29

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES.

CAMPUS ARAGÓN

CONFIGURACIÓN DE UN SISTEMA DE ENTREGA DE DOCUMENTOS LATEX EN INTERNET, IMPLEMENTANDO UN MECANISMO DE ÚNICO-ENVIÓ MÚLTIPLE RECEPCIÓN CON EL LENGUAJE DE PROGRAMACIÓN JAVA

T E S I S

QUE PARA OBTENER EL TITULO DE

INGENIERO EN COMPUTACIÓN

P R E S E N T A :

JOSÉ EDUARDO MARTÍNEZ CORDERO

ASESOR:
ING. DONACIANO JIMÉNEZ VÁZQUEZ

MÉXICO 2002

Con profundo amor

A mis Padres.

Quienes son el motivo, la inspiración y objetivo de todos mis esfuerzos.
A Ustedes, quienes me han dado la vida y Su vida, sin pedir nada a cambio.
A Ustedes, que siempre me han brindado su amor, apoyo, confianza y consejo.
A Ustedes, por quienes agradezco profundamente a Dios por tenerlos como Padres.

Toda la vida mi único objetivo ha sido hacerlos sentir orgullosos y de esta manera pagarles de cierta forma todos los esfuerzos y sacrificios que han hecho por mi; porque al ver una sonrisa de satisfacción en sus rostros es la mejor recompensa que pueda tener; porque esta ha sido la única forma de poder demostrarles que también estoy orgulloso de ser Su Hijo y decir también que Ustedes son mis Padres.

Sólo quiero decirles que el esfuerzo y la espera no han sido en vano.
TODO va por Ustedes.

Espero haber cumplido con el objetivo.

Su Hijo que los respeta y ama.

Eduardo

A mis hermanos,
Jorge y Gustavo.

Para que siempre busquen superarse y siempre, **siempre** sean los mejores en lo que hagan. Sirva pues, este esfuerzo como un aliciente para que se superen y también como ejemplo de que si yo pude lograrlo, Ustedes con muchísima más razón.

A Cris
Por comprenderme, por apoyarme, por tu empuje, por los momentos vividos, simplemente por ser tú.
Gracias por estar a mi lado. TQM

A las familias Cordero Núñez y Martínez Casarrubias
Por haberme alentado siempre; porque nunca de nadie de Ustedes recibí un mal consejo, sino al contrario siempre estuvieron ahí, impulsándome a seguir adelante.
Gracias por ser mi Familia.

Y por último, a mi *Alma Mater*

A la UNAM, mi querida Universidad.
Por haberme abierto sus puertas y de esta manera ampliar mis horizontes.
Porque me ha permitido conocer gente que ha cambiado mi vida.
Porque nunca, nunca cruzó en mi mente la idea de estudiar en otra Institución que no fuera ella, la Universidad.
Porque soy orgullosamente Puma.
Porque siempre pondré mi mejor empeño en poner su nombre en alto y así demostrar que sus egresados somos los mejores.

Por todo lo anterior, sólo me queda gritar un ¡Gooooooooya! en su honor y con todo el corazón.

¡ MEXICO, PUMAS, UNIVERSIDAD !

El árbol que llena los brazos de un hombre, crece de un pequeño tallo.
La torre de nueve pisos se hizo uniendo piedra tras piedra.
El viaje de mil leguas comienza por un solo paso.

Lao-Tse

Indice

Capítulo 1. Introducción 1

Capítulo 2. Antecedentes 3

2.1 Familia de protocolos TCP/IP 4
2.1.1 El protocolo IP
2.1.1.1 Direcciones IP
2.1.1.2 Clasificación de las direcciones IP
2.1.1.3 Reglas especiales de direccionamiento IP
2.1.2 El protocolo TCP
2.1.1 La interfaz socket
2.1.3.1 El paradigma de E/S de UNIX
2.1.3.2 Adición de soporte E/S en red a través de sockets
2.1.3.3 Comunicación en red a través de sockets

2.2 Modelo Cliente - Servidor 15
2.2.1 Características del cliente
2.2.2 Características del servidor

2.3 Programación orientada a objetos 20
2.3.1 Objetos
2.3.2 Clases
2.3.3 Abstracción de datos, Herencia y Polimorfismo
2.3.3.1 Abstracción de datos
2.3.3.2 Herencia
2.3.3.3 Polimorfismo
2.3.4 Reutilización de código

2.4 El lenguade de programación Java 29
2.4.1 Comparando a Java y C++
2.4.2 Independencia de plataforma
2.4.3 Características de Java
2.4.4 Applets

Capítulo 3. Mecanismos de entrega multipunto 37

3.1 Entrega por difusión (broadcast) 38
3.1.1 Direcciones de difusión IP
3.1.1.1 Difusión limitada
3.1.1.2 Difusión a las subredes

3.2 Entrega por multidifusión (multicast) 40
3.2.1 Multidifusión IP: direcciones de clase D

3.2.2 Grupos de multidirusión IP
3.2.3 Envío y recepción de paquetes IP multidifusión
3.2.3.1 Campto Time to Live (TTL) del datagrama IP
3.2.4 Protocolo IGMP
3.2.1 Ruteo multidifusión
3.2.2 Componentes de una implementación de IP multidifusión pura
3.2.3 La red multicast MBONE

Capítulo 4. El programa LaTeX 54

4.1 TeX y LaTeX 55

4.2 Características de LaTeX 55

4.3 Elaboración de documentos 57
4.3.1 Caracteres especiales
4.3.2 Comandos
4.3.3 Estructura general de un documento en LaTeX
4.3.4 Ambientes
4.3.5 Estilo de documentos
4.3.6 Tipos de fuentes y tamaños
4.3.7 Texto matemático
4.3.7.1 Fuentes y simbolos matemáticos
4.3.7.2 Ambientes matemáticos
4.3.7.3 Expresiones simples y ecuaciones
4.3.8 Arreglos

Capítulo 5. Propuesta de Desarrollo 73

5.1 Visualización y distribución de documentos matemáticos en Internet 74
5.1.1 Texto ASCII
5.1.2 Trabajando con ASCII y HTML
5.1.3 Trabajando con imagenes y HTML

5.2 El programa IDVI para presentación de documentos LaTeX 77
5.2.1 Instalación de IDVI
5.2.2 Preparación de un documento para vista con IDVI
5.2.3 Módulos que componen el código fuente de IDVI

5.3 Programación multitarea 87
5.3.1 Threads
5.3.2 Sincronización de threads
5.3.2.1 Monitores
5.3.2.2 Comunicación entre threads

5.4 Programando aplicaciones multicast con Java 92
5.4.1 Transmisión de paquetes multicast
5.4.2 Recepción de paquetes multicast
5.5 Modificación a los módulos de IDVI para que soporten Multicast 94
5.5.1 Módulos multicast
5.5.1.1 Clase Java para el cliente Multicast: IDVIMulticastCliente
5.5.1.2 Clase Java para el servidor Multicast: IDVIMulticastServidor

Capítulo 6. Caso de Aplicación. 105

6.1 Vic y Vat 106
6.1.1 Vat
6.1.2 Vic
6.2 Preparación del curso 109
6.2.1 Elaboración de documentos

Capítulo 7. Perspectivas a futuro 117
7.1 Aplicaciones multimedia
7.2 Tecnología "push"
7.3 Panorama actual

Capítulo 8. Conclusiones 122

Bibliografía 126

Capítulo 1
Introducción

Uno de los conceptos que están adquiriendo actualmente gran auge gracias a Internet es el de la **educación a distancia**. Muchas universidades alrededor del mundo han empezado a realizar proyectos experimentales y algunas otras ya proporcionan servicios más sofisticados. Sin embargo, debido principalmente a que es un campo relativamente nuevo en Internet, uno de los problemas a los que se enfrenta la educación a distancia es el de hacer llegar la misma información a un grupo de alumnos que puede estar disperso en distintas localidades, ya sea dentro del mismo campus universitario o fuera de él (tal vez en cualquier parte del mundo).

Las tecnologías de red tienen que enfrentar hoy en día este tipo de retos: el de la transmisión efectiva de datos hacia múltiples destinos evitando saturar la red con información repetida, lo que ocasiona que se haga un uso inadecuado de los recursos.

Actualmente el mecanismo más utilizado para comunicar computadoras por Internet es el de transmisión punto-a-punto, en el cual la comunicación se da solamente entre dos nodos de la red, el cual es ineficiente para este tipo de necesidades.

El objetivo de esta tesis es presentar una alternativa funcional y de fácil implementación para este tipo de retos, es decir el desarrollo de una aplicación de distribución simultánea de documentos desde un solo punto central hacia múltiples destinos, maximizando la utilización de recursos de red y la eficiencia de la entrega.

La aplicación esta pensada principalmente para profesores del área físico-matemáticas (aunque puede ser utilizada prácticamente por profesores de cualquier otra área) ya que permite la distribución de documentos LaTeX, los cuales contienen simbología matemática. Con esta aplicación el profesor tiene el control sobre que información se les envía a los alumnos y cuando debe enviárselas, y además ésta llegara a todos los alumnos, independientemente del lugar en el que se encuentren.

Para facilitar la lectura de la tesis, se muestra a continuación un resumen de cada capítulo.

Capítulo 2. Antecedentes. En esta capítulo se realiza una revisión de los conceptos teóricos básicos para el desarrollo de esta tesis.

Capítulo 3. Mecanismos de entrega de paquetes. En este capítulo se examinan los mecanismos de entrega de paquetes en redes.

Capítulo 4. El programa LaTeX. Este capítulo esta dedicado a presentar detalles de LaTeX, el software que permite la creación de documentos que contengan texto matemático.

Capítulo 5. Propuesta de desarrollo. Se conjuntan los temas tratados en los capítulos anteriores para dar forma al proyecto de tesis: la entrega de documentos LaTeX a múltiples destinos simultáneamente.

Capítulo 6. Caso de aplicación. Se muestra un ejemplo de la utilidad del proyecto, enfocado a la educación a distancia.

Capítulo 7. Perspectivas a futuro. Que se espera en un futuro no muy lejano para la distribución de información a grandes cantidades de destinos optimizando los recursos de red y de una forma efectiva.

Capítulo 8. Conclusiones.

Capítulo 2

Antecedentes

El trabajo de desarrollar software es cada día más difícil. Hoy en día lo que se espera de una aplicación es que sea gráfica, fácil de usar, robusta, trabaje en red y que esté lista para funcionar la más pronto posible.

Para cumplir con estos requisitos, el programador debe tener conocimientos detallados en muy diversas áreas como manejo de bases de datos, protocolos de comunicación, interoperabilidad entre sistemas operativos y multimedia. Ahora se espera que se pongan todas estas cosas juntas y se programen nuevas aplicaciones que trabajen de forma distribuida en múltiples sistemas operativos corriendo en diversos procesadores. Además, todo debe trabajar en forma eficiente usando los recursos actuales de Internet.

Es por eso que antes de comenzar con el desarrollo del proyecto, es necesario explicar algunos conceptos importantes relacionados con el mismo. Es necesario tenerlos en cuenta debido a que son usados en capítulos posteriores.

2.1 Familia de protocolos TCP/IP.

Una red está formada por un conjunto de computadoras interconectadas para compartir recursos e intercambiar información. Con el fin de permitir que distintas arquitecturas diseñadas por diferentes fabricantes puedan interactuar entre sí, la *Organización de Estándares Internacionales* (ISO, por sus siglas en inglés) inició en 1977 la definición de un modelo de referencia llamado el *Modelo de Interconexión de Sistemas Abiertos* (o mejor conocido como modelo OSI), que divide la arquitectura de red en 7 capas, cada una con funciones específicas independientes unas de las otras.

En este modelo, la capa *n* en una computadora realiza sus funciones comunicándose con la capa del mismo nivel en la otra computadora. Esta comunicación se lleva a cabo a través de reglas bien definidas (protocolos) para una arquitectura de red en particular.

Las funciones de la capa *n* sirven para que ésta pueda ofrecer un servicio a la capa inmediatamente superior en la misma computadora. De esta manera se van agregando servicios conforme se asciende por las capas de la arquitectura hasta llegar a la capa de aplicación. El modelo OSI no define protocolos ni servicios, éstos dependen de la implementación específica de cada arquitectura de red. Lo que el modelo define es la función que debe realizar cada capa.

El modelo de red definido por el conjunto de protocolos TCP/IP es mucho más simple que el definido por otras arquitecturas de red. Conceptualmente, como se muestra en la figura 2.1, una red de redes TCP/IP proporciona tres conjuntos de servicios; su distribución en la figura sugiere una dependencia entre ellos. En el nivel inferior, un servicio de entrega sin conexión proporciona el fundamento sobre el cual se apoya el resto. En el siguiente nivel, un servicio de transporte confiable proporciona una plataforma de alto nivel de la cual dependen las aplicaciones.

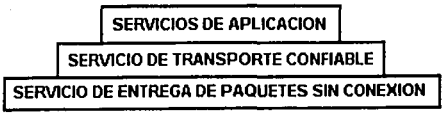


Figura 2.1 Estratificación en capas de TCP/IP

2.1.1 El protocolo IP

El protocolo que define el mecanismo de entrega sin conexión y no confiable es conocido como Protocolo Internet, y por lo general se le identifica por sus iniciales IP. El protocolo IP proporciona tres definiciones importantes. Primero, define la unidad básica para la transferencia de datos utilizada a través de una red de redes TCP/IP. Es decir, especifica el formato exacto de todos los datos que pasarán a través de una red TCP/IP. Segundo, el software IP realiza la función de *ruteo*, seleccionando la ruta por la que todos los paquetes serán enviados. Tercero, además de aportar especificaciones formales para el formato de los datos y el ruteo, el IP incluye un conjunto de reglas que le dan forma a la idea de entrega de paquetes no confiable. Las reglas caracterizan la forma en que las computadoras y los ruteadores deben procesar los paquetes, cómo y cuándo se deben generar los mensajes de error y las condiciones bajo las cuales los paquetes pueden ser descartados.

2.1.1.1 Direcciones IP

Las diferentes partes de Internet están conectadas por un conjunto de computadoras denominadas *ruteadores*, que interconectan las redes. Estas redes pueden ser Ethernet, Token Ring o en ocasiones líneas telefónicas.

Las líneas telefónicas y las redes Ethernet son equivalentes a los camiones y aviones del Servicio Postal. Son el medio a través del cual el correo va de un lugar a otro. Los ruteadores son como sucursales postales; estos equipos deciden cómo dirigir los paquetes de información, de la misma manera que una oficina postal decide cómo distribuir los sobres por correo. No toda subestación o ruteador cuentan con una conexión a cada uno de los ruteadores de la red. Cada subestación sólo necesita conocer las conexiones con las que cuenta y cuál es el mejor "siguiente salto" para acercar el paquete a su destino.

¿Cómo sabe la red a dónde se dirige la información? Si se quiere enviar una carta, no basta con poner el papel escrito en el buzón y esperar a que sea entregado. Es necesario poner el papel con la información en un sobre, escribir el domicilio del

destionatario y pegar los timbres postales. De la misma manera que la Oficina de Correos tiene reglas que definen la operación de su red, también existen reglas que definen la operación de Internet. Las reglas son llamadas protocolos. El protocolo Internet (IP) se hace cargo de establecer domicilios o se asegura que los ruteadores sepan qué hacer con la información que les llega.

Para entregar datos entre dos computadoras en Internet, es necesario mover los datos a traves de la red hasta el destino correcto, y dentro de ese destino hacer llegar la información hacia el usuario o proceso correctos.

TCP/IP usa tres mecanismos para cumplir con estas tareas:

- Direccionamiento
- Ruteo
- Multiplexado.

Una dirección especifica una conexión a una determinada red, es decir el punto de entrada a una computadora desde una red determinada. Una dirección no se encuentra intrínsecamente asociada a una computadora específica ya que si movemos una computadora de lugar tendremos que modificar su dirección.

2.1.1.2 Clasificación de las direcciones IP

Para lograr una comunicación universal entre distintos equipos a través de una red como Internet es necesario que el esquema de direccionamiento sea de tipo jerárquico, ya que sería imposible que una entidad administradora centralizada asignara la dirección de cada una de las computadoras conectadas a la red. Un esquema jerárquico de direcciones proporciona la ventaja de que los responsables de asignarlas sólo tengan que administrar un conjunto manejable de éstas. El NIC (Network Information Center) es el organismo que se encarga de descentralizar la asignación de direcciones en Internet.

El protocolo internet (IP) mueve datos entre hosts en forma de paquetes. Cada paquete es entregado en la dirección especificada en el campo **destination address** (dirección destino) del encabezado del paquete. La dirección de destino es una

dirección estándar de 32 bits (4 bytes) que contiene la información necesaria para identificar una red y un host especifico dentro de esa red.

Cualquier dirección IP se dividen en dos componentes, uno que identifica a una red en especifico y el otro que identifica a una computadora especifica dentro de esa red. Estos dos campos de la dirección reciben los nobres de **network ID** y **host ID** respectivamente, pero el formato de estas partes no es el mismo en todas las direcciones IP. El numero de bits usados para identificar a una red, y el numero usado para identificar el host, varia dependiendo de la clase de la dirección.

La primera jerarquia establecida formalmente, mostrada en al figura 2.2, es la que divide a los tipos de direcciones de acuerdo al formato del campo network ID en tres clases : **A, B** y **C**.

El software IP rapidamente determina la clase de una dirección al examinar los primeros bits de la dirección. IP sigue las siguientes reglas para determinar la clase de una dirección:

- Las direcciones de **clase A** se distinguen porque comienzan con un **0**. En ellas, los primeros 8 bits especifican el network ID, y los siguientes, el número de host dentro de esa red (host ID). Con una dirección de este tipo se pueden tener hasta 126 redes cada una hasta con 16,777,216 computadoras conectadas a cada una de ellas.

- Las direcciones de **clase B** se distinguen porque comienzan con los bits **10**, y en ellas, los 16 primeros bits identifican a la red y 16 al host. Aquí se pueden tener hasta 16,384 redes con 65,536 computadoras conectadas a cada una de ellas.

- Las direcciones de **clase C** comienzan con **110**, y asignan 24 bits al numero de red, los restantes 8 son para el numero de host, por lo que se pueden tener 2,097,152 redes con 256 computadoras conectadas.

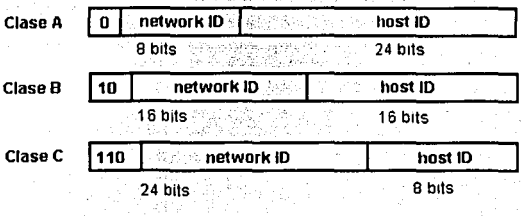


Figura 2.2 Formato de las direcciones IP.

Para un manejo eficiente de las direcciones IP, éstas son usualmente escritas como un conjunto de 4 numeros decimales separados por puntos, a esta forma se le denomina **notación decimal con puntos**. Cada uno de estos numeros se encuentra en el rango de 0 a 255 (los posibles valores decimales que puede tomar un byte).

De esta manera, la dirección IP binaria (32 bits) se escribe de la forma **w.x.y.z** donde:

w = primer byte en decimal
x = segundo byte en decimal
y = tercer byte en decimal.
z = cuarto byte en decimal.

Ejemplo:

Dirección IP binaria:	**10000100000011010000111010000001**
Tipo de red:	**Clase B** (los primeros dos bits son **10**)
División en Bytes:	**10000100 00001101 00001110 0000001**
Equivalente en decimal:	**132 13 14 129**
Notación con puntos:	**132.13.14.129**

En la tabla 2.1 se muestran los rangos de direcciones disponibles para cada uno de los tipos de redes en TCP/IP:

Tipo	Dirección más baja	Dirección más alta
A	0.1.0.0	126.0.0.0
B	128.0.0.0	191.255.0.0
C	192.0.1.0	223.255.255.0
D	224.0.0.0	239.255.255.255

Tabla 2.1 Rangos de direcciones por tipo de red.

Por varias razones prácticas (sobre todo por limitaciones de hardware), la información enviada a través de las redes IP se divide en pedazos de tamaño distinto, llamados paquetes. La cantidad de información en un paquete normalmente se encuentra entre 1 y 1500 bytes de largo. Esto previene que cualquier usuario monopolice la red, permitiendo que todos tengan un acceso equitativo. También significa que cuando la red se sobrecarga, su comportamiento sólo desmerece un poco para todos los usuarios: la red no se inutiliza cuando algunos usuarios la monopolizan.

2.1.1.3 Reglas especiales de direccionamiento IP.

Algunas direcciones permiten definir como destino redes completas en lugar de computadoras específicas. Si por ejemplo hablamos de la dirección 132.248.0.0 estamos hablando de una dirección que especifica una red.

Cuando una parte de la dirección contiene ceros, se da por supuesto que se trata de *este*, así, si una computadora envía un mensaje a la dirección 0.0.57.5, se enviará un mensaje al host con el identificador 57.5 dentro de *esta red*, es decir la red a la que pertenece la computadora transmisora.

La dirección de red 127 es una dirección especial, que sirve para probar el software de red. El uso de esta dirección nunca deberá causar tráfico en la red, ya que

los paquetes nunca salen de la computadora que los envío. Esta dirección se conoce como de **loopback.**

En las **direcciones de difusión** todos los bits son puestos a uno y son usadas para hacer llegar un paquete de información hacia todas las computadoras dentro de la misma red en lugar de hacerlo llegar a una sola. La figura 2.3 muestra los casos especiales de direccionamiento IP.

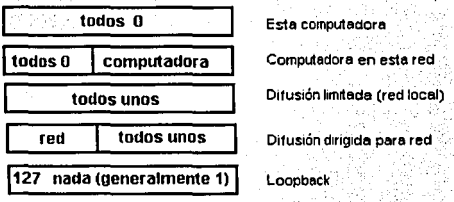


Figura 2.3 Casos especiales de direcciones IP.

Podemos encontrar también un tipo especial de dirección IP, la cual especifica direcciones de multicast; para este tipo de direcciones se sigue la siguiente regla:

- Los tres primeros bits empiezan con **1110.** Estas direcciones son comúnmente llamadas **direcciones de clase D,** pero no se refieren realmente a ninguna red en especial. Los numeros asignados a este tipo de red son direcciones de multicast. Las direcciones de multicast son usadas para enviar información a grupos de computadoras, en distintas redes, todas al mismo tiempo.

2.1.2 El protocolo TCP

Una de las propiedades más impresionantes de Internet es que, en un nivel básico, el protocolo IP es todo lo que se necesita para participar en una comunicación de red, no obstante hay que resolver varios problemas:

- La mayoría de las transferencias de información es mayor a 1500 bytes.
- En ocasiones se presentan errores: ocasionalmente los paquetes pueden ser extraviados.
- Los paquetes pueden llegar en desorden.

Para evitar esto, la capa de red de TCP nos permitirá enviar grandes cantidades de información y corregirá las alteraciones que puedan ser causadas por la red.

TCP es el protocolo que se menciona frecuentemente junto con el IP y que se utiliza para resolver los problemas anteriormente mencionados. El protocolo TCP toma la información que se desea enviar y la divide en segmentos. Además, enumera cada segmento para que el receptor pueda verificar la información y ponerla en el orden adecuado. Para que el protocolo TCP pueda enviar esta secuencia de números a través de la red, cuenta con su propio sobre que le permite escribir en él la información requerida para su reordenamiento. Un segmento de la información a transmitir se coloca en el sobre del protocolo TCP. Este sobre es puesto, a su vez, dentro del sobre del protocolo IP y posteriormente es transmitido a la red. Una vez que se pone algo en un sobre IP, la red lo puede transmitir.

Del lado del destinatario, una parte del software de TCP reúne los sobres, extrae la información de ellos y la pone en el orden adecuado. Si algún sobre se pierde en la transmisión, el receptor solicita su retransmisión al emisor. Una vez que el protocolo TCP tiene toda la información en el orden adecuado, la pasa a la aplicación del programa que esté utilizando sus servicios.

La descripción anterior del funcionamiento del protocolo TCP es ligeramente utópica. En la realidad, los paquetes no sólo se pierden, además de esto pueden ser modificados por el mal funcionamiento durante la transmisión a través del medio. TCP también resuelve este tipo de problemas. Así como coloca la información en un sobre, el protocolo calcula algo llamado número de verificación (checksum). Este número permite que el receptor TCP detecte errores en el paquete transmitido.

Cuando un paquete llega a su destino, el receptor calcula el número de verificación y lo compara con el enviado por el transmisor. Si no coinciden, significa que ocurrió un error en la transmisión. El receptor deshecha el paquete y solicita la retransmisión.

2.1.3 La interfaz socket.

La base para la E/S en red se centra en una abstracción conocida como **socket**. Los sockets son vistos como una generalización de mecanismo de acceso a archivos, los cuales proporcionan un punto final para la comunicación. La interfaz socket nació con la versión BSD de UNIX, y actualmente es muy popular y es soportada por muchos fabricantes de software.

2.1.3.1 El paradigma de E/S de UNIX.

Los sistemas primitivos de E/S salida de UNIX siguen un paradigma que algunas veces se denomina **open-read-write-close** (abrir, leer, escribir, cerrar). Antes de que un proceso de usuario pueda ejecutar operaciones de E/S, se hace una llamada a la función **open** para especificar el archivo o dispositivo que se va a usar y obtiene el permiso para usarlo. La llamada a open devuelve un pequeño entero denominado **descriptor de archivo** que el proceso utiliza cuando ejecuta las operaciones de E/S en el archivo o dispositivo abierto. Una vez que se ha abierto un objeto, el proceso de usuario hace una o mas llamadas a las funciones **read** o **write** para transferir datos. Read transfiere datos dentro del proceso de usuario; write transfiere datos del proceso de usuario al archivo o dispositivo. Luego de completar todas las operaciones de transferencia, el proceso de usuario llama a la función **close** para informar al sistema operativo que ha terminado de usar el objeto.

2.1.3.2 Adición de soporte de E/S en red a través de sockets.

Al igual que con el acceso a archivos, los programas de aplicación requieren que el sistema operativo genere un socket cuando se necesita. El sistema devuelve un entero pequeño que utiliza el programa de aplicación para hacer referencia al socket recientemente creado.

Como en el caso de los archivos, las aplicaciones solicitan la creación de un socket cuando se requiere, el sistema regresa entonces un número entero denominado descriptor de socket. Los sockets hacen ejecuciones exactamente iguales a los archivos

o dispositivos de UNIX, de manera que pueden utilizarse las operaciones tradicionales (como read y write). Una vez que un programa de aplicación crea un socket y una conexión TCP del socket a un destino externo, el programa puede hacer uso de write para mandar un flujo de datos a través de la conexión (el programa de aplicación en el otro extremo puede usar read para recibirlo). Cuando un socket ya no se requiere, puede hacerse una llamada a close.

La diferencia principal entre los descriptores de archivos y los descriptores de socket, es que el sistema operativo enlaza un descriptor de archivo a un archivo o dispositivo específico cuando la aplicación llama a open, pero puede crear sockets sin enlazarlos a direcciones de destino específicas. La aplicación puede elegir proporcionar una dirección de destino cada vez que utiliza el socket (es decir, cuando se envían paquetes), o elegir asociar la dirección de destino a un socket y evadir la especificación de destino repetidamente (es decir, cuando se hace una conexión TCP).

2.1.3.3 Comunicación en red a través de sockets.

Tomando como base el paradigma de UNIX: open-read-write-close (abrir, leer, escribir, cerrar) para archivos, dispositivos y sockets, la secuencia para entablar una comunicación de red usando sockets sería básicamente la siguiente:

1.- Creación del socket.

Inicalmente, un socket se crea en el **unconnected state** (estado no conectado), lo que significa que el socket no esta asociado con ningún destino externo. La llamada para crear un socket es la siguiente:

descriptor = socket (af, tipo, protocolo)

Dónde:

descriptor es un número entero, el cual identifica al descriptor de socket.

El argumento **af** especifica la familia de protocolos que se utilizará con el socket. Estas incluyen AF_INET (TCP/IP), Xerox PUP Internet (AF_PUP) entre otros.

El argumento **tipo** especifica el tipo de comunicación deseado. Orientado a conexiones y confiable (SOCK_STREAM) o sin conexión (SOCK_DGRAM) y un tipo especial que permite a programas privilegiados accesar a protocolos de bajo nivel (SOCK_RAW).

Debido a que puede haber múltiples protocolos que soporten el mismo tipo de servicio, se utiliza el argumento **protocol**.

2.- Asociar el socket con un destino.

La llamada de sistema **connect** (conectar) enlaza un destino permanente a un socket, colocándolo en el **connected state** (estado conectado). Un programa de aplicación debe llamar a connect para establecer una conexión antes de que pueda transferir datos a través de un socket de flujo confiable.

La llamada de sistema connect tiene la forma:

```
connect (socket, destaddr, addrlen)
```

El argumento *socket* es el descriptor entero del socket que se va a conectar. El argumento destaddr es una estructura de dirección socket en la que se especifica la dirección de destino a la que deberá enlazarse el socket. Finalmente, addrlen especifica la longitud en bytes de la dirección destino.

3.- Transmisión de datos.

Una vez que el programa de aplicación ha establecido el socket, puede usarlo para transmitir datos. Para esto se hace uso de la llamada write, la cual tiene la forma:

```
write (socket, buffer, longitud)
```

El argumento socket es el descriptor de socket. El argumento buffer contiene la dirección de datos que se van a enviar, y el argumento longitud especifica el número de bytes que se van a mandar. Como en la mayoría de las llamadas de sistema de UNIX,

write devuelve un código de error para la aplicación que lo llama, lo cual permite al programador saber si la operación tuvo éxito o no.

4.- Recepción de datos.

Para recibir datos a través de un socket, una aplicación debe hacer uso de la llamada read, la cual tiene la forma:

```
read (descriptor, buffer, longitud)
```

donde el argumento *descriptor* es un número entero que indica el descriptor de socket desde el cual se pueden leer los datos; el argumento *buffer* especifica la dirección de memoria en la que se deben almacenar los datos leídos, y el argumento *longitud* especifica el número máximo de bytes que pueden leerse.

5.- Cerrar el socket.

Para finalizar la comunicación, se debe hacer una llamada a close. Su sintaxis es la siguiente:

```
close (socket)
```

En dónde socket es el descriptor del socket a cerrar.

2.2 Modelo Cliente - Servidor.

En un sistema **cliente/servidor**, uno o más clientes y uno o más servidores, junto con el sistema operativo y los protocolos de comunicación, conforman el ambiente que permite y facilita el cómputo distribuido.

En una aplicación basada en esta arquitectura existen dos procesos independientes, en lugar de uno solo. De esta forma se puede repartir el trabajo a través de varias computadoras en una red. Estos dos procesos, cliente y servidor, se comunican mediante un protocolo bien definido. Esta técnica permite la comunicación

entre distintas computadoras (servidores de archivos, estaciones de trabajo con alta calidad de graficación, etc.), para que cada una de ellas se dedique a realizar el trabajo que hace mejor.

De manera introductoria, se puede decir que un servidor es un sistema o un programa que provee de algún servicio a otros sistemas a través de una red. Un ejemplo típico es un servidor de archivos, que permite el acceso a información remota a cualquier usuario a través de la red. Un cliente es un sistema o programa que requiere y recibe alguna acción de un servidor.

La arquitectura cliente/servidor es una forma de cómputo en red en el que ciertas funciones solicitadas por clientes son servidas por los procesos adecuados. Un servidor es un administrador de recursos. Un recurso puede ser identificado como algo físico (una impresora) o abstracto (una base de datos). Un cliente es un usuario de los recursos que administra el servidor.

De manera general, para que se inicie la comunicación entre un cliente y un servidor es necesario establecer una sesión. El servidor debe estar esperando que algún cliente trate de establecer una sesión y éste se reserva el derecho de establecer comunicación con uno o más clientes. Así, el servidor se encargará de atender a cada cliente y establecer los mecanismos que seguirá para la distribución de sus servicios.

Típicamente, una aplicación (cliente) comenzará una transacción mediante una sesión (figura 2.4) , ejecutará una o varias operaciones en el servidor, éste le regresará algún resultado (figura 2.4) y terminará la transacción. Los servidores están estructurados para trabajar en un ciclo infinito, de tal manera que puedan aceptar conexiones de los clientes en cualquier momento.

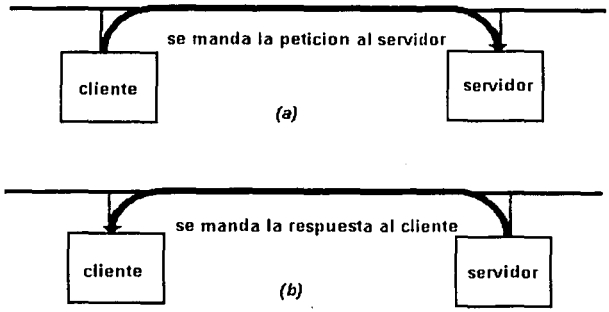


Figura 2.4 Ejemplo del modelo cliente servidor.
En el inciso (a) el cliente manda una petición al servidor, y
en el inciso (b) el cliente devuelve una respuesta.

2.2.1 Características del cliente.

En un sistema cliente/servidor, un cliente es un proceso que interactúa con el usuario, observando las siguientes características:

a) Presenta la interfaz de usuario (UI)
   Esta interfaz permite al usuario introducir sus consultas de recuperación y análisis de datos, así como recibir los resultados de dichas consultas, típicamente en un ambiente gráfico (GUI).
b) Forma consultas o comandos en un lenguaje predefinido, para su presentación al servidor.
c) Se comunica con el servidor por medio de una metodología de comunicación de procesos determinada y transmite consultas o comandos al servidor.
d) Realiza análisis de datos sobre resultados de la consulta o el comando que regresan del servidor y lo presenta al usuario.

2.2.2 Características del servidor.

Un ***servidor*** es un proceso o conjunto de procesos que deben existir en un equipo de cómputo que da servicio a uno o más clientes, y generalmente tiene las siguientes características:

1.- Un servidor da servicio a un cliente. La naturaleza y el grado de este servicio es definido por el objetivo de la implementación, es decir, un servicio puede requerir un mínimo de capacidad de procesamiento en el lado del servidor, como es el caso de los servidores de archivos o de los servidores de impresión, o bien puede necesitar de un procesamiento intensivo, como se da en el caso de los servidores de bases de datos.

2.- Un servidor solamente responde a las consultas o comandos de los clientes. Esto significa que ningún servidor inicia la conversación con un cliente, tampoco atiende directamente interfaces con el usuario final. Simplemente actúa como repositorio de datos (servidor de archivos), o de conocimiento (servidores de bases de datos) o como prestador de servicios de impresión.

Sin embargo, un servidor si puede iniciar una conversación con otro servidor, solicitándole un servicio que a su vez le permitirá atender el requerimiento de un cliente.

3.- Un servidor ideal hace transparente todo el esquema cliente/servidor al cliente y usuario. Un cliente que se comunica con un servidor no tiene porque estar enterado de la plataforma de hardware y software que intenta acceder, así como de la tecnología de comunicación que hace posible ese enlace. Para ello es deseable y recomendable que en un ambiente de servidores múltiples, los servidores se comunican entre sí para proporcionar un servicio al cliente sin que éste conozca de esta múltiple existencia, ni de la comunicación entre servidores.

Un sistema cliente/servidor ofrece soluciones a las desventajas de los clientes centralizados tanto en mainframes como en servidores locales, al poseer las siguientes características:

- **Es inteligente a nivel de equipo de escritorio, ya que el cliente es el responsable de la interfaz con el usuario.**

El cliente transforma las consultas o comandos del usuario a un lenguaje predefinido que es comprendido por el servidor y presenta los resultados que el servidor le envía como respuesta, con lo que se obtiene mayor capacidad de proceso a un menor costo.

- **Permite compartir los recursos del servidor de manera óptima.**

Estos recursos pueden ser tanto del procesador, como almacenamiento secundario y periféricos. Un cliente puede pedir al servidor que realice procesos grandes, o puede pedirle que corra grandes aplicaciones (p. ej. Servidores de bases de datos), y como fruto obtener solamente los resultados de ese proceso.

- **Utilización óptima de la red.**

Dado que los clientes se comunican con el servidor a través de un lenguaje predefinido, como puede ser SQL, y el servidor sólo remite al cliente los resultados de la consulta o comando, reduciendo el tráfico en la red de los sistemas centralizados que transfieren los archivos de datos en su totalidad.

- **Permite cierta independencia sobre el sistema operativo y los protocolos de comunicación.**

Esto facilita el mantenimiento de las aplicaciones y asegura su portabilidad.

- **Permite desarrollos más flexibles a un menor costo.**

Hoy en día, la mayoría de los componentes de un sistema cliente/servidor, están disponibles comercialmente, provenientes de diversos proveedores, lo que da a las organizaciones libertad de elección, sin por ello exentarlas de los desarrollos internos.

2.3 Programación Orientada a Objetos.

El término Programación Orientada a Objetos (**POO**), hoy en día ampliamente utilizado, es difícil de definir. No es un concepto nuevo, sino que ha sido el fruto del desarrollo de diversas técnicas de programación desde principios de la década de los 70, aunque sea en la década de los 90 cuando ha aumentado su difusión, popularidad y uso. No obstante, se puede definir a la **programación orientada a objetos como una técnica o estilo de programación que utiliza objetos como el bloque esencial de construcción,** con la cual se mejora el desarrollo de los programas y la fiabilidad de los programas resultantes.

En *programación convencional*, los programas se dividen en dos componentes: **procedimientos** y **datos**. Cada procedimiento actúa como una caja negra, es un componente que realiza una tarea específica, como realizar una operación arimética, un método de ordenación o visualizar una ventana. Si las cajas negras se dividen correctamente, se puede escribir código para cada una sin preocuparse de lo que hacen internamente otras cajas negras. La principal ventaja de utilizar este método es que ayuda a desarrollar programas que son modulares y transportables.

Sin embargo, cuando se utilizan los métodos de la programación orientada a objetos, un programa agrupa en **componentes independientes** tanto procedimientos como datos y a cada uno de estos componentes se le considera un **objeto**.

Los objetos son en realidad tipos abstractos de datos. Un **tipo abstracto de dato** es un tipo de dato definido por el programador junto con el conjunto de operaciones que se pueden realizar sobre ellos. Se denominan abstractos para diferenciarlos de los tipos de datos fundamentales o básicos definidos, por ejemplo en C, tales como **int**, **char** y **float**.

2.3.1 Objetos

Un *objeto* es una unidad que contiene datos y las funciones que operan sobre esos datos. Los datos se denominan **miembros dato** y las funciones miembros función, funciones miembro o **métodos**.

Los datos y los métodos se encapsulan en una única entidad. Los datos están ocultos y solo mediante las funciones miembro es posible tener acceso a ellos. Los términos encapsulación de datos y ocultación de datos son términos clave empleados en este tipo de programación.

Un objeto es cualquier entidad del mundo real que uno se pueda imaginar:

- ***Objetos físicos.***

  Automóviles en una simulación de tráfico.

  Aviones en un sistema de control de tráfico aéreo.

  Componentes electrónicos en un programa de diseño de circuitos.

  Animales mamíferos.

- ***Elementos de interfaces gráficas de usuario***

  Ventanas

  Iconos

  Menús

- ***Estructuras de datos.***

  Arreglos

  Pilas

  Colas

  Árboles binarios.

- ***Tipos de datos definidos por el usuario.***

  Números complejos

  Hora del día.

  Puntos de un plano.

Por ejemplo, supongamos que se dispone de un objeto tal como una ventana en la pantalla, y se desea definir las cuatro operaciones requeridas para mover el objeto ventana en pantalla:

1. Mover el objeto ventana a la derecha.
2. Mover el objeto ventana a la izquierda
3. Mover el objeto ventana hacia arriba
4. Mover el objeto ventana hacia abajo.

Utilizando métodos de programación convencional, se puede escribir un procedimiento independiente para cada operación. Estos procedimientos actúan como cajas negras y suponen un mecanismo idóneo para dividir operaciones. Desgraciadamente, los datos que definen el objeto son independientes de cada uno de los procedimientos.

Aplicando la metodología de la programación orientada a objetos, combinamos datos y procedimientos en un único paquete. Para ello el primer paso es crear un objeto ventana. Esta ventana contiene los procedimientos que definen cómo se mueve la ventana. Cuando la ventana recibe una orden tal como mover a la izquierda se ejecuta el procedimiento correspondiente (ver figura 2.5)

Figura 2.5 El objeto ventana "sabe" como comportarse ante los mensajes de movimiento.

Los objetos soportan una serie de características específicas de los mismos:

- Se agrupan en tipos denominados clases
- Contienen datos internos que definen su estado actual.
- Soportan ocultación de datos.

- Pueden heredar propiedades de otros objetos.
- Pueden comunicarse con otros objetos enviando o pasando mensajes.
- Tienen métodos que definen su comportamiento.

Los objetos pueden ser tratados como variables. La principal diferencia es que se puede llamar a cualquiera de las funciones públicas que pertenecen al objeto, o dicho de otra manera, se puede enviar un mensaje a la función.

2.3.2 Clases

En la la programación orientada a objetos se suele decir que los objetos son miembros de clases. Una **clase** es un tipo definido por el usuario que determina las estructuras de datos y las operaciones asociadas con ese tipo. Las clases son como plantillas o modelos que describen cómo se construyen ciertos tipos de objetos. Cada vez que se construye un objeto de una clase, se crea una **instancia** de esa clase. Por consiguiente los objetos son instancias de clases. Una clase es una colección de objetos similares y un objeto es una instancia de una definición de una clase. Una clase puede tener muchas instancias y cada una es un objeto independiente. Una clase es simplemente un modelo que se utiliza para describir uno o más objetos del mismo tipo.

Una de las características fundamentales de una clase es de ocultar tanta información como sea posible. Por consiguiente, es necesario imponer ciertas restricciones en el modo en que se puede manipular una clase y de cómo se pueden utilizar los datos y el código dentro de una clase.

Una clase puede contener partes públicas y partes privadas. Por defecto, todos los miembros definidos en la clase son privados. Para hacer las partes de una clase públicas (esto es, accesible desde cualquier parte del programa) deben declararse explícitamente como publicas (**public**). Escencialmente, cualquier objeto accesa a a cualquier método o función del objeto a través de sus funciones y datos públicos. Dado que una característica clave de la POO es la ocultación de datos, se debe tener presente que aunque se pueden tener cualquier cantidad de variables públicas, en lo posible se debe tratar de limitar o eliminar su uso. En su lugar, debe hacerse todo lo posible por declarar todos los datos como privados y controlar el acceso a ellos por medio de

funciones públicas. La figura 2.6 muestra gráficamente la relación de los conceptos arriba mencionados.

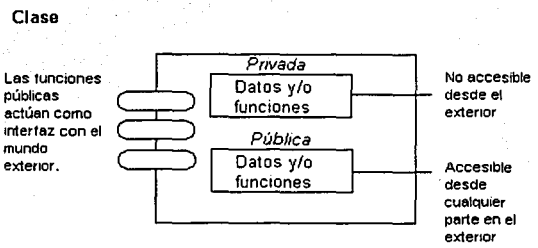


Figura 2.6 Diagrama de una clase

2.3.3 Abstracción de datos, Herencia, Polimorfismo.

La programación orientada a objetos es una técnica para organizar programas. Su importancia reside en el modo que se diseñan los programas y no en los detalles de las operaciones individuales. En particular, los programas orientados a objetos se organizan a través de los objetos y las clases, que contienen los datos y las funciones que actúan sobre esos datos. Los conceptos fundamentales de la POO son: abstracción de datos (clases), herencia y polimorfismo.

2.3.3.1 Abstracción de datos.

La abstracción se define como la "extracción de las propiedades esenciales de un concepto". Con la abstracción de datos, las estructuras de datos y elementos se pueden utilizar sin preocuparse sobre los detalles exactos de la implementación. Por ejemplo,

los números reales son abstracciones en todos los lenguajes de programación; el programador no tiene que preocuparse por la representación binaria exacta de un número real cuando le asigna un valor; no se requiere conocer la manera en que se realiza una multiplicación binaria para multiplicar valores reales.

La abstracción de datos permite no preocuparse de los detalles no esenciales. La abstracción de datos existe en casi todos los lenguajes de programación. Las estructuras de datos y los tipos de datos son un ejemplo de abstracción. Los procedimientos y funciones son otro ejemplo. Una clase es un modelo o plantilla que describe las estructuras de datos y las acciones que se realizan sobre ellas.

2.3.3.2 Herencia.

Una característica muy importante de los objetos y las clases es la **herencia.** La herencia es la propiedad que permite a los objetos construirse a partir de otros objetos. El concepto de herencia está presente en nuestras vidas diarias donde las clases se dividen en subclases, es una ley de la naturaleza. Un ejemplo muy simple es el que los niños toman o reciben atributos de sus padres. Así por ejemplo, las clases de animales se dividen en mamíferos, anfibios, insectos, pájaros, etc. La clase de vehículos se divide en automóviles, autobuses, camiones, motocicletas, etc.

El principio de este tipo de subdivisión es que cada subclase comparte características comunes con la clase de la que se deriva. Los automóviles, camiones, autobuses y motocicletas (que pertenecen a la clase vehículo) tienen ruedas y un motor (características en común).

Además de las características compartidas con otros miembros de la clase, cada subclase tiene sus propias características particulares: los autobuses, por ejemplo, tienen un gran número de asientos, una televisión para los pasajeros, mientras que las motocicletas tienen dos ruedas y un solo asiento.La herencia impone una relación jerárquica entre clases en la cual una clase hija hereda de su clase padre.

La herencia es también natural en el mundo de la programación, y existen numerosos casos de ella. Por ejemplo, un cuadro de diálogo tiene muchas cosas en común con una ventana. De hecho, se puede decir que un cuadro de diálogo hereda ciertos atributos de una ventana, tales como un borde, un título, un cursor, etc.

El uso de la herencia en la POO implica la creación de nuevas clases a partir de otras previamente existentes (figura 2.7). Por ejemplo, se puede tener una *clase casa*, que conste de una cocina, una sala, una recámara y un baño. Se puede utilizar esta clase como un bloque de construcción y tomándola como base crear mansiones o casas más grandes. Es decir, sólo basta añadir a la *clase casa* mas objetos tales como: habitaciones, baños, un jardín, alguna estancia, y tendremos una lujosa mansión. En la terminología de la POO se dice que la *clase mansión* se **deriva** de la *clase casa*. Una clase utilizada para derivar nuevas clases se conoce como **clase base** y una clase creada a partir de otra se llama **clase derivada.**

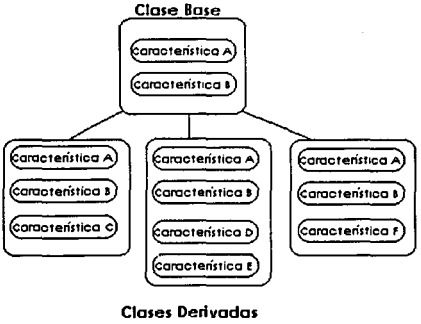


Figura 2.7 Herencia en la programación orientada a objetos

En general la herencia se aplica para extender y reutilizar el código existente:

- Los objetos se pueden construir de modo incremental a partir de otros objetos y pueden compartir código y estructuras de datos.
- Los algoritmos generales se pueden escribir de modo que se pueden reutilizar para nuevos tipos de objetos, incluso después de que los algoritmos originales se han compilado.

2.3.3.3 Polimorfismo.

Una de las características más importantes de la POO es la capacidad de que diferentes objetos respondan a órdenes similares de manera diferente. Esta característica se denomina polimorfismo y los objetos que lo soportan se denominan objetos polimórficos.

El polimorfismo se refiere al hecho de que una misma operación puede tener diferente comportamiento en diferentes objetos. Por ejemplo, supongamos que se dispone de las clases **cuadrado, triángulo y círculo**, cuyos objetos representan las correspondientes figuras geométricas. Los objetos de estas clases pueden comprender un mensaje **area()** que calcula el área correspondiente a cada figura. Sin embargo, la respuesta al mensaje **area()** será diferente para cada uno de los objetos, ya que cada uno calculará el área de la figura geométrica que le corresponde con una fórmula diferente. El mecanismo para calcular el área de los objetos difiere de una figura a otra, pero todas las figuras realizan esta tarea en respuesta al mismo mensaje (ver figura 2.8).

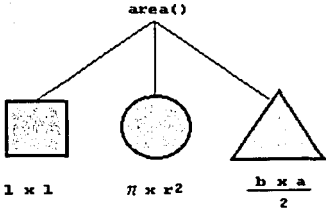


Figura 2.8 ***Polimorfismo.*** *Cada uno de los objetos responde de manera distinta al mismo mensaje.*

2.3.4 Reutilización de código.

La reutilización es un concepto muy importante que permite crear código que se puede utilizar más facilmente en un programa. La reutilización reduce los costos de desarrollo, aumenta la velocidad de desarrollo de procesos y reduce las necesidades de comprobación o verificación.

En la programación orientada a objetos, el concepto de herencia proporciona una importante extensión a la idea de reutilización. Un programador puede tomar una clase existente, y sin modificarla, añadir características y posibilidades adicionales a la misma. Estas operaciones se realizan por derivación de una clase nueva a partir de una clase existente. La nueva clase heredará la propiedad de la antigua, pero es posible añadir nuevas características propias.

Una vez que una clase se ha escrito, creado, depurado y utilizado, se puede difundir entre otros programadores para que puedan utilizarla es sus propios programas. Esta propiedad se denomina reutilización. Es similar a la forma en que una biblioteca de funciones de un lenguaje procedimental se puede incorporar en programas diferentes.

La reutilización del software es un término muy importante, al grado que se ha llegado a comparar con el concepto de *chip de hardware*, queriendo significar que esta propiedad es similar a la reutilización del hardware mediante circuitos integrados. Los objetos en POO son fácilmente reutilizables y es una de las razones principales para justificar la utilización de la metodología orientada a objetos en la mayoría de los casos. La herencia aumenta considerablemente la potencia de la reutilización de software; es un proceso que mejora la reutilización de componentes claves, utilizando, modificando y redefiniendo parcialmente las clases producidas por otros programadores y almacenadas en una biblioteca. Todo este mecanismo tiene lugar sin tener acceso al código fuente de la clase base, sino sólo a su interfaz, por esta razón, cualquier aplicación POO suele venir con un conjunto de clases predefinidas, que permitirá ahorrar tiempo y esfuerzos en el desarrollo de las aplicaciones.

2.4 El lenguaje de programación Java.

Java es un lenguaje de programación que Sun Microsystems comenzó a desarrollar en 1991 y que en la actualidad ha alcanzado un nivel de madurez lo suficientemente alto como para que sus cualidades empiecen a ser apreciadas.

Y es que precisamente sus muy particulares características son las que la convierten en la herramienta casi ideal para crear el nuevo tipo de aplicaciones que el mundo computacional empieza a demandar. Su potencial es tan grande que puede ser la tecnología que domine la computación en los próximos años.

2.4.1 Comparando a Java y C++

En su sintaxis Java es similar a C++, lo que permite que los programadores se familiaricen rápidamente con el lenguaje. Pero éste ha tomado también lo mejor de varios lenguajes orientados a objetos entre los que se encuentran Eiffel, SmallTalk, Object C y Cedar/Mesa.

Si se compara con C++, se podría pensar que Java es un mero subconjunto de este lenguaje, pues muchas de las propiedades que hicieron tan popular en su momento al primero, han sido eliminadas por completo en el segundo.

Los diseñadores del lenguaje tomaron a C++ como modelo del nuevo lenguaje, sin embargo, encontraron muchos problemas con los aspectos complicados del lenguaje, tales como herencia múltiple de clases y errores en el manejo dinámico de la memoria.

En Java se han eliminado características tales como el preprocesador, estructuras (struct), uniones (union), funciones y el uso del goto, pero sobre todo se prescinde totalmente del uso de apuntadores, por lo que todo lo que podía hacerse con estas características no aplica para Java, sin embargo puede implementarse en forma de objetos sin ningún problema.

Muchas palabras reservadas también son comunes entre ambos lenguajes, lo que permite que un programador pueda pasar de un ambiente al otro sin gran problema. La figura 2.9 muestra esta relación.

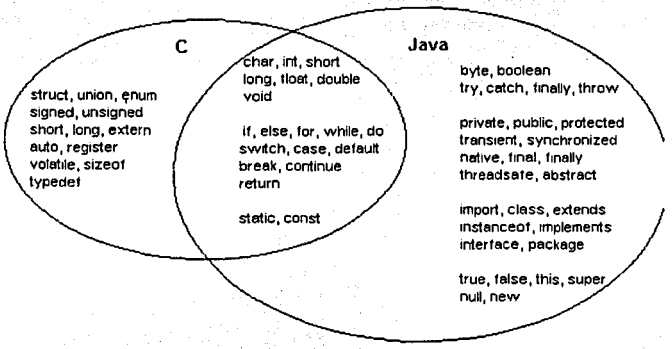


Figura 2.9 Palabras reservadas comunes a Java y C.

Por ejemplo, al quitar los apuntadores se eliminó de golpe la fuente del 90% de los errores de programación en cualquier programa en C. Las razones más comunes para recurrir a apuntadores en C es el manejo de arreglos y cadenas de texto, pero en Java existen mecanismos lo suficientemente flexibles para manejar estas importantes estructuras, sin necesidad de complicarse la vida con apuntadores. Por otra parte, desarrollar cosas como multimedia, crear animaciones, sicronizar sonido con eventos o interactuar con el ratón han sido muy simplificados en Java.

2.4.2 Independencia de plataforma

Java es un lenguaje **independiente de plataforma**, lo que significa que los programas desarrollados con este lenguaje pueden ejecutarse en cualquier sistema informático sin necesidad de efectuar cambios en el código fuente o en el código objeto.

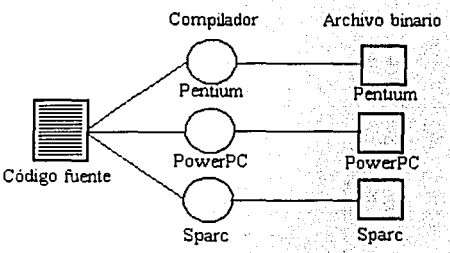


Figura 2.10 Compilación tradicional

Otros lenguajes de programación no presentan esta facilidad, ya que si se desea que un programa funcione en distintas plataformas, hay que compilar el código fuente para cada una de ellas, lo que en ocasiones se convierte en una tarea bastante complicada (figura 2.10).

Java logra la independencia de hardware utilizando un formato especial para los programas compilados, este código, llamado **byte-code**, puede ser leído y ejecutado en cualquier computadora que tenga instalado un intérprete Java que implementa la máquina virtual Java, el cual debe de estar escrito especialmente para la plataforma en la cual se ejecute (figura 2.11).

La máquina virtual Java es el pegamento mágico que une a todos los equipos físicos, sistemas operativos e interfaces gráficas, ofeciendo a los programadores una sola plataforma, con lo que un programa se piensa, desarrolla y genera en código Java entonces este programa se utiliza en cualquier máquina virtual Java.

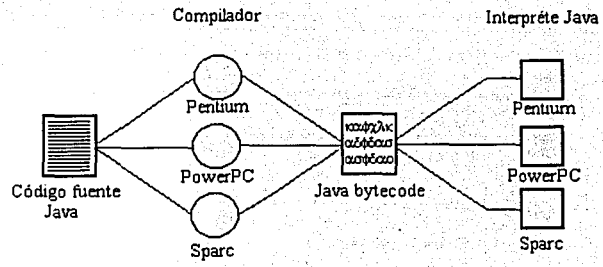


Figura 2.11 El bytecode logra la independencia de hardware de Java.

Para crear un programa de Java, primero hay que utilizar un editor de textos ASCII para crear el archivo fuente. Después de terminar de escribir el código fuente, que se salva siempre con una extensión ***.java***, se compila el programa para generar un archivo binario que contiene el ***byte-code,*** este archivo tiene una extensión ***.class***. Estos archivos son los que el intérprete carga y ejecuta. Ya que los archivos ***byte-code*** son completamente portables entre los sistemas operativos, pueden ser ejecutados en cualquier sistema que tenga un intérprete de Java.

Pero Java no solo es un lenguaje de programación es mucho más que eso, es también un ambiente de programación, conocido como JDK (Java Developers Kit) que incluye todo un conjunto poderoso de herramientas de desarrollo. Estas herramientas incluyen:

- Un compilador (**javac**), el cual es el encargado de generar el bytecode.
- Un intérprete (**java**), el cual implementa la Máquina Virtual Java (JVM) y que se encarga de interpretar el bytecode y traducirlo al lenguaje máquina.
- Un depurador (**debugger**)
- Un desensamblador.
- Un visor de applets (**appletviewer**).
- Una herramienta para documentar los archivos fuente (**javadoc**).
- Un empaquetador de programas (**jar**)
- Un conjunto de paquetes que conforma la **API** de java.

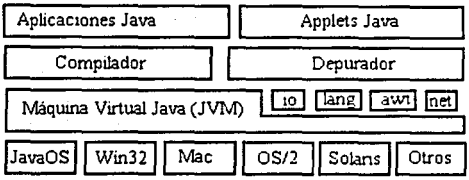


Figura 2.12 La plataforma de cómputo de Java

2.4.3 Características de Java.

Java es un lenguaje simple, orientado a objetos, compilado, independiente de plataforma, robusto, seguro y extensible. Pero es probable que sólo mencionando estos atributos no sirva de mucho para entender el verdadero poder del lenguaje. La siguiente lista de características ayudaran a entender los conceptos:

Simple. Los desarrolladores decidieron eliminar algunas características innecesarias de otros lenguajes de programación de alto nivel. Por ejemplo, Java no usa la aritmética de punteros tan común en el lenguaje C; no implemanta conversión implícita de tipos de datos; no hace uso de estructuras o uniones; no existe la sobrecarga de operadores; no se permite la herencia múltiple y no existen los archivos de cabecera.

Orientado a objetos. Al igual que C++, Java utiliza el concepto de **clase** para ordenar el código en módulos lógicos. Las clases de Java pueden heredar de otras clases, pero la herencia múltiple, en donde una clase hereda métodos y campos de más de una clase, no esta permitida. Los objetos son creados a partir de las clases en tiempo de ejecución.

Java tiene una amplia colección de clases, las cuales están organizadas en paquetes (packages), las cuales pueden ser utilizadas en cualquier programa. La clase **Object** (que se encuentra en el paquete java.lang) sirve como la clase raíz de la jerarquía de clases del lenguaje.

Compilado. Antes de que se pueda ejecutar un programa escrito en Java, se debe de compilar. El compilador genera un archivo byte-code, el cual puede ser interpretado en cualquier sistema operativo que tenga un intérprete Java. Este intérprete lee el archivo y traduce los comandos byte-code a los comandos en lenguaje máquina que se pueden ejecutar directamente por la computadora en la que se esta ejecutando el intérprete. Se puede decir entonces que **Java es un lenguaje compilado e interpretado.**

Portable. La posibilidad de tomar un programa Java escrito en una estación de trabajo Solaris y ejecutarlo en una PC o una Macintosh sin necesidad de modificar nada tiene un enorme potencial. La portabilidad también se extiende a la interfaz gráfica de usuario (GUI), la cual ha representado un serio problema en el desarrollo de aplicaciones distribuidas.

Dinámico y distribuido. Java es un lenguaje dinámico, lo que quiere decir que cualquier clase puede ser cargada en un programa que se este ejecutando en cualquier momento. Java también es llamado un lenguaje distribuido, principalmente por el alto soporte que proporciona para el trabajo en red. Esta característica es mejor apreciada cuando se combina con su capacidad de carga de clases. Juntas, estas caracterísitcas permiten que un programa pueda obtener y ejecutar el código desde cualquier parte de Internet.

Alto desempeño. Se podría pensar que al ser Java un lenguaje interpretado su desempeño se vería seriamente afectado. Sin embargo Java mantiene un alto nivel de desempeño utilizando para ello varias técnicas.

Multitarea. Los programas de Java pueden manejar varios hilos de ejecución, lo que permite a los programas manejar varias tareas en paralelo. Por ejemplo, un programa puede mostrar una imagen en pantalla mientras continua validando la entrada de información desde el teclado.

Seguro. Porque el intérprete de Java controla todos los accesos al sistema que realize un programa. Además se asegura de que ningun virus o código perjudicial este asociado con la ejecución de un programa. Los programas de Java no pueden hacer que el sistema se caiga. Cuando el intérprete descubre un error serio, se crea una excepción, esta excepción puede ser capturada y determinar acciones de acuerdo al tipo de excepción, pero sin ningun riesgo de tirar el sistema.

Debido a que los punteros no son utilizados por el lenguaje, los programas no pueden tener acceso a áreas del sistema en las cuales no tengan ninguna autorización.

Extensible. Los programas de Java pueden utilizar métodos nativos, los cuales son funciones escritas en otro lenguaje de programación. Este soporte permite generalmente a los programadores escribir funciones que pueden ejecutarse más rápidamente que las funciones equivalentes escritas en Java. Los métodos nativos se conectan dinamicamente al programa en Java, es decir, se asocian al programa en tiempo de ejecución.

Familiar. Esta es una de las características más importantes. Java esta derivado de los lenguajes C y C++ y muchas de sus palabras clave son idénticas a las de estos lenguajes, además, Java esta basado en el desarrollo de muchos lenguajes a lo largo de la historia, y reúne muchas características de ellos. Por esta razón, Java se puede entender rápidamente y fácilmente por cualquier persona con experiencia con lenguajes de programación modernos.

2.4.1 Applets.

Java se puede utilizar para crear dos tipos de programas: ***applets*** y ***aplicaciones independientes (stand-alone)***.

Un applet es una especie de mini-aplicación, diseñada para correr dentro del contexto de un visor de applets (applet viewer) como puede ser un web browser o bien alguna otra aplicación programada para este efecto. Los applets se diferencian de otras aplicaciones en muchas formas, una de las más importantes es que hay una serie de restricciones de seguridad que se aplican a los applets y las cuales indican las cosas que

un applet puede y no puede hacer cuando se esta ejecutando. Un applet a menudo puede considerarse como un programa o código **no confiable**, de tal manera que, por ejemplo, no debe permitírsele el acceso al sistema local de archivos, ya que un applet maligno podría borrar toda la información contenida en el disco duro de una computadora.

Desde el punto de vista de un programador, una de las diferencias entre un applet y las aplicaciones, es que los applets no tienen un método **main()** u otro punto de entrada desde el cual pueda indicarsele al programa que empiece a ejecutarse.

```
public class ejemplo1 {
   public static void main (String[] args) {
      System.out.println("Hola Mundo");
   }
}
```

Figura 2.13 Código de una aplicación stand-alone

```
import java.applet.*;
import java.awt.*;

public class ejemplo2 extends Applet {
   public void paint (Graphics g) {
      g.drawString ("Hola mundo", 10, 10);
   }
}
```

Figura 2.14 Código de un applet (puede observarse la ausencia del método main)

Capítulo 3

Mecanismos de entrega multipunto

El servicio más importante de una red de cómputo consiste en un sistema de entrega de paquetes, y el método más utilizado para este propósito es el de **unicast** o **unidifusión**, es decir una comunicación **punto-a-punto**. Este esquema de entrega funciona muy bien solo si se desea establecer comunicación entre dos computadoras; pero existen ocasiones en que es necesario que una computadora transmita paquetes de datos y que estos sean recibidos no por una, sino por más de una computadora al mismo tiempo.

Si se desea que una computadora en la red envie la misma información hacia varios destinos usando un servicio unicast, entonces es necesario generar *n* copias del paquete y enviar cada una de ellas a los *n* destinos deseados; esta técnica de transmisión de paquetes es conocida como **envío de mensajes unicast replicados.** La desventaja de este esquema de entrega es que mientras se desee hacer llegar los paquetes hacia más destinos se necesita mayor ancho de banda, el cual es un recurso finito y en ocasiones escaso.

Una mejor manera de transmitir datos desde una fuente hacia múltiples destinos es usando un servicio de entrega de **multidifusión.** Con un servicio multidifusión, una computadora puede enviar datos a varios destinos usando sólo una copia de información. Este mecanismo se basa en una técnica de transmisión de paquetes a un grupo de computadoras, las cuales comparten una sola dirección y que pueden encontrarse ya sea en una misma red o distribuidas en varias redes.

3.1 Entrega por difusión (broadcast)

La forma más común de entrega multipunto en la mayoría de las tecnologías de red es la difusión. La entrega por difusión significa que la red y no la computadora fuente entrega una copia de cada paquete para cada destino.

En muchas tecnologías de hardware de red (por ejemplo Ethernet), la difusión puede ser tan eficiente como la transmisión normal de un solo paquete; en otras, la difusión encuentra apoyo en el software de red pero implica un mayor retraso que la transmisión simple por hardware.

3.1.1 Direcciones de difusión IP.

Recordemos que una dirección IP codifica tanto la identificación de la red a la que esta conectada una computadora, así como la identificación de la misma computadora. Conceptualmente, cada dirección consiste un par de identificadores: **netid** y **hostid**, en donde *netid* identifica una red y *hostid* una computadora localizada dentro de esa red. La mayor ventaja de la codificación de información de las direcciones IP de esta manera consiste en permitir un ruteo eficiente.

Otra ventaja significativa del esquema de direccionamiento en una red de redes como Internet, es que éste incluye una **dirección de difusión** (figura 3.1) que se refiere a todas las computadoras dentro de una red. De acuerdo con el estándar, cualquier campo *hostid* consistente en solamente 1s, esta reservado para la difusión.

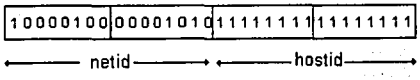


Figura 3.1 Dirección de difusión para una red de tipo B. El campo hostid solo consiste de 1s

3.1.1.1 Difusión limitada.

Técnicamente, *la dirección de difusión IP se conoce como dirección de difusión dirigida*, debido a que contiene tanto una identificación válida de red como el campo hostid de difusión. Una dirección de difusión dirigida se puede interpretar sin ambiguedades en cualquier punto de una red de redes ya que identifica en forma única a la red objetivo, además de especificar la difusión en dicha red. Las direcciones de difusión dirigida proporcionan un mecanismo poderoso que permite que un sistema remoto envie un solo paquete que sera publidifundido en la red especificada.

Desde el punto de vista del direccionamiento, la mayor desventaja de la difusión dirigida es que requiere un conocimiento de la dirección de red. Otra forma de dirección de difusión, llamada **dirección de difusión limitada** o **dirección de difusión en red local** (figura 3.2), proporciona una dirección de difusión para la red local, independientemente de la dirección IP asignada. La dirección de difusión local consiste en treinta y dos unos.

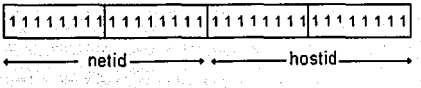


Figura 3.2 Una dirección IP de difusión limitada a la red local consta de treinta y dos unos

Una computadora puede utilizar la dirección de difusión limitada como parte de un procedimiento de arranque antes de conocer su dirección IP o la dirección IP de la red local. Sin embargo, una vez que la computadora conoce la dirección IP correcta para la red local, tiene que utilizar la difusión dirigida.

Como regla general, los protocolos TCP/IP restringen la difusión al menor número posible de máquinas.

3.1.1.2 Difusión a las subredes.

La técnica más utilizada para permitir que una sola dirección de red abarque muchas redes físicas se conoce como *direccionamiento de subred, ruteo de subred o utilización de subredes.*

La manera más sencilla de entender el direccionamiento de subred es imáginandose que una localidad tiene asignada una sola dirección IP de tipo B, pero tiene dos o más redes físicas. Sólo los ruteadores locales saben que existen muchas redes físicas y como rutear el tráfico entre ellas. El resultado es una forma de ruteo jerárquico.

Llevar a cabo la difusión en una arquitectura de subred es más difícil. Recordemos que el campo hostid llenado con 1s denota difusión a todas las computadoras en la red especificada en el campo netid. Desde el punto de vista de subred, esta dirección indica que se debe entregar una copia del paquete a todas las máquinas en la red, inclusive si residen en subredes físicas separadas. Operacionalmente, la difusión hacia una dirección así sólo tiene sentido si todos los ruteadores que interconectan a las subredes están de acuerdo en propagar el paquete hacia todas las redes físicas.

Dentro de un grupo de redes con subredes, es posible transmitir por difusión hacia una subred específicica indicando la subred a la que se desea hacer llegar la difusión.

3.2 Entrega por multidifusión (Multicast)

Una mejor manera de transmitir datos desde una fuente hacia múltiples destinos es usando un servicio de entrega de **multidifusión**. Con un servicio multidifusión, un nodo puede enviar datos a varios destinos usando sólo una copia de información.

Usando multidifusión se puede enviar un paquete a 5, 10, 100 o más computadoras al mismo tiempo, por lo que el ancho de banda es usado de una manera más eficiente. Además no es necesario conocer las direcciones de todas las computadoras a las que se desea enviar la información, ya que todas comparten una

misma dirección lógica y los mensajes se hacen llegar a esa dirección sin importar que se encuentren en distintas redes físicas.

Se puede hacer una analogía entre la entrega por multidifusión y las señales de televisión. Una estación de televisión origina una señal, la cual, para ser transmitida, toma solo una parte del ancho de banda disponible. De esta manera cualquier persona que tenga una televisión puede captar la señal, la señal es ignorada por aquellas personas quienes no quieren sintonizarla o por quienes no tienen un aparato de televisión, y si alguna persona apaga su televisor nadie se verá afectado por esta acción.

3.2.1 Multidifusión IP : direcciones de clase D.

En TCP/IP la multidifusión se vale de la dirección de destino del datagrama IP para especificar que se va a realizar una entrega de multidifusión. La multidifusión IP utiliza direcciones especiales de **clase D**. Los primeros cuatro bits contienen **1110** e identifican a la dirección como de multidifusión (figura 3.3). Los 28 bits restantes (recordemos que una dirección IP consta de 32 bits) especifican a un grupo de multidifusión en particular. No existe otra estructura en el grupo de bits. En particular, el campo de grupo no identifica el origen del grupo ni contiene una dirección de red como en las direcciones A, B o C. Cuando se expresa en notación decimal con puntos, el rango de direcciones de multidifusión abarca de la dirección **224.0.0.0** a la **239.255.255.255.**

Las direcciones de multidifusión IP sólo pueden emplearse como direcciones de destino y no de origen. Estas nunca podrán aparecer en el campo de dirección fuente de un datagrama ni pueden aparecer en una ruta fuente o en el registro de una opción de rutas.

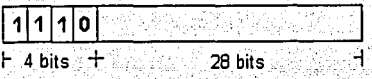


Figura 3.3 Formato de las direcciones IP de tipo D (Multidifusión)

A pesar de que el rango de direcciones multidifusión es muy amplio, no todas las direcciones estan disponibles para su uso arbitrario, debido a que algunas direcciones han sido reservadas para usos específicos. Entre estas direcciones reservadas tenemos a la dirección **224.0.0.0** la cual no puede ser asignada a ningún grupo; la dirección **224.0.0.1** es una dirección importante, ya que esta asignada permanentemente al **grupo de todos los hosts**, el cual incluye a todas las computadoras y ruteadores que participan en la multidifusión IP en una red local; la dirección **224.0.0.2** incluye a todos los ruteadores multidifusión en la LAN; el rango de direcciones **224.0.0.0** al **224.0.0.255** está reservado para los protocolos de ruteo y protocolos de mantenimiento. Otras direcciones y rangos han sido reservados para aplicaciones específicas, tal como el rango de la dirección **224.0.13.0** a la **224.0.13.255** el cual esta reservado para el servicio de Net News. Todas las direcciones reservadas aparecen en el RFC 1700 "Números asignados", asimismo la IANA (Internet Assigned Numbers Authority) también mantiene una lista de grupos de multidifusión registrados y de las direcciones asignadas a estos grupos.

3.2.2 Grupos de multidifusión IP.

El direccionamiento por multidifusión proporciona una forma limitada de difusión en la cual un subconjunto de computadoras en una red acuerda compartir una dirección de multidifusión (dirección especial de clase D), de esta manera, todas las máquinas en el grupo recibirán una copia de cada paquete enviado hacia tal dirección de multidifusión. Al conjunto de computadoras participantes se le conoce como **grupo de multidifusión**, y puede ser un grupo de cero o más computadoras.

Para unirse a un grupo de multidifusión, los programas de aplicación dentro de una computadora deben indicarle a la tarjeta de red que acepte los paquetes que van dirigidos hacia las direcciones de multidifusión en las que desean participar. La participación en un grupo multidifusión es dinámica, esto quiere decir que cualquier computadora puede integrarse o abandonar un grupo multidifusión en cualquier momento. No hay una restricción en cuanto a la localización o la cantidad de miembros y una computadora puede ser miembro de más de un grupo al mismo tiempo.

A diferencia de la difusión, en la cual todos los mensajes llegan a todas las terminales sin importar si quieren recibir la información o no, la multidifusión permite que cada estación elija si quiere participar en la comunicación y de que manera va a participar en ella. Así pues, una computadora tiene tres opciones al intervenir en una comunicación multidifusión:

a) no participar en ella;
b) participar solo enviando, pero no recibiendo y
c) participar enviando y recibiendo.

La primera y segunda opciones son sencillas de lograr. La tercera es más compleja, ya que la computadora deberá mantener un registro de que aplicaciones desean participar en determinado grupo de multidifusión, para pasarles las copias de los paquetes correspondientes, además deberá informar al ruteador local a que grupos pertenecen las aplicaciones correspondientes. Adicionalmente, si la computadora tiene múltiples tarjetas de red, puede ser miembro de grupos diferentes en cada una de ellas.

Los grupos multidifusión pueden ser catalogados como **permanentes** o **temporales**. Un grupo permanente tiene asignada una dirección *bien conocida* (asignada por la IANA), y todas aquellas direcciones multidifusión que no estan reservadas para grupos permanentes están disponibles para su asignación a grupos temporales.

La ventaja de la multidifusión reside en la capacidad para limitar la difusión: todas las computadoras en un grupo de multidifusión pueden ser alcanzadas con un solo paquete de transmisión, además de que las computadoras que eligen no participar en algún grupo en particular no reciben los paquetes enviadas a los grupos de multidifusión aún cuando se encuentren en la misma red.

3.2.3 Envio y recepción de paquetes IP multidifusión.

Para enviar un paquete multidifusión, el transmisor especifica la dirección del grupo de destino. Los paquetes multidifusión son enviados de la misma manera que son enviados los paquetes unicast.

Cuando se deseen enviar datagramas al grupo de multidifusión, sólo será necesario especificar en el campo de dirección destino del datagrama IP, la dirección multidifusión que tiene asignada el grupo. Un datagrama multidifusión es enviado a todos los miembros del grupo de estaciones con la misma filosofía de **entrega con el mejor esfuerzo** de los datagramas IP convencionales: no se garantiza la entrega del datagrama a todos los miembros del grupo ni la pérdida de paquetes, ni el orden con respecto a otros datagramas.

Comparada con el envío de paquetes, la recepción multidifusión es mucho más compleja, particularmente sobre una WAN. Para recibir datagramas, una aplicación debe pedir darse de alta en un grupo multidifusión determinado. La petición de pertenencia al grupo es enviada al ruteador y si es necesario éste la pasa a un ruteador intermedio y así para todos los ruteadores intermedios entre el nuevo nodo y todos los nodos que ya están dados de alta en el grupo. Al mismo tiempo, la tarjeta de red empieza a identificar los mensajes que van dirigidos a la dirección del grupo en el que se acaba de dar de alta para poder procesar cualquiera que vaya dirigido al grupo.

Los ruteadores intermedios (si los hay) hacen llegar entonces la petición de pertenencia a los ruteadores de los nodos finales, los cuales mapean entre la dirección de grupo y su dirección física asociada y envían el mensaje usando esta última dirección. Las tarjetas de red de los receptores entonces reciben el mensaje y pasan el paquete a las capas de TCP/IP, las cuales lo procesan y envían su contenido a la aplicación final.

A diferencia de las direcciones unicast, donde una tarjeta de red esta asociada a una sola dirección IP, una dirección multidifusión esta asociada dinámicamente a una o más tarjetas de red localizadas en una o más redes. *Una dirección de grupo multidifusión no esta atada a un conjunto de direcciones unicast.* Los ruteadores que manejan multidifusión no necesitan saber la lista o la cantidad de miembros que existen en cada grupo sino sólo los grupos para los cuales hay por lo menos un miembro en la subred. Un ruteador multidifusión conectado a una red Ethernet sólo necesita asociar una sola dirección Ethernet multidifusión para cada grupo que contenga un miembro local.

3.2.3.1 Campo Time To Live (TTL) del datagrama IP.

Cada paquete multidifusión usa el campo TTL (Time To Live) del encabezado del datagrama IP como un parámetro para definir el alcance de un paquete. Este campo especifica la duración, en segundos, del tiempo que el datagrama tiene permitido permanecer en la red. La idea es sencilla, el campo TTL controla el número de saltos que un paquete multidifusión puede efectuar. Cada vez que un ruteador reenvía un paquete, este campo es decrementado en uno. Un paquete multidifusión cuyo campo TTL haya expirado (igual a 0) es descartado por los ruteadores, sin enviar ningún mensaje de error al transmisor. Este mecanismo previene que los mensajes permanezcan un tiempo indeterminado en la red y ocasionen tráfico innecesario.

Si un transmisor especifica en el campo TTL un valor de 1, el mensaje llega a todos los miembros que se encuentren en la red local. Si en un paquete se especifica un valor TTL mayor a 1, los ruteadores multidifusión conectados a la red local toman entonces la responsabilidad de reenviar el paquete. El paquete es enviado hacia otras redes que tengan algún miembro en el grupo de destino y que puedan ser alcanzadas mientras el TTL no sea cero.

En algunos casos, si este campo es menor a algún valor determinado, los ruteadores no lo reenvían, previniendo de esta manera que los paquetes salgan o pasen por algunas subredes. Esto provee un mecanismo para limitar el tráfico multidifusión a solo algunas subredes (campos universitarios o redes empresariales).

Los valores más comunes para el campo TTL son:

- **1** para hacer llegar mensajes en una red local.
- **15** para un sitio o redes privadas.
- **63** para una región (por ejemplo un país)
- **127** para todo el mundo.

3.2.4 Protocolo IGMP.

¿Cómo saben los ruteadores multidifusión cuando deben entregar algún paquete multidifusión dentro de sus subredes? Para poder entregar los paquetes multidifusión eficientemente, los ruteadores que intervienen en la transmisión multidifusión deben conocer si es que existe al menos un miembro de algún grupo multidifusión en alguna de las redes a las que están conectados directamente para entregar el paquete. El ruteador local se pone en contacto con otros ruteadores multidifusión, pasando información hacia los miembros y estableciendo rutas. La idea es muy similar a la difusión de rutas tradicional entre ruteadores de redes convencionales.

El protocolo IGMP (Internet Group Management Protocol) es usado por los ruteadores para determinar la existencia de miembros de algún grupo multidifusión en la(s) red(es) a la(s) que están directamente conectados. La primera versión del protocolo data de 1988 y actualmente es un estándar Internet. Su descripción se encuentra en el RFC 1112, y de acuerdo a esta descripción los mensajes IGMP se envían dentro de los paqutes IP de acuerdo al siguiente formato que se muestra en la figura 3.5.

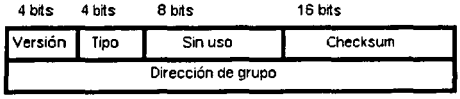


Figura 3.5 Formato de envío de mensajes IGMP

Dónde:

- **Versión** indica la versión del protocolo; actualmente 1.
- **Tipo**, el cual indica el tipo de mensaje que se envía, puede tomar dos valores:

1 = Petición de pertenencia a grupo
2 = Reporte de pertenencia a grupo

- **Sin usar**, se llena con ceros cuando se envía y se ignora cuando es recibido

- **Checksum**, es un numero binario de 16 bits que es el complemento a uno de los 64 bits que conforman el mensaje.
- **Dirección de grupo**, cuando se envía un mensaje de Petición de pertenencia a grupo este campo es generalmente llenado con ceros; en un mensaje de Reporte de Pertenencia a Grupo el campo se llena con la dirección de grupo que es reportada.

Los mensajes IGMP son encapsulados sobre paquetes IP. El protocolo es hasta cierto punto muy sencillo, ya que solo maneja dos tipos de mensajes: **Petición de Pertenencia a Grupo** y **Reporte de Pertenecia a Grupo**. Para determinar si existe algún miembro de un grupo multidifusión en una subred local, el ruteador se encarga de enviar periódicamente un mensaje IGMP multidifusión de *Petición de Pertenencia a Grupo* a cada una de las subredes a las que se encuentra directamente conectado, pidiendo a las computadoras en las subredes que contesten si es que están dadas de alta en algún grupo multidifusión.

El mensaje es enviado a la dirección 224.0.0.1 (grupo de todas las computadoras) con un TTL con valor de 1, para asegurar que el mensaje no llegue más alla de la red local. Cada computadora que pertenezca a algún grupo multidifusión responde con un mensaje IGMP de Reporte de Pertenencia a Grupo a la dirección de grupo a la que pertenence, por lo que la respuesta llega a todos los miembros de su grupo que existan dentro de la subred (figura 3.6)

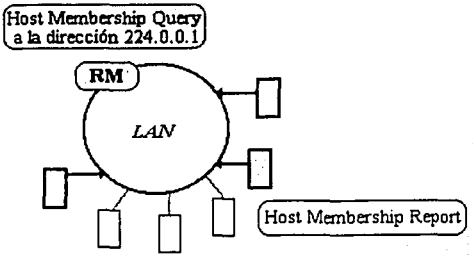


Figura 3.6 Funcionamiento del protocolo IGMP.

Si existe más de un miembro de un grupo multidifusión en la misma subred no es necesario que todos los miembros respondan, ya las respuestas de todos ocasionarían tráfico innecesario en la red. Para solucionar este problema, las computadoras esperan un tiempo aleatorio para responder que pertenecen a un grupo y sólo envían su respuesta si no han recibido la respuesta generada por algún otro miembro, de esta manera se asegura que sólo un miembro responda y que no se congestione el tráfico de la red con respuestas innecesarias.

IGMP también es utilizado por los protocolos de ruteo multidifusión para intercambiar información entre ruteadores acerca de si tienen miembros de grupos multidifusión conectados a sus redes. IGMP también es usado para identificar a un ruteador dedicado especialmente a este servicio dentro de una red local.

El ancho de banda necesario para intercambiar toda la información de pertenencia a grupos multidifusión usado por el protocolo IGMP es casi nulo, por lo que el método en la mayoría de los casos funciona eficientemente.

3.2.5 Ruteo multidifusión.

Rutear tráfico multidifusión es un problema aún más complejo.

Los algoritmos y protocolos de ruteo multidifusión generalmente siguen alguna de dos técnicas básicas, dependiendo principalmente de la distribución de los miembros del grupo a través de la red.

La primera técnica esta basada en la suposición de que los grupos multidifusión están uniformemente distribuidos en la red y de que el ancho de banda es muy grande, por ejemplo, que todos los nodos de la red pertenezcan al grupo. Los protocolos que utilizan esta técnica son el DVMRP (Distance Vector Multidifusión Routing Protocol o Protocolo de Vector de Distancias Multidifusión), MOSPF (Multidifusión Open Shortest Path First) y PIM-DM (Protocol Independent Multidifusión - Dense Mode).

La segunda técnica de ruteo multidifusión esta basada en la suposición de que los miembros de un grupo multidifusión están esparcidos por toda la red y que el ancho de banda tal vez no sea en todos los casos el deseado, por ejemplo, en muchas subredes a través de Internet. En este caso los protocolos de ruteo deben usar técnicas más selectivas para trabajar apropiadamente. Entre los protocolos que implementan esta

técnica están: CBT (Core Based Trees) y PIM-SM (Protocol Independent Multidifusión - Sparse Mode).

3.2.6 Componentes de una implementación de IP multidifusión "pura".

Para poder implementar una comunicación de tipo multidifusión pura (es decir, que no se acuda a ninguna otra técnica especial para lograr la comunicación), los nodos participantes en ella (transmisores y receptores) y la infraestructura de red entre ellos (ruteadores intermedios y finales) deben ser configurados para soportar comunicación multidifusión. Los requerimientos para los nodos finales incluyen:

- Soporte para la transmisión y recepción de paquetes IP multidifusión en su implementación del protocolo TCP/IP.
- Software que implemente IGMP para comunicar peticiones de pertenencia a un grupo multidifusión y recibir tráfico multidifusión.
- Tarjetas de red que interpreten adecuadamente las direcciones multidifusión.
- Software de aplicación elaborado especialmente para correr sobre IP multidifusión.

El tráfico en la red puede ser optimizado usando switches que funcionen como **filtros multidifusión** (figura 3.7). Un switch funcionando como filtro permite que los paquetes sean enviados únicamente a los nodos participantes, ya que de otra manera, el tráfico se expanderia a todos los segmentos de la red local.

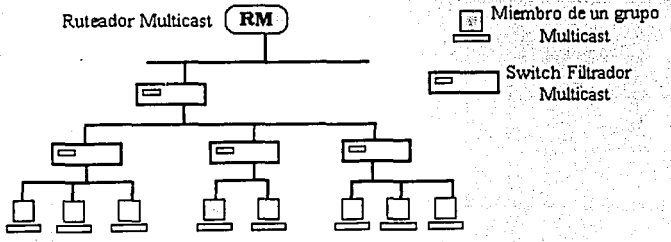


Figura 3.7 Filtros multicast.

Los requerimientos anteriores son suficientes si se quiere implementar una comunicación multidifusión en una LAN; si se desea expandir el tráfico de comunicación multidifusión a una WAN entonces se requiere que:

- Todos los ruteadores intermedios entre los transmisores y receptores soporten puedan direccionar paquetes multidifusión.
- Si existen firewalls, será necesario reconfigurarlos para que permitan el paso a los paquetes multidifusión.

Actualmente la mayoría del hardware y software de red soportan IP multidifusión, elementos tales como ruteadores, switches, tarjetas de red, sistemas operativos, implementaciones de TCP/IP y software de aplicación manejan sin ningún problema la comunicación multidifusión como se muestra en la figura 3.8.

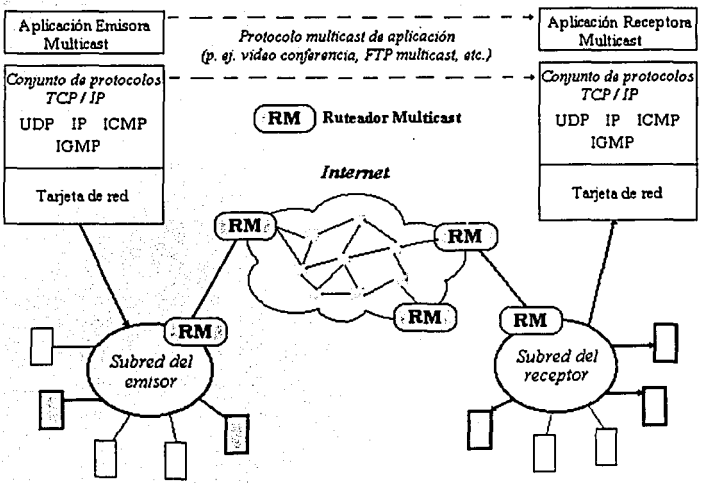


Figura 3.8 Componentes para una implementación Multicast.

3.2.7 La red multitcast MBONE.

Desafortunadamente hoy en día no se puede implementar una red multidifusión pura, ya que una gran cantidad de los ruteadores conectados a Internet no conocen como manejar los mensajes multidifusión, o los que lo saben hacer, no están configurados para rutear este tipo de paquetes, por lo que todas las transmisiones en Internet son hechas a través de mensajes unicast.

En 1992, la IETF (Internet Engineering Task Force) decidió atacar este problema, y decidió implementar en software lo que no se podía hacer en hardware: el ruteo multidifusión a través de una red unicast. De esta manera se decidió construir una **red virtual** sobre la base ya instalada (Internet) y el software que permitía enviar paquetes multidifusión a través de dicha red virtual. Así fué como nació el **MBONE** (Multicast Backbone), una red virtual que comparte los mismos medios que Internet pero hasta cierto punto independiente de ella.

El MBONE permite que los paquetes multidifusión puedan pasar a través de ruteadores que sólo conocen como manejar tráfico unicast. El software que utiliza al MBONE **encapsula** los paquetes multidifusión dentro de paquetes unicast de tal manera que los ruteadores puedan manejarlos y entregarlos adecuadamente.

Esta técnica de encapsulamiento de paquetes multidifusión (figura 3.10) dentro de paquetes unicast es conocida como **tuneles multidifusión** (figura 3.11). Esta técnica permite conectar nodos de grupos multidifusión a través de una infraestructura de red que no soporta el ruteo multidifusión. Al **encapsular** los datagramas multidifusión en un datagrama unicast, se logra hacerlos llegar a su destino, ya que cualquier ruteador soporta y sabe manejar la comunicación unicast.

Cuando los paquetes unicast que llevan encapsulados los paquetes multidifusión llegan a un ruteador que si sabe como manipular los paquetes multidifusión o si llegan a una estación de trabajo que este ejecutando el software apropiado, los paquetes son reconocidos como el tipo de paquetes que en realidad son (paquetes multidifusión) y son enviados finalmente a los miembros del grupo multidifusión destino.

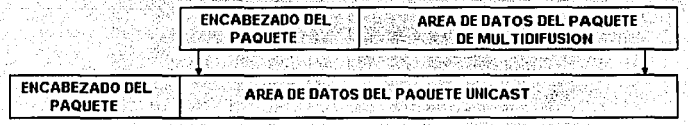


Figura 3.10 Un paquete de multidifusión es encapsulado en un datagrama IP convencional. Los ruteadores de multidifusión utilizan esta encapsulación para realizar la entrega mediante tuneles a través de ruteadores que no manejan la multidifusión.

En MBONE las computadoras (estaciones de trabajo o ruteadores) que soportan multidifusión reciben el nombre de **mrouter (multidifusión router)**. Los mrouters son ruteadores que pueden manejar tráfico multidifusión, o más comúnmente son estaciones de trabajo que están ejecutando un software especial y que interactuan con los ruteadores estándar.

Entonces ¿cuál es la diferencia entre multidifusión y el MBONE?. Multidifusión es un método para enviar paquetes a más de un sitio a la vez. El MBONE es un conjunto de subredes que implementan multidifusión y que usan la infraestructura unicast de Internet para intercambiar paquetes multidifusión.

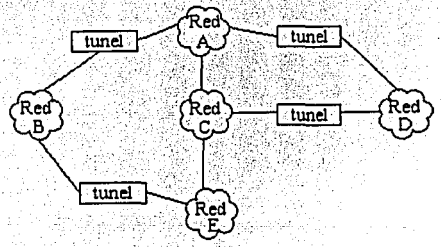


Figura 3.11 Tuneles que permiten la comunicación Multicast sobre Internet.

Hoy en día multidifusión es usado para aplicaciones de audio, herramientas de trabajo en grupo, etc. Las conferencias multidifusión generalmente involucran tres tipos de medios: audio, video y pizarrón electrónico (una especie de block de notas que todos los participantes pueden compartir). Tal vez la más común aplicación del MBONE es la videoconferencia. Desde luego la calidad de transmisión de video a través del MBONE aún no es muy buena, pero con sólo algunos frames por segundo, la calidad del video es suficientemente buena para la mayoría de los propósitos.

Una vez que hemos analizado los distintos mecanismos de entrega multipunto y que se han mostrado sus ventajas contra un servicio de entrega punto a punto, estamos en posibilidad de sacar partido de tales ventajas para lograr uno de nuestros objetivos: implementar un mecanismo de único-envío, múltiple recepción para la distribución de documentos; pero antes de mostrar como haremos esto, es necesario detenernos un poco y ver que tipo de documentos queremos entregar o distribuir.

Es por eso que el siguiente capítulo trata del programa LaTeX, un programa utilizado principalmente para la elaboración de documentos que contengan texto matématico.

Capítulo 4

El programa LaTeX

Elaborar documentos que contengan simbología matemática no es precisamente una tarea que pueda considerarse fácil de realizar, aún con la ayuda de los poderosos procesadores de palabras actuales, tales como Word de Microsoft, WordPerfect o Lotus Word Pro.

Estos procesadores de palabras ayudan a elaborar documentos de gran calidad, muestran en pantalla una presentación muy cercana al resultado que se obtendrá al momento de imprimir el documento, gracias a que implementan la característica **WYSIWYG** (lo que ve es lo que obtiene, por sus siglas en inglés). Sin embargo, aún cuando estos paquetes son muy poderosos, en ocasiones es muy complicado poder incluir simbología matemática en los documentos que se generan con ellos, lo que presenta una gran desventaja para las personas que quieren elaborar documentos con estas características.

Afortunadamente existe otro tipo de aplicaciones que llenan este nicho descuidado por los procesadores de palabras. De entre estas aplicaciones una que destaca significativamente es **LaTeX.**

LaTeX (pronunciado ***latek***) es un ***programa formateador de textos*** más que un procesador de palabras. Fué concebido con la finalidad de que el usuario no se preocupe por la apariencia de los documentos y en cambio se concentre más en el contenido de los mismos. Pero sobre todo, facilita la manipulación de simbología matemática, lo que permite insertar ecuaciones matemáticas muy complejas, que con otros paquetes es muy difícil obtener.

4.1 TeX y LaTeX.

LaTeX tiene sus orígenes en el programa TeX el cual fué escrito por Donald E. Knuth. TeX es un lenguaje de composición de documentos muy robusto y extensible, que funciona mediante comandos que especifican el cómo, cuando y donde aparecen los elementos de un texto.

TeX funciona de la siguiente manera: el usuario escribe en un editor de textos ASCII un archivo que contiene tanto el texto y los comandos que especifican el formato que se aplicará al documento. Este archivo (llamado a menudo ***archivo de entrada***) es compilado entonces por el programa TeX el cual, si no detecta algún error, devuelve un nuevo archivo con la representación binaria de las páginas del documento.

En 1985, poco tiempo después de la aparición de TeX, Leslie Lamport creó una serie de **macros** (programas) que facilitaban aún más la tarea de crear un documento usando TeX, estas macros usan los comandos TeX que especifican la estructura lógica de un texto, tales como el formato del título, las numeraciones para las páginas, los capítulos, las secciones, figuras, referencias, etc. En parte por su apellido y en parte para señalar la flexibilidad de su invención, Leslie Lamport bautizó a su sistema con el nombre de LaTeX.

4.2 Características de LaTeX.

LaTeX es un programa *formateador de textos* muy poderoso y flexible usado para la elaboración de documentos principalmente en ambientes UNIX, gracias a lo cual se ha convertido en una herramienta muy popular entre los usuarios de este sistema operativo, quienes lo han convertido prácticamente en un formato estándar con el cual se pueden elaborar y presentar artículos de gran calidad a muchas publicaciones científicas y técnicas alrededor del mundo.

LaTeX fué diseñado para permitir que los usuarios no se preocupen por la apariencia de los documentos, sino para que se concentren más en el contenido del mismo y dejar la presentación al programa. Un típico archivo de entrada para LaTeX luciría como el de la *figura 4.1*.

```
\documentstyle{article}
    \title{Ejemplo}
    \author{Eduardo Martinez}
    \date{Marzo 2001}
\begin{document}
\maketitle
\center{Este es un documento escrito
 en \LaTeX}
\end{document}
```

Figura 4.1 Texto fuente de un documento LaTeX

Una vez que se procesa este archivo de entrada con LaTeX se obtiene el resultado que se muestra en la *figura 4.2.*

Ejemplo

Eduardo Martínez

Marzo 2001

Este es un documento escrito en LaTeX

Figura 4.2 Resultado final al formatear el texto fuente con LaTex

4.3 Elaboración de documentos.

Elaborar un documento con LaTeX es muy parecido a escribir un programa en algún lenguaje de programación: primero debe editarse el archivo fuente (archivo de entrada) y después debe compilarse. La **edición** puede llevarse a cabo con cualquier editor ASCII y contienen tanto el texto del documento como los comandos que definen el formato del documento; la **compilación** se lleva a cabo con LaTeX (*figura 4.3*)

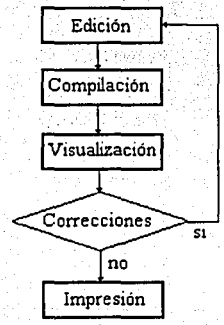


Figura 4.3 Ciclo de creación de un documento LateX.

Al momento de compilar un archivo de entrada, LaTeX puede encontrarse trabajando en alguno de los siguientes modos:

- ***Modo párrafo.*** Este modo es el más común y se activa cuando LaTeX esta procesando texto ordinario. En este modo, LaTeX divide el texto en líneas y en páginas.
- ***Modo matemático.*** LaTeX se encuentra en modo matemático cuando esta procesando una fórmula matemática. En este modo los caracteres y las cadenas de caracteres son considerados como símbolos matemáticos y

cualquier espacio en blanco es ignorado, salvo aquellos que son necesarios para determinar el final de algún comando.

Una vez compilado el archivo de entrada, se generará un archivo binario, que contiene la descripción del documento, dicho archivo puede ser entonces abierto con otro programa para visualizarlo en pantalla o bien mandarlo inmediatamente a la impresora.

El compilador se encargará de dar el formato necesario al texto, de acuerdo al tipo de documento y a los comandos de formato. El texto se mostrará en el mismo orden y con todos sus caracteres tal y como aparece en el archivo. En general LaTeX considera a los espacios en blanco que aparezcan en el texto como separadores de palabras, y cualquier cantidad de espacios en blanco consecutivos son considerados como uno solo, asimismo una línea en blanco define un nuevo párrafo y al igual que con los espacios en blanco, cualquier cantidad de líneas vacías son equivalentes a una sola.

4.3.1 Caracteres especiales.

Los siguientes caracteres juegan un papel especial en el ambiente de LaTeX y son llamados caracteres especiales de impresión o simplemente **caracteres especiales**:

```
# $ % & ~ _ ^ \ { }
```

El símbolo **$** es usado para delimitar el inicio y final de una ecuación matemática; el caracter **%** es usado para insertar comentarios en el archivo de entrada, también puede ser utilizado para poder incluir un salto de línea en el archivo fuente sin que afecte la continuidad del texto; la diagonal (****) es utilizada en las definiciones de comandos; las llaves **{ }** son usadas para encerrar los argumentos de los comandos o para delimitar el alcance de un comando; el underscore (**_**) es usado para generar subíndices; el caracter circunflejo (**^**) es usado para generar superíndices; el ampersand (**&**) se utiliza para separar los elementos dentro de un arreglo.

En algunas ocasiones se deseará mostrar alguno de los caracteres especiales dentro del documento, por lo que hay que anular el significado especial que tienen para

LaTeX, para esto hay que usar una técnica presente en varios lenguajes de programación (p. ej. C o Perl) llamada *escape de caracteres*, la cual consiste en poner una diagonal (\) precediendo al caracter al que se le quiere quitar el significado especial. Por ejemplo, si se desea incluir el símbolo `$` dentro de alguna línea de texto sólo hay que escribir `\$`. Se sigue el mismo procedimiento para otros caracteres especiales, por ejemlo, la secuencia `\#` mostrará un `#` , `\{` generará `{`, etc.

4.3.2 Comandos.

El archivo de entrada, además de contener el texto del documento, también incluye los **comandos** que definen la estructura: el formato, el tipo y el tamaño de las fuentes, la numeración de las páginas, posicionamiento de los títulos y encabezados, la generación de ecuaciones, solo por mencionar algunos (figura 4.4). Estos comandos son visibles y entendibles (recordemos que se encuentran escritos en texto ASCII). Para obtener el resultado final, LaTeX requiere compilar el archivo de entrada para procesarlo y a partir de las definiciones de los comandos encontrados poder generar el documento final.

Un comando en LaTeX se declara de alguna de las dos siguientes formas:

- Una diagonal (\) seguida por un caracter especial
- Una diagonal (\) seguida por una cadena de caracteres.

Los comandos deben de ser terminados con un caracter que no sea una letra (nonletter): un espacio en blanco, un número, un signo de puntuación o un caracter especial. Además algunos comandos aceptan que se les pasen argumentos, algunos de los cuales son obligatorios y otros son opcionales. Los argumentos obligatorios se especifican dentro de un par de llaves (`{ }`) mientras que los argumentos opcionales se especifican dentro de un par de corchetes (`[ ]`). Los comandos son *caso-sensibles*, es decir, que un comando escrito con mayúsculas es totalmente distinto a uno escrito con minúsculas, por ejemplo el comando `\Gamma` que muestra la letra griega gamma en

mayúsculas (Γ), es diferente del comando **\gamma** que muestra la misma letra pero en minúsculas (γ).

4.3.3 Estructura general de un documento en LaTeX.

Todo archivo de entrada de LaTeX debe seguir la siguiente estructura general:

```
\documentstyle{estilo}
preámbulo
\begin{document}
cuerpo
%comentarios
\end{document}
```

El **preámbulo** contiene los comandos que afectan el formato de todo el documento. El **cuerpo** contiene el texto y los comandos que afectan algunas porciones del texto. Se pueden agregar **comentarios** al archivo para documentar ciertas partes del documento. Para insertar un comentario sólo hay que poner un símbolo de porcentaje (%), y cualquier cosa que siga hasta el final de la línea donde aparece la marca de comentario es ignorado por el compilador. Consideremos el siguiente ejemplo:

```
\documentstyle{article}
\begin{document}
Este es un documento sencillo.
Este es un ejemplo de ecuación $ E = mc^2 $
\end{document}
```

El comando **\documentstyle{}** define el **ambiente global** del documento, el cual determina el tamaño de los encabezados, tamaño de los márgenes, espacios entre líneas, ambientes locales entre otras muchas características, de acuerdo al estilo que se le indique, en este caso se aplicarán las características de un artículo (article).

Los comandos **\begin{}** y **\end{}** definen un **ambiente local**. Todas las líneas que se encuentren entre **\begin{document}** y **\end{document}** definen el cuerpo del documento. Las líneas que se encuentren entre los comandos **\documentstyle{}** y **\begin{document}** constituyen el **preámbulo**. La mayoría de los comandos que pueden modificar el ambiente global aparecen en esta parte.

Los pasos a seguir para elaborar un documento usando LaTeX pueden resumirse en:

1. Incluir el texto y los comando en un solo archivo (ver *figura 4.4*).

a) Elaborar el archivo en cualquier editor ASCII

b) Darle al archivo la extensión .tex

2. Compilar el archivo con LaTeX (ver *figura 4.5*)de la siguiente manera:

latex archivo.tex

El resultado es un archivo con el mismo nombre pero con extensión dvi (*archivo.dvi*).

3. Ver el archivo dvi (ver *figura 4.6*). Si no se esta satisfecho con el resultado o son necesarias algunas correcciones entonces regresar al paso No. 1, de otra manera ir al paso No. 4.

4. Impresión del documento final.

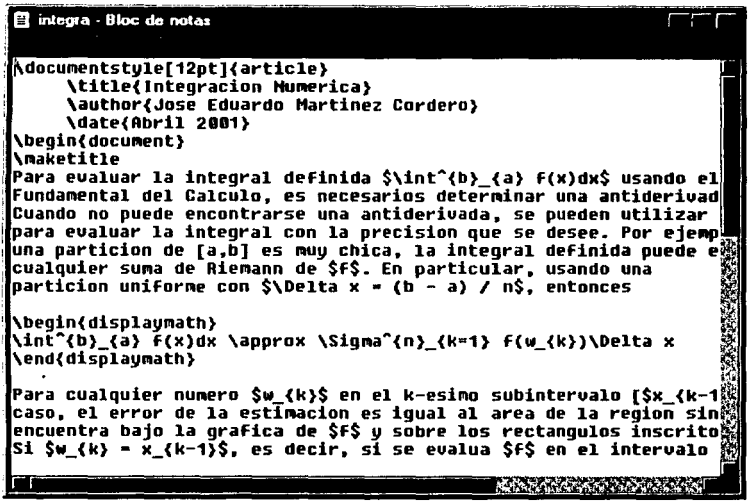


Figura 4.4. Edición de un documento LaTeX con un editor de textos ASCII

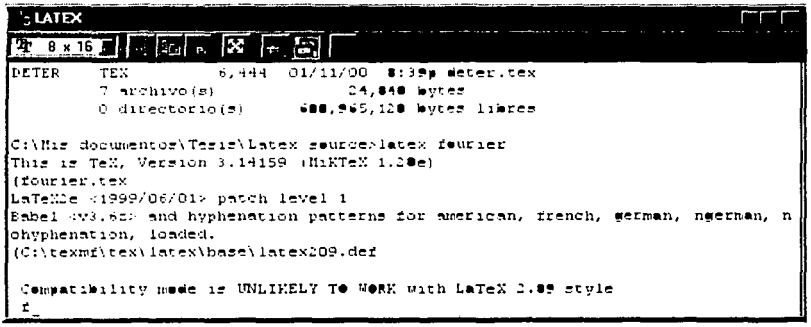


Figura 4.5. Inicio de la compilación de usando LaTeX.

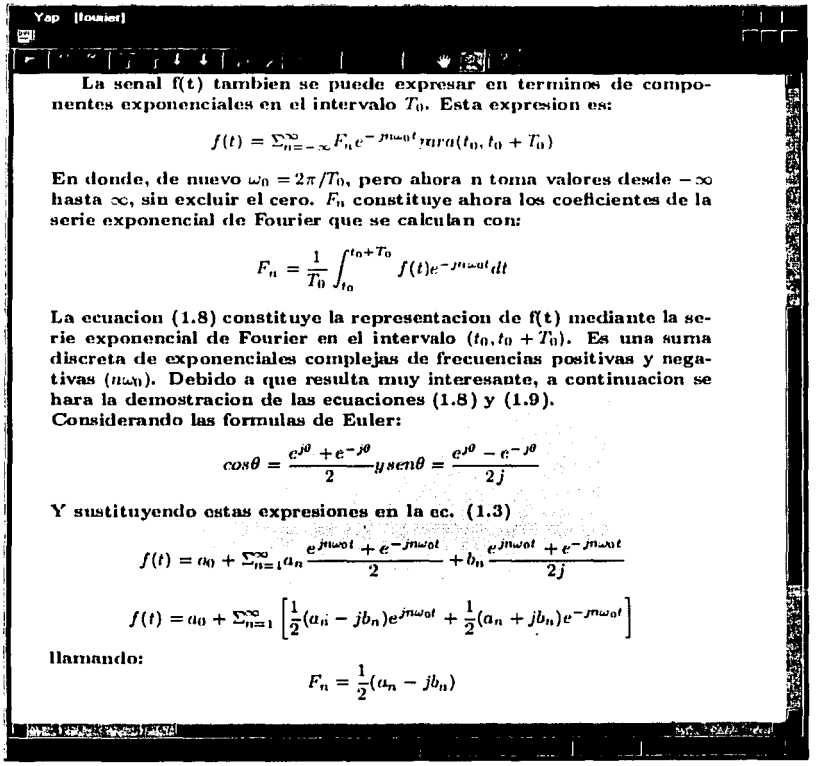


Figura 4.6 Visualización del documento DVI creado por LaTeX.

4.3.4 Ambientes.

Gran parte de la tarea del formateo de texto se realiza a base de declaraciones que definen **ambientes**. En un archivo de entrada, los ambientes pueden identificarse con los comandos **`\begin{}`** y **`\end{}`**, los cuales definen ambientes locales. Las declaraciones que definen ambientes pueden dividirse en:

1. Declaraciones que definen el **ambiente global** del documento.
2. Declaraciones que crean **ambientes locales.**
3. Declaraciones que modifican el **ambiente actual.**
4. Declaraciones que modifican el **estilo del texto.**

Ambientes locales.

La tarea de formateo hecho con LaTeX se realiza a base de comandos dentro de ambientes locales. De la misma manera que `\begin{document}` y `\end{document}` definen el cuerpo del documento, `\begin{}` y `\end{}` definen ambientes locales. Los ambientes locales se pueden dividir en diferentes categorías:

1. Ambientes de formateo de texto.

- **center.** Alineación centrada para un segmento de texto.
- **flushleft.** Alineación hacia la izquierda de un área de texto.
- **flushright.** Alineación hacia la derecha de un área de texto.
- **quote.** Marca un área que será indentada con respecto a todo el texto.
- **verse.** Define una alineación para poemas.

2. Ambientes de tabla

- **tabbing.** Define un área para poner en una tabla.
- **tabular.** Define un área para tabla estructurada.

4. Ambientes matemáticos.

- **array.** Ambiente para escribir arreglos y matrices.
- **math, displaymath y equation.** Para escribir ecuaciones matemáticas.

4.3.5 Estilos de documentos.

Al empezar a elaborar cualquier documento, se le debe indicar a LaTeX que tipo de documento se desea crear. Ya que dependiendo del documento, el programa determinará las características necesarias para poder generar la representación final de ese documento.

El comando **\documentstyle** debe de ser el primer comando en aparecer en cualquier documento. Este comando hace mucho más que marcar el inicio del documento, ya que indica el tamaño por default de la fuente, la medida de los margenes, la numeración de las páginas y que el documento final se formateará para ser impreso por un solo lado de la hoja. Estos son valores por omisión y son los mismos para todos los estilos de documentos.

El argumento estilo describe el formato total del documento y puede tomar uno de los siguientes valores:

- **article**, para publicación de documentos en revistas o periódicos.
- **report**, para elaborar reportes técnicos.
- **letter**, para la creación de cartas, memorandums o documentos oficiales.
- **book**, para la elaboración de documentos extensos tales como libros.
- **slide**, para crear diapositivas para presentaciones.
- **curriculum**, para la elaboración de curriculums vitae.
- **music**, Estilo que permite escribir caracteres musicales.

El comando **\documentstyle** también acepta varios argumentos opcionales y si se desea usar más de un argumento, sólo hay que separarlos por comas:

11pt y 12pt - determina el tamaño de la fuente
twoside - prepara el documento para impresión por ambos lados de una hoja.
twocolumn - produce un documento formateado a dos columnas.

4.3.6 Tipos de fuentesy tamaños.

Se pueden cambiar los valores para las caracteristicas de las fuentes que el comando **\documentstyle** posee por omisión. Por ejemplo, si se quisiera usar una fuente de 12 puntos en lugar de 10 y un espaciado doble entre lineas en vez de uno solo, se escribiria un comando como el siguiente:

\documentstyle[12pt]{article}.

Hay muchos tipos diferentes de fuentes y tamaños disponibles para su uso. Las declaraciones que se muestran en la *figura 4.7* afectan el tamaño de la fuente para el texto que se encuentre entre las llaves.

{\tiny }	tiny
{\scriptsize }	scriptsize
{\footnotesize }	footnotesize
{\small }	small
{\normalsize }	normalsize
{\large }	large
{\Large }	Large
{\huge }	huge
{\Huge }	Huge

Figura 4.7 Declaraciones que modifican el tamaño de las fuentes

El tamaño definido es 10-puntos y corresponde al de \normalsize, pero puede variar para algunos estilos de documentos o para sus opciones. El nuevo tamaño toma efecto inmediatamente despues de que aparece el comando de tamaño.

Las declaraciones mostradas en la *tabla 4.1* son usadas para cambiar el estilo de fuente.

Declaración	Efecto	Declaración	Efecto
\rm	Roman	\it	*Italicas*
\em	*Enfásis*	\bf	**Boldface**
\sl	Slanted	\sf	Sans serif
\sc	Small caps	\tt	`Typewriter`

Tabla 4.1 Declaraciones que modifican el estilo de fuente.

4.3.7 Texto matemático

Una de las características más poderosas de LaTeX es su capacidad para manejar simbología matemática compleja. A continuación se explica como se puede generar este tipo de simbología.

4.3.7.1 Fuentes y símbolos matemáticos.

LaTeX provee una gran cantidad de símbolos matemáticos. Algunos tan comunes como +, -, =, < o > , pueden ser obtenidos directamente del teclado, otros, más elaborados, son generados por medio de comandos en modo matemático.

Las letras del alfabeto griego también pueden ser obtenidas mediante comandos que llevan el nombre de la letra que se quiere generar, por ejemplo:

`\alpha`	(α)	`\beta`	(β)
`\pi`	(π)	`\omega`	(ω)
`\gamma`	(γ)	`\delta`	(δ)

También existen comandos para desplegar apropiadamente funciones trigonométricas ,logarítmicas o exponenciales, limites, determinantes, integrales y sumatorias, las mas importantes se muestran en la *tabla 4.2*

\arccos	\arcsin	\arctan	\cos
\cosh	\cot	\coth	\csc
\det	\exp	\inf	\lim
\liminf	\limsup	\ln	\log
\max	\min	\sec	\sin
\sinh	\sup	\tan	\tanh
\int	\sum	\prod	\coprod

Tabla 4.2 Comandos comunes para desplegar simbología matemática

4.3.7.2 Ambientes matemáticos.

LaTeX proporciona tres **ambientes** para incluir una ecuación matemática en un documento, estos ambientes son :

Ambiente math. Este ambiente permite colocar expresiones como a+b+g=180° dentro de una línea de texto (tal y cómo en esta línea). La expresión debe estar encerrada entre un par de símbolos $, pero también pueden ser usados los delimitadores \(y \) o los comandos **\begin{math}** y **\end{math}.** Por ejemplo:

```
$ Ax^{2} + Bx + C = 0 $
```
genera $Ax^2 + Bx + C = 0$

Ambiente displaymath. Permite colocar la expresión con una alineción centrada con respecto del texto (ver *figura 4.8*)

Para usar este ambiente hay que incluir los comandos \[para inciar y \] para terminar, también se pueden usar un par de $$ o usar los comandos **\begin{displaymath} y \end{displaymath}.**

```
\begin{displaymath}
\alpha+\beta + \gamma =180°
\end{displaymath}
```

$$\alpha+\beta+\gamma=180°$$

Figura 4.8 Ecuación generada con el ambiente displaymath

Ambiente equation. Es idéntico al ambiente displaymath, con la única diferencia de que este ambiente agrega automáticamente en el margen derecho un número consecutivo de ecuación (ver *figura 4.9*), el cual es incrementado automáticamente en cada capítulo o artículo. Para hacer uso de este ambiente sólo hay que incluir los **comandos `\begin{equation}` y `\end{equation}`**, y entre ellos la expresión matemática a generar.

```
\begin{equation}
\alpha + \beta + \gamma = 180°
\end{equation}
```

$$\alpha+\beta+\gamma=180° \qquad (4.1)$$

*Figura 4.9 Ecuación generada con el **ambiente equation***

4.3.7.3 Expresiones simples y ecuaciones.

La creación de expresiones matemáticas implica el uso de subíndices, superíndices, fracciones, raíces, alfabeto griego y símbolos matemáticos (sumatorias, integrales, etc.). Aquí se muestra como generar algunos de estos componentes en LaTeX.

Subíndices y superíndices. Estos son creados con los caracteres _ (subíndice) y ^ (superíndice). Los cuales pueden ser usados solos o en combinación, como en los ejemplos de la *tabla 4.3*.

Comando LaTeX	Resultado	Comando LaTeX	Resultado
`$ x_2 $`	x_2	`$ x_{2}y_{3}`	x_2y_3
`$ x_{3b}`	x_{3b}	`$ x^3 $`	x^3
`$ x^{3}y^{2}`	x^3y^2	`$ 5^x $`	5^x

Tabla 4.3 Ejemplos de índices y superíndices

Fracciones. Las fracciones se generan utilizando el comando `\frac`, el cual tiene la siguiente sintaxis:

```
\frac{num}{den}
```

Donde los argumentos **num** y **den** identifican al numerador y al denominador respectivamente.

Raíces. Las raíces se generan usando el comando `\sqrt`.

Sumatorias. Para las sumatorias se usa el comando `\sum`. Se usan los comandos de subíndices y superíndices para mostrar los límites de las sumatoria:

```
\sum_{n=0}^{\infty} x_{n}
```

4.3.8 Arreglos.

El ambiente **array** permite generar arreglos de caracteres los cuales son usados para crear estructuras tales como matrices o determinantes. Cada una de las columnas del arreglo puede ser alineada hacia la izquierda, centrada o hacia la derecha (l, c o r). Cada columna es separada por el comando \\, excepto por la última columna en la cual se omite. Las columnas son separadas por el símbolo & (ver *figura 4.10*).

Comandos LaTeX	Resultado

`\[ \begin{array}{ccc}`
`a & 14 & c \\`
`d - 3 & e & f \\`
`g & h & \lambda`
`\end{array} \]`
	a 14 c
	d - 3 e f
	g h λ

Figura 4.10 Un arreglo de elementos

He aquí otro ejemplo más complejo utilizando y combinando varios de los comandos de LaTeX:

$$det \begin{vmatrix} x_0 & x_1 & x_2 & \dots & x_n \\ x_1 & x_2 & x_3 & \dots & x_n \\ x_2 & x_3 & x_4 & \dots & x_n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & x_{n+1} & x_{n+2} & \dots & x_{2n} \end{vmatrix} > 0$$

Lo cual quedaría codificado como:

```
\[  \det  \left |
```

```
\begin{array}{lllll}
x_0  & x_1  &  x_2  & \cdots  &  x_n    \\
x_1  & x_2  &  x_3  & \cdots  &  x_{n+1}  \\
x_2  & x_3  &  x_4  & \cdots  &  x_{n+2}  \\
x_0  & x_1  &  x_2  & \cdots  &  x_n    \\
\vdots  & \vdots  &  \vdots  &  \vdots  &  \vdots  \\
x_n  & x_{n+1}  &  x_{n+2}  & \cdots  &  x_{2n}
\end{array}
\right |  > 0  \]
```

En este capítulo hemos mostrado como con LaTeX se pueden elaborar fácilmente documentos que contengan simbología matemática, no importando el nivel o la cantidad de ésta.

Es así, que uniendo los conceptos vistos este capítulo y en el capítulo anterior (Mecanismos de entrega Multipunto), logramos concretar otro paso para lograr llegar a nuestro objetivo, es decir ya tenemos las bases necesarias para poder implementar un "sistema de distribución de documentos LaTeX en Internet usando un mecanismo de único-envío multiple recpción".

El siguiente capítulo constituye la parte medular de esta tesis, ya que tratará de la implementación propia de nuestra aplicación, la cual se llevará a cabo con Java (ver 2.4 El lenguaje de programación Java).

Capítulo 5
Propuesta de Desarrollo

Uno de los conceptos que están adquiriendo actualmente gran auge gracias a Internet es el de **educación a distancia**. Algunas universidades alrededor del mundo han empezado a realizar proyectos experimentales y algunas otras ya proporcionan algún servicio más sofisticado.

Debido principalmente a que es un campo relativamente nuevo en Internet, uno de los problemas a los que se enfrenta la educación a distancia es el de hacer llegar la misma información a un grupo de alumnos que puede estar disperso en distintas localidades, ya sea dentro del mismo campus universitario o fuera de él (tal vez en cualquier parte del mundo).

Hasta este momento se han discutido algunos conceptos importantes que están íntimamente involucrados en el desarrollo del proyecto, tales como la programación orientada a objetos, el lenguaje de programación Java, el modelo de comunicación cliente/servidor, modelos de comunicación multipunto. Por lo tanto, es momento de relacionarlos y ver cual será el desarrollo de nuestra propuesta.

5.1 Visualización y distribución de documentos matemáticos en Internet.

Mostrar documentos matemáticos en Internet no es precisamente una tarea sencilla, ya que no existe un método estándar para representar notaciones matemáticas en el Web. Hay muchas maneras de hacerlo y cada una de ellas tiene ventajas y desventajas sobre las otras. A continuación veremos algunas de las más comunes en la actualidad.

5.1.1 Texto ASCII

ASCII son las siglas de American Standard Code for Information Interchange. El conjunto de caracteres que agrupa este código es reconocido actualmente por prácticamente cualquier computadora.

Escribir expresiones matemáticas con ASCII es conveniente si se usa el correo electrónico, o un formulario Web, o en cualquier situación en que sea necesario y obligado usar caracteres estándar. Esta notación también es de gran ayuda si sólo se desea usar código HTML.

Escribiendo fracciones con ASCII.

Supongamos que se deseamos escribir en ASCII la expresión "*uno sobre dos x*" las posibles maneras de escribirlo en ASCII serían :

a)
```
 1
---
2x
```

b) `1/(2x)`

Ambas opciones son correctas, sin embargo si se desea poner la expresión en una sola linea es más conveniente usar la opción **b)**. Si se usan caracteres ASCII para escribir fracciones hay que tener cuidado de incluir paréntesis en los lugares adecuados; por ejemplo la expresión **`1/2x`** puede presentar confusiones al momento de interpretarla, ya que puede ser tomado como: "***uno sobre 2x***" o como "***un medio de x***",

lo que, obviamente, no es lo mismo. Los corchetes también pueden ser utilizados de la misma forma que los paréntesis.

Escribiendo exponentes.

¿Cómo se puede representar en ASCII la expresión: ***x al cuadrado más x elevado a la quinta potencia***? He aquí dos maneras de hacerlo:

a)
```
   2    5
  x + x
```

b)
```
x^2 + x^(5y)
```

En este caso, la opción **b)** es la más adecuada ya que la opción **a)** puede presentar confusiones al momento de leerla: no se sabe si el 2 y el 5 son parte de la ecuación o son numeros independientes. En la opción **b)** nuevamente es necesario indicar con paréntesis cuales términos deben de ser considerados para la exponenciación.

Escribiendo raíces

No hay una manera sencilla de escribir el símbolo de la raíz cuadrada en ASCII, por lo que la manera más sencilla y común de escribir una raíz con caracteres ASCII es escribiendo: Sqrt(x).

También es posible expresar las raíces como potencias. Por lo tanto la raíz cúbica de siete se escribiría como: `7^(1/3)`

5.1.2 Trabajando con ASCII y HTML.

Consideremos la siguiente ecuación:

$$x = \frac{-b \pm Sqrt(b2 - 4ac)}{2a}$$

Esta ecuación puede ser generada fácilmente con código HTML como el siguiente:

```
<b>
<pre>
        -b &#177; Sqrt(b<sup>2</sup> - 4ac)
  x = ------------------
                2a
</pre>
</b>
```

Ventajas: este tipo de notación es muy apropiada si sólo se desea mostrar expresiones simples por lo que sólo se necesita saber un poco del lenguaje HTML estándar para poder elaborar las ecuaciones, además de que el ancho de banda utilizado en la transmisión de los documentos es considerablemente bajo, ya que únicamente se envían caracteres a través de la red.

Desventajas: con este método no se pueden mostrar símbolos tales como raíces, sumatorias, integrales, matrices y cualquier otro tipo de simbología matemática compleja.

5.1.3 Trabajando con imagenes y HTML.

Otro método sencillo de incluir simbología matemática en una página Web es insertar un dibujo (imagen) de la expresión que se quiera mostrar. Este dibujo puede elaborarse en cualquier paquete gráfico que soporte los formatos GIF o JPG (fig 5.1)

HTML proporciona una instrucción muy sencilla para incluir una imagen dentro de un documento:

```
<image src=imagen>
```

Dónde *imagen* indica el archivo GIF o JPG que se desea insertar en la página.

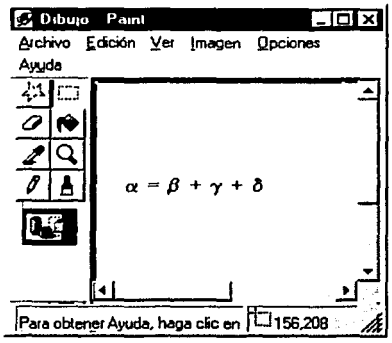


Figura 5.1 Editando una ecuación como una imagen.

Ventajas: Este es un método muy sencillo que requiere aun menos codificación HTML que su contraparte ASCII.

Desventajas: A mayor número de imagenes, mayor el ancho de banda necesario para transmitirlas y mayor tiempo de respuesta del servidor http. Además se tiene que utilizar un paquete gráfico adicional para elaborar las imagenes.

5.2 El programa IDVI para presentación de documentos LaTeX.

Ninguna de las alternativas anteriores satisface plenamente la necesidad que hemos planteado: la visualización de texto matemático en Internet. En el Capítulo 4 (El programa LaTeX) hemos visto que LaTeX es una muy buena alternativa para elaborar documentos con estas características. Sería muy bueno contar con un programa que nos permitiera leer un archivo creado con LaTeX y además de poder visualizarlo en cualquier navegador para Internet. Afortunadamente esta aplicación existe y su nombre es **IDVI**, es gratuito y además se puede obtener el código fuente para implementar las mejoras que uno considere pertinentes, y eso es precisamente lo que vamos a realizar.

IDVI es una aplicación escrita totalmente en Java, la cual fué escrita por Garth A. Dickie con el objetivo de mostrar documentos LaTeX en un navegador para Internet

que soporte Java. IDVI puede utilizarse para la visualización de documentos en dos formas: de manera local, es decir, en la misma máquina donde se encuentran los documentos y de manera remota, es decir, a través de la red.

El software puede obtenerse gratuitamente del sitio oficial de IDVI: `http://www.geom.umn.edu/java/idvi/` . Además de poder bajar los archivos binarios también es posible obtener el código fuente, sobre el cual se pueden realizar libremente las modificaciones que se deseen, siempre y cuando el programa resultante no sea usado con fines comerciales, en cuyo caso tiene que contactarse primero con el autor por cuestiones de derechos de propiedad intelectual.

Considerando lo anterior, y para evitar cualquier problema al modificar el código fuente, contacté directamente a Garth A. Dickie vía correo electrónico para informarle acerca del proyecto y preguntarle si podía hacer las modificaciones necesarias al programa para que funcionara en un modelo multicast, y esto fué lo que amablemente me contestó:

Date: Fri, 18 Dec 1998 09:10:40 -0500
To: Eduardo Martinez <mcjeduardo@yahoo.com>
From: "Garth A. Dickie" <dickie@ra.avid.com> Add to Address Book
Re: Question
Hi,
You can go ahead and make any changes you want to.
IDVI is a display program. It runs on the client, but opens dvi and font files which happen to be on the server. If requests font files as it discovers that it needs them.
You might have to send the font files first in a broadcast model, then the dvi file. This would involve changes to the DVIfont class, to get it to parse font files from an already-open stream.
If you can find a way to make it useful, more power to you!

Best Regards,
Garth

5.2.1 Instalación de IDVI

IDVI fué desarrollado en un ambiente UNIX, por lo que su autor recomienda ampliamente su instalación en uno de estos sistemas, debido principalmente a que los scripts de instalación están programados en `bourne shell (sh)`. Por esta causa, el software fué instalado en una PC con sistema operativo Linux. La instalación del software es muy sencilla, sólo hay que seguir los siguientes pasos para lograr una instalación éxitosa:

1. ***Editar el archivo makefile***

Es necesario modificar las variables de instalación para que reflejen las características del equipo en el cual se va a llevar a cabo la instalación:

FONTSOURCE. Esta variable indica la ruta del directorio donde se encuentran localizados los archivos de fuentes.

```
FONTSOURCE=/var/lib/texmf/fonts/pk/ljfour/public/cm
```

DPI. Esta variable es usada para indicar los puntos por pulgada (Dots Per Inch) con los que los caracteres serán desplegados.

```
DPI=300
```

IDVILIB. Los archivos de utilierías, así como los archivos de clases son guardados en este directorio.

```
IDVILIB=/usr/local/lib/idvi
```

IDVIBIN. Los archivos binarios se localizan en este directorio.

```
IDVIBIN=/usr/local/bin
```

IDVIWWW . Esta variable indica el URL que debe ser utilizado para accesar los archivos de clases a través de Internet.

```
IDVIWWW=http://arenque.dgsca.unam.mx/lib/idvi
```

IDVIWWWDIR. Esta es la ruta del directorio local correspondiente al URL indicado en la variable `IDVIWWW`.

```
IDVIWWWDIR=/home/httpd/html/lib/idvi
```

NETSCAPE. Indica el programa a utilizar para cargar el applet y visualizar los documentos.

```
NETSCAPE=netscape
```

2. ***Generar los scripts de instalación.***

Los scripts de instalación idvi y copyfont se generan al teclear el comando make, estos scripts son generados de acuerdo a las definiciones que contengan las variables de instalación del paso 1.

3. ***Instalar el software.***

Ejecutar el comando `make install`, para copiar los scripts `idvi` e `idvi.local` al directorio `IDVIBIN`; para copiar los archivos `jdclasses.zip`, `idviclasses.zip`, `images.prefix`, `fonts.prefix`, `copyfont`, `template.html` y `psheader.ps` al directorio `IDVILIB` y para crear un directorio en `IDVILIB` en el cual habrá ligas hacia los archivos de fuentes instalados en `FONTSOURCE`. Estos pasos pueden ser ejecutados uno a la vez usando los siguientes comandos: `make installbin`, `make installlib` `make installfonts`.

Ejecutar el comando `make installwww` para copiar las clases contenidas en el archivo `idviclasses.zip` al directorio `IDVIWWWDIR/classes`, y para crear un directorio de fuentes que el servidor web sea capaz de accesar.

4. ***Asegurar que otros usuarios pueden usar el software.***

Si el directorio `IDVIBIN` no se encuentra definido en la variable PATH de los usuarios, entonces hay que indicarles cual es esta ruta, o bien crear ligas simbólicas a los comandos `idvi` e `idvi.local` hacia el directorio `/usr/local/bin`, el cual es la ruta donde generalmente todos los usuarios tienen acceso a los archivos binarios. La variable de ambiente `CLASSPATH` es una variable muy importante, debido a que esta

variable indica la(s) ruta(s) dónde el compilador de Java o Netscape pueden encontrar los archivos .class (archivos de clases o byte code)

Esto se hace añadiendo la ruta `IDVILIB/idviclasses.zip` a esta variable, lo cual se puede hacer añadiendo la siguiente línea al archivo de inicialización que es ejecutado cada vez que se entra al sistema:

```
CLASSPATH=/usr/local/lib/idvi/idviclasses.zip
```

O bien si la variable ya esta definida, pero aún no tiene definida la ruta de clases de IDVI se pude usar la siguiente línea:

```
CLASSPATH=/usr/local/lib/idvi/idviclasses.zip;$CLASSPATH
```

5.2.2 Preparando un documento para vista con IDVI.

Antes de querer utilizar a IDVI para mostrar documentos LaTeX, es necesario realizar algunos pasos de prepararación para IDVI reconozca cuantas páginas tiene el documento, los niveles de acercamiento (zoom) que se pueden aplicar a ese documento y los archivos HTML para su publicación en el Web.

Para cada uno de los documentos que se quieran mostrar deben seguirse los siguientes pasos:

1. Crear un documento LaTeX y compilarlo para generar el archivo dvi correspondiente:

```
latex documento.tex
```

2. Crear un nuevo directorio en el cual se guardaran los archivos html necesarios para el despliege del documento en el web:

```
mkdir html
cd html
```

3. Ejecutar el comando idvi sobre el archivo dvi creado en el paso 1 para crear los archivos html en el directorio actual:

```
idvi ../documento.dvi -title "Un ejemplo"
```

El comando idvi determinará la cantidad de páginas que componen al archivo dvi y creará un archivo HTML por cada una de las páginas del documento, es decir, si el archivo contiene 3 páginas se crearán 3 archivos HTML. Además se crearán otros archivos HTML para las diferentes escalas de acercamiento (zoom) que se puede hacer sobre el documento, por default se generan cuatro por cada página del documento. Esto nos da un total de 12 archivos HTML para un documento de 3 páginas. Los nombres de estos archivos siguen el siguiente formato:

page*n*scale*x*.html

Dónde ***n*** indica la página del documento y ***x*** la escala a la que corresponde ese archivo. Para nuestro ejemplo de un documento de 3 páginas se generarían los siguientes 12 archivos HTML:

page1scale1.html page1scale2.html page1scale3.html page1scale4.html
page2scale1.html page2scale2.html page2scale3.html page2scale4.html
page3scale1.html page3scale2.html page3scale3.html page3scale4.html

Además, los archivos HTML generados por IDVI sirven para especificar:

- El título del documento, el cual aparece en la parte superior del web browser.
- La posición del applet en la pantalla.
- El lugar donde se encuentran los archivos de fuentes.
- La escala que se debe usar.
- El archivo dvi que debe ser desplegado.

De esta manera, un archivo llamado page1scale3.html podría lucir de la siguiente manera:

```
<html>
  <head>
    <title>Curso de Integracion </title>
  </head>
  <body bgcolor=#FFFFFF>
  <applet codebase=../../idviclasses/
              code=ibook.release.DVIOnePageApplet
              width=50 height=700>
        <param name=fontBase value=../../pkfonts/>
        <param name=pageMax  value=30>
   </applet>
   </body>
</html>
```

4. Finalmente hay que verificar la apariencia del documento usando un web browser que soporte Java , por ejemplo Netscape o Internet Explorer (ver *figura 5.2*)

5.2.3 Módulos que componen el código fuente de IDVI.

Como ya se mencionó anteriormente, el código fuente de IDVI puede ser obtenido y ser modificado libremente. Parte de este código será modificado para poder hacer de IDVI una aplicación multicast.

IDVI es una aplicación muy completa la cual se compone de varios módulos, cada uno de los cuales realiza una tarea específica, esta organización permite al programador localizar fácilmente los módulos que necesita modificar.

La organización del código fuente dentro de la estructura de directorios del sistema operativo es la siguiente:

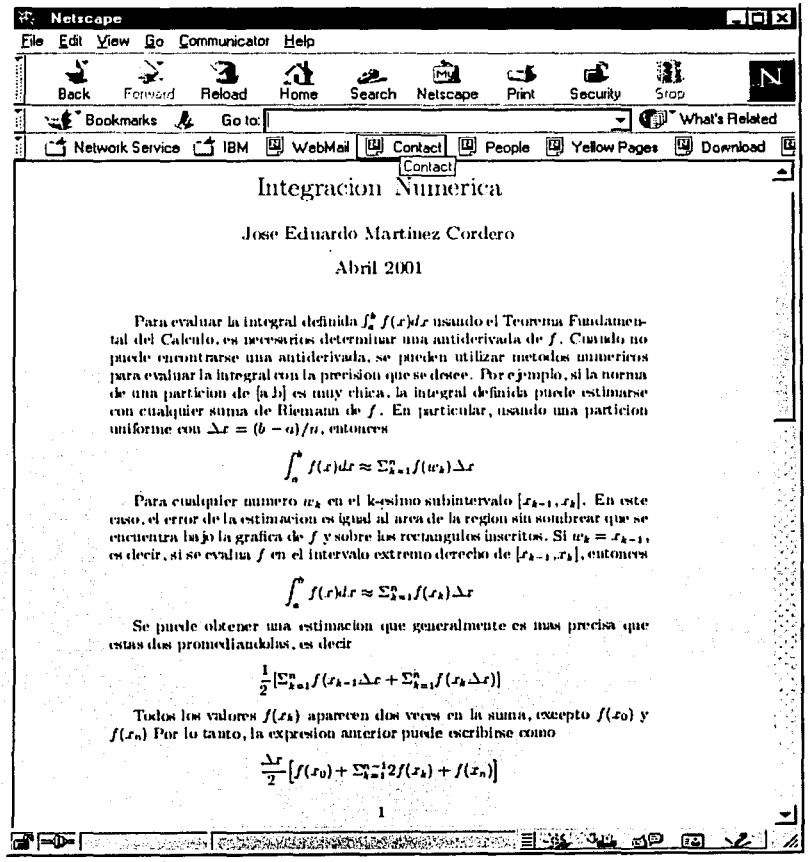


Figura 5.2. Visualización del documento usando IDVT como applet dentro de Netscape

```
ibook/v10
ibook/v10/applet
ibook/v10/applets
ibook/v10/applets/colorchange
ibook/v10/applets/hithere
ibook/v10/applets/text
ibook/v10/awt
ibook/v10/colors
ibook/v10/html
ibook/v10/idvi
ibook/v10/idvi/controls
ibook/v10/idvi/display
ibook/v10/idvi/dvi
ibook/v10/idvi/font
ibook/v10/idvi/io
ibook/v10/idvi/split
ibook/v10/parameter
ibook/v10/util
```

Los directorios **ibook/v10/idvi** e **ibook/v10/idvi/dvi** contienen el código fuente de las clases que se modificarán para hacer nuestra implementación.

En el directorio **ibook/v10/idvi** es donde se encuentran las principales clases y el applet usado para desplegar los documentos. Entre las clases que se encuentran en este directorio podemos mencionar las siguientes:

IDVI. Aquí se definen las variables globales y constantes, tales como los valores por omisión y los mensajes que aparecen en el applet. En esta clase se realizarán cambios menores, principalmente de traducción de mensajes del inglés al español.

IDVIContext. Interface implementada por el applet, la cual proporciona información global, tal como el URL del código, el URL del archivo dvi, acceso a los parametros del applet y un registro de los applets hijos. No sufrirá cambios.

MessageContext. Esta clase proporciona un mecanismo para desplegar un mensaje. En IDVI los mensajes aparecen en la parte baja de la ventana de controles. No sufrirá cambios.

ShowDocumentContext. Esta clase es la que permite cambiar de una página a otra. Interpreta comandos como "idvi:pageoffset:+1" (pasar a una página siguiente de la actual), o cambiar URLs como "pagina3.dvi" a "pagina3scale4.html". No sufrirá cambios.

DVIOnePageApplet. Esta es la principal clase y por lo tanto la más importante (pero también la más complicada) de todo el sistema. Implementa las todas las clases anteriores. Muestra la ventana de controles, carga el documento haciendo una llamada a la clase DVIDocument y lo despliega usando la clase *ViewPanel*. Entre las tareas más importantes que ejecuta se encuentran las siguientes:

1. Visualiza una página de un archivo DVI.
2. Carga, descomprime y escala los archivos de fuentes.
3. Proporciona controles de navegación.
4. Maneja multiples vistas de un mismo archivo, incluyendo las escalas de acercamientos (zoom).

Esta clase no sufrirá cambios, pero se crearán dos nuevas clases tomando como base ésta: `DVIOnePageReciver` y `DVIOnePageServer`, las cuales serán respectivamente las clases principales para el cliente y el servidor de nuestra implementación.

En el directorio **`ibook/v10/idvi/dvi`** se encuentran otras clases de vital importancia para el proyecto, de entre las cuales es necesario mencionar a las siguientes:

DVIDocument. Esta clase carga un documento desde un URL. El proceso se realiza asíncronamente. Carga los archivos de fuentes de acuerdo a las definiciones en el encabezado del archivo dvi.

DVIStreamDocument. Esta clase permite leer el documento y verificar sus características. Esta clase será modificada para controloar el envío o recepción de documentos hacia un grupo multicast.

5.3 Programación multitarea.

En esta parte veremos como con Java se pueden programar aplicaciones multitarea usando un concepto sencillo pero poderoso: los **threads.** Es importante tocar este tema ya que IDVI usa extensivamente threads de control para sincronizar la carga y visualización de un documento y nuestra implementación debe ser capaz de controlar esos threads, además de implementar los suyos propios.

5.3.1 Threads.

Dentro de las clases estándar de Java, se encuentran dos clases que pueden considerarse como la médula espinal de la programación multitarea en Java: java.lang.Thread y java.lang.Runnable. Estas clases permiten programar threads de control en las aplicaciones y manejar los threads en términos de recursos de máquina y del estado de ejecución del thread.

Un thread tiene las siguientes características:

Cuerpo del Thread. Esta constiuido de las instrucciones que el thread ejecutará, las cuales se definen dentro del cuerpo del método `run()`. Hay dos formas de codificar un método `run()`:

- Extendiendo a la clase Thread y sobreescribiendo el método `run()`
- Creando un nuevo Thread pasandole como argumento una clase del tipo Runnable. Una clase puede implementar la interface Runnable y entonces ser pasada como argumento al constructor del Thread.

Estado del Thread. Un thread puede encontrarse en alguno de los cuatro estados siguientes: `new`, `runnable`, `nonrunnable` o `dead`. Los métodos del thread y la máquina virtual Java controlan el estado de los threads. El estado de un

thread (ver *figura 5.3*) determina también que métodos pueden ejecutarse en un estado determinado, por ejemplo, no puede detenerse (stop) un thread si éste no se esta ejecutando (running).

1. **Nuevo thread.** En el momento en que se invoca al constructor se crea un nuevo thread por lo que se encuentra en el estado new, y aún no se esta ejecutando. Los únicos métodos que pueden invocarse en este estado son `start()` y `stop()`.
2. **`start()`.** Este método cambia el estado del thread a runnable , por lo que la máquina virtual hará una llamada al método `run()` para ejecutar el thread.
3. **`stop()`.** Este método termina la ejecución de un thread.
4. **`destroy()`.** Detiene un thread sin importar el estado en que se encuentre.
5. **`suspend()`.** Este método causa que un thread pase al estado nonrunnable . El thread permanecerá en ese estado hasta que se haga una llamada al método `resume()`.
6. **`resume()`.** Al invocar a éste método el thread vuelve a pasar al estado `runnable`.
7. **`sleep(long millis)`** . Este método causa que un thread pase al estado nonrunnable por la cantidad especificada de milisegundos.
8. **`wait()`.** Este es un método de la clase Object. Permite que un thread espere a que un objeto sea dejado de utilizar para poder obtener el control sobre ese objeto.
9. **`yield()`.** Este método sólo puede ser invocado cuando el thread esta en ejecución (`runnable`) . No cambia el estado del thread, pero permite que se ejecuten otros threads de igual prioridad.

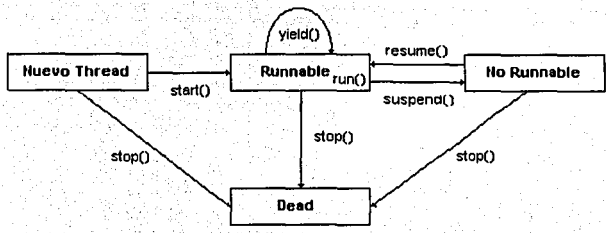


Figura 5.3 Diagrama del estado de un thread.

Prioridad del Thread. Cada thread tiene una prioridad. Cuando los threads son creados se les asigna la misma prioridad del thread que los creó. La prioridad de un thread se especifica con un numero entero, el cual puede variar de 1 a 10 (`Thread.MIN_PRIORITY` a `Thread.MAX_PRIORITY`) , y determina como será atendido. En cualquier momento, el thread que cuente con una mayor prioridad será atendido antes que uno de menor prioridad.

Si dos o más threads tienen la misma prioridad, la máquina virtual los atiende siguiendo una política *round-robin*, es decir se asignan tiempos iguales de ejecución para cada uno de los threads hasta que todos terminan.

5.3.2 Sincronización de Threads.

La sincronización es una parte critica en la mayoría de los programas multitarea, debido a que en estos programas intervienen múltiples threads, cada uno de los cuales trabajan con datos y recursos compartidos. El acceso a estos recursos y datos compartidos debe ser controlado para garantizar la correcta operación del programa.

5.3.2.1 Monitores.

Para controlar el acceso a ciertas secciones críticas en el código de un progrma, en Java se utiliza un mecanismo denominado **monitor.** Un monitor es básicamente un **candado** que es usado para proteger ciertas secciones de código. Cuando un thread

ejecuta una sección crítica, éste adquiere el candado. Si otro thread trata de ejecutar la misma sección de código cuando otro thread tiene el candado, entonces es forzado a esperar hasta que el candado es liberado. Un thread adquiere el monitor al invocar a un método que ha sido definido como `synchronized`, obligando a cualquier otro thread que quiera ejecutar ese mismo método a esperar hasta que se libere el monitor. Un monitor es asociado por cada objeto que tenga uno o más métodos declarados como synchronized :

```
public synchronized void mi_método () {}
```

5.3.2.2 Comunicación entre threads.

El mecanismo de bloqueo (monitores) implementado con los métodos `synchronized` permite controlar a los threads, de tal manera que no interfieran unos con otros, sin embargo también es necesario contar con un mecanismo para la comunicación entre threads.

Para este fin, el método `wait()` obliga a un thread esperar a ejecutarse hasta que una condición específica se cumpla; y los métodos `notify()` y `notifyAll()` notifican a los threads que esten esperando que algo ha ocurrido, y que la condición por la que están esperando puede haberse cumplido.

Cuando se utlizen lo métodos de notificación y `wait()` es muy importante seguir el siguiente patrón de codificación; la sección de código que espera a que una condición se cumpla debe verse como:

```
synchronized void hazCuandoCondicion () {
   while ( !condicion )
      wait();
// Sección de código que se ejecuta cuando la condición
// es verdadera.
}
```

Analizemos porque debe codificarse de esta manera:

- Toda sección crítica es ejecutada dentro de la parte de código declarada como **`synchronized`**, de no declararse así el estado del objeto no sería estable. Por ejemplo, si el método no fuera **`synchronized`**, entonces, despues de la sentencia while , no habrá garantía de que la condición permanezca como verdadera (`true`) debido a que otro thread podría cambiar las condiciones para la cual la variable debe ser verdadera.

- Uno de los aspectos importantes de la definición de `wait()` es que cuando éste método obliga al thread a hacer una pausa, éste libera el candado que tiene sobre el objeto. Cuando un thread es reinicializado después de haber sido notificado, éste vuelve a adquirir el candado.

- La variable de condición ***siempre*** debe ser evaluada dentro de un ciclo. Nunca debe evaluarse dentro de una sentencia condicional `if`.

Por otro lado, los métodos de notificación son invocados por métodos `synchronized` que cambian una o más condiciones, las cuales son esperadas por uno o más threads. Estos métodos generalmente se codifican como:

```
synchronized void cambiarCondicion () {
   // Cambiar algunos valores usados en el método que
   // espera una condición.
   notifyAll();
}
```

Debido a que multiples threads pueden estar esperando por el mismo objeto. Al usar `notify()`, Java toma uno de los threads que estan esperando y lo "despierta". No se puede determinar cual thread será despertado, se debe usar siempre el método `notifyAll()` para despertar a todos los threads que están esperando usar el objeto, en vez de a uno solo.

5.4 Programando aplicaciones multicast con Java.

Java posee un paquete muy completo de clases con las cuales es posible programar aplicaciones multicast.

Las clases que implementan multicast en Java hacen uso del concepto de *socket*, el cual (como ya vimos) es un manipulador de comunicaciones que sirve como puente entre la red y una aplicación. Sólo que para el caso de comunicación multicast, el concepto de socket se amplia al de *socket multicast*. La clase MulticastSocket es usada para crear un socket multicast. Esta clase permite enviar paquetes IP multicast usando el protocolo UDP.

Para empezar a participar en la comunicación multicast es necesario crear un *socket multicast*. Para esto sólo basta con llamar al constructor por omisión de la clase:

```
public MulticastSocket() throws SocketException;
```

5.4.1 Transmisión de paquetes multicast.

Despues se debe crear un DatagramPacket, con la dirección de un grupo de multicast. Una vez creado el datagrama puede ser enviado usando el método send, el cual recibe como parámetro un valor para el camp TTL (Time To Live, *ver 3.2.5.1 Campo Time To Live (TTL) del datagrama IP)* y hay que evitar en lo posible utilizar un valor muy grande para este parámetro, para no causar mucho tráfico en la red.

Analizemos las siguientes líneas de código fuente:

```
1.- int multiPort = 2222;
2.- int ttl = 1;
3.- InetAddress multiAddr =
                    InetAddress.getByName("239.10.10.10");
4.- byte[] multiBytes = new byte[256];
5.- DatagramPacket multiDatagram = new
DatagramPacket(multiBytes,multiBytes.length,multiAddr,multiPort);
6.- MulticastSocket multiSocket = new MulticastSocket();
7.- multiSocket.send(multiDatagram, ttl);
```

Explicación.

1.- Se especifica el número de puerto UDP sobre el cual se llevará a cabo la comunicación multicast.

2.- Se le asigna el valor de 1 al campo TTL (Time To Live). Recordemos que el valor de 1 limita la difusión del paquete multicast a la red local.

3.- Especificamos la dirección IP multicast a la que la aplicación se va a unir.

4.- Aquí especificamos la cantidad máxima de bytes que podrán ser enviados en un paquete multicast: 256 en este caso.

5.- Creamos el paquete multicast, con las características asociadas (*socket*).

6.- Abrimos el socket (*connect*).

7.- Enviamos información, utilizando el socket (*write*).

5.4.2 Recepción de paquetes multicast.

Para recibir datagramas una aplicación debe crear un socket hacia un puerto UDP y darse de alta en el grupo multicast. A travez del socket la aplicación puede entonces recibir datagramas UDP. Analizemos el siguiente fragmento de código fuente:

```
1.- MulticastSocket receiveSocket = new
                                MulticastSocket(multiPort);
2.- receiveSocket.joinGroup(multiAddr);
3.- receiveSocket.receive(multiDatagram);
```

Explicación.

1.- Creamos el socket y lo asociamos a un número de puerto UDP (*socket*).

2.- La aplicación lanza una petición IGMP de pertenencia a grupo a través del método `joinGroup()`.

3.- Se recibe la información que llega en un paquete multicast (*read*)

Desde el momento en que se invoca al método `joinGroup()`, la aplicación puede empezar a recibir cualquier paquete IP multicast transmitido al grupo en el que se acaba de unir.

Para abandonar un grupo de multicast se usa el método `leaveGroup()`. El socket creado con MulticastSocket debe de ser cerrado una vez que la comunicación ha finalizado.

```
1.- receiveSocket.leaveGroup(multiAddr);
```

La aplicación manda un mensaje IGMP para abandonar el grupo multicast.

```
2.- receiveSocket.close();
```

Se cierra el socket, terminando definitivamente la comunicación.

5.5 Modificación a los módulos de IDVI para que soporten Multicast.

A continuación se explicarán las modificaciones hechas a los módulos de IDVI para permitir que esta aplicación se convirtiera en una aplicación multicast. Es en esta parte dónde se reúnen todas las piezas vistas en secciones o capítulos anteriores y se hace sentido del porque se mencionan a lo largo de este trabajo.

5.5.1 Módulos multicast.

La parte más importante de transmisión de datos esta implementada en dos clases: **`IDVIMulticastCliente`** e **`IDVIMulticastServidor`**, son éstas clases las que hacen todo el trabajo de transmisión y recepción de documentos. Estas clases son el corazón de la aplicación y en ellas se implementan la mayoría de los conceptos tratados en los capítulos previos, es por eso que analizará cada una de ellas para indicar detalladamente cual es su funcionamiento.

Para facilitar la lectura y análisis de las clases se numerará cada una de las líneas para posteriormente indicar que es lo que se esta efectuando en cada una de ellas.

5.5.1.1 Clase Java para el cliente Multicast: IDVIMulticastCliente

```
1.- import java.net.*;
2.- import java.lang.*;
3.- import java.io.*;
4.- import ibook.v10.idvi.*;
5.- public class IDVIMulticastCliente implements
                                                    Runnable {
6.- private static final int DATAGRAM_BYTES = 256;
7.- private int puertoMulticast;
8.- private InetAddress mcastIP;
9.- private MulticastSocket mcastSocket;
10.-private ByteArrayInputStream inputStream;
11.-private boolean keepReceiving = false;
12.-public  static boolean doneReceiving = false;
13.-public  static boolean gotInputStream = false;
14.-private MessageContext contexto;
15.- public IDVIMulticastCliente(String dirMulticast,
                    int mport, MessageContext context) {
16.- try {
17.-       mcastIP = InetAddress.getByName( dirMulticast );
18.-       puertoMulticast = mport;
19.-       mcastSocket= new MulticastSocket(puertoMulticast);
20.-       mcastSocket.joinGroup(mcastIP);
21.-       contexto = context;
22.- } catch (UnknownHostException excpt) {
23.-      System.err.println("Direccion desconocida:"+excpt);
24.-      System.exit(1);
25.- } catch (SocketException excpt) {
26.- System.err.println("No es posible obtener un
                                        socket" + excpt);
27.- } catch (IOException excpt) {
28.-      System.err.println("No es posible E/S " + excpt );
29.-      System.exit(1);
30.- }
31.- Thread cliente = new Thread(this);
32.- cliente.start();
33.- } // fin del constructor.
34.- public void run ( ) {
```

```
35.- DatagramPacket mcastPacket;
36.- byte [] Buffer;
37.- byte [] mcastBuffer;
38.- int i, offset = 0;
39.- String protocolo;
40.- Thread.currentThread().setPriority(5);
41.- try {
42.-  Buffer = new byte[3840];
43.-  for ( int j = 0; j < 12; j++ ) {
44.-        mcastBuffer = new byte[DATAGRAM_BYTES];
45.-        mcastPacket = new DatagramPacket(mcastBuffer,
46.-                                         mcastBuffer.length);
47.-      mcastSocket.receive(mcastPacket);
48.-   contexto.showMessage("Paquete #"+ j + "recibido");
49.-     for ( i = 0; i < DATAGRAM_BYTES; i++, offset++ )
50.-            Buffer[offset] = mcastBuffer[i];
51.-  }
52.-  inputStream = new ByteArrayInputStream( Buffer );
53.- }  // try
54.- catch (IOException excpt) {
55.-    System.err.println("Fallo la E/S: " + excpt );
56.- }
57.- try {
58.-  mcastSocket.leaveGroup(mcastIP);
59.- } catch (SocketException excpt) {
60.- System.err.println ("Socket problem leaving group: "
                                                  + excpt );
61.- } catch (IOException excpt) {
62.- System.err.println ("Problema con E/S : " + excpt );
63.- }
64.- mcastSocket.close();
65.- setDoneReceiving();
66.- } // fin run()
67.- private synchronized void setDoneReceiving () {
68.-   doneReceiving = true;
69.-   notifyAll();
70.- }
71.- public synchronized InputStream getStream() {
72.-  while ( !doneReceiving ) {
73.-     try {
74.-         wait();
```

```
75.-        } catch (InterruptedException e) { }
76.-    }
77.-    return inputStream;
78.- }
79.- }
80.- // fin de la clase IDVIMulticastCliente
```

Explicación

1-4 En estas primeras líneas se utiliza la sentencia `import` la cual es utilizada para indicarle al compilador el lugar en el que debe buscar clases a las que se hacen referencia dentro del código.

5 Este es el nombre de la clase: `IDVIMulticastCliente`, la cual es definida como pública (*ver 2.3.1.2 Clases*) e implementa la interface `Runnable`. Se ha implementado esta última interface debido a que deseamos que el código se comporte como un Thread (*ver 5.3.1 Threads*) y una de las maneras en que se puede hacer esto es que la clase `Thread` tome como argumento una clase que implemente la interface `Runnable`.

6 La variable indica la cantidad máxima de bytes de cada uno de los paquetes que se transmitirán. Es declarada como final, lo que indica que su valor es constante y no puede ser cambiado en ninguna parte del programa.

7 Esta variable indicará que puerto se utilizará (ver *2.1.2 El protocolo TCP*)

8 Aquí se indica la dirección IP del grupo multicast en el cual se distribuirán los documentos (ver *3.2.2 Multidifusión IP: direcciones de clase D*).

9 Las operaciones de E/S de red deben ser controladas por un socket (ver *2.1.3 La interfaz socket*) sobre el cual se realizará las operaciones de envio y recepción de paquetes (ver *2.1.3.3 Comunicación en red a través de sockets*).

10 La programacion de E/S de archivos en Java, esta basada en los **flujos** (*streams*). Las clases de flujos estan divididas en dos categorías: flujos de bytes (*byte stream*) y de caracteres (*character stream*). Este flujo será utilizado para ir recibiendo los paquetes enviados por el servidor.

12 La variable doneReceiving será utilizada como bandera para indicar a los demás threads que se ha terminado de recibir el documento, y de que el proceso normal de desplegado del documento puede continuar (ver *5.3.2.2 Comunicación entre threads*)

13 Es una variable booleana que indica si se pudo obtener el flujo de entrada, de lo contrario no es posible continuar con el programa.

14 Esta es una variable que indica el lugar donde se desplegarán los mensajes de status del programa.

15 Este es el constructor de la clase (*ver 2.3.1.2 Clases*), el cual debe cumplir con el requisito de tener el mismo nombre que la clase. Se reciben como parametros la direccion IP Multicast y el puerto que se utilizara para la comunicacion.

16-30. El bloque conocido como "`try-catch`" permite implementar el manejo de errores de una manera muy sencilla (ver *2.4 El lenguaje de programación Java*). La palabra clave "try" precede a un segmento de código que puede ocasionar algún error (o mejor dicho en términos de Java: una excepción), y "catch" permite ejecutar una acción determinada para un error específico. Las líneas 16 a 30 son una parte crítica del sistema por lo que es codificada en uno de estos bloques.

20 En esta línea se envia una petición IGMP (ver *3.2.6 Protocolo IGMP*) de pertenencia a grupo multicast (ver *figura 3.6 Funcionamiento del protocolo IGMP*) a través del método joinGroup, para poder de esta manera recibir la información que se genere en el grupo (ver *5.4.2 Recepción de paquetes multicast*).

22-30 Las líneas que implementan el manejo de errores en esta parte del código, tales como que la dirección multicast indicada sea una dirección desconocida o no válida; que no sea posible crear el socket, con lo cual no se podría establecer la comunicación entre el servidor y el cliente; o bien que no se pueda abrir el archivo LaTeX que se pretende enviar (ver el *Capítulo 4 El programa LaTeX*).

31 Debido a que la clase implementa la interface Runnable, ésta es pasada como argumento al constructor de la clase Thread, permitiendo de esta manera que ésta clase se convierta a si misma en un thread (ver *5.3.1 Threads*), el cual en este momento tiene el estado de *new* (ver *fig. 5.3 Diagrama del estado de un thread*).

32 El método `start()` del Thread cliente ocasiona que el método run() del thread se ejecute (ver *5.3.1 Threads*).

34 Este es el método `run()` que es ejecutado en el momento que se invoca el método `start()` y que permite que el thread entre en el estado de *runnable* (ver *figura 5.3 Diagrama del estado de un thread*).

35 Se crea una variable del tipo datagrama; recordemos que el protocolo transmite datos en forma de paquetes (ver *2.1.1 El protocolo IP*), por lo que esta variable permitira recibirlos y tratarlos localmente.

36-39 Aquí se declaran variables que permitiran que los datos que llegan de la red sean tratados en memoria de una manera más sencilla, y banderas de la estructura de control *for*.

40 Se establece la prioridad más alta para este Thread (ver *5.3.1.3 Prioridad del Thread*), de tal manera que cualquier otro thread que pudiera existir deberá esperar a que éste termine para poder ejecutarse.

41-56 Esta parte de código también es codificada en un bloque "try-catch" ya que cualquier error en la recepción de los paquetes es tratado aquí.

42 Se crea un buffer de memoria en el cual se irá guardando el documento LaTeX que será recibido por la red.

47 El socket multicast ahora recibe paquetes (hace más bien una operación de read, de acuerdo a lo indicado en *2.1.3.3 Comunicación en red a través de sockets*) de tamaño máximo establecido por la variable `DATAGRAM_BYTES`.

49-50 Se copian en el buffer de memoria todos los datos recibidos por la red

52 Se crea el flujo de entrada (pasando como argumento el buffer de memoria), el cual contiene ahora los datos obtenidos por los paquetes multicast recibidos. Y como ahora se tiene el flujo de entrada, no importa de dónde hayan llegado los datos puesto que ya se tienen localmente y se pueden procesar sin ningún problema. Una vez obtenido este flujo de entrada los demás módulos de IDVI se encargarán de interpretar los bytes que contiene y se desplegará el documento.

53-56 Una vez más se hace el tratamiento de errores necesario. En este caso se procesan los errores de E/S que se pudieran presentar en la recepción de los paquetes.

58 Se envía un mensaje de que esta computadora abandonará el grupo multicast (ver *5.4.2 Recepción de paquetes multicast*).

59-63 Aquí se procesan los errores que se pueden presentar al tratar de abandonar el grupo multicast o bien los errores de E/S.

64 Se cierra el socket con lo cual se da por concluida la comunicación vía red (ver *2.1.3.3 Comunicación en red a través de sockets*).

65 Se invoca al método `setDoneReceiving` el cual indica a los demás threads que el documento ha sido recibido éxitosamente y que pueden continuar su procesamiento normal de desplegado del documento (ver *5.3.2.2 Comunicación entre threads*). Estos, a su vez, utilizarán el flujo de entrada de la línea 52 para saber que es lo que desplegarán.

67-79 Los métodos `setDoneReceiving` y `getStream` son declarados como `synchronized`, lo cual permite convertirlos en **monitores** (ver *5.3.2.1 Monitores*). Un monitor es básicamente un **candado** que es usado para proteger ciertas secciones de código. Cuando un thread ejecuta una sección crítica, éste adquiere el candado. Si otro thread trata de ejecutar la misma sección de código cuando otro thread tiene el candado, entonces es forzado a esperar hasta que el candado es liberado. Como se puede ver, el método `getStream()` devolverá el flujo creado en la *línea 52* única y exclusivamente cuando éste reciba la notificación de que el documento fué recibido, indicación que se hace en la *línea 65*.

5.5.1.2 Clase Java para el servidor Multicast: IDVIMulticastServidor

```
1.- import java.net.*;
2.- import java.lang.*;
3.- import java.io.*;
4.- import ibook.v10.idvi.*;
5.- public class IDVIMulticastServidor implements Runnable {
6.- private static final byte   TTL = 1;
7.- private static final int    DATAGRAM_BYTES = 256;
8.- private int                 mcastPort;
9.- private InetAddress         mcastIP;
10.-private MulticastSocket     mcastSocket;
11.-private MessageContext      contexto;
12.-private InputStream         inputStream;
13.-public  static boolean      doneSending = false;

14.-public IDVIMulticastServidor(String mdir, int mport,
                    URL urlDocument,  MessageContext context) {
15.-    try {
16.-       mcastIP = InetAddress.getByName( mdir );
17.-       mcastPort = mport;
```

```
18.-      mcastSocket = new MulticastSocket(mcastPort);
19.-      inputStream = urlDocument.openStream();
20.-      contexto = context;
21.-  } catch (UnknownHostException excpt) {
22.-      System.err.println("Direccion multicast
                          desconocida: " + excpt);
23.-      System.exit(1);
24.-  } catch (SocketException excpt) {
25.-      System.err.println("No es posible obtener un
                           socket " + excpt);
26.-      System.exit(1);
27.-  } catch (IOException excpt) {
28.-      System.err.println("No es posible ejecutar
                                operaciones de E/S" + excpt );
29.-      System.exit(1);
30.-  }
31.-  Thread servidor = new Thread( this );
32.-  servidor.start();
33.-}
34.-public void run ( ) {
35.-DatagramPacket mcastPacket;
36.-int i, bytes;
37.-byte [] mcastBuffer;
38.-String protocolo;
39.-Thread.currentThread().setPriority ( 5 );
40.-try {
41.-   mcastBuffer = new byte[DATAGRAM_BYTES];
42.-   i = 1;
43.-   mcastBuffer = new byte[DATAGRAM_BYTES];
44.-   while ( (bytes=inputStream.read(mcastBuffer, 0,
45.-                                   DATAGRAM_BYTES)) != -1 ) {
46.-      contexto.showMessage("Enviando paquete # " + i++ );
47.-      mcastPacket = new DatagramPacket(mcastBuffer,
                          mcastBuffer.length, mcastIP,mcastPort);
48.-      mcastSocket.send (mcastPacket, TTL);
49.-      contexto.showMessage("Bytes transmitidos"+bytes);
50.-   } // while
51.-} // try
52.-catch (IOException excpt) {
53.-  System.err.println("Error al intentar mandar el
                                          documento\n");
```

```
54.-} // catch
55.-mcastSocket.close();
56.-setDoneSending();
57.-} // run
58.-private synchronized void setDoneSending () {
59.-  doneSending = true;
60.-  notifyAll();
61.-}
62.-} // fin de la clase IDVIMulticastServidor.
```

Explicación

1-62 Puede observarse que el código del servidor es muy parecido al del cliente, por lo que no se analizarán todas las líneas, sino únicamente en aquellas en las que exista una significante diferencia.

6 Se declara una constante llamada TTL (*ver 3.2.5.1 Campo Time To Live (TTL) del datagrama IP*), este campo especifica la duración, en segundos, del tiempo que un paquete multicast tiene permitido permanecer en la red.

7 La constante DATAGRAM_BYTES indica la cantidad máxima de bytes de cada uno de los paquetes que se transmitirán. Se le ha asignado el mismo valor que su contraparte en la clase cliente, para simplificar los mecanismos de transmisión de datos.

8-12 Estas variables tienen el mismo significado que las declaradas en el código del cliente.

13 La variable doneSending será utilizada como bandera para indicar a los demás threads que se ha terminado de enviar el documento, y de que el proceso normal de desplegado del documento puede continuar (ver *5.3.2.2 Comunicación entre threads*), lo cual permite al usuario que controla la aplicación ver en su propia pantalla el documento que esta enviando.

14 Este es el método constructor de la clase (*ver 2.3.1.2 Clases*), el cual debe cumplir con el requisito de tener el mismo nombre que la clase. Se reciben como parametros la direccion IP Multicast,el puerto que se utilizara para la comunicacion, un URL que indica la localización de los documentos LaTeX que se van a enviar, y una variable de contexto para enviar los mensajes de avance del envio de la información a los clientes.

15-21 En este segmento de código se hace un tratamiento de errores igual al del cliente.

17-18 Aquí vemos como se trata de abrir un socket multicast con el método `MulticastSocket()` para iniciar la comunicación (ver 5.4 *Programando aplicaciones multicast con Java*), pero un punto curioso es que puede observarse la ausencia del método `joingGroup` utilizado en el cliente ¿por qué?es que acaso no queremos enviar nuestra información a un grupo multicast, y por lo tanto debemos unirnos a él? Recordemos que una computadora tiene tres opciones al intervenir en una comunicación multicast: *a)* no participar en ella; *b)* participar solo enviando, pero no recibiendo y *c)* participar enviando y recibiendo (ver *3.2.4 Grupos de multidifusión IP*), debido a que este es el código del servidor solo nos interesa enviar los documentos, pero no recibirlos. Es por esta misma razón que el constructor recibe como parámetro la dirección IP multicast, ya que como transmisor debe especificar en los paquetes que genere la dirección del grupo destino (ver *3.2.5 Envío y recepción de paquetes IP multidifusión).*

40-51 Esta sección se encarga de enviar los documentos LaTeX que se encuentran en el URL que se paso como argumento al constructor del servidor, al ***grupo Multicast*** destino; el documento también es mostrado en el servidor, de tal manera que el método sincronizado `setDoneSending()` indica a los threads de la IDVI que el documento ha sido enviado éxitosamente y que puede ser mostrado en pantalla. Es por esta misma razón que no deseamos que el servidor reciba los documentos (líneas 17-18) , ya que él mismo los despliega una vez que confirma que los ha enviado. Esta sección implementa el segmento de código discutido en la sección *5.4.1: Transmisión de paquetes multicast.*

55 Se cierra el socket para dar por concluida la transmisión.

56 Se notifica a los threads que están esperando de que el documento ha sido enviado satisfactoriamente haciendo una llamada al método `setDoneSending`

58-61 Este es el método sincronizado `setDoneSending`, el cual controla la ejecución de los threads (ver *5.3.2.2 Comunicación entre threads*).

Al programar esta aplicación, se han podido conjuntar todos los conceptos vistos en los capítulos anteriores, logrando nuestro objetivo, ya que ésta es una aplicación cliente/servidor (ver *2.2 Modelo Cliente - Servidor*), escrita en Java (ver *2.4 El Lenguaje de programación Java*), utilizando la metodología orientada a objetos (ver *2.3 Programación Orientada a Objetos*), que distribuye documentos LaTeX (ver *Capítulo 4. El programa LaTeX*), utilizando un mecanismo de entrega multipunto (ver *Capítulo 3. Mecanismos de entrega multipunto*), usando como protocolos de comunicación a TCP/IP para trabajar en Internet (ver *2.1 Familia de protocolos TCP/IP*), empleando sockets como el mecanismo de control de E/S de red (ver *2.1.3 La interfaz socket*) y que además no esta limitada a funcionar en un sistema en particular, sino que puede correr en cualquier plataforma para el cual exista una maquina virtual de Java (ver *2.4.2 Independencia de plataforma*).

Es de esta manera, pues, que se ha logrado "Configurar un sistema de entrega de documentos LaTeX en Internet, usando un mecanismo de único-envío, múltiple recepción con el lenguaje de programación Java".

Con la finalidad de mostrar la utilidad de nuestro programa, el siguiente capítulo mostrará un caso de aplicación enfocado a la educación a distancia.

Capítulo 6
Caso de Aplicación

En el capítulo anterior se mostró la forma en que se han ampliado las características del programa IDVI para convertirlo en una aplicación cliente/servidor que nos permita enviar documentos LaTeX hacia un grupo prácticamente ilimitado de computadoras conectadas a un grupo multicast.

Pero una herramienta con estas características, por si sola no sería muy útil si no se logra que haya una manera de establecer una interacción entre alumnos y profesores: el profesor debe poder explicar su clase, ya que de poco serviría el solo enviarles los documentos, por lo que su voz, y ¿por que no? tambien su imagen, deben poder ser transmitidas a todo el grupo y de esta manera poder tener una verdadera clase virtual. A su vez, los alumnos, en algún momento tendrán dudas, por lo que también sería conveniente de que formularan sus preguntas de viva voz al profesor y que éste pueda contestarlas por el mismo medio.

Es por eso que el próposito de este capítulo es mostrar un ejemplo de la integración de IDVI Multicast junto con un par de aplicaciones para distribución de audio y video multicast (VIC y VAT) que puede servir como guía para preparar un curso de ejemplo para usarlo cómo educación a distancia

Hay que hacer notar que tratar de definir una metodología de educación a distancia esta más allá del propósito de esta tesis, así que sólo se limitará el alcance de este capítulo a mostrar la forma en que puede crearse un documento LaTeX que contenga algunos temas de Cálculo y Algebra Lineal y la configuración de las herramientas de distribución de audio y video multicast para armar una solución completa y barata que pueda servir como ejemplo y guía para la preparación de un curso de educación a distancia.

6.1 Vic y Vat

Para poder ofrecer una experiencia multimedia de audio y video se utilizarán un par de aplicaciones multimedia diseñadas para tal próposito: Vic y Vat. Su distribución es gratuita y existen varias versiones precompiladas para diferentes versiones de sistemas operativos. En nuestro caso utilizaremos las versiones disponibles para Windows.

6.1.1 Vat.

Al igual que Vic, Vat (Visual Audio Tool) es una aplicación multimedia de tiempo real también desarrollada por el Network Research Group del Laboratorio Lawrence Berkeley (figura 6.1) la cual permite realizar audio conferencias a través de Internet. Vat esta basado en el standard de Internet RTP (Real Time Protocol) desarrollado por el grupo de trabajo Audio/Video Transport del IETF (Internet Task Force). Aunque vat puede ser ejecutado en una comunicación punto a punto usando direcciones IP unicast, esta diseñado principalmente para funcionar como una aplicación de audio conferencia multiusuario.

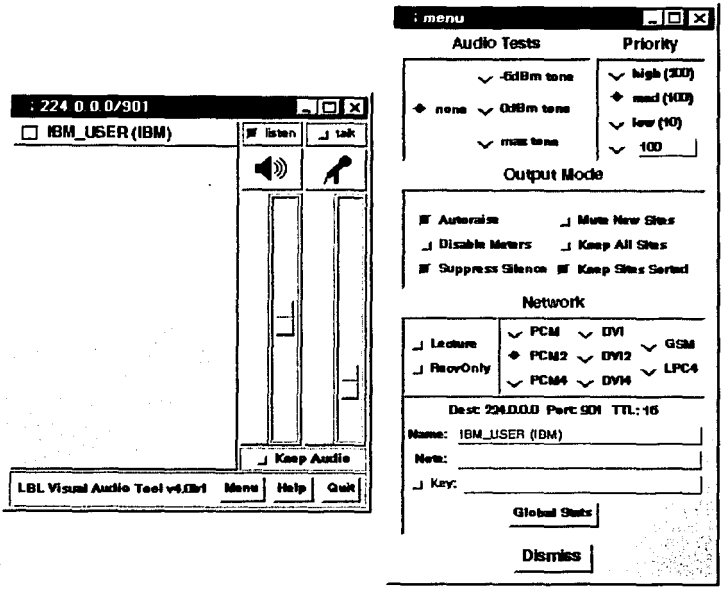


Fig. 6.1 Aspecto general de una sesión con Vat.

6.1.2 Vic

Vic es una aplicación diseñada para permitir la transmisión de video en tiempo real hacia grupos multicast desarrollada por el Network Research Group del Laboratorio Lawrence Berkeley en colaboración con la Universidad de California (*figura 6.2*). Vic esta diseñado con una arquitectura muy flexible y extensible que soporta ambientes heterogéneos y diferentes configuraciones. Por ejemplo, en ambientes con gran ancho de banda el video puede ser transmitido usando compresión por hardware,

mientras que en ambientes como Internet la codificación del video puede ser llevada a cabo por medio del software.

Vic también permite configurarse en la modalidad de "seguir al que habla", esto quiere decir que interactuando con vat, vic muestra el video transmitido por la persona que esta hablando en el momento.

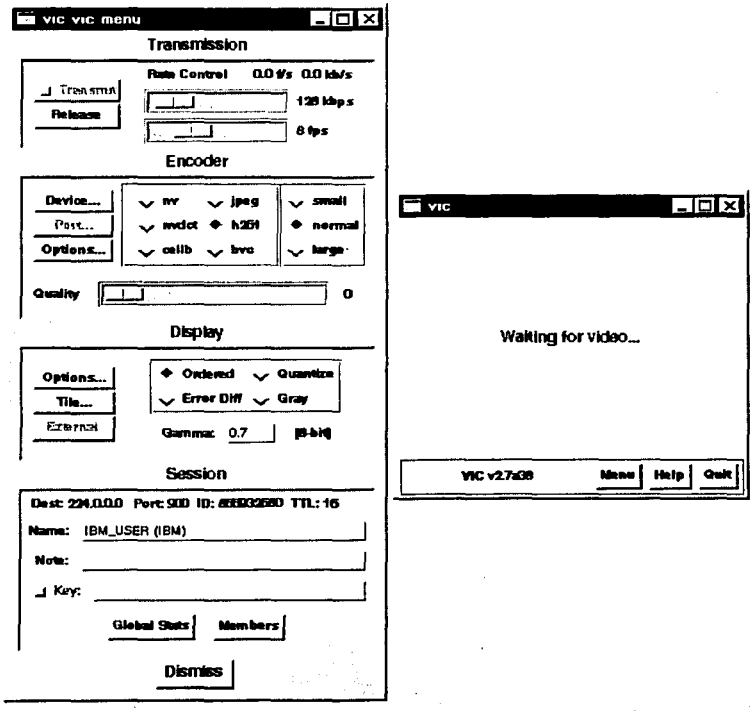


Figura 6.2 - Aspecto general de una sesión con Vic

6.2 Preparación del curso

Antes de que podamos iniciar nuestra clase, debemos contar con el material necesario para impartirla, por lo que el primer paso a seguir es preparar nuestros apuntes con LaTeX.

El contenido y forma de los apuntes va a depender mucho del estilo y enfoque que se les quiera imprimir. Por fines prácticos no se ha elaborado un curso completo tal y como sería en la realidad, sin embargo puede elaborarse cualquier tema, no importando el nivel de simbología matemática que se vaya a utilizar de acuerdo a lo avanzado del curso, desde operaciones algebraicas, ecuaciones en general, determinantes, sistemas de ecuaciones lineales, cálculo diferencial, integración numérica, serie de Fourier, etc.

6.2.1 Elaboración de los documentos

Como se se vio en el ***capítulo 4***, el ciclo de creación de un documento LaTeX, consiste en tres fases: en la captura del documento fuente, conocida como **fase de edición**; la **fase de formateo**, la cual convierte el archivo fuente (texto ascii) en un archivo binario (IDVI) y por último la **fase de visualización** del documento final (ver *4.3 Elaboración de documentos*).

La primera fase, la de edición puede ser hecha con cualquier editor de textos, siempre y cuando el archivo se guarde como un archivo de texto plano (ASCII). La *figura 6.3* muestra el aspecto de edición del código fuente para el tema de "Operaciones fundamentales con expresiones algebraicas".

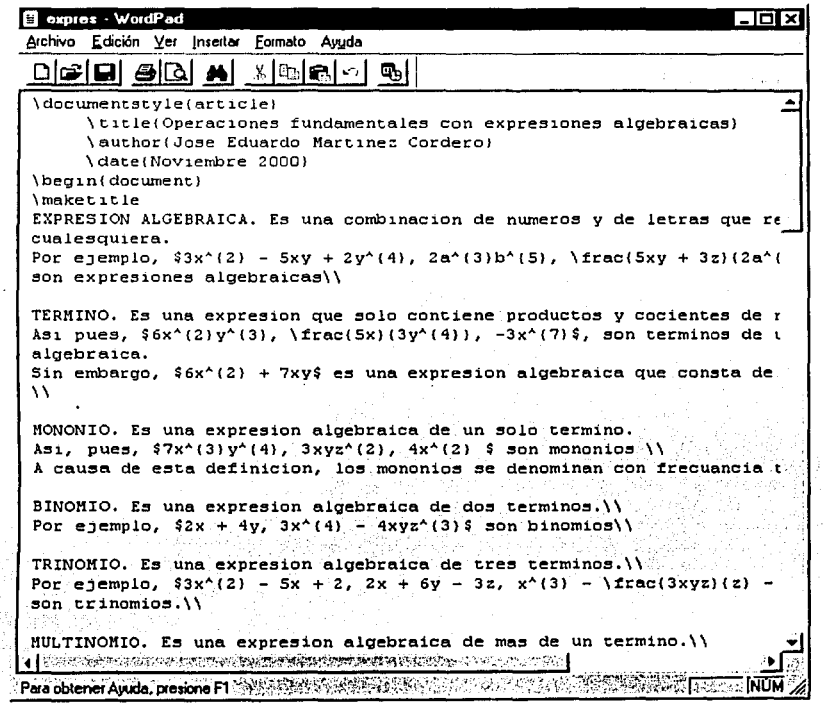


Figura 6.3 Edición del archivo fuente LaTeX para el curso de demostración.

Una vez formateado se procede a su visualización, tal y como se muestra en las *figura 6.4.*

Si se esta conforme con el resultado, se procede a probarlo localmente con IDVI; hay que prepar los documentos de acuerdo a lo visto en *5.2.2 "Preparación de un documento para su vista con* IDVI ", las *figuras 5.2, 6.5 y 6.6.* muestran la forma en que luciría en el navegador .

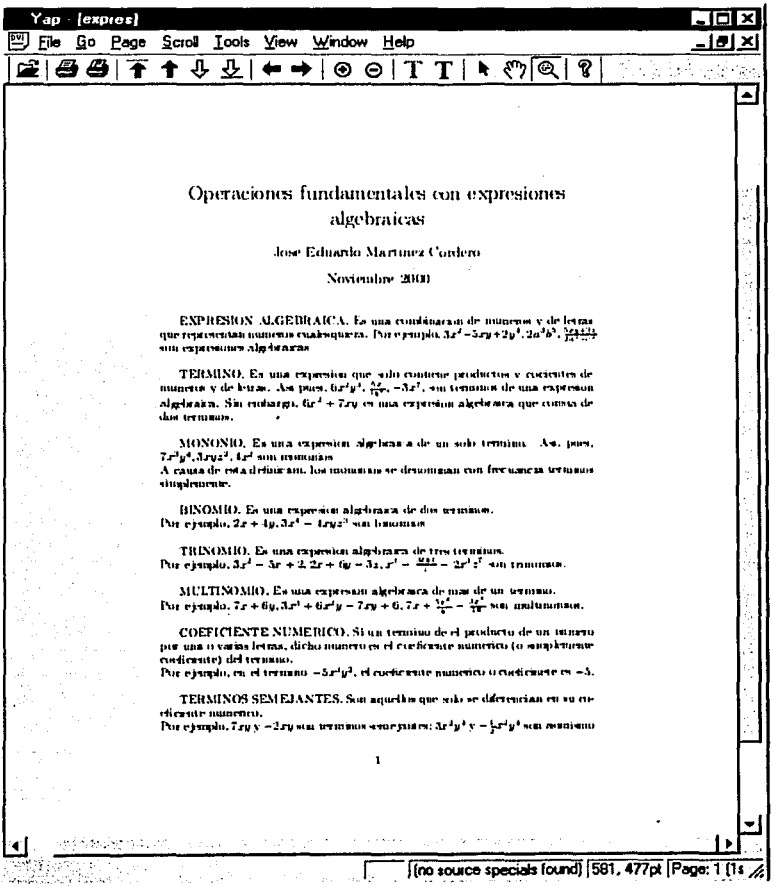


Figura 6.4 Visualización del documento con un visor para documentos DVI.

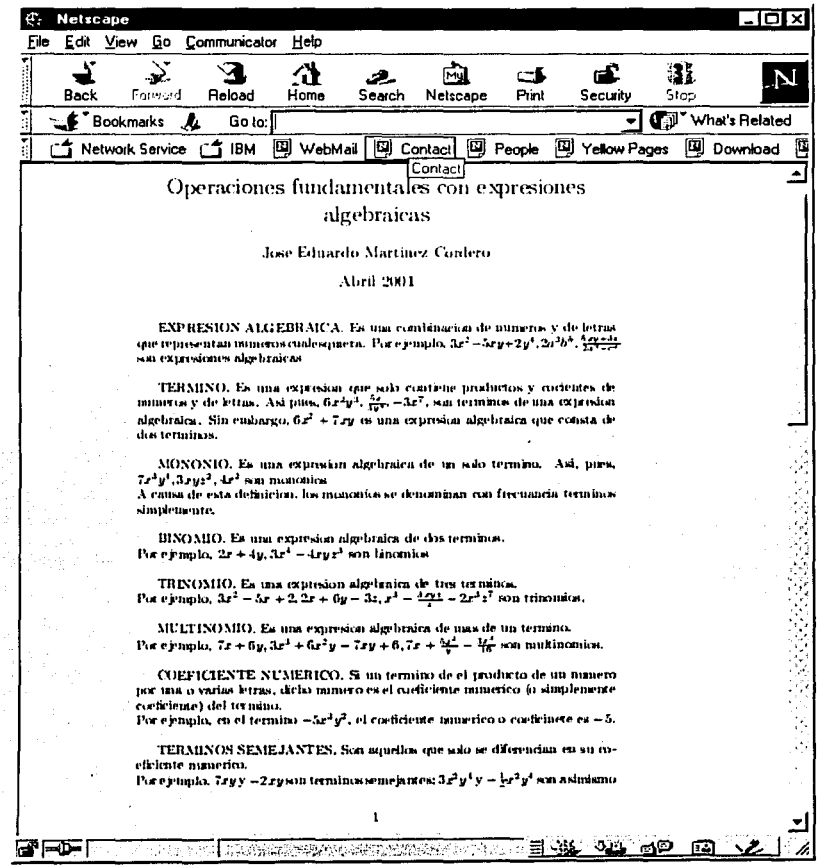
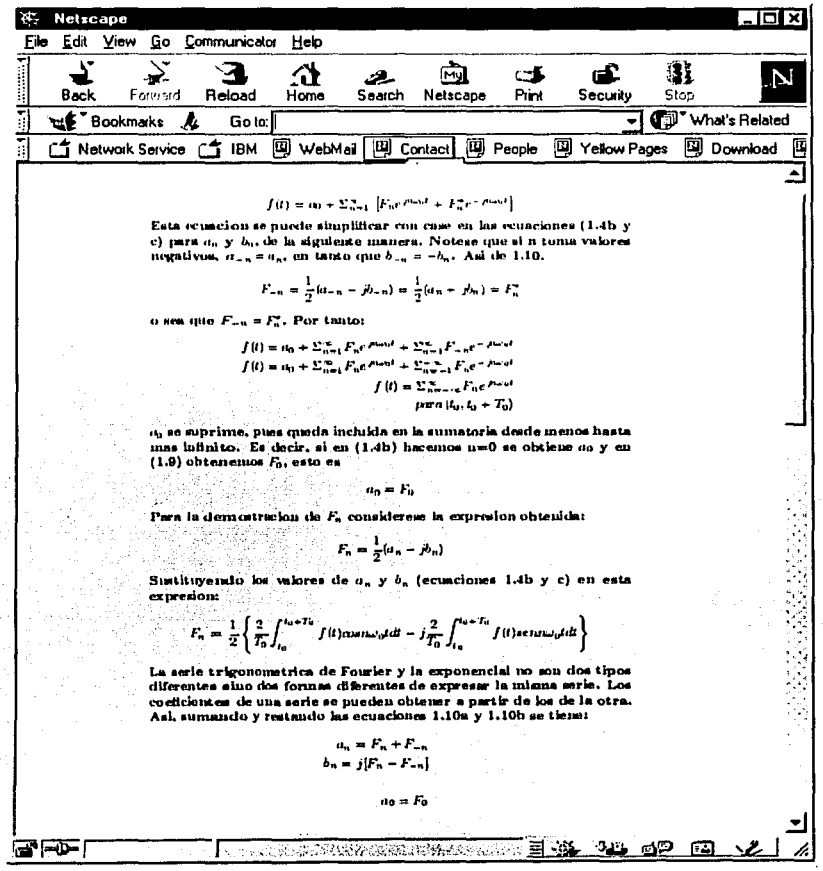


Figura 6.5 Ventana de Netscape mostrando la primer página del documento



$$f(t) = a_0 + \Sigma_{n=1}^{\infty} \left[F_n e^{jn\omega_0 t} + F_n^* e^{-jn\omega_0 t}\right]$$

Esta ecuacion se puede simplificar con ease en las ecuaciones (1.4b y c) para a_n y b_n, de la siguiente manera. Notese que si n toma valores negativos, $a_{-n} = a_n$, en tanto que $b_{-n} = -b_n$. Asi de 1.10.

$$F_{-n} = \frac{1}{2}(a_{-n} - jb_{-n}) = \frac{1}{2}(a_n - jb_n) = F_n^*$$

o sea que $F_{-n} = F_n^*$. Por tanto:

$$f(t) = a_0 + \Sigma_{n=1}^{\infty} F_n e^{jn\omega_0 t} + \Sigma_{n=1}^{\infty} F_{-n} e^{-jn\omega_0 t}$$
$$f(t) = a_0 + \Sigma_{n=1}^{\infty} F_n e^{jn\omega_0 t} + \Sigma_{n=-1}^{-\infty} F_n e^{-jn\omega_0 t}$$
$$f(t) = \Sigma_{n=-\infty}^{\infty} F_n e^{jn\omega_0 t}$$
$$para\ (t_0, t_0 + T_0)$$

a_0 se suprime, pues queda incluida en la sumatoria desde menos hasta mas infinito. Es decir, si en (1.4b) hacemos n=0 se obtiene a_0 y en (1.9) obtenemos F_0, esto es

$$a_0 = F_0$$

Para la demostracion de F_n consideremos la expresion obtenida:

$$F_n = \frac{1}{2}(a_n - jb_n)$$

Sustituyendo los valores de a_n y b_n (ecuaciones 1.4b y c) en esta expresion:

$$F_n = \frac{1}{2}\left\{\frac{2}{T_0}\int_{t_0}^{t_0+T_0} f(t)\cos n\omega_0 t\,dt - j\frac{2}{T_0}\int_{t_0}^{t_0+T_0} f(t)\operatorname{sen} n\omega_0 t\,dt\right\}$$

La serie trigonometrica de Fourier y la exponencial no son dos tipos diferentes sino dos formas diferentes de expresar la misma serie. Los coeficientes de una serie se pueden obtener a partir de los de la otra. Asi, sumando y restando las ecuaciones 1.10a y 1.10b se tiene:

$$a_n = F_n + F_{-n}$$
$$b_n = j[F_n - F_{-n}]$$

$$a_0 = F_0$$

Figura 6.6 Ventana de Netscape mostrando otra página del documento

Una vez que se tiene todo listo, se esta en posibilidad de empezar la clase. La *figura 6.7* muestra como podría lucir el escritorio del profesor que iniciará la clase. Pueden notarse cuatro ventanas, las cuales (iniciando de izquierda a derecha y de arriba hacia abajo) representan respectivamente: el navegador en el que corre el applet IDVI Multicast, mostrando el documento a distribuir, en este caso se trata de Netscape; en seguida se encuentra la ventana de controles, la cual permite controlar la secuencia en que serán desplegados los documentos, esta ventana esta disponible únicamente para la parte del servidor, ya que el profesor es el único que tiene el control del despliegue (ver también *figuras 6.8 y 6.9*); a continuación la ventana de vic, para permitir la transimisión de audio, y finalmente la ventana de vat para la entrega de video.

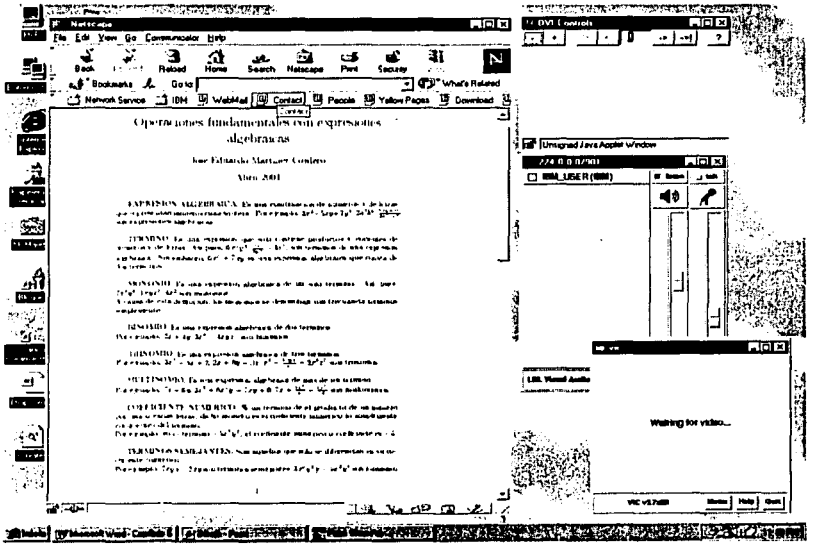


Figura 6.7 : Aspecto general del ambiente del servidor de una sesión de clases a distancia

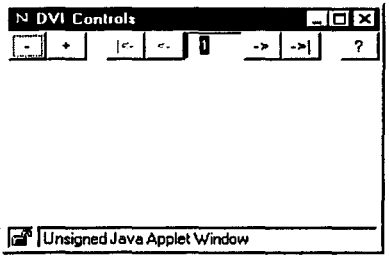


Figura 6.8 Ventana de controles. Las flechas hacia la derecha envían hacia la siguiente página del documento; las de la izquierda hacia la página anterior.

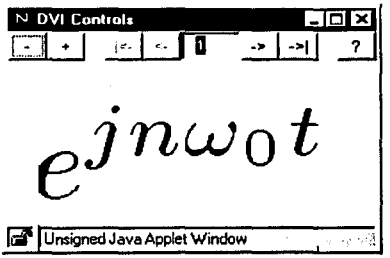


Figura 6.9 Ventana de controles con zoom. El profesor tiene la capacidad de hacer un acercamiento a ciertas áreas del documento. Esta característica es únicamente disponible en el servidor.

El aspecto del escritorio de un estudiante podría lucir como se muestra en la *figura 6.10*. Nótese la ausencia de la ventana controladora, ya que como vimos, la parte cliente de la aplicación no permite a los alumnos cambiar arbitrariamente la página que deben visualizar. También se muestran las ventanas de vic y vat, y podemos ver que en la ventana de vat aparece ya un listado de los usuarios que actualmente estan conectados a esta clase. Finalmente, debido a que es una aplicación escrita en Java, la clase puede ser recibida en cualquier navegador que soporte este lenguaje de programación; aquí podemos ver que el alumno utiliza el Microsoft Explorer para recibir los documentos, mientras que el profesor (*figura 6.7*) utiliza Netscape para la distribución.

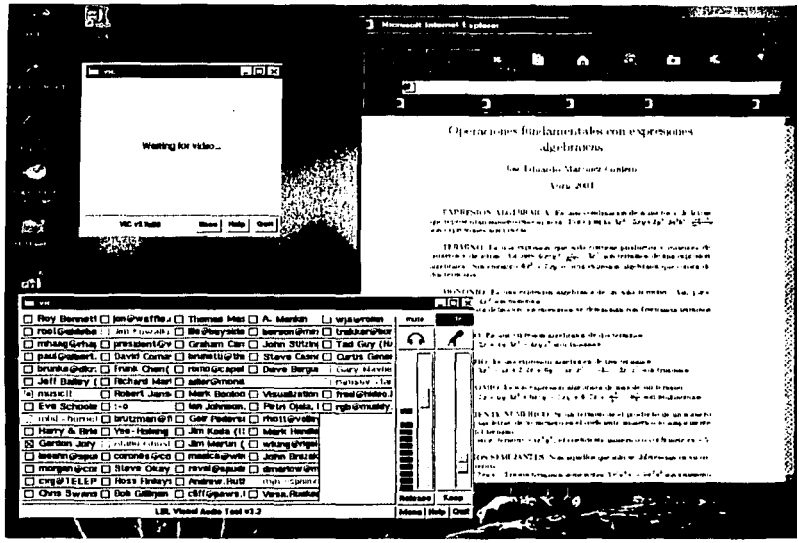


Figura 6.10 Posible ambiente en el escritorio de un estudiante. Con las ventanas de Vic, Vat e Internet Explorer activas.

Como se puede observar gráficamente en las figuras de este capítulo, se ha logrado el objetivo propuesto para este trabajo de tesis: la distribución de documentos LaTeX hacia múltiples destinos, y se ha visto además su aplicación en un posible curso de educación a distancia.

En el capítulo siguiente se verán cuales son las perspectivas a futuro de aplicaciones de este tipo.

Capítulo 7

Perspectivas a futuro

El modelo de comunicación de redes de **multicast** está tomando un gran auge sobre todo para aplicaciones multimedia interactivas; ya sea para enviar datos de video y audio de conferencias en línea o para enviar grandes volúmenes de datos hacia varios nodos o destinos muy específicos.

El avance en este sentido ha impulsado el desarrollo de tecnologías de comunicación más eficientes. El método de enlace entre nodos de una red *multicasting* abre paso a la implementación a nivel empresarial de tecnologías multimedia interactiva y permitir así el uso eficiente de ellas.

Cualquier aplicación multimedia o de distribución de datos que requiere transmisión punto-multipunto será beneficiada por los servicios del multicast que están disponibles en equipos de red (por ejemplo, ruteadores).

Sin los servicios multicast, las aplicaciones son forzadas a enviar la información a grupos de usuarios con paquetes unicast; esto significa que un servidor de video, por ejemplo, tendría que enviar una copia individual de cada paquete de datos a todos y cada uno de los usuarios de un grupo antes de poder mandar el próximo paquete; este

tipo de transmisión es ineficiente para el servidor y para la red, ya que genera gran cantidad de tráfico que afecta su desempeño.

En contraste, los servicios multicast basados en red permiten a los servidores de video y otras fuentes de datos enviar un solo paquete que será copiado y distribuido automáticamente por los dispositivos de la red (por ejemplo, ruteadores del backbone) a cada estación de trabajo del grupo de usuarios sin importar qué tan cerca o lejos se localicen.

7.1 Aplicaciones multimedia.

Las aplicaciones multimedia interactivas están guiando a las universidades y empresas hacia una nueva generación de tecnologías de alta velocidad.

Esta tecnología podría capitalizarse en diversas ramas académicas y empresariales: la capacitación, conferencias virtuales, clases virtuales, juntas, etc; en general, teleconferencias, así como la transmisión de datos en grandes volúmenes (literatura, revistas, periódicos, folletos, cursos, etc.).

Con los avances que ha tenido y sigue teniendo la tecnología multimedia interactiva podrían cambiar los hábitos de trabajo y elevarse la productividad de las personas: imaginemos por ejemplo que (además de poder enviar datos de video hacia determinados puntos en una red, ya sea de manera remota o de manera local, en tiempo real) se puedan realizar juntas de trabajo virtuales. Es decir, sin moverse de un lugar, se podrá sostener una reunión con personas que residan en cualquier parte del mundo. Y no solamente eso, sino que con una mejor calidad y un costo menor que el de las soluciones existentes hasta el momento.

Imaginemos que los gerentes de una empresa que tiene oficinas en el Distrito Federal, Monterrey, Guadalajara y Aguascalientes, realizan juntas semanales para ver el estado en que se encuentran estas sucursales. Con esta tecnología integrada, se podrían sostener estas reuniones sin necesidad de moverse físicamente; solo hará falta que se acuerde un horario y que se haga uso de estos recursos (multicasting y multimedia), para que este problema quede resuelto de manera eficiente.

También podrían sostenerse sesiones multiusuario sobre documentos compartidos de cualquier herramienta de trabajo de hoja de cálculo, procesador de textos o presentaciones.

7.2 Tecnología "push".

Los proveedores de contenidos están usándolo para distribuir información del Web hacia varios servidores al mismo tiempo. Algunos están enviando grandes cantidades de datos no en línea, y otros, servicios de cable-noticias, en los que el contenido fijo viaja hacia diferentes lugares de manera simultánea. Un ejemplo muy ilustrativo es el caso de la compañía Pointcast. La red de PointCast es un servicio gratuito de noticias Internet que permite al usuario estar informado al difundir noticias e información personalizadas directamente a la pantalla de la computadora. Uno selecciona las noticias que está interesado en recibir, tales como noticias de CNN, el periódico Wall Street, y muchos más..

7.3 Panorama actual.

La transmisión IP Multicast está diseñada para reducir la congestión en redes públicas y privadas por medio de un modelo de transmisión punto multipunto en vez del modelo que se emplea actualmente que es punto a punto.

En el primer caso, varios usuarios pueden acceder a un flujo de datos y en el segundo, en cambio, cada usuario genera uno individual, lo cual produce congestión en la red.

Debido a que IP Multicast reduce la congestión en la red así como el esfuerzo de los servidores y al mismo tiempo da paso a nuevas aplicaciones, los visionarios de Internet han comenzado a demandar de los Proveedores de Servicios de Internet (ISP) el enlace multicasting con la esperanza de prevenir un desastre en el backbone de Internet en la próxima década.

Además, existen reportes que indican, por ahora, que muchos de los Proveedores de Servicios de Internet han implementado Multicast ya sea como un servicio con costo

adicional o como parte integral de sus servicios, y una gran cantidad de los corporativos están cambiando a Multicast.

Las cosas han llegado a tal punto que -mientras la reunión de IP Multicast del año pasado estaba orientada a convencer a los proveedores de servicios de Internet, desarrolladores de software y fabricantes de equipo para que ofrecieran el soporte Multicast-, este año la reunión fue para discutir la administración del tráfico multicast así como el ruteo interdominios que se requerirá si el IP Multicasting se lanza comercialmente como se pretende.

Van Jacobson, científico en jefe de Cisco Systems, integrante del consejo del Network Research Group en el Lawrence Berkeley National Lab, piensa que "las mejoras tecnológicas incluyendo los protocolos y herramientas estándar probados, se combinarán con la demanda empresarial para hacer que este año se lance al ámbito comercial la iniciativa de IP multicasting".

Es importante mencionar que el grupo Berkeley fue el que instrumentó la creación de herramientas de trabajo colaborativo multicasting y Jacobson fue el creador de la idea del protocolo independiente Multicast, un protocolo ampliamente difundido para esta tarea.

Existen muchas subredes de redes públicas y privadas que han tomado la decisión de instalar esta tecnología en sus instalaciones; aunque, para que este servicio logre descongestionar la red de redes, es necesario todavía que las subredes (*islas*) en donde esta tecnología ya está funcionando, se conecten entre sí dominio a dominio tal como sucedió cuando Internet se hizo público. Como ya vimos este es el objetivo que el MBONE ha venido persiguiendo desde hace algunos años.

Por lo pronto lo que se puede visualizar es que las empresas empiezan a empujar hacia este tipo de comunicaciones dado que en la actualidad existen ya muchas redes con esta tecnología instalada.

En respuesta a la creciente necesidad de aplicaciones multimedia y distribución de datos en tiempo real, la mayoría de los fabricantes de equipos para red ya están incorporando el soporte para esta tecnología, entre ellos se puede citar a 3Com, Nortel Networks (fusión de Bay Networks y Northern Telecom), Lucent Technologies, Extreme Networks, Cisco Systems y Fore Systems, como los principales, por lo que los servicios de multicast de la capa de la red vienen disponibles desde la fábrica en los

ruteadores high-end, concentradores de ruteo y switches. Este desarrollo requiere que los diseñadores de redes institucionales y corporativos evalúen varios protocolos multicast, tales como: MOSPF, PIM, DVMRP, IGMP.

Asimismo la iniciativa IP multicast (http://www.ipmulticast.com) ha reunido a una gran cantidad de empresas con el objetivo de impulsar a la tecnología y aplicaciones multicast y permitir que su uso se difunda.

Aunque multicast requiere de nuevos protocolos, en muchos casos éstos pueden ser ubicados de manera sencilla con una simple actualización de software en los dispositivos de ruteo; sin embargo, hay otros dispositivos que necesitarán mayor memoria o en casos graves una actualización en los CPUs.

Capítulo 8
Conclusiones

Vivimos actualmente una era de cambios tecnológicos constantes que están cambiando drásticamente la manera en que nos comunicamos, las distancias se están acortado cada vez más gracias a los avances en la tecnología, e Internet esta ayudando en gran medida a que todos estos cambios sean posibles. Este paso acelerado provoca que tecnologías que hace algunos meses eran innovadoras hoy en día sean totalmente obsoletas.

Sin embargo si el ancho de banda no fuera una limitante, estaríamos viviendo transformaciones aún más dramáticas que las que hemos experimentado hasta ahora. La información y los servicios que potencialmente pueden concretarse, se ven frenados por un cuello de botella llamado líneas de comunicación.

Uno de los campos que se han visto afectados por esta limitante es el de la educación a distancia: desde hace algunos años han existido diversos intentos para ir moldeando la propuesta educativa de varias instituciones universitarias, de manera tal, que se beneficien con la implementación de las nuevas tecnologías con el fin ofrecer mayores beneficios a los estudiantes; pero a pesar de la velocidad a la que se producen los cambios tecnológicos y la rapidez con la que se incorporan a un modelo educativo son distintas, no ha sido posible integrar con mucho éxito las nuevas tecnologías al modelo educacional a distancia.

Desde hace varios años, prevalece la creencia de que algún día el ancho de banda será ilimitado, o que su costo será despreciable. Sin embargo, mientras esto sucede, las instituciones deben enfrentar un costo, cada día más alto, en el rubro de telecomunicaciones.

En la actualidad, entre los servicios que desde el punto de vista tecnológico ya son factibles, pero que requieren de un gran ancho de banda para ser comercialmente aceptados, encontramos entre otros: Video on Demand (p. ej. bajar películas por Internet a petición del usuario); música por Internet y distribución de software por Internet.

Lo que se ha podido demostrar con este trabajo de tesis es que aplicando conceptos ya existentes tales como multicast no es necesario hacer grandes desembolsos de dinero para implementar una solución de educación a distancia, ya que se puede utilizar la infraestructura actual de Internet; además se asegura que la utilización del ancho de banda será óptima, al no estar sobrecargando la red con información repetida y redundante; es una aplicación de fácil instalación y manejo; es una aplicación barata, ya que se basa principalmente en tecnologías gratuitas: Linux, LaTeX y Java.

El punto de partida del trabajo lo constituye la problemática de enviar paquetes de información desde una fuente a múltiples destinos simúltaneamente sin afectar los recursos de red; enseguida, hacer de esta distribución de paquetes a múltiples destinos, algo provechoso, aplicando la técnica al envío de documentos completos que contengan información útil al usuario final; y finalmente presentar una propuesta de aplicación del sistema enfocada en la llamada educación a distancia a través de Internet.

Como resultado tangible de esta tesis, se ha mejorado el sistema IDVI de visualización de documentos LaTeX creado por Garth A. Dickie, para permitir que se convierta en una aplicación cliente/servidor que usa un método de envío multicast. Estas mejoras permiten que desde el servidor de documentos un profesor tenga el control de los documentos que puede enviar a sus alumnos y en el momento en que él

lo decide; y en la parte del cliente, los alumnos recibirán los documentos según los vaya enviando el profesor.

Gracias a que es una aplicación pensada para Internet, los documentos pueden llegar a prácticamente un número ilimitado de personas en cualquier parte del mundo, permitiendo que no sean necesarios grandes desmbolsos de dinero haciendo que el concepto de educación a distancia sea toda una realidad, finalmente alcanzando el objetivo de este trabajo de tesis.

Bibliografía

Late Nigth. Advanced Java.
Jason Wehling, Vidya Bharat.
QUE Editorial
USA, 1996

Advanced Techniques for Java Developers
Daniel J. Berg, J. Steven Fritzinger.
Ed. Wiley & Sons.
USA, 1997.

The Java Programming Language. Second Edition.
Ken Arnold, James Gosling
Ed. Addison-Wesley.
USA, 1998

Java. Distributed Computing.
Jim Farley.
Ed. O'Reilly & Associates Inc.
USA, 1998

Java for C/C++ Programmers.
Michael C. Daconta.
Ed. Wiley & Sons.
USA, 1996

Java in a Nutshell.
David Flanagan.
Ed. O'Reilly & Associates Inc.
USA, 1997

Concurrent Java Programming. Second Edition. Design Principles and Patterns.
Doug Lea
Ed. Sun Microsystems Press
USA, 1999

C++ A su alcance. Un enfoque orientado a objetos.
Luis Joyanes Aguilar
McGraw-Hill/Interamericana de España
España, 1994

Redes globales de información con Internet y TCP/IP
Douglas E. Comer
Prentice Hall Hispanoamericana S.A
México, 1996

Referencias en Internet

IDVI

IDVI
http://www.geom.umn.edu/java/idvi/

LaTeX

TeX Users Group Home Page
http://www.tug.org

CervanTeX. Grupo de usuarios de TeX hispanohablantes
http://apolo.us.es/CervanTeX/QuesTeX.html

LaTeX para el absolutamente inexperto
http://mailweb.udlap.mx/~aleph/alephzero8/latex.html

Primeros pasos en LaTeX
http://members.es.tripod.de/bausela/

Multicast

IP Multicast Initiative
http://www.ipmulticast.com

How IP Multicast Works. An IP Multicast Initiative White Paper.
A theenical overview of IP Multicast concepts, addressing, group management and approaches to routing
http://www.ipmulticast.com/community/whitepapers/howipmcworks.html

Introduction to IP Multicasting Routing
By Chuck Semeria and Tom Maufer
http://www.3com.com/nsc/501303.html

mrouted - IP multicast routing daemon
http://www.neosoft.com/neosoft/man/mrouted.8.html

Frequently Asked Questions (FAQ) on the Multicast Backbone (MBONE)
http://www.cs.columbia.edu/~hgs/internet/mbone-faq.html

MBONE: Multicasting Tomorrow's Internet
http://www.savetz.com/mbone/

RFC1112 . Host Extensions for IP Multicasting
http://www.faqs.org/rfcs/rfc1112.html

IP Multicast Backgrounder
http://www.ipmulticast.com/community/whitepapers/backgrounder.html