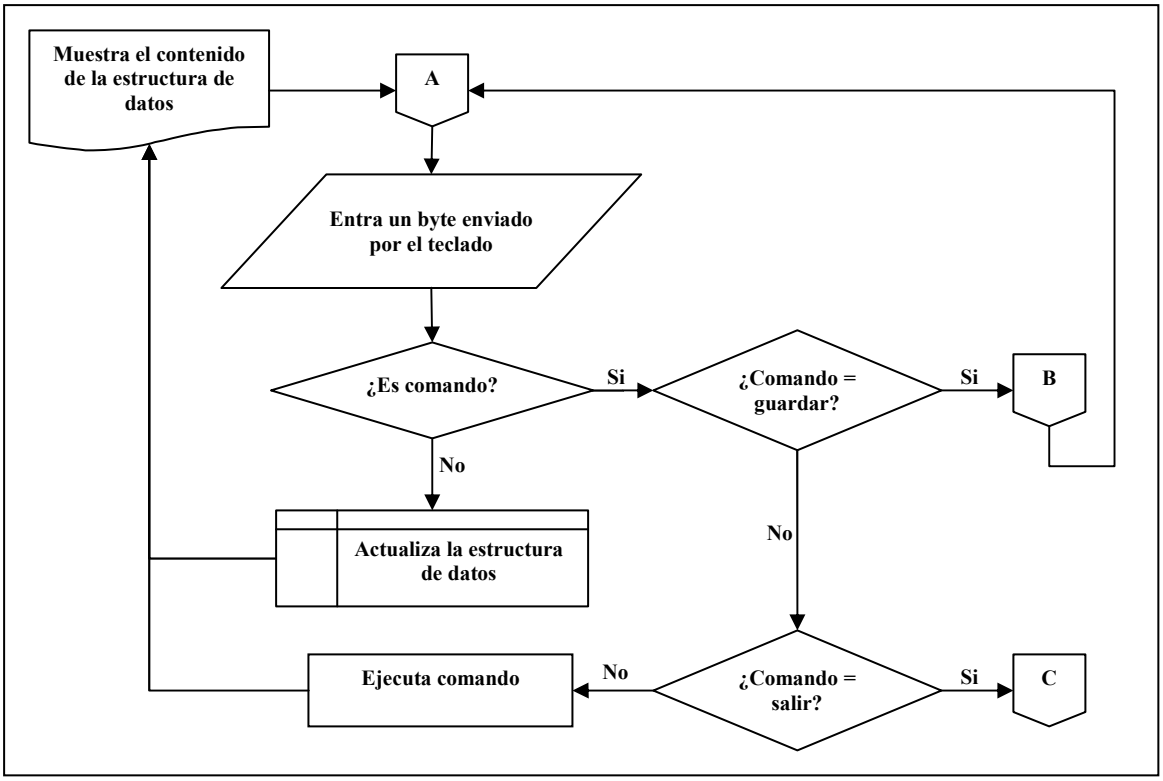


Figura 1.6

3. **Guardado del escrito.** La función de guardar es la acción de guardar el contenido de la **estructura de datos*** en un **archivo***, comúnmente de forma diferente a la definida de esta, ya que la información en memoria volátil, se maneja de manera conveniente para su manipulación, y en un archivo de forma más fácil de leer. Cuando el documento es recientemente creado, comúnmente no existe el archivo, por lo que es necesario crearlo. En caso de que se haya cargado un archivo existente, simplemente se sobrescribe, pero si no se cargo el archivo, y el archivo existe, generalmente se corrobora si se desea realmente sobrescribir, como se muestra en la **Figura 1.7**.

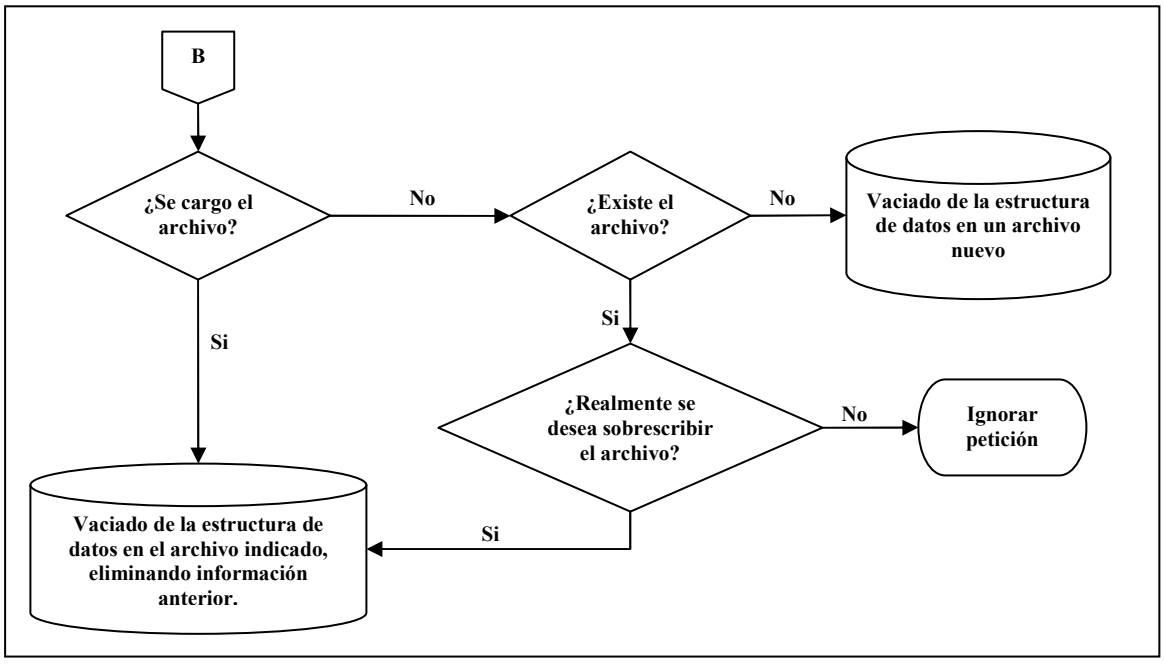


Figura 1.7

4. **Terminar la aplicación.** Comúnmente, en los editores de texto se recomienda guardar el escrito antes de salir de la aplicación y no perder el trabajo realizado, ya que la **estructura de datos*** donde se trabaja, reside en la memoria volátil y en cuanto se termina la aplicación, o se apaga el equipo, la estructura es borrada inmediatamente por el **Sistema operativo*** **(**administrador de memoria) y por consecuencia es ignorada la información capturada. Por tal motivo, el editor verifica si existen cambios realizados en la estructura de datos antes de perderla, y en caso de ser así realiza la pregunta correspondiente para determinar si se guardarán los datos o no, como ser muestra en la **Figura 1.8**.

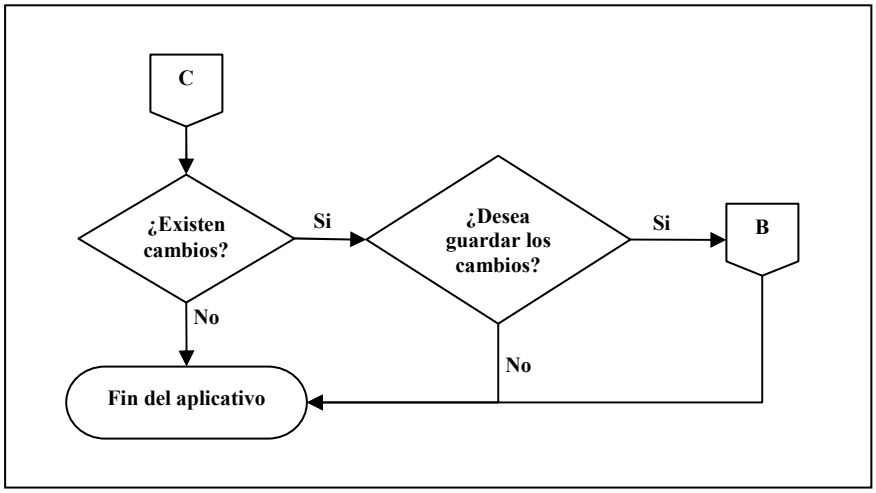


Figura 1.8

Es importante recalcar, para fines de este modelo, que en la mayoría de las **aplicaciones*** de **software*** que generan algún archivo**,** trabajan con dos entidades importantes: la **estructura de datos** y el **archivo*.** Dando pauta a una serie de métodos totalmente diferentes de almacenar y leer los **datos*** utilizados en cada **aplicación***, y por consecuente, la redundancia en estos métodos en cada **aplicación*** de este tipo.

b Aplicaciones Cliente Servidor.

*Con el paso de los años, los usuarios de los **ordenadores***, fueron necesitando acceder a mayor cantidad de información y de forma más rápida, por lo que fue surgiendo la necesidad de un nuevo **ordenador***: el servidor.*

*Un servidor (del inglés SERVER), es un **ordenador*** que permite compartir sus periféricos a otros **ordenadores*** **[2].**

Se sugiere que una computadora de tipo Servidor sea una **supercomputadora*** ya que, como se menciona en el párrafo anterior, su función es principalmente compartir tanto sus recursos como periféricos entre varios equipos de menor capacidad o rendimiento por medio de un enlace de **red*** y un **protocolo*** específico. Algunos ejemplos mencionados en la bibliografía Redes Locales **[2]** son:

***Servidor de Archivos**. Mantiene los archivos en subdirectorios privados y compartidos para los usuarios de red.*

***Servidor de Impresión.** Tiene conectadas una o varias impresoras que comparte con los demás usuarios.*

Servidor de Comunicación.** Permite enlazar diferentes **redes locales o un **red*** local con grandes **ordenadores*** o mini ordenadores.*

***Servidor de correo electrónico.** Proporciona servicio de correo electrónico para la red.*

Servidor Web.** Proporciona un lugar para guardar y administrar los documentos HTML que puedan ser accesibles por los usuarios de la **red a través de los navegadores.*

***Servidor FTP.** Se utiliza para guardar los archivos que puedan ser descargados por los usuarios de la red.*

Servidor Proxy.** Se utiliza para monitorizar el acceso entre las redes. Cambia la IP de los paquetes de los usuarios para ocultar los datos de la red interna a **Internet, y cuando recibe contestación externa, la devuelve al usuario que la ha solicitado. Su uso reduce la amenaza de piratas que visualicen el tráfico de la red para conseguir información sobre los ordenadores de la red interna.*

Esto es posible gracias a la ayuda de programas especializados que residen en el equipo servidor, conocidos también como: *"servicios locales"* y *"Servicios de red"*.

Servicios locales. Programas administran, codificar o decodificar algún recurso como uno o varios archivos, dispositivo o periférico de alguna computadora. Para el caso en que un servicio necesite la comunicación a periféricos o dispositivos normalmente se realiza mediante un **protocolo*** que solo lo entiende el dispositivo o periférico y el este servicio.

Un servicio local es utilizado solo por aplicaciones, programas locales o bien por el propio sistema operativo. Un ejemplo de este es el programa que administra la conexión a una **red***, donde su función es proporcionar un enlace de comunicación (tarjeta de red) a los demás programas que residen en el mismo equipo, como se ilustra en la **Figura 1.9.**

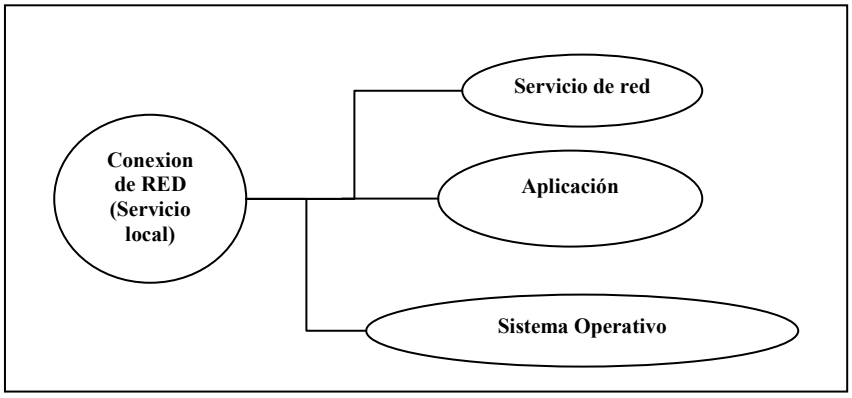


Figura 1.9

Servicios de red. Es aquel programa que con ayuda de uno varios programas (servicios) locales explota o comparte los recursos del equipo en el que reside y son transferidos, por medio de un enlace de comunicación, a equipos de cómputo externos que los soliciten.

Por ejemplo, el Servicio utilizado para un Servidor FTP utiliza un canal de comunicación (con ayuda de un **protocolo*** de comunicación de bajo nivel como el TCP/IP, IPX, etc.) sobre una conexión de **red*** para atender a aquellas aplicaciones externas que soliciten la descarga o carga de archivos, además de apoyarse del **Sistema Operativo*** (en este caso nuestro servicio local) para acceder a los archivos almacenados en el equipo local, como se muestra en la **Figura 1.9.**

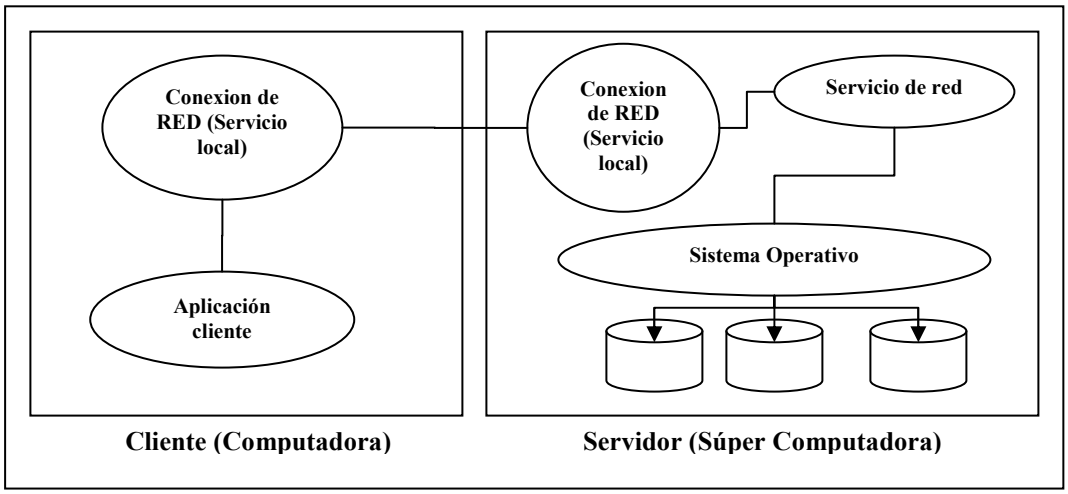


Figura 1.9

Este modelo se caracteriza en que la aplicación cliente reside en otro equipo (estación de trabajo), el cual, normalmente es de menor rendimiento y se enfoca a la explotación del servidor, sin embargo, es posible que la aplicación cliente resida en el servidor, lo cual no es recomendable, ya que si algún programa provoca un error en el equipo, puede afectar gravemente a los servicios locales y de red del mismo, perdiendo gran cantidad de recursos o información.

Un equipo de cómputo tipo Cliente, es aquel que por medio de un enlace de red a un equipo de tipo Servidor, explota los recursos otorgados por este. Un Cliente puede ser otro servidor o bien una estación de trabajo (PC, Teléfono móvil, Palm, Laptop, etc.). Este cuenta con un proceso básico de comunicación que solo sirve para solicitar un recurso, a diferencia del servicio de red Servidor, este no es permanente, ya que solo se crea en una sesión de trabajo (se conecta al servidor), explota los recursos disponibles y termina el enlace, a diferencia del servicio de red Servidor que espera permanentemente las solicitudes que lleguen a presentarse, teniendo un ciclo de vida permanente.

Existen **sistemas Operativos*** especializado para las tareas de un servidor, que permiten una mejor ejecución de procesos de servicio, ya sean de red o locales, dando como resultado un alto índice de respuesta a quien solicita (Clientes), a diferencia de un **Sistema Operativo*** cliente que se enfoca al rendimiento de una aplicación cliente que solo solicita o explota los recursos de un servidor.

c Sistemas de Bases de datos

Una **aplicación*** que genera un archivo es útil y eficiente cuando se requiere el manejo de información homogénea, finita y moderada, sin embargo cuando se requiere almacenar, recuperar o actualizar un conjunto de **datos*** masivos e incluso heterogéneos como es una imagen, un escrito, un número, palabra o cualquier otro tipo de **dato***, la aplicación requiere de otro programa o aplicación, que administre y controle tanto las peticiones necesarias, como los **datos*** a almacenar.

Esto es posible a la utilización de **aplicaciones*** especializadas en el manejo y recuperación de **datos***, como son los **Sistemas de bases de datos**.

Un sistema de bases de datos es un sistema computarizado de información para el manejo de datos por medio de paquetes de software, llamados Sistemas de Manejo de Bases de Datos (o en sus siglas en inglés DBMS). Los tres componentes principales de un Sistema de Bases de Datos, son el ***Hardware**** *el* ***Software**** *DBMS y los datos por manejar* [1].

Dicho en otras palabras, es aquel intérprete de los datos almacenados en algún dispositivo físico (**hardware***) comúnmente por medio de un **programa*** de **computadora*** (**software***). Generando así el concepto de base de datos

Base de Datos: *Colección de archivos interrelacionados creados bajo los criterios de un Sistema Manejador de Bases de datos. El contenido de una base de datos se obtiene combinando datos de todas las fuentes de una organización* [1].

Actualmente, la mayoría de Sistemas Manejadores de Bases de datos, han optado por ser una **aplicación*** despachadora de datos (o servidor de datos) bajo el esquema Cliente Servidor antes discutido, este cumple básicamente con las siguientes tareas:

- **Manejar entidades relacionadas.** Dentro de las bases de datos es posible almacenar un **datos*** heterogéneos y relacionarlos gracias a las reglas de normalización, dando origen a los Sistemas Manejadores de Bases de Datos Relacionales (o por sus siglas en inglés RDBMS). Las tres primeras reglas de normalización, fueron perfiladas por el Dr. E.F.Codd en su escrito de 1972, "Further Normalization of the Data Base Relational Model" (Referente a la normalización de las Bases de Datos Relacionales). Las siguientes reglas definidas, han sido teorizadas por matemáticos/Algebristas posteriores.
- **Garantizar la Integridad de la información.** Los sistemas Manejadores de Bases de Datos, se han dado a la tarea de proteger los datos lo mejor posible, por lo que se han impuesto restricciones dentro de las base de datos para evitar cualquier tipo de ambigüedad, como son los candados, que evitan duplicidad de información, impedir que se genere información que no coincida con alguna información ya existente, etc.
- **Controlar y administrar la concurrencia de peticiones.** Actualmente no solo se esperan las peticiones de una sola **aplicación***, sino que es posible recibir peticiones de n **aplicaciones***, **programas*** o **procesos*,** que solicitan alguna petición simultáneamente y es necesario tener alguna entidad que administre estas peticiones debido a que existen recursos dentro de una **computadora*** que no es posible manipular por dos o mas procesos en un mismo tiempo.
- **Mantener la seguridad en todo momento.** Los esquemas de seguridad, varían según el Sistema Manejador de base de datos, sin embargo la mayoría coincide en que lo primordial es definir lineamientos entre usuarios y recursos, comúnmente determinados por el usuario que administra los **datos*** (conocido también como Administrador de Bases de datos). Para poder identificar un usuario dentro del sistema, se requiere acceder mediante un alias (o nombre de usuario) y una contraseña, y una vez que ha sido identificado, es posible determinar los privilegios y alcances que tiene hacia los recursos.

Otro atributo por el cual están inclinándose estos sistemas es el trabajar con los llamados "proceso almacenados", lo cual no es más que definir un proceso dentro de la base de datos para el trabajo con los datos del lado del servidor, agilizando los procesos de búsqueda, validación o de transacción.

Actualmente la forma de trabajo de los Sistemas Manejadores de Bases de Datos (SMBD), se consideran una arquitectura Cliente Servidor, donde el Servidor, es un Sistema Manejador de Base de Datos visto como una aplicación de **software*** establecido en equipos de cómputo como la **microcomputadora*** o una **supercomputadora*** (Recomendable), y el cliente es otro aplicativo de **software***, en donde comúnmente se implementa en una arquitectura sencilla como es el caso de la **microcomputadora*** o en otras ocasiones en el mismo equipo donde reside el Servidor. El servidor de Datos (SMBD), como todo software, reside en Memoria **RAM*** ejerciendo la labor de verificar, cargar, leer y administrar los recursos que residen en el equipo Servidor, además de estar atento a cualquier petición de alguna aplicación cliente que conozca el **protocolo*** de comunicación y sea capaz de identificarse con un nombre de usuario y contraseña existentes en el Servidor, una vez establecido el enlace remoto o local (en caso de que la aplicación cliente resida el mismo equipo que el Servidor), se genera un **proceso*** llamado Instancia, el cual es controlado por el servidor de datos, esto es, que cuando se genera un enlace de conexión se crea una instancia en el servidor para atender a cada cliente particularmente. En la **Figura 1.8** se ilustra mediante un ejemplo, el funcionamiento básico de un Servidor de datos.

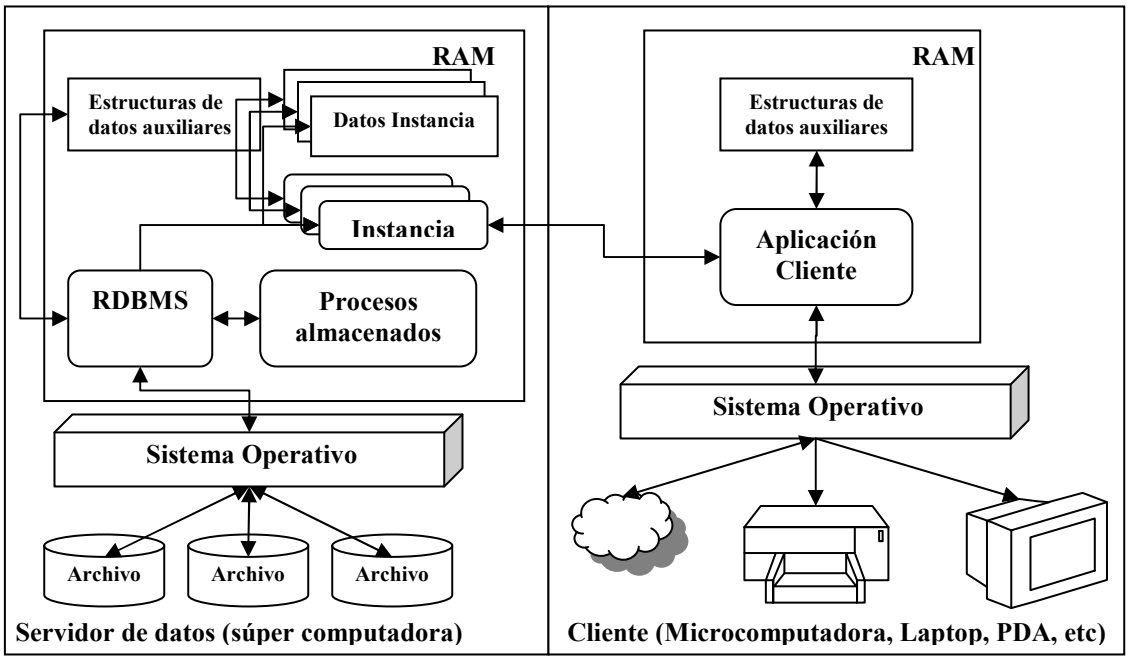


Figura 1.8

Nota. Este ejemplo se asemeja a los términos definidos por el RDBMS llamado Oracle, sin embargo las arquitecturas, nomenclaturas y términos difieren según el Proveedor del SMBD, llámese SQL Server, Internase, Postgress, Informix, etc.

Algo importante que destacar de esta ilustración, es que al igual que las aplicaciones que generan archivos, las aplicaciones tanto cliente, como servidor, también hace uso de estructuras de datos para manipular los **datos*,** en un espacio de la memoria RAM. También se observa que el almacenamiento de datos en dispositivos físicos de forma es similar, ya sea de forma directa por el Servidor o indirecta por el cliente, se almacenan en entidades lógicas (**archivos***).

Es importante mencionar, que el trabajo de una arquitectura Cliente – Servidor, se rige bajo ciertas normas de seguridad, nomenclatura y protocolo, como se detallo en puntos anteriores.

I.III.IV. Representación lógica de datos en un medio electrónico

La representación de datos en un medio electrónico puede ser tan compleja como se requiera. Un ejemplo de esto es la representación de datos sugerida en el libro de Sistemas de Bases de Datos y Uso **[1]**

*En los sistemas comunes de almacenamiento, los elementos de datos (**campos***), se guardan en campos de longitud fija y con un orden predeterminada, de tal forma que el programador puede conocer el contenido de cada campo al referirse al número de carácter predeterminado del **registro**.*

En esta bibliografía, se definen cuatro formas diferentes para la representación de **campos***:

Posicional. *Los datos de un **registro** se almacenan en campos consecutivos de longitud fija en un orden predeterminado. Esta disposición es sencilla de realizar y conveniente de recuperar la información, pero tiene la desventaja de que desperdicia espacio si la información guardada no utiliza todo el espacio definido, quedando vacío.*

Campo:	**EMP-NO**	**EMP_NOMBRE**	**NACIONALIDAD**	**DEPT**
Valor:	105¢¢	J.¢COLE¢¢¢¢¢¢¢	CANADIAN¢¢	ACCNT¢¢

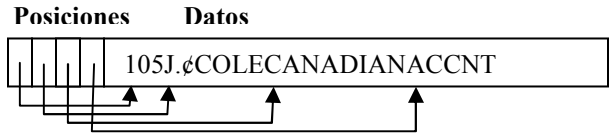
Donde ¢ determina un espacio en la cadena

Relacional. *Esta técnica elimina los espacios desperdiciados mediante el uso de un delimitador que indica el final de un campo. El delimitador debe ser un carácter especial que no se usará en ningún otro logar de la base de datos.*

Campo:	**EMP-NO@EMP_NOMBRE@NACIONALIDAD@DEPT**
Valor:	105@ J.¢COLE@CANADIAN@ACCNT

Donde ¢ determina un espacio en la cadena y @ es el carácter delimitador.

Indexada. *Se utilizan aquí señaladotes para indicar el principio de cada campo en un registro almacenado. El señalador puede ser una dirección absoluta o relativa.*



Donde ¢ determina un espacio en la cadena.

Etiquetada. *Cuando un archivo tiene muchos valores por omisión, se puede usar la técnica etiquetada para almacenar únicamente los valores diferentes a los omitidos.*

Posiciones **Datos**

N105¢¢**M**J.¢COLE¢¢¢¢¢¢¢**D**ACCNT

N: EMP-NO
M: EMP-NOMBRE
C: NACIONALIDAD (No se muestra debido a que el valor por ausencia es CANADIAN)
D: DEPT
Donde ¢ determina un espacio en la cadena

Estas formas de representar los datos es muy recurridas actualmente por la mayoría de los **programas*** para el registro (en un archivo) o su transferencia de información (por algún enlace de conexión o periférico de salida) en una secuencia continua de **bytes***, en donde cada **byte***, representa una estructura de datos, uno o varios caracteres, número o símbolo. Esta cadena en memoria volátil es conocido como Buffer, en dispositivos de almacenamiento como **archivo*** y en transferencia de redes como paquete.

Las formas de almacenamiento, ya mencionadas, tienen la ventaja, o desventaja, de ser leídas por aplicaciones básicas como es el caso del editor de Texto, recuperando la información enviada o registrada de forma fácil y comprensible al ojo humano, pero no siempre se desea que todo individuo acceda a cierta información, por lo que a continuación se hace detalla de la importancia y formas de su protección.

I.IV. Seguridad y protección de la información

En la actualidad la información es almacenada y transmitida por medios electrónicos, dando pauta a que individuos maliciosos puedan extraer información de manera directa de algún medio de almacenamiento como es el caso de un **archivo*** o un enlace de red. Por lo que se han optado varias medidas y medios para la protección de la información, como son:

Autentificación del usuario. *La autentificación y el cifrado se facilitan mediante pares matemáticamente relacionados de códigos digitales o claves. Una de las claves de cada par se divulga públicamente, más que la otra se mantiene en estricto secreto.*

A todos los transmisores de datos, ya se trate de personas, un programa de ***software**** *u otra entidad, se le distribuye un par de claves mediante un sistema de criptografía de clave pública* **[2].**

Un ejemplo (antes mencionado) es el Sistema Manejador de Base de Datos, que identifica a la entidad o persona para determinar el alcance de los datos que se le ha definido en los recursos disponibles.

Protección de directorios y archivos. *Cuando un usuario crea un directorio, un archivo o un objeto, se convierte automáticamente en su propietario (también durante el proceso de instalación se adjuntaron propietarios a todos los directorios y archivos que se crearon).*

Un propietario puede asignar permisos a sus directorios, archivos u objetos aunque no puede transferir su propiedad a otros usuarios **[2]**.

Comúnmente esta tarea es realizada por los **Sistemas Operativos***, en donde existe un súper usuario capaz de asignar propiedad a cada usuario definido en el **Sistema Operativo***, de los **archivos*** u objetos existentes y el privilegio que les corresponde, o bien, cada usuarios sobre sus propios **archivos*** u objetos.

La encriptación de datos. Desde los primeros tiempos, el hombre ha tenido la necesidad de plasmar un pensamiento, historia o algún otro tipo de información en algún medio para transmitirlo, guardarlo o bien tenerlo al alcance para recordarlo. Sin embargo por recelo, seguridad o por otras circunstancias, se ha recurrido a plasmar la información en símbolos o textos incomprensibles a entidades ajenas, dando origen a la criptología: *Del griego kryptos (ocultar) y grafos (escribir), literalmente escritura oculta, la* ***criptografía*** *es el arte o ciencia de cifrar y descifrar información utilizando técnicas matemáticas que hagan posible el intercambio de mensajes de manera que sólo puedan ser leídos por las personas a quienes van dirigidos* **[U6]**.

Un ejemplo de este, se retoma de la Edad Media, cuando los gobiernos se comunicaban con sus embajadores por medio de mensajes cifrados. Otro ejemplo es el del año 1453, cuando el gobierno Italiano establece un grupo dedicado exclusivamente al estudio de la criptografía, con el fin de perfeccionar los métodos de encriptación de sus mensajes, así como para descifrar los de sus enemigos **[U7]**.

Actualmente en los sistemas electrónicos es posible cifrar un bloque de datos (Buffer) por un proceso de cifrado con el fin de ser descifrado por otro proceso que solo el pueda leer. Este bloque de datos cifrado puede ser almacenado en un archivo, manipularlo en memoria volátil, o bien ser trasferido por un enlace de conexión a otra computadora o medio.

Auditorias. *Permiten supervisar los sucesos relacionados con la seguridad de la información, como son los accesos a archivos y carpetas, administración de cuentas de usuarios y grupos, el inicio y finalización de sesión de los usuarios, el uso de servicios tanto locales, etc*. **[2].**

Gracias a las auditorias, es posible recrear sucesos desconocidos o por lo menos tener la bitácora de los usuarios que laboraron sobre recursos o privilegios de un Sistema de información. Por ejemplo, cuando existe un error inesperado, causante de baja del sistema o corrupción de datos, es posible determinar, gracias a las bitácoras de los Sistemas operativos, que aplicaciones se ejecutaban

en el momento y determinar cual de estas es sospechosa del problema, incluyendo usuario(s) que pueden hacer uso de esta.

La seguridad, ya sea a nivel de red, archivo, aplicación o de Sistema de información, se ha vuelto una herramienta necesaria e indispensable para el envío y transmisión de datos, por lo que el modelo presentado en el capítulo 3, considera la posibilidad de codificar un bloque de datos mediante algún método de codificación y decodificación.

Capítulo II

Aplicaciones tipo cliente servidor

II. Aplicaciones tipo Cliente Servidor

Antes de entrar completamente a los conceptos de cliente servidor, es necesario dar a conocer otros tipos de aplicaciones para finalmente tener un punto comparativo entre los diferentes tipos de aplicaciones mas conocidos.

Como se detalló en el Capítulo anterior, la forma usual para el envió y almacenamiento de datos es el cargar un bloque continuo de bytes a una estructura de datos situados en la memoria volátil (**RAM***) para ser accedida y manipulada rápidamente por algún proceso. Sin embargo, cuando existe más de una aplicación con objetivos similares, se observan ciertas semejanzas que pueden convertirse en deficiencias en su desarrollo:

Aplicaciones que generan archivos. Existen herramientas de software que se han empleado para almacenar información en un medio electrónico, como en el caso de los editores de texto, imágenes o video, cuando estas herramientas guardan datos en un archivo, utilizando un solo método de escritura en el mejor de los casos. Sin embargo cuando esta misma información es extraída y enviada a la memoria volátil en una **estructura de datos*** para su fácil acceso. Aunque, en la mayoría de los casos, esta estructura es única y por consiguiente el método de manipulación, navegación, lectura y escritura de la estructura también lo es.

Transferencia de datos. En la necesidad de enviar y recibir información, las aplicaciones requieren de la transferencia de datos en base a uno o varios protocolos, cada protocolo se encarga de enviar cierta información de forma particular, por lo que cada vez que un cliente recibe o envía un bloque de datos debe saber la forma en que es transmitida la información para su codificación y decodificación.

Aplicaciones Cliente Servidor. Si este tipo de aplicativos los definimos como aquella aplicación que genera información, como es el caso de un editor de texto, y además, esta es compartida, distribuida o transferida, podemos pensar que para la creación de una aplicación de este tipo, es necesario (en muchas ocasiones), definir mecanismos de lectura, escritura y manipulación de datos y posiblemente, un nuevo protocolo de comunicación para su transferencia. Por lo tanto, podemos deducir, que la duplicidad de procesos en este tipo de aplicativos es muy considerable.

Estos son algunos ejemplos de la complejidad que implica el diseñar y construir programas de **software*** (ya sea de tipo cliente, servidor o local) debido a que regularmente cada programador utiliza procesos únicos para la lectura y escritura de bloques de datos, a generando en el transcurso de los años la redundancia de procesos en las diferentes plataformas, como en los proveedores de software. Por lo que este capítulo hace notar la importancia de unificar procesos tan comunes como los son la lectura y **escritura de datos***, y finalmente demostrar que esta unificación reducirá el tiempo inversión en el desarrollo de **software*** y consecuentemente, el margen de error al momento de almacenar o leer cualquier tipo de dato con un solo mecanismo (proceso).

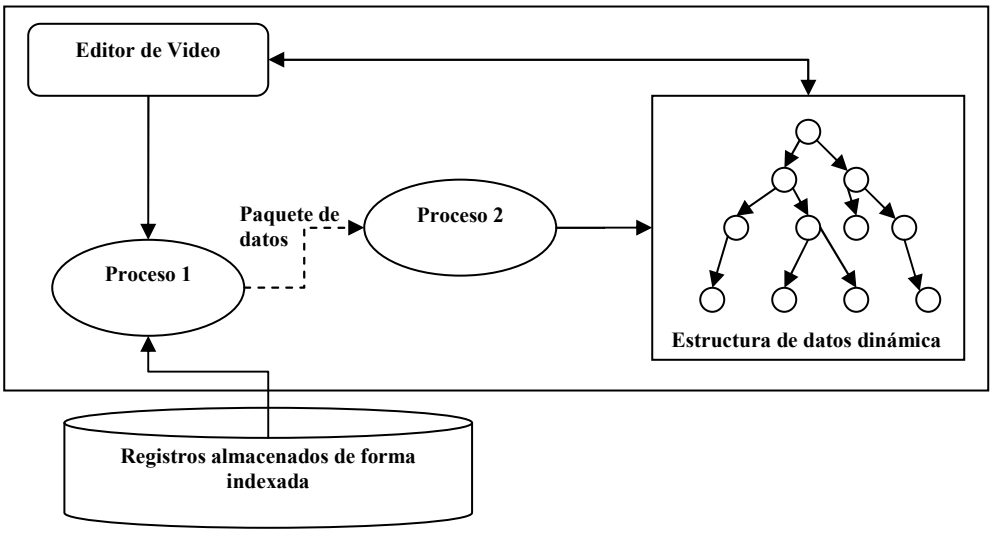
II.I. Deficiencias en la lectura y escritura de datos en un archivo

En la inmensa variedad de programas que generan algún tipo de archivo, es común redundar los procesos de lectura y escritura de datos, debido a que cada uno utilizan un formato único en cada programa, como es el caso de la lectura de datos contenidos en un archivo y su vaciado a una **estructura de datos*** situada en la memoria volátil. Desde el nacimiento de las **estructuras de datos*** han facilitado la búsqueda, definición y manipulación de datos. En los últimos años, han existido dos **estructuras de datos*** básicas:

Estructuras de datos estáticas. *También conocidas como arreglos o registros, reciben este nombre debido a que durante la compilación se les asigna espacio de memoria, y este permanece inalterable a lo largo de la ejecución del programa, es decir, las variables no pueden crearse ni destruirse durante la ejecución del programa.*

Estructuras de datos dinámicas. *Este tipo de estructuras es generado a partir de un tipo de datos conocido con el nombre de puntero o referencia. Un dato puntero almacena una dirección o referencia a un dato propiamente dicho* **[4]**.

Un ejemplo de este, es la forma de operar un editor de video que almacena la secuencia de imágenes en un archivo de "*forma indexada"*, como se muestra en el punto I.IV, y en la memoria volátil se emplea una **Estructura de Datos*** dinámica para su manipulación, tendríamos la siguiente perspectiva:



En este ejemplo, el **Editor de video** emplea al **Proceso 1,** para la extracción y decodificación de los datos almacenados en un archivo (de forma indexada), una vez decodificado el bloque de datos es enviada al **Proceso 2** para su lectura y vaciado a una estructura de datos específica, para finalmente ser leída por el **Editor de Video.**

Pero en caso de que se deseé fabricar un software que se comporte de la misma forma, se tendría que "rehacer", tanto el Proceso 1, como el 2, para la lectura de un archivo a una estructura de datos específica.

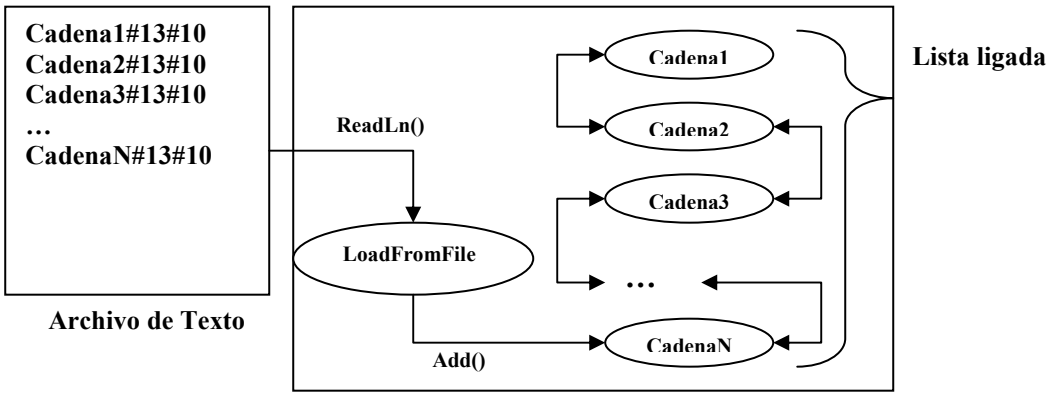
Hoy en día existen nuevas **Estructuras de Datos*** complejas que ha logrado conceptuar un proceso como un tipo de dato, creando un concepto conocido como **objeto*** el cual combina tipo de dato de tipo dato conocidos como *atributos* y tipos de datos de tipo procesos conocidos como *comportamientos*.

*Un **Objeto** es un tipo de dato que envuelve los datos y código en uno. Antes de la Programación Orientada a Objetos (POO), los datos y el código eran elementos totalmente diferentes* **[3]**

Cabe mencionar que esta es solo una burda definición de objeto, ya que para el "*Paradigma Orientado a Objetos*", existen obras completas que definen normas y conceptos para conceptualizar y especificar lo que es un objeto en el contexto de la programación de computadoras. En donde uno de los puntos más importantes es que un objeto, no solo es juntar código con datos, sino que implica también la representación de algún objeto o entidad en una estructura de datos con procesos anidados, con un comportamiento y ciertos atributos.

Una solución para la estandarización para la lectura datos contenidos en un archivo es un, tipo de dato de tipo llamado Clase (definición de un objeto) de la empresa de **Software*** Borland denominada "TStringList", el cual utiliza una **Estructura de Datos*** dinámica conocida como "lista ligada" *(Colección de elementos llamados generalmente nodos, el orden entre nodos se establece por medio de punteros, es decir direcciones o referencias de nodos* **[4]***)*, y su función principal es vaciar el contenido de un archivo de texto a una lista ligada, para finalmente manipular los datos o nodos de la lista ligada en memoria virtual.

Cabe resaltar que el archivo de texto utiliza normalmente el formato de tipo *Etiquetado (como se mencionó en el punto I.IV)* para la representación de cadenas. Esto es que, una secuencia de caracteres tipos ASCII representan un párrafo o renglón delimitados por otra secuencia de caracteres conocido como *salto de carro* (secuencia de los códigos ASCII 13 y 10 o bien únicamente con el código 10), por lo que un objeto del tipo TStringList, lee el archivo de texto, y al identificar un salto de carro, vacía o identifica la cadena inmediatamente anterior al salto del carro, como un nuevo nodo dentro de la lista ligada, como se ilustra a continuación:



Nota: el símbolo # representa la **concatenación*** de un código ASCII en una cadena, en este caso el salto de carro**.**

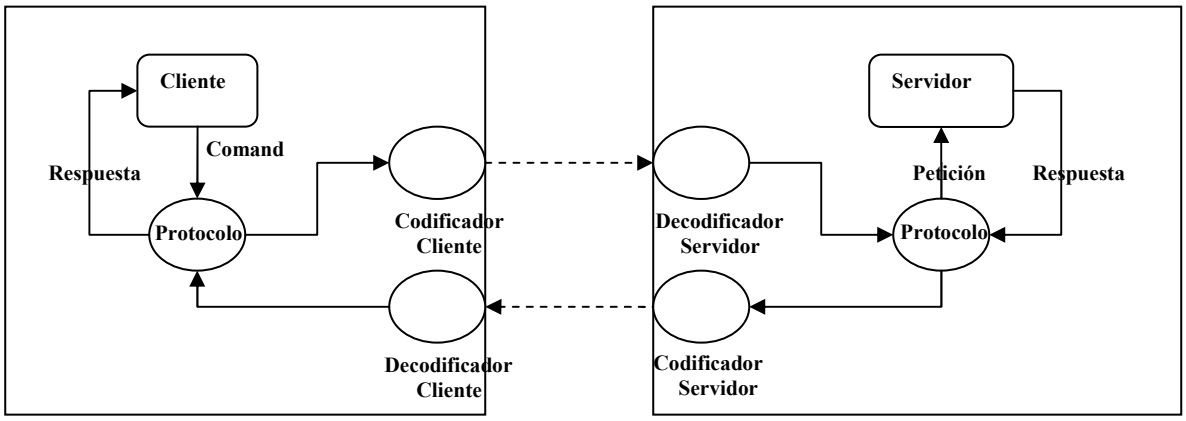
En este ejemplo, el objeto *Lista* es definido por la clase TStringList, la cual define cada uno de los atributos de la lista ligada (como son el número de nodos y la lista de referencias de cada uno de los nodos contenidos en la lista) además de definir sus comportamiento ó métodos (leer las cadenas desde un archivo con el proceso *LoadFromFile*, agregar un nodo al final de la lista con *Add*, o insertar un nodo entre dos cadenas con *Insert*).

Esta forma de cargar un texto o escrito a una lista ligada, resulta ser una opción óptima y simple para aplicaciones que haga uso del mismo, como son, archivos de configuración, documentos, ayuda, etc. Sin embargo, para la lectura y almacenamiento de otro tipo de datos, como para el caso de un video, sería necesario generar un nuevo **objeto*** especializado en la lectura de un archivo con un formato especifico y óptimo para la lectura y almacenamiento de imágenes.

II.II. Deficiencias en la transferencia de datos

Para lograr la transmisión de datos de forma electrónica, como anteriormente se detalló, es necesario determinar el **protocolo*** de comunicación para el envío y recepción de información, las formas de atender las peticiones de los clientes, así como reconocer y solucionar cualquier error de comunicación. Por lo que cada protocolo de comunicación conocido, determina sus comandos y alcances que ejerce sobre un equipo remoto.

En el siguiente ejemplo se ilustra como una aplicación Cliente envía un comando a un Servidor y recibe un bloque de datos como respuesta.



En este ejemplo, el proceso en cargado de llevar a cabo el **Protocolo,** como se mencionó en el capítulo anterior, es el que *establece las reglas de comunicación*, sin embargo, antes de saber si procede o no un comando, o si la respuesta es satisfactoria, es necesario determinar lo que representa el bloque de datos recibido mediante los procesos **codificador** y **decodificador**, ya sea un comando, un dato obtenido, un error o una notificación, dependiendo de las reglas establecidas por el mismo.

En este caso, si se quisiera implementar un nuevo protocolo, es necesario establecer los procesos de codificación y decodificación propios del protocolo, para determinar que tipo de información es, incluso si fue recibida o transmitida correctamente, y por tanto el mismo proceso que represente dichas reglas de comunicación (Protocolo). Estos procesos son en muchas ocasiones únicos tanto para el lado del cliente como para el lado del servidor, por lo que número de procesos por rehacer es considerable.

Actualmente, existen varias tecnologías para el envío y recepción de Estructuras de Datos* que identifiquen su contenido por si misma. Una opción establecida para este fin, son los objetos XML, utilizados en plataformas Web como solución a la premisa de identificar el tipo de dato o entidad recibida o enviada por un protocolo de comunicación SOAP* (Protocolo de Arquitectura Orientada a Servicios

*XML es un lenguaje basado en texto aumentado, este es un rápido y conveniente estándar para el intercambio de datos en la web. Tal como en **HTML***, los datos son identificados por etiquetas (tags) (identificadores limitados pos símbolos de mayor y menor que: `<...>`). Colectivamente, las etiquetas son conocidas como márgenes (markup).*

*Pero a diferencia de **HTML***, las etiquetas XML identifican los datos mejor como cuando son mostrados. Mientras una etiqueta **HTML*** expresa algo como "Despliega este dato con fuente en negrita" (`<b>...</b>`), una etiqueta XML actúa como el nombre de un campo en un programa. Esto pone una etiqueta de una pieza de datos que lo identifique (Por ejemplo, `<message>...</message>`).*

*De la misma forma en que se definen los nombres de los campos para una **estructura de datos***, existe la libertad de usar cualquier etiqueta XML que le de sentido en una aplicación dada.*

Normalmente, para múltiples aplicaciones hacen uso del dato XML, estos deben ser agregados sobre el nombre o etiqueta que corresponda.

Este es un ejemplo de un mismo dato utilizado para una aplicación de mensajería:

```
<message>
  <to>you@yourAddress.com</to>
  <from>me@myAddress.com</from>
  <subject>XML Is Really Cool</subject>
  <text>
    How many ways is XML cool? Let me count the ways...
  </text>
</message>
```

El XML es la mejor representación de datos de intercambio para la web. Es extraordinario cuando es usado en conjunción con programas de la plataforma Java que envían y reciben información. Así como para aplicaciones cliente - servidor, por ejemplo, se podría transmitir datos con formato XML de ida y vuelta entre el cliente y el servidor.

Se espera que, XML sea una respuesta potencial para el intercambio de datos en todo tipo de transacciones, tanto como se vaya empleando. (Por ejemplo, ¿Un programa de e-mail podría contar con etiquetas llamadas <FIRST> y <LAST>, o <FIRSTNAME> y <LASTNAME>?) La necesidad de los estándares comunes podría generar parte de la estandarización de la industria específica de esfuerzos de los próximos años **[U10]**.

La estandarización implementada en XML, ha tomado gran auge en el almacenamiento y transferencia e identificación de datos. Sin embargo, para poder utilizar esta tecnología, es necesario cargar en un programa cliente y otro servidor capaz de traducir o **compilar*** la estructura XML para posteriormente identificar cada uno de las etiquetas. Este proceso de identificación tiene un precio elevado en cuanto a rendimiento, por lo que es utilizado, la mayoría de las veces, del lado del Servidor (en una computadora de gran rendimiento), además de estar limitado a la definición de un solo registro o entidad (objeto).

Esto nos da pauta para pensar, que si solo se trabajara en estructuras bien definidas como es el caso de **JAVA*,** el estándar establecido por XML es 100% funcional y necesario, sin embargo, en el mundo cotidiano, no solo se manejan objetos ni estructuras bien definidas, sino, también, hablamos de comandos, respuestas, mensajes, archivos y toda clase de datos que no siempre es conveniente definir en una estructura, por lo que el modelo presentado en este trabajo, sugiere una solución alterna a esta tecnología, capaz de identificar a cada tipo de dato conocido.

II.III. Aplicativo Cliente Servidor

En un modelo cliente Servidor, la redundancia de procesos es común, debido a que la mayoría de estas aplicaciones tiene un comportamiento similar:

- Identificación de usuario
- Atención a las peticiones del cliente
- Finalización de la sesión

Identificación de usuario, en cada uno de los protocolos de aplicación se establecen las reglas de seguridad necesarias para identificar a cada usuario que desee establecer una conexión. Para esto, es necesario contar con algoritmos de encriptación, una base de datos donde se defina cada uno de los *usuarios*, sus *privilegios*, los recursos disponibles dentro del contexto del aplicativo, así como la relación entre *el usuario y recurso*, *el recurso y privilegio* y *el usuario y privilegio*. En el mejor de los casos, son reutilizados los recursos (Usuarios, privilegios y recursos) definidos por el Sistema Operativo, sin embargo no siempre es posible contar con estos recursos, por lo que cada aplicativo es orillada a crear sus propios algoritmos, recursos y bases de datos.

Atención a las peticiones del cliente. Para la atención especializada de clientes, es necesario definir un **protocolo*** capaz resolver peticiones necesarias para lograr un fin en común entre cliente y servidor. Sin embargo, no siempre es posible utilizar el mismo protocolo, o parte del mismo, debido a que cada protocolo es creado para un fin muy específico y no siempre es posible reutilizar métodos de identificación de peticiones o comando de un protocolo ya establecido, es decir cada protocolo contiene, en muchas ocasiones, diferentes comandos, así como las formas y métodos para la identificación de comando y recursos al momento de atender una petición.

Finalización de la sesión. Es normal encontrar en un protocolo de comunicación, el comando para finalizar una sesión es el comando QUIT, y en otro protocolo es el comando EXIT. Esto es consecuencia de que los comandos son diferentes en cada protocolo y que cada uno utiliza palabras similares para un mismo comando. Esto nos da a entender la discrepancia que existe entre los proveedores de software ya que, si no se comparte un solo comando, mucho menos un proceso en común.

Pensemos ahora que el servidor requiere leer un archivo, decodificarlo y finalmente transmitirlo a un cliente. Pudiéramos pensar en una lista de procesos candidatos a duplicarse en aplicativos Cliente Servidor:

Cliente - Codificar una petición para ser leído por un Servidor
Servidor - Decodificar la petición del cliente
Servidor - Procesamiento de la petición
Servidor - Codificar el resultado para ser devuelto al cliente
Cliente – Decodificar el resultado y procesarlo

Suponiendo que se tiene un cliente de pocos recursos (Workstation Laptop, Palm, etc.) y un software específico (PalmOS, Symbian, Win95, Pocket PC, etc.) y un servidor de arquitectura

robusta (Súper Computadora), con una plataforma especializada en Servidores (UNIX, Windows Server, Linux, Solaris, Sun, etc.), se quiere construir una aplicación Cliente Servidor, que transfiera información de algún archivo o medio accesible dentro del servidor y enviado al cliente cuando lo desee, es importante tener en cuenta los siguientes puntos.

Lo primero en que se tendría que pensar, es el lenguaje programación adecuado a cada plataforma, y de ser posible, reutilizar parte del código utilizado tanto en el Cliente como en el Servidor. Sin embargo, esto en muchos de los casos no es posible llevarlos a la práctica, ya que cada plataforma tiene un lenguaje nativo de programación y funciones propias del mismo o simplemente son desarrollados por distintos programadores.

Posteriormente definir procedimiento únicos como es el caso de la codificación y decodificación de comandos y respuestas, tanto para el Cliente como para el servidor, sin embargo, esto no siempre es fácil, ya que en la mayoría de los casos, los procesos se ajustan a las capacidades y funciones disponibles en el lenguaje de programación y plataforma resultando dos procesos totalmente diferentes tanto para el Cliente como para el Servidor.

Finalmente, lo más recomendable sería el reutilizar el procesos codificador y decodificador utilizado para almacenar los datos en un archivo, sin embargo, un archivo no siempre se ve ni se maneja de la misma forma en las plataformas, es decir, en una plataforma puede ser una secuencia continua de códigos ASCII y en otra puede ser un una colección continúa de registros.

Por lo que se resalta la importancia y necesidad de utilizar un solo lenguaje de programación y una sola coleccione de funciones u objetos, paradigmas y definición de los recursos, para realizar una aplicación Cliente Servidor reutilizando los mismos mecanismos, conceptos y en mejor de los casos la misma colección de objetos.

II.IV. Redundancia de procesos

Como ya se ha mencionado, la reutilización de procesos es un factor muy importante para la optimización de recursos, tiempo de desarrollo y reducción en el margen de error, como se menciona en el paradigma “Orientado a Objetos”.

Características importantes del análisis y diseño orientado a objetos:

1. *Cambian nuestra forma de pensar sobre los sistemas. Para la mayoría de las personas, la forma de pensar OO es más natural que las técnicas del análisis y diseño estructurado. Después de todo, todo está formado por objetos.*
2. *Los sistemas suelen construirse a partir de objetos ya existentes. Esto lleva a un alto grado de reutilización, a un ahorro de dinero, un menor tiempo de desarrollo y una mayor confiabilidad del sistema.*
3. *La complejidad de los objetos que podemos utilizar sigue en aumento, puede que nuestros objetos, se construyan a partir de otros. Estos a su vez están constituidos por otros objetos, etcétera.*
4. *El depósito **CASE*** debe contener una creciente biblioteca de tipos de objetos, algunos comprados y otros construidos en casa. Es muy probable que estos tipos de objetos sean más*

poderosos conforme crezca su complejidad. La mayoría de este tipo de objetos serán diseñados de forma que se adapten a las necesidades de los diferentes sistemas.

5. *La creación de sistemas con un funcionamiento correcto es más fácil con las técnicas OO. Esto se debe, en parte a que las clases OO están diseñadas para reutilizarse; y en parte, en que las clases están autocontenidas y divididas en métodos. Cada método se puede construir, depurar y modificar con relativa facilidad.*
6. *Las técnicas OO se ajustan de manera natural a la tecnología **CASE***. Existen ciertas herramientas elegantes y poderosas para la manipulación OO. Muchas otras herramientas **CASE*** necesitan ciertas mejoras para controlar el análisis y diseño orientado a objetos* [5]

Por lo que el presente modelo pretende realizar un análisis y diseño orientado a objetos, con el fin de aprovechar los beneficios antes mencionados. Cabe destacar que también es necesario definir uno o varios tipos de objeto que nos permitan la interacción y definición de mecanismos capaces de llevar las tareas antes expuestas en forma de Objetos. Otra necesidad importante es seleccionar el un lenguaje de programación OO, capaz de llevar a cabo los conceptos más importantes que conlleva el paradigma Orientado a objetos.

Finalmente, al utilizar un lenguaje, diseño y análisis Orientado a Objetos, se podría reutilizar varios de los procesos antes mencionados, por ejemplo, un mecanismo o tipo de objeto, de lectura de datos, podría ser utilizado en una aplicación tipo Cliente, como en una tipo Servidor.

II.V. El paquete de datos como una solución (hipótesis)

La hipótesis en la que se basa este modelo, es la idea de definir un solo mecanismo o tipo de Objeto para almacenar, leer, transmitir y recibir cualquier tipo de información, previamente definido. Este mecanismo o tipo de objeto, se apoyaría en una estructura capaz de definir casi cualquier tipo de dato, desde un carácter, hasta una secuencia de registros.

Esto lograría varias ventajas en la construcción de aplicaciones que requieran transferir, almacenar y reconocer algún tipo de dato, en un medio de almacenamiento o de transferencia.

Para lograr este objetivo se plantea definir un paquete de datos que contenga cualquier tipo de dato, y de ser posible ser identificado por si mismo, con el fin de ser identificado casi para cualquier contexto. Los procesos que nos auxiliarían en el manejo del mismo serían:

- **Codificar**: Almacena y codifica un paquete de datos en una cadena continua de bytes (Buffer), para ser almacenada en un archivo o bien ser transmitida hacia un enlace de red, aplicación o periférico.
- **Decodifica**r Decodifica un paquete de datos almacenado en una cadena continua de bytes (ya sea en memoria virtual o en un archivo) y vaciarla a una estructura de datos, cuando este sea requerido.
- **Navegar**: Navega sobre una cadena continua de bytes (ya sea en memoria virtual o en un archivo), extrayendo la información requerida, cuando las dimensiones del paquete sean extremas, y no sea posible o recomendable vaciarlo a una estructura de datos en la memoria Virtual.

- **Interpretar**: Interpreta el contenido del paquete y lo procesa, enfocado principalmente a un protocolo de comunicación de petición respuesta.

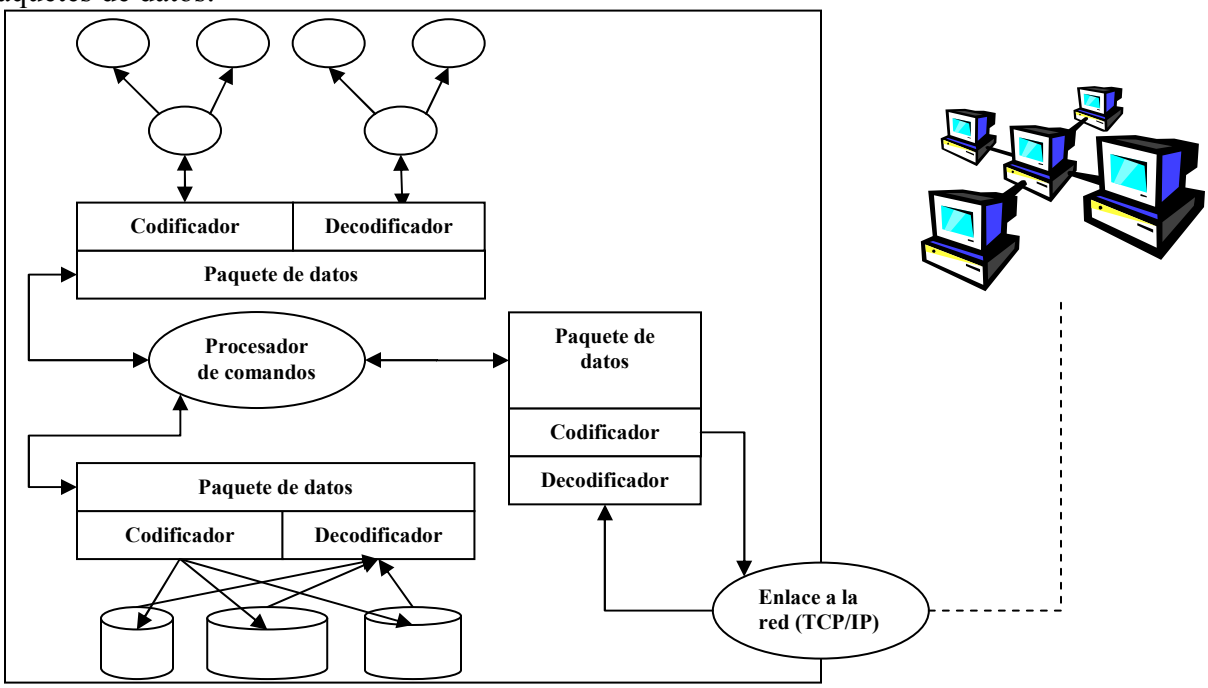
Para lograr una mayor portabilidad y la reutilización de código es necesario seleccionar un lenguaje de programación que sea preferentemente Orientado a Objetos y funcional casi en cualquier plataforma, Cliente o Servidor, como es el caso de JAVA.

Existen cinco objetivos principales en la creación del lenguaje java:

- *Hacer uso de la metodología conocida como programación Orientada a objetos.*
- *Permitir ejecutar el mismo programa en múltiples plataformas.*
- *Incorporar el soporte para el trabajo en red entre computadoras*
- *Ser diseñado para ser ejecutar código desde un fuente remoto de forma segura.*
- *Fácil de usar y la reutilizar partes benéficas de viejos lenguajes Orientados a Objetos como C y C++* **[U11]**.

Por lo que JAVA sería el lenguaje de programación idóneo para la implementación de este modelo, logrando que los procesos o componentes que nos auxilien en la manipulación del Paquete de datos sean idénticos tanto del lado del Cliente como del Servidor y por consecuente en una local. Pensando que si definimos una estructura que se defina por si misma, además de aprovechar

A continuación se muestra un ejemplo de lo que se podría lograr con la utilización de un lenguaje como Java, y con un paquete que defina su contenido por si mismo manipulación de paquetes de datos:



Con esto se concluye que el modelo sería una herramienta eficaz para la implementación de aplicaciones cliente servidor, gracias a su portabilidad y manejo de información tanto local como remota así como su adaptabilidad a las necesidades básicas de comunicación.

Capítulo III

Modelo para el almacenamiento y lectura de paquetes de datos

II. Modelo para el almacenamiento y lectura de paquetes de datos

El presente modelo se enfoca a la definición e interpretación de datos de una forma rápida y eficiente, capaz de ser enviada por un medio de comunicación, guardado en un dispositivo de almacenamiento o bien en memoria virtual, para su consulta o manipulación.

Para lograr una lectura correcta, el modelo se sugiere identificar el tipo de dato al momento de leer un conjunto paquete de datos, y almacenarlo o leerlo en una estructura que corresponda al tipo de dato definido.

Desde los principios de la computadora, se ha definido un tipo de dato según como haya sido definido por el programa que lo gestiona. Esto es que un conjunto de bytes es interpretado por el Sistema operativo o aplicativo que maneja.

Por ejemplo, un carácter es representado por un byte, un número entero por 4 bytes, etc. Por lo que se debe definir una interpretación estándar para que otros programas puedan interpretarlo de la misma forma que el programa origina, el "Modelo para el almacenamiento y lectura de paquetes de datos" define únicamente la forma de cómo interpretar el paquete y poderlo interpretar o manipular.

II.I. Metodología

La metodología para solucionar el problema de homologación en el transporte de información así como de su lectura y escritura es utilizando datagramas, semejante al uso que se les da en las redes de conmutación de paquetes (x25).

A finales de los años sesenta, EE.UU. diseño una nueva tecnología denominada conmutación de paquetes, mediante el cual toda la información adicional que sale de una terminal para ser transmitida por la red es dividida en bloques de una determinada longitud llamados paquetes. A cada paquete se le añade una información adicional al comienzo del mismo y, así, se puede mover por la red de forma independiente. Si en un momento dado, una ruta o un nodo de comunicación quedara fuera de servicio, los paquetes se desviarían por otras rutas para llegar a su destino **[2].**

Para este tipo de redes nace un nuevo tipo de comunicación denominado datagrama, el cual es definido de la siguiente manera:

*Un **datagrama** permite que cada paquete recibido por la red se entregue en la dirección de destino especificada con independencia de cualquier otro paquete que dicho terminal envíe o haya enviado formado parte del mismo mensaje* **[2].**

En base a esta tendencia de incluir la información en un especie de paquete y de ser identificada mediante una cabecera al comienzo del mismo, se plantea que al definir todo tipo de información en un paquete de datos (datagrama), y ser identificado mediante una cabecera, es posible ser leída en cualquier plataforma que reconozca este modelo.

Al ser un paquete de datos o datagrama estándar, es posible suponer que este puede ser homologada en la mayor parte de plataformas conocidas. Es decir, si se implementara método tan simple como el de leer las instrucciones de un shampoo y saber como aplicarlo o el instructivo de aparato electrónico para saber usarlo.

Por lo que en este capítulo se mostrara como definir tipos de datos básicos de información necesarios para construir el datagrama, posteriormente se define una cabecera estándar para diferentes tipos de paquete y finalmente se define el contenido dependiendo el tipo de paquete que se requiera.

II.II. Tipos de datos

El uso de un tipo de dato es definido en los lenguajes de programación:

Los lenguajes de programación proveen algunos tipos de datos primitivos como bloques de básicos de construcción para programas y otras composiciones de tipos mas especializados. Típicamente incluyen varios tipos de datos primitivos como enteros, puntos flotantes, y tipos de cadena. Aunque bloques de construcción básicos como arreglos, registros y referencias para relacionar entre piezas de datos no pueden ser incluidos en los tipos de datos primitivos, pueden ser vistos como la colección de varios valores primitivos.

Como se mencionó anteriormente, en los programas actuales, solo se limita a tipos de datos estáticos o primitivos tomando como bloques básicos, los tipos de datos primitivos los cuales suelen ser estáticos, de modo que las estructuras dinámicas quedan fuera del alcance de los bloques básicos del programa. El presente modelo pretende ir más allá, de modo que, con un paquete de datos, podemos almacenar y manejar cualquier tipo de dato (dinámico y estático), no es capaz de definir e identificar datos primitivos sino dinámicos. En la siguiente tabla se muestran algunos tipos de datos que pudieran definirse en un contexto informático, (objetos o paquetes de datos mas socorridos), definiendo un identificador para cada paquete.

Carácter identificador	Tipos de almacenamiento	Descripción
C	Cursor o Tabla	Estructura definida compuesta por n registros y m columnas, en donde cada columna o campo se define por un tipo de dato como son: carácter, numérico y fecha principalmente
T	Texto	Conjunto de caracteres secuenciales, en donde cada renglón se define por el código ASCII 10 o bien por el 10 + 13, y el final de la cadena por el 0, o bien, es determinado por una longitud determinada
B	Binario	Conjunto de códigos ASCII secuenciales por una longitud definida
E	Error	Este puede ser un utilizado para mensajes entre aplicaciones y esta constituido por un número identificador y el texto correspondiente
F	Función o comando	Una función o comando, es utilizada por protocolos o requerimientos solicitados entre aplicaciones, y se define como el tipo de dato a regresar, el identificador del comando, y los n parámetros requeridos para esta función, este se puede ver similar a un cursor con sus campos.
N	Nulo	Este valor es solo una bandera para determinar que el dispositivo de almacenamiento esta vacío

El modelo sugiere un paquete definido por dos partes: en la cabecera y el cuerpo o Buffer para lograr identificar el paquete y obtener su contenido:

La Cabecera: Bloque de datos que definen las dimensiones del paquete, así como el tipo de dato que almacena (paquete). Es importante mencionar que gracias a este, es posible determinar si el paquete esta íntegro y si el programa que lo interpreta, tiene la capacidad de leerlo, o de determinar si este es paquete esperado.

El cuerpo o Buffer: Espacio o bloque destinado a tener el dato definido por la cabecera, cualquier incongruencia entre este y la cabecera, debe ser motivo de una excepción del sistema o una cancelación dentro del proceso que lo interpreta.

Con estos elementos es posible la movilidad del paquete así como su identificación en algún medio de almacenamiento, o medio de comunicación logrando su interpretación y manipulación donde quiera que este resida.

II.III. Paquetes de datos

Ordinariamente, en los sistemas distribuidos locales o remotos se trabaja con un sin fin de estructuras de datos específicas, pero para demostrar el alcance de este modelo, se han definido 6 tipos de paquetes.

Cabecera

La cabecera, como anteriormente se mencionó, debe contener la información necesaria para la identificación y lectura del conjunto de bytes o bloque de datos a leer.

Al igual que otros programas, para poder definir una estructura estándar entre los diferentes lenguajes de programación, estas estructuras deben definirse en tipos de datos conocidos, por lo que la cabecera, se define como una estructura de bytes estática de 6 elementos, los cuales son de tipos numéricos conocidos:

Entero Corto: Bloque de 2 bytes equivalentes a 16 bits, con lo que es posible representar hasta el número 65,536 (2 elevado a la 16). También es posible representar códigos estándares como el Unicode (Conjunto de caracteres estándar que utiliza hasta 65,536 símbolos creado para representar diferentes escrituras, incluso estándares anteriores).

Entero: Bloque de 4 bytes equivalentes a 32 bits, con el que es posible representar enteros ordinarios de un rango de -2,147,483,648 hasta 2,147,483,647.

Entero largo: Bloque de 8 bytes equivalentes a 64 bits, con lo que es posible representar un número entero muy grande (de -2 el exponencial 63 hasta 2 el exponencial 63).

A continuación se describe la estructura de la cabecera mediante los tipos definidos previamente, definiendo el tipo de paquete así como sus características.

Dato/Tipo	Tipo	Cursor	Texto	Binario	Mensaje	Función	Nulo
Tamaño de la cabecera	Entero largo	Número de bytes que ocupa la cabecera, previendo que en futuras versiones del modelo se realicen ajustes a esta.					
Tamaño del Buffer	Entero largo	N bytes de longitud del bloque que contiene la información del bloque					0
Tipo*	Entero corto	'C'	'T'	'B'	'M'	'F'	'N'
Renglones	Entero	n Registros	n Renglones	1	1	1	0
Columnas	Entero corto	n Columnas		0	2	n Parámetros	0
Comprimido	Entero corto	0: Determina que el contenido del buffer no esta comprimido 1: Determina que el contenido del buffer esta comprimido					0
Encriptado	Entero corto	0: Determina que el contenido del buffer no fue alterado Otro: Determina la identificación del método de encriptación utilizado en la información del buffer					0

* El campo Tipo, está definido como un entero que corresponde a los caracteres definidos por el estándar UNICODE.

Para la identificación y lectura del paquete, es necesario proceder con el siguiente algoritmo.

1. Integridad de la paquete: Verificar que el tamaño de bytes en el buffer sea mayor o igual al tamaño de la cabecera. En caso de no contar con el tamaño necesario para leer la cabecera, se esperará a que se completen los bytes necesarios, o bien que un tiempo de espera determinado se cumpla.
2. Paquete incompleto: En caso de no cumplir con el tamaño necesario para la lectura de la cabecera, ni del buffer, se entenderá que se finalizó el tiempo y no se obtuvieron los bytes esperados, por lo que se terminará la lectura del paquete con el siguiente error: "Tiempo de espera terminado"
3. Paquete completo: En caso contrario, se obtendrán los datos de la cabecera, de los campos antes definidos.
4. Lectura del buffer o bloque: Se terminará exitosamente la lectura de la cabecera y se procederá con la verificación de integridad del bloque de datos.

Para la representación del comportamiento de este, se han utilizado diagramas de estado para su representación gráfica.

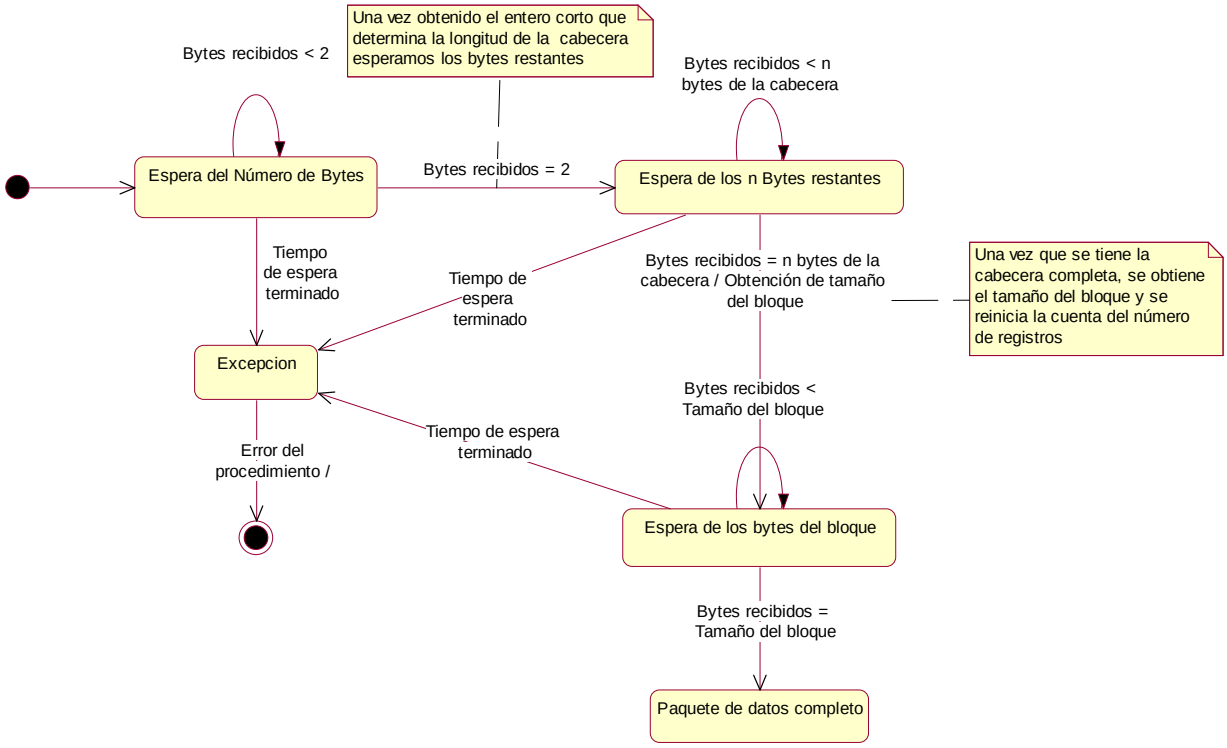
Los aspectos del comportamiento dinámico, normalmente llamados aspectos de control se tratan mediante diagramas de estado. Aquí los estados o modos se pueden anidar y enlazar de varias formas para representar el comportamiento secuencial o concurrente. Las transiciones entere estados se desencadenan normalmente mediante sucesos que se pueden modificar según las condiciones.

Lectura de la cabecera:

Lo primero en que se enfoca el método de lectura del paquete es en determinar parte de la integridad del mismo, ya que de otra forma sería imposible determinar que el paquete estaría completo. Para esto es importante hacer notar que se debe ser imprescindible de donde vienen los datos, ya que estos pueden provenir de una conexión a una red, de un dispositivo de almacenamiento o bien de la memoria virtual, por lo que se recomienda determinar la integridad de la misma.

Para obtener el bloque de bytes correspondiente a la cabecera, es necesario obtener los dos primeros bytes del bloque, ya que este determinará el tamaño únicamente de la cabecera, una vez obtenido el tamaño de la cabecera, se espera el tamaño completo de la cabecera.

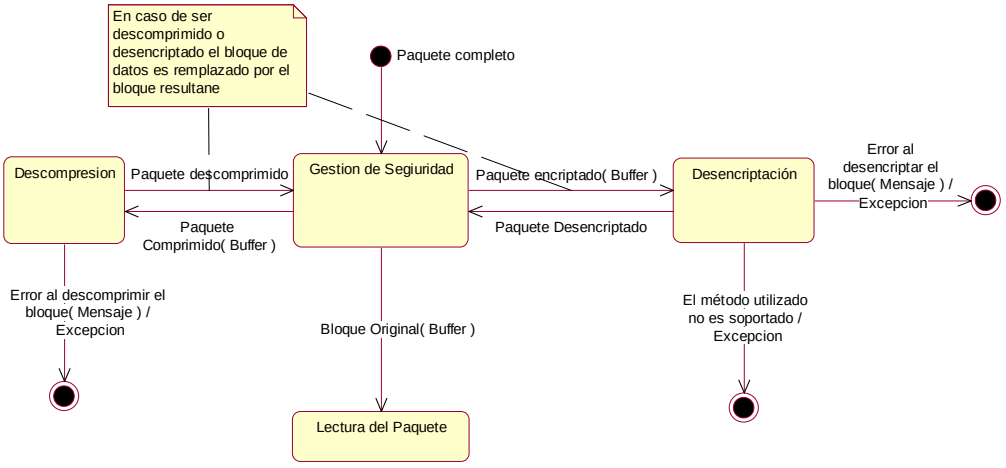
Al término de la lectura de la cabecera, el tercer y cuarto byte, corresponden al tamaño del bloque, por lo que es posible determinar si el paquete está completo.



Hasta este momento, es posible determinar que el paquete esta completo y es posible continuar con su lectura, no antes descomprimirlo o descifrar el bloque de datos, como se describe a continuación.

III.IV. Seguridad y protección de la información

Al momento de tener la cabecera y el bloque de datos completos, es posible determinar si el bloque es encriptado o comprimido y reestablecer el bloque de datos a su estado ordinario para su lectura o manipulación.



Para este estado de gestión de seguridad no hay que olvidar lo importante y variado que puede ser el método de encriptación, debe contarse con un método capas de no ser fácil de descifrar o muy conocido, ya que se perdería el sentido de seguridad, por lo que se sugiere un método abierto de encriptación para aquellas aplicaciones que utilizaran este modelo, esto puede ser mediante una clase abstracta que pueda ser redefinida por la aplicación la cual cumpla con los métodos y comportamientos necesarios.

En el caso de la descompresión se sugieren métodos estándares o simples para que estos puedan ser utilizados por otras aplicaciones cuando este sea de uso compartido, sin necesidad de un código de seguridad. El sugerido y mas utilizado por la comunidad de código libre es el llamado gzip, con el que se podría realizar esta implementación. En caso de contar con otro método de descompresión se sugiere, al igual que en la encriptación un código especial para cada uno de los diferentes métodos y el uso de clases abstractas para el fácil reemplazo de estas.

III.V. Lectura del paquete

Como se determinó anteriormente, en este estado, ya tenemos el bloque de datos listo para identificar, ya que sabemos que está completo, descifrado y descomprimido. En esta lectura, no se pretende cargar los datos del paquete en una estructura de datos, sino que únicamente verificar su integridad para que tengamos la posibilidad de acceder, manipular o simplemente mostrar su contenido.

Es importante recordar que el bloque de datos no es más que un conjunto continuo de bytes que nos representan alguna información. Por lo que es necesario saber identificar los tipos de datos que debiéramos interpretar.

Estamos acostumbrados a la identificación estática de elementos, es decir, antes de leer un conjunto de bytes, es necesario saber su estructura para no leer un conjunto erróneo de datos o bien, cuando esperamos el envío de datos de algún dispositivo, no sabemos con exactitud si fue un error, éxito o simplemente no llegó completo el conjunto de bytes esperados, por lo que cada tipo de dato tomará la siguiente estructura:

Tamaño	Conjunto de bytes
4 bytes	**N bytes**

De tal forma que no importa el dato que sea siempre sabremos el que longitud de bytes se están esperando, incluso determinar cuando no es posible leerlo. Esto deberá interpretarse como un mini paquete considerado como la parte atómica del bloque de datos.

Los tipos de datos son diversos, pero para algunos fines prácticos, es necesario limitarnos, para identificarlos cómodamente como son los siguientes cuatro tipos básicos:

1. **Texto**. El tamaño definirá la longitud de la cadena, en caso de que este sea 0 se entenderá que el bloque destinado para el conjunto de bytes, no existe.
2. **Numérico.** El tamaño nos identificará la longitud y el tipo de número almacenado, o leído:

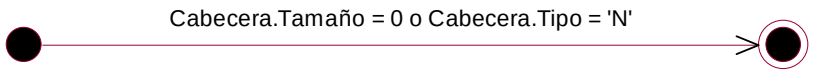
identificador	Tamaño en bytes	Descripción
0	0	Nulo
1	1	Entero muy corto
2	2	Entero corto
3	4	Entero largo
4	8	Flotante

3. **Fecha Hora.** Este siempre tendrá solo tres tamaños definidos:
   - **4.** Cuando solo se tiene la fecha (sin hora), el día se define por el primer byte, el mes por el segundo byte y el año se define por los dos últimos bytes.
   - **3.** Cuando solo se tiene la hora, la hora se define por el primer byte, los minutos por el segundo byte y los segundos por el tercer byte
   - **7.** Cuando se tiene tanto la fecha como la hora, se retoma la definición de 4 bytes y la hora toma la quinta posición, los minutos la sexta posición y los segundos la séptima

4. **Binario.** El tamaño definirá el número de bytes en el buffer
5. **Nulo.** En este caso como no se tiene un tamaño real, siempre el tamaño será de cero.

Lectura de un paquete tipo Nulo

Para determinar que el paquete tiene un valor nulo, es decir que no hay ningún valor definido, simplemente se verificará que el Tamaño definido en la cabecera sea 0 o bien que el tipo definido en la cabecera sea el valor “N”. Esto es ilustrado en el siguiente diagrama



Lectura de un paquete tipo Cursor

Para determinar que el paquete que se está leyendo sea de tipo cursor, habrá que verificar que el valor del campo "tipo" sea "C". Una vez identificado el cursor, se procederá a leer su contenido mediante el siguiente algoritmo.

1. Verificar que el valor "Tipo", tenga el valor 'C'
2. Leer la cabecera específica del cursor, compuesta por los siguientes campos:

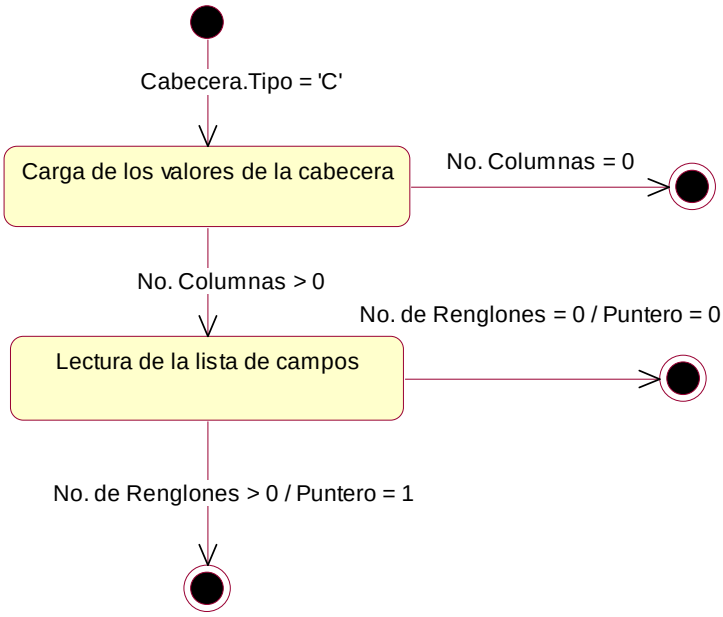
Dato/Tipo	Tipo	Descripción
Nombre del cursor	Texto	Nombre del cursor que se esta leyendo
Nombre de la Base datos	Texto	Nombre de la base de datos a la que pertenece
Dueño	Texto	Dueño del recurso o quien generó el cursor

3. Leer la lista de campos según el número de columnas definidas en la cabecera inicial. Cada uno de los campos se define con los siguientes campos:

Dato/Tipo	Tipo	Descripción
Nombre del Campo	Texto	Nombre del i ésimo campo contenido en cursor
Tipo de dato del campo	Entero Corto	Tipo de dato del i ésimo campo tal como se definió en la tabla correspondiente a los tipos de datos.

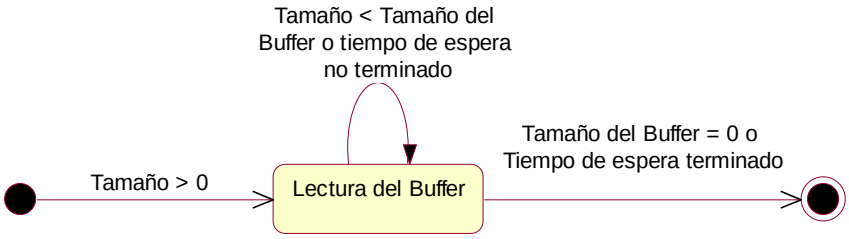
4. Una vez leída la lista de campos, se posiciona el puntero en la primera posición, preparándose para la lectura de cada uno de los registros, con respecto a la lista de campos obtenida. En caso de no tener registros el puntero tendrá el valor cero.

Para ilustrar este proceso se presenta el siguiente diagrama:



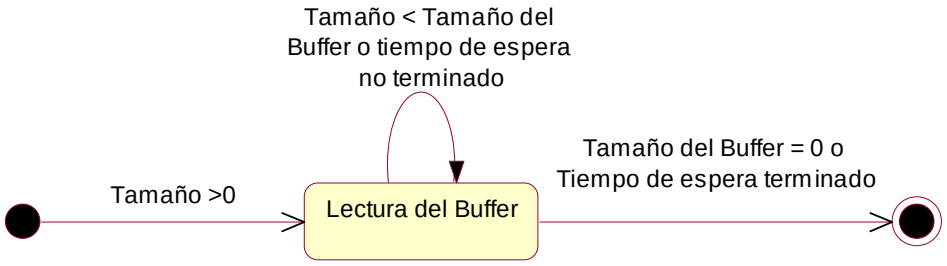
Lectura de un paquete tipo Texto

Una vez leída la cabecera, e identificar que el tipo contiene el valor 'T', y el tamaño con valor mayor a cero, se vaciará el contenido a un arreglo de caracteres con terminación en 0, o en la clase o estructura correspondiente a una cadena de caracteres. Para esto se verificará que el tamaño sea mayor que cero y se esperará un determinado tiempo para verificar que el servidor esté mandando información.



Lectura de un paquete tipo Binario

Al identificar que el tipo de dato contenido la cabecea es 'B', se procede a leer el buffer mientras el tamaño definido en la cabecera sea mayor que cero y que el tiempo de espera definido no haya pasado.

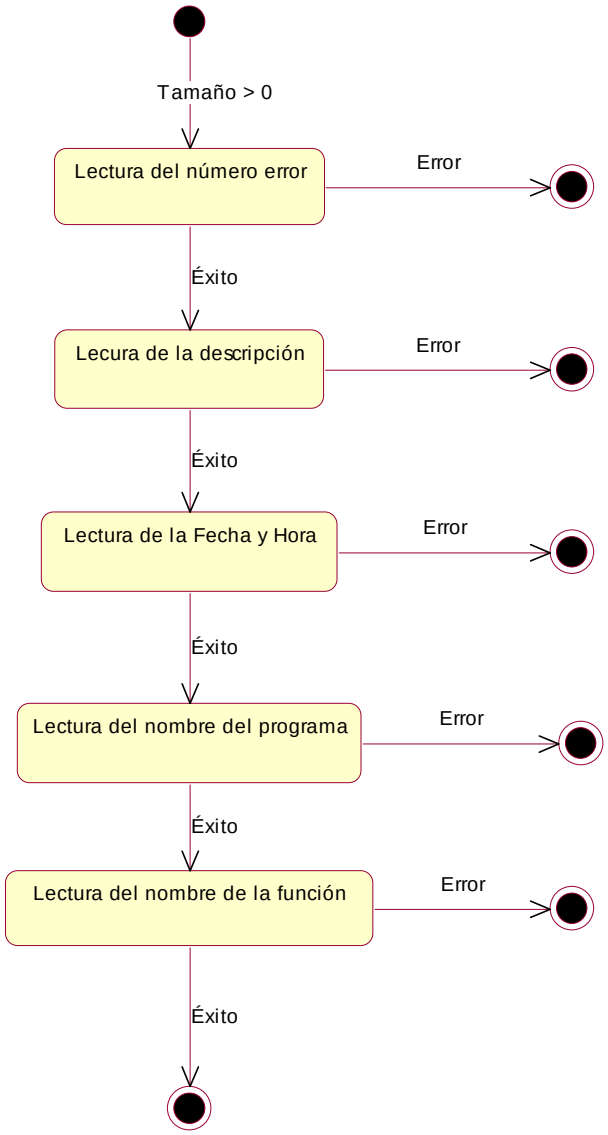


Lectura de un paquete tipo Mensaje

En ocasiones el Servidor o tipo de almacenamiento, nos indica que se ha almacenado un error, esto puede ser en una bitácora o respuesta de algún Servidor, esto regularmente se determina por los siguientes valores, para determinar la causa u origen del mensaje.

Dato/Tipo	Tipo	Descripción
Número de Mensaje	Numérico	Número o tipo de mensaje al que se refiere
Descripción	Texto	Descripción completa del mensaje
Fecha	Fecha Hora	Fecha y hora en que se produjo el mensaje
Programa	Texto	Programa en el que se produjo el mensaje
Función	Texto	Nombre de la función en la que se produjo el mensaje

Para la lectura de un paquete de tipo mensaje, es simplemente leer los campos antes definidos, previniendo los posibles errores que se pudiesen generar, como se ilustra a continuación:



Lectura de un paquete tipo Función

Tanto en Bases de datos como en algunos protocolos se es necesario indicar operaciones tanto en el lado del servidor, como en el cliente, por lo que se es útil, el almacenar o leer de algún medio, una función específica. Para identificar y leer una función específica, nos apoyaremos en la siguiente estructura:

Dato/Tipo	Tipo	Descripción
Número de la función	Texto	Nombre de la función recibida o almacenada
Tipo de dato a regresar	Entero Corto	Tipo de dato que regresa la función

A continuación, se lee la lista de parámetros que recibe la función, en donde cada parámetro se define con los siguientes campos:

Dato/Tipo	Tipo	Descripción
Tipo de dato	Texto	Nombre de la función recibida o almacenada
Nombre del parámetro	Entero Corto	Tipo de dato del parámetro
Valor del parámetro	Buffer	Buffer con el valor del parámetro

III.VI. Escritura del paquete

Uno de los fines de este modelo es la trascripción de un dato en memoria a otro en disco duro, por lo que la escritura de un paquete de datos que reside en memoria o es recibido por medio de una red, este es guardado íntegramente en un dispositivo de almacenamiento sin necesidad de un proceso adicional o costoso. En la figura 3.1, se muestra esta dinámica:

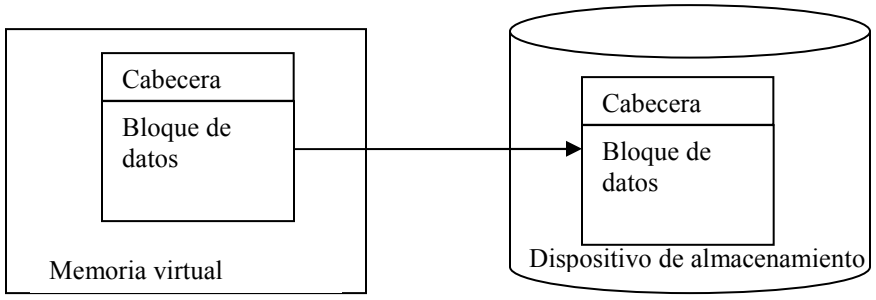


Figura 3.1

Como es de suponer, gracias al modelo ya no existe la necesidad de un proceso de trascripción a otras estructuras por lo que hace la escritura mucho más rápida y eficiente. Al igual que la transferencia de este es transparente, definiéndose como parte atómica de todo un sistema de información.

Capítulo IV

Aplicaciones del Modelo

IV. Aplicaciones del modelo

Una vez definido el funcionamiento y características del modelo en el capítulo 3, el aplicarlo es en base a la necesidad que se quiera aplicar. En este capítulo trata de ejemplificar algunas aplicaciones que podrían construirse bajo el modelo, ilustrando como se interpretarían los paquetes, como se enviarían por una red, como se guardarían en un medio de almacenamiento, o simplemente como copiarían en memoria virtual para su consulta y manipulación.

Cabe recordar que al manejar este tipo de paquetes se tienen muchas ventajas sobre el manejo de información, interpretación, debido a que se utilizan diagramas de estados es posible implementar máquinas de estado para su interpretación y no se necesitaría de un compilador o de un proceso extra para su interpretación, además que al implementar métodos básicos de almacenamiento, es posible suponer su fácil transportación

IV.I. Aplicación Cliente Servidor en función a un protocolo definido

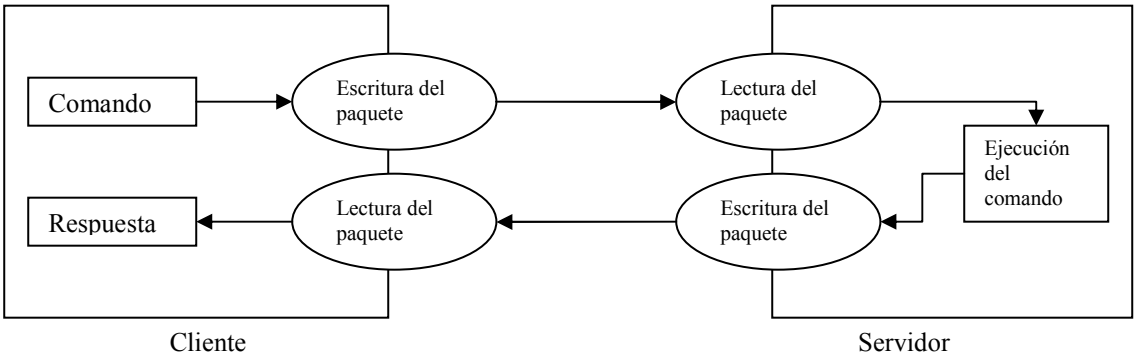
Para la transferencia de este paquete mediante un medio de transferencia es necesario definir un protocolo que interprete el paquete de datos y posteriormente los comando que este reciba.

Un algoritmo que bien podría servir para definir un protocolo que muestre el funcionamiento y las ventajas del modelo podría ser el siguiente.

Envío y ejecución de un paquete:

- **Cliente**: Prepara el paquete con el comando a ejecutar, comprime, encripta y envía por la red.
- **Servidor**: Recibe el paquete de datos, descomprime y lo descifra, en caso de reconocer el comando este lo ejecuta y regresa un nuevo paquete como respuesta. Este paquete de respuesta puede ser cualquier tipo de paquete: mensaje, cursor, comando, bloque binario, etc.
- **Cliente:** El proceso cliente recibe el paquete de respuesta y lo interpreta según su requerimiento. En caso de haber un error al ejecutar el comando, el servidor podría enviar un paquete tipo mensaje con la descripción del error.

A continuación se ilustra en el siguiente diagrama:



Como se puede ver, el proceso de lectura y escritura de ambos extremos (cliente y servidor) es idéntico, por lo que se deduce que se reaprovechan los procesos de entrada y salida. Y en caso de que tuviera el servidor el paquete listo, simplemente lo envía y ya.

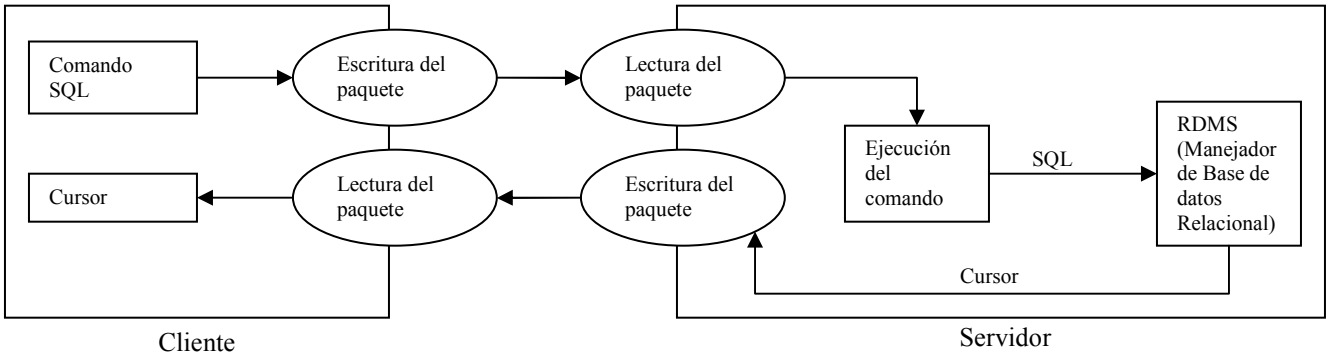
IV.II. Aplicación Cliente Servidor con acceso a una base de datos

En el ambiente de base de datos es común la necesidad de consultar un dato en una gran cantidad de registros. En un equipo con gran capacidad, donde el manejo de cursores o conjuntos de datos, esto esta definido por el propio sistema operativo, sin embargo en un ambiente limitado, como en una Palm o teléfono celular, no es posible contar con componentes de interpretación estándar, debido a la limitación de recursos.

Este tipo de aplicaciones obedece a un modelo de petición respuesta donde el comando podría ser un estándar como el SQL (Lenguaje de consulta estructurado, o por sus siglas en inglés Structured Query Language) o bien el DML (Lenguaje de manipulación de datos o por sus siglas en inglés Data Manipulation Language), y como resultado el servidor regresa un cursor, un mensaje de error, o bien un paquete con el numero de registros afectados.

Otro punto importante en un manejador de base de datos es que difiere según su fabricante, y el aplicar el presente modelo nos evita el ajustar todo el aplicativo a las particularidades del manejador. Esto es que solo el módulo servidor debe ajustarse a estas particularidades.

Para dar un ejemplo de este se ilustra en la siguiente imagen el enlace que necesario desde el cliente al Manejador:



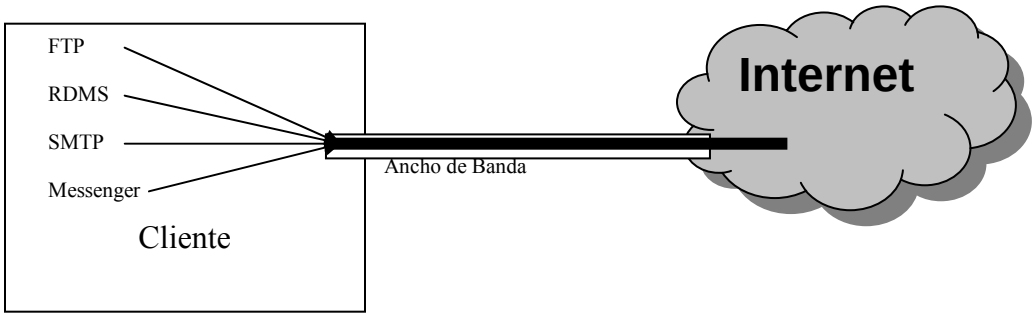
Nótese que este es solo el caso exitoso para la transferencia de paquetes, sin embargo es necesario definir una sesión a nivel usuario, por lo que el módulo servidor tendría que gestionar identificación del usuario mediante un módulo de seguridad adicional, o en el caso de contar con un manejador de base de datos, se recomienda utilizar la seguridad que ya contiene Manejador, es decir el usuario y contraseña que proporciona el cliente, debe ser previamente configurado en el manejador de base de datos.

IV.III. Aplicación Cliente Servidor con manejo de peticiones

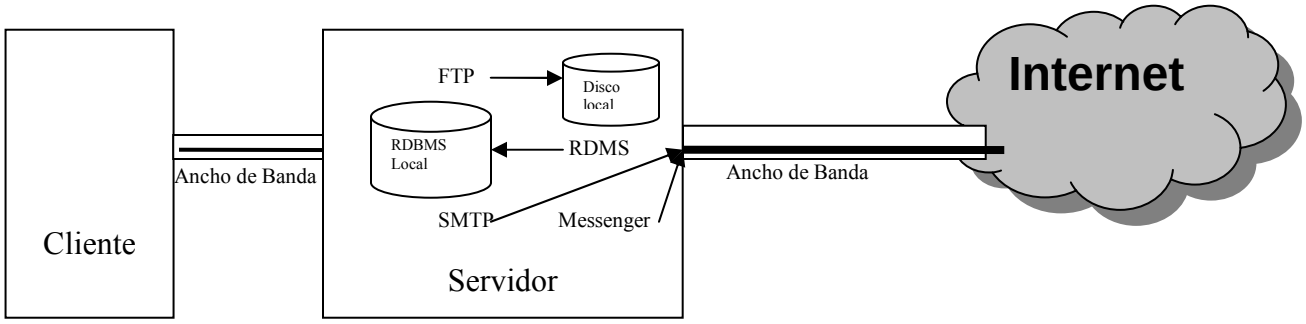
Existen diferentes tipos de aplicaciones cliente servidor, de modo que si un aplicativo requiere de de un tipo de servidor requiere levantar una nueva sesión para interactuar con esta, entre los servicios más comunes son:

- FTP: Utilizado para enviar y recibir archivos en otro equipo o servidor
- RDBMS: Acceso a una base de datos, para consulta o manipulación de datos remotamente
- Mensajería: Programa de conversaciones de texto en Internet, conocidos como “Messenger”
- SMTP: Envío y consulta de corros electrónicos.

Cuando un cliente inicia una sesión a cada uno de estos servicios, entorpece el ancho de banda, debido al cuello de botella que se genera en la red. Esto no es significativo en un ambiente local en un equipo robusto de cómputo (Laptop, PC o Servidor), sin embargo en un dispositivo móvil, donde la comunicación es vía celular y el ancho de banda no es idóneo, esto provoca un caos entre las múltiples tareas que este debe gestionar:



En el caso de que el módulo Servidor proporcionara todos estos servicios mediante el manejo de comandos, el servidor es el que realizaría estas conexiones, sin embargo esto no es tan significativo si el servidor es un equipo de cómputo robusto y la conexión a la red es de un ancho de banda considerable. Y en el caso de que este proporcionara varios de los servicios la respuesta sería mucho más eficiente y rápida, en el siguiente ejemplo se muestra como el servidor, además de gestionar la recepción de paquetes provee el servicio de RDBMS y FTP.



Nótese como se libera el ancho de banda al distribuir la carga de servicios como locales y externos, también la diferencia entre el ancho de banda entre el servidor y el cliente, en este ejemplo se da a entender que el servidor tiene un servicio de Internet fijo, el cual tiene mayor capacidad y velocidad. Implícitamente esto convierte al módulo servidor en un intérprete de múltiples comandos que necesariamente debe gestionar y controlar, así como su seguridad y/o integridad.

IV.IV. Aplicación Cliente Servidor

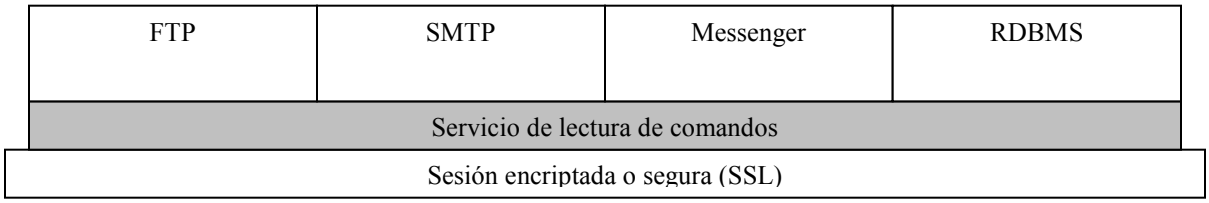
Para cada una de las aplicaciones cliente servidor antes mencionado en este capítulo, es necesario definir un ambiente seguro donde se identifique al usuario mediante una contraseña. Implicando una compleja gestión de seguridad, no obstante, en el caso de que se reutilizara el usuario para todos los servicios que provea el servidor, esta gestión se reduciría considerablemente. Para el caso de proveer un servicio antes mencionado, es necesario contar con el usuario y contraseña correcta.

- Para el FTP se debe contar con un usuario vigente en el Sistema operativo
- En el RDBMS, un usuario dado de alta por el Administrador de la base de datos
- En el caso del SMTP, una cuenta de correo activa

- Para el Messenger, un usuario registrado en este servicio, usualmente el mismo del SMPT
- Incuso, para el servidor receptor de paquetes o comandos, es el mismo caso

En el aspecto personal es casi imposible tener un solo usuario y contraseña para todos estos servicios, ya que las cuentas de correo y contraseñas se deben apegar al estándar de cada uno de los proveedores de servicio. No obstante, en una administración centralizada, como lo es una empresa si es posible, es decir a los empleados se les registra una sola vez y se configura cada uno de estos servicios de forma automática, esto implicaría que un usuario con una contraseña y cuenta de correo activa, tendría acceso a estos servicios, incluso desde un dispositivo con pocos recursos.

Para dar mayor seguridad a una sesión entre cliente y servidor, es de considerar una sesión segura, donde se genera un túnel encriptado a nivel TCPIP, sumando todos estos puntos hablaríamos de una sesión de varias capas, donde un solo usuario y una sola sesión del cliente al servidor facilitan la gestión de seguridad.



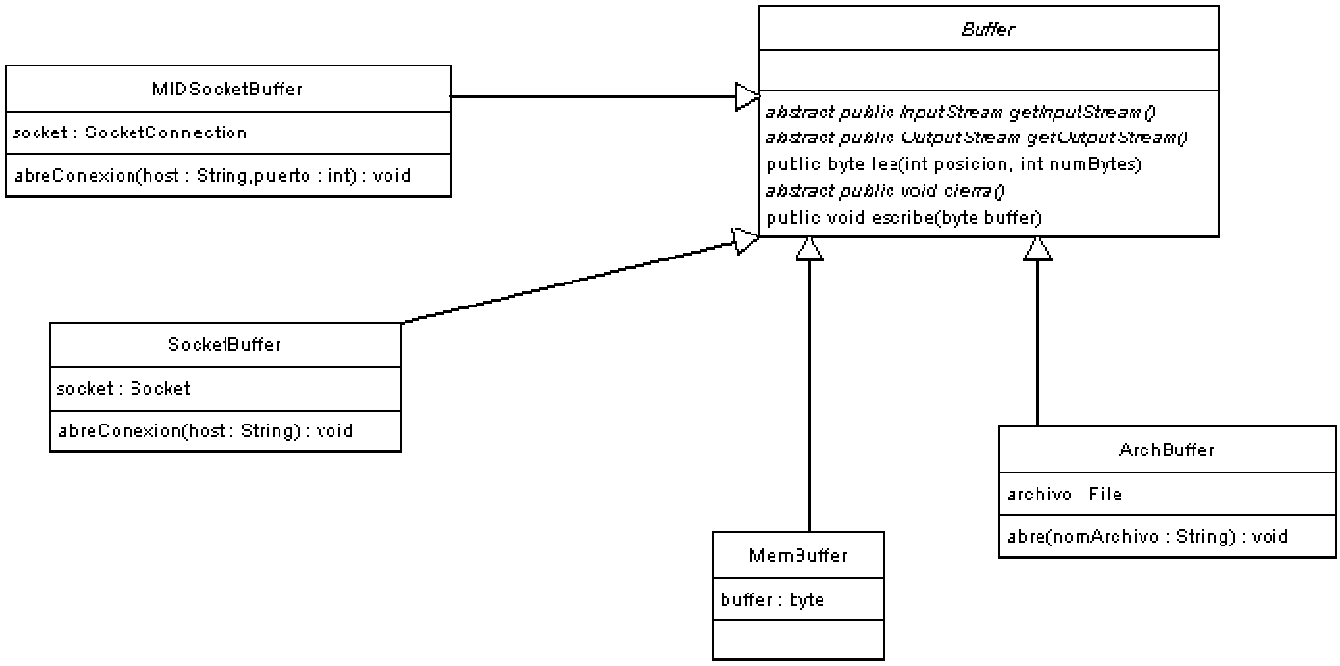
Por lo que se concluye que el modelo, a parte de ser una interfaz genérica para cualquier arquitectura cliente servidor, lleva consigo múltiples beneficios que deben considerarse.

IV.V. Implementación del modelo

Siendo Java un lenguaje de fácil portabilidad y aceptación en la mayoría de las plataformas conocidas y teniendo una colección de clases proporcionadas por la empresa Sun Microsystems, dueña del lenguaje de programación Java. Estas colecciones de clases son conocidas como API (por sus siglas en ingles Application Program Interfaz), que es un conjunto de convenciones Internacionales que definen como debe invocarse una determinada función de un programa desde una aplicación. Las API definidas por Sun Microsystems, son publicadas en la página de Internet http://java.sun.com/reference/api/ de la cual nos basamos en la especificación mas básica ya que esta será la mas aceptada por la mayoría de las plataformas.

La definición de clases más reducida de Java, es la denominada “Java 2 Micro Edition” (J2ME), publicada en la página http://java.sun.com/javame/reference/apis/jsr118/. En esta definición de clases y en todas las definiciones de Java, aparece un paquete de clases estándar para la lectura y escritura (java.io) de la cual podemos partir para la ejemplificación del modelo. Para este es indispensable pensar en una nueva API donde definamos las diferentes clases que definan el comportamiento del mismo basado en las clases disponibles en el paquete java.io, como son el InputStream, OutputStream, socketConnection, Socket y File.

Lo primero en que debemos pensar es en los diferentes accesos a los recursos de almacenamiento que soportaría, como se muestra en el siguiente diagrama de clases:



Donde la clase Buffer es una súper clase abstracta y por lo tanto las clases hijas (MIDSocket, SocketBuffer, MemBuffer y ArchBuffer) deben implementar los métodos abstractos, que en este caso se denotaran con letra cursiva. La finalidad de esta familia de clases es que al momento de definir un buffer (cualquiera), contenga un comportamiento y una identidad definida por la súper clase, es decir, que no importa de donde venga la información, si tiene un comportamiento como el del buffer, es posible manipular un archivo una memoria o un socket como si fuera una clase Buffer

Para la definición de una súper clase que solo define un comportamiento pero no la implementación de los métodos, es necesario declararlos abstractos, como se muestra en el siguiente ejemplo:

```
package buffer;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;


/**
 *
 * @author Fernando
 */
public abstract class Buffer {
  abstract public InputStream getInputStream() throws IOException;
```

```
  abstract public OutputStream getOutputStream() throws IOException;

  abstract public void cierra()throws IOException;

  public byte[] lee(int posicion, int numBytes) throws IOException{
    int           ch  = 0;
    int           i   = 1;
    InputStream   in  = getInputStream();
    byte[]        buf = new byte[numBytes];

    // Leemos hasta la posicion indicada
    while (ch != -1 && i < posicion){
      ch = in.read();
      i = i++;
    }
    i = 0;
    // Leemos el buffer
    while (ch != -1 && i < numBytes){
      ch = in.read();
      if (ch != -1)
        buf[i] = (byte)ch;
      i = i++;
    }
    // Cerramos el objeto de lectura
    in.close();
    return buf;
  }

  public void escribe(byte[] buf, int tam)throws IOException{
    OutputStream  out  =  getOutputStream();
    int     i    =  0;

    for (i = 0; i < tam; i++)
      out.write(buf[i]);
  }

}
```

Para las clases hijas como la **MIDSocketBuffer**, debe implementar los métodos abstractos de la clase Buffer para la transferencia de datos en una red accediendo a un host en determinado puerto. Utilizando las clases disponibles en una plataforma MID (Mobile Information Device) o dispositivos móviles publicadas para esta plataforma en http://java.sun.com/javame/reference/apis/jsr118/ en el paquete javax.microedition.io y haciendo uso de las clases estándares como son SocketConnection, InputConnection y OutputConnection.

```
package buffer;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import javax.microedition.io.*;
```

```
public class MIDSocketBuffer extends Buffer {

  public SocketConnection socket;

  public void abreConexion(String host, int puerto) throws IOException {
    socket = (SocketConnection) Connector.open("socket://" + host + ":" +
puerto);
  }

  public InputStream getInputStream()throws IOException{
    if (socket == null)
      return null;
    return socket.openInputStream();
  }

  public OutputStream getOutputStream()throws IOException{
    if (socket == null)
      return null;
    return socket.openOutputStream();
  }

  public void cierra()throws IOException {
    socket.close();
  }

}
```

La clase **SocketBuffer**, esta pensada, al igual que MIDSocketBuffer, para la transferencia de datos en una red para paliaciones con soporte de la API de Java estándar (J2SE), publicada en http://java.sun.com/javase/6/docs/api/ en el paquete java.net, con la idea de que acceda a un puerto y a una dirección en la red e implemente los métodos abstractos de la clase Buffer con la ayuda de otras clases como el atributo tipo Socket definido en el paquete java.net.

```
package buffer;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;

public class SocketBuffer extends Buffer {

  public Socket socket;

  public void abreConexion(String host, int puerto) throws IOException {
    socket = new Socket(host, puerto);
  }

  public InputStream getInputStream()throws IOException{
    if (socket == null)
      return null;
    return socket.getInputStream();
  }
```

```
  public OutputStream getOutputStream()throws IOException{
    if (socket == null)
      return null;
    return socket.getOutputStream();
  }

  public void cierra()throws IOException{
    socket.close();
  }

}
```

Nótese que este hereda de la clase Buffer tal como la Clase MIDSocketBuffer ya que la Clase Buffer esta pensada para definirse de una sola forma tanto para plataformas de java Micro Edition (J2ME)

La clase **MemBuffer** utiliza un arreglo de memoria ayudándose de un arreglo de bytes el cual emula un InputStream y OutputStream para el manejo de memoria, y al igual que su clase padre es posible implementar el mismo código en cualquier plataforma dado que las clases utilizadas son estándar y no es necesario crear una clase para cada plataforma.

```
package buffer;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public class MemBuffer extends Buffer{
      ByteArrayOutputStream  buffer;

      public void abre(int tamanio){
            buffer = new ByteArrayOutputStream(tamanio);
      }

      public byte[] getBuffer(){
            return buffer.toByteArray();
      }
      public void cierra() throws IOException {
            buffer = null;
      }

      public InputStream getInputStream() throws IOException {
            return new ByteArrayInputStream(buffer.toByteArray());
      }

      public OutputStream getOutputStream() throws IOException {
            return buffer;
      }

    }
```

La clase **ArchBuffer** es pensada para trabajar directamente en un archivo en base a las ya conocidas clases InputStream y OutputStream através de las clases FileInputStream y FileOutputStream, de la siguiente manera:

```
package buffer;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.File;
import java.io.InputStream;
import java.io.OutputStream;

public class ArchBuffer extends Buffer {

      File archivo;
      public void cierra() throws IOException {
            archivo = null;
      }

      public void abre(String nomArchivo)throws IOException{
            archivo = new File(nomArchivo);
            if (!archivo.exists())
                  archivo.createNewFile();
      }

      public InputStream getInputStream() throws IOException {
            return new FileInputStream(archivo);
      }

      public OutputStream getOutputStream() throws IOException {
            return new FileOutputStream(archivo);
      }

}
```

Esta clase solo es soportada por las aplicaciones en Java de edición estándar ya que para dispositivo pudiera variar el acceso a estos recursos por lo que es mejor implementar la clase correspondiente de cada dispositivo.

Hasta aquí se han dado ejemplos de cómo con la ayuda del lenguaje Java, es posible homologar los diferentes recursos de información básicos a los que un dispositivo puede acceder, ya que como todos heredan de la clase Buffer podemos transferir información de una manera sencilla. Por ejemplo, si quisiéramos enviar de un archivo podríamos hacerlo de forma transparente, no importando que contenga el buffer este pasara directamente al otro.

```
// Creamos el buffer destino en este caso la memoria RAM
MemBuffer bufDest = new MemBuffer();
// Creamos el buffer origen
archBuffer bufOri = new ArchBuffer();
```

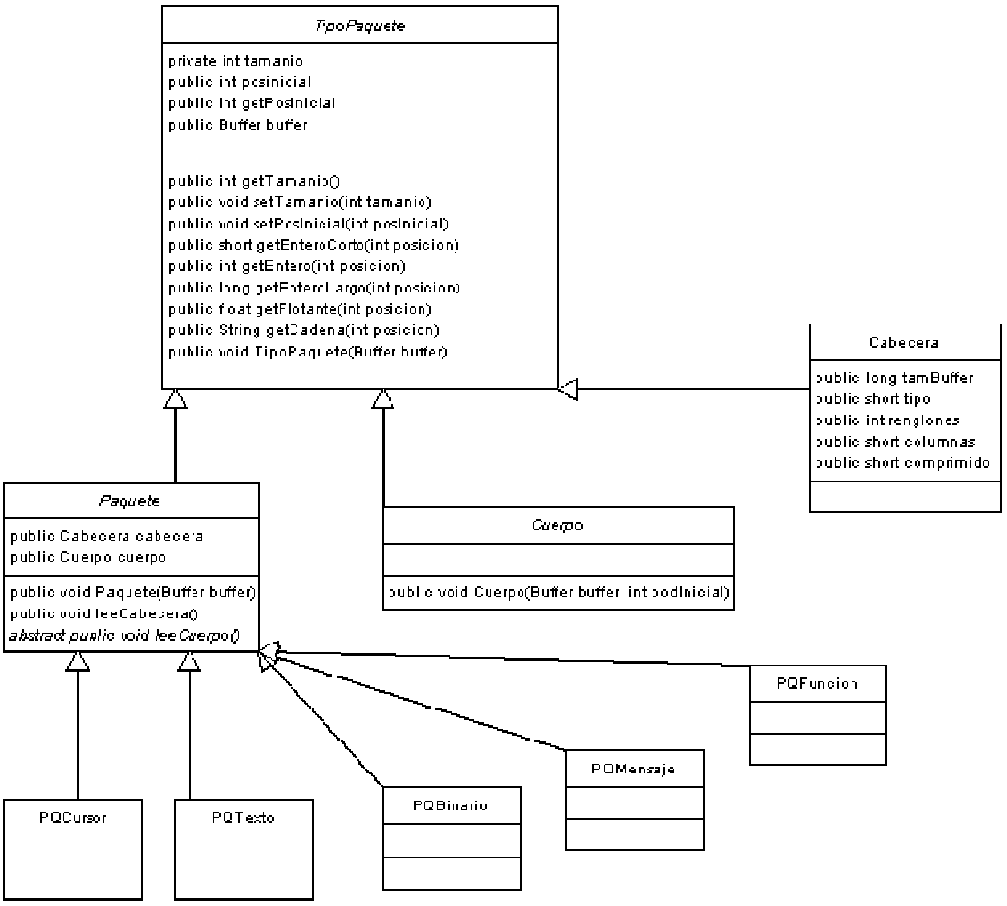
```
// Abrimos el archive
bufOri.abre("c:\\in\\origen.dat");

// Obtenemos los objetos de entrada y salida
inputStream in = bufOri.getInputStream();
outputStream out = bufDest.getOutputStream();

// Copiamos de un al otro
while (in.availible > 0){
  Out.write(in.read());
}
in.close();
out.close();
```

Mostrando así, que al tener un patrón como este, es posible portar las diferentes clases a diferentes situaciones o plataformas apuntando a diferentes recursos.

Una vez definidos los tipos de memoria o buffers y ver algunas de sus ventajas, es posible definir una nueva familia de clases que definen al mismo modelo como por ejemplo:



En este modelo se muestra como es posible diseñar y pensar diferentes formas para implementar el modelo, en este modelo por ejemplo, tenemos las siguientes definiciones:

TipoPaquete: Clase capaz de manipular un buffer y leer o escribir mediante éste, diferentes tipos de datos básicos definidos en el capítulo 3, como son el entero corto, entero, entero largo, flotante y cadena, siendo esta la herramienta idónea para trabajar con el paquete.

Cabecera: Clase que hereda el comportamiento de un TipoPaquete, capaz de leer y definir según se definió en el modelo, para cualquier paquete y tener disponibles los atributos de esta, como son el tamaño del buffer o cuerpo, el tipo de cabecera, el número de renglones y columnas para los paquetes que lo requieran y las banderas de encriptada y comprimido.

Cuerpo: Al igual que la clase Cabecera hereda de un TipoPaquete y la finalidad es tener el acceso a la información pura del datagrama o paquete mediante un buffer dentro de otro buffer.

Paquete: Clase abstracta la cual define la estructura básica de un paquete o datagrama formado por una cabecera y un cuerpo que al heredar de TipoPaquete contiene las funcionalidades básicas para acceder a la información de un buffer y aplicarlas a las clases de Cabecera y Paquete de modo que es un buffer el cual pudiera utilizar dos buffers para leer unos solo, es decir una cabecera y un cuerpo. Esta es definición abstracta de un paquete solo define el comportamiento básico, ya que cada paquete maneja de forma diferente su información según la necesidad de cada uno.

PQCursor: Clase que debe manejar de manera la cabecera y el cuerpo para administrar los registros y renglones de un cursor o tabla el cual pueda ser portable y a su vez fácil de leer para su consulta.

PQTexto: Clase que debe implementar las funciones necesarias para escribir o leer un texto guardado en un buffer cualquiera, haciendo uso de la cabecera y el cuerpo tal como se especifica en el capítulo 3.

PQBinario: Clase que deberá almacenar una secuencia bytes en el buffer y definir las dimensiones en la cabecera.

PQMensaje: Clase similar al PQTexto pero enfocado a párrafos cortos para mostrarse en una alerta en pantalla, incluso, una clase mas avanzada podría definir el título del texto, tipo de mensaje e ícono del mismo

PQFuncion: Clase capas de guardar y leer por medio de la cabecera y el cuerpo tanto el nombre de la función y sus parámetros

Por ejemplo, supongamos que se quiere crear un archivo con un paquete tipo texto, primero se crea el buffer de salida:

```
ArchBuffer buffer = new ArchBuffer("c:\\pruebas\\paquete.txt");
```

Posteriormente se crea el paquete para administrar el texto, con el buffer como parámetro:

```
PQTexto paquete = new PQTexto(buffer);
```

Donde el constructor de esta clase podría pasar por referencia a la cabecera y al cuerpo el buffer que este recibe y leer el buffer. Si al encontrar verificar que no se tiene nada en el bufer este podría definir una cabecera y un cuerpo sin información, quedando a la espera de un nuevo texto para guardarlo, por ejemplo:

```
paquete.agrega(“Inicio del texo”);
paquete.agrega(“FIN del texto”);
```

Donde la función agrega un texto al buffer del cuerpo el texto mas un salto de carro y a la cabecera incrementa el número de renglones de manera directa al buffer que en este caso es el archivo *paquete.txt*.

No debe pasarse por alto que estas deben ser generadas como se especifica en el capítulo 3 para lograr la funcionalidad de este modelo. Una vez demostrada este pequeño ejemplo, es posible definir cualquier tipo de información con este modelo e implementar una clase para el manejo de este y poderla transportar a las diferentes plataformas que soporte el lenguaje de programación.

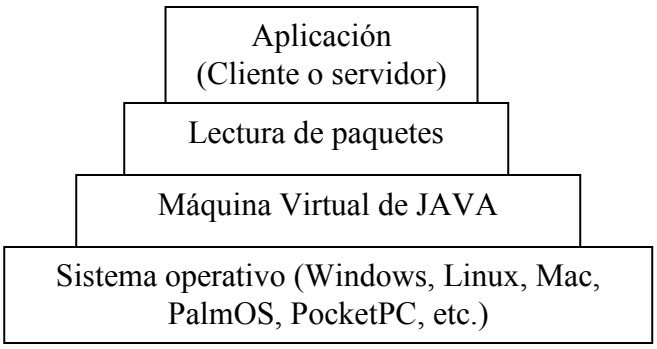
Conclusión

Conclusión

Actualmente contamos con herramientas de software muy sofisticadas capaces de soportar la interpretación de lenguajes como el XML, HTML o XHTML, para el envío de datos por la red, pero lograr que un equipo genere y/o compile estos códigos de cuarta generación es necesario tener un equipo con de alto rendimiento para su ejecución, dejando a los dispositivos de pocos recursos un problema de comunicación el cual muchas veces es solventado rehaciendo los procesos de almacenamiento básicos.

Debido a la duplicidad de tareas de mandar la información de una forma, de recibirla de una segunda y mostrarla en una tercera, se a propuesto el objetivo de encontrar una solución simple para homogeneizar la información de modo de que pueda ser leída, almacenada y transmitida de forma eficiente y directa, y como se muestra en este trabajo, se propone un modelo de almacenamiento y lectura directo de paquetes de datos; en base a una estructura estándar capaz de ser leída en la mayoría de las plataformas. Esta estructura se define como un paquete de datos que describe su contenido de manera compacta, y es capaz de ser almacenada, transmitida y leída de forma eficiente y directa. Otro punto importante es también el identificar un lenguaje de programación portable para poder reutilizar los métodos tanto de almacenamiento y lectura de estos paquetes.

Esta estandarización de código al igual que el manejo de paquetes de datos tomaría un rol de simplicidad de manejo de datos y reutilización de código en las diferentes aplicaciones, siendo una interfaz estándar en los sistemas de cómputo y los diferentes sistemas operativos, como se muestra en la siguiente figura, donde se representa cada una da las capas de una aplicación por una barra.



Una vez definidos el tipo de aplicaciones, y descrito los posibles paquetes que se pudieran construir, es posible determinar que el modelo ha cumplido con el objetivo general de encontrar una forma eficiente y simple de homologar la información en un paquete como se demostró en el capítulo cuatro donde se muestran algunos ejemplos de su aplicación en el lenguaje Java. En cuanto a los objetivos específicos podemos encontrar que:

1. Encontrar una forma simple para la compactar la información de forma que esta se identifique por si misma:

Al definir una información en un paquete de datos o datagrama el cual se describe por si mismo mediante su cabecera, se define una estructura compacta de fácil lectura.

2. Almacenar de manera directa y eficiente cualquier tipo de información

   En la implementación de clases abstractas como el InputStream y OutputStream en el modelo es posible el almacenamiento eficiente y directo de diferentes orígenes y diferentes tipos de información.

3. Transferir de manera eficiente y directa la información, sin necesidad de transformarla a diferentes formas: código, estructuras de datos, cadenas limitadas por caracteres, etc.

   Al definir una Buffer abstracto, y al definir clases con capacidad de transferir estructura definida de manera simple, entre ellas, podemos concluir que al transferir una estructura como esta no es necesario pasarla a otro tipo de estructura ya que esta por ser abstracta es pasada de una clase a otra e incluso de por una red de manera simple.

4. Tener la capacidad de navegar o leer directamente la información sin necesidad de cargar a una estructura de datos.

   Al momento de tener una cabecera como la que se presenta en el modelo es posible implementar métodos que nos den acceso a la información, como puede ser el paquete de texto, pudiera leerse directamente cada renglón con un método capaz de indexar la posición de cada renglón y posteriormente ir a la posición y extraer el renglón.

5. Reutilizar métodos de lectura y escritura de información en las diferentes plataformas

   Una vez definida una clase con recursos que utilizan la mayoría de las plataformas es posible portarla a diferentes plataformas como es la clase Buffer que se describió en capítulo 4, el cual utiliza recursos soportados por J2SE y por J2ME.

6. Encontrar un lenguaje de programación portable de modo que muchos de los métodos y bloques de código sean reutilizados en diferentes plataformas.

   En el capítulo 4, se muestra como JAVA es el lenguaje que mas se adapta a la implementación del modelo por su portabilidad y recursos estándares.

   Con esto concluimos que el modelo junto con JAVA cumple con los objetivos

GLOSARIO

Aplicación. Son los programas con los cuales el usuario final interactúa, es decir, son aquellos programas que permiten la interacción entre el usuario y la computadora. Esta comunicación se lleva acabo cuando el usuario elige entre las diferentes opciones o realiza actividades que le ofrece el programa.

Archivo. o **fichero informático.** es una entidad lógica compuesta por una secuencia finita de bytes, almacenada en un sistema de archivos ubicada en la memoria secundaria de un ordenador. Los archivos son agrupados en directorios dentro del sistema de archivos y son identificados por un **nombre de archivo**. El nombre forma la identificación única en relación a los otros archivos en el mismo directorio

ASCII. Sigas en ingles de American Estandar Code for Information Interchange. O bien es el código estándar americano para el intercambio de información para la representación de símbolos básicos de la computadora, alfabeto, números, signos numéricos entre otros.

Bit. Su nombre se debe a la contracción de Binary Digit, es la mínima unidad de información y puede ser un cero o un uno

Byte. Es la también conocida como el octeto, formada por ocho bits, que es la unidad básica, las capacidades de almacenamiento en las computadoras se organiza en potencias de dos, 16, 32, 64

Bytes. Secuencia continua de uno o varios Byte, definido en un medio físico, la cual representa un o varios Datos.

Campo. Término común que se le ha dado a un atributo de una entidad y su función es identificar los elementos del dato que utiliza el sistema actual o entidad, la forma en que se define es por un nombre, tipo y su tamaño (**J2EE pp107**).

Carpeta o directorio. Tipo de fichero en el que se organizan otros ficheros o archivos de forma jerárquica arbórea. En la World Wide Web (WWW), un directorio es una guía temática, estructurada en temas generales y sub temas. En los sistemas de ordenador, el directorio se organiza en archivos. Por ejemplo, en Windows se les denomina carpetas. En una red de ordenadores, se refiere al conjunto de información contenida, como contraseñas, identidades de los usuarios y, a veces, incluso a las redes a las que tienen acceso.

CASE. Se refiere a la ingeniería de software asistida por computadora, El crecimiento de la industria CASE reviste una particular importancia para el desarrollo de software. Las herramientas CASE son el equivalente para la industria de las herramientas CAD

(Diseño Asistido por Computadoras), para actividades como la organización de un circuito y el diseño de un chip (**ANADISOO**).

Casettes. También llamado **cinta casette**, o **casete**. Es un medio de almacenamiento de datos magnético y analógico. Se compone de una tira de plástico de grosor fino para que sea flexible y alargada tanto que puede llegar a tener varios metros de longitud. Esta tira de plástico lleva una fina capa de material magnético que guardará los datos a orientarse los polos magnéticos gracias a la acción de un cabezal lector/escritor. Esta cinta está protegida por una caja rectangular y plana de plástico que tiene dos bobinas con capacidad de giro donde están unidas a ambos extremos de la cinta (la tira de plástico) y a su vez está embobinado en alguna de las dos.

CHIP. También conocido como circuito integrado, es un pastilla en la que se encuentran todos o casi todos los componentes electrónicos necesarios para realizar alguna función. Estos componentes son transistores en su mayoría, aunque también contienen resistencias, diodos, condensadores, etc.

Código ASCII. American **S**tandard **C**ode for **I**nformation **I**nterchange (Código Estadounidense Estándar para el Intercambio de Información) es un código de caracteres basado en el alfabeto latino tal como se usa en inglés moderno y otras lenguas occidentales. Creado aproximadamente en 1963 por el Comité Estadounidense de Estándares (ASA) como una refundición o evolución de los conjuntos de códigos utilizados entonces en telegrafía. Más tarde, en 1967, se incluyen las minúsculas y se redefinen algunos códigos de control para formar el código conocido como **US-ASCII**

Compilar Traducir a lenguaje máquina un programa escrito en lenguaje simbólico (**PEQLAROUS**).

Computadora. Una **computadora** (Hispanoamérica) u **ordenador** (España) es un dispositivo electrónico compuesto básicamente de un procesador, memoria y dispositivos de entrada/salida (E/S) . La característica principal de la computadora, respecto a otros dispositivos similares, como una calculadora no programable, es que con él se pueden realizar tareas muy diversas, cargando distintos programas en la memoria para que los ejecute el procesador. Siempre se busca optimizar los procesos, ganar tiempo, hacerlo más fácil de usar y simplificar las tareas rutinarias.

CPU (Central Processing Unit). Parte de una computadora que interpreta y porta las instrucciones contenidas en el software.

Dato. Significa simplemente hechos, entidades independientes sin evaluar, por lo que puede pueden ser o no numéricos.

Directorio. Ver la definición de carpeta.

Disco duro: Es un dispositivo de almacenamiento, que nació como evolución del **Disco Flexible**. Tiene una capacidad mucho mayor (hoy en día es habitual que pasen de los 40 Gb) y es mucho más rápido, pero no está diseñado para ser llevado de un sitio a otro, sino para permanecer dentro del ordenador (salvo algunas pocas excepciones, que sí son portables). También se le llama "disco fijo", y (hoy en desuso) "disco winchester".

Disco Flexible. Es una pieza plástica redonda y plana cubierta con óxido de hierro y encerrada en una cubierta de plástico o vinil.

La unidad contiene un eje que hace girar el disco y cabezas de lectura/escritura que se pueden mover hacia adentro o hacia afuera mientras el disco gira para ubicarse en cualquier lugar de la superficie del disco.

Ejecutar. El sentido que se le da a esta palabra, en el contexto de un Sistema Operativo, es el de enviar a un procesador un segmento de código, conocido también como programa.

EEPROM. (Electrically-erasable programmable read-only memory*)*. O ROM electrónicamente borrable programable, es un chip de memoria que retiene sus datos cuando se apaga la energía. Puede ser programado, borrado y reprogramado eléctricamente. Hay un límite de veces en que una EEPROM puede ser borrada y reprogramada, de unas 100.000 a 1.000.000 de veces. Puede ser leída un número ilimitado de veces.

Estructura de datos. En programación, una estructura de datos es una forma de organizar un conjunto de datos elementales con el objetivo de facilitar su manipulación. Un dato elemental es la mínima información que se tiene en un sistema

GSM (Global System for Mobile Communications). Es el sistema de comunicaciones estándar para teléfonos móviles más popular del mundo. Gracias a la ubicuidad del estándar GSM, es posible la cobertura internacional llamada comúnmente “Contrato de cobertura”, entre los operadores del teléfono móvil. GSM difiere significativamente ante sus predecesores tanto señal y canales de conversación digitales, visto como el sistema de teléfonos móviles de segunda generación (2G).

Hardware. El hardware se refiere a todos los componentes físicos (que se pueden tocar) de la computadora: discos, unidades de disco, monitor, teclado, mouse, impresora, placas, chips y demás periféricos. En cambio, el software es intocable, existe como ideas, conceptos, símbolos, pero no tiene sustancia. Una buena metáfora sería un libro: las páginas y la tinta son el hardware, mientras que las palabras, oraciones, párrafos y el significado del texto son el software. Una computadora sin software sería tan inútil como un libro con páginas en blanco.

HTML. Lenguaje de marcado de hipertexto, lenguaje que se utiliza para crear páginas Web (**J2EE** pp. 259)

iDEN (**Integrated Digital Enhanced Network**). Es una tecnología de comuniciaciones móviles, desarrollada por Motorola, el cual provee a los usuarios el beneficio de una estructura de radio y teléfono celular. Nextel es el mayor detallista de servicios iDEN en Estados Unidos de América. iDEN localiza mas usuarios en espacio espectral dado, comparado con sistemas de celular análogo, por el uso de múltiples divisiones de acceso a simultáneo (Time Division Multiple Access (TDMA)).

Información. Es un conjunto ordenado de datos los cuales pueden recuperarse de acuerdo a las necesidades del usuario

Interfaz. La idea fundamental en el concepto de interfaz es el de mediación, entre hombre y máquina. La interfaz es lo que "media", lo que facilita la comunicación, la interacción, entre dos sistemas de diferente naturaleza, típicamente el ser humano y una máquina como el computador. Esto implica, además, que se trata de un sistema de traducción, ya que los dos "hablan" lenguajes diferentes: verbo-icónico en el caso del hombre y binario en el caso del procesador electrónico.

Internet. Red telemática internacional, que procede de una red militar norteamericana (Arpanet creada en 1969). Interconexión de múltiples redes que utilizan un mismo protocolo de comunicación (TCP-IP). Los servicios que ofrecen son la consulta de información (sitios Web), la mensajería electrónica, el comercio electrónico etc. (**PEQLAROUS**)

JAVA. Lenguaje de programación encargado de compilar el código fuente Java en bytecode que después es ejecutado por la máquina virtual Java (Java Virtual Machina o JVM), interpretando al momento de ejecución (**J2EE pp. 11**)

Megabytes. El **Megabyte** (**MB**) es una unidad de medida de cantidad de datos informáticos. Es un múltiplo binario del byte, que equivale a 2^{20} (1 048 576) bytes, traducido a efectos prácticos como 10^6 (1 000 000) bytes.

Microcomputadora. Una computadora pequeña (es lo mismo que computadora de escritorio, computadora personal, PC.)

Mainframe. Con este nombre se indica un gran ordenador que es capaz de soportar simultáneamente a miles de usuarios

Ordenador. Nombre original que se le dio a la calculadora, posteriormente a una computadora.

Periférico. Elemento de un sistema de tratamiento de la información distinto a la unidad central que sirve para memorizar datos o comunicar con el exterior. (**PEQLAROUS**)

Procesador. Es un subsistema de un sistema de procesamiento de información que procesa o cambia la información recibida de un objeto de alguna manera antes de transmitirla a un observador. De igual modo, el subsistema procesador de un sistema de procesamiento de datos procesa la información recibida después de haber sido

codificado en datos por el subsistema de entrada. Estos datos son procesados entonces por el subsistema de procesamiento antes de ser enviados como datos al subsistema de salida donde se descodifica para volver a convertirse en información.

Proceso. Un proceso es un concepto manejado por el sistema operativo que consiste en el conjunto formado por:

- Las instrucciones de un programa destinadas a ser ejecutadas por el microprocesador.
- Su estado de ejecución en un momento dado, esto es, los valores de los registros de la CPU para dicho programa.
- Su memoria de trabajo, es decir, la memoria que ha reservado y sus contenidos.
- Otra información que permite al sistema operativo su planificación.

Esta definición varía ligeramente en el caso de sistemas operativos multihilo, donde un proceso consta de uno o más *hilos*, la memoria de trabajo (compartida por todos los hilos) y la información de planificación. Cada hilo consta de instrucciones y estado de ejecución.

Programa. Un programa, o también llamado programa informático, programa de computación o programa de ordenador, es simplemente un conjunto de instrucciones para una computadora. Las computadoras necesitan de los programas para funcionar, y un programa no hace nada a menos que sus instrucciones sean ejecutadas por el procesador. Un programa se puede referir tanto a un programa ejecutable como a su código fuente, el cual es transformado en un ejecutable cuando es compilado.

Protocolo. Los protocolos son reglas de comunicación que permiten el flujo de información entre computadoras distintas que manejan lenguajes distintos, por ejemplo, dos computadores conectados en la misma red pero con protocolos diferentes no podrían comunicarse jamás, para ello, es necesario que ambas "hablen" el mismo idioma, por tal sentido, el protocolo TCP/IP fue creado para las comunicaciones en Internet, para que cualquier computador se conecte a Internet, es necesario que tenga instalado este protocolo de comunicación.

- **Sintaxis**: Se especifica como son y como se construyen.
- **Semántica**: Que significa cada comando o respuesta del protocolo respecto a sus parámetros/datos.
- **Procedimientos de uso de esos mensajes**: Trato de errores y banderas del mismo protocolo

RAM. Es el acrónimo inglés de *Random-Access Memory* (memoria de acceso aleatorio). Memoria de semiconductor en la que se puede tanto leer como escribir. Se trata de

una memoria volátil, es decir, pierde su contenido al desconectar la energía eléctrica. Se utilizan normalmente como memorias temporales para almacenar resultados intermedios y datos similares no permanentes.

Registro. Un registro es un conjunto de campos que contienen los datos que pertenecen a una misma repetición de entidad. Se le asigna automáticamente un número consecutivo (número de registro) que en ocasiones es usado como índice aunque lo normal y práctico es asignarle a cada registro un campo clave para su búsqueda.

En informática, y concretamente en el contexto de una base de datos relacional, un registro (también llamado fila o tupla) representa un ítem único de datos implícitamente estructurados en una tabla. En términos simples, una tabla de una base de datos puede imaginarse formada de *filas* y *columnas* o campos. Cada fila de una tabla representa un conjunto de datos relacionados, y todas las filas de la misma tabla tienen la misma estructura.

ROM. **(Read-only memory**). Es usado como medio de almacenamiento de una computadora. Debido a que en este medio no es posible escribir datos fácilmente, esto es usado principalmente para la distribución oculta de algún firmware (software que es muy exclusivo para algún dispositivo, y no requiere de actualizaciones frecuentes)

Shells (Intérprete de comandos). Es la arte fundamental de un sistema operativo encargada de ejecutar las órdenes básicas para el manejo del sistema. También se denomina *shell*. Suelen incorporar características tales como control de procesos, redirección de entrada/salida y un lenguaje de órdenes para escribir programas por lotes

Sistema Operativo. Conjunto de programas o software destinado a permitir la comunicación del usuario con un ordenador y gestionar sus recursos de manera cómoda y eficiente. Comienza a trabajar cuando se enciende el ordenador, y gestiona el hardware de la máquina desde los niveles más básicos.

Sistema de Archivos. Consta de tipos de datos abstractos, que son necesarios para el almacenamiento, organización jerárquica, manipulación, navegación, acceso y consulta de datos.

Software. También conocido como programático y aplicación informática, es la parte lógica del ordenador, esto es, el conjunto de instrucciones (programas) que puede ejecutar el hardware para la realización de las tareas de computación a las que se destina. Así podemos decir que el software es el conjunto de instrucciones "programas" que permiten la utilización de un periférico (hardware).

Súper computadora. También conocida como computadora central o *mainframe* es una computadora grande, potente y costosa usada principalmente por una gran compañía para el procesamiento de una gran cantidad de datos.

La capacidad de una computadora central se define tanto por la velocidad de su CPU como por su gran memoria interna, su alta y gran capacidad de almacenamiento externo, sus resultados en los dispositivo E/S rápidos y considerables, la alta calidad de su ingeniería interna que tiene como consecuencia una alta fiabilidad y soporte técnico caro pero de alta calidad. Una computadora central puede funcionar durante años sin problemas ni interrupciones y las reparaciones del mismo pueden ser realizadas mientras está funcionando

Teclado. Un teclado de computadora es un periférico, físico o virtual (por ejemplo teclados en pantalla o teclados láser), utilizado para la introducción de órdenes y datos en una computadora. Tiene su origen en los teletipos (dispositivo telegráfico) y las máquinas de escribir eléctricas, que se utilizaron como los teclados de los primeros ordenadores y dispositivos de almacenamiento (grabadoras de cinta de papel y tarjetas perforadas). Aunque físicamente hay una gran cantidad de formas, se suelen clasificar principalmente por la distribución de teclado de su zona alfanumérica, pues salvo casos muy especiales es común a todos los dispositivos y fabricantes (incluso para teclados árabes y japoneses).

UEs (User Equipment). Nombre dado en sistema de teléfonos móviles UMTS (3G) al equipo del usuario. Este corresponde burdamente a una estación móvil (Mobile Station) en sistemas GSM, conocida como MS.

UMTS (Universal Mobile Telecommunications System). Tecnología de telecomunicaciones GSM, para teléfonos móviles de tercera generación. El servicio asociado a esta tecnología provee la habilidad de transferir tanto como voz y datos (una llamada telefónica) y un dato (como descargar información e intercambiar correos electrónicos o mensajes instantáneos)

UNICODE. Es la norma de codificación de caracteres. Su objetivo es asignar a cada posible carácter de cada posible lenguaje un número y nombre único, a diferencia de la mayor parte de los juegos ISO como el ISO-8859-1, que sólo definen los necesarios para un idioma o zona geográfica.

USIM (Universal Subscriber Identity Module). El módulo subscriptor de identificación universal, es una aplicación que se ejecuta sobre una tarjeta inteligente usada para operaciones de red en la carga de un ambiente inicial (Home Environment (HE)). El USIM almacena la llave secreta asignada junto con el centro de autentificación (Authentication Center (AuC).

Bibliografía

[1] Sistemas de Bases de Datos Administración y Uso
De Alice Y. H. Tsai
Editorial Prentice Hall.

[2] Redes Locales 2ª Edición
De José Luis Raya y Cristina Raya
Editorial Alfaomega – RA-MA

[3] Devéloper's Guide Borland Delphi 4 for Windows 95 & Windows NT

Inprise Corporation, 100 Enterprise Way

Impresa en Estados Unidos de América

[4] Estructuras de datos Segunda edición
Cairó Guardati
Editorial McGrawHill

[5] Análisis y Diseño Orientado a Objetos
James Martin y James J. Odell
Editorial Prentice Hall

[6] Manual de referencia J2EE

Jim Keogh

Mc Graw Hill

[7] Pequeño Larouse Ilustrado
Diccionario ilustrado 2006

URLS

[U1]: http://www.monografias.com/trabajos16/memorias/memorias.shtml
28 de Abril del 2005, 10:50 hrs.

[U2]: http://www.monografias.com/trabajos16/memorias-auxiliares/memorias-auxiliares.shtml
29 de Abril del 2005, 18:10 hrs.

[U3]: http://es.wikipedia.org/wiki/Sistema_de_archivos
02 de Mayo del 2005 17:22 hrs.

[U4]: http://en.wikipedia.org/wiki/SIM_card
02 de Mayo del 2005 17:52 hrs.

[U5]: http://es.wikipedia.org/wiki/Memoria_flash
03 de mayo del 2005 13:54 hrs.

[U6]: http://es.wikipedia.org/wiki/Criptograf%C3%ADa
12 de Mayo del 2005 12:03 hrs.

[U7]: http://www.redsegura.com/Temas/CRhistoria.html
12 de Mayo del 2005 12:08 hrs.

[U8]: http://en.wikipedia.org/wiki/Compact_disk
19 de Mayo del 2005 11:08 hrs.

[U9]: http://en.wikipedia.org/wiki/CD-ROM
19 de Mayo del 2005 11:55 hrs.

[U10]: http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html
24 de Mayo del 2005

[U11]: http://en.wikipedia.org/wiki/Java_programming_language
31 de Mayo del 2005

[U12]: http://www.ctisa.com/diccionario.htm

29 de Abril de 2005, 18:18 hrs.

[U13]: http://es.wikipedia.org/wiki/Portada

29 de Abril de 2005, 17:35 hrs.

[U14]: http://en.wikipedia.org/wiki/

2 de Mayo de 2005, 16:28 hrs.

[U15]: http://www.monografias.com/trabajos6/inus/inus.shtml
3 de mayo del 2005, 14:25 hrs.

[U16]: http://www.laopinion.com/glossary/m.html
3 de mayo del 2005, 14:35 hrs.