UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

APLICACION DISTRIBUIDA DE DIAGNOSTICO DE VINCULOS DE UN SERVIDOR DE WWW

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A :

ANIBAL JESUS AVELAR ROSALES

DIRECTOR: ING. LUCILA P. ARELLANO MENDOZA

MEXICO, D.F., SEPTIEMBRE DEL 2000

Dedicatorias

A mis padres ... por apoyarme siempre.

A mi familia ... por creer en mi.

A Lidia ... por ser mi motivación.

A todos los que colaboraron y me ayudaron ... gracias.

Agradecimientos

Un agradecimiento especial a la Dirección General de Servicios de Cómputo Académico (DGSCA) y al Instituto de Investigaciones en Matemáticas Avanzadas y en Sistemas (IIMAS). Por darme recursos, equipo y todo lo necesario para la realización e implementación de este proyecto.

Prólogo

Durante el presente trabajo, se verá cómo se puede realizar una aplicación distribuida de diagnóstico de vínculos de un servidor de WWW.

En el capítulo 1 se verán varios conceptos sobre redes de computadoras. Se conocerán los tipos y topologías de red más usuados. Se definirá qué es un protocolo de comunicación de red y para que sirve. El modelo OSI. Describiremos algunos tipos de protocolos y el caso especial de la familia de protocolos TCP/IP. Por último, se verá que son y cómo funcionan los sockets de conexión de red.

En el capítulo 2 se verá como está estructurada la red Internet y cuál es su mecanismo de funcionamiento. Que servicios ofrece Internet.

En el capítulo 3 se verá uno de los servicios de Internet que más desarrollo ha tenido en los últimos años: el World Wide Web. Se describirá como funciona el protocolo de comunicación HTTP y como funcionan los servidores de HTTP. Se verán los tipos básicos de navegadores(browsers) para Web que existen actualmente.

En el capítulo 4 se dará un breve resumen del lenguaje de programación Java. Se describirán cuáles son sus ventajas y cuáles sus desventajas. Se verá que son los Java Applet y su relación con el World Wide Web. Como se construyen interfaces gráficas con el nuevo esquema de construcción de GUI´s de Java: el Swing. Por último, se denotará el esquema de seguridad en Applets y por qué está definido así.

En el capítulo 5 se verá el paradigma Cliente/Servidor y su uso en Internet. Se aprenderá a programar aplicaciones en red con Java utilizando el paquete java.net. Se verán ejemplos de uso de algunas clases para programación en red que posee Java.

En el capítulo 6 se verá qué es una aplicación distribuida y qué tipos de aplicaciones distribuidas existen. Qué es lo que debe cumplir un programa o modelo para poder considerarse distribuido. Se aprenderá a programar aplicaciones distribuidas por medio del paquete RMI(Remote Method Interface) de Java.

Por último, en el capítulo 7 se desarrollará una aplicación distribuida de análisis de vínculos de un URL en un servidor de WWW. Componentes que se necesitan para construir la aplicación. Elementos de diseño. Construcción de la aplicación. Explicación detallada de las partes que componen el sistema. Interacción con el usuario. Interfaz gráfica de la aplicación.

Finalmente se presentarán las conclusiones de éste proyecto.

Indice

Objetivo

Realizar una aplicación distribuida cuya principal función será la de verificar la validez de los hipervínculos en un sitio WWW(World Wide Web). El sistema deberá probar tanto los vínculos internos (los que ligan a páginas o imágenes dentro del mismo servidor Web local) como los externos (los que ligan a paginas o imágenes en un servidor Web remoto).

Deberá generar un reporte completo de los vínculos que son válidos y de los que no lo son, así como las características del vínculo (sí son archivos HTML, JPG, GIF, CGI, EXE, etc.). El reporte dirá en forma clara y concisa, a través de una interfaz gráfica, el estado de un sitio WWW.

La aplicación será distribuida y multiplataformas, es decir, una aplicación que se podrá distribuir por la red, y ser ejecutada sobre cualquier tipo de arquitectura y sistema operativo.

Introducción

Contexto del problema

Los sitios WWW (World Wide Web) o Web son dinámicos. Al menos deberían serlo. Deberían cambiar regular y rutinariamente. Se deben añadir páginas y conjuntos de páginas al surgir la necesidad, vínculos a documentos internos y a sitios externos, los archivos antiguos deben mantenerse actualizados para complementar al nuevo contenido que surge. Lo último que se quiere hacer, es conducir a los visitantes del sitio a un callejón sin salida, por lo que todos los vínculos en los que se haga clic deben llevar aun destino. De otra forma los visitantes perderán respeto por el sitio y la confianza en sus capacidades. Un sitio Web disfuncional puede dañar a una empresa en imagen, confianza y credibilidad, que es exactamente lo que necesita para mantener el negocio en un lugar predominante en la mente de sus clientes. Si el sitio tiene errores de descuido (incluso pequeños, como errores de ortografía) su imagen decae, porque los visitantes lo ven menos creíble. Pero nada destruye tanto como los vínculos que llevan a ninguna parte, imágenes que no pueden ser localizadas y formas que no funcionan.

En los primeros días del diseño Web (1994, aprox.), sólo se montaba el sitio Web en un servidor, se ponían los archivos y se hacia clic en todos los vínculos para asegurarse de que conducían al lugar correcto, y se usaba un editor de texto común y corriente para hacer los cambios que fueran necesarios. Entonces los sitios no eran tan complejos como ahora, porque la mayor parte eran desarrollados por personas con un limitado número de archivos a desplegar y sin el tiempo necesario para poner más información sobre el Web. Así que realizar un diagnóstico Web total de un sitio era relativamente sencillo.

Pero los sitios de Web de hoy tienen cientos, incluso hasta miles, de archivos, y diferentes personas agregan y alteran sus áreas particulares cada día. Se insertan archivos, se agregan nuevos vínculos ya sea a archivos internos o sitios externos. No hay forma de manejar y verificar los cambios sin ayuda.

Debe existir alguna manera de hacer un diagnóstico del estado de un sitio Web, de una manera mas automática y rápida. Una aplicación que además, funcione sobre cualquier plataforma y sea distribuida, permitiendo con esto poder ejecutarlo desde cualquier lugar. Como se sabe, existen distintos tipos de hardware sobre distintos tipos de arquitectura de computadoras empleando diferentes tipos de sistemas operativos que ejecutan tipos muy diversos de software, todo esto, provoca un caos de incompatibilidad entre lo sistemas, que en ocasiones es imposible salvar. Por tanto, se requiere de una aplicación que pueda ejecutarse en el mayor número de plataformas sin necesidad de muchos cambios o configuraciones difíciles. Si ha esto, se le agrega que la aplicación será distribuida por medio de la red Internet, se tendrá un sistema que podrá ser ejecutado desde cualquier plataforma, en cualquier momento y en cualquier lugar.

La importancia de realizar una aplicación como ésta, radica en la necesidad que tiene un Webmaster (el encargado de administrar un servidor Web) en conocer el estado del servidor que administra, y poder corregir fallas y defectos que tenga su sitio. Esto sería de gran ayuda para un administrador de Web, cuyo objetivo es mantener en óptimas condiciones el sitio. Cada vez más empresas están incursionando en la red de Internet para realizar trabajos de publicidad y de negocios, lo cual hace que el estado óptimo del servidor de Web sea de vital importancia. Además, los investigadores y científicos que consultan y comparten información a través de este servicio de red, necesitan de un sitio Web lo suficientemente confiable para que sus investigaciones se lleven acabo más rápido y sin menos contratiempos.

Solución al problema

Ya se vio cuál es la importancia de una aplicación como ésta. Por lo que se necesita un software que verifique todas las ligas o vínculos de un sitio Web en particular de una manera más rápida y automática, sin necesidad de darle clic uno por uno a cada vínculo. Un software de diagnóstico que no sólo diga cuantas ligas o vínculos tiene un sitio WWW, sino que además nos informe cuáles de esos vínculos son exitosos(están dirigidos aun sitio real) y cuales son fallidos(no existe el vínculo). Informe además, si todas las imágenes contenidas en el sitio, existen y podrán ser cargadas cuando se ingrese a través de un visualizador(browser) de sitios WWW. Existen sitios que tienen un complejo número de vínculos, localizar un vínculo disfuncional es casi imposible, por lo que una aplicación que de al usuario una forma de seleccionar páginas con problemas de URL específicos es una manera de simplificar y aislar el problema.

1
Introducción a las redes de computadoras

Por norma general, se recurre a la informática para satisfacer las crecientes necesidades de información y se suele empezar con unas pocas o una única computadora y unos cuantos periféricos. Poco a poco se van ampliando tanto los recursos de hardware como recursos de software para gestión de la información. Esta ampliación suele llevar asociado un problema de redundancias, tanto de software, datos, hardware, etc. Por ejemplo, cada nuevo equipo va a necesitar de su propia impresora para imprimir informes (redundancia hardware), los datos almacenados en uno de los equipos es muy probable que sean necesarios en otro de los equipos por lo que será necesario copiarlos en este otro equipo (redundancia de datos), las computadoras que trabajan con los mismos datos tendrán que tener los mismos programas para manejar dichos datos (redundancia software). Pues bien, todos estos problemas tienen una fácil solución: las redes de computadoras.

Una red está formada por computadoras con sus periféricos y por elementos que conectan entre sí dichas computadoras. Para compartir impresoras basta con un conmutador, pero si se desea compartir eficientemente archivos y ejecutar aplicaciones de red, hace falta tarjetas de interfaz de red, cables y cualquier medio físico que permita a las computadoras intercambiar bytes de información con otros equipos.

Una red mucho más compleja conecta todas las computadoras de una empresa o compañía en el mundo. Las grandes empresas suelen hacer uso de otras redes de comunicaciones, como puede ser la red telefónica, para interconectar sus redes a otras redes más grandes.

Los beneficios del uso de una red de computadoras son los siguientes:

- Se pueden compartir periféricos costosos, como son impresoras, ploters, modems, tarjetas o scaners. Y así evitar la necesidad de tener un dispositivo por cada equipo.

- Se pueden compartir grandes cantidades de información mediante el empleo de gestores de bases de datos en red. Con ello se evita la redundancia de datos y se facilita el acceso y la actualización de los datos.

- La red se convierte en un mecanismo de comunicación entre los usuarios conectados a ella, ya que permite el envío de mensajes, ya sea entre usuarios de la red local o entre usuarios de otras redes o sistemas informáticos, programando reuniones o intercambiando archivos de todo tipo.

- Se trata de un sistema que intenta ser seguro. Se implementa el uso de sistemas que tratan de impedir que determinados usuarios accedan a áreas de información concretas, o que puedan leer la información pero no modificarla. El acceso a la red está controlado mediante nombres de usuario y claves de acceso. Estos sistemas varían en su nivel de implementación de seguridad, existiendo redes que son más seguras que otras, dependiendo del software y hardware que usen.

Aunque se pueden utilizar diversos sistemas de interconexión vía los puertos series y paralelos, estos sistemas baratos no ofrecen la velocidad e integridad que tiene un sistema operativo de red seguro y con altas prestaciones que permita manejar muchos usuarios y recursos.

1.1 Tipos de redes

Ante la necesidad que se tiene de poder compartir información de manera rápida y confiable, se desarrollaron distintos tipos de arquitecturas y tipos de redes. Éstas permiten una mayor integración y disponibilidad de los datos desde cualquier computadora. Sus características varían según el número de computadoras conectadas a la red, al tipo de hardware y software que manejan, o a la extensión geográfica que abarcan, entre más lejos están ubicadas las computadoras se usa un tipo de configuración distinta.

Las computadoras que forman parte de una red pueden estar constituidas por distinto hardware (procesadores RISC, CISC, computadoras MAC, computadoras SUN, PCs, etc.) y operar bajo distintos sistemas operativos (Windows, Amiga, MacOS, Linux, Unix, etc.). Para que se puedan comunicar entre ellos es necesario que exista un camino físico (cables, ruteadores, gateways, modems, satélites, etc.) y que empleen el mismo protocolo de comunicaciones (TCP/IP, NETBEUI, IPX/SPX, etc.).

Los sistemas operativos de red intentan dar la sensación de que los recursos remotos a los que accede el usuario son locales a la computadora desde la cual se está trabajando. Por ejemplo, un usuario puede estar consultando la información de una base de datos. El usuario en ningún momento tiene conocimiento de si la información a la cual está accediendo se encuentra en su propia computadora o en otra distinta dentro de su red local o en cualquier otra parte del mundo.

Se listan a continuación los tipos más importantes de redes de computadoras:

Red de Área Local (Local Area Network, LAN)

Es un sistema de comunicación entre computadoras, que permite compartir información y recursos, con la característica de que la distancia entre las computadoras debe ser pequeña. La topología o la forma de conexión de la red, depende de algunos aspectos como la distancia entre las computadoras y el medio de comunicación entre ellas ya que éste determina, la velocidad del sistema.

Red de Área Extendida (Wide Area Network, WAN)

Es un sistema de comunicación entre computadoras, que al igual que la red de área local, permite compartir información y recursos, con la característica de que la distancia entre las computadoras es amplia (de un país a otro, de una ciudad a otra, de un continente a otro). Son comúnmente dos o más redes de área local interconectadas, generalmente a través de una amplia zona geográfica.

Algunas redes de área extendida están conectadas mediante líneas rentadas a la compañía telefónica (destinadas para este propósito), soportes de fibra óptica y, otras por medio de sus propios enlaces terrestres y aéreos de satélite.
Tal como se ilustra en la figura, una red de área extendida podría ser la red constituida en una universidad en la que se han conectado las redes de área local existentes en cada uno de los distintos departamentos o facultades.

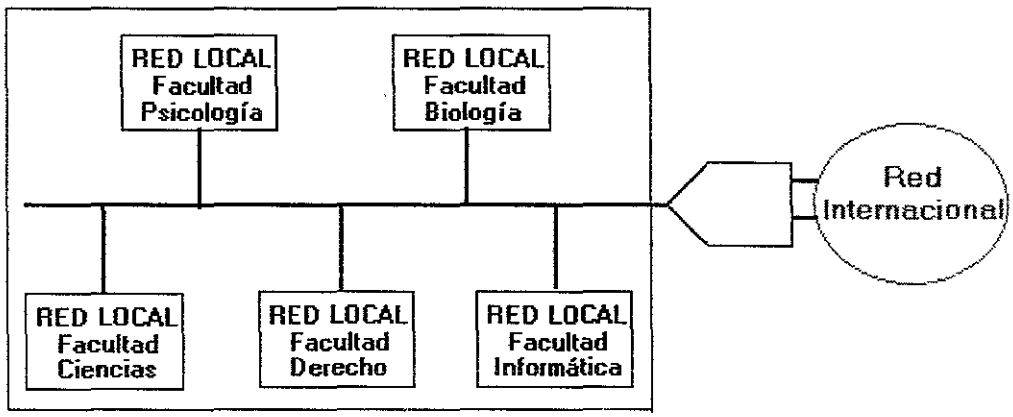


Figura 1.1 Red WAN

Red metropolitana (Metropolitan Area Network, MAN)

Es una red que conecta redes de área extendida en una determinada área geográfica. Son normalmente redes de fibra óptica de gran velocidad que conectan segmentos de red local de una área especifica, como un campus, un complejo industrial o una ciudad. Estas redes están interconectadas a otras redes de nivel superior con enlaces T1 de líneas telefónicas (o vía satélite), capaces de transmitir hasta 1.54 Megabytes por segundo.

Red Columna Vertebral (Backbone Network)

Es una red de alto rendimiento formada por líneas telefónicas especiales de alta velocidad (enlaces T3 que pueden transmitir 4.5 Megabytes por segundo), cables de fibra óptica y enlaces vía satélite. A una red columna vertebral se conectan otras redes de menor rendimiento encargadas de transmitir datos entre computadoras centrales, locales o otras redes de tránsito.

Red Internacional (INTERNETworking)

Son un conjunto de redes independientes (de área local y área extensa) que se encuentran conectadas entre sí, permitiendo el intercambio de datos y constituyendo por lo tanto una red mundial que resulta el medio idóneo para el intercambio de información, mercadotecnia, distribución de datos de todo tipo e interacción personal con otras personas.

Su auge en los últimos años ha permitido un intercambio de información, que nos ha acercado cada vez más hacia lo que se conoce como comunidad mundial. Nosotros a partir de este momento la llamaremos simplemente Internet.

Tipos de computadoras de red

Las computadoras que forman parte de una red pueden desarrollar dos tipos de funciones:

· Servidor.
· Estación de trabajo.

El servidor es aquel o aquellas computadoras que van a compartir sus recursos hardware y software con los demás equipos de la red. Es empleado tanto por su potencia de cálculo, como por la información que gestiona, y los recursos que comparte. Las computadoras que toman el papel de estaciones de trabajo aprovechan o tienen a su disposición los recursos que ofrece la red, así como los servicios que proporcionan los servidores a los cuales pueden acceder. También se puede pensar en computadoras híbridas, que hacen a la vez de servidores y de estaciones de trabajo. Existen dos tipos de servidores:

· Servidores dedicados: son aquellas computadoras que están exclusivamente a disposición de la red.

· Servidores no dedicados: además de tomar el papel de servidores también pueden utilizarse como estaciones de trabajo.

El diseño de la infraestructura de una red de computadoras es una de las labores más importantes que debe llevar a cabo el ingeniero que la esté montando para una empresa. La red más pequeña puede constar de un único servidor y unas cuantas estaciones de trabajo conectadas mediante tarjetas de red y cable coaxial, redes más grandes pueden estar constituidas por varios servidores y una gran cantidad de estaciones de trabajo con sedes distribuidas alrededor de todo el mundo, lo cual conlleva el empleo de redes de comunicación (la red telefónica, Internet, etc.) para interconectar entre sí los equipos de todas las sedes.

1.2 Topologías básicas

La topología de una red hace referencia a cómo se distribuye u organiza el conjunto de computadoras dentro de la red. Cada una ofrece ventajas y desventajas quedando en nosotros, dependiendo de nuestras necesidades, escoger la que mejor nos convenga.

Básicamente existen 6 topologías de red:

- Punto a punto
- Multipunto
- Estrella (Star)
- Canal (Bus)
- Anillo (Ring)
- Árbol

RED PUNTO A PUNTO

Se considera que es la más sencilla por tener una sola computadora, una línea de comunicación (a través del sistema telefónico) y una terminal. Esta terminal puede ser de lote distante, es decir, que se utiliza para introducir trabajos y datos de una computadora desde un lugar remoto para realizar procesamientos por lotes más tarde.

En este tipo de red la computadora no necesita ser grande; sin embargo normalmente tiene una computadora grande como sistema anfitrión.

Punto a Punto

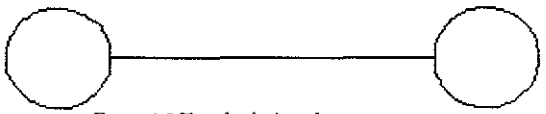


Figura 1.2 Topología de red punto a punto

RED MULTIPUNTO

Aquí en lugar de haber una sola estación remota hay varias terminales distantes, que se pueden conectar por líneas de comunicación independientes a la computadora o pueden (enviar datos de varios dispositivos a la vez en una misma línea) multiplexarse. Esta red se puede considerar como la base de los otros tipos de redes teniendo en cuenta que en la multipunto sólo hay un nodo inteligente y en las demás la mayoría de los puntos del sistema lo son, sin que se necesite del sistema central.

Multipunto

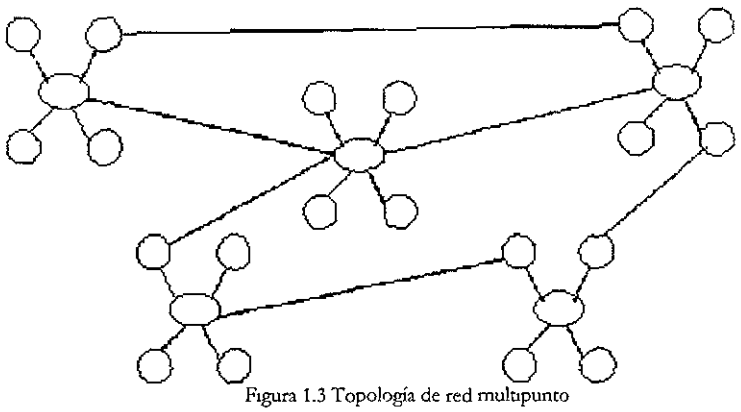


Figura 1.3 Topología de red multipunto

RED ESTRELLA

Conectar un conjunto de computadoras en estrella es uno de los sistemas más antiguos. Equivale a tener una computadora central (el servidor de archivos o Server), encargada de controlar la información de toda la red. Dicha información abarca desde los mensajes entre usuarios, datos almacenados en un archivo en particular, manipulación de archivos, etc.
Para poder instalar este tipo de red, cada una de las computadoras utilizadas como estaciones de trabajo necesitan de una tarjeta de conexión para lograr la interfaz con la computadora central.

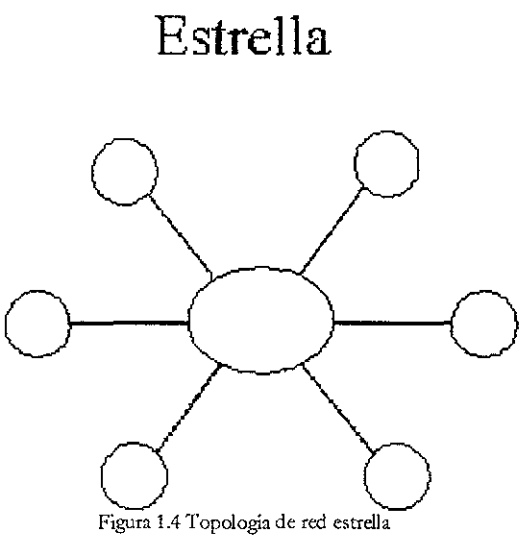


Figura 1.4 Topología de red estrella

RED EN CANAL

Permite conectar a todas las computadoras de la red en una sola línea compartiendo el mismo canal de datos (bus), de ahí su nombre. A fin de poder identificar hacia cual de las computadoras de toda la red se está dirigiendo, se añade un sufijo al paquete de información, éste contiene la dirección de la computadora que debe recibir la información en particular.
Cada una de las computadoras revisa el mensaje y compara la dirección de la terminal de recepción, en caso de no ser igual a la propia, se rechaza y en caso de ser igual la dirección, se acepta el mensaje.

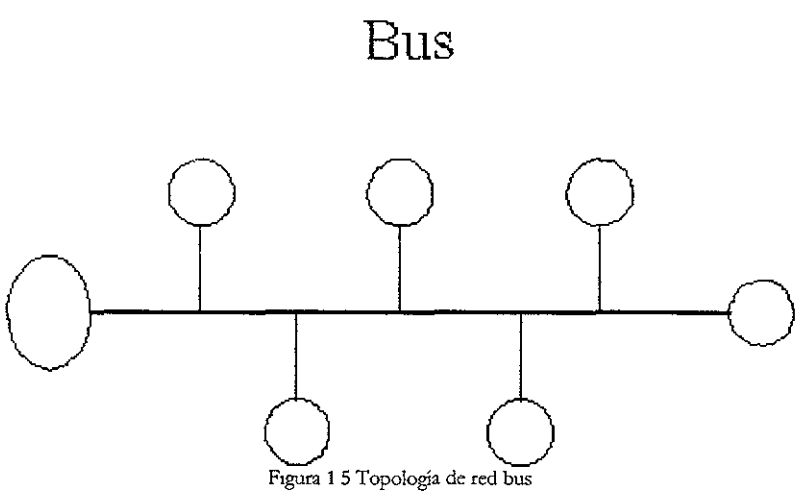


Figura 1 5 Topología de red bus

RED ANILLO

Es la más difundida actualmente, consiste en unir una serie de computadoras en un circuito cerrado formando un anillo por donde circula la información en una sola dirección, factor que permite tener un control de recepción de mensajes.
La forma interna de comunicación, de una computadora a otra, es similar a la del canal de datos (Bus), sólo que en este caso se le añade la dirección de la computadora que envía el mensaje para que la terminal receptora pueda contestar a la terminal emisora.

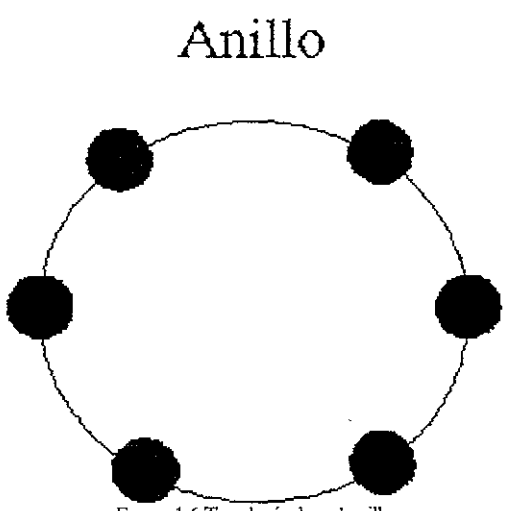


Figura 1 6 Topología de red anillo

RED ÁRBOL

Es una red completamente distribuida. Una red jerárquica representa una red en donde las computadoras alimentan de información a otras computadoras que a su vez alimentan a otras. Las computadoras que se utilizan como dispositivos remotos pueden tener recursos de procesamiento independiente y recurren a los recursos en niveles superiores e inferiores conforme se requiere información u otros recursos.

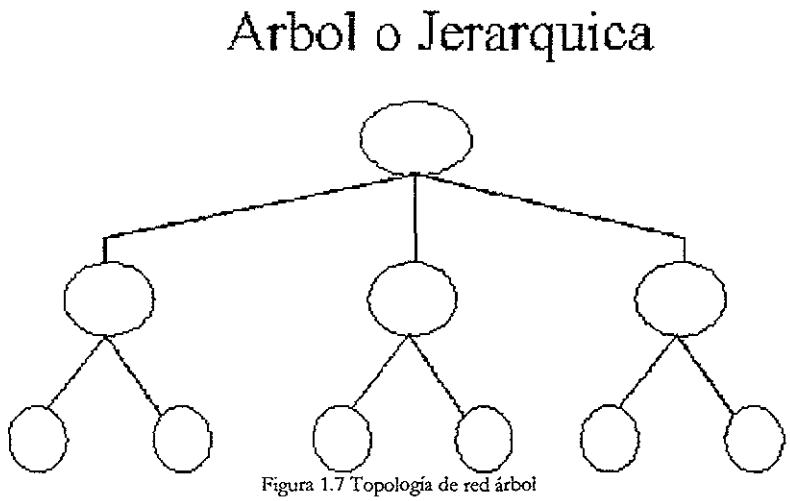


Figura 1.7 Topología de red árbol

Los seis tipos de conexión mencionados son los principales para comunicar una serie de computadoras de la misma familia.

Formas de conexión

Una vez definida la forma de instalación de la red, es posible configurar cada una de las estaciones de trabajo de 4 formas distintas, dependiendo de la configuración, la computadora podrá enviar o recibir mensajes y usar o compartir periféricos.

Redirector (RDR) Es la forma más simple de conexión de una computadora en red, esta terminal o estación de trabajo sólo podrá enviar mensajes a las diferentes terminales y tendrá acceso a los periféricos de la red.

Receptor (RCV) Esta configuración incluye las capacidades del redirector dentro de las capacidades del receptor. El receptor está capacitado para recibir y enviar mensajes y utilizar los periféricos de la red.

Mensajero (MSG) Esta configuración incluye las capacidades del redirector y del receptor. El mensajero está capacitado para recibir y enviar mensajes, utilizar los periféricos de la red, guardar mensajes recibidos en esa terminal y recibir o transmitir mensajes a otras redes o nodos.

Servidor (SRV) El servidor de la red es el que configura toda la red en sí, permitiendo definir los periféricos a compartir, las prioridades de las distintas terminales, los volúmenes privados y públicos en las distintas computadoras, y otros parámetros importantes.

1.3 Protocolos de comunicación

Los protocolos de comunicaciones definen las reglas para la transmisión y recepción de la información entre los nodos de la red, de modo que para que dos nodos se puedan comunicar entre sí es necesario que ambos empleen la misma configuración de protocolos. Un interfaz, sin embargo, es el encargado de la conexión física entre los equipos, definiendo las normas para las características eléctricas y mecánicas de la conexión.

El modelo OSI

El modelo OSI (Open System Interconection), define como los fabricantes pueden crear productos que funcionen con los productos de otros vendedores sin la necesidad de controladores especiales o equipamientos opcionales. Cada nivel de la jerarquía de protocolos OSI tiene una función especifica y define un nivel de comunicaciones entre sistemas.
El modelo OSI es utilizado por prácticamente la totalidad de las redes del mundo. Este modelo fue creado por el ISO (Organización Internacional de Normalización), y consiste en siete niveles o capas donde cada una de ellas define las funciones que deben proporcionar los protocolos con el propósito de intercambiar información entre varios sistemas. Esta clasificación permite que cada protocolo se desarrolle con una finalidad determinada, lo cual simplifica el proceso de desarrollo e implementación. Cada nivel depende de los que están por debajo de él, y a su vez proporciona alguna funcionalidad a los niveles superiores. Los siete niveles del modelo OSI son los siguientes:

Aplicación	El nivel de aplicación es el destino final de los datos donde se proporcionan los servicios al usuario.
Presentación	Se convierten e interpretan los datos que se utilizarán en el nivel de aplicación
Sesión	Encargado de ciertos aspectos de la comunicación como el control de los tiempos.
Transporte	Transporta la información de una manera fiable para que llegue correctamente a su destino.
Red	Nivel encargado de encaminar los datos hacia su destino eligiendo la ruta destino y eligiendo la ruta más efectiva.
Enlace	Enlace de datos. Controla el flujo de los mismos, la sincronización y los errores que puedan producirse.
Físico	Se encarga de los aspectos físicos de la conexión, tales como el medio de transmisión o el hardware.

Entre los protocolos propios de una red de área local podemos distinguir dos principales grupos. Por un lado están los protocolos de los niveles físico y de enlace, niveles 1 y 2 del modelo OSI, que definen las funciones asociadas con el uso del medio de transmisión: envío de los datos a nivel de bits y trama, y el modo de acceso de los nodos al medio. Estos protocolos vienen unívocamente determinados por el tipo de red (Ethernet, Token Ring, etc.). El segundo grupo de protocolos se refiere a aquellos que realizan las funciones de los niveles de red y transporte, niveles 3 y 4 de OSI, que se encargan básicamente del encaminamiento de la información y garantizar una comunicación extremo a extremo libre de errores.

Estos protocolos transmiten la información a través de la red en pequeños segmentos llamados paquetes. Si una computadora quiere transmitir un archivo grande a otro, el archivo es dividido en paquetes en el origen y vueltos a ensamblar en la computadora destino. Cada protocolo define su propio formato de los paquetes en el que se especifica el origen, destino, longitud y tipo del paquete, así como la información redundante para el control de errores.

Los protocolos de los niveles 1 y 2 dependen del tipo de red, mientras que para los niveles 3 y 4 hay diferentes alternativas, siendo TCP/IP la configuración más extendida. Lo que la convierte en un estándar de facto. Por su parte, los protocolos OSI representan una solución técnica muy potente y flexible, pero que actualmente esta escasamente implantada en entornos de red de área local.

Los protocolos TCP/IP

TCP/IP es el protocolo común utilizado por todas las computadoras conectadas a Internet, de manera que éstas puedan comunicarse entre sí. Hay que tener en cuenta que en Internet se encuentran conectadas computadoras de clases muy diferentes y con hardware y software incompatibles en muchos casos, además de todos los medios y formas posibles de conexión. Aquí se encuentra una de las grandes ventajas del TCP/IP, pues este protocolo se encargará de que la comunicación entre todos sea posible. TCP/IP es compatible con cualquier sistema operativo y con cualquier tipo de hardware.

TCP/IP no es un único protocolo, sino que es un conjunto de protocolos que cubren los distintos niveles del modelo OSI. Los dos protocolos más importantes son el TCP (Transmisión Control Protocol) y el IP (Internet Protocol), que son los que dan nombre al conjunto. En Internet se diferencian cuatro niveles o capas en las que se agrupan los protocolos, y que se relacionan con los niveles OSI de la siguiente manera:

Aplicación: Se corresponde con los niveles OSI de aplicación y presentación. Aquí se incluyen protocolos destinados a proporcionar servicios, tales como correo electrónico(SMTP), transferencia de archivos(FTP), conexión remota(TELNET) y el de transferencia de hipertexto HTTP(Hypertext Transfer Protocol).

Transporte: Coincide con los niveles de transporte y sesión del modelo OSI. Los protocolos de este nivel, tales como TCP y UDP, se encargan de manejar los datos y proporcionar la fiabilidad necesaria en el transporte de los mismos.

Internet: Es el nivel de red del modelo OSI. Incluye al protocolo IP, que se encarga de enviar los paquetes de información a sus destinos correspondientes. Es utilizado con esta finalidad por los protocolos del nivel de transporte.

Enlace: Los niveles OSI correspondientes son el de enlace y el nivel físico. Los protocolos que pertenecen a este nivel son los encargados de la transmisión a través del medio físico al que se encuentra conectada cada máquina(*host*), como puede ser una línea punto a punto o una red Ethernet.

La figura 1.8 muestra la correlación existente entre el modelo teórico de capas o niveles de red propuestos por la Organización de Estándares Internacional (OSI) y el modelo empleado por las redes TCP/IP.

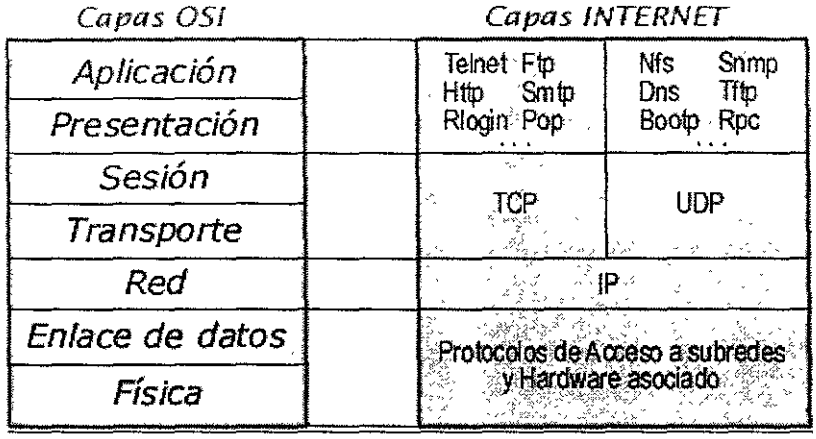


Figura 1.8 Correlación Capas OSI vs Capas INTERNET

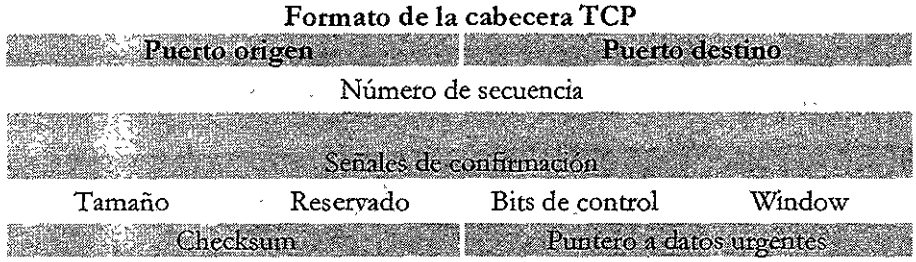
El TCP/IP necesita funcionar sobre algún tipo de red o de medio físico que proporcione sus propios protocolos para el nivel de enlace de Internet. Por este motivo hay que tener en cuenta que los protocolos utilizados en este nivel pueden ser muy diversos y no forman parte del conjunto TCP/IP. Sin embargo, esto no debe ser problemático puesto que una de las funciones y ventajas principales del TCP/IP es proporcionar una abstracción del medio de forma que sea posible el intercambio de información entre medios diferentes y tecnologías que inicialmente son incompatibles.
Para transmitir información a través de TCP/IP, ésta debe ser dividida en unidades de menor tamaño. Esto proporciona grandes ventajas en el manejo de los datos que se transfieren y, por otro lado, esto es algo común en cualquier protocolo de comunicaciones. En TCP/IP cada una de estas unidades de información recibe el nombre de "datagrama" (datagram), y son conjuntos de datos que se envían como mensajes independientes.

TCP (Transmission Control Protocol).

El protocolo de control de transmisión (TCP) pertenece al nivel de transporte, siendo el encargado de dividir el mensaje original en datagramas de menor tamaño, y por lo tanto, mucho más manejables. Los datagramas serán dirigidos a través del protocolo IP de forma individual. El protocolo TCP se encarga además de añadir cierta información necesaria a cada uno de los datagramas. Esta información se añade al inicio de los datos que componen el datagrama en forma de cabecera.

La cabecera de un datagrama contiene al menos 160 bits que se encuentran repartidos en varios campos con diferente significado. Cuando la información se divide en datagramas para ser enviados, el orden en que éstos lleguen a su destino no tiene que ser el correcto. Cada uno de ellos puede llegar en cualquier momento y con cualquier orden, e incluso puede que algunos no lleguen a su destino o lleguen con información errónea. Para evitar todos estos problemas el TCP numera los datagramas antes de ser enviados, de manera que sea posible volver a unirlos en el orden adecuado. Esto permite también solicitar de nuevo el envío de los datagramas individuales que no hayan llegado o que contengan errores, sin que sea necesario volver a enviar el mensaje completo.

Formato de la cabecera TCP



A continuación de la cabecera puede existir información opcional. En cualquier caso el tamaño de la cabecera debe ser múltiplo de 32 bits, por lo que puede ser necesario añadir un campo de tamaño variable y que contenga ceros al final para conseguir este objetivo cuando se incluyen algunas opciones. El campo de tamaño contiene la longitud total de la cabecera TCP expresada en el número de palabras de 32 bits que ocupa. Esto permite determinar el lugar donde comienzan los datos.

Dos campos incluidos en la cabecera y que son de especial importancia son los números de puerto de origen y puerto de destino. Los puertos proporcionan una manera de distinguir entre las distintas transferencias, ya que una misma computadora puede estar utilizando varios servicios o transferencias simultáneamente, e incluso puede que por medio de usuarios distintos. El puerto de origen contendrá un número cualquiera que sirva para realizar esta distinción. Además, el programa cliente que realiza la petición también debe conocer el número de puerto en el que se encuentra el servidor adecuado. Mientras que el programa del usuario utiliza números prácticamente aleatorios, el servidor deber tener asignado un número estándar para que pueda ser utilizado por el cliente. (Por ejemplo, en el caso de la transferencia de archivos FTP el número oficial es el 21). Cuando es el servidor el que envía los datos, los números de puertos de origen y destino se intercambian.

En la transmisión de datos a través del protocolo TCP la fiabilidad es un factor muy importante. Para poder detectar los errores y pérdida de información en los datagramas, es necesario que el cliente envíe de nuevo al servidor unas señales de confirmación una vez que se ha recibido y comprobado la información satisfactoriamente. Estas señales se incluyen en el campo apropiado de la cabecera del datagrama (Acknowledgment Number), que tiene un tamaño de 32 bit. Si el servidor no obtiene la señal de confirmación adecuada transcurrido un período de tiempo razonable, el datagrama completo se volverá a enviar. Por razones de eficiencia los datagramas se envían continuamente sin esperar la confirmación, haciéndose necesaria la numeración de los mismos para que puedan ser ensamblados en el orden correcto.

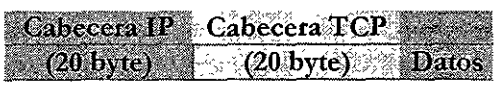
También puede ocurrir que la información del datagrama llegue con errores a su destino. Para poder detectar cuando sucede esto se incluye en la cabecera un campo de 16 bit, el cual contiene un valor calculado a partir de la información del datagrama completo (checksum). En el otro extremo el receptor vuelve a calcular este valor, comprobando que es el mismo que el suministrado en la cabecera. Si el valor es distinto significaría que el datagrama es incorrecto, ya que en la cabecera o en la parte de datos del mismo hay algún error.

La forma en que TCP numera los datagramas es contando los bytes de datos que contiene cada uno de ellos y añadiendo esta información al campo correspondiente de la cabecera del datagrama siguiente. De esta manera el primero empezará por cero, el segundo contendrá un número que será igual al tamaño en bytes de la parte de datos del datagrama anterior, el tercero con la suma de los dos anteriores, y así sucesivamente. Por ejemplo, para un tamaño fijo de 500 bytes de datos en cada datagrama, la numeración sería la siguiente: 0 para el primero, 500 para el segundo, 1000 para el tercero, etc.

Existe otro factor más a tener en cuenta durante la transmisión de información, y es la potencia y velocidad con que cada una de las computadoras puede procesar los datos que le son enviados. Si esto no se tuviera en cuenta, la computadora de más potencia podría enviar la información demasiado rápido al receptor, de manera que éste no pueda procesarla. Este inconveniente se soluciona mediante un campo de 16 bit (Window) en la cabecera TCP, en el cual se introduce un valor indicando la cantidad de información que el receptor está preparado para procesar. Si el valor llega a cero será necesario que el emisor se detenga. A medida que la información es procesada este valor aumenta indicando disponibilidad para continuar la recepción de datos.

IP (Internet Protocol) versión 4.

El IP es un protocolo que pertenece al nivel de red, por lo tanto, es utilizado por los protocolos del nivel de transporte como TCP para encaminar los datos hacia su destino. IP tiene únicamente la misión de encaminar el datagrama, sin comprobar la integridad de la información que contiene. Para ello se utiliza una nueva cabecera que se antepone al datagrama que se está tratando. Suponiendo que el protocolo TCP ha sido el encargado de manejar el datagrama antes de pasarlo al IP, la estructura del mensaje una vez tratado quedaría así:



Protocolos alternativos a TCP

TCP es el protocolo más utilizado para el nivel de transporte en Internet, pero además de éste existen otros protocolos que pueden ser más convenientes en determinadas ocasiones. Tal es el caso de UDP y ICMP.

UDP (User Datagram Protocol)

El protocolo de datagramas de usuario (UDP) puede ser la alternativa al TCP en algunos casos en los que no sea necesario el gran nivel de complejidad proporcionado por el TCP. Puesto que UDP no admite numeración de los datagramas, este protocolo se utiliza principalmente cuando el orden en que se reciben los mismos no es un factor fundamental, o también cuando se quiere enviar información de poco tamaño que cabe en un único datagrama.

Cuando se utiliza UDP la garantía de que un paquete llegue a su destino es mucho menor que con TCP debido a que no se utilizan las señales de confirmación. Por todas estas características la cabecera del UDP es bastante menor en tamaño que la de TCP. Esta simplificación resulta en una mayor eficiencia en determinadas ocasiones.

ICMP (Internet Control Message Protocol)

El protocolo de mensajes de control de Internet (ICMP) es de características similares al UDP, pero con un formato aún más simple. Su utilidad no está en el transporte de datos "de usuario", sino en los mensajes de error y de control necesarios para los sistemas de la red.

1.4 Sockets

En el año 1980 la agencia ARPA (Advanced Research Projects Agency, Agencia para la Investigación de Proyectos Avanzados) fundó un grupo de investigación en la Universidad de Berkeley (California) para llevar a cabo la implementación del software TCP/IP dentro del propio núcleo del sistema operativo UNIX. Como parte del proyecto, los diseñadores crearon una *interfaz* para permitir que las aplicaciones pudieran acceder a los servicios que brinda el software TCP/IP. Decidieron hacer uso de las llamadas al sistema UNIX existentes siempre que fuera posible y añadir nuevas llamadas para dar soporte a las funciones TCP/IP que no se podían implementar adecuadamente con el conjunto de llamadas disponibles. Al fruto de este proyecto se le conoce como la *interfaz socket*, y el sistema operativo resultante se dio a conocer como *UNIX Berckeley* o *BSD UNIX*. La interfaz socket desarrollada en Berkeley ha sido tan ampliamente aceptada por los grandes distribuidores de computadoras que se ha convertido en un *standard de facto*.

Debido a que los diseñadores de Berkeley tenían la intención de dar soporte a una gran variedad de protocolos de comunicación definieron funciones que dan soporte a las comunicaciones de red en general, y emplearon parámetros para hacer que las comunicaciones TCP/IP sean un caso especial. De hecho, a lo largo del diseño, consiguieron proporcionar generalidad más allá del TCP/IP. Permitieron el empleo de múltiples familias de protocolos, con todos los protocolos TCP/IP representados como una familia única (*PF_INET*).

En resumen, la Interfaz Socket de Berkeley proporciona funciones generalizadas que dan soporte a las comunicaciones en red empleando para ello muchos de los protocolos disponibles hoy en día. Las llamadas socket hacen referencia a todos los protocolos TCP/IP como una única familia. Las llamadas permiten al programador especificar el tipo de servicio requerido, en vez del nombre de un protocolo específico.

La abstracción Socket

La interfaz socket añade una nueva abstracción para las comunicaciones en red, el *socket*. Al igual que para los archivos, cada socket activo es identificado por un entero llamado el *descriptor de socket*. UNIX aloja los descriptores de socket en la misma tabla que los descriptores de archivo. Por lo tanto, una aplicación no puede tener un descriptor de socket y un descriptor de archivo con el mismo valor.

Los sistemas de tipo UNIX, definen socket como un punto virtual de comunicación que permiten intercambiar datos con otro proceso de usuario. Este último proceso puede estar situado en la misma máquina o en otra conectada a la red. Al igual que los números de puerto en los protocolos de transporte permiten la utilización del canal de comunicación físico por varios procesos sin entorpecerse, los sockets realizan la misma función a nivel de programación.

La idea general que subyace bajo el concepto de socket es que una única llamada al sistema es suficiente para crear cualquier socket. Una vez el socket ha sido creado, una aplicación debe realizar llamadas al sistema adicionales para especificar el uso que se desea hacer del mismo.

Empleo de Sockets

Una vez el socket ha sido creado, puede ser utilizado para esperar conexiones o para iniciar una conexión. Al socket empleado por un servidor para esperar conexiones entrantes se le conoce como *socket pasivo*, mientras que al socket empleado por un cliente para iniciar una conexión se le conoce como *socket activo*. La única diferencia entre los sockets activos y pasivos radica en como son utilizados por la aplicación que los ha creado; ya que éstos se crean del mismo modo.

El programador ve el socket como un lugar en el que puede escribir y del que puede leer, como si de un archivo se tratase (salvando lógicamente las diferencias de tratamiento). Por supuesto, un socket deberá estar asociado a un número de puerto y a un protocolo específico del nivel de transporte, y el programador lo utilizará mediante un descriptor y realizando llamadas al sistema.

Especificación de la dirección de red

Cuando se crea un socket, no se define información detallada acerca de cómo será utilizado. En concreto, el socket carece información acerca del puerto de protocolo y la dirección IP tanto de la máquina local como de la máquina remota. Antes de que una aplicación pueda utilizar un socket, debe especificar una o ambas direcciones.

Los protocolos TCP/IP definen un punto final de comunicación o *dirección de red*, el cual consta de una dirección IP y un número de puerto de protocolo. Otras familias de protocolos definen sus propios mecanismos para formar las direcciones de red. Debido a que la abstracción socket acomoda múltiples familias de protocolos, no especifica como definir las direcciones de red ni tampoco un formato de dirección para un protocolo en particular. En vez de ello, permite que cada familia de protocolos pueda especificar el formato de sus direcciones de red.

Para dar libertad a las familias de protocolos en la representación de sus direcciones de red, la abstracción socket define una *familia de direcciones* para cada tipo de direcciones. Una familia de protocolos puede emplear una o más familias de direcciones para definir su representación de direcciones de red. Todos los protocolos TCP/IP emplean una única representación de direcciones, cuya familia de direcciones es denotada por la constante *AF_INET*.

Tipos de sockets

El tipo de socket describe la forma en la que se transfiere información a través de ese socket.

Sockets Stream (TCP, Transport Control Protocol)

Son un servicio orientado a conexión donde los datos se transfieren sin encuadrarlos en registros o bloques. Si se rompe la conexión entre los procesos, éstos serán informados.

El protocolo de comunicaciones con streams es un protocolo orientado a conexión, ya que para establecer una comunicación utilizando el protocolo TCP, hay que establecer en primer lugar una conexión entre un par de sockets. Mientras uno de los sockets atiende peticiones de conexión (servidor), el otro solicita una conexión (cliente). Una vez que los dos sockets estén conectados, se pueden utilizar para transmitir datos en ambas direcciones.

Los sockets stream permiten a los procesos comunicarse a través de TCP. Una vez establecidas las conexiones, los datos se pueden leer y escribir a/desde los sockets como un flujo(stream) de bytes. Algunas aplicaciones de servicios TCP son:

- FTP, File Tranfer Protocol
- SMTP, Simple Mail Transfer Protocol
- Telnet, servicio de conexión de terminal remoto

Sockets Datagrama (UDP, User Datagram Protocol)

Son un servicio de transporte sin conexión. Son más eficientes que TCP, pero no está garantizada la fiabilidad. Los datos se envían y reciben en paquetes, cuya entrega no está garantizada. Los paquetes pueden ser duplicados, perdidos o llegar en un orden diferente al que se envió.

El protocolo de comunicaciones con datagramas es un protocolo sin conexión, es decir, cada vez que se envíen datagramas es necesario enviar el descriptor del socket local y la dirección del socket que debe recibir el datagrama. Como se puede ver, hay que enviar datos adicionales cada vez que se realice una comunicación.

Los sockets datagrama permiten a los procesos utilizar el protocolo UDP para comunicarse a y desde esos sockets por medio de bloques. UDP es un protocolo no fiable y la entrega de los paquetes no está garantizada. Servicios UDP son:

- SNMP, Simple Network Management Protocol
- TFTP, Trivial File Transfer Protocol (versión de FTP sin conexión)
- VMTP, Versatile Message Transaction Protocol (servicio fiable de entrega punto a punto de datagramas independiente de TCP)

Sockets Raw

Son sockets que dan acceso directo a la capa de software de red subyacente o a protocolos de más bajo nivel. Se utilizan sobre todo para la depuración del código de los protocolos.
Los sockets raw proporcionan acceso al Internet Control Message Protocol, ICMP, y se utiliza para comunicarse entre varias entidades IP.

Diferencias entre Sockets Stream y Datagrama

Ahora se nos presenta un problema, ¿qué protocolo, o tipo de sockets, debemos usar - UDP o TCP? La decisión depende de la aplicación cliente/servidor que estemos escribiendo.

En UDP, cada vez que se envía un datagrama, hay que enviar también el descriptor del socket local y la dirección del socket que va a recibir el datagrama, luego éstos son más grandes que los TCP. Como el protocolo TCP está orientado a conexión, tenemos que establecer esta conexión entre los dos sockets antes de nada, lo que implica un cierto tiempo empleado en el establecimiento de la conexión, que no existe en UDP.

En UDP hay un límite de tamaño de los datagramas, establecido en 64 kilobytes, que se pueden enviar a una localización determinada, mientras que TCP no tiene límite; una vez que se ha establecido la conexión, el par de sockets funciona como los streams: todos los datos se leen inmediatamente, en el mismo orden en que se van recibiendo.

UDP es un protocolo desordenado, no garantiza que los datagramas que se hayan enviado sean recibidos en el mismo orden por el socket de recepción. Al contrario, TCP es un protocolo ordenado, garantiza que todos los paquetes que se envíen serán recibidos en el socket destino en el mismo orden en que se han enviado.

Los datagramas son bloques de información del tipo lanzar y olvidar. Para la mayoría de los programas que utilicen la red, el usar un flujo TCP en vez de un datagrama UDP es más sencillo y hay menos posibilidades de tener problemas. Sin embargo, cuando se requiere un rendimiento óptimo, y está justificado el tiempo adicional que supone realizar la verificación de los datos, los datagramas son un mecanismo realmente útil.

En resumen, TCP parece más indicado para la implementación de servicios de red como un control remoto (rlogin, telnet) y transmisión de archivos (ftp, http); que necesitan transmitir datos de longitud indefinida. UDP es menos complejo y tiene una menor sobrecarga sobre la conexión; esto hace que sea el indicado en la implementación de aplicaciones cliente/servidor en sistemas distribuidos montados sobre redes de área local.

Dominios de comunicaciones

El mecanismo de sockets está diseñado para ser todo lo genérico posible. El socket por sí mismo no contiene información suficiente para describir la comunicación entre procesos. Los sockets operan dentro de dominios de comunicación, entre ellos se define si los dos procesos que se comunican se encuentran en el mismo sistema o en sistemas diferentes y cómo pueden ser direccionados.

Dominio Unix

Bajo Unix, hay dos dominios, uno para comunicaciones internas al sistema y otro para comunicaciones entre sistemas.

Las comunicaciones intrasistema (entre dos procesos en el mismo sistema) ocurren (en una máquina Unix) en el dominio Unix. Se permiten tanto los sockets stream como los datagrama. En el dominio Unix no se permiten sockets de tipo Raw.

Dominio Internet

Las comunicaciones intersistemas proporcionan acceso a TCP, ejecutando sobre IP (Internet Protocol). De la misma forma que el dominio Unix, el dominio Internet permite tanto sockets stream como datagrama, pero además permite sockets de tipo Raw.

Principales llamadas al sistema para Sockets

Las llamadas al sistema para la gestión de sockets pueden dividirse en dos categorías: las llamadas primarias, necesarias para la comunicación con sockets y las llamadas adicionales que facilitan la labor del programador.

Como ideas fundamentales establecemos que un socket es un canal que puede abrirse (**socket**) con una dirección de destino (**bind**); mantenerse listos y escuchando en espera de conexiones nuevas (**listen y accept**), enviarse datos (**write**), y recibirse (**read**), y por último cerrarse el canal (**close**).

El siguiente gráfico muestra el esquema general de interacción Cliente/Servidor con sockets de conexión:

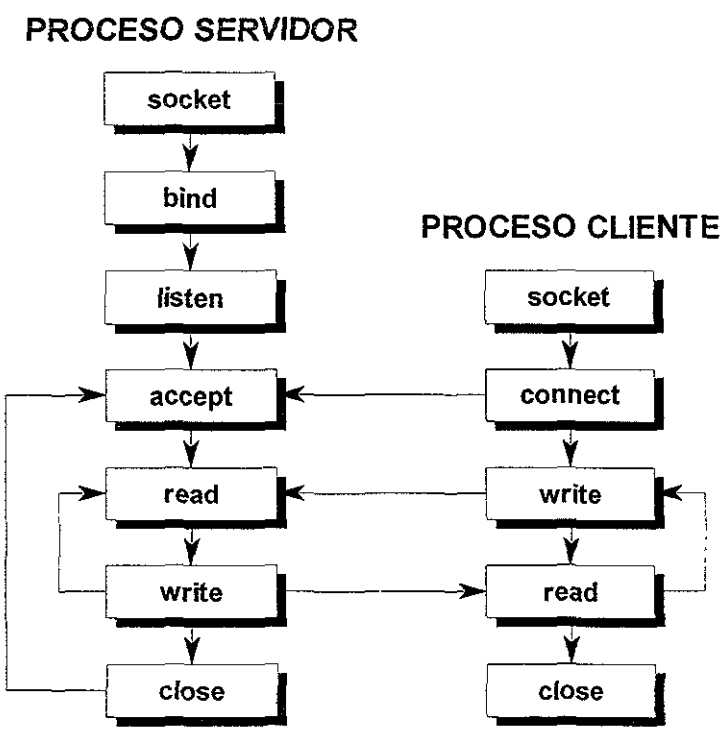


Figura 1.9 Interacción Cliente/Servidor con sockets de conexión

2
INTERNET

Hay varios puntos de vista para definir Internet. Por un lado se puede interpretar como una mega red, esto es, una red de redes de computadoras, considerando las líneas de comunicaciones, el hardware de los equipos y los protocolos de comunicaciones. Sin embargo, otro enfoque, que parece más adecuado es pensar en las redes como el medio a través del cual se envía y acumula la información. Desde este punto de vista Internet puede ser interpretada como la información y los servicios que circulan por esta red. En pocas palabras un sistema distribuido de información.

En este capítulo se describirá la evolución que experimentado la red Internet en los últimos años. Su esquema de operación. Los servicios que ofrece.

2.1 Historia

Internet nace, a fines de la década de los 60, como un proyecto del Departamento de Defensa de los Estados Unidos para diseñar un sistema de comunicaciones distribuido. El propósito de esta red era posibilitar las comunicaciones entre las autoridades en caso de un ataque nuclear. Las redes de comunicación de ese tiempo estaban diseñadas de modo que cada nodo de la red dependía del nodo anterior. Si se destruía un nodo toda la red sería inutilizada.

A partir de esto, se diseñó una pequeña red descentralizada que conectaba computadoras en cuatro universidades en Estados Unidos, estructurada conceptualmente como lo que hoy entendemos por red.

Esta red se llamó ARPANET y uno de sus objetivos fue compartir los recursos de las super computadoras entre los investigadores universitarios. Pronto esta red se convierte en una oficina postal de alta velocidad, al convertirse el correo electrónico en el medio de comunicación preferido en estas universidades, con el objeto de compartir y colaborar en proyectos de investigación.

En 1971 ARPANET se extiende a 23 sitios (*hosts* o máquinas) conectando a los centros de investigación de universidades y de gobierno en los Estados Unidos. En 1973 ARPANET se internacionaliza al conectar dos centros en Inglaterra y en Noruega.

Alrededor de 1975 DARPA declara el proyecto como un éxito y le pasa su administración al Departamento de Comunicaciones de Defensa de Norteamérica. Hacia 1981, ARPANET tenía 213 nodos y creciendo a un ritmo de un nodo cada 20 días. A mediados de los 80 se crea el protocolo TCP/IP (Transfer Control Protocol/Internet Protocol) que permite tener un lenguaje común para todas las computadoras conectadas que en 1983 fueron adoptados por ARPANET.

ARPANET se componía de cientos de computadoras pertenecientes a universidades, centros de investigación militar y algunas compañías, conectadas entre sí. El servicio más popular entonces, era el e-mail (correo electrónico) que permitía una fácil y rápida comunicación entre diferentes personas conectadas a ARPANET. El sistema operativo que más se utilizaba era UNIX y en especial una versión de UNIX desarrollada por la universidad de California en Berkeley llamada BSD UNIX.

Entonces se comienza a visualizar esta colección de redes (networks) como una gran red: Internet***working***, que finalmente se abreviaría Internet. En esta misma época, IBM introdujo las computadoras personales y comenzó la revolución de la computación personal. Al mismo tiempo se introdujeron las computadoras poderosas de tamaño pequeño listas para redes (como Sun) que permitieron que muchas compañías ingresaran a Internet y se comunicaran entre sí.

Para 1985 las redes locales en computadoras personales ya estaban madurando y esto ayudó a completar la idea de Internet. Ya se podía tener redes, sub-redes, y conectar redes de área ancha (Wide Area Networks) con redes locales (Local Area Networks). En 1986 surge "The Supercomputer Centers Program" iniciado por "The National Science Foundation (NSF)", el propósito de este programa era hacer que los recursos de super cómputo puedan llegar a usuarios un poco más comunes. Ellos establecieron 5 centros de super cómputo en diferentes áreas de los Estados Unidos y construyeron una red que los uniera a todos. La NSF basó sus protocolos de comunicación en los protocolos de Internet y se originó lo que se conoció como NSFNET que fue el corazón de Internet hasta 1995. Para entonces se utilizaba el e-mail, ftp, telnet, gopher y otros servicios de Internet.

En 1989 un equipo de investigación en Suiza pertenecientes al CERN (Centro Europeo de Investigación de Partículas), desarrolló una serie de protocolos para transferir hipertexto vía Internet. A principios de 1990, un grupo de personas pertenecientes al National Center for Supercomputing Applications (NCSA) mejoraron esos nuevos protocolos y desarrollaron el NCSA Mosaic, el primer navegador(browser) que hacia el uso de Internet algo fácil. Fue entonces cuando comenzó el boom del World Wide Web que atrajo a miles de personas hacia Internet.

Internet va a cambiar nuestra forma de percibir las cosas. Como se puede ver, la comunidad que se desarrolla en Internet no es nueva, y aunque antes era dominada principalmente por investigadores y universidades, ahora se está adentrando poco a poco en nuestra vida diaria. Personas de diferentes países tienen un medio de comunicación accesible y rápido. Barreras ideológicas se rompen y se mezclan. Cuando antes existía una falta de información, ahora tenemos una sobredosis de la misma: Universidades, bibliotecas, museos, manuales, libros, etc. todos en línea. Internet ha desarrollado una cultura propia, basada en una sociedad virtual compuesta por personas de todas partes del mundo.

La gran rapidez con la que Internet se ha expandido y popularizado en los últimos años ha supuesto una revolución muy importante en el mundo de las comunicaciones, llegando a causar cambios en muchos aspectos de la sociedad.

2.2 ¿Cómo funciona?

Lo que se conoce como Internet es en realidad una red de redes, la interconexión de otras redes independientes de manera que puedan compartir información entre ellas a lo largo de todo el planeta. Para ello es necesario el uso de un protocolo de comunicaciones común. El protocolo que proporciona la compatibilidad necesaria para la comunicación en Internet es el TCP/IP.

La forma general de establecer una comunicación a través de Internet es:

1) Indicar la dirección IP de la máquina con la que queremos conectar y
2) Especificar el número de puerto dentro de esa máquina a través del cual queremos establecer la comunicación.

Transferencia de datos

Como elemento básico es necesario aclarar que, en general, se usa un solo cable para comunicar varias computadoras por lo que esto obliga a compartir esta línea de comunicación.

Compartir un camino de transmisión introduce retrasos y congestiones, al igual que en una autopista (o en una manguera). Por otro lado, el darle derecho exclusivo de vía (o acceso) a una comunicación específica implica que los otros usuarios deben esperar su turno hasta que la transmisión cese.

La tecnología que se usa en Internet y en todos los LAN está en la idea básica de turnarse de manera justa y pareja en el uso de la línea.

Para evitar demoras, la tecnología de redes limita la cantidad de datos que pueden ser enviados. Esto se denomina "cambio de paquetes" (packet switching) y la unidad de datos que es transferida se llama "paquete" (packet).

En resumen, una computadora divide los datos en paquetes y los envía uno a uno. Si dos computadoras comparten una línea ellos se deben turnar en el envío de los paquetes, de modo que los mensajes cortos no deban esperar que los mensajes largos terminen de ser transferidos.

Se debe observar que aunque existe un tamaño máximo, no todos los paquetes tienen el mismo tamaño. Algunos programas interactuan con una computadora enviando solo la información de una tecla que se presiona, tan pronto el usuario lo hace. Otras aplicaciones, como vídeo por ejemplo, necesitan enviar paquetes más grandes.

Este sistema parece casi instantáneo, ya que un LAN típico puede transferir mil paquetes por segundo. De este modo el retraso es mínimo.

Es importante señalar que este sistema para compartir la línea es automático, esto es, no depende de las computadoras ni de un hardware específico. De este modo, los sistemas de "cambio de paquetes" se adaptan en forma automática al "tráfico".

Direcciones

Cada paquete enviado necesita indicar en el mensaje la computadora que lo originó, así como la computadora de destino. Para lograr esto, cada computadora de una red tiene una dirección única. Así, un paquete incluye dos direcciones, la del originador y la del receptor. Cuando un paquete circula por la red, la tarjeta de comunicaciones de una computadora "observa" la red y captura los paquetes dirigidos a su dirección.

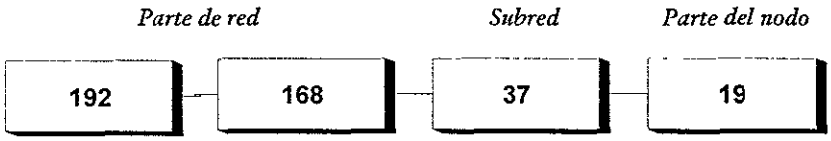


Figura 2.1 Ejemplo de una dirección IP

Como ya hemos visto, para comunicarse entre sí, las computadoras cuentan con una tarjeta de comunicaciones. En la red Internet, se usa la tecnología ethernet. Para poder distinguir a las computadoras de una red, las tarjetas tienen un único número asignado de fábrica llamado ethernet address (o bien dirección ethernet). A su vez, toda computadora conectada a Internet tiene una única dirección mundial. Esta dirección se conoce como número IP. Las direcciones son asignadas por un organismo, que puede ser local, por ejemplo su prestador de servicios. Las numeraciones son del tipo M.N.P.Q donde M, N, P, Q son número entre 0 y 255. Un pequeño cálculo nos muestra que las posibilidades teóricas son 4.294.967.296. Sin embargo, a las organizaciones que forman parte de Internet no se le asignan números de manera individual, sino de acuerdo a clases. Existen tres clases: A, B y C.

- **Clase C**

Consiste de una numeración en la que las tres primera cifras son fijas y la tercera abierta al organismo: 146.83.186.x, lo que permite asignar 256 números IP al interior de la organización.

- **Clase B**

Consiste de una numeración en la que las dos primeras cifras son fijas y la tercera y cuarta abiertas al organismo: 146.83.x.y, lo que permite asignar 256x 256 (65.536) números IP al interior de la organización.

- **Clase A**

Consiste de una numeración en la que la primera cifra es fija y tanto la segunda como la tercera y cuartas son abiertas al organismo: 146.x.y.z, lo que permite asignar 256x256x256 (16.777.216) números IP al interior de la organización.

Tabla de direcciones IP de Internet

Clase	Primer byte	Identificación de red	Identificación de hosts	Número de redes	Número de hosts
A	1 ... 126	1 byte	3 byte	126	16,387,064
B	128 ... 191	2 byte	2 byte	16,256	64,516
C	192 ... 223	3 byte	1 byte	2,064,512	254

En la clasificación de direcciones anterior se puede notar que ciertos números no se usan. Algunos de ellos se encuentran reservados, como es el caso de las direcciones cuyo primer byte sea superior a 223 para las clases D y E, la clase D se reserva a la transmisión de mensajes de difusión múltiple (multicast), mientras que la clase E es destinada a investigación y desarrollo. El valor 127 en el primer byte se utiliza en algunos sistemas para propósitos especiales. También es importante notar que los valores 0 y 255 en cualquier byte de la dirección no pueden usarse normalmente por tener otros propósitos específicos.

El número 0 está reservado para las máquinas que no conocen su dirección, pudiendo utilizarse tanto en la identificación de red para máquinas que aún no conocen el número de red a la que se encuentran conectadas, en la identificación de host para máquinas que aún no conocen su número de host dentro de la red, o en ambos casos.

El número 255 tiene también un significado especial, puesto que se reserva para el "broadcast". El broadcast es necesario cuando se pretende hacer que un mensaje sea visible para todos los sistemas conectados a la misma red. Esto puede ser útil si se necesita enviar el mismo datagrama a un número determinado de sistemas, resultando más eficiente que enviar la misma información solicitada de manera individual a cada uno. Otra situación para el uso de broadcast es cuando se quiere convertir el nombre por dominio de una computadora a su correspondiente número IP y no se conoce la dirección del servidor de nombres de dominio más cercano.

Lo usual es que cuando se quiere hacer uso del broadcast se utilice una dirección compuesta por el identificador normal de la red y por el número 255 (todo unos en binario) en cada byte que identifique al host. Sin embargo, por conveniencia también se permite el uso del número 255.255.255.255 con la misma finalidad, de forma que resulte más simple referirse a todos los sistemas de la red.

El broadcast es una característica que se encuentra implementada de formas diferentes dependiendo del medio utilizado, y por lo tanto, no siempre se encuentra disponible. En ARPAnet y en las líneas punto a punto no es posible enviar broadcast, pero sí que es posible hacerlo en las redes Ethernet, donde se supone que todas las computadoras prestarán atención a este tipo de mensajes.

En el caso de algunas organizaciones extensas puede surgir la necesidad de dividir la red en otras redes más pequeñas (subnets). Como ejemplo una red de clase B que, naturalmente, tiene asignado como identificador de red un número de dos bytes. En este caso sería posible utilizar el tercer byte para indicar en qué red Ethernet se encuentra un host en concreto. Esta división no tendrá ningún significado para cualquier otra computadora que esté conectada a una red perteneciente a otra organización, puesto que el tercer byte no será comprobado ni tratado de forma especial. Sin embargo, en el interior de esta red existirá una división y será necesario disponer de un software de red especialmente diseñado para ello. De esta forma queda oculta la organización interior de la red, siendo mucho más cómodo el acceso que si se tratara de varias direcciones de clase C independientes.

Debido a esta manera de repartir los números, en general no todos los números IP son asignados a nodos y como consecuencia de esto ya se observó que faltarán números IP. En este momento se está realizando el cambio a una nueva numeración (definida hace un tiempo) que permitirá que cada cm^2 del mundo cuente con más de 100 números IP. De hecho, los analistas han dicho que los interruptores de luz, los electrodomésticos, etc., contarán con números IP. Esta nueva numeración se ha dado en llamar IPng, que significa "IPnext generation". El cambio será dramático, pero se piensa llevar a cabo muy lentamente, probablemente con la ayuda de parches de software, con el propósito de mantener la red activa durante toda la transición.

Dominios

Recordar que las direcciones en Internet son globales, es decir, válidas en todo el mundo y por lo tanto únicas en este mismo contexto. Para todos es difícil recordar un número largo o una serie de números, es por esta razón que a las direcciones en Internet se le asignan nombres. Estos nombres siguen un mismo esquema, que se detallará a continuación.

Se comienza el análisis de los nombres con dos ejemplos: www.sun.com y avelar.mpsnet.net.mx. Para entender esta nomenclatura se dice primero que cada una de estas direcciones se llama un dominio. Cada parte del dominio llamada un subdominio, está separada por un punto. En los ejemplos, el primer dominio tiene tres subdominios y el segundo tiene cuatro subdominios. La lectura de estos dominios se hace de derecha a izquierda. El subdominio de más a la derecha se llama el dominio de nivel superior, es el más general. A medida que se leen de izquierda a derecha los subdominios se hacen cada vez más específicos.

En el primer ejemplo, el subdominio "com" especifica que la computadora es de una institución comercial. El siguiente subdominio, "sun", indica la institución, en este caso Sun Microsystems y finalmente el último subdominio corresponde al nombre de una computadora (servidor) específico llamado "www" (se acostumbra usar este tipo de nombre de dominio para especificar servidores HTTP de WWW, este nombre de dominio es un alias del nombre real de dominio del servidor).

En el segundo ejemplo, el primer subdominio, "mx", indica que la computadora está en México, el segundo subdominio, "net", indica que es una institución dedicada al cómputo, el tercer subdominio, "mpsnet", indica un departamento dentro de la institución y el cuarto subdominio la computadora específica llamada avelar. En los nombres de los dominios no se diferencia entre mayúsculas o minúsculas, sin embargo se prefiere escribir todo en minúsculas.

Dominios de Nivel Superior (Primer Nivel)

En general hay dos tipos de dominios de primer nivel: el más antiguo llamado dominio organizacional y el más nuevo llamado dominio geográfico. Los dominios organizacionales están basados en un esquema de direcciones desarrollado antes de la internacionalización de la Internet y es casi exclusivamente en Estados Unidos. La idea era que el domino de primer nivel mostrara el tipo de organización responsable por la computadora. La tabla muestra las distintas categorías usadas.

Dominio	Significado
com	Organización comercial
edu	Institución educacional
gov	Organización gubernamental
int	Organización Internacional
mil	Militares
net	Organización de Redes
org	Organización sin fines de lucro

Una vez que la Internet se expandió internacionalmente, fue necesario introducir dominios más específicos, para lo cual se utilizó una abreviación para la designación geográfica de dos letras, adoptando la abreviación usada históricamente por las líneas aéreas. La siguiente tabla muestra algunos ejemplos.

Dominio	Significado
ar	Argentina
au	Australia
ca	Canadá
ch	Suiza (Cantones de Helvecia)
cl	Chile
es	España
de	Alemania
fr	Francia
jp	Japón
mx	México
uk	Reino Unido
us	Estados Unidos

Los datos transmitidos por la red no utilizan los nombres asignados, si no los números IP de cada nodo, para que esto funcione hay una, o más bien varias bases de datos llamadas Servidores de Nombres de Dominio ("Domain Name Servers" o DNS) que traducen de nombre a número y viceversa.

Números de puerto de protocolo

La comunicación entre computadoras se lleva a cabo mediante el intercambio de paquetes de bytes. La semántica de los conjuntos de bytes que recibe una computadora viene dictada por la aplicación a la cual van destinados. Los paquetes de información que se difunden a través de una red de computadoras son encaminados hacia un *host* (equipo o computadora) concreto y dentro de dicho *host* a un puerto concreto.

Se puede pensar en un puerto como en un canal de comunicación. Cada computadora dispone de un total de 65000 canales o puertos, los cuales pueden o no estar activos. Para que un puerto esté activo es necesaria una aplicación que tome el control del mismo y sea capaz de gestionar los paquetes de bytes que llegan por dicho puerto. Cuando un *host* recibe un paquete examina su cabecera para averiguar a que puerto va destinado, si existe una aplicación escuchando dicho puerto, entonces se le pasan los bytes del paquete para que esta los interprete y actúe consecuentemente. El *host* no responderá a peticiones de conexión encaminadas hacia un puerto para el cual no existe ninguna aplicación escuchando.

Es decir, de los paquetes de bytes remitidos hacia una computadora en concreto, sólo se va a atender aquellos paquetes para los cuales existe una aplicación escuchando en el puerto al cual van encaminados. Existe una serie de puertos standard utilizados universalmente para varios servicios.

Algunos de ellos son :

SERVICIO	PUERTO	DESCRIPCIÓN
FTP	21	File Transfer Protocol.
TELNET	23	Permite el acceso a una cuenta en un equipo remoto.
SMTP	25	Para enviar correo electrónico.
POP3	110	Para obtener correo electrónico del servidor.
HTTP	80	Protocolo para WWW.
NNTP	119	Grupos de noticias de Internet.
GOPHER	70	Navegador modo texto.

Estos puertos están asignados en el standard RFC 1700.

Arquitectura Cliente/Servidor

A pesar de la gran variedad de servicios disponibles en Internet, el software que implementa un servicio usa un esquema fijo: Cliente/Servidor. Bajo este esquema, cada computadora conectada a la red es a la vez potencialmente un cliente (que requiere servicios) y un servidor (que provee servicios). Es necesario aclarar que en realidad lo que se denomina cliente o servidor es un programa específico. Las computadoras no se comunican entre sí, sólo los programas lo hacen y es en este sentido que un programa provee los servicios que otro programa requiere.

Desde el punto de vista de una aplicación, el TCP/IP, al igual que muchos otros protocolos de comunicación, implementa un mecanismo fiable para la transmisión de datos entre computadoras. En concreto, el TCP/IP permite que un programador pueda establecer comunicación de datos entre dos programas de aplicación, tanto si ambos se están ejecutando en la misma máquina, como en máquinas distintas unidas por algún camino físico (una red local, conexión telefónica directa entre computadoras, computadoras conectas a Internet, etc.).

Hay que tener presente que el protocolo TCP/IP especifica los detalles y mecanismos para la transmisión de datos entre dos aplicaciones comunicantes, pero no dictamina cuando ni porque deben interactuar ambas aplicaciones, ni siquiera especifica como debería estar organizada una aplicación que se va a ejecutar en un entorno distribuido. Es tarea del diseñador de la aplicación distribuida el establecer un protocolo de comunicación y sincronización adecuado.

El modelo Cliente/Servidor da solución al problema del *rendezvous* (reencuentro). Según este modelo, en cualquier par de aplicaciones comunicantes, una de las partes implicadas en la comunicación debe iniciar su ejecución y esperar (indefinidamente) hasta que la otra parte contacte con ella. Por un lado, el cliente es la computadora que se encarga de efectuar una petición o solicitar un servicio. El cliente no posee control sobre los recursos, sino que es el servidor el encargado de manejarlos. Por otro lado, la computadora remota que actúa como servidor evalúa la petición del cliente y decide aceptarla o rechazarla consecuentemente. Una vez que el servidor acepta el pedido la información

requerida es suministrada al cliente que efectuó la petición, siendo este último el responsable de proporcionar los datos al usuario con el formato adecuado. Esta solución es importante, ya que el TCP/IP no proporciona ningún mecanismo que automáticamente lance procesos para atender mensajes entrantes, debe de existir un proceso en espera que sea capaz de atender dichos mensajes (peticiones de servicio).

La arquitectura Cliente/Servidor es una forma específica de diseño de aplicaciones, aunque también se conoce con este nombre a las computadoras en las que estas aplicaciones son ejecutadas. Se debe precisar que cliente y servidor no tienen que estar necesariamente en computadoras separados, sino que pueden ser programas diferentes que se ejecuten en la misma computadora.

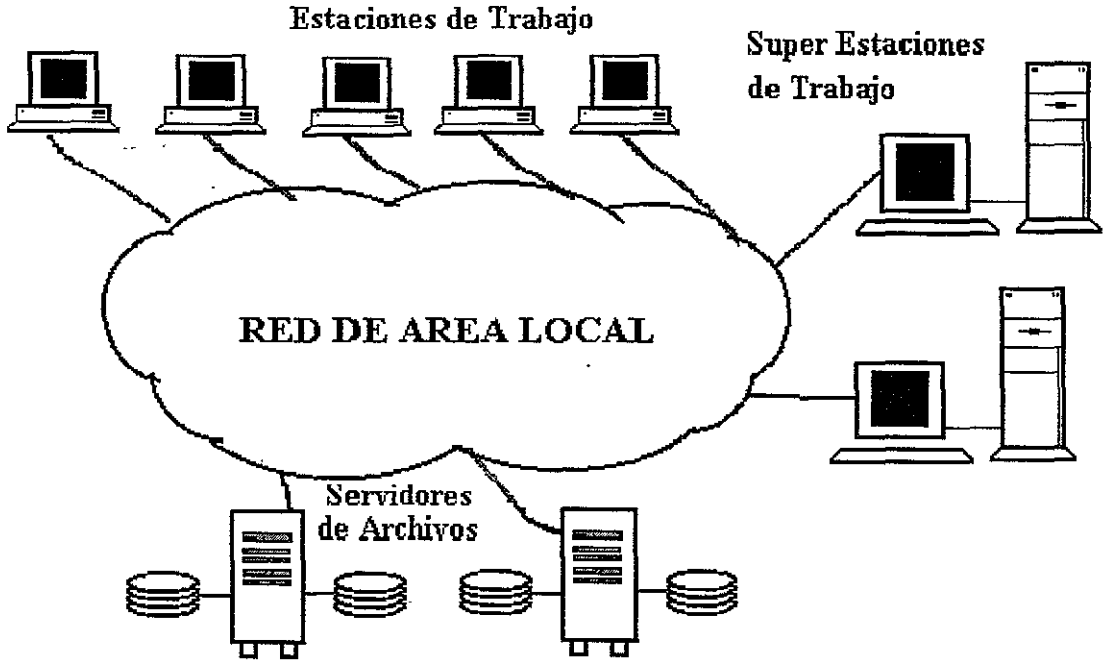


Figura 2.2 Modelo de un Sistema Cliente/Servidor

2.3 Servicios que ofrece

Los servicios que podemos utilizar desde una computadora conectado a Internet son muy diversos. Podemos definir servicio como un conjunto de programas y utilidades que nos permiten realizar una determinada tarea.

WWW (World Wide Web)

El Proyecto World Wide Web nació en respuesta a la necesidad que la comunidad científica internacional tenía de nuevos sistemas de distribución de la información. Se trata sin duda del servicio más conocido actualmente. Es un sistema de información basado en páginas que contienen hipertexto. En el siguiente capítulo se describirá más a profundidad.

FTP (File Transfer Protocol)

Este servicio (Protocolo de transferencia de archivos) permite el intercambio de información entre computadoras distantes, por lo que se puede enviar y recibir archivos entre distintas máquinas. Sería equivalente a conectarse a un servidor de archivos, donde se busca algo qué nos interesa (programas, documentos, drivers...).

Para conectarse a un servidor de este tipo se necesita tener instalado el programa cliente apropiado o bien hacer Telnet a una máquina que lo tenga. Hay dos formas de acceder a servidores FTP:

La primera es mediante una cuenta local en la máquina y la segunda es haciendo un FTP anónimo (en este caso a servidores públicos de software). Para hacer FTP anónimo, el login (o nombre de usuario) deberá ser anonymous y el password (o contraseña) la dirección de correo electrónico en algún servidor.

Existen servidores de Universidades, compañías informáticas, empresas que ofrecen todo tipo de archivos que van desde drivers hasta programas completos, pasando por documentos, etc.

Obviamente, los programas que se podrán conseguir a través de estos servidores no serán nunca de carácter comercial. Se podrán encontrar programas de costo compartido, shareware, y programas de dominio público, freeware.

Otro dato a destacar es el hecho de que casi todos los archivos que circulan por la red tienen formato comprimido. Esto acelera las transmisiones y ahorra el espacio de disco de los servidores. El formato de compresión por excelencia es el ZIP aunque también se pueden encontrar archivos comprimidos con ARJ y, cada vez más, archivos "autodescomprimibles" (extensión .EXE).

Actualmente para hacer FTP es bastante común utilizar programas de entorno gráfico (de fácil manejo y muy intuitivos). Entre los programas de este tipo se puede destacar: CuteFTP y WS_FTP.

E-MAIL (Correo electrónico)

Este es otro de los servicios más populares dentro de Internet. Permite enviar mensajes (y/o archivos) como si de correo postal se tratara, pero con la diferencia de que se recibirán inmediatamente después de mandarlos y prácticamente nunca se pierde.

Cada usuario de la red dispone de una dirección electrónica que le identifica en todo Internet (el equivalente postal lo tenemos con nuestra dirección particular). Un ejemplo de dirección electrónica es avelar@servidor.unam.mx. Estas direcciones se basan en la misma estructura de las direcciones IP y nombres de dominio analizados anteriormente. La única diferencia es el símbolo @ que se encarga de enlazar el "quién" con el "dónde" de la dirección. En este caso, sería el usuario con nombre "avelar" correspondiente a la máquina "servidor.unam.mx".

El correo electrónico presenta numerosas posibilidades de utilización. Además de las ya clásicas de comunicación entre varias personas conectadas podemos utilizar las listas (o grupos) de distribución. Estos grupos corresponden a listados de direcciones agrupados por temáticas comunes. Al enviar un mensaje a una lista concreta, automáticamente, lo recibirán todos sus miembros.

TELNET

Mediante esta aplicación es posible conectarse a una computadora remota. De esta forma, se pueden ejecutar programas y disponer de los recursos disponibles en dicha computadora. Para poder hacerlo, la computadora a la que queremos conectarnos debe de soportar accesos al mismo y lo normal es que soporte varios accesos simultáneos. Normalmente, cuando alguien se conecta a otra computadora mediante Telnet se conecta a un servidor trabajando en UNIX o en otro sistema operativo multitarea.

NEWSGROUPS

Son lugares dentro de Internet en los que tiene lugar diversas "charlas" o "tertulias". Se puede imaginar un tablón de anuncios en el que diversas personas van dejando mensajes sobre diversos temas. Cada uno puede llegar y pegar su mensaje. Los demás lo podrán ver y si es de su interés contestar con otro apunte que se añadirá en el tablón. Finalmente, los mensajes irán caducando con el paso del tiempo.

Se trata de grupos públicos, ordenados por jerarquías, donde todo el mundo puede escribir sobre lo que quiera y todo el mundo puede leerlo. Estos "forúms" son ideales para preguntar dudas, comentar noticias, estar siempre al día de esa materia que más interesa.

ARCHIE

Es una herramienta de búsqueda de información en Internet. Se trata de una base de datos de acceso público que permite localizar un archivo determinado dentro de la familia de servidores FTP.

Archie mantiene un índice actualizado de los archivos que hay en la red. Actualmente existe un servidor Archie en cada uno de los países importantes que existen en la red. Para acceder a un servidor Archie (si no se dispone de la correspondiente aplicación cliente) debemos hacer Telnet a una máquina que sí disponga de este tipo de aplicaciones. Otra forma para utilizar Archie es mediante el servicio de WWW. El método es sencillo: se dice cuál es el archivo de interés y se le dan algunos parámetros sobre cómo se quiere que se realice la búsqueda.

GOPHER

Este servicio nació en respuesta a los problemas que existían en Internet a la hora de encontrar información o recursos. Funciona presentando en la pantalla un menú de opciones cuyos títulos dan una idea clara de lo que contiene. Para conectarse a un servidor Gopher también se necesita un programa especial cliente Gopher.

Actualmente este recurso se encuentra en vías de extinción y casi absolutamente en desuso.

VERONICA

Se trata de otra herramienta para buscar información. Se puede decir que Verónica es al Gopher lo que Archie es al FTP. Dado que los servidores Gopher empezaron a proliferar se tuvo la necesidad de crear una utilidad que permitiera localizar de una manera eficaz la información dentro de los mismos. Así surgieron los servidores llamados Veronica (Very Easy Rodent Oriented Netwide Index to Computerized Archives). A diferencia de Archie, Veronica no es un servidor, se accede a ella a través de los propios Gopher.

WAIS (Wide Area Information Services)

WAIS es una herramienta cliente que permite hacer búsquedas en bases de datos indexadas por servidores WAIS. Cuando no se dispone de un cliente Wais propio se debe conectar a uno mediante Telnet.

3
WWW (World Wide Web)

El Proyecto World Wide Web nació en respuesta a la necesidad que la comunidad científica internacional tenía de nuevos sistemas de distribución de la información

El WWW se pensó originalmente como un medio de distribución de la información entre equipos investigadores geográficamente dispersos para la comunidad de físicos de altas energías vinculados al CERN (el Laboratorio Europeo de Física de Partículas localizado en Génova que es financiado por 19 países) Se pretendía que los recursos disponibles en formato electrónico, que residían en computadoras distintas conectadas a la red, fuesen accesibles para cada investigador desde su propia terminal de forma clara y simple, sin necesidad de aprender varios programas distintos. Todos los recursos existentes deberían integrarse en una red hipertextual gestionada por computadoras.

En este capítulo se verá la historia del WWW. Su modo de operación. Cuál es el esquema de funcionamiento de los servidores de HTTP. Navegadores para WWW.

3.1 Historia

En el mes de Marzo de 1989 Tim Berners-Lee del Laboratorio Europeo de Física de Partículas, localizado en Génova (que es abreviado como CERN) envío un circular el cual proponía el desarrollo de un "sistema de hipertexto" con el propósito de obtener información compartida fácil y eficientemente entre equipos de investigadores geográficamente separados en la comunidad de Física de Alta Energía.

Los tres componentes importantes de la propuesta son los siguientes:

- Una interfaz de usuario consistente.
- La habilidad de incorporar una gran cantidad de rangos de tecnología y tipos de documentos.
- Ser "universal", es decir, que cualquier persona en cualquier lugar de la red, en una gran variedad de computadoras diferentes, pueda leer el mismo documento como cualquier otro y muy fácil.

Más de un año después, en el mes de octubre de 1990, el proyecto fue presentado y dos meses después el proyecto World Wide Web comenzó a tener éxito. El trabajo comenzó con el primer navegador (llamado WWW), y por el final de 1990 este navegador y otro para el sistema operativo NeXTStep fueron introducidos. La mayor parte de los principios de acceso de hipertexto y la lectura de diferentes tipos de documentos estaban implementados.

En el verano de 1991 se ofrecieron seminarios acerca del Web. En Octubre de 1991 se instalo la "compuerta" para las búsquedas del Web a través de WAIS y posteriormente después del final de 1991 el CERN anuncio en general el Web a la comunidad de Física de Alta Energía.

Esencialmente, 1992 fue el año de desarrollos en materia del Web. Los navegadores de WWW estuvieron disponibles vía FTP desde el CERN y el Web fue presentado a una gran variedad de organizaciones y audiencias. En Enero de 1993, 50 servidores de Web estaban en operación, en ese mismo tiempo el navegador Viola para el sistema X Windows(sistema de ventanas de varios sistema Unix) estuvo disponible. Viola fue el líder en la tecnología de navegación del Web.

Otros dos navegadores estuvieron disponibles al principio de 1993. EL navegador de CERN para Macintosh puso a las Mac en el juego del Web y al mismo tiempo apareció el Mosaic. En Febrero de 1993, la primera versión de X Mosaic (Mosaic para X Windows) fue ofrecida por NCSA (Centro Nacional para aplicaciones de supercomputo en Champaign, Illinois); que fue diseñado por Marc Andreessen. En Marzo de 1993, el tráfico del WWW ocupo el 0.1 % del total del tráfico del "backbone" de Internet. Seis meses después, el Web comenzó a demostrar su potencial expandiendo a 1 % del tráfico del backbone.

El mismo incremento fue evidente también en el número de servidores de Web, que para Octubre de 1993 se incremento aproximadamente a 500. Al final de 1993, el proyecto del Web comenzó a recibir reconocimientos técnicos, y artículos del Web y Mosaic comenzaron a aparecer en prestigiadas publicaciones. Poco tiempo después apareció el navegador Cello, un navegador alternativo desarrollado por el Instituto de Información Legal en la Universidad Cornell, para usuarios de Windows de Microsoft.

Importantes desarrollos vinieron en 1994. Primero, se expandió el trabajo en desarrollo del acceso seguro al Web. Segundo, la licencia de Mosaic para comercializar sus desarrollos. Los desarrolladores del Mosaic de NCSA (Andreessen y otros) su unieron para formar Mosaic Communication Corporation (hoy conocida como Netscape Communication Corporation).

En Julio de 1994, el CERN comenzó a dar un giro al proyecto del Web hacia un nuevo grupo llamado la Organización W3, formado principalmente por el CERN y el MIT (Instituto de Tecnología de Massachusetts).

A través del curso de los siguientes meses de 1994 y 1995, esta "aventura" de desarrollo fue transformada en una colección de organizaciones llamadas The World Wide Web Consortium (Consorcio WWW). Encabezada por el fundador del Web, Tim Berners-Lee, el consorcio opera con los fondos que sus miembros otorgan.

Hoy en día el WWW permite que personas separadas por grandes distancias, pueden comunicarse y verse en tiempo real al costo de una llamada local. Se observa con interés la gran cantidad de compras que se realizan en línea, como se prestan servicios y se promocionan cualquier clase de productos. Como temas de investigación son discutidos y analizados por grupos de discusión, poniendo en las manos del mundo las soluciones y conclusiones de los problemas.

Todos éstos y más beneficios que para nombrarlos completamente tomaría mucho tiempo, no pasaban por la mente del más innovador y soñador pocos años atrás, y sin embargo hoy son la realidad que mueve al mundo, acercando cada vez más el momento de la integración y la globalización total. Pero lo mejor no es solo esto, el WWW seguirá mejorando y en pocas palabras hará la vida mucho más sencilla.

3.2 ¿Cómo funciona?

El WWW responde a un modelo Cliente/Servidor. Se trata de un paradigma de división del trabajo informático en el que las tareas se reparten entre un número de clientes que efectúan peticiones de servicios de acuerdo con un protocolo, y un número de servidores que responden a estas peticiones. En el Web los clientes demandan hipertextos a los servidores.

Para desarrollar un sistema de este tipo ha sido necesario:

a) Un nuevo protocolo que permite saltos hipertextuales, es decir, de un nodo origen a otro de destino, que puede ser texto, imágenes, sonido, animaciones, vídeo, etc. Este protocolo se denomina HTTP (HyperText Transfer Protocol) y es el lenguaje que hablan los servidores de Web.

b) Inventar un nuevo lenguaje para representar hipertextos que incluyera información sobre la estructura y formato de representación y, especialmente, indicara el origen y destinos de los saltos de hipertexto. Este lenguaje es el HTML (HyperText Markup Language).

c) Idear una forma de codificar las instrucciones para los salto hipertextuales de un objeto a otro de la Internet, las URI(Universal Resource Identifier).

d) Desarrollar aplicaciones cliente para todo tipo de plataformas y resolver el problema de cómo se accede a la información que está almacenada, y que ésta sea disponible a través de los diversos protocolos (FTP, HTTP, WAIS...) y que representen a su vez información multiformato (texto, imágenes, animaciones, etc.). Con este fin aparecen varios clientes, entre los que destacan MOSAIC del NCSA, el Netscape Communicator (de Netscape Communications Corporation) y el Internet Explorer (de Microsoft).

HTML

El HTML (Hyper Text Markup Language) es un lenguaje que sirve para escribir hipertexto, es decir, documentos de texto presentado de forma estructurada, con enlaces (links) que conducen a otros documentos o a otras fuentes de información relacionadas, y con inserciones multimedia (gráficos, sonido...) que pueden estar en tu propia máquina o en máquinas remotas de la red. Para hacerlo, basta situar el puntero del ratón encima de ellos y pulsar el botón.
El nodo de llegada (lugar a donde hizo referencia el link) puede ser otro hipertexto o también un nodo no hipertextual integrado en la red: un servidor gopher, un grupo de news, una búsqueda en una base de datos Wais, etc.

El lenguaje HTML ha sido diseñado teniendo presentes dos normas básicas:

1. Define la estructura y componentes de un documento, no la forma concreta en que estos componentes se presentan cuando un cliente los visualiza.
2. No está atado a ningún entorno particular. El contenido de un documento HTML puede ser interpretado y visualizado en computadoras con arquitecturas de características muy diferentes.

Estas dos condiciones son muy importantes, ya que permiten que una determinada información pueda ser vista por usuarios diferentes, con independencia de las capacidades de su entorno de trabajo. Un documento HTML sólo contiene la especificación de los elementos que lo componen (títulos, párrafos, imágenes, etc.), y es responsabilidad de cada cliente el mostrar esta información de la manera más adecuada, en función de sus capacidades y las características del entorno.

Se puede decir que HTML es un lenguaje interpretado, ya que son los navegadores(browsers) los encargados de procesar y representar su contenido. Un navegador o visualizador de Web tiene mucha libertad para ajustar la presentación de un documento HTML en función de los recursos disponibles.

Por lo tanto, se debe ser consciente que, dependiendo del entorno de trabajo de cada cliente (sistema operativo, tipo y versión del navegador, etc.), un documento HTML puede ser visualizado de manera diferente.

Uniendo los servidores HTTP y los documentos HTML, se dispone de un sistema distribuido de acceso a información, que combina un formato de presentación muy atractivo con un sencillo mecanismo de navegación por la información, basado en la selección de elementos activos.

La descripción del lenguaje se basa en especificar en el texto la estructura lógica del contenido (títulos, párrafos de texto normal, enumeraciones, definiciones, citas, etc.) así como los diferentes efectos que se quieren dar (especificar los lugares del documento donde se debe poner cursiva, negrita, o un gráfico determinado) y dejar que luego la presentación final de dicho hipertexto se realice por un programa especializado(como Internet Explorer, o Netscape).

Por supuesto, la estética de los documentos escritos en HTML no se limita a texto digamos normal; consigue todos los efectos que habitualmente se pueden producir con un moderno procesador de textos: negrita, cursiva, distintos tamaños y fuentes, tablas, párrafos tabulados, sangrías, incluso texto y fondo de página de colores, y muchos más.

Una página Web es un sistema que ofrece hipertextos (como el de la Figura 3.1). Crear una página Web significa escribir una documento de texto en el lenguaje HTML. Publicar una página Web significa instalar este documento creado en HTML en un servidor HTTP (que tiene una estructura específica).

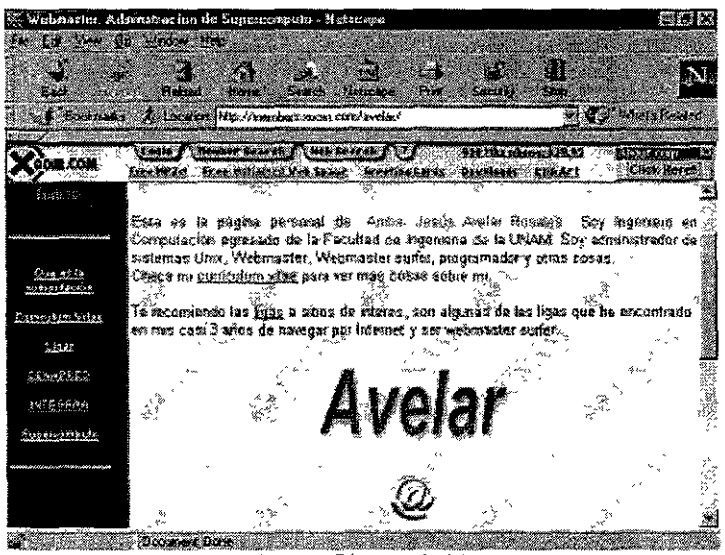


Figura 3.1 Página Web típica

URL y URI

Para conseguir la integración entre links o enlaces de una página Web se necesita un esquema de nombres común con el que localizar los documentos en cualquiera de estos sistemas de información de Internet.

Los URL("Uniform Resource Locators", Localizadores Uniformes de Recursos), proporcionan la dirección de un recurso en la Web. Una nueva forma en perspectiva para identificar recursos es llamada URN("Uniform Resource Name", Nombre Uniforme de Recurso). Juntos son llamados URI("Uniform/Universal Resource Identifiers", Identificadores Uniformes de Recursos).

El sistema de direccionamiento que utiliza WWW es el URI. A través de un URI localizamos un documento en el sistema de información universal que nos genera el WWW. Los URL, son "localizadores" de direcciones dentro de la red, que relacionan un servicio con un servidor. Constituyen la herramienta esencial del Web, ya que permiten la localización y conexión con cualquier servidor y recurso de la Internet.

Se podría decir que el URL es el camino completo que tendría que seguir un usuario para encontrar un archivo en Internet, especificando incluso la aplicación utilizada. Es una extensión lógica del nombre de un archivo a la red y a la aplicación. De esta forma, un URL nombra a los recursos de una forma única y además especifica el tipo de acceso al objeto designado.

Un URL consta de un nombre de esquema que se utiliza para designar el protocolo de acceso al objeto (documento), el nombre del *host*(equipo o computadora) junto con el puerto TCP/IP de la aplicación (éste en muchos casos opcional), y del camino y el propio nombre del archivo en este *host*. Para referirse a una parte de un documento se añade un sufijo que localiza un cierto término referencial. La sintaxis de una dirección URL es la siguiente:

esquema : // host.dominio[: puerto] / camino / archivo[# término]

Un sufijo "?" seguido por palabras separadas con el carácter "+" permite ejecutar una búsqueda. Las referencias entre documentos cuyos URLs coinciden en algunas partes pueden ser abreviados formando un nombre relativo. Los esquemas de los distintos servicios Internet son los siguientes: file, news, http, telnet, gopher, wais, x500. Los programas clientes que no hablan todos estos protocolos pueden ser configurados para usar pasarelas con ciertas direcciones URL.

3.3 Servidor HTTP

HTTP (HyperText Transfer Protocol) es un protocolo muy simple implementado sobre TCP/IP. Es similar en muchas funcionalidades al protocolo de Gopher. El cliente HTTP envía al servidor un identificador de documento con o sin palabras de búsqueda y el servidor responde con documentos HTML o simplemente texto. Es un protocolo que no mantiene una conexión permanente.

Se diseñó especialmente para entender las exigencias de un sistema hipermedia distribuido como es el World Wide Web. Sus principales características son:

- **Ligereza**: reduce la comunicación entre clientes y servidores a intercambios discretos, de modo que no sobrecarga la red y permite saltos hipertextuales rápidos.
- **Generalidad**: puede utilizarse para transferir cualquier tipo de datos (según el estándar MIME sobre el tráfico multimedia y que incluye también los que se desarrollen en el futuro).
- **Extensibilidad**: contempla distintos tipos de transacciones entre clientes y servidores y el futuro desarrollo de otros nuevos.

Una conexión entre un cliente y un servidor consiste en cuatro transacciones fundamentales:

- **El establecimiento de la conexión**: el cliente se conecta al servidor mediante una conexión TCP/IP al puerto 80, que está reservado para HTTP por la IANA (Internet Assigned Numbers Authority).
- **La petición**: mensaje del cliente al servidor, especificando qué información requiere.
- **La respuesta**: transmisión del servidor, que manda la información al cliente.
- **La conclusión**: uno de los participantes de la comunicación cierra la conexión.

Cada vez que un cliente realiza una petición real a un servidor HTTP, se ejecutan los siguientes pasos:

- Un usuario accede a una URL, seleccionando un enlace de un documento HTML o introduciéndola directamente en el campo "Location" del cliente Web(navegador).
- El cliente Web decodifica la URL, separando sus diferentes partes. Así identifica el protocolo de acceso, la dirección DNS o IP del servidor, el posible puerto opcional (el valor por defecto es 80) y el objeto requerido del servidor.
- Se abre una conexión TCP/IP con el servidor, llamando al puerto TCP correspondiente.
- Se realiza la petición. Para ello, se envía el comando necesario (GET, POST, HEAD, etc.), la dirección del objeto requerido (el contenido de la URL que sigue a la dirección del servidor), la versión del protocolo HTTP empleada (casi siempre HTTP/1.0) y un conjunto variable de información, que incluye datos sobre las capacidades del navegador o browser, datos opcionales para el servidor, etc.
- El servidor devuelve la respuesta al cliente. Consiste en un código de estado y el tipo de dato MIME de la información de retorno, seguido de la propia información.
- Se cierra la conexión TCP.

Este proceso se repite en cada acceso al servidor HTTP. Por ejemplo, si se accesa un documento HTML en cuyo interior están insertadas cuatro imágenes, el proceso anterior se repite cinco veces, una para el documento HTML y cuatro para las imágenes.

En la actualidad se ha mejorado este procedimiento, permitiendo que una misma conexión se mantenga activa durante un cierto periodo de tiempo, de forma que sea utilizada en sucesivas transacciones. Este mecanismo, denominado "HTTP Keep Alive", es empleado por la mayoría de los clientes y servidores modernos. Esta mejora es imprescindible en una Internet saturada, en la que el establecimiento de cada nueva conexión es un proceso lento y costoso.

Veamos el proceso completo con un ejemplo:

1. Desde un cliente se solicita la URL http://www.mcc.unam.mx/~avelar/index.html

2. Se abre una conexión TCP/IP con el puerto 80 del sistema www.mcc.unam.mx.

3. El cliente realiza la solicitud, enviando algo similar a esto:

```
GET /~avelar/index.html HTTP/1.0   // Operación solicitada + objeto + versión de HTTP

Accept: text/plain                 // Lista de tipos MIME que acepta o entiende el cliente
Accept: text/html
Accept: audio/*
Accept: video/mpeg
.. .. ..
Accept: */*                        // Indica que acepta otros posibles tipos MIME

User-Agent: Mozilla/3.0 (WinNT; I) // Información sobre el tipo de cliente

Línea en blanco(\n)                //  Indica el final de la petición
```

4. El servidor responde con la siguiente información:

```
HTTP/1.0 200 OK                    // Status de la operación; en este caso, correcto

Server: NCSA 1.4                   // Tipo y versión del servidor. Puede ser Apache, Netscape, MSI, NCSA
```

MIME-version: 1.0 // Versión de MIME que maneja

Content-type: text/html // Definición MIME del tipo de datos a devolver

Content-length: 254 // Longitud de los datos que contiene el documento HTML

Last-modified: 29-Jul-2000 12:30:00 // Fecha de modificación de los datos

Línea en blanco(\n)

<HTML> // Comienzo de los datos

<HEAD><TITLE>Pagina de A. Avelar</TITLE></HEAD>

<BODY>
.. .. ..
</HTML>

5. Se cierra la conexión.

Estructura de los mensajes HTTP

El diálogo con los servidores HTTP se establece a través de mensajes formados por líneas de texto, cada una de las cuales contiene los diferentes comandos y opciones del protocolo. Sólo existen dos tipos de mensajes, uno para realizar peticiones y otro para devolver la correspondiente respuesta. La estructura general de los dos tipos de mensajes se puede ver en el siguiente esquema:

Mensaje de solicitud	**Mensaje de respuesta**
Comando HTTP + parámetros	Resultado de la solicitud
Cabeceras del requerimiento	Cabeceras de la respuesta
(línea en blanco)	(línea en blanco)
Información opcional	Información opcional

La primera línea del mensaje de solicitud contiene el comando que se solicita al servidor HTTP, mientras que en la respuesta contiene el resultado de la operación, un código numérico que permite conocer el éxito o fracaso de la operación. Después aparece, para ambos tipos de mensajes, un conjunto de cabeceras (unas obligatorias y otras opcionales), que condicionan y matizan el funcionamiento del protocolo.

La separación entre cada línea del mensaje se realiza con un par CR-LF (retorno de carro más nueva línea). El final de las cabeceras se indica con una línea en blanco, tras la cual se pueden incluir los datos transportados por el protocolo, por ejemplo, el documento HTML que devuelve un servidor o el contenido de un formulario que envía un cliente.

El conocimiento y empleo de los mensajes HTTP es necesario en las siguientes situaciones:

- Para diseñar módulos CGI, ya que es preciso construir una respuesta similar a la que el servidor HTTP proporciona al cliente.
- Para diseñar aplicaciones independientes que soliciten información a un servidor(automatizar la recuperación de documentos, construir robots de búsqueda, o robots de verificación de status) se debe construir una cabecera HTTP con la información de la petición al servidor.

Comandos del protocolo

Los comandos o verbos de HTTP representan las diferentes operaciones que se pueden solicitar a un servidor HTTP. El formato general de un comando es:

Nombre del comando Objeto sobre el que se aplica Versión de HTTP utilizada

Cada comando actúa sobre un objeto del servidor, normalmente un archivo o aplicación, que se toma de la URL de activación. La última parte de esta URL, que representa la dirección de un objeto dentro de un servidor HTTP, es el parámetro sobre el que se aplica el comando. Se compone de una serie de nombres de directorios y archivos, además de parámetros opcionales para las aplicaciones CGI.

El estándar HTTP/1.0 recoge únicamente tres comandos, que representan las operaciones de recepción y envío de información y chequeo de estado:

- **GET**: Se utiliza para recoger cualquier tipo de información del servidor. Se utiliza siempre que se pulsa sobre un enlace o se teclea directamente a una URL. Como resultado, el servidor HTTP envía el documento correspondiente a la URL seleccionada, o bien activa un módulo CGI, que generará a su vez la información de retorno.
- **HEAD**: Solicita información sobre un objeto (archivo): tamaño, tipo, fecha de modificación, etc. Es utilizado por los gestores de caches de páginas o por los servidores proxy, para conocer cuándo es necesario actualizar la copia que se mantiene de un archivo. También se utiliza para conocer el status de un objeto(recurso o archivo) sin necesidad de bajar el propio objeto, esto como veremos más adelante, es fundamental para la implementación de nuestra aplicación.
- **POST**: Sirve para enviar información al servidor, por ejemplo los datos contenidos en un formulario. El servidor pasará esta información a un proceso encargado de su tratamiento (generalmente una aplicación CGI). La operación que se realiza con la información proporcionada depende de la URL utilizada. Se utiliza, sobre todo, en los formularios.

Un cliente Web(navegador) selecciona automáticamente los comandos HTTP necesarios para recoger la información requerida por el usuario. Así, ante la activación de un enlace, siempre se ejecuta una operación GET para recoger el documento correspondiente. El envío del contenido de un formulario utiliza GET o POST, en función del atributo de <FORM METHOD="...">. Además, si el cliente Web tiene un cache de páginas recientemente visitadas, puede utilizar HEAD para comprobar la última fecha de modificación de un archivo, antes de traer una nueva copia del mismo.

Posteriormente se han definido algunos comandos adicionales, que sólo están disponibles en determinadas versiones de servidores HTTP, con motivos eminentemente experimentales. La última versión de HTTP, denominada 1.1, recoge estas y otras novedades, que se pueden utilizar, por ejemplo, para editar las páginas de un servidor Web trabajando en remoto.

- **PUT**: Actualiza información sobre un objeto del servidor. Es similar a POST, pero en este caso, la información enviada al servidor debe ser almacenada en la URL que acompaña al comando. Así se puede actualizar el contenido de un documento.

- **DELETE**: Elimina el documento especificado del servidor.

- **LINK**: Crea una relación entre documentos.

- **UNLINK**: Elimina una relación existente entre documentos del servidor.

Las cabeceras

Son un conjunto de variables que se incluyen en los mensajes HTTP, para modificar su comportamiento o incluir información de interés. En función de su nombre, pueden aparecer en los requerimientos de un cliente, en las respuestas del servidor o en ambos tipos de mensajes. El formato general de una cabecera es:

Nombre de la variable : Cadena ASCII con su valor

Los nombres de variables se pueden escribir con cualquier combinación de mayúsculas y minúsculas. Además, se debe incluir un espacio en blanco entre el signo ":" y su valor. En caso de que el valor de una variable ocupe varias líneas, éstas deberán comenzar, al menos, con un espacio en blanco o un tabulador.

Cabeceras comunes para peticiones y respuestas

- **Content-Type**: descripción MIME de la información contenida en este mensaje. Es la referencia que utilizan las aplicaciones Web para dar el correcto tratamiento a los datos que reciben.
- **Content-Length**: longitud en bytes de los datos enviados, expresado en base decimal.
- **Content-Encoding**: formato de codificación de los datos enviados en este mensaje. Sirve, por ejemplo, para enviar datos comprimidos (x-gzip o x-compress) o encriptados.
- **Date**: fecha local de la operación. Las fechas deben incluir la zona horaria en que reside el sistema que genera la operación. Por ejemplo: Sunday, 29-Jul-2000 12:21:22 GMT+01. No existe un formato único en las fechas.
- **Pragma**: permite incluir información variada relacionada con el protocolo HTTP en el requerimiento o respuesta que se está realizando. Por ejemplo, un cliente envía un "Pragma: no-cache" para informar de que desea una copia nueva del recurso especificado.

Cabeceras sólo para peticiones del cliente

- **Accept**: campo opcional que contiene una lista de tipos MIME aceptados por el cliente. Se pueden utilizar * para indicar rangos de tipos de datos; tipo/* indica todos los subtipos de un determinado medio, mientras que */* representa a cualquier tipo de dato disponible.
- **Authorization**: clave de acceso que envía un cliente para acceder a un recurso de uso protegido o limitado. La información incluye el formato de autorización empleado, seguido de la clave de acceso propiamente dicha.
- **User-agent**: cadena que identifica el tipo y versión del cliente que realiza la petición. Por ejemplo, los browsers de Netscape envían cadenas del tipo "User-Agent: Mozilla/3.0 (WinNT; I)".

Cabeceras sólo para respuestas del servidor HTTP

- **Allow**: informa de los comandos HTTP opcionales que se pueden aplicar sobre el objeto al que se refiere esta respuesta. Por ejemplo "Allow: GET, POST".
- **Expires**: fecha de expiración del objeto enviado. Los sistemas de cache deben descartar las posibles copias del objeto pasada esta fecha. Por ejemplo, Expires: Thu, 12 Nov 2000 00:00:00 GMT+1. No todos los sistemas lo envían. Puede cambiarse utilizando un <META EXPIRES> en el encabezado de cada documento.
- **Location**: informa sobre la dirección exacta del recurso al que se ha accedido. Cuando el servidor proporciona un código de respuesta de la serie 3xx, este parámetro contiene la URL necesaria para accesos posteriores a este recurso.
- **Server**: cadena que identifica el tipo y versión del servidor HTTP. Por ejemplo "Server: NCSA 1.4".
- **WWW-Autenticate**: cuando se accede a un recurso protegido o de acceso restringido, el servidor devuelve un código de estado 401, y utiliza este campo para informar de los modelos de autentificación válidos para acceder a este recurso.

Códigos de estado del servidor

Ante cada transacción con un servidor HTTP, éste devuelve un código numérico que informa sobre el resultado de la operación, como primera línea del mensaje de respuesta. Estos códigos aparecen en algunos casos en la pantalla del cliente, cuando se produce un error. El formato de la línea de estado es:

Versión de protocolo HTTP utilizada Código numérico de estado(tres dígitos) Descripción del código numérico

Dependiendo del servidor, es posible que se proporcione un mensaje de error más elaborado, en forma de documento HTML, en el que se explican las causas del error y su posible solución.

Existen cinco categorías de mensajes de estado, organizadas por el primer dígito del código numérico de la respuesta:

- 1xx : mensajes informativos. Por ahora (en HTTP/1.0) no se utilizan, y están reservados para un futuro uso.
- 2xx : mensajes asociados con operaciones realizadas correctamente.
- 3xx : mensajes de redirección, que informan de operaciones complementarias que se deben realizar para finalizar la operación.
- 4xx : errores del cliente; el requerimiento contiene algún error, o no puede ser realizado.
- 5xx : errores del servidor, que no ha podido llevar a cabo una solicitud.

Los más comunes se recogen en la siguiente tabla:

Código	Comentario	Descripción
200	OK	Operación realizada satisfactoriamente.
201	Created	La operación ha sido realizada correctamente, y como resultado se ha creado un nuevo objeto, cuya URL de acceso se proporciona en el cuerpo de la respuesta. Este nuevo objeto ya está disponible.
202	Accepted	Igual que el anterior pero el nuevo objeto no está disponible por el momento. En el cuerpo de la respuesta se debe informar sobre la disponibilidad de la información.
204	No Content	La operación ha sido aceptada, pero no ha producido ningún resultado de interés.
301	Moved Permanently	El objeto al que se accede ha sido movido a otro lugar de forma permanente. El servidor proporciona, además, la nueva URL en la variable "Location" de la respuesta. Algunos navegadores acceden automáticamente a la nueva URL.
302	Moved Temporarily	Igual que el anterior pero el objeto al que se accede ha sido movido a otro lugar de forma temporal.
304	Not Modified	Cuando se hace un GET condicional, y el documento no ha sido modificado, se devuelve este código de estado.
400	Bad Request	La petición tiene un error de sintaxis y no es entendida por el servidor.
401	Unauthorized	La petición requiere una autorización especial, que normalmente consiste en un nombre y clave que el servidor verificará. El campo WWW-Autenticate informa de los protocolos de autentificación aceptados para este recurso.
403	Forbidden	Está prohibido el acceso a este recurso. No es posible utilizar una clave para modificar la protección.
404	Not Found	La URL solicitada no existe.
500	Internal Server Error	El servidor ha tenido un error interno, y no puede continuar con el procesamiento.
501	Not Implemented	El servidor no tiene capacidad, por su diseño interno, para llevar a cabo el requerimiento del cliente.
502	Bad Gateway	El servidor, que está actuando como "proxy" o pasarela, ha encontrado un error al acceder al recurso que había solicitado el cliente.
503	ServiceUnavailable	El servidor está actualmente deshabilitado, y no es capaz de atender el requerimiento.

Aunque HTTP está implementado sobre TCP/IP, no existe impedimento alguno para que pueda ser implementado para otro tipo de redes. Solamente hay que proyectar las estructuras de petición y respuesta a las unidades del transporte de datos de la red que se utilice.

3.4 Navegadores (browsers) para la WWW

Hemos visto que para acceder a servidores Web(protocolo HTTP) era necesario de disponer de los programas cliente adecuados. Estos programas se denominan browsers o navegadores.

EL WWW ganó su popularidad actual hasta finales de 1993 y principios de 1994, cuando la Versión de Mosaic para Microsoft Windows, (un navegador con interfaz gráfica para usuario [IGU]), fue desarrollada. Una Versión X (Unix) estaba disponible mucho antes, pero a causa de la gran base instalada de computadoras funcionando con Microsoft Windows, el WWW no experimentó el fenomenal crecimiento hasta que la Versión para Windows llegó a estar disponible. Debido a que NCSA se consolida públicamente, Mosaic estaba libremente disponible para cualquiera que quiera copiarlo(lo que también aumentó su popularidad). NCSA otorga licencia sobre Mosaic para poder ser distribuido o agregado a paquetes de programas comerciales tales como protocolos TCP/IP. NCSA también permite modificar el código fuente de Mosaic para que las compañías comerciales puedan escribir sus versiones propias de navegadores, usando la fuente de Mosaic para crear una versión de "valor-agregado". Actualmente, NCSA emplea una compañía comercial llamada Spyglass para autorizar Mosaic.

Siguiendo la explosión de popularidad de Mosaic, muchas compañías comerciales desarrollaron sus navegadores propios por lo que hoy hay muchos browsers de donde escoger. Uno que ha desafiado el liderazgo de Mosaic es el Navegante Netscape de Netscape Comunicaciones, los que reclutaron a Marc Andreessen, el creador del liderazgo de Mosaic, de NCSA. Netscape ofreció su primera versión (1.0N) de Netscape para Windows libre de cargo a usuarios individuales y organizaciones no-lucrativas. Netscape agregó características nuevas que Mosaic no tenía en ese momento, ejemplos tales como múltiples instancias de clientes y métodos más eficaces de transferencia y despliegue de datos. Estas características hicieron de Netscape muy popular.

Lista de algunos browsers disponibles:

Gratis o Pago Voluntario

- **Mosaic** - Disponible a través del Centro Nacional para Aplicaciones en Supercomputadora (Windows, Macintosh, Unix[X]).

- **Lynx** - Disponible a través de la Universidad de Kansas (Unix[X], Dos, Vax).

Comerciales

- **Netscape** - Disponible a través de Netscape Comunicaciones (Windows, Macintosh, Unix[X]) (copias de evaluación disponibles) (http://www.netscape.com).

- **Internet Explorer** - Disponible a través de Microsoft (Windows, Windows NT, Macintosh, Unix[X]) (http://www.microsoft.com).

- **Mosaic** Mejorado - Disponible a través de Spyglass, Inc. (Windows, Windows NT, Macintosh, Unix[X]) (http://www.spyglass.com/).

- **Hot Java** - Disponible a través de Sun Microsystems (provee soporte para animación) (SunOs y Solaris) (http://www.sun.com).

- **Webscape** - Disponible a través de Silicon Graphics, Inc., incluye soporte para el Lenguaje de Marcadores para Realidad Virtual (VRML) (SGI, SUN, Windows, etc.) (http://www.sgi.com).

4
Java Applets

Sun Microsystems, líder en servidores para Internet, cuyo lema desde hace mucho tiempo es "the network is the computer" (lo que quiere dar a entender que el verdadero computador es la red en su conjunto y no cada máquina individual), es quien ha desarrollado el lenguaje Java, en un intento de resolver simultáneamente todos los problemas que se le plantean a los desarrolladores de software por la proliferación de arquitecturas incompatibles, tanto entre las diferentes máquinas como entre los diversos sistemas operativos y sistemas de ventanas que funcionaban sobre una misma máquina, añadiendo la dificultad de crear aplicaciones distribuidas en una red como Internet.

En este capítulo se verán fundamentos del lenguaje de programación Java. Qué son y cómo se implementan los Applets de Java. Diseño de interfaces gráficas de usuario(GUI) con los paquetes de AWT y Swing. Y por último el esquema de seguridad en Java.

Hace algunos años, Sun Microsystems decidió intentar introducirse en el mercado de la electrónica de consumo y desarrollar programas para pequeños dispositivos electrónicos. Tras unos comienzos dudosos, Sun decidió crear una filial, denominada FirstPerson Inc., para dar margen de maniobra al equipo responsable del proyecto.

El mercado inicialmente previsto para los programas de FirstPerson eran los equipos domésticos: microondas, tostadoras y, fundamentalmente, televisión interactiva. Este mercado, dada la falta de pericia de los usuarios para el manejo de estos dispositivos, requería unas interfaces mucho más cómodas e intuitivas que los sistemas de ventanas que proliferaban en el momento.

Otros requisitos importantes a tener en cuenta eran la fiabilidad del código y la facilidad de desarrollo. James Gosling, el miembro del equipo con más experiencia en lenguajes de programación, decidió que las ventajas aportadas por la eficiencia de C++ no compensaban el gran costo de pruebas y depuración. Gosling había estado trabajando en su tiempo libre en un lenguaje de programación que él había llamado Oak, el cual, aún partiendo de la sintaxis de C++, intentaba remediar las deficiencias que iba observando.

Los lenguajes al uso, como C o C++, deben ser compilados para un chip, y si se cambia el chip, todo el software debe compilarse de nuevo. Esto encarece mucho los desarrollos y el problema es especialmente acusado en el campo de la electrónica de consumo. La aparición de un chip más barato y, generalmente, más eficiente, conduce inmediatamente a los fabricantes a incluirlo en las nuevas series de sus cadenas de producción, por pequeña que sea la diferencia en precio ya que, multiplicada por la tirada masiva de los aparatos, supone un ahorro considerable. Por tanto, Gosling decidió mejorar las características de Oak y utilizarlo.

El primer proyecto en que se aplicó este lenguaje recibió el nombre de proyecto Green y consistía en un sistema de control completo de los aparatos electrónicos y el entorno de un hogar. Para ello se construyó una computadora experimental denominada *7 (Star Seven). El sistema presentaba una interfaz basada en la representación de la casa de forma animada y el control se llevaba a cabo mediante una pantalla sensible al tacto. En el sistema aparecía Duke, la actual mascota de Java. Pero el proyecto Green tuvo algunas dificultades. El comercio para dispositivos electrónicos de consumo inteligentes no crecía tan rápido como Sun lo había anticipado. Peor aún, un gran contrato para el cual Sun competía fue ganado por otra compañía.

Una vez que en Sun se dieron cuenta de que a corto plazo la televisión interactiva no iba a ser un gran éxito, urgieron a FirstPerson a desarrollar con rapidez nuevas estrategias que produjeran beneficios. No lo consiguieron y FirstPerson cerró en la primavera de 1994.

Pese a lo que parecía ya un olvido definitivo, Bill Joy, cofundador de Sun y uno de los desarrolladores principales del Unix de Berkeley, juzgó que Internet podría llegar a ser el campo de juego adecuado para disputar a Microsoft su primacía casi absoluta en el terreno del software, y vio en Oak el instrumento idóneo para llevar a cabo estos planes. Se descubrió entonces que ya había un lenguaje para computadoras llamado Oak. Cuando un grupo de gente de Sun visitó un lugar de café local, el nombre de "Java" fue sugerido y permaneció.

El World Wide Web estalló en popularidad en 1993 y la gente de Sun vio el potencial inmediato de utilizar Java para crear páginas de Web con el llamado contenido dinámico. Esto dio nueva vida al proyecto.

Tras modificar partes del diseño, el lenguaje Java fue presentado en agosto de 1995. Java generó interés inmediato en la comunidad de negocios por el interés en el WWW. Java no es un lenguaje académico como Pascal o un lenguaje diseñado por una persona o por un pequeño grupo para su uso local como C o C++. Java fue diseñado por razones comerciales para el desarrollo de Internet.

4.1 Fundamentos de Java

Funcionamiento de la JVM (Java Virtual Machine)

Una máquina virtual corre en varios sistemas operativos y procesadores y provee un ambiente de ejecución uniforme. Una máquina virtual es una emulación por software de un CPU que no tiene contraparte física.

El tipo de máquina virtual en la cual nosotros estamos interesados es una máquina virtual que ejecuta Java byte-code. La JVM ejecuta Java bytecode y ejecuta operaciones específicas del sistema en favor de una aplicación de Java.

Características

Las características principales que nos ofrece Java respecto a cualquier otro lenguaje de programación, son:

Simple

Java ofrece la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de éstos. C++ es un lenguaje que adolece de falta de seguridad, pero C y C++ son lenguajes más difundidos, por ello Java se diseñó para ser parecido a C++ y así facilitar un rápido y fácil aprendizaje.

Java reduce en un 50% los errores más comunes de programación con lenguajes como C y C++ al eliminar muchas de las características de éstos. Por su diseño Java es fácil de aprender. Uno debe darse cuenta de que muchas de las dificultades iniciales en aprender lenguajes orientados a objetos es actualmente causado por los conceptos bases de programación orientada a objetos, no la sintaxis básica del lenguaje así mismo.

Portable

Más allá de la portabilidad básica por ser de arquitectura independiente, Java implementa otros estándares de portabilidad para facilitar el desarrollo. Los enteros son siempre enteros y además, enteros de 32 bits en complemento a 2. Además, Java construye sus interfaces de usuario a través de un sistema abstracto de ventanas de forma que las ventanas puedan ser implantadas en entornos Unix, Pc o Mac.

Interpretado

El intérprete Java (sistema run-time) puede ejecutar directamente el código objeto. Enlazar (linkar) un programa, normalmente, consume menos recursos que compilarlo, por lo que los desarrolladores con Java pasarán más tiempo desarrollando y menos esperando por la computadora. No obstante, el compilador actual del JDK(Java Development Kit, entorno de desarrollo de Java) es lento. Por ahora, que todavía no hay compiladores específicos de Java para las diversas plataformas, Java es más lento que otros lenguajes de programación, como C++, ya que debe ser interpretado y no ejecutado como sucede en cualquier programa tradicional.

Se dice que Java es de 10 a 30 veces más lento que C, y que tampoco existen en Java proyectos de gran envergadura como en otros lenguajes. La verdad es que ya hay comparaciones ventajosas entre Java y el resto de los lenguajes de programación, y una gran cantidad de folletos electrónicos en favor y en contra de los distintos lenguajes contendientes con Java. Lo que se suele dejar de lado en todo esto, es que primero habría que decidir hasta que punto Java, un lenguaje en pleno desarrollo y todavía sin definición definitiva, está maduro como lenguaje de programación para ser comparado con otros; como por ejemplo con Smalltalk, que lleva más de 20 años en el mercado.

La verdad es que Java para conseguir ser un lenguaje independiente del sistema operativo y del procesador que incorpore la máquina utilizada, es tanto interpretado como compilado. El código fuente escrito con cualquier editor se compila generando el byte-code. Este código intermedio es de muy bajo nivel, pero sin alcanzar las instrucciones máquina propias de cada plataforma. El byte-code corresponde al 80% de las instrucciones de la aplicación.

Ese mismo código es el que se puede ejecutar sobre cualquier plataforma. Para ello hace falta el run-time, que sí es completamente dependiente de la máquina y del sistema operativo, que interpreta dinámicamente el byte-code y añade el 20% de instrucciones que faltaban para su ejecución. Con este sistema es fácil crear aplicaciones multiplataforma, pero para ejecutarlas es necesario que exista el run-time correspondiente al sistema operativo utilizado.

Arquitectura Neutral

Para establecer Java como parte integral de la red, el compilador Java compila su código a un archivo objeto de formato independiente de la arquitectura de la máquina en que se ejecutará. Cualquier máquina que tenga el sistema de ejecución (run-time) puede ejecutar ese código objeto, sin importar en modo alguno la máquina en que ha sido generado. Actualmente existen sistemas run-time para Solaris 2.x, SunOs 4.1.x, Windows 95 y 98, Windows NT, Linux, Irix, Aix, Mac, Apple y probablemente haya grupos de desarrollo trabajando en el porting a otras plataformas.

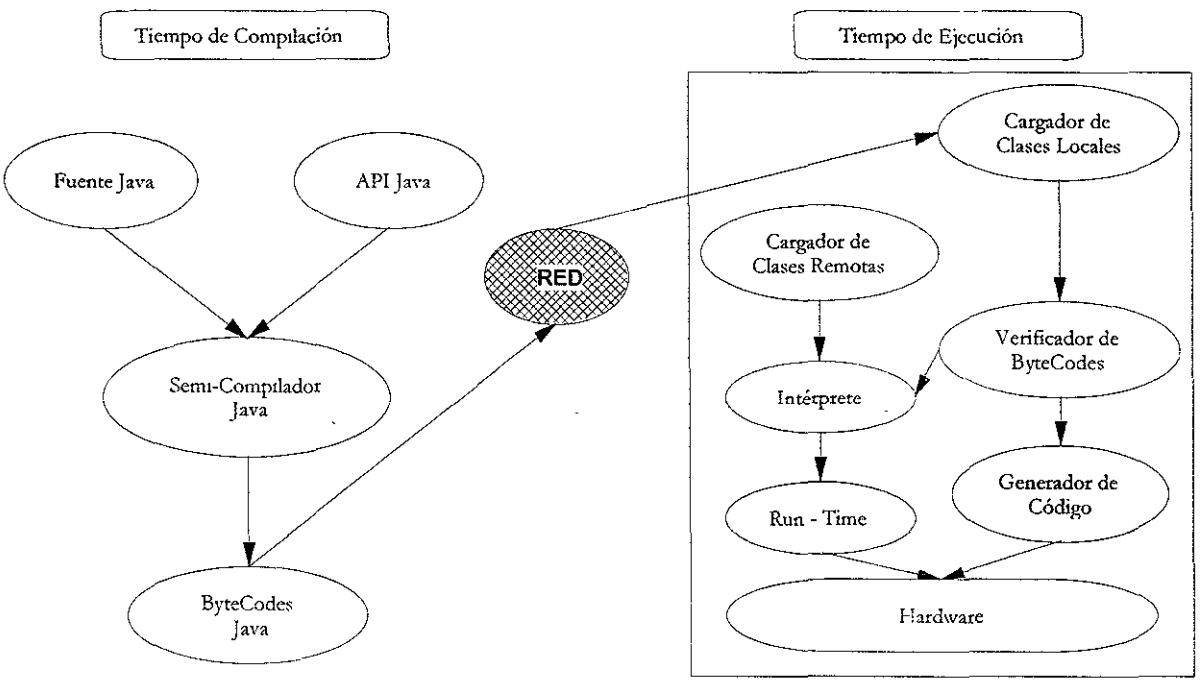


Figura 4.1 Arquitectura de Java

El código fuente Java se "compila" a un código de bytes de alto nivel independiente de la máquina. Este código (byte-codes) está diseñado para ejecutarse en una máquina hipotética que es implementada por un sistema run-time, que sí es dependiente de la máquina.

Hay APIs de Java que también entran en contacto directo con el hardware y serán dependientes de la máquina, como ejemplo de este tipo de APIs podemos citar:

Java 2D: gráficos 2D y manipulación de imágenes
Java Media Framework : Elementos críticos en el tiempo: audio, video...
Java Animation: Animación de objetos en 2D
Java Share: Interacción entre aplicaciones multiusuario
Java 3D: Gráficos 3D y su manipulación

Orientado a Objetos

Java implementa la tecnología básica de C++ con algunas mejoras y elimina algunas cosas para mantener el objetivo de la simplicidad del lenguaje. Java trabaja con sus datos como objetos y con interfaces a esos objetos. Soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y polimorfismo. Las plantillas de objetos son llamadas, como en C++, clases y sus copias, instancias. Estas instancias, como en C++, necesitan ser construidas y destruidas en espacios de memoria.

Java incorpora funcionalidades inexistentes en C++ como por ejemplo, la resolución dinámica de métodos. En C++ se suele trabajar con librerías dinámicas (DLLs) que obligan a recompilar la aplicación cuando se retocan las funciones que se encuentran en su interior. Este inconveniente es resuelto por Java mediante una interfaz específica llamada RTTI (RunTime Type Identification) que define la interacción entre objetos excluyendo variables de instancias o implementación de métodos. Las clases en Java tienen una representación en el run-time que permite a los programadores interrogar por el tipo de clase y enlazar dinámicamente la clase con el resultado de la búsqueda.

Distribuido

Java se ha construido con extensas capacidades de interconexión TCP/IP. Existen librerías de rutinas para acceder e interactuar con protocolos como http y ftp. Esto permite a los programadores acceder a la información a través de la red con tanta facilidad como a los archivos locales.

La verdad es que Java en sí no es distribuido, sino que proporciona las librerías y herramientas para que los programas puedan ser distribuidos, es decir, que se corran en varias máquinas, interactuando. Java tiene la habilidad de tratar objetos localizados a través de la red como si ellos fueran locales. Utilizando una llamada estándar RMI (remote method invocation - invocación de métodos remotos).

Robusto

Java realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. La comprobación de tipos en Java ayuda a detectar errores, lo antes posible, en el ciclo de desarrollo. Java obliga a la declaración explícita de métodos, reduciendo así las posibilidades de error. Maneja la memoria para eliminar las preocupaciones por parte del programador de la liberación o corrupción de memoria. También implementa los arrays auténticos, en vez de listas enlazadas de punteros, con comprobación de límites, para evitar la posibilidad de sobre-escribir o corromper memoria resultado de punteros que señalan a zonas equivocadas. Estas características reducen drásticamente el tiempo de desarrollo de aplicaciones en Java.

Además, para asegurar el funcionamiento de la aplicación, realiza una verificación de los byte-codes, que son el resultado de la compilación de un programa Java. Es un código de máquina virtual que es interpretado por el intérprete Java. No es el código máquina directamente entendible por el hardware, pero ya ha pasado todas las fases del compilador: análisis de instrucciones, orden de operadores, etc., y ya tiene generada la pila de ejecución de órdenes.

Dinámico

Java se beneficia todo lo posible de la tecnología orientada a objetos. Java no intenta conectar todos los módulos que comprenden una aplicación hasta el tiempo de ejecución. Las librería nuevas o actualizadas no paralizarán las aplicaciones actuales (siempre que mantengan el API anterior).

Java también simplifica el uso de protocolos nuevos o actualizados. Si su sistema ejecuta una aplicación Java sobre la red y encuentra una pieza de la aplicación que no sabe manejar, Java es capaz de traer automáticamente cualquiera de esas piezas que el sistema necesita para funcionar.

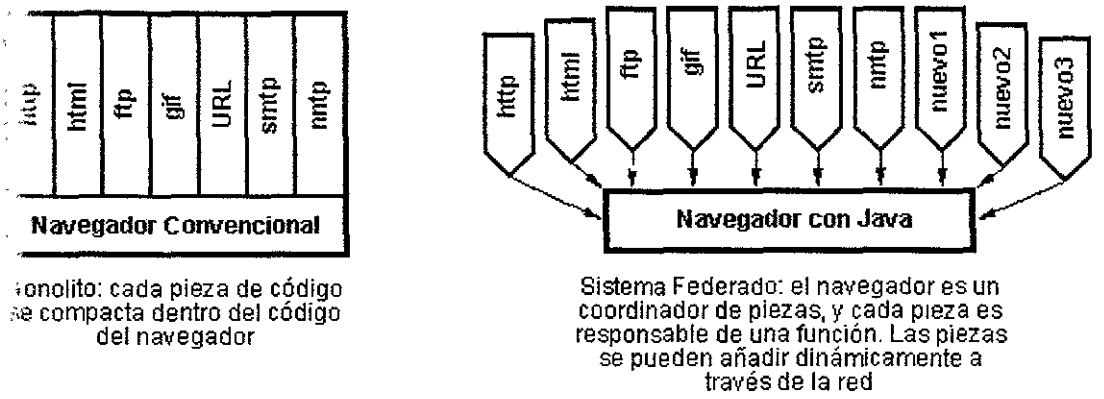


Figura 4.2 Java es dinámico

Java, para evitar que los módulos de byte-codes o los objetos o nuevas clases, haya que estar trayéndolos de la red cada vez que se necesiten, implementa las opciones de persistencia, para que no se eliminen cuando se limpie la caché de la máquina.

Seguro

Dada la naturaleza distribuida de Java, donde las applets se bajan desde cualquier punto de la red, la seguridad se impuso como una necesidad de vital importancia. A nadie le gustaría ejecutar en su computadora programas con acceso total a su sistema, procedentes de fuentes desconocidas. Así que se implementaron barreras de seguridad en el lenguaje y en el sistema de ejecución en tiempo real.

Multihilos (multithreaded)

Al ser multithreaded (multihilos, en mala traducción), Java permite muchas actividades simultáneas en un programa. Los threads (a veces llamados, procesos ligeros), son básicamente pequeños procesos o piezas independientes de un gran proceso. Al estar los threads construidos en el lenguaje, son más fáciles de usar y más robustos que sus homólogos en C o C++.

El beneficio de ser multithreaded consiste en un mejor rendimiento interactivo y mejor comportamiento en tiempo real. Aunque el comportamiento en tiempo real está limitado a las capacidades del sistema operativo subyacente (Unix, Windows, etc.), supera a los entornos de flujo único de programa (single-threaded) tanto en facilidad de desarrollo como en rendimiento.

Cualquiera que haya utilizado la tecnología de navegación concurrente, sabe lo frustrante que puede ser esperar por una gran imagen que se está trayendo. En Java, las imágenes se pueden ir trayendo en un thread independiente, permitiendo que el usuario pueda acceder a la información en la página sin tener que esperar por el navegador.

Producir applets

Java puede ser usado para crear dos tipos de programas: aplicaciones independientes y applets. Las aplicaciones independientes se comportan como cualquier otro programa escrito en cualquier lenguaje, como por ejemplo el navegador de Web HotJava, escrito íntegramente en Java. Por su parte, las applets son pequeños programas que aparecen embebidos(incluidos) en las páginas Web, como aparecen los gráficos o el texto, pero con la capacidad de ejecutar acciones muy complejas, como animar imágenes, establecer conexiones de red, presentar menús y cuadros de diálogo para luego emprender acciones, etc.

Versiones

- JDK 1.0.x : Primeras versiones, incluyen los principales módulos básicos de Java: i/o, red, applets, utilerías, awt, etc.

- JDK 1.1.x : El manejo de eventos en awt es distinto. Se incluyen: RMI's, JavaBeans, jar's, JNI's(con stubs-fragmentos). Primeras versiones de Swing como una ampliación de awt.

- JDK 1.2 : Cambia el manejo de JNI(sin stubs). Necesario para compilar Java3D. Se formaliza Swing.

Creación de Paquetes

Un paquete es utilizado para indicar que una clase pertenece a un grupo de clases. Utilizado para evitar la sobre - población de clases(todas las clases en un mismo directorio). Un paquete es un conjunto de clases. Un paquete sirve para organizar clases.

Java incluye varios paquetes dependiendo de la versión siendo los más importantes y que utilizaremos dentro del proyecto :

java.applet Aplicación de Java para Internet: Applet, AudioClip, etc.
java.awt (Abstract Window Toolkit) Familia para desplegar objetos, define componentes gráficos: Color, Canvas, Frame, Graphics, Button, Panel, Window, etc.
javax.swing Nueva familia de componentes gráficos que mejora al awt.
java.io Manejo de Entradas/Salidas. File, OutputStream, InputStream, StreamTokenizer, etc.
java.lang Incluye lo básico: System, Short, Object, Void, etc. Es el default (no es necesario poner java.lang.System).
java.net Para aplicaciones de red: DatagramPacket, Socket, URL, etc.
java.rmi Para invocación de métodos remotos (aplicación en cómputo distribuido): Naming, Remote, ServerError, etc.
java.security Para seguridad: Acl, Key, PrivateKey, Signer, etc.
java.util Conjunto de utilerías: Hashtable, Stack, Random, Calendar, Vector, etc.

4.2 Java Applets y el World Wide Web

La definición más extendida indica que un applet es "una pequeña aplicación accesible en un servidor Internet, que se transporta por la red, se instala automáticamente y se ejecuta localmente como parte de un documento web". Un applet es una aplicación pretendidamente corta(nada impide que ocupe más de un gigabyte, a no ser por el pensamiento de que se va a transportar por la red) basada en un formato gráfico sin representación independiente: es decir, se trata de un elemento que se agrega a otras aplicaciones; es un componente en su sentido estricto.

Lo que ocurre es que, dado que no existe una base adecuada para soportar aplicaciones industriales Java en las cuales insertar nuestras mini-aplicaciones, los applets se han construido mayoritariamente, y con gran acierto comercial, como pequeñas aplicaciones interactivas, con movimiento, luces y sonido en Internet.

Actualmente hay un auge sorprendente de la navegación web. Los documentos web pueden contener variedad de texto, gráficos de todas clases y proporcionar enlaces hipertexto hacia cualquier lugar de la red. Los navegadores utilizan documentos escritos en lenguaje HTML. La combinación actual de navegadores HTML/WWW están limitados a texto y gráficos. Si se quiere reproducir un sonido o ejecutar un programa de demostración, primero tenemos que bajarnos(download) el archivo en cuestión y luego utilizar un programa en nuestro computador capaz de entender el formato de ese archivo, o bien cargar un módulo(Plug-in) en nuestro navegador para que pueda interpretar el archivo que hemos bajado.

Hasta hace poco, la única forma de realizar una página web con contenido interactivo, era mediante la interfaz CGI (Common Gateway Interface), que permite pasar parámetros entre formularios definidos en lenguaje HTML y programas escritos en Perl o en C. Esta interfaz resulta muy incómoda de programar y es pobre en sus posibilidades.

El lenguaje Java y los navegadores con soporte Java, proporcionan una forma diferente de hacer que ese navegador sea capaz de ejecutar programas. Con Java se puede reproducir sonido directamente desde el navegador, se pueden visitar páginas con animaciones, se puede enseñar al navegador a manejar nuevos formatos de archivos, e incluso, transmitir vídeo por las líneas telefónicas.

Utilizando Java, se pueden eliminar los inconvenientes de la interfaz CGI y también se pueden añadir aplicaciones que vayan desde experimentos científicos interactivos de propósito educativo a juegos o aplicaciones especializadas. Es posible implementar publicidad interactiva y periódicos personalizados. Por ejemplo, alguien podría escribir un programa Java que implementara una simulación química interactiva. Utilizando un navegador con soporte Java, un usuario podría recibir fácilmente esa simulación e interaccionar con ella, en lugar de conseguir simplemente un dibujo estático y algo de texto. Lo recibido cobra vida. Además, con Java podemos estar seguros de que el código que hace funcionar el experimento químico no contiene ningún trozo de código malicioso que dañe al sistema. El código que intente actuar destructivamente o que contenga errores, no podrá traspasar los muros defensivos colocados por las características de seguridad y robustez de Java.

Además, los Applets proporcionan una nueva forma de acceder a las aplicaciones. El software viaja transparentemente a través de la red. No hay necesidad de instalar las aplicaciones, ellas mismas vienen cuando se necesitan. Por ejemplo, la mayoría de los navegadores del Web pueden procesar un reducido número de formatos gráficos (típicamente GIF y JPEG).

Si se encuentran con otro tipo de formato, el navegador estándar no tiene capacidad para procesarlo, tendría que ser actualizado para poder aprovechar las ventajas del nuevo formato. Sin embargo, un navegador con soporte Java puede enlazar con el servidor que contiene el algoritmo que procesa ese nuevo formato y mostrar la imagen. Por lo tanto, si alguien inventa un nuevo algoritmo de compresión para imágenes, el inventor sólo necesita estar seguro de que hay una copia en código Java de ese algoritmo instalada en el servidor que contiene las imágenes que quiere publicar. Es decir, los navegadores con soporte Java se actualizan a sí mismos sobre la marcha, cuando encuentran un nuevo tipo de archivo o algoritmo.

Funcionamiento de Java Applets

El archivo de código fuente puede ser escrito mediante cualquier editor ascii convencional o, para mayor comodidad, con el editor suministrado con el paquete del lenguaje Java(según la versión o herramienta utilizada para desarrollo programas de Java). Una vez creado el archivo .java con el código del programa, se compila, generándose un archivo intermedio con los byte-codes, de extensión .class. Una vez generado el archivo .class, éste ya puede ser interpretado en cualquier máquina virtual de Java.

Un applet es una mínima aplicación Java diseñada para ejecutarse en un navegador Web. Por tanto, no necesita preocuparse por un método main() ni en dónde se realizan las llamadas. El applet asume que el código se está ejecutando desde dentro de un navegador. Espera como argumento el nombre del archivo HTML que debe cargar, no se le puede pasar directamente un programa Java. Este archivo HTML debe contener una marca especial ó tag de HTML que especifica el código que cargará el navegador. Este archivo HTML debe contener el siguiente texto:

```
<APPLET CODE="HolaMundo.class" WIDTH=150 HEIGHT=25>
</APPLET>
```

El navegador crea un espacio de navegación, incluyendo un área gráfica, donde se ejecutará el applet, entonces llamará a la clase applet apropiada. La página Web en la que se encuentra embebido el applet se habrá bajado desde la red o bien lo podemos cargar desde el disco duro. En cualquier caso, para ver la applet funcionando hace falta un navegador capaz de ejecutar Java, como los ya descritos anteriormente Netscape y Microsoft Internet Explorer.

El navegador carga las clases especificadas en la etiqueta <applet> dinámicamente, a medida que van siendo necesitadas y se les pasa al código fuente (a los byte-codes) el verificador de código de bytes.

Ahora será ya la máquina virtual de Java la que vaya interpretando los byte-codes y generando las instrucciones para su propia arquitectura.

Ejemplo : Applet

Para entender bien como funciona un applet de Java tenemos el siguiente ejemplo:

1. Existe un código de Java en un servidor de Web. (Los códigos de Java se caracterizan por tener la extensión *.class).
2. Una persona en Internet, con un browser compatible con Java, realiza una conexión al servidor. Por ejemplo al URL:
   http://ww.mcc.unam.mx /~avelar/Hola.html
3. El servidor envía el documento HTML y el código en Java (*.class).
4. En la computadora del usuario remoto llegan el código byte-code y el documento HTML. La Máquina Virtual de Java, que está en el browser, transforma el código Java en un código que entienda la máquina local. Se ejecuta el programa dentro de la página de Web en la maquina local.
5. Si el usuario realiza otra conexión a otro URL o se sale del browser, el programa se deja de ejecutar y en la computadora no queda rastro de él.

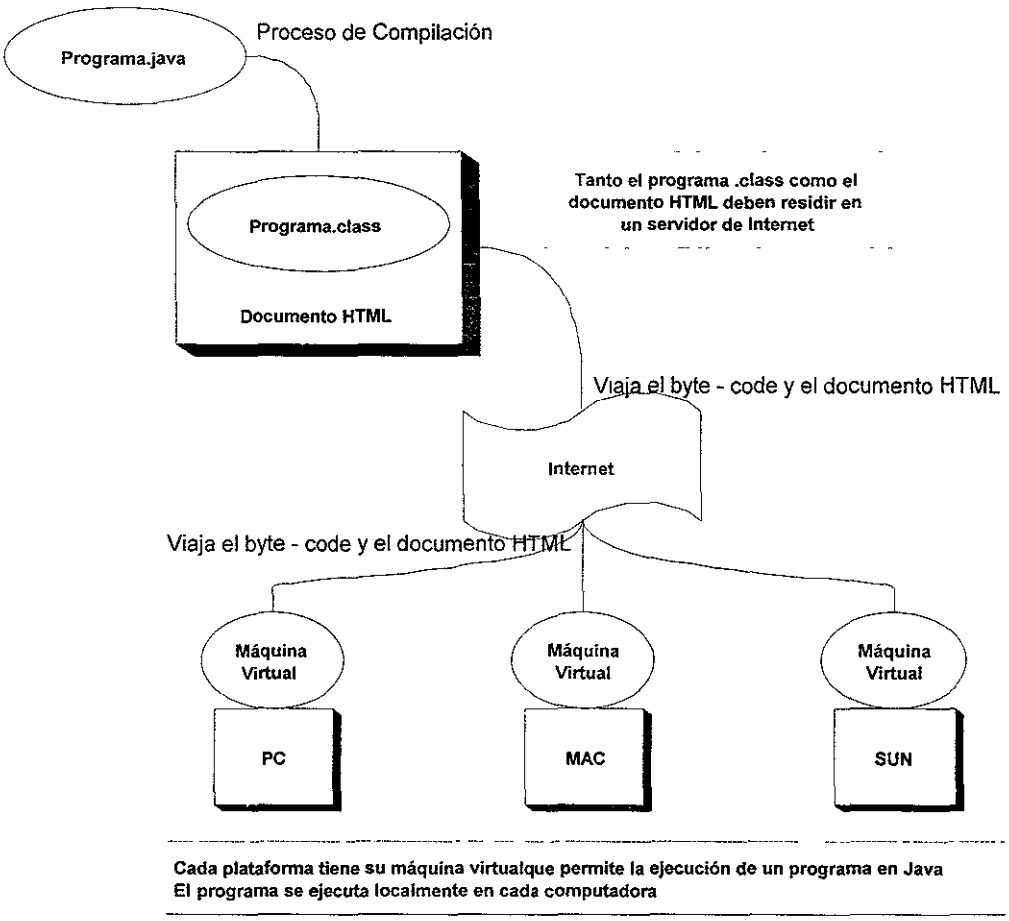


Figura 4.3 Esquema de funcionamiento de los Java Applets

Ciclo de vida de un Applet

Cuando un applet se carga, comienza su ciclo de vida, que pasaría por las siguientes fases:

- **init()**
  Inicializa los recursos que va a solicitar el applet, y los elementos que pueda contener, tales como botones, campos de texto, cajas de verificación, etc. Este método se ejecuta cada vez que el applet se carga.
- **start()**
  Comienza la ejecución del applet después de haberse inicializado, o cuando se vuelve a visitar la página que contiene al applet.
- **stop()**
  Detiene la ejecución del applet.
- **destroy()**
  Desaloja los recursos solicitados por el applet.

Qué puede y qué no puede hacer un Applet

Por razones de seguridad, un applet que se ha cargado a través de la red tiene muchas restricciones de seguridad. Una de ellas es que normalmente no puede leer o escribir archivos en la computadora donde se está ejecutando. Otra es que un applet no puede hacer conexiones a través de la red excepto hacia el *host*(equipo o computadora) del que ha venido. A pesar de estas restricciones los applets pueden hacer algunas cosas que no se esperan. Por ejemplo, un applet puede llamar a los métodos públicos de otro applet que esté en la misma página.

4.3 Java Gui´s

Casi todos los applets tienen una interfaz gráfico de usuario (GUI). UI (user interface) es un término que se refiere a todos los caminos de comunicación entre un programa y sus usuarios. UI no es sólo lo que ve el usuario, también es lo que el usuario oye y siente. Incluso la velocidad con la que un programa interactua con el usuario es una parte importante del UI del programa.

El entorno Java proporciona clases para las siguientes funcionalidades del UI:

- Presentar un UI gráfico (GUI)
  Este es el UI preferido por la mayoría de los programas Java. El resto de esta sección se concentra sobre este punto.

- Ejecutar Sonidos
  Justo ahora, los applets pueden ejecutar sonidos, pero las aplicaciones no pueden (al menos de una forma portable).

- Obtener información de configuración
  Los usuarios pueden configurar la información del applet utilizando argumentos de la línea de comandos (sólo las aplicaciones) y parámetros (sólo los applets).

- Grabar las preferencias del usuario utilizando propiedades
  Para la información que las aplicaciones necesitan guardar cuando no se están ejecutando, puedes utilizar las propiedades. Normalmente los applets no pueden escribir propiedades en el sistema local de archivos, debido a las restricciones de seguridad.

- Obtener y mostrar texto utilizando los canales de entrada, salida y error estándar
  Los canales de entrada, salida y error estándar son un forma al viejo estilo de presentar una interfaz de usuario.

Los applets y las aplicaciones presentan información al usuario y le invitan a interactuar utilizando un GUI. La parte del entorno Java llamada Herramientas de Ventanas Abstractas (Abstract Windows Toolkit - AWT) contiene un completo conjunto de clases para escribir programas GUI.

AWT

AWT es el acrónimo del Abstract Window Toolkit para Java. Se trata de una biblioteca de clases Java para el desarrollo de Interfaces de Usuario Gráficas. La versión del AWT que Sun proporciona con el JDK se desarrolló en sólo dos meses y es la parte más débil de todo lo que representa Java como lenguaje. El entorno que ofrece es demasiado simple, no se han tenido en cuenta las ideas de entornos gráficos novedosos, sino que se ha ahondado en estructuras orientadas a eventos, llenas de callbacks y sin soporte alguno del entorno para la construcción gráfica; la simple acción de colocar un dibujo sobre un botón se vuelve una tarea bastante complicada.

La estructura básica del AWT se basa en Componentes y Contenedores. Estos últimos contienen Componentes posicionados a su respecto y son Componentes a su vez, de forma que los eventos pueden tratarse tanto en Contenedores como en Componentes, corriendo por cuenta del programador el encaje de todas las piezas, así como el tratamiento de los eventos adecuados. Existen herramientas de composición visual que pueden ahorrar el trabajo de fijar los componentes. Sin embargo, los detalles del diseño se realizan todavía a mano.

El AWT proporciona muchos componentes GUI estándar, como botones, listas, menús y áreas de texto. También incluye contenedores (como ventanas y barras de menú) y componentes de alto nivel (cómo un cuadro de diálogo para abrir y guardar archivos).

Otras clases del AWT incluyen aquellas que trabajan en un contexto gráfico (incluyendo las operaciones de dibujo básico), imágenes, eventos, fuentes y colores. Otro grupo importante de clases del AWT son los controladores de distribución o disposición que controlan el tamaño y la posición de los componentes.

Utilizar los Componentes del AWT

La figura 4.4 muestra el árbol de herencia para todas las clases componentes del AWT.

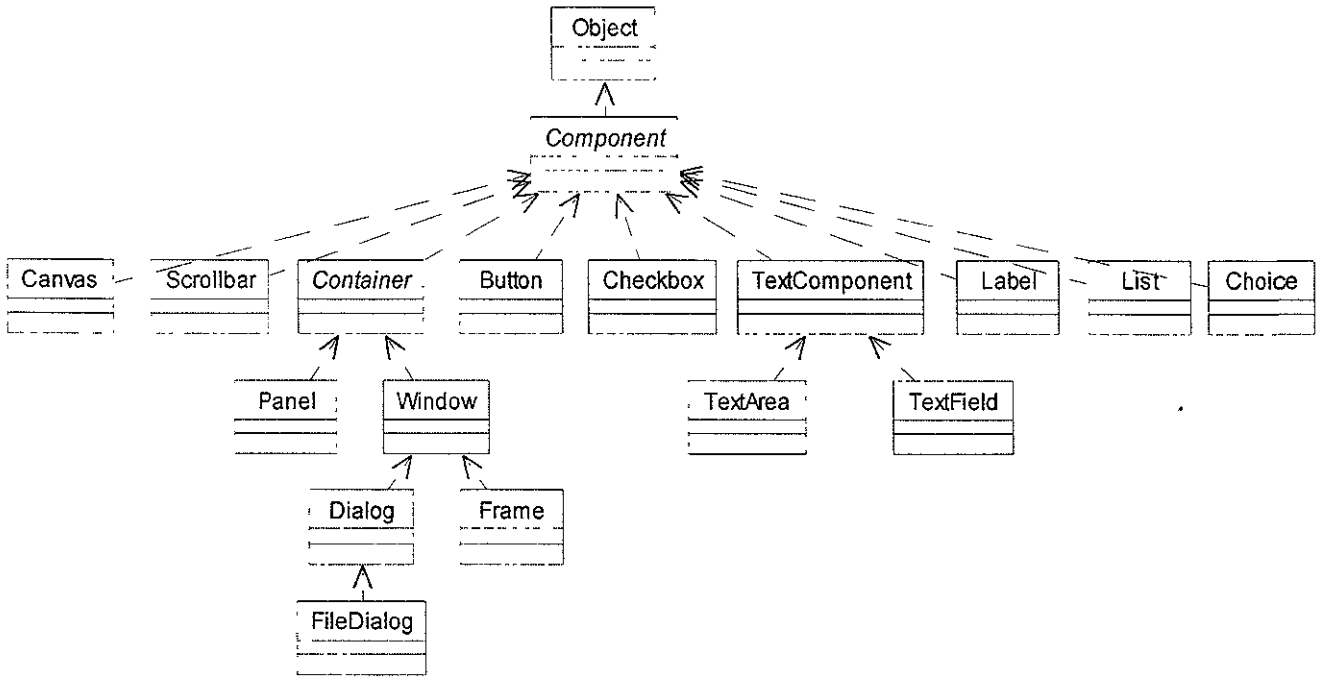


Figura 4.4 Arbol de herencia de componentes AWT

Como muestra la figura, todos los componentes, excepto los componentes relacionados con los menús, descienden de la clase Component del AWT. A causa de las restricciones de las distintas plataformas (como la imposibilidad de seleccionar el color de fondo de los menús), todos los componentes relacionados con los menús se han sacado fuera de la clase Component. En su lugar, los componentes de menús descienden de la clase MenuComponent del AWT.

Swing

Swing es un componente principal de las JFC(Java Foundation Classes), lo cual es el resultado de un esfuerzo de colaboración muy grande entre Sun, Netscape, IBM y otras empresas. Swing proporciona gran cantidad de controles GUI muy útiles, que ofrecen la posibilidad de cambiar fácil y rápidamente el aspecto y sensación(L&F) de un único componente o grupo de componentes.

Aunque Swing esté separado de AWT, se implementa en términos de clases AWT básicas. El AWT proporciona la interfaz entre el sistema de ventanas nativo subyacente y los componentes GUI de Java. Swing utiliza esta interfaz, pero no se apoya en componentes del AWT para hacer uso de objetos nativos. En lugar de ello, los componentes de Swing están escritos en Java puro. Esto ofrece ventajas significativas. Permite a los componentes ser independientes del sistema de ventanas nativo, lo cual implica que pueden ejecutarse en cualquier sistema de ventanas que admita el AWT. También permite a los componentes ser independientes de cualquier limitación de los sistemas de ventanas nativos. Esta independencia permite a Swing controlar y adaptar su aspecto y sensación.

El API utilizado para el proyecto es Swing 1.1, que se proporciona con la versión JFC 1.1. Para escribir programas utilizando componentes Swing, lo primero que se debe hacer es descargar las versiones apropiadas del JDK y del JFC.

Swing funciona tanto con el JDK 1.1 como con el JDK 1.2. Como regla general se debería descargar la última versión de JDK 1.1 ó de JDK 1.2.

Si se está utilizando la versión 1.1 del JDK, se necesita descargar la última versión del JFC 1.1. Si estamos utilizando el JDK 1.2 no es necesario instalar el JFC ya que está incluido en la versión del JDK 1.2. Desafortunadamente los navegadores comerciales como Netscape y Internet Explorer no soportan aun el Swing para el JDK 1.2. Por lo que, para el proyecto de la aplicación distribuida se utilizará JDK 1.1.7 y las JFC 1.1.

Reglas generales para el uso de componentes Swing

Cuando se construya un GUI que utilice componentes Swing, se debería evitar la utilización de componentes pesados que no sean Swing, y se deberían poner los componentes Swing dentro de un contenedor Swing de alto nivel.

El paquete Swing define dos tipos de componentes:

- Contenedores de alto nivel (JFrame, JApplet, JWindow, JDialog)
- Componentes de peso ligero (Jcualquier cosa, como JButton,JPanel, y JMenu)

Los contenedores de alto nivel proporcionan el marco de trabajo en el que existen los componentes de peso ligero. Específicamente, un contenedor de alto nivel Swing proporciona un área en la que los componentes de peso ligero Swing se dibujan a sí mismos. Los contenedores de alto nivel Swing también proporcionan otras características Swing como un espacio para la barra de menús, ampliar el manejo de eventos y de pintado, y soporte de accesibilidad.

Aquí se puede ver un árbol de herencia GUI para una programa Swing Applet típico que implementa una ventana que contiene dos botones, un campo de texto y una lista:

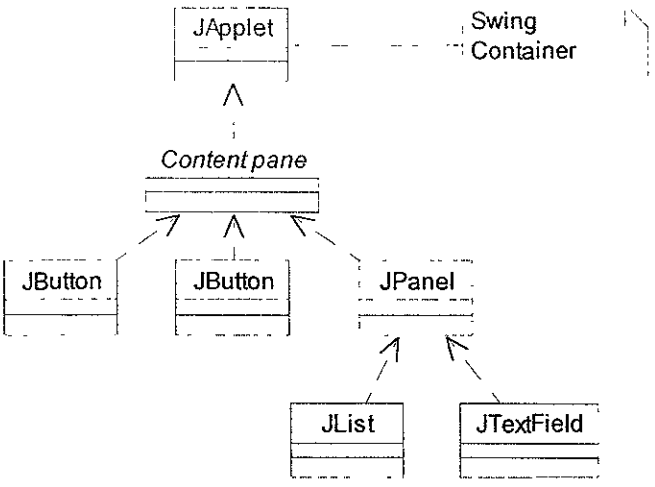


Figura 4.5 Arbol de herencia de una aplicación típica Swing

Problemas de los Java Applets

Existen varios problemas que se pueden presentar a la hora de realizar una aplicación que utilice Applets, pero en particular las más importantes son:

Si los Applets aparecen en una ventana ya existente del navegador. Al contrario que las aplicaciones basadas en GUI, los applets no tienen que crear una ventana independiente para mostrarse. Pueden hacerlo si tienen una buena razón, pero normalmente sólo se muestran en la ventana del navegador.

Las clases Applet (y todos los archivos de datos que éstas utilizan) son cargados a través de la red, lo que puede ser bastante lento. Los Applets pueden hacer muchas cosas para reducir el tiempo de arranque. La subclase Applet puede ser una clase pequeña que muestre inmediatamente un mensaje de estado. Y, si algunas de las clases del applet o los datos no se utilizan por el momento, el applet puede precargar las clases o los datos en un thread en segundo plano.

4.4 Seguridad en Java

¿Por qué tanto revuelo acerca de la seguridad en Java? ¿Por qué no se habla con la misma aprensión de seguridad en C o seguridad en Perl?

La respuesta no es difícil de entender: los programas en C, por ejemplo, están compilados para una arquitectura determinada (un programa compilado para Unix no funciona en Mac o en Windows) y es el usuario el que los ejecuta en su computadora (bien descargados desde la red, desde un disco flexible o un CD-ROM) bajo su entera responsabilidad y sabiendo cuándo los ejecuta.

Sin embargo, la característica más atractiva de Java desde el punto de vista de Internet, son sus applets, que se pueden ejecutar en cualquier plataforma con cualquier sistema operativo, pero constituyen también su talón de Aquiles. Si desde cualquier página Web visitada se puede enviar un applet que se ejecutará en la computadora sin nuestro conocimiento, todo tipo de ataques podría tener lugar: esos applets podrían cifrar el contenido del disco duro y luego el autor pedir una suma millonaria por entregar la clave; podrían introducir virus; podrían robar todo tipo de información de la computadora; podrían explotar recursos de nuestro sistema, como ciclos de CPU, y un largo etcétera fácil de imaginar.

Con el fin de que estos escenarios no fueran posibles, se implementó un sistema de seguridad integrado por cuatro líneas de defensa. Se describirán los ataques y las formas de defensa, se verán qué son y cómo se escriben applets hostiles y cómo protegerse de ellos.

El modelo de las cuatro capas

El modelo de seguridad de Java se conoce como modelo del patio de juegos (Sandbox model), aludiendo a esos rectángulos con arena donde se deja jugando a los niños pequeños, de manera que puedan hacer lo que quieran dentro del mismo, pero no puedan salir al exterior.

En concreto, este modelo se implementa mediante la construcción de cuatro barreras o líneas de defensa:

- Primera línea de defensa: Características del lenguaje/compilador
- Segunda línea de defensa: Verificador de código de bytes
- Tercera línea de defensa: Cargador de clases
- Cuarta línea de defensa: Gestor de Seguridad

Es importante señalar que aunque se hable de barreras de defensa, no se trata de barreras sucesivas. Es decir, no se trata de que si se traspasa la primera barrera, hay que superar la segunda, y luego la tercera y por fin la última, como muros uno detrás de otros. Más bien hay que imaginar una fortaleza con cuatro muros, y basta que se penetre uno de ellos para que la fortaleza caiga en manos del enemigo. Así que más que de líneas de defensa, habría que hablar de varios frentes.

Características del lenguaje/compilador

Java fue diseñado con las siguientes ideas en mente:

- Evitar errores de memoria
- Imposibilitar acceso al SO
- Evitar que caiga la máquina sobre la que corre

Con el fin de llevar a la práctica estos objetivos, se implementaron las siguientes características:

- Ausencia de punteros
  Protege frente a imitación de objetos, violación de encapsulación, acceso a áreas protegidas de memoria, ya que el programador no podrá referenciar posiciones de memoria específicas no reservadas, a diferencia de lo que se puede hacer en C y C++.

- Gestión de memoria
  Ya no se puede gestionar la memoria de forma tan directa como en C, (no hay malloc). En cambio, se instancian objetos, no se reserva memoria directamente (new siempre devuelve un handler), minimizando así la interacción del programador con la memoria y con el SO.

- Recogida de basura
  El programador ya no libera la memoria manualmente mediante free(fuente muy común de errores en C y C++, que podía llegar a producir el agotamiento de la memoria del sistema). El recogedor de basura de Java se encarga de reclamar la memoria usada por un objeto una vez que éste ya no es accesible o desaparece.

- Arrays con comprobación de límites
  En Java los arrays son objetos, lo cual les confiere ciertas funciones muy útiles, como la comprobación de límites. Para cada subíndice, Java comprueba si se encuentra en el rango definido según el número de elementos del array, previniendo así que se referencien elementos fuera de límite.

- Control de métodos y variables de clases
  Las variables y los métodos declarados privados sólo son accesibles por la clase o subclases herederas de ella y los declarados como protegidos, sólo por la clase.

- Métodos y clases final
  Las clases y los métodos (e incluso los datos miembro) declarados como final no pueden ser modificados o sobrescritos. Una clase declarada final no puede ser ni siquiera extendida.

Pero, ¿qué ocurriría si modifico un compilador de C para producir códigos de byte de Java, pasando por alto todas las protecciones suministradas por el lenguaje y el compilador de Java que acabamos de describir?. Con el fin de evitar esa forma de ataque, se construyó la segunda línea de defensa, el verificador de código de bytes.

Verificador de Códigos de Bytes

Sólo permite ejecutar código de bytes de programas Java válidos, buscando intentos de:

- fabricar punteros,
- ejecutar instrucciones en código nativo,
- llamar a métodos con parámetros no válidos,
- usar variables antes de inicializarlas,
- etc.

El verificador efectúa cuatro pasadas sobre cada archivo de clase:

En la primera, se valida el formato del archivo
En la segunda, se comprueba que no se instancien subclases de clases final
En la tercera, se verifica el código de bytes: la pila, registros, argumentos de métodos, opcodes
En la cuarta, se finaliza el proceso de verificación, realizándose los últimos tests

Si el verificador aprueba un archivo .class, se le supone que cumple ya con las siguientes condiciones:

- Acceso a registros y memoria válidos
- No hay overflow o underflow de pila
- Consistencia de tipo en parámetros y valores devueltos
- No hay conversiones de tipos ni castings ilegales

Aunque estas comprobaciones sucesivas deberían garantizar que sólo se ejecutarán applets legales, ¿qué pasaría si la applet carga una clase propia que reemplace a otra crítica del sistema, p.e. SecurityManager?. Para evitarlo, se erigió la tercera línea de defensa, el cargador de clases.

Cargador de Clases

A la hora de ejecutarse las applets en la máquina, se consideran tres dominios con diferentes niveles de seguridad:

- La máquina local (el más seguro)
- La red local guardada por el firewall (seguro)
- La Internet (inseguro)

En este contexto, no se permite a una clase de un domino de seguridad inferior sustituir a otra de un dominio superior, con el fin de evitar que una applet cargue una de sus clases para reemplazar una clase crítica del sistema, soslayando así las restricciones de seguridad de esa clase.

Este tipo de ataque se imposibilita asignando un espacio de nombres distinto para clases locales y para clases cargadas de la red. Siempre se accede antes a las clases del sistema, en vez de a clases del mismo nombre cargadas desde una applet.

's, las clases de un dominio no pueden acceder métodos no públicos de clases de otros dominios. Aun así, osible que algún recurso del sistema resultase fácilmente accesible por cualquier clase?.

itarlo, se creó la cuarta línea de defensa, el gestor de seguridad.

: de Seguridad

ión de seguridad la realiza la clase abstracta SecurityManager, que limita lo que las applets pueden o no hacer. revenir que sea modificada por una applet maliciosa, no puede ser extendida por ningún applet. Entre sus nes de vigilancia, se encuentran el asegurar que las applets no acceden al sistema de archivos, no abren ones a través de Internet, no acceden al sistema, etc.

lo, va a introducir una serie de restricciones de seguridad o prohibiciones.

cciones de seguridad

imer lugar, es importante hacer notar que las restricciones son distintas para cada navegador, ya que cada uno su propia política de seguridad. La política de Netscape Navigator, por ejemplo, es mucho más restrictiva que lotJava de Sun.

alquier caso, la mayoría de los navegadores presentarán las siguientes restricciones a la hora de ejecutar applets:

- se puede trabajar con el sistema de archivos: leer, escribir, borrar, renombrar, listar, conseguir información, utar programas, etc.
- se pueden establecer conexiones de red a máquinas distintas que la que envió el applet
- se permite acceso al sistema
- se permite manipulación de threads
- se pueden cargar métodos nativos
- se pueden evitar mensajes de alerta en las ventanas creadas por el applet
- se pueden crear subclases de SecurityManager en un applet

reviar, todas las restricciones anteriores pueden reducirse a dos reglas:

- o se puede trabajar con archivos en la máquina del usuario a menos que éste lo permita
- se puede conectar a nada en la red más que a la máquina que envió la applet

ráctica, estas restricciones son tan estrictas para applets descargadas remotamente que sólo pueden acceder rsos muy limitados dentro del modelo de patio de juegos (Sandbox model), como expresa la figura 4.6:

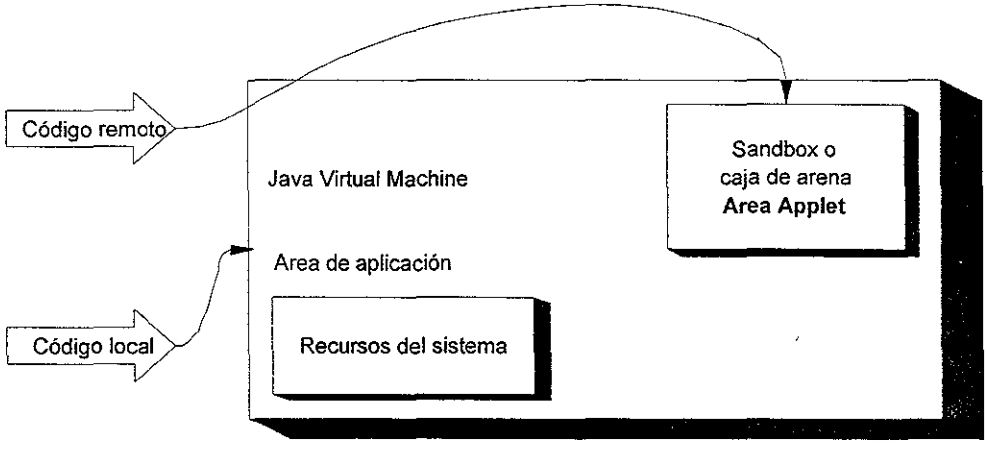


Figura 4 6 Modelo de seguridad de Java

Restricción de conexiones de red

Si las applets fueran capaces de establecer conexiones indiscriminadamente, se podrían producir los siguientes tipos de ataque:

- Ataques desde dentro de un firewall: el applet hostil podría descargarse desde una inocente página Web, pasando los controles del firewall, y una vez dentro, hacerse pasar por la máquina que la ha descargado para establecer conexiones o recibirlas.

- Suplantación de usuario, explotando la confianza de otras máquinas: una vez que el usuario incauto ha descargado inadvertidamente la applet y se encuentra en ejecución en su sistema, la applet podría comunicarse con otras máquinas usando la dirección IP de la máquina del pobre usuario y enviar por ejemplo correos ofensivos.

- Escucha de puertos que suplanten a un servicio (p.e. servidor Web, escuchando en el puerto 80 y respondiendo a todas las peticiones de páginas con sus propias páginas. Algo terrible para el sitio Web suplantado).

Para evitarlo, el gestor de seguridad se cuida de comprobar todos los intento de conexión.

Restricción de acceso al sistema

A las applets se les niega la capacidad de ejecutar programas en la máquina local y de acceder a propiedades del sistema, ya que estas capacidades podrían usarse para robo o destrucción de información o denegación de servicio:

- Terminar la sesión del navegador (simplemente llamando a System.exit()).
- Ejecutar comandos que manipulen los archivos u otros servicios.
- Conocer información privada del sistema o que puede revelar debilidades.

En vista de las restricciones impuestas por el modelo de seguridad de Java, muchos desarrolladores de software se encuentran ante la impotencia de escribir aplicaciones realmente útiles, puesto que se les niega la capacidad de leer/escribir en disco o de abrir conexiones a otras máquinas.

Si se quiere que las applets sean algo más que dibujos animados para realzar el impacto visual de una página y lleguen a convertirse en verdaderas aplicaciones productivas y realmente útiles, los usuarios de la aplicación Java necesitan tener garantizado el acceso a los recursos, para eso existen dos mecanismos.

El primero es creando firmas certificadas para applets. Esto permite reconocer applets que son seguros y con ello permitirle el acceso al sistema y sus recursos. Una applet firmada trata al código del Java Applet como local, con acceso completo a los recursos del sistema. Los applets sin firmar se ejecutan con acceso restringido a la caja de arena(model SandBox). La certificación de applets por medio de firmas parece adecuado para aplicaciones que no sean corporativas y abiertas, ya que cada sistema debe certificar al applet antes de poder ejecutarlo, y generando con esto dificultad y perdida de tiempo para su uso.

El segundo es creando una aplicación distribuida donde el usuario no debe preocuparse por ningún tipo de restricción de seguridad, sólo se conecta, lo ejecuta y listo. Para lograr esto, se debe crear una aplicación cliente/servidor, en la cual el cliente será el applet(el cual tiene muchas restricciones de seguridad) y el servidor será una aplicación que se ejecutara desde una máquina remota y llevará a cabo las operaciones que eran prohibidas para el applet. Está aplicación estará en contacto con el cliente - applet por medió de métodos remotos que enviarán al cliente toda la información que ésta requiera. Con este enfoque se saltan las restricciones de seguridad de los applets y a su vez se hace que la aplicación sea distribuida, ya que nos permitirá poder ejecutarla desde cualquier lugar y plataforma que soporte Java Applets. Este enfoque será usado para realizar la aplicación distribuida.

Para tener una visión concreta y amplia de cómo funcionara la aplicación y antes de comenzar el diseño de la misma, se verá que es la programación en redes, y como la maneja Java. Además, qué son y cómo programar aplicaciones distribuidas con Java.

5
Programación en redes

Para entender un poco más la estructura de Internet se hace necesario recordar el concepto de Cliente/Servidor. En este capítulo se verá qué es y cuál es su funcionamiento. Además, se verá como programar en redes con el paquete java.net. Se verán ejemplos sencillos de cómo establecer comunicación vía red entre aplicaciones Java. Las clases URL, URLConnection y HttpURLConnection.

5.1 Paradigma Cliente/Servidor e Internet

En un principio, es decir en las primeras redes, se conectaban periféricos, como impresoras, cintas de respaldo y terminales, a grandes computadoras (mainframes) de modo que esta gran computadora controlaba las operaciones de los periféricos. Esta interacción se conoce como maestro-esclavo. A medida que las computadoras comenzaron a ser más baratas, fue posible adquirir varias computadoras al interior de una organización.

Debido a la capacidad de las computadoras de comunicarse entre sí (en tanto podían enviar y recibir paquetes) y a que eran capaces de funcionar de manera independiente a esta tecnología de redes se le llamó computación distribuida. De este modo, en la computación distribuida, las computadoras en una red pueden interactuar con otras computadoras de manera arbitraria (sin distinción de clase, esto es, una computadora personal se puede conectar con otra PC o bien con un mainframe). La red Internet está diseñada bajo este esquema.

A pesar de la gran variedad de servicios disponibles en Internet, el software que implementa un servicio usa un esquema fijo: Cliente/Servidor. Bajo este esquema, cada computadora conectada a la red es a la vez potencialmente un cliente (que requiere servicios) y un servidor (que provee servicios). Es necesario aclarar que en realidad lo que se denomina cliente o servidor es un programa específico. Las computadoras no se comunican entre sí, sólo los programas lo hacen y es en este sentido que un programa provee los servicios que otro programa requiere.

El problema del *rendezvous (reencuentro)*

Desde el punto de vista de una aplicación, el TCP/IP, al igual que muchos otros protocolos de comunicación, implementa un mecanismo fiable para la transmisión de datos entre computadoras. En concreto, el TCP/IP permite que un programador pueda establecer comunicación de datos entre dos programas de aplicación, tanto si ambos se están ejecutando en la misma maquina, como en maquinas distintas unidas por algún camino físico (una red local, conexión telefónica directa entre computadoras, computadoras conectas a Internet,...).

Hay que tener presente que el protocolo TCP/IP especifica los detalles y mecanismos para la transmisión de datos entre dos aplicaciones comunicantes, pero no dictamina cuando ni porque deben interactuar ambas aplicaciones, ni siquiera especifica como debería estar organizada una aplicación que se va a ejecutar en un entorno distribuido. Es tarea del diseñador de la aplicación distribuida el establecer un protocolo de comunicación y sincronización adecuado.

En la práctica el esquema de programación más utilizado para la implementación de aplicaciones distribuidas es el paradigma cliente - servidor. La motivación fundamental para el empleo del paradigma cliente - servidor surge del problema del *rendezvous*. Para entender dicho problema, se debe imaginar a un técnico de computadoras que inicia la ejecución de dos programas en máquinas distintas y que tiene la intención de que dichos programas puedan comunicar entre sí. Una vez iniciado el primer programa este envía un mensaje a su contraparte. La conexión con la maquina a la cual va dirigido el mensaje se puede establecer en un intervalo de unos pocos milisegundos (recordar que las computadoras funcionan a velocidades muchos ordenes de magnitud por encima de los humanos), por lo que el proceso recién lanzado determina que su contraparte todavía no existe, con lo cual emite un mensaje de error y finaliza su ejecución. Mientras tanto, el técnico inicia la ejecución del segundo proceso. Desafortunadamente, el segundo proceso no puede comunicar con el primero ya que este ha concluido su ejecución. Incluso si los dos procesos intentan establecer la comunicación continuamente, estos pueden ejecutarse tan rápidamente, que la probabilidad de colisiones es muy alta.

Modelo Cliente/Servidor

El modelo Cliente/Servidor da solución al problema del *rendezvous*. Según este modelo, en cualquier par de aplicaciones comunicantes, una de las partes implicadas en la comunicación debe iniciar su ejecución y esperar (indefinidamente) hasta que la otra parte contacte con ella. Esta solución es importante, ya que el TCP/IP no proporciona ningún mecanismo que automáticamente lance procesos para atender mensajes entrantes, debe de existir un proceso en espera que sea capaz de atender dichos mensajes (peticiones de servicio).

La arquitectura Cliente/Servidor es una forma específica de diseño de aplicaciones, aunque también se conoce con este nombre a las computadoras en las que estas aplicaciones son ejecutadas. Se debe precisar que cliente y servidor no tienen que estar necesariamente en computadoras separados, sino que pueden ser programas diferentes que se ejecuten en la misma computadora

Muchos administradores hacen que ciertos programas de comunicaciones se inicien automáticamente cuando el sistema arranca, de este modo se aseguran que la computadora estará preparada para aceptar ciertas solicitudes de servicio. Después de iniciar su ejecución, cada uno de estos programas se queda en espera de la siguiente petición para el servicio que oferta.

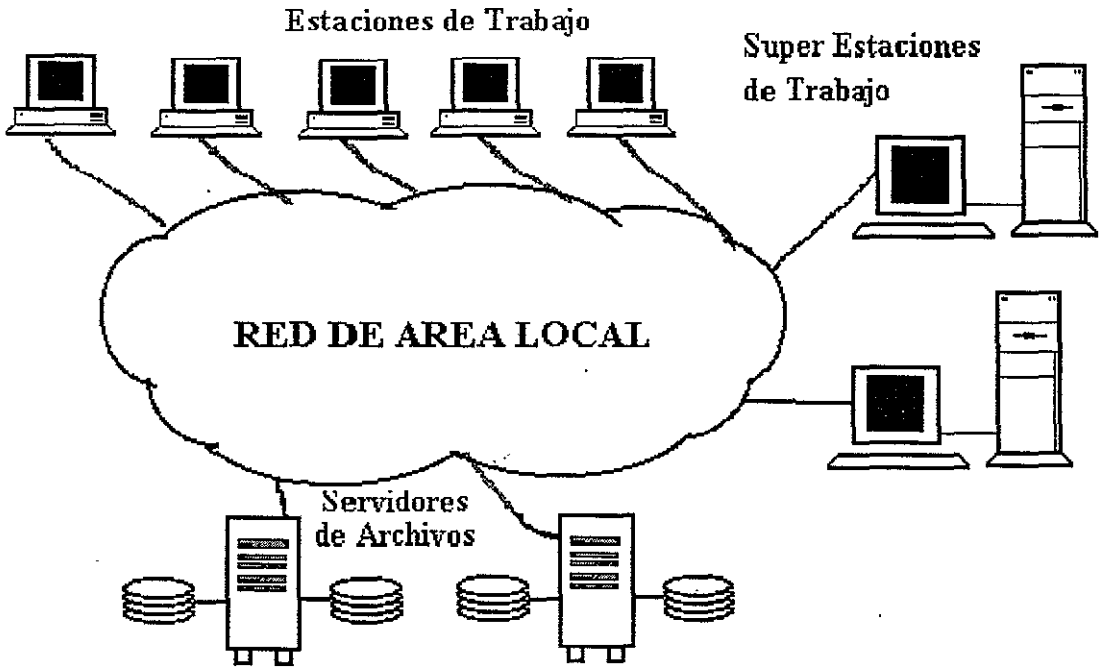


Figura 5.1 Modelo de un Sistema Cliente/Servidor

5.2 Programación en redes con el paquete java.net

El entorno Java tiene la capacidad de escribir programas que utilizan e interactúan con los recursos de Internet y el World Wide Web. De hecho, los navegadores que soportan Java utilizan esta capacidad del entorno Java para transportar y ejecutar applets a través de la red.

Dentro del paquete java.net se encuentran con las siguientes clases:

1. HttpURLConnection
2. InetAddress
3. ServerSocket
4. Socket
5. URL
6. URLConnection
7. URLEncoder
8. URLStreamHandler

Ellas permiten abrir conexiones TCP entre máquinas distintas, y la manipulación de URL(Universal Resource Locator). Existen clases para abrir conexiones UDP, pero esas no se verán ya que la aplicación será implementada sobre el protocolo TCP.

Trabajo en red básico

Las computadoras que se ejecutan en Internet se comunican unas con otras utilizando los protocolos TCP y UDP, que son los protocolos de 4 capas:

Aplicación	(HTTP, FTP, Telnet, ...)
Transporte	(TCP/IP, UDP, ...)
Red	(IP, ...)
Enlace	(device driver, ...)

Las clases del paquete java.net proporcionan capacidades de programación en red de forma independiente al sistema operativo basándose principalmente en los sockets como primitiva de comunicación. Cuando dos aplicaciones se comunican, primero establecen una conexión, y a través de la misma envían y reciben información. Los sockets son puntos finales de enlace de comunicaciones entre procesos. Son un punto de comunicación bidireccional entre programas funcionando en lugares distintos en la red.

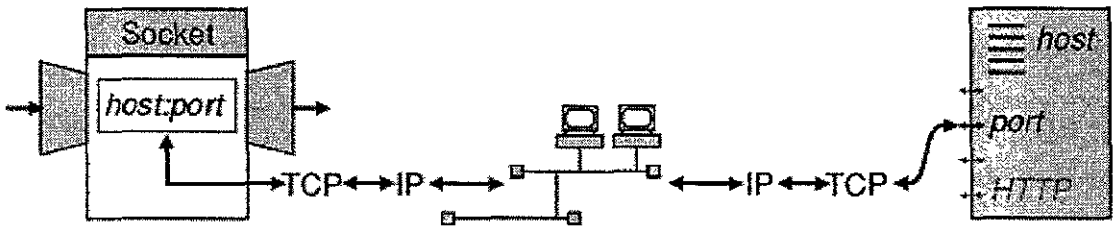


Figura 5.2 Comunicación con sockets en Internet

Cuando se escriben programas Java que se comunican a través de la red, se está programando en la capa de aplicación. Típicamente, no se necesita trabajar con las capas TCP y UDP, en su lugar se puede utilizar las clases del paquete java.net. Estas clases proporcionan comunicación de red independiente del sistema. Sin embargo, se necesita entender la diferencia entre TCP y UDP para decidir que clases Java deberán utilizar los programas

TCP garantiza que los datos enviados por una parte de la conexión realmente llegan a la otra parte y en el mismo orden en el que fueron enviados (de otra forma daría un error).

Las aplicaciones que requieren fiabilidad, canales punto a punto para comunicarse, utilizan TCP para ello. Hyper Text Transfer Protocol (HTTP), File Transfer Protocol (FTP), y Telnet son ejemplos de aplicaciones que requieren un canal de comunicación fiable. El orden en que los datos son enviados y recibidos a través de la red es crítico para el éxito de estas aplicaciones; cuando se utiliza HTTP para leer desde una URL, los datos deben recibirse en el mismo orden en que fueron enviados, de otra forma se tendrá un archivo HTML revuelto, un archivo Zip corrupto o cualquier otra información no válida.

Para muchas aplicaciones esta garantía de fiabilidad es crítica para el éxito de la transferencia de información desde un punto de la conexión al otro. Sin embargo, otras formas de comunicación no necesitan esta comunicación tan estricta y de hecho lo que hace es estorbar porque la conexión fiable anula el servicio.

Considera, por ejemplo, un servicio de hora que envía la hora actual a sus clientes cuando estos lo piden. Si el cliente pierde un paquete, ¿tiene sentido volver a enviar el paquete? No porque la hora que recibiría el cliente ya no sería exacta. Si el cliente hace dos peticiones y recibe dos paquetes del servidor en distinto orden, realmente no importa porque el cliente puede imaginarse que los paquetes no están en orden y pedir otro. El canal fiable, aquí no es necesario, causando una degradación del rendimiento, y podría estorbar a la utilidad del servicio.

Otro ejemplo de servicio que no necesita un canal de fiabilidad garantizada es el comando ping de Unix. El único objetivo del comando ping es comprobar la comunicación entre dos programas a través de la red. De hecho, ping necesita conocer las caídas o los paquetes fuera de orden para determinar lo buena o mala que es la conexión. Así un canal fiable invalidaría este servicio.

El protocolo UDP proporciona una comunicación no garantizada entre dos aplicaciones en la red. UDP no está basado en la conexión como TCP. UDP envía paquetes de datos, llamados datagramas de una aplicación a la otra. Enviar datagramas es como enviar una carta a través del servicio de correos: el orden de envío no es importante y no está garantizado, y cada mensaje es independiente de los otros.

Puertos

Generalmente hablando, una computadora tiene una sola conexión física con la red. Todos los datos destinados a esa computadora en particular llegan a través de la conexión. Sin embargo, los datos podrían ser utilizados por diferentes aplicaciones ejecutándose en la computadora. ¿Entonces cómo sabe la computadora a qué aplicación enviarle los datos? La respuesta es a través del uso de los puertos.

Los datos transmitidos por Internet están acompañados por una información de dirección que identifica la computadora y el puerto al que están destinados. Una computadora está identificada por su dirección IP de 32 bits, esta dirección se utiliza para enviar los datos a la computadora correcta en la red. Los puertos están identificados por un número de 16 bits, que TCP y UDP utilizan para enviar los datos a la aplicación correcta.

En aplicaciones basadas en la conexión, una aplicación establece una conexión con otra aplicación uniendo un socket a un número de puerto. Esto tiene el efecto de registrar la aplicación con el sistema para recibir todos los datos destinados a ese puerto. Dos aplicaciones no pueden utilizar el mismo puerto: intentar acceder a un puerto que ya está utilizado dará un error.

En comunicaciones basadas en datagramas, los paquetes de datagramas contienen el número de puerto del destinatario.

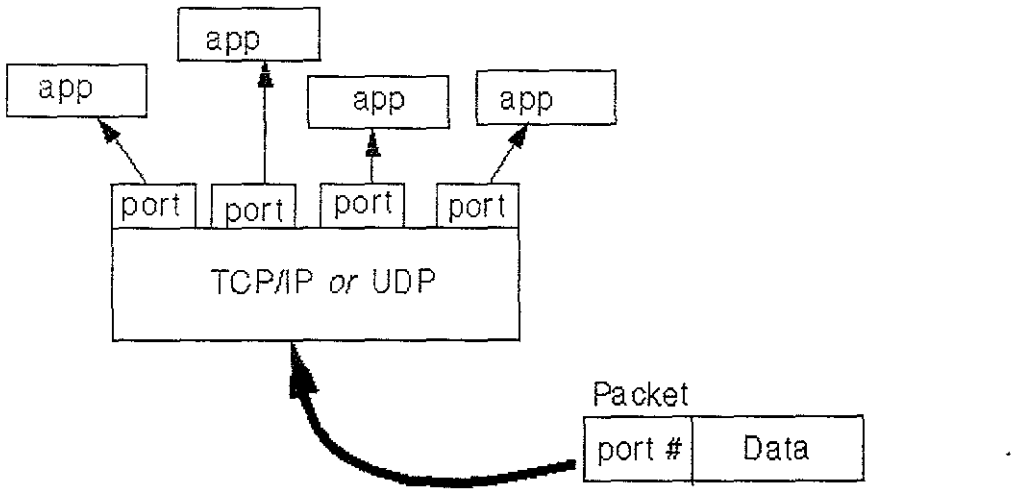


Figura 5.3 Comunicaciones basadas en datagramas utilizando un puerto de conexión disponible

Los números de puertos tienen un rango de 0 a 65535 (porque los puertos están representados por un número de 16 bits). Los puertos entre los números 0 – 1023 están reservados y restringidos para servicios bien conocidos como HTTP, FTP y otros servicios del sistema. Las aplicaciones que se realicen con el paquete java.net no deben intentar unirse a estos puertos. Los puertos que están reservados para los servicios bien conocidos como HTTP y FTP son llamados puertos bien conocidos.

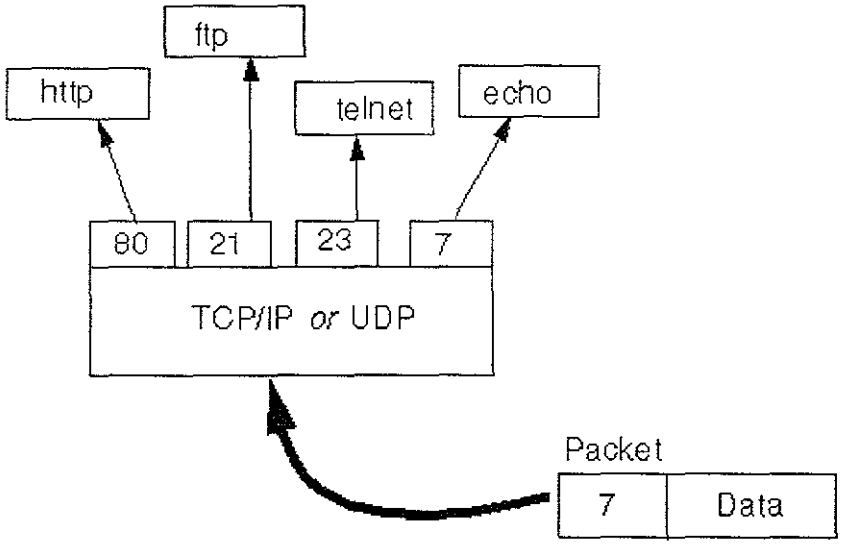


Figura 5.4 Comunicaciones basadas en datagramas que utilizan puertos conocidos

A través de las clases del paquete java.net, los programas Java pueden utilizar TCP o UDP para comunicarse a través de Internet. Las clases URL, URLConnection, HttpURLConnection, Socket, y SocketServer utilizan el TCP para comunicarse a través de la red. Las clases DatagramPacket y DatagramServer utilizan UDP.

Modelo de comunicaciones con Java

En Java, crear una conexión socket TCP/IP se realiza directamente con el paquete java.net. A continuación se muestra un diagrama de lo que ocurre en el lado del cliente y del servidor:

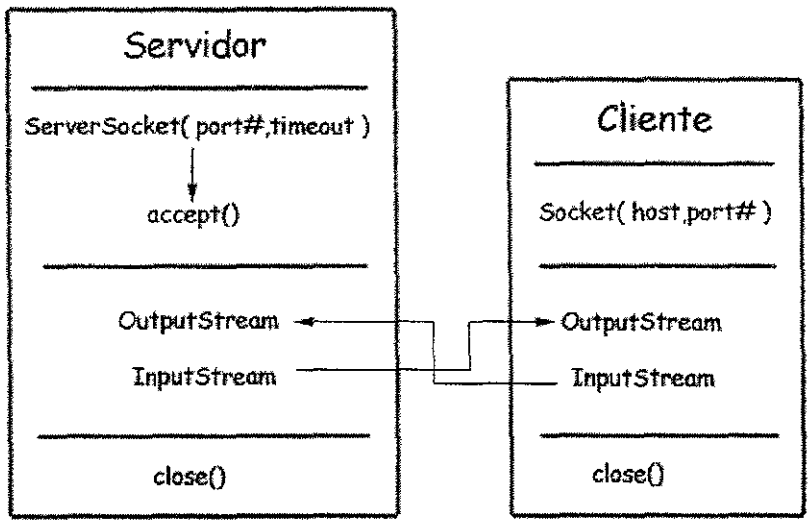


Figura 5 5 Modelo de comunicaciones Cliente/Servidor por medio de sockets en Java

El modelo de sockets en Java es:

- El servidor establece un puerto y espera durante un cierto tiempo (timeout segundos), a que el cliente establezca la conexión. Cuando el cliente solicite una conexión, el servidor abrirá la conexión socket con el método accept().
- El cliente establece una conexión con la máquina *Host* a través del puerto que se designe en port#
- El cliente y el servidor se comunican con manejadores InputStream y OutputStream

Como se vio en el capítulo pasado los applets sólo pueden establecer conexiones con la máquina desde la cual se transfirió su código debido al modelo de seguridad que maneja Java. Esto está implementado en el JDK y en el intérprete de Java de Netscape. Esto reduce en gran manera la flexibilidad de las fuentes de datos disponibles para los applets. El problema es que si se permite que un applet se conecte a cualquier máquina de la red, se podrían utilizar los applets para inundar la red desde una computadora con un cliente Netscape del que no se sospecha y sin ninguna posibilidad de rastreo.

Apertura de Sockets

Si se está programando un cliente, el socket se abre de la forma:

```
Socket miCliente;
try {
      miCliente = new Socket( "maquina",numeroPuerto );
   } catch( IOException e ) {
     System.out.println( e );
   }
```

Donde "maquina" es el nombre de la máquina en donde se está intentando abrir la conexión y "numeroPuerto" es el puerto (un número) del servidor que está corriendo sobre la misma máquina. Cuando se selecciona un número de puerto, se debe tener en cuenta que los puertos en el rango 0-1023 están reservados para usuarios con muchos privilegios (superusuarios o root). Estos puertos son los que utilizan los servicios estándar del sistema como email, ftp o http. Para las aplicaciones que se desarrollen, se debe asegurar de seleccionar un puerto por encima del 1023.

Si se está programando un servidor, la forma de apertura del socket es la que muestra el siguiente ejemplo:

```
Socket miServicio;
try {
      miServicio = new ServerSocket( numeroPuerto );
   } catch( IOException e ) {
     System.out.println( e );
   }
```

A la hora de la implementación de un servidor también se necesita crear un objeto socket desde el ServerSocket para que esté atento a las conexiones que le puedan realizar clientes potenciales y poder aceptar esas conexiones:

```
Socket socketServicio = null;
try {
      socketServicio = miServicio.accept();
   } catch( IOException e ) {
     System.out.println( e );
   }
```

Creación de Streams de Entrada

En la parte cliente de la aplicación, se puede utilizar la clase DataInputStream para crear un stream de entrada que esté listo a recibir todas las respuestas que el servidor le envíe.

```
DataInputStream entrada;
try {
      entrada = new DataInputStream( miCliente.getInputStream() );
   } catch( IOException e ) {
     System.out.println( e );
   }
```

La clase DataInputStream permite la lectura de líneas de texto y tipos de datos primitivos de Java de un modo altamente portable; dispone de métodos para leer todos esos tipos como: read(), readChar(), readInt(), readDouble() y readLine(). Se debe utilizar la función que se crea necesaria dependiendo del tipo de dato que se espere recibir del servidor.

En el lado del servidor, también se usa DataInputStream, pero en este caso para recibir las entradas que se produzcan de los clientes que se hayan conectado:

```
DataInputStream entrada;
try {
      entrada =
         new DataInputStream( socketServicio.getInputStream() );
   } catch( IOException e ) {
     System.out.println( e ),
   }
```

Creación de Streams de Salida

En el lado del cliente, se puede crear un stream de salida para enviar información al socket del servidor utilizando las clases *PrintStream o DataOutputStream:*

```
PrintStream salida;
try {
      salida = new PrintStream( miCliente.getOutputStream() );
   } catch( IOException e ) {
     System.out.println( e );
   }
```

La clase PrintStream tiene métodos para la representación textual de todos los datos primitivos de Java. Sus métodos write y println() tienen una especial importancia en este aspecto. No obstante, para el envío de información al servidor también se puede utilizar DataOutputStream:

```
DataOutputStream salida;
try {
      salida = new DataOutputStream( miCliente.getOutputStream() );
   } catch( IOException e ) {
     System out.println( e );
   }
```

La clase DataOutputStream permite escribir cualquiera de los tipos primitivos de Java, muchos de sus métodos escriben un tipo de dato primitivo en el stream de salida. De todos esos métodos, el más útil quizás sea writeBytes().

En el lado del servidor, se puede utilizar la clase PrintStream para enviar información al cliente:

```
PrintStream salida;
try {
      salida = new PrintStream( socketServicio.getOutputStream() );
   } catch( IOException e ) {
     System.out.println( e );
   }
```

Pero también se puede utiliza la clase DataOutputStream como en el caso de envío de información desde el cliente.

Cierre de sockets

Siempre se deben cerrar los canales de entrada y salida que se hayan abierto durante la ejecución de la aplicación.

En la parte del cliente:

```
try {
      salida.close(),
      entrada.close();
      miCliente.close();
   } catch( IOException e ) {
      System out.println( e );
   }
```

Y en la parte del servidor:

```
try {
      salida.close();
      entrada.close();
      socketServicio.close();
      miServicio.close();
   } catch( IOExceptuon e ) {
      System out.println( e );
   }
```

La clase URL

Si se ha navegado por la World Wide Web, indudablemente se habrá alguna vez oído el término URL y se habrá utilizado URLs para acceder a varias páginas HTML de la Web. Entonces, ¿qué es exactamente una URL?.

Como se vio en el capítulo 3, URL es un acrónimo que viene de Uniform Resource Locator y es una referencia (una dirección) a un recurso de Internet.

Algunas veces es más sencillo (aunque no enteramente acertado) pensar en una URL como el nombre de un archivo en la red porque la mayoría de las URLs se refieren a un archivo o alguna máquina de la red. Sin embargo, se debe recordar que las URLs pueden apuntar a otros recursos de la red, como consultas a bases de datos, o salidas de comandos.

Lo siguiente es un ejemplo de una URL:

http://java.sun.com/

Esta URL particular direccióna la Web de Java hospedada por Sun Microsystems. La URL anterior, como otras URLs, tiene dos componentes principales:

- El identificador de protocolo
- El nombre del recurso

En el ejemplo, http es el identificador de protocolo y //java.sun.com/ es el nombre del recurso.

El identificador de protocolo indica el nombre del protocolo a utilizar para buscar ese recurso. El ejemplo utiliza el protocolo Hyper Text Transfer Protocol(HTTP), que es utilizado típicamente para servir documentos de hypertexto. HTTP es sólo uno de los diferentes protocolos utilizados para acceder a los distintos tipos de recursos de la red.

Otros protocolos incluyen File Transfer Protocol(ftp), Gopher(gopher), File(file), y News(news) y otros vistos anteriormente en el capítulo 2.

El nombre del recurso es la dirección completa al recurso. El formato del nombre del recurso depende completamente del protocolo utilizado, pero la mayoría de los formatos de nombres de recursos contienen uno o más de los siguientes componentes:

- **Nombre del *Host***
  Nombre de la máquina donde reside el recurso.
- **Nombre de archivo**
  El path al archivo dentro de la máquina
- **Número de puerto**
  El número de puerto a conectar (normalmente es opcional)
- **Referencia**
  Una referencia a una posición marcada dentro del recurso; normalmente identifica una posición específica dentro de un archivo (normalmente es opcional)

Para muchos protocolos, el nombre del *Host* y el nombre del archivo son obligatorios y el número de puerto y la referencia son opcionales. Por ejemplo, el nombre de recursos para una URL HTTP debería especificar un servidor de la red (el nombre del *Host)* y el path al documento en esa máquina (nombre de archivo), y también puede especificar un número de puerto y una referencia. En la URL mostrada anteriormente, java.sun.com es el nombre del *Host* y la barra inclinada '/' es el path para el nombre del archivo que por default es /index.html.

Cuando se construya una URL, se pone primero el identificador de protocolo, seguido por dos puntos (:), seguido por el nombre del recurso, de esta forma:

ProtocoloID: nombre de Recursos

El paquete java.net contiene una clase llamada URL que utilizan los programas Java para representar una dirección URL. Tus programas Java pueden construir un objeto URL, abrir una conexión con el, leer y escribir datos a través de esa conexión.

Crear una URL con el paquete java.net

La forma más sencilla de crear un objeto URL es crearlo desde una Cadena que represente la forma "Leíble" de la dirección URL. Esta es la forma típica que otra persona utilizaría para decirte una URL. Por ejemplo, para decirle la dirección de Gamelan que contiene una lista de sitios sobre Java, se puede dar de la siguiente forma:

http://www.gamelan.com/

En el programa Java, se puede utilizar una cadena que contenga el texto anterior para crear un objeto URL:

```
URL gamelan = new URL("http://www.gamelan.com/");
```

El objeto URL anterior representa una URL absoluta. Una URL absoluta contiene toda la información necesaria para alcanzar el recurso en cuestión. También se pueden crear objetos URL desde una dirección URL relativa.

Crear una URL relativa a otra

Una URL relativa sólo contiene la información suficiente para alcanzar el recurso en relación a (o en el contexto de) otra URL.

Las especificaciones de las URL relativas se utilizan frecuentemente en archivos HTML. Por ejemplo, suponer que se ha escrito un archivo HTML llamado HomePage.html. Dentro de esta página, hay enlaces a otras páginas, graficos.html y Preferencias.html, que están en la misma máquina y en el mismo directorio que HomePage.html.

Estos enlaces a graficos.html y Preferencias.html desde HomePage.html podrían especificarse sólo como nombres de archivos, de la siguiente forma:

```
<a href="graficos.html">graficos</a>
<a href="preferencias.html">Preferencias</a>
```

Estas direcciónes URL son URLs relativas. Esto es, las URL estás especificadas en relación al archivo en el que están contenidas: HomePage.html.

En los programas java, se puede crear un objeto URL desde una especificación URL relativa. Por ejemplo, suponer que ya se ha creado una URL para "http://www.gamelan.com/" en el programa, y que sabe los nombres de varios archivos en ese site(Gamelan.network.html, y Gamelan.animation.html). Se pueden crear URLs para cada archivo de Gamelan simplemente especificando el nombre del archivo en el contexto de la URL original de Gamelan. Los nombres de archivos son URLs relativas y están en relación a la URL original de Gamelan.

```
URL gamelan = new URL("http://www.gamelan.com/");

URL gamelanNetwork = new URL(gamelan, "Gamelan.network.html");
```

Este código utiliza un constructor de la clase URL que permite crear un objeto URL desde un objeto URL (la base) y una URL relativa.

Este constructor también es útil para crear URL llamados anclas (también conocidos como referencias) dentro de un archivo. Por ejemplo, suponer que el archivo "Gamelan.network.html" tiene una referencia llamada BOTTOM que está al final del archivo. Se puede utilizar el constructor de URL relativa para crear una URL como esta:

```
URL gamelanNetworkBottom = new URL(gamelanNetwork, "#BOTTOM");
```

La forma general de este constructor de URL es:
URL(URL URLbase, String URLrelativa)

El primer argumento es un objeto URL que especifica la base de la nueva URL, y el segundo argumento es una cadena que especifica el resto del nombre del recurso relativo a la base. Si URLbase es null, entonces este constructor trata URLrelativa como si fuera una especificación de una URL absoluta. Y al revés, si relativeURL es una especificación de URL absoluta, entonces el constructor ignora baseURL.

Otros Constructores de URL

La clase URL proporciona dos constructores adicionales para crear un objeto URL. Estos constructores son útiles cuando trabajan con URLs como URLs HTTP, que tienen los componentes del nombre del *Host*, el nombre del archivo, el número de puerto y una referencia en la parte del nombre del recurso de la URL. Estos dos constructores son útiles cuando no se tiene una cadena que contiene la especificación completa de la URL, pero si conocen algunos componentes de la URL.

Por ejemplo, si se ha diseñado un navegador con un panel similar al explorador de archivos que le permite al usuario utilizar el ratón para seleccionar el protocolo, el nombre del *Host*, el número del puerto, el nombre del archivo, se puede construir una URL a partir de estos componentes. El primer constructor crea una URL desde un protocolo, un nombre de *Host* y un nombre de archivo. El siguiente código crea una URL del archivo Gamelan.network.html en la site de Gamelan:

URL gamelan = new URL("http", "www.gamelan.com", "/Gamelan.network.html");

Esto es equivalente a URL("http://www.gamelan.com/Gamelan.network.html"). El primer argumento es el protocolo, el segundo es el nombre del *Host* y el último argumento es el path del archivo. Observar que el nombre del archivo contiene la barra inclinada (/) al principio. Esto indica que el nombre de archivo está especificado desde la raíz del *Host*.

El último constructor de URL añade el número de puerto a la lista de los argumentos utilizados por el constructor anterior.

URL gamelan = new URL("http", "www.gamelan.com", 80, "/Gamelan.network.html");

Esto crea un objeto URL con la siguiente dirección URL:

http://www.gamelan.com:80/Gamelan.network.html

Si se construye una URL utilizando uno de estos constructores, se puede obtener una cadena que contiene la dirección URL completa, utilizando el método toString() de la clase URL o el método toExternalForm() equivalente.

MalformedURLException

Cada uno de los cuatro constructores de URL lanza una MalformedURLException si los argumentos del constructor son nulos o el protocolo es desconocido.

Típicamente, se querrá capturar y manejar esta excepción. Así normalmente se debería introducir un constructor de URL en un par try/catch.

```
try {
      URL myURL = new URL(. . .)
   } catch (MalformedURLException e) {
      // Aquí va el código del manejador de excepciones
   }
```

Analizar una URL

La clase URL proporciona varios métodos que permiten preguntar a los objetos URL. Puede obtener el protocolo, nombre de *Host*, número de puerto, y nombre de archivo de una URL utilizando estos métodos accesores:

getProtocol()
Devuelve el componente identificador de protocolo de la URL.

getHost()
Devuelve el componente nombre del *Host* de la URL.

getPort()
Devuelve el componente número del puerto de la URL. Este método devuelve un entero que es el número de puerto. Si el puerto no está seleccionado, devuelve -1.

getFile()
Devuelve el componente nombre de archivo de la URL.

getRef()
Obtiene el componente referencia de la URL.

Se debe recordar que no todas las direcciones URL contienen estos componentes. La clase URL proporciona estos métodos porque las URLs de HTTP contienen estos componentes y quizás son las URLs más utilizadas. La clase URL está centrada sobre HTTP.

Se pueden utilizar estos métodos getXXX() para obtener información sobre la URL sin importar el constructor que se haya utilizado para crear el objeto URL.

La clase URL, junto con estos métodos accesores, libera de tener que analizar la URL de nuevo. Dando a cualquier cadena la especificación de una URL, y sólo creando un nuevo objeto URL y llamando a uno de sus métodos accesores para la información que se necesite. Este pequeño programa de ejemplo crea una URL partiendo de una especificación y luego utiliza los métodos accesores del objeto URL para analizar la URL:

```
import java.net.*;
import java.io.*;

class ParseURL {
   public static void main(String[] args) {
      URL aURL = null;
      try {
         aURL = new URL("http://java.sun.com:80/tutorial/intro.html#DOWNLOADING");
         System.out.println("protocol = " + aURL.getProtocol());
         System.out.println("Host = " + aURL.getHost());
         System.out.println("filename = " + aURL.getFile());
         System.out.println("port = " + aURL.getPort());
         System.out.println("ref = " + aURL.getRef());
      } catch (MalformedURLException e) {
         System.out.println("MalformedURLException: " + e);
      }
   }
}
```

Aquí se tiene la salida mostrada por el programa:

```
protocol = http
Host = java.sun.com
filename = /tutorial/intro.html
port = 80
ref = DOWNLOADING
```

Leer Directamente desde una URL

Después de haber creado satisfactoriamente una URL, se puede llamar al método openStream() de la clase URL para obtener un canal desde el que poder leer el contenido de la URL. El *método retorna un objeto* java.io.InputStream por lo que se puede leer normalmente de la URL utilizando los métodos normales de InputStream.

Leer desde una URL es tan sencillo como leer de un canal de entrada. El siguiente programa utiliza openStream() para obtener un stream de entrada a la URL "http://www.yahoo.com/". Lee el contenido del canal de entrada y lo muestra en la pantalla.

```
import java.net.*;
import java.io.*;

class OpenStreamTest {
   public static void main(String[] args) {
      try {
```

```
        URL yahoo = new URL("http://www.yahoo.com/");
        DataInputStream dis = new DataInputStream(yahoo.openStream());
        String inputLine;
        while ((inputLine = dis.readLine()) != null) {
          System.out.println(inputLine);
        }
        dis.close();
      } catch (MalformedURLException me) {
        System.out.println("MalformedURLException: " + me);
      } catch (IOException ioe) {
        System.out.println("IOException. " + ioe);
      }
    }
}
```

Cuando se ejecute el programa, se deberán ver los tags HTML y el contenido textual del archivo HTML localizado en "http://www.yahoo.com/" desplazándose la ventana de comandos.

O se podría ver el siguiente mensaje de error:

IOException: java.net.UnknownHostException: www.yahoo.com

El mensaje anterior indica que se podría tener seleccionado un proxy y por eso el programa no puede encontrar el servidor www.yahoo.com. (Si es necesario, se debe preguntar al administrador de la red por el proxy del servidor.).

Conectar con una URL

Si se ha creado satisfactoriamente una URL, se puede llamar al método openConnection() de la clase URL para conectar con ella. Cuando se haya conectado con una URL, se habrá inicializado un enlace de comunicación entre un programa Java y la URL a través de la red. Por ejemplo, se puede abrir una conexión con el motor de búsqueda de Yahoo con el código siguiente:

```
try {
  URL yahoo = new URL("http://www.yahoo.com/");
  yahoo.openConnection();
} catch (MalformedURLException e) {    // nueva URL() fallada
  ...
} catch (IOException e) {              // openConnection() fallada
  ...
}
```

Si es posible, el método openConnection() crea una nuevo objeto URLConnection (si no existe ninguno apropiado), lo inicializa, conecta con la URL y devuelve el objeto URLConnection. Si algo va mal -- por ejemplo, el servidor de Yahoo está apagado -- el método openConnection() lanza una IOException.

Ahora que se ha conectado satisfactoriamente con la URL se puede utilizar el objeto URLConnection para realizar algunas acciones como leer o escribir a través de la conexión.

Leer y Escribir a través de un objeto URLConnection

Si se ha utilizado satisfactoriamente openConnection() para inicializar comunicaciones con una URL, se tendrá una referencia a un objeto URLConnection. La clase URLConnection contiene muchos métodos que permiten comunicarse con la URL a través de la red. URLConnection es una clase centrada sobre HTTP, muchos de sus métodos son útiles sólo cuando trabajan con URLs de HTTP.

Sin embargo, en la mayoría de los protocolos, URL permite leer y escribir desde una conexión por eso esta página enseña como leer y escribir desde una URL a través de un objeto URLConnection.

Leer desde un objeto URLConnection

El siguiente programa realiza la misma función que el mostrado en Leer Directamente desde una URL. Sin embargo, mejor que abrir directamente un stream desde la URL, este programa abre explícitamente una conexión con la URL, obtiene un stream de entrada sobre la conexión, y lee desde el stream de entrada:

```
import java.net.*;
import java.io.*;

class ConnectionTest {
   public static void main(String[] args) {
      try {
         URL yahoo = new URL("http://www.yahoo.com/");
         URLConnection yahooConnection = yahoo.openConnection();
         DataInputStream dis = new DataInputStream(yahooConnection.getInputStream());
         String inputLine;
         while ((inputLine = dis.readLine()) != null) {
            System.out.println(inputLine);
         }
         dis.close();
      } catch (MalformedURLException me) {
         System.out.println("MalformedURLException: " + me);
      } catch (IOException ioe) {
         System.out.println("IOException: " + ioe);
      }
   }
}
```

La salida de este programa debería ser idéntica a la salida del programa que abría directamente el stream desde la URL. Se puede utilizar cualquiera de estas dos formas para leer desde una URL. Sin embargo, algunas veces leer desde una URLConnection en vez de leer directamente desde una URL podría ser más útil ya que se puede utilizar el objeto URLConnection para otras tareas (como escribir sobre la conexión URL) al mismo tiempo.

De nuevo, si en vez de ver la salida del programa, se viera el siguiente mensaje error:

```
IOException: java.net.UnknownHostException: www.yahoo.com
```

Podrías tener activado un proxy y el programa no podría encontrar el servidor de www.yahoo.com.

Escribir a una URLConnection

Muchas páginas HTML contienen formularios; campos de texto y otros objeto GUI que le permiten introducir datos en el servidor. Después de teclear la información requerida e iniciar la petición pulsando un botón, el navegador que se utiliza escribe los datos en la URL a través de la red. Después de que la otra parte de la conexión (normalmente un script cgi-bin) en el servidor de datos, los procesa, y le envía de vuelta una respuesta, normalmente en la forma de una nueva página HTML. Este escenario es el utilizado normalmente por los motores de búsqueda.

Muchos scripts cgi-bin utilizan el POST METHOD para leer los datos desde el cliente. Así, escribir sobre una URL frecuentemente es conocido como "posting" a URL. Los scripts del lado del servidor utilizan el método POST METHOD para leer desde su entrada estándar.

Algunos scripts cgi-bin del lado del servidor utilizan el método GET METHOD para leer sus datos. Sin embargo, el método POST METHOD es más rápido en su ejecución, más versátil y no tiene limitaciones sobre los datos que pueden ser enviados a través de la conexión como el GET METHOD que ya es obsoleto.

Java también pueden interactuar con los scripts cgi-bin del lado del servidor. Sólo se debe poder escribir a una URL, así se proporcionan los datos al servidor. El programa puede hacer esto siguiendo los siguientes pasos:

1. Crear una URL.
2. Abrir una conexión con la URL.
3. Obtener un stream de salida sobre la conexión. Este canal de entrada está conectado al stream de entrada estándar del script cgi-bin del servidor.
4. Escribir en el stream de salida.
5. Cerrar el stream de salida.

Un programa de ejemplo que ejecuta una escritura a un script a través de la red utilizando un URLConnection:

```
import java.io.*;
import java.net.*;

public class ReverseTest {
   public static void main(String[] args) {
      try {
         if (args.length != 1) {
            System.err.println("Usage:  java ReverseTest string_to_reverse");
            System.exit(1);
         }
         String stringToReverse = URLEncoder.encode(args[0]);

         URL url = new URL("http://java.sun.com/cgi-bin/backwards");
         URLConnection connection = url.openConnection();

         PrintStream outStream = new PrintStream(connection.getOutputStream());
         outStream.println("string=" + stringToReverse);
         outStream.close();

         DataInputStream inStream = new DataInputStream(connection.getInputStream());
         String inputLine;

         while ((inputLine = inStream.readLine()) != null) {
            System.out.println(inputLine);
         }
         inStream.close();
      } catch (MalformedURLException me) {
         System.err.println("MalformedURLException: " + me);
      } catch (IOException ioe) {
         System.err.println("IOException: " + ioe);
      }
   }
}
```

Frecuentemente, como en este ejemplo, cuando se escribe en una URL se está pasando información al script cgi-bin que lee la información que se escribe, realiza alguna acción y luego envía la información de vuelta mediante la misma URL.

Uso de HttpURLConnection

La clase HttpURLConnection es un subclase de URLConnection, provee una acceso directo a los parámetros HTTP involucrados en una conexión Cliente/Servidor del protocolo HTTP.

Esta clase está especializada en la gestión de protocolos HTTP. Es utilizada automáticamente por un objeto URL cuando se habré una conexión.

```
/*
 * Ejemplo de uso de la clase HttpURLConnection
 */

import java.io.*;
import java.net.URL;
import java.net.URLConnection;
import java.net.HttpURLConnection;

public class wget {
  static final PrintStream out = System.out;
  static final String commandName = wget.class.getName();

  public static void main(String args[]) throws IOException {
    try {
      while (args[0].charAt(0) == '-') {
        String argument = args[0].substring(1);
        args[0] = args[1];
      }
    } catch (ArrayIndexOutOfBoundsException e) {
      out.println( "USAGE: " + commandName + " <url>" );
      System.exit( 1 );
    }

    URLConnection url = (new URL(args[0])).openConnection();

    if (url instanceof HttpURLConnection) {
      readHttpURL((HttpURLConnection) url);
    } else {
      readURL(url);
    }
  }

  public static final void readURL(URLConnection url) throws IOException {
    DataInputStream in = new DataInputStream(url.getInputStream());
    printHeader(url);
    try {
      while (true) {
        writeChar((char) in.readUnsignedByte());
      }
    } catch (EOFException e) {
      verbose("\n");
    } catch (IOException e) {
      out.println( e + ": " + e.getMessage());
      verbose("\n");
    }
    System.exit(0);
  }
```

```
public static final void readHttpURL(HttpURLConnection url) throws IOException {

  url setAllowUserInteraction (true);
  verbose(commandName + ": Contacting the URL . .");
  url.connect();

  verbose(commandName + "· Connect. Waiting for reply ...");
  DataInputStream in = new DataInputStream(url.getInputStream());

  try {
   if (url.getResponseCode() != HttpURLConnection.HTTP_OK) {
    out.println(commandName + ": " + url.getResponseMessage());
   } else {
    printHeader(url);
    while (true) {
     writeChar((char) in.readUnsignedByte());
    }
   }
  } catch (EOFException e) {

   verbose("\n");
   verbose(commandName + ": HTTP/1.0 " + url.getResponseCode() +" " + url.getResponseMessage());
   url.disconnect();

  } catch (IOException e) {
   out.println( e + ": " + e.getMessage());
   verbose("\n");
   out.println(commandName + ": I/O Error " + url.getResponseMessage());
  }
  System.exit(0);
 }

 public static final void printHeader(URLConnection url) {
  verbose(wget.class.getName() + ": Content-Length   : " + url.getContentLength() );
  verbose(wget.class.getName() + ": Content-Type     : " + url.getContentType() );
  if (url.getContentEncoding() != null)
   verbose(wget.class.getName() + ": Content-Encoding : " + url.getContentEncoding() );
   verbose("");
 }

 public final static void writeChar(char c) {
  out.print(c);
 }

 public static final void verbose(String s) {
  out.println( s );
 }

}
```

Esta clase es la que se utilizará junto con las clases URLConnection y URL, para el desarrollo de la aplicación distribuida de diagnóstico de vínculos, la cual está basada totalmente sobre el protocolo HTTP. Estas clases dan la funcionalidad suficiente para el manejo del protocolo HTTP, en el cuál esta basado el análisis de vínculos que realiza la aplicación. Así se evita la utilización de sockets directamente, ya que estas clases hacen uso de los sockets pero a un nivel mucho más abstracto y dosificado, dejando en Java el manejo a bajo nivel de las comunicaciones entre máquinas. Sólo se debe preocupar por crear las clases y realizar solicitudes, Java se encargará de abrir y cerrar las conexiones.

6
Aplicaciones distribuidas

Los programadores que diseñan aplicaciones para un entorno de computación distribuido tienden a seguir una directriz común: "Conseguir que la aplicación distribuida tenga un comportamiento lo más parecido posible a la versión no distribuida".

En este capítulo se definirá qué son las aplicaciones distribuidas y cuál es su funcionamiento básico. Cuáles son los objetivos y características de los sistemas distribuidos. Algunos tipos de sistemas distribuidos. Además, se verá como diseñar y construir una aplicación distribuida con Java RMI. Elementos de diseño e implementación. Se verán varias cuestiones y aclaraciones sobre el uso de hilos de control en RMI.

6.1 Arquitectura de aplicaciones distribuidas

El objetivo principal de una computación distribuida es proporcionar un entorno que esconda la localización geográfica de los centros de cálculo y servicios dando la impresión de que son locales.

Por ejemplo, un sistema de bases de datos convencional almacena la información en la misma máquina en la que se encuentran los programas de aplicación que acceden a los datos. Una versión distribuida de dicho sistema de bases de datos permite que los usuarios accedan a la información desde computadoras distintas a aquella en la cual residen los datos. Si las aplicaciones que gestionan la base de datos distribuida están bien diseñadas, un usuario nunca sabrá si la información a la cual accede es local o remota.

Objetivos y características de los Sistemas Distribuidos

Un Sistema Distribuido es, expresado de una forma general, "aquel sistema que se ejecuta en una colección de máquinas sin memoria compartida, pero que aparece ante los usuarios como una sola computadora".

Esta propiedad es llamada por algunos autores como "la imagen de un único sistema". Otros tienen un punto de vista diferente y mencionan que un Sistema Distribuido es: "Aquel sistema que se ejecuta en una colección de máquinas enlazadas por una red pero que actúan como un uniprocesador virtual". No importa la forma en que se exprese, la idea esencial es que los usuarios no deben ser conscientes de la existencia de varios CPU's en el sistema.

No hay una característica única que permita distinguir a un Sistema Distribuido, algunos autores mencionan al incremento en la capacidad de cómputo y mejoras en la distribución y compartición de recursos como sus objetivos más importantes.

Los autores del Manual de Referencia(ANSA) definen las consecuencias de la distribución en términos de "Separación" y "Transparencia". La separación de componentes es una propiedad inherente de los Sistemas Distribuidos. Estas consecuencias incluyen la necesidad de comunicación para el sistema de administración, y de técnicas de integración. La separación permite la real ejecución en paralelo de los programas, la contención de las fallas de los componentes y la recuperación de estás sin la paralización de todo el sistema, el uso del aislamiento y el interbloqueo como un método de reforzar la seguridad y las políticas de protección, y el incremento del crecimiento o contracción del sistema mediante la adición o substracción de componentes.

El término Transparencia esta definido como: "La ocultación de la separación entre el usuario y el programador de la aplicación, tal que el sistema es percibido como un todo y no como una colección de componentes independientes". Las implicaciones de la Trasparencia son de gran influencia en el diseño de los sistemas de software.

ANSA identifica 8 formas de transparencia:

- **Acceso Transparente**: permite acceder a archivos locales o remotos y a otros tipos de objetos usando las mismas instrucciones.
- **Localización Transparente**: permite accesar a un objeto determinado sin necesidad de conocer su localización.
- **Concurrencia Transparente**: permite a varios usuarios o programas de aplicaciones a operar *concurrentemente o compartiendo datos sin una interfaz entre ellos.*
- **Replicación Transparente**: permite múltiples instancias de archivos y otros datos usados e incrementa la veracidad y desempeño sin el conocimiento de las replicas por parte de los usuarios o por los programas de aplicación.
- **Transparencia en Fallas**: posibilita el ocultamiento de fallas, permitiendo a los usuarios y programas de aplicación completar sus tareas a pesar de fallas de componentes de hardware o de software.
- **Transparencia en Migración**: permite el movimiento de los objetos del sistema sin afectar la operación de los usuarios de los programas de aplicación.

- **Transparencia en el Desempeño**: permite al sistema ser reconfigurado para mejorar el desempeño al variar la carga.
- **Escalabilidad Transparente**: permite al sistema y a las aplicaciones expandirse en forma escalar sin cambiar la estructura del sistema o los algoritmos de aplicación.

Tipos de Sistemas Distribuidos

Existen principalmente tres tipos de sistemas que pueden considerarse distribuidos:

- Sistemas Débilmente Acoplados (Cliente/Servidor)
- Sistemas Multiprocesadores Fuertemente Acoplados
- Sistemas de Array de Procesadores

Sistemas Débilmente Acoplados (Cliente/Servidor)

A los sistemas, donde la compartición de recursos necesaria para proveer un servicio integral de cómputo, está dada por algunas de las computadoras de la red y éstas son accesadas por medio del software del sistema que corre en todas estas máquinas, usando la red para coordinar su trabajo y para trasmitir los datos entre ellas, se les conoce como Sistemas Débilmente Acoplados, para distinguirlos de otros tipos de sistemas de cómputo que pueden pretender nombrarse "distribuidos".

Los Sistemas Débilmente Acoplados son particularmente efectivos en la explotación del poder de cómputo y flexibilidad que brindan las computadoras del tipo monousuario, también llamadas Estaciones de Trabajo (del termino en inglés Workstation), posibilitando el acceso compartido de datos y recursos localizados en otras computadoras que funcionan como servidores por medio de una red local de alta velocidad.

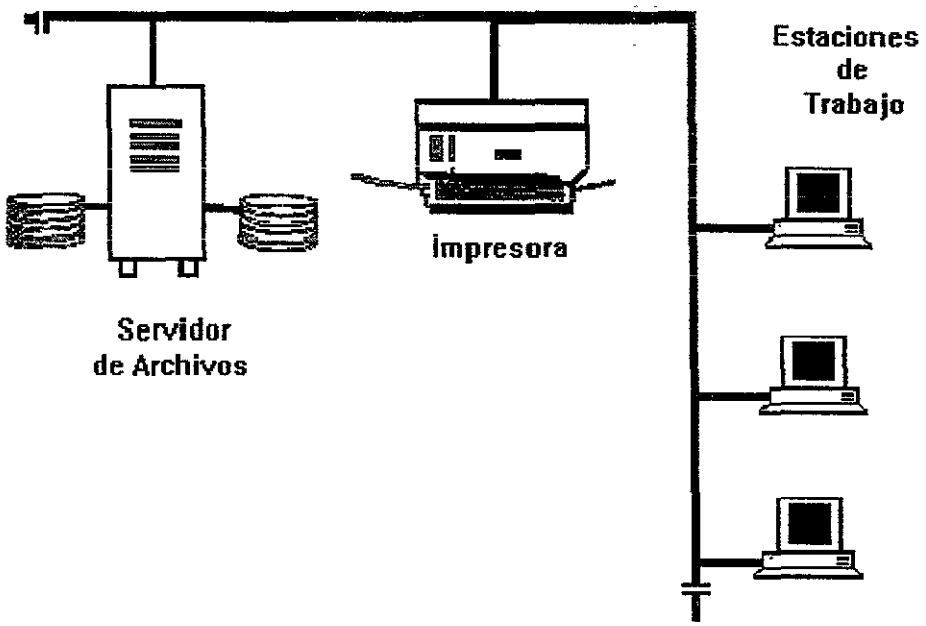


Figura 6.1 Modelo simple de un Sistema Distribuido Débilmente Acoplado Cliente/Servidor

Este modelo simple, definido en términos de una red uniendo a solo dos tipos de computadoras en el sistema, es conocido como Modelo Cliente/Servidor. La mayoría de los Sistemas Distribuidos actuales generalmente se adaptan a este modelo (ejemplo: el Sistema Distribuido Xerox y las redes de Estaciones de Trabajo Sun).

Además del Modelo Cliente/Servidor, otros dos tipos de Sistemas Débilmente Acoplados son: el Modelo de Pila de Procesadores (Pool Procesor Model) y el Modelo Integral (Integrated Model).

Modelo Cliente/Servidor

Como se vio anteriormente un sistema basado en el modelo Cliente/Servidor es algo muy parecido al mostrado en la Figura 6.2, una variante mas elaborada de este modelo seria la siguiente:

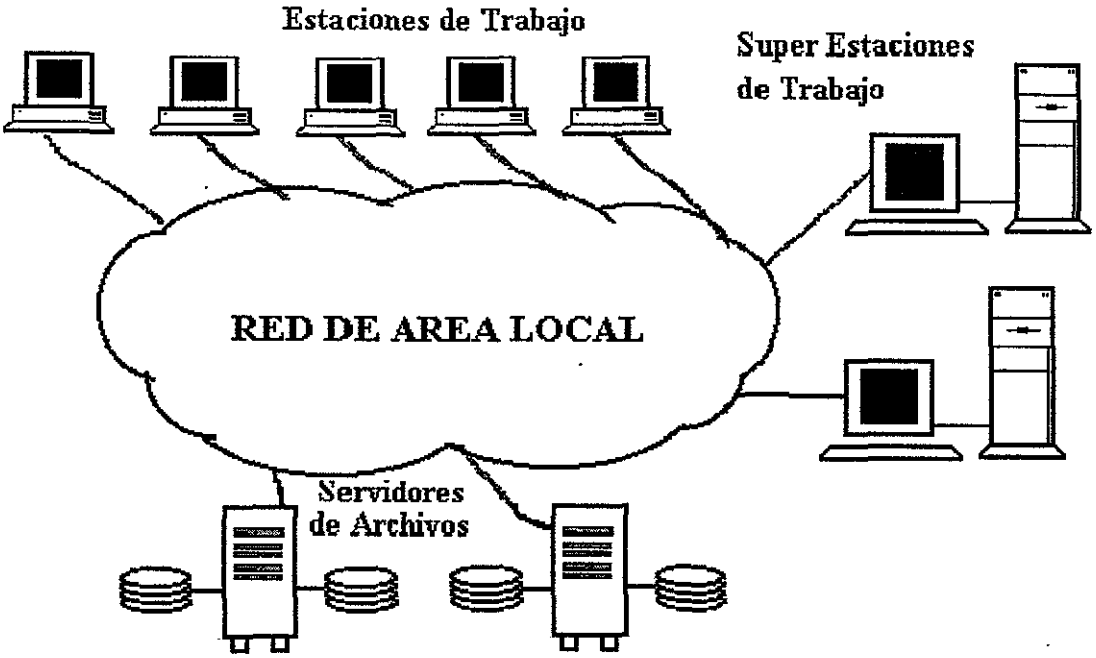


Figura 6.2 Modelo de un Sistema Cliente/Servidor.

Las Estaciones de Trabajo pueden ser de diferentes tipos, como por ejemplo, las Estaciones de Trabajo estándar y las Super Estaciones de Trabajo de alto de desempeño mostradas en la Figura 6.2. Cada Estación de Trabajo provee el poder de cómputo para ejecutar programas de aplicación y para soportar la interfaz gráfica de cada usuario, pero muchas aplicaciones requieren el acceso a archivos compartidos, tales como programas, bases de datos, tableros de avisos, librerías de software y documentación de usuario, y a otros recursos compartidos tales como impresoras y cintas magnéticas.

Las Estaciones de Trabajo están integradas por medio del uso de software de comunicación que les permiten a todas ellas el acceso al mismo conjunto de Servidores. Estos servidores proveen el acceso a los dispositivos compartidos, archivos y otros recursos de la red. Esto da como resultado la compartición simple de dispositivos de hardware tales como impresoras, acceso a datos compartidos a través de servidores de archivos y otros servicios varios que son designados para remplazar las funciones de un Sistema Operativo centralizado. Por ejemplo, un servicio de autentificación es usualmente proveído para la verificación de la identidad del usuario y autorizarlo a usar los recursos del sistema, así como también es otorgado un servicio de gateway network, permitiendo el acceso de las Estaciones de Trabajo de una LAN hacia una WAN.

Modelo de Pila de Procesadores.

El primer Sistema Distribuido basado en el Modelo de Pila de Procesadores fue el "Cambrige Distributed Computing System" desarrollado en el Laboratorio de Cómputo de la Universidad de Cambrige, Inglaterra. Los usuarios accesan al sistema desde terminales que están unidas a la red mediante computadoras que realizan la función de PAD. Los PAD son simples servidores que proveen un servicio de conexión a la red entre cada terminal y la computadora con la cual el usuario esta actualmente interactuando.

Este modelo de Sistema Distribuido se basa en un banco de procesadores con suficiente memoria para cargar y correr cualquier programa del sistema o aplicación disponible en el mismo. La Pila de Procesadores no tiene terminales o unidades de disco unidas directamente a ella, pero los usuarios pueden accesar a ella por medio de las terminales conectadas a las computadoras PAD que son las encargadas de manejar y asignar a la Pila de Procesadores especificada por el usuario, según sus requerimientos. Si el tipo de Pila de Procesadores que se requirió esta disponible, este es asignado al usuario en forma exclusiva hasta que la sesión del usuario sea terminada. Cuando la Pila de Procesadores es asignada, el servidor "esclavo" puede entonces ser requerido para cargar la Pila de Procesadores con un programa. Entonces una copia del sistema operativo es cargada en la Pila de Procesadores que está asignada al usuario, el usuario puede interactuar directamente con esta desde su terminal, emitiendo comandos para cargar y ejecutar programas, accediendo y manipulando archivos o desempeñando cualquier otra función que proporcione el sistema operativo. El usuario puede continuar trabajando con el mismo sistema operativo por el tiempo que dura la sesión, o invocando al servidor "esclavo" de nuevo para cargar un nuevo sistema operativo. Otro de los servicios proporcionados incluyen servidores de archivos, servidores de nombres (name servers), servidores de inicialización (o boot servers, este tipo servidor otorga la facilidad de cargar un programa ejecutable desde un servidor de archivos a la Pila de Procesadores asignada al usuario, o incluso arrancarla), servidores de reloj (time servers, proporcionan tomas de tiempo para la sincronización de los procesos) y servidores de impresión (que proporcionan facilidades para imprimir archivos en impresoras unidas al servidor).

El Modelo Integral

Los sistemas operativos usados en las computadoras de sistemas centralizados interpretan los nombres de archivo y de usuario con relación al sistema local de conexión, y ejecutan los programas usando el procesador local, memoria y otros recursos. Cuando varios de estos sistemas son conectados por medio de una red local, el sistema operativo de cada computadora es generalmente extendido para permitir a los usuarios de una computadora copiar archivos desde otras computadoras, así como para correr procesos en otras computadoras. En un nivel más ambicioso, un conjunto de computadoras pueden ser manejadas por un Sistema Operativo Distribuido simple que las hace aparecer a los usuarios como un sistema compuesto de una sola computadora. El Sistema Locus, desarrollado en la Universidad de California en los Ángeles, es un ejemplo de un Sistema Operativo Distribuido tipo UNIX para enlazar computadoras multiusuarios y otros tipos de computadoras. Sistemas como el Locus proveen un entorno distribuido integrado en el cual cada computadora tiene un alto grado de autonomía pero permitiendo la compartición de datos usando un esquema de nombramiento global. Las computadoras pueden ser Estaciones de Trabajo o computadoras multiusuario, pero cada una corre un conjunto completo de software estándar y trata con sus propias aplicaciones y servicios, como por ejemplo, un servicio de archivo. Cuando un programa va a ser ejecutado el sistema decide en que computadora correrá este (de forma arbitraria), localiza el archivo del programa ejecutable y lo carga en la computadora seleccionada. En tales sistemas los programas pueden leer y escribir el contenido de los archivos sin considerar su localización. Los nombres de archivo son globales, permitiendo a los usuarios en cualquier computadora en el sistema a referirse a ellos usando el mismo nombre de esquema. El mapeo de los nombres de usuario con los identificadores usados por estos, permite decidir la propiedad y permisos de acceso a los archivos, lo que es realizado de forma uniforme a través de todo el sistema. Un sistema mas o menos afín a este tipo de sistemas es el Newcastle Connection el cuál es una modificación del sistema operativo UNIX que permite a los programas a usar el archivo de operaciones de esté (por ejemplo read y write) para referirse a cualquier archivo en una interconexión de sistemas UNIX. El Sistema NFS de SUN incluye actualmente muchas de las características de Locus y Newcastle Connection.

Sistemas Multiprocesadores Fuertemente Acoplados

Los sistemas de este tipo integran a varios procesadores dentro de un sistema de hardware integrado bajo el control de un solo Sistema Operativo. El sistema operativo asigna los procesadores y los espacios de memoria a las tareas de los usuarios y les permite que estas corran concurrentemente. El hardware de entorno incluye memoria compartida o una conexión de alta velocidad entre los distintos procesadores y unidades de memorias del sistema con un solo sistema unificado de direccionamiento.

El uso de memoria compartida o espacios de direccionamiento virtuales permite a las tareas de los usuarios comunicarse unas con otras y con el sistema operativo a través de variables compartidas y tablas como en los sistemas convencionales monoprocesadores o multiprocesadores.

En los Sistemas Multiprocesadores Fuertemente Acoplados con compartición de memoria, el número de procesadores que pueden ser útilmente desplegados está limitado por el ancho de banda de la memoria. Las computadoras Multiprocesadoras Mainframes con un número pequeño de procesadores, unidos en un arreglo Fuertemente Acoplado han estado disponibles desde hace varios años por parte de los distribuidores de computadoras más importantes: por ejemplo la CDC6600 (1970) y el sistema C.MMP.

Sistemas de Array de Procesadores

Un Array de procesadores es análogo a una computadora convencional con un gran número de unidades aritmético lógicas (ALU) unidas en un array regular. Estas pueden ser usadas para desempeñar cálculos de matrices y otras operaciones regulares en forma paralela con los arrays de datos. Un ejemplo: los sistemas ICL DAP y Illiac 4.

Ventajas de los Sistemas Distribuidos

- **Respuesta predecible**: Los Sistemas Distribuidos son particularmente atractivos para los usuarios quienes tienen tareas diversas, interactivas, y que requieren de una capacidad de procesamiento significativa. Las Estaciones de Trabajo con capacidades de procesamiento por arriba de 10 MIPS ya están disponibles en este momento. La dedicación del poder de procesamiento para dar seguridad a los usuarios individuales y una rápida respuesta en el desempeño de la mayoría de las tareas interactivas.

- **Capacidad de expansión**: El administrador de un Sistema Distribuido es capaz de extender el sistema dependiendo de la demanda de servicios sin remplazar ningún componente. El poder de cómputo de los sistemas distribuidos más pequeños consiste de dos Estaciones de Trabajo y un Servidor, la capacidad de los más grandes puede consistir de hasta de varios cientos de Estaciones de Trabajo y de varios servidores de archivos, de impresión, y otros tipos de servidores de propósito especial. Las Estaciones de Trabajo y los servidores pueden ser adicionados como sea requerido por las demandas de operación o para proporcionar nuevos servicios. El parámetro limitante es el ancho de banda de la red, ya que cada Estación de Trabajo activa en la red ocupa un espacio de está para comunicarse. Las actuales redes locales han demostrado que tienen la capacidad para soportar varios cientos de Estaciones de Trabajo.

- **Compartición de recursos**: Considerar el problema de la compartición de recursos en un medio ambiente operativo que incluye a varias computadoras. Si las computadoras son componentes de un solo Sistema Distribuido, los periféricos como las impresoras y discos de almacenamiento pueden ser compartidos entre todas las computadoras en la red. Si éstas no están conectadas a cada una de las computadoras, deberán ocupar sus propios periféricos. Por ejemplo, en un Sistema Distribuido cada Estación de Trabajo podrá no tener disco de almacenamiento o solo uno pequeño para almacenamiento temporal. El acceder a archivos permanentes en un disco de gran tamaño puede ser proporcionado por un servidor de archivos. Una impresora individual puede ser compartida por el mismo medio o sea mediante un servidor de impresión.

- **Replicación**: La veracidad y rápido acceso a la información almacenada puede lograrse manteniendo varias copias de los datos en diferentes servidores.

- **Protección contra fallas**: Cuando uno de los componentes de un Sistema Distribuido falla, la mayoría de los trabajos en progreso en la red no necesita ser interrumpido. Solo el trabajo que estaba usando el componente que fallo deberá ser mudado para que ocupe un componente en buen estado, Por ejemplo, un usuario deberá cambiarse de Estación de Trabajo si la suya se descompone. Similarmente un servicio de archivo puede ser restablecido en otra computadora cuando la computadora que lo proporcionaba falla. Los servicios de archivo pueden estar presentes en más de una computadora, para ser usados cuando el servidor activo falle.

Desventajas de los Sistemas Distribuidos

- **Pérdida de flexibilidad en la asignación de memoria y recursos de procesamiento:** En un sistema centralizado o en uno bajo el modelo Fuertemente Acoplado de Multiprocesamiento, todos los recursos de procesamiento y de memoria son asignados por el sistema operativo en cualquier forma que los requiera la tarea que en ese momento este cargada. En un Sistema Operativo Distribuido la capacidad de procesamiento y de memoria de las Estaciones de Trabajo determina el tamaño de la tarea más grande que se podrá desempeñar.

- **Dependencia en el desempeño de la red y rentabilidad:** La falla en la red local causa que los servicios a los usuarios sean interrumpidos. La sobrecarga en la red degrada el desempeño y el tiempo de respuesta a los usuarios. Mucho se ha trabajado en lo referente al diseño redes confiables tolerantes a fallas, por lo que afortunadamente las fallas de la red ocurren muy poco frecuentemente, pero este pequeño margen de error puede considerarse como una desventaja en teoría.

- **Fallas de seguridad:** Para lograr una gran capacidad de expansión, muchas de las interfaces de software para los sistemas distribuidos son accesibles a los clientes. Cualquier cliente que tenga acceso a un servicio básico de comunicación puede acceder también a los servidores. Este tipo de plataforma abierta es atractiva para los desarrolladores, pero se necesitan medidas de protección de software para proteger los servicios del sistema contra violaciones de acceso intencionales o accidentales, para tener un buen control de acceso y mantener la privacidad de la información individual o de grupo. Por ejemplo el software de una Estación de Trabajo no puede ser confiable para desempeñar tareas como las de autentificación de usuarios.

Elementos de Diseño

La meta principal para el diseñador de un Sistema Distribuido es dar un efecto "Transparente" de está distribución de los elementos que componen al sistema a los ojos de los usuarios, es decir, que los usuarios vean al sistema como un todo perfectamente integrado; ellos normalmente no tendrán que estar conscientes de todos los componentes de hardware y software por los que este constituido el mismo. En suma, el sistema deberá ejecutar las tareas de los usuarios de forma "Consistentemente" y "Efectivamente".

Como referencia que pueda ayudarnos en la explicación de estos elementos del diseño, se hará referencia a 4 distintos tipos de tareas que realizan los sistemas distribuidos. El nivel de las tareas de usuario desempeñadas por un Sistema Distribuido, está definido por los programas de aplicación (esto es, programas que desempeñan cálculos, editan documentos, recuperan información desde bases de datos, que procesan datos, etc.), pero el trabajo realizado par desempeñar estas tareas debe ser considerado dentro de alguno de los 4 puntos siguientes:

1. Ejecución de Programas de Aplicación.
2. Manejo de la interfaz de usuario en favor de los programas de aplicación, esto es, controlando las entradas vía el teclado y el mouse, así como desplegando información en la pantalla de la Estación de Trabajo o terminal.
3. Acceso compartido de recursos a favor de los programas de aplicación (por ejemplo archivos de impresión).
4. Comunicación con sistemas o procesos externos en favor de los programas de aplicación (por ejemplo usando Redes de área amplia, o el manejo de eventos de tiempo real).

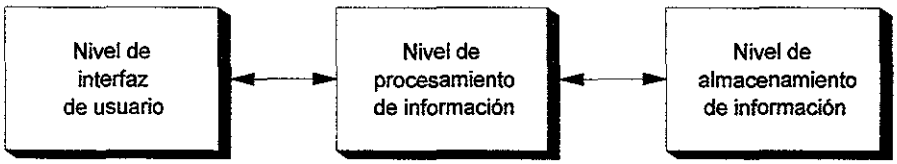
La asignación de estos 4 tipos de trabajo con una red local esta fuertemente relacionado a la arquitectura del modelo adoptado. En un modelo Cliente/Servidor, cada programa de aplicación es ejecutado en la Estación de Trabajo del usuario quien lo a solicitado; en un modelo de Pila de Procesadores en que cada aplicación es ejecutada en uno de los Procesadores de la Pila en forma dedicada; en el modelo Integral las aplicaciones corren en un sistema compartido multiusuario o en Estaciones de Trabajo compartidas.

La asignación de las tareas del sistema involucradas en los puntos 2, 3 y 4 se realizara de distintas formas y los componentes de software del sistema que desempeñan estas labores están definidos en procesos localizados en servidores aislados, es decir, corriendo estos en Estaciones de Trabajo, en computadoras-servidores aislados, o en sistemas multiusuario. En los modelos Cliente/Servidor y de Pila de Procesadores, el propósito es aislar las tareas del sistema de las de programas de aplicación y correrlas en computadoras diferentes, aunque en el modelo Integral esto es un poco diferente, esto es debido al uso de los recursos tales como Gateways, impresoras y los necesarios para la compartición de la información, por ejemplo en el uso de Bases de Datos.

Aplicaciones Distribuidas

Una aplicación distribuida es una aplicación cuyo procedimiento se distribuye por múltiples computadoras de una red. Las aplicaciones distribuidas son capaces de servir a la vez a usuarios múltiples y, dependiendo de sus diseño, hacer un uso más adecuado de lo recursos de procesamiento. Estas aplicaciones se basan en los sistemas distribuidos, pero no necesariamente implementan todas sus características.

Las aplicaciones distribuidas se implementan típicamente como sistemas Cliente/Servidor organizados en conformidad con la interfaz del usuario, el procesamiento de información y las capas de procesamiento de la información.



6.3 Organización de las aplicaciones distribuidas

La capa de interfaz de usuario viene implementada por una aplicación cliente. Los programas navegadores de Web constituyen un ejemplo de componente de interfaz de usuario de las aplicaciones distribuidas.
La capa de procesamiento de la información implementa la aplicación cliente, una aplicación servidor o una aplicación con soporte de servidor. Un ejemplo es cuando el usuario teclea una dirección URL en un navegador y ésta es interpretada de manera que se compruebe que ese URL es efectivamente una dirección URL válida.
La capa de almacenamiento de la información la implementan servidores de bases de datos, servidores de Web, servidores FTP, servidores de archivos y cualquier otro servidor cuya finalidad sea la de almacenar y recuperar información.

Aplicaciones Distribuidas en Internet

La popularidad de Internet y el Web ha supuesto la congestión de la red. Las computadoras son accesibles directamente entre sí a través del protocolo TCP/IP. Esta conectividad mundial ha hecho crecer las aplicaciones distribuidas que se ejecutan en la estructura Cliente/Servidor de Internet. Estas aplicaciones admiten la comunicación Cliente/Servidor en base a protocolos específicos de la aplicación como son HTTP, FTP y otros. Normalmente, un programa cliente se ejecuta en múltiples computadoras *host*. El cliente utiliza el TCP para conectarse con un servidor que escucha en un puerto conocido. El cliente realiza una o más solicitudes al servidor. Este servidor procesa las solicitudes del cliente, y reenvía la respuesta al cliente.

Applet de intranet

En un entorno de intranet(redes privadas que implementan TCP/IP), los sistemas de información corporativa admiten servicios que se adaptan a las necesidades organizativas de la empresa. Estos servicios constan de aplicaciones que admiten áreas productivas como son la gestión, la contabilidad, el marketing, la distribución, etc. Estos servicios de intranet pueden ser implementados por medio de servicios Cliente/Servidor, como un Web interno de la empresa.

Los applet de Java ofrecen la posibilidad de ejecutar la capa de la interfaz del cliente y parte de la capa de procesamiento de la información de aplicaciones comerciales en conjunto con un navegador de Web. Este modelo es popular porque desarrolla aplicaciones distribuidas en intranet y porque se puede utilizar con aplicaciones en Internet. Permite que navegadores, servidores Web y otros servidores distribuyan aplicaciones comerciales.

6.2 Construcción de aplicaciones distribuidas con Java RMI

Sistemas Distribuidos Orientados a Objetos

Existen varios modelos para la construcción de aplicaciones distribuidas en Internet. Siendo los más importantes:

- **El entorno de computación distribuida(DCE).** Desarrollado por la Open Software Foundation(llamada ahora Open Group). Integra una serie de servicios y tecnologías fundamentales para construir aplicaciones distribuidas. El DCE es de soporte intermedio, ya que no se trata de un producto autónomo, sino de un paquete de servicios que se integra en un sistema operativo o entorno operativo.

- **El modelo de objeto componente distribuido(DCOM).** Es el planteamiento de Microsoft al desarrollo de sistemas distribuidos. El DCOM se basa en COM, que constituye el núcleo de la estrategia de desarrollo orientado a objetos de Microsoft. Proporciona una serie de herramientas para construir aplicaciones distribuidas principalmente sobre plataformas Windows, usando como mecanismo de programación el modelo orientado a objetos.

- **La arquitectura de intermediación de solicitud de objetos remotos(CORBA).** Proporciona un verdadero enfoque orientado a objetos para el desarrollo de aplicaciones distribuidas. Es independiente del lenguaje de programación utilizado. Se reconoce como un estándar internacional y lo reconocen casi todas las principales marcas. Esto permite realizar aplicaciones con tecnologías cruzadas.

- **Invocación remota de métodos(RMI).** Implementación de Java para la construcción de aplicaciones remotas. Aunque CORBA es una implementación mucho más robusta y avanzada que RMI, tiene la desventaja de ser independiente del lenguaje, lo que hace más difícil su construcción. RMI al estar desarrollada en Java, es mucho más eficaz y fácil de usar en lo que respecta a aplicaciones desarrolladas totalmente en Java.

Java RMI: Descripción General, Modelo de Objetos Distribuidos

Arquitectura de RMI

Un sistema distribuido orientado a objetos provee los mecanismos necesarios para un manejo transparente de la comunicación, de modo que el programador se ocupe de la lógica de la aplicación. Esta idea de abstraer la parte de comunicación fue introducida por primera vez con RPC (Remote Procedure Call). RPC no sigue el paradigma de orientación a objetos por lo que el diseño de sistemas distribuidos orientados a objetos debe hacerse desde los cimientos.

El sistema de Invocación Remota de Métodos (RMI) de Java permite a un objeto que se está ejecutando en una Máquina Virtual Java (VM) llamar a métodos de otro objeto que está en otra VM diferente.

Objetivos de RMI (*Remote Method Invocation)*

1. Permitir invocación remota de métodos en objetos que residen en diferentes Máquinas Virtuales
2. Permitir invocación de métodos remotos por Applets
3. Integrar el Modelo de Objetos Distribuidos al lenguaje Java de modo natural y preservando en lo posible la semántica de objetos en Java

4. Permitir la distinción entre objetos locales y remotos
5. Preservar la seguridad de tipos (type safety) dada por el ambiente de ejecución Java
6. Permitir diferentes semánticas en las referencias a objetos remotos: no persistentes (vivas), persistentes, de activación lenta
7. Mantener la seguridad del ambiente dada por los Security Managers y Class Loaders
8. Facilitar el desarrollo de aplicaciones distribuidas

Funcionalidades requeridas:

- Localización de objetos remotos: En RMI se implementa un Registry (registro) que actúa como servidor de nombres y permite la localización de objetos remotos
- Comunicación con objetos remotos: A cargo del Sistema RMI, es transparente para el programador
- Paso de parámetros y resultados (objetos locales y remotos) en métodos remotos: Incluye el manejo de referencias y la carga dinámica de clases.

El Sistema RMI se puede analizar en términos de un modelo de capas. Cada capa tiene una función específica. El programador solo se ocupa de la capa de Aplicación; las demás capas son responsabilidad del sistema RMI.

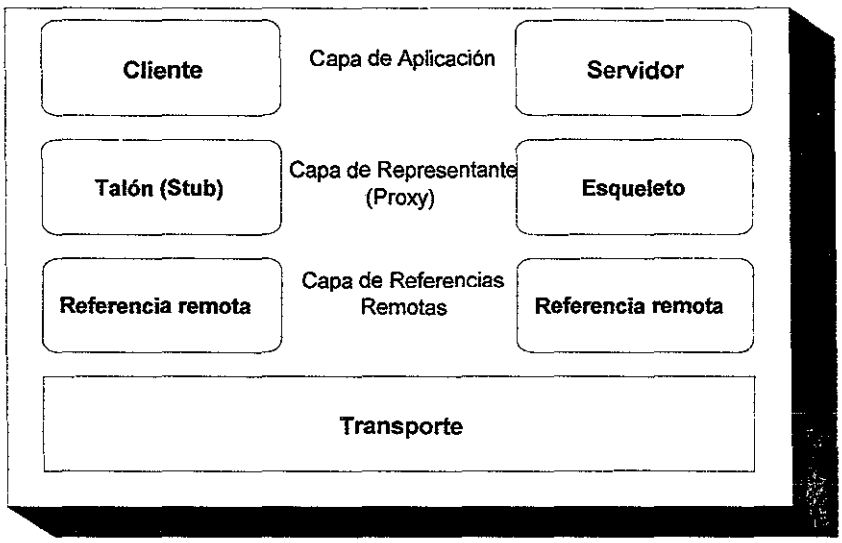


Figura 6..4 Modelo de capas RMI

La función de las distintas capas se explica a continuación:

- Capa de Aplicación. En esta capa se encuentran los objetos que implementan a la aplicación. En general en este nivel se distinguen dos tipos de agentes: los clientes, que son objetos que invocan métodos o hacen peticiones a otros objetos remotos y los servidores, que son objetos que reciben peticiones de otros objetos remotos.

- Capa de Representantes (Proxy). En esta capa se encuentran los objetos que actúan como representantes locales de objetos remotos. Se encargan del embalaje y desembalaje (marshalling) de las invocaciones, argumentos y resultados de métodos remotos. En RMI existen dos tipos de representantes: los talones (stubs) del lado de los clientes y los esqueletos (skeletons) del lado de los servidores.

- Capa de Referencias Remotas. En esta capa se realiza la interpretación de las referencias a objetos remotos, las referencias locales a talones o esqueletos se resuelven a sus contrapartes remotas y estos datos, junto con los paquetes "embalados" (marshalled) que contienen las invocaciones o los resultados de invocaciones se pasan a la Capa de Transporte.

- Capa de Transporte. En esta capa se encuentra el protocolo de comunicación, se encarga de transportar los mensajes que intercambian los distintos objetos. RMI utiliza por omisión el protocolo TCP/IP.

Creación de una Aplicación distribuida

Las aplicaciones RMI normalmente comprenden dos programas separados: un servidor y un cliente. Una aplicación servidor típica crea un montón de objetos remotos, hace accesibles unas referencias a dichos objetos remotos, y espera a que los clientes llamen a estos métodos u objetos remotos. Una aplicación cliente típica obtiene una referencia remota de uno o más objetos remotos en el servidor y llama a sus métodos.

RMI proporciona el mecanismo por el que se comunican y se pasan información del cliente al servidor y viceversa. Cuando es una aplicación algunas veces nos referimos a ella como: Aplicación de Objetos Distribuidos.

Funcionamiento básico:

Servidor:

- Crea objeto remoto
- Crea referencia al objeto remoto
- Espera a que un cliente invoque un método en el objeto remoto

Cliente:

- Obtiene referencia a un objeto remoto en el servidor
- Invoca un método remoto

Las aplicaciones de objetos distribuidos necesitan:

- **Localizar Objetos Remotos**
  Las aplicaciones pueden utilizar uno de los dos mecanismos para obtener referencias a objetos remotos. Puede registrar sus objetos remotos con la facilidad de nombrado de RMI rmiregistry. O puede pasar y devolver referencias de objetos remotos como parte de su operación normal.

- **Comunicar con Objetos Remotos**
  Los detalles de la comunicación entre objetos remotos son manejados por el RMI; para el programador, la comunicación remota se parecerá a una llamada standard a un método Java.

- **Cargar Bytecodes para objetos que son enviados**
  Como RMI permite al llamador pasar objetos Java a objetos remotos, RMI proporciona el mecanismo necesario para cargar el código del objeto, así como la transmisión de sus datos.

La figura 6.7 muestra una aplicación RMI distribuida que utiliza el registro para obtener referencias a objetos remotos. El servidor llama al registro para asociar un nombre con un objeto remoto. El cliente busca el objeto remoto por su nombre en el registro del servidor y luego llama a un método. La figura también muestra que el sistema RMI utiliza una servidor Web existente para cargar los byte-codes de la clase Java, desde el servidor al cliente y desde el cliente al servidor, para los objetos que necesita.

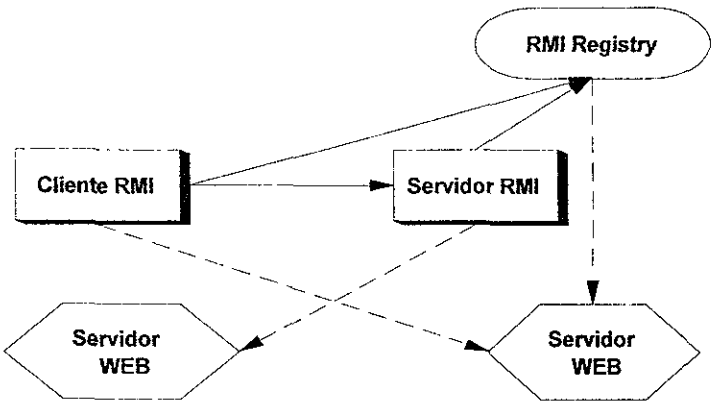


Figura 6.5 El sistema RMI utiliza un servidor Web para cargar los byte-codes de la clase Java, desde el servidor al cliente y desde el cliente al servidor.

Ventajas de la Carga Dinámica de Código

Una de las principales y únicas características de RMI es la habilidad de descargar los byte - codes (o simplemente, código) de una clase de un objeto si la clase no está definida en la máquina virtual del recibidor. Los tipos y comportamientos de un objeto, anteriormente sólo disponibles en una sola máquina virtual, ahora pueden ser transmitidos a otra máquina virtual, posiblemente remota. RMI pasa los objetos por su tipo verdadero, por eso el comportamiento de dichos objetos no cambia cuando son enviados a otra máquina virtual. Esto permite que los nuevos tipos sean introducidos en máquinas virtuales remotas, y así extender el comportamiento de una aplicación dinámicamente.

Interfaces, Objetos y Métodos Remotos

Una aplicación distribuida construida utilizando RMI de Java, al igual que otras aplicaciones Java, está compuesta por interfaces y clases. Las interfaces definen métodos, mientras que las clases implementan los métodos definidos en las interfaces y, quizás, también definen algunos métodos adicionales. En una aplicación distribuida, se asume que algunas implementaciones residen en diferentes máquinas virtuales. Los objetos que tienen métodos que pueden llamarse por distintas máquinas virtuales son los objetos remotos.

Un objeto se convierte en remoto implementando una interfaz remota, que tenga estas características:

- Una interfaz remota desciende de la interfaz java.rmi.Remote.
- Cada método de la interfaz declara que lanza una java.rmi.RemoteException, además de cualquier excepción específica de la aplicación.

El RMI trata a un objeto remoto de forma diferente a como lo hace con los objetos no-remotos cuando el objeto es pasado desde una máquina virtual a otra. En vez de hacer una copia de la implementación del objeto en la máquina virtual que lo recibe, RMI pasa un "stub" para un objeto remoto. El "stub" actúa como la representación local o "proxy" del objeto remoto y básicamente, para el llamador, es la referencia remota. El llamador invoça un método en el "stub" local que es responsable de llevar a cabo la llamada al objeto remoto.

Un "stub" para un objeto remoto implementa el mismo conjunto de interfaces remotas que el objeto remoto. Esto permite que el "stub" sea tipeado a cualquiera de las interfaces que el objeto remoto implementa. Sin embargo, esto también significa que sólo aquellos métodos definidos en una interfaz remota están disponibles para ser llamados en la máquina virtual que lo recibe.

Crear Aplicaciones Distribuidas utilizando RMI

Cuando se utiliza RMI para desarrollar una aplicación distribuida, se debe seguir estos pasos generales:

1. Diseñar e implementar los componentes de la aplicación distribuida.
2. Compilar los fuentes y generar stubs.
3. Hacer las clases accesibles a la red.
4. Arrancar la aplicación.

Diseñar e implementar los componentes de nuestra aplicación distribuida.

Primero, decidimos la arquitectura de nuestra aplicación y se determina qué componentes son objetos locales y cuales deberían ser accesibles remotamente. Este paso incluye:

- **Definir las Interfaces Remotas.** Un interfaz remota especifica los métodos que pueden ser llamados remotamente por un cliente. Los clientes programan las interfaces remotas, no la implementación de las clases de dichas interfaces. Parte del diseño de dichas interfaces es la determinación de cualquier objeto local que sea utilizado como parámetro y los valores de retorno de esos métodos; si alguno de esas interfaces o clases no existen aún también tenemos que definirlos.
- **Implementar los Objetos Remotos.** Los objetos remotos deben implementar uno o varias interfaces remotas. La clase del objeto remoto podría incluir implementaciones de otras interfaces (locales o remotos) y otros métodos (que sólo estarán disponibles localmente). Si alguna clase local va a ser utilizada como parámetro o cómo valor de retorno de alguno de esos métodos, también debe ser implementada.
- **Implementar los Clientes.** Los clientes que utilizan objetos remotos pueden ser implementados después de haber definido las interfaces remotas, incluso después de que los objetos remotos hayan sido desplegados.

Compilar los fuentes y generar stubs.

Este es un proceso de dos pasos. En el primer paso, se utiliza el compilador javac para compilar los archivos fuentes de Java, los cuales contienen las implementaciones de las interfaces remotas, las clases del servidor, y del cliente. En el segundo paso es utilizar el compilador rmic para crear los stubs de los objetos remotos. RMI utiliza una clase stub del objeto remoto como un proxy en el cliente para que los clientes puedan comunicarse con un objeto remoto particular.

Se necesita entonces:

- javac para compilar las fuentes
- rmic para generar los stubs

Hacer accesibles las clases en la red.

En este paso, se tiene que hacer que todos los archivos de clases Java asociados con las interfaces remotas, los stubs, y otras clases que necesitemos descargar en los clientes, sean accesibles a través de un servidor Web.

Arrancar la aplicación.

Arrancar la aplicación incluye ejecutar el registro de objetos remotos de RMI, el servidor y el cliente.

Se necesita entonces:

- Ejecutar el registro remoto RMI de objetos: rmiregistry

- Ejecutar el servidor
- Ejecutar el cliente

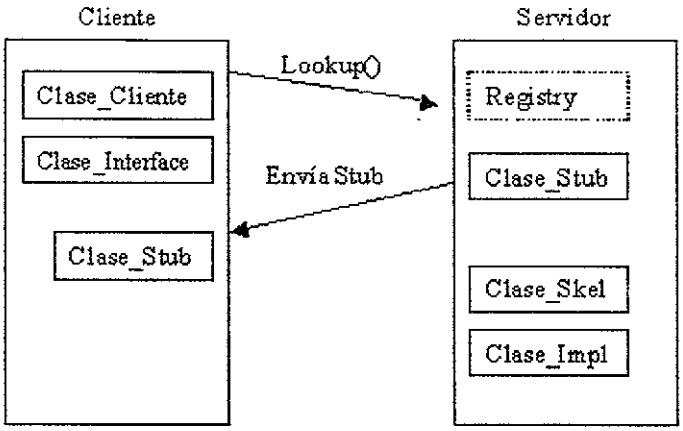


Figura 6.6 Funcionamiento de una aplicación distribuida RMI Java

La habilidad para realizar tareas arbitrarias esta permitida por la naturaleza dinámica de la plataforma Java, que se extiende a través de la red mediante RMI. Una aplicación que tiene la habilidad de descargar código dinámicamente recibe el nombre de "aplicación basada en comportamiento". Dichas aplicaciones normalmente requieren infraestructuras que permitan agentes. Con RMI, dichas aplicaciones son parte del mecanismo básico de programación distribuida de Java.

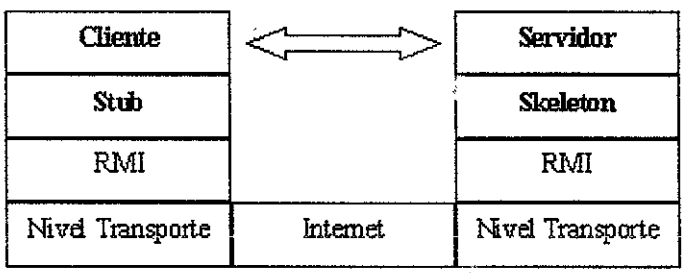


Figura 6.7 Esquema a bajo nivel de una aplicación distribuida RMI

Definiciones básicas para generar aplicaciones distribuidas con java RMI

Invocación a Método Remoto:

- Acción de invocar un método de una interfaz remota en un objeto remoto
- Tiene la misma sintaxis de una invocación a un método local

Difiere de una invocación local en:

- Argumentos no remotos en invocaciones remotas se pasan por valor (copia)
- Objetos remotos se pasan por referencia
- Hay Excepciones Remotas. Estas excepciones no son de tipo runtime, por lo que deben ser cachadas o lanzadas de manera adecuada.

Paso de Parámetros a Métodos Remotos

Hay 3 Mecanismos básicos (paso de argumentos y resultados)

1. Tipos primitivos: se pasan por valor (copia). todos son serializables
2. Objetos Remotos: se pasan por referencia (talones, usados para invocar métodos remotos).
3. Objetos Locales: se pasan por valor (solo si son serializables), se crea un nuevo objeto en la Máquina Virtual que recibe la copia.

Seguridad

Un programa RMI tiene las siguientes características básicas para establecer seguridad:

- RMI instala un Administrador de Seguridad
- El Administrador asegura que las clases cargadas cumplan el estándar de seguridad de Java
- Una aplicación puede definir sus propias políticas de seguridad en vez de utilizar el Administrador de RMI

Observaciones

- Los objetos remotos no viajan, en cambio se envían referencias (que permiten crear talones o sustitutos) a quienes los reciben, ya sea como argumentos de un método o como resultados de un método
- Un talón se debe convertir al tipo (cast) de las interfaces remotas que implemente la clase del objeto remoto al que corresponde. El tipo del talón determina que métodos remotos se pueden invocar en el objeto remoto (aquellos que se declaran en la interfaz correspondiente)
- Si un objeto remoto implementa varias interfaces remotas un cliente solo puede convertir el talón a una de ellas (y por lo tanto solo podrá invocar los métodos declarados en esa interfaz)
- Dos talones que se refieren al mismo objeto remotos en el mismo servidor se consideran iguales bajo la operación equals

Java RMI: Hilos de Control

Cuestiones y aclaraciones sobre el uso de hilos de control en RMI. Un hilo de control es un proceso hijo independiente que ejecuta una tarea especifica. Esto es lo que realmente sucede en la interacción de un cliente con un servidor de RMI.

1 Grupos de Hilos de Control

Cada hilo de control es miembro de un grupo (ThreadGroup), por omisión es miembro del grupo que lo crea. Esto permite establecer políticas de seguridad en términos del acceso a operaciones sobre hilos: un hilo solo puede iniciar o detener a hilos de su grupo. Los grupos de hilos forman una jerarquía tipo árbol.

2 Atomicidad de operaciones en Java

La Máquina Virtual de Java garantiza que las operaciones sobre tipos primitivos son atómicas y funcionan bien en contextos de invocación concurrente sin sincronización explícita. Se incluyen aquí todas las operaciones de acceso y asignación a todos los tipos escalares excepto long y double, las cuales si hay concurrencia se efectúan con entrelazado (interleaving).

3 Hilos de Control en RMI

La invocación concurrente de métodos remotos se ejecuta de manera automática en hilos de control separados, según sea necesario. De modo que la especificación no es clara al respecto, la invocación de un método remoto sí se efectúa en un hilo separado, se tiene verdadera invocación concurrente. Las invocaciones sucesivas de un mismo cliente se realizan en un mismo hilo para optimizar manejo de recursos. En general se tiene un hilo por cliente de un objeto remoto.

Si la implementación de un método remoto está sincronizada, aunque las invocaciones de varios clientes se efectúen en hilos separados se tendrá que el acceso al método es mutuamente exclusivo. Supongamos que un cliente pide varias referencias de un mismo objeto remoto, ¿cuántas conexiones al servidor tiene?. Básicamente tendrá:

a. Una conexión al registro (a cargo de las llamadas lookup)
b. Una conexión al objeto remoto (a cargo de las llamadas DGC dirty)
c. Una conexión por método remoto en proceso

En general se tiene que un objeto remoto tiene una conexión abierta (socket) por la que escucha invocaciones de clientes. Del lado del cliente hay varios hilos a cargo de las llamadas al recolector de basura que están pendientes de renovar los contratos (leases), de ser posible estas llamadas se hacen usando las conexiones abiertas, pero en caso necesario se abren nuevas conexiones. Adquirir varias referencias a un objeto remoto no abre nuevas conexiones.

4 ¿Cuál es el costo de RMI en términos de hilos de control?

- Cada objeto remoto tiene un hilo escuchando el puerto donde se exporta el objeto.
- Cada invocación aceptada se atiende en un hilo separado, éste lee los parámetros y devuelve el resultado.
- En el cliente, el hilo que hace la invocación se bloquea mientras ésta se procesa y recibe el resultado.
- Cada objeto remoto tiene un hilo encargado de cerrar conexiones de clientes cuando éstas vencen (reaper thread).
- Uno o varios hilos secundarios (background) detectan cuando las referencias remotas se recolectan y hacen llamadas al recolector distribuido para renovar o desechar contratos con el objeto remoto.
- Es importante destacar que un objeto remoto solo atiende en un puerto, aunque exporte muchas referencias a los clientes.

5 Escalabilidad de RMI

El número de hilos que pueden crearse en el servidor para atender a clientes está limitado por el sistema operativo, en Windows es de aprox. 150. Este asunto es poco mencionado en la documentación, tampoco se mencionan qué factores determinan la escalabilidad de RMI.

EJEMPLO

Pasos a seguir para crear una Aplicación Distribuida con RMI

1. Editar los archivos Java fuente

- Se define la interfaz remota

```
package examples.hello;

public interface Hello extends java.rmi.Remote {
    String sayHello() throws java.rmi.RemoteException;
}
```

- Implementación de la interfaz e implementación del servidor

```
package examples.hello;

import java.rmi.*;
import java.rmi.server.*;

public class HelloImpl  extends UnicastRemoteObject implements Hello
{
```

```
        public HelloImpl() throws RemoteException {
            super();
        }

        public String sayHello() throws RemoteException {
            return "Hola Mundo !";
        }

        public static void main(String args[])
        {
            // Crea e instala el administrador de seguridad
            System.setSecurityManager(new RMISecurityManager());
            try {
                HelloImpl obj = new HelloImpl( );
                Naming.rebind("hello",obj);
                System.out.println("HelloServer registrado");
            } catch (Exception e) {
              System.out.println("HelloImpl err: " + e.getMessage());
            }
        }
    }
```

- Implementación del cliente

```
package examples.hello;

  import java.awt.*;
  import java.rmi.*;

  public class HelloApplet extends java.applet.Applet {
      String message = "";
      public void init() {
          try {
              Hello obj = (Hello)Naming.lookup("Hello");
              message = obj.sayHello();
           } catch (Exception e) {
              System.out.println("HelloApplet exception: " +  e.getMessage()); }
      }
      public void paint(Graphics g) {
          g.drawString(message, 25, 50);
      }
  }
```

- Implementación de una página HTML

```
<HTML>
<title>Hola Mundo</title>
<center> <h1>Hola Mundo</h1> </center>
El mensaje desde el servidor HelloServer es:
<p>
<applet codebase="../.."  code="examples.hello.HelloApplet"  width=500 height=120>
</applet>
</HTML>
```

2. Compilar los archivos fuente

```
% javac examples/hello/*.java
```

3. Generar el stub y el skeleton

% rmic <clase que implementa la interfaz>

4. Iniciar el RMI Registry

% rmiregistry

5. Ejecutar el servidor

% java <servidor>

6. Ejecutar el cliente

% java <cliente> si es una aplicación.

O cargar la página web que contiene el applet cliente.

7

Aplicación distribuida de diagnóstico de vínculos de un URL en un servidor de WWW

Ya se vio cómo funciona la red Internet y cómo funciona el protocolo HTTP. También se vio cómo programar aplicaciones en red con Java. Además, como realizar aplicaciones distribuidas con el paquete RMI de Java.

En este el último capítulo se verá como realizar una aplicación, que aprovechando las ventajas que ofrece el servicio WWW y las funcionalidades que ofrece el protocolo HTTP, se pueden crear aplicaciones con Java que se pueden distribuir fácilmente por la red Internet y ejecutarse en la máquina local, todo esto de una manera sencilla y transparente para el usuario.

Lo que realizará esta aplicación distribuida es un análisis de los vínculos rotos de un sitio WWW. Funcionará como un robot escudriñador de vínculos que dirá el status de cada vínculo(link o liga) de un sitio de Web. La aplicación funcionará bajo el esquema de Cliente/Servidor. El servidor será una aplicación independiente RMI y el cliente será un RMI – Applet.

El servicio de WWW hace uso del protocolo HTTP(HyperText Transfer Protocol) para realizar las transferencias de hipertexto o código HTML entre dos o más computadoras, así como, intercambiar recursos de multimedia (imágenes, audio, vídeo). Este protocolo está basado en el principio de cliente/servidor, como se vio en el capítulo 2, funciona básicamente de la siguiente forma: un cliente establece una conexión con el servidor para realizar una petición, el servidor acepta la conexión con el cliente y le envía una respuesta, el cliente la recibe y la interpreta. Pues para realizar una prueba de los vínculos necesitamos ejecutar una porción del procedimiento HTTP como lo hace el visualizador(browser), pero la aplicación de diagnóstico emulara el clic de mouse en todos los URL(vínculos) que encuentre en el sitio, hará la conexión y la solicitud al servidor de WWW que haya sido especificado, y en cuanto el software verifique que el recurso solicitado está en el servidor (es decir, existe), la conexión se cancela y con esto el vínculo se confirma sin necesidad de bajar los archivos reales. Con esto se comprueban todos los vínculos de un sitio y cuál es su estado.

Para poder implementar la aplicación se necesita del manejo del protocolo HTTP(HyperText Transfer Protocol) para poder hacer la conexión con los servidores de WWW y poder accesar su información. Para realizarlo, a un nivel de programación de aplicaciones, se hará uso de clases de Java que implementan sockets de comunicación entre procesos de red y realizan las conexiones aún nivel abstracto de programación. Estas clases realizan la conexión entre el servidor HTTP y la aplicación. El servidor HTTP nos devolverá un resultado a la petición que le realizamos. Este resultado contendrá información sobre el status del vínculo.

La aplicación funcionará como una aplicación Cliente/Servidor, y se podrá distribuir por la red aprovechando las facilidades del esquema de Applets de Java. La aplicación hará uso del paquete RMI para establecer la comunicación entre el cliente y el servidor. Llamaremos a la aplicación JURLrobot.

¿ Porqué en Java ?

Se usará el lenguaje de programación Java, porque maneja la programación en redes con sockets de conexión a un nivel bastante amplio y dosificado. Incluso ya tiene implementadas varias clases que hacen uso del protocolo HTTP.

Además, Java es un lenguaje bastante portable, cuyo código puede correr sobre casi cualquier plataforma sin necesidad de volver a compilarlo. Java permite la programación de Applets, los cuales son programas que pueden ser ejecutados desde un navegador o visualizador de documentos Web, teniendo así, una aplicación multiplataformas y distribuida a la vez.

Como se vio en el capítulo anterior, Java proporciona los mecanismos necesarios para la programación de aplicaciones distribuidas (librerías de RMI). Una aplicación distribuida permite ejecutar parte de la aplicación en una computadora (servidor) y otra parte (cliente) desde cualquier otra computadora conectada a la red. En nuestro caso, la red será la red global de Internet, por lo que se tendrá un numero ilimitado de clientes que ejecuten la aplicación y se conecten a un solo servidor. Esto genera un ahorro en tiempo de ejecución y uso de memoria, para cada cliente. Java además, permite la programación de aplicaciones multitarea, es decir, aplicaciones que pueden estar realizando varios procesos dentro de la misma aplicación, teniendo un ahorro en tiempo de ejecución aún más grande. Se tendrá con esto, un mayor rendimiento en al aplicación ya que al estar corriendo procesos independientes se pueden dividir tareas, asignando a cada proceso una tarea diferente, consiguiendo con esto disminuir considerablemente el tiempo de ejecución del programa.

Qué elementos necesitamos

- Se necesita un software para desarrollo de aplicaciones Java.

Se escogió JDK (Java Development Kit) de Sun MicroSystems por ser el más documentado y estar desarrollado por la empresa que creó Java. Se usará la versión 1.1.8 del JDK. Se uso la versión 1.1.8 por ser una versión que puede generar código que los navegadores de Web actuales pueden interpretar. El navegador Netscape 4.8 para Windows o Netscape 4.05 para Unix pueden interpretar código Java de Applets desde versiones 1.0.5 hasta 1.1.8. El navegador IE puede soportar Java Applets desde su versión 3, pero son las versiones 4 y 5 las que mejor trabajan con Java Applets, estas soportan versiones de Java desde las versiones 1.1.1 hasta 1.1.8. Aunque ya está en el mercado la versión 1.2 de Java la aplicación no la usa hasta que la mayoría de los navegadores puedan soportarla.

- Se desarrollará sobre plataforma Linux Red Hat versión 6.2

Usamos Linux, por ser el sistema operativo que ofrecía mayor rendimiento para el desarrollo de aplicaciones en red sobre plataformas PC. Como la aplicación trabajara sobre un entorno de red conectado a Internet, Linux era la opción mas accesible, barata y confiable. Además, que soporta perfectamente el paquete JDK de Java de Sun MicroSystems (en su versión portada a Linux por el proyecto Java-Linux http://www.blackdown.org). Además, Linux soporta perfectamente todo el ambiente de desarrollo de los protocolos TCP/IP y la interfaz socket.

- Una conexión a Internet

Para poder probar el sistema se requiere una conexión a Internet, ésta puede ser vía modem, o una conexión directa que este conectada sobre una red LAN. Esta es básica si se pretende probar el sistema a un nivel alto. Sin embargo, al momento del desarrollo no es necesario, ya que Linux nos proporciona las herramientas suficientes para no necesitar está conexión (servidor HTTP, navegador, soporte de comunicación TCP/IP específicamente HTTP, etc.). Al momento de las pruebas finales una conexión a Internet es indispensable.

- Un servidor de HTTP dedicado conectado a Internet

Como se dijo en el punto anterior, no es necesario al momento del desarrollo tener una máquina dedicada para que nos de el servicio de un servidor de HTTP, ya que Linux lo puede implementar localmente sin necesidad de estar conectado a Internet. Pero para poder distribuir la aplicación cuando este terminada se necesita forzosamente un servidor de HTTP dedicado, éste nos servirá para que los clientes se puedan conectar al sistema accesando una dirección URL específica. Este servidor deberá estar corriendo siempre en una estación de trabajo o server, que por supuesto este conectada a Internet de manera dedicada, para que espere conexiones en cualquier momento.

El servidor HTTP será Apache versión 1.3.3 y el server será una Ultra Sparc 10 sobre sistema Solaris 2.6. Esta máquina se encuentra en el laboratorio del IIMAS del posgrado en Ciencia e Ingeniería en Computación.

Una vez que se tienen todos los elementos se puede comenzar a desarrollar la aplicación distribuida por medio del lenguaje Java.

7.1 Aplicación distribuida para el análisis de vínculos que contiene algún servidor de WWW haciendo uso del protocolo HTTP

El esquema básico de la aplicación en diagrama de bloques es el siguiente:

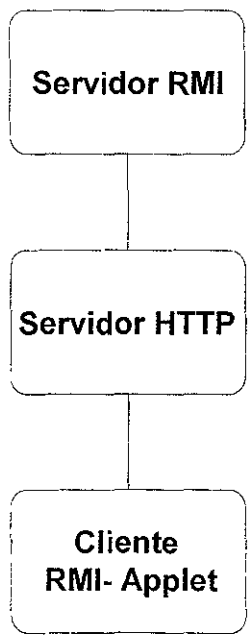


Figura 7.1 Esquema básico de la aplicación

La aplicación se divide en 3 bloques básicos. A continuación se describirán cada uno de ellos:

El Servidor RMI

Es una aplicación realizada en Java, que estará corriendo en una máquina servidor (Server Sun Ultra 10). Este programa será el encargado de realizar las transacciones que los clientes RMI - Applet soliciten. Esta aplicación implementa Java RMI(Remote Method Interface) para establecer la comunicación con los clientes. La comunicación entre el servidor RMI y los clientes RMI – Applet se realiza de forma transparente. RMI es el que se encarga de que los datos no pierdan su integridad y consistencia. Además, controla los posibles errores y perdidas que se pudieran dar durante la transmisión de datos por la red Internet.

Las funciones básicas de este servidor serán:

- Se conecta a un URL en un sitio Web obteniendo (petición GET) la información solicitada
- Parseara (Parser) la información obtenida del URL seleccionando solamente los vínculos (ligas o links)
- Devolverá el resultado de está selección de vínculos. Separando por tipos de links (imágenes, documentos, javascript, java applets, etc.).
- Se conecta a un URL en un sitio Web obteniendo (petición HEAD) el encabezado de status de cada vínculo (ligas o links)
- Devolverá el status de cada vínculo probado. Dirá que links son válidos y cuáles no lo son. Dirá además, información adicional de cada vínculo.
- Controlará el número de accesos de clientes al sistema para evitar que se sature.

El Servidor HTTP

Es un servidor de HTTP Apache configurado para recibir peticiones desde el puerto 80. Éste será la interfaz por la que se accesará a los clientes RMI - Applet. Como dijimos anteriormente, el sistema será accesado y ejecutado por medio de un navegador(pudiendo ser Netscape o Internet Explorer). El servidor HTTP dará el acceso a la página de inicio del sistema. De ahí los clientes serán llamados y cargados a las maquinas locales que lo vayan solicitando. La primera vez que un cliente nuevo se conecta desplegará un mensaje solicitando instalar el Plug - in Java 1.2. Se deberá instalar el Plug - in ya que de lo contrario no se podrá ejecutar el Applet. Una vez instalado la aplicación comienza a ejecutarse en la maquina local.

El Cliente RMI – Applet

Es una aplicación realizada en Java. Esta programada para correr como un applet que se llama remotamente y se ejecuta localmente en las máquinas locales que lo soliciten accesando una página Web. Implementa el esquema de RMI como el medio para comunicarse con el servidor RMI, el cual le proveerá de todas las peticiones que solicite. Una vez que se carga el programa en la máquina local está lista para hacer peticiones al servidor. Su interfaz gráfica será como la de un navegador de Web, pero en lugar de desplegar la información HTML, realizará un análisis de vínculos del sitio. RMI será el encargado de establecer la comunicación con el servidor por lo que se realiza de forma transparente.

Las funciones básicas de este cliente serán:

- Permitir la captura de un URL en un sitio Web para realizar el análisis de sus vínculos.
- Desplegar una lista de todas las ligas contenidas en el URL del sitio Web.
- Permitir el análisis de cada liga y sus correspondiente status.
- Desplegar en forma detallada el resultado del análisis de cada liga o vínculo.

Ahora se desarrollara y describirá con más detalle cada bloque destacando la forma de operación de cada una, así como su funcionamiento.

7.1.1 Servidor RMI

Como vio en el capítulo anterior, un servidor RMI funciona a nivel programación básicamente así:

- Crea objeto remoto: Define un objeto con las características para ser llamado remotamente.
- Crea referencia al objeto remoto: Define que objetos remotos serán accesibles desde una máquina remota.
- Espera a que un cliente invoque un método en el objeto remoto: La aplicación está en espera de peticiones de clientes, y cada vez que llega una nueva petición de un cliente crea un proceso independiente que atiende la petición, mientras el servidor se regresa a esperar nuevas peticiones, de manera que parezca que siempre está disponible para atender peticiones. Este funcionamiento es lo que permite atender en forma ordenada cada solicitud.

El servidor RMI realizará las siguientes 2 operaciones fundamentales cada vez que recibe una petición de un cliente RMI – Applet:

A. Establecer conexión a un URL solicitado, seleccionando sus vínculos.

B. Verificar el status de cada vínculo del URL solicitado.

A. Establecer conexión a un URL solicitado, seleccionando sus vínculos

La operación A sigue el esquema siguiente:

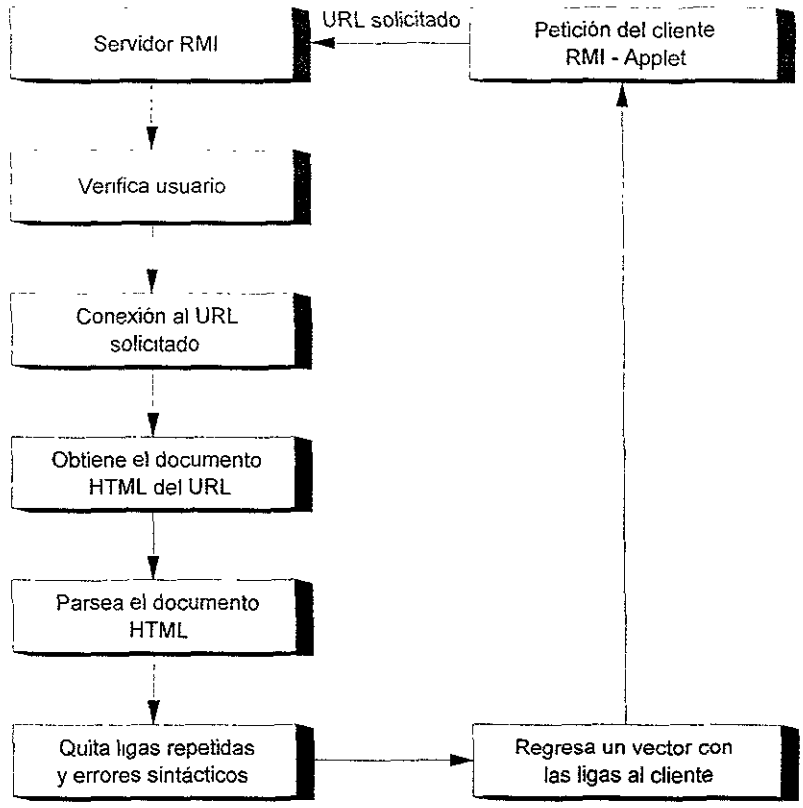


Figura 7.2 Funcionamiento del servidor RMI operación 'A.'

A continuación se describe cada una de las partes que componen al servidor RMI en su operación 'A'.

- *Servidor RMI*

Como se dijo anteriormente, el servidor está en espera a que un cliente invoque un método remoto (haga una petición) y de respuesta a cada petición recibida de parte del los clientes RMI - Applet. Estará escuchando peticiones que le lleguen por el puerto 4444(se quiso que fuera en este puerto, ya que fue establecido el RMI Registry ahí).

- *Verifica usuario a cada petición*

Cada vez que un cliente RMI - Applet realiza una petición al servidor RMI, éste verifica si la petición realizada fue hecha por algún cliente previamente registrado. El servidor sólo admitirá 20 usuarios conectados al mismo tiempo.

La primera vez que un cliente se conecta el sistema pide un 'login' y un 'password'; el login siempre será diferente a la de cualquier otro usuario (de los 20 permitidos simultáneamente) conectado en ese mismo momento, el password será siempre la palabra clave "bugaboo".

A todas las peticiones sucesivas que el cliente realice con el servidor, éste verificará si el login ya se encontraba registrado. Si esta registrado realiza la operación solicitada. De lo contrario, manda un mensaje de error.

Cada cliente RMI – Applet tendrá 10 minutos para interactuar con el sistema, y cuando termine dicho lapso de tiempo el servidor RMI retirará el 'login' de la lista de usuarios registrados. El esquema de funcionamiento es:

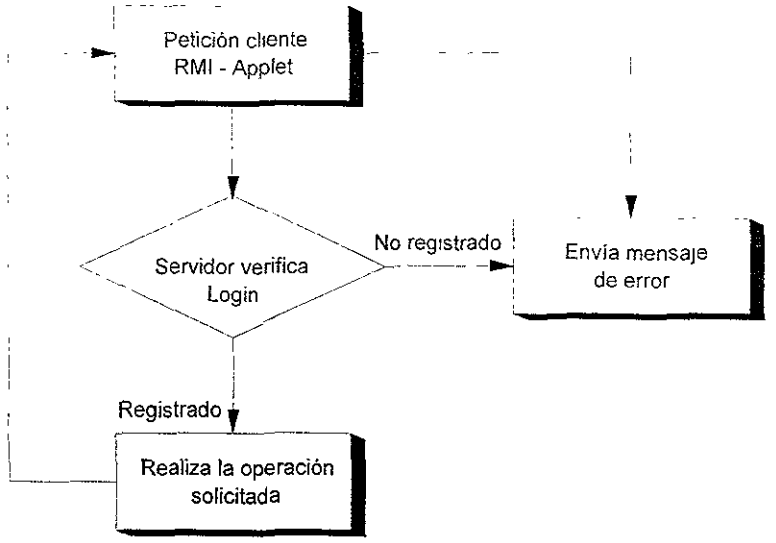


Figura 7.3 Esquema de verificación de usuario

El esquema de autentificación es muy simple, ya que la aplicación no contempla inicialmente un sofisticado sistema de seguridad y de autentificación. Sin embargo, resultaría conveniente más adelante, para futuras mejoras al sistema(que no son contempladas dentro de este trabajo de investigación), implementar un sistema de autentificación más avanzado, en donde incluso se contemple una base de datos de usuarios que pueden ingresar al sistema y un sistema de cifrado de datos para que tanto el 'login' y el 'password' viajen en forma cifrada por la red y no puedan ser robados o interceptados. Esto se verá al final, cuando se mencionen las propuestas de futuras mejoras al sistema.

- *Conexión al URL solicitado*

El cliente RMI - Applet solicita un URL específico al servidor RMI. El servidor intenta conectarse al URL solicitado y si es un URL válido se conecta por medio del protocolo HTTP al sitio. Si no es válido envía en mensaje de error con la respuesta que obtuvo de parte del servidor HTTP con el que intento establecer conexión. Si no existe el servidor HTTP o no es válido el nombre de dominio(DNS) envía también un error. Si el URL hacía referencia a una imagen, aplicación, u otro recurso que no es un documento HTML envía solo el status que el servidor HTTP regreso para tal recurso.

- *Obtiene el documento HTML del URL solicitado*

Una vez que el servidor RMI se conectó al URL solicitado obtiene el documento HTML. El servidor emplea la petición GET del protocolo HTTP para obtener el documento. Como vio en el punto anterior, sólo documentos HTML serán obtenidos, cualquier otro tipo de documento o recurso diferente no será accesado, sólo se verificara su status.

- *Quita ligas repetidas y con errores sintácticos*

Una vez que el servidor RMI selecciona y parsea las ligas del documento HTML, lo siguiente es quitar la información redundante y completar las ligas que tienen un URL relativo sustituyéndolo por un URL absoluto. Además, en corregir si se puede, las ligas que tengan errores sintácticos.

Quitar la información redundante, es quitar todos los URL que estén repetidos, ya que a veces en un solo documento HTML, puede haber varias ligas que hacen referencia a un mismo URL. Completar las que tienen un URL relativo por un URL absoluto, es necesario porque para cuando se llegue a la etapa de comprobar el status de la liga se necesita el URL absoluto para realizar ésta comprobación. Por último, se corrigen los ligas que tuvieran algún error sintáctico de código HTML.

- *Regresa un vector con las ligas al cliente*

Se almacenan los URLs, ya corregidos y seleccionados, en un vector que contendrá la información sobre los vínculos que contiene el documento HTML solicitado por el cliente. Este vector será regresado a través del mecanismo de RMI al cliente RMI - Applet que lo solicito.

B. Verificar el status de cada vínculo del URL solicitado

La operación B sigue el esquema siguiente:

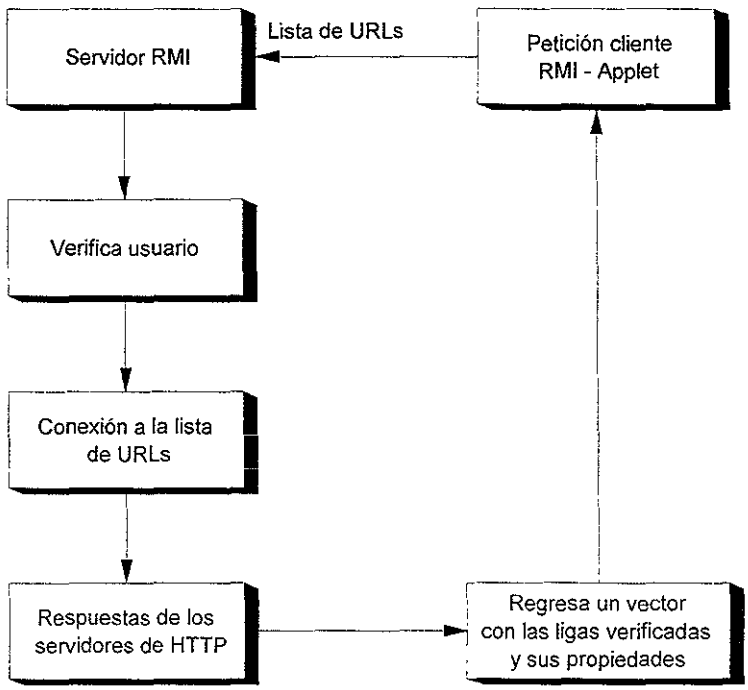


Figura 7.4 Esquema de funcionamiento del servidor RMI operación 'B'

A continuación se describirá cada una de las partes que componen al servidor RMI en su operación 'B'.

- *Servidor RMI*

Igual que en la operación 'A', el servidor está en espera a que un cliente invoque un método remoto (haga una petición) y de respuesta a cada petición realizada por parte de los clientes RMI - Applet. Estará escuchando peticiones que le lleguen por el puerto 4444(se quiso que fuera en este puerto).

- *Verifica usuario a cada petición*

Igual que la operación 'A', cada vez que un cliente RMI - Applet realiza una petición al servidor RMI, éste verifica si la petición realizada fue hecha por algún cliente previamente registrado.

Para todas las peticiones que el cliente realice con el servidor, éste verificará si el login del cliente ya se encontraba registrado. Si estaba registrado realiza la operación solicitada. De lo contrario, manda un mensaje de error. Se considera que en la operación 'A' el proceso de validación de usuario inicial ya fue realizada, así que en está operación solo se verificara que el usuario todavía este en sesión El esquema de operación es el mismo que para la operación 'A'.

- *Conexión a la lista de URLs y respuestas de los servidores de HTTP*

Estas dos operaciones se realizan en conjunto para obtener un vector que contendrá el resultado de la verificación de los vínculos. El servidor RMI verifica el status de todos los vínculos contenidos en el vector, que fue creado en la operación 'A', y que contienen la lista de URLs del sitio al que previamente se conecto el usuario. Para cada vínculo de la lista se realiza una conexión por medio del protocolo HTTP al servidor de Web correspondiente para verificar el status del vínculo.

Ahora bien, la manera que el servidor RMI verifica cada liga se muestra en el siguiente esquema:

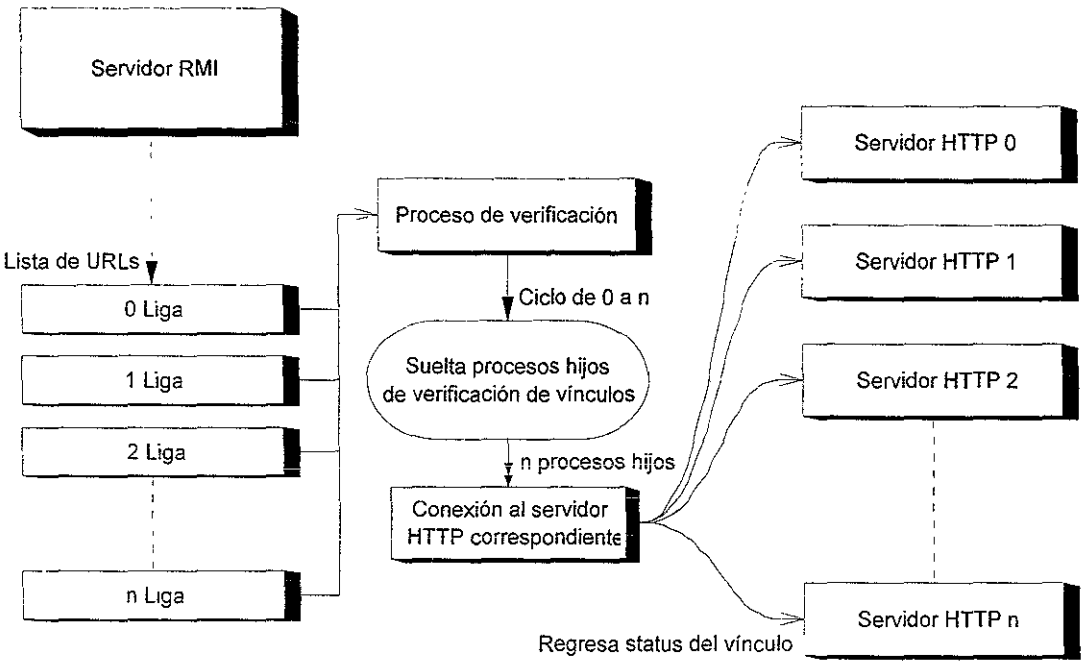


Figura 7 5 Esquema de conexión para verificación de vínculos

El servidor RMI recibe la petición de verificar la lista de URLs que le envía el cliente RMI – Applet. El servidor tiene esta lista de URLs en forma de un vector secuencial de 0 hasta n ligas. Al recibir la orden de parte del cliente de verificar el status de cada liga contenida en este vector, el servidor inicia el proceso de verificación. Ejecuta un ciclo de 0 hasta n y comienza la ejecución de procesos hijos (hilos de control o threads), estos procesos se inician en forma secuencial, pero una vez que está corriendo cada proceso el servidor no sabe cual de ellos terminara primero, compitiendo todos ellos por los recursos en round-robin(todos contra todos).

A este tipo de funcionamiento se le llama proceso no determinista, ya que cada proceso es un proceso que verifica un vínculo, y no se sabe que proceso será el que termine primero y obtenga el resultado de la verificación. Se toma en consideración de que es más probable que las primeras ligas contenidas en el vector de URLs sean verificadas primero. Esto depende de la velocidad de conexión y de la transmisión de datos que haya entre el servidor RMI y los sitios de los servidores HTTP.

La función que realizan cada uno de estos procesos es la de conectarse al servidor HTTP correspondiente a cada liga y verificar su status por medio de comandos del protocolo HTTP. Los comandos usados del protocolo HTTP son solo dos: HEAD para la verificación normal del status del vínculo, y GET para casos especiales.

- *Regresa un vector con las ligas verificadas y sus propiedades*

Una vez que el proceso de verificación inicio y los procesos hijos van terminando de verificar los URLs, se va llenando un vector con la información del vínculo y sus propiedades. Este vector contiene toda la información relevante sobre el vínculo (nombre completo del vínculo, status, tipo de vínculo, etc.). El vector es regresado a través del mecanismo de RMI al cliente RMI – Applet que hizo la petición de verificación de vínculos.

El cliente RMI - Applet, una vez que recibe la información de respuesta del servidor RMI, visualiza los datos producto de la verificación del status de cada liga. Se llena una tabla que dirá información de cómo fue el resultado de la verificación de cada vínculo. Esta tabla es visualizada por el cliente y permitirá conocer de una manera mas confiable y rápida el status de todas las ligas.

7.1.2 Servidor HTTP

Es un servidor de HTTP Apache configurado para recibir peticiones desde el puerto 80. Este servidor será la interfaz por la que se accesa a los clientes RMI - Applet. Como se dijo anteriormente, el sistema será accesado y ejecutado por medio de un navegador(pudiendo ser Netscape o Internet Explorer, en versiones recientes). Este servidor nos dará el acceso a la página de inicio del sistema. De ahí los clientes serán llamados y cargados a las maquinas locales que lo vayan solicitando. La primera vez que un cliente nuevo se conecta desplegará un mensaje solicitando instalar el Plug-in Java 1.2 y la ubicación exacta de donde bajarlo para instalarlo. Se deberá instalar ya que de lo contrario no se podrá ejecutar el Applet. Una vez instalado el Plug-in la aplicación se comenzará a ejecutar en la maquina local.

Este servidor es el que se encarga de distribuir al cliente RMI – Applet para que sea accesado por medio de un navegador común y se ejecute en las máquinas locales de los usuarios que se conectaron al sistema. Esto permite usar al protocolo HTTP como vía de comunicación e interfaz para que los clientes puedan ingresar al sistema sin necesidad de instalar un software adicional en las máquinas locales. Desafortunadamente, los navegadores actuales no soportan por default a las librerías gráficas de Swing – Java que son las que permiten dar una buena interfaz gráfica a los clientes, sin estas librerías el cliente no funcionara. Por esto, es necesario instalar el Plug – in Java 1.2, para poder correr los clientes RMI – Applet. Esto sin embargo, se deberá realizar solo la primera vez que se conecta un cliente. Esta operación se realiza automáticamente desde el IE (Internet Explorer), no así con el navegador Netscape que necesita bajar el paquete y después instalarlo. Hasta que los navegadores actuales no den soporte para Swing –Java el proceso de instalar el Plug – in Java es inevitable.

La dirección URL donde residirá el sistema en su etapa de implementación es la siguiente:

http://www.mcc.unam.mx/~avelar/jurlrobot/viewer.html

7.2 Despliegue de la información del estado de los vínculos en una interfaz gráfica (Java Applet Swing).

7.2.1 Cliente RMI – Applet

Esta es la última parte de la cual está compuesto el sistema y sirve básicamente para conectar, visualizar y analizar los resultados obtenidos del servidor RMI.

Esta programado para correr como un applet que se llama remotamente y se ejecuta localmente en las maquinas que lo soliciten, conectándose a una página Web de inicio del sistema. También implementa RMI como el medio para comunicarse con el servidor, el cual le proveerá de todas las peticiones que solicite. Una vez que se carga el programa en la máquina local está lista para realizar peticiones al servidor RMI.

Su interfaz gráfica es como la de un navegador o visualizador de Web, pero en lugar de desplegar información HTML, realizará un análisis de vínculos de los sitios Web.

Como se vio en el capítulo anterior, un cliente RMI funciona a nivel programación básicamente así:

- Obtiene referencia a un objeto remoto en el servidor: Una aplicación cliente típica obtiene una referencia remota de uno o más objetos remotos en el servidor. Esto se necesita ya que solos los objetos remotos que fueron referenciados por el servidor serán accesibles por un cliente.
- Invoca un método remoto: Realiza una petición al servidor para que ejecute un método remoto y devuelva un resultado.

Ahora se describirá más detalladamente al cliente RMI - Applet. Las funciones básicas de este cliente serán:

- Permitir la captura de un URL en un sitio Web para realizar el análisis de sus vínculos.
- Desplegar una lista de todas las ligas contenidas en el URL del sitio Web.
- Permitir el análisis de cada liga y sus correspondiente status.
- Desplegar en forma gráfica el resultado del análisis de cada liga o vínculo.

El esquema básico de funcionamiento del cliente RMI - Applet se describe a continuación :

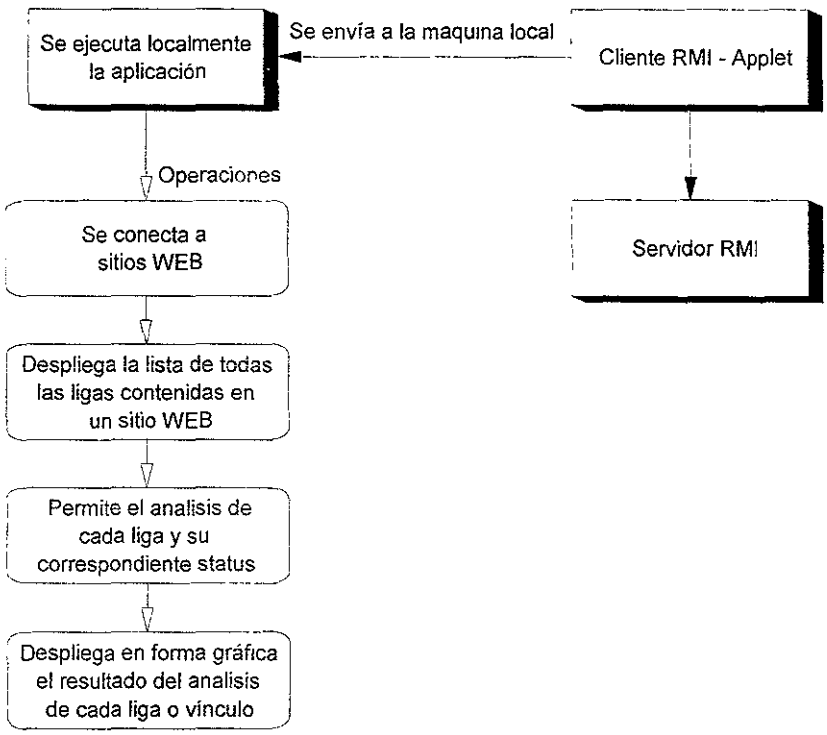


Figura 7.6 Esquema de funcionamiento del cliente RMI – Applet.

Descripción de las pantallas de la interfaz gráfica del cliente RMI – Applet:

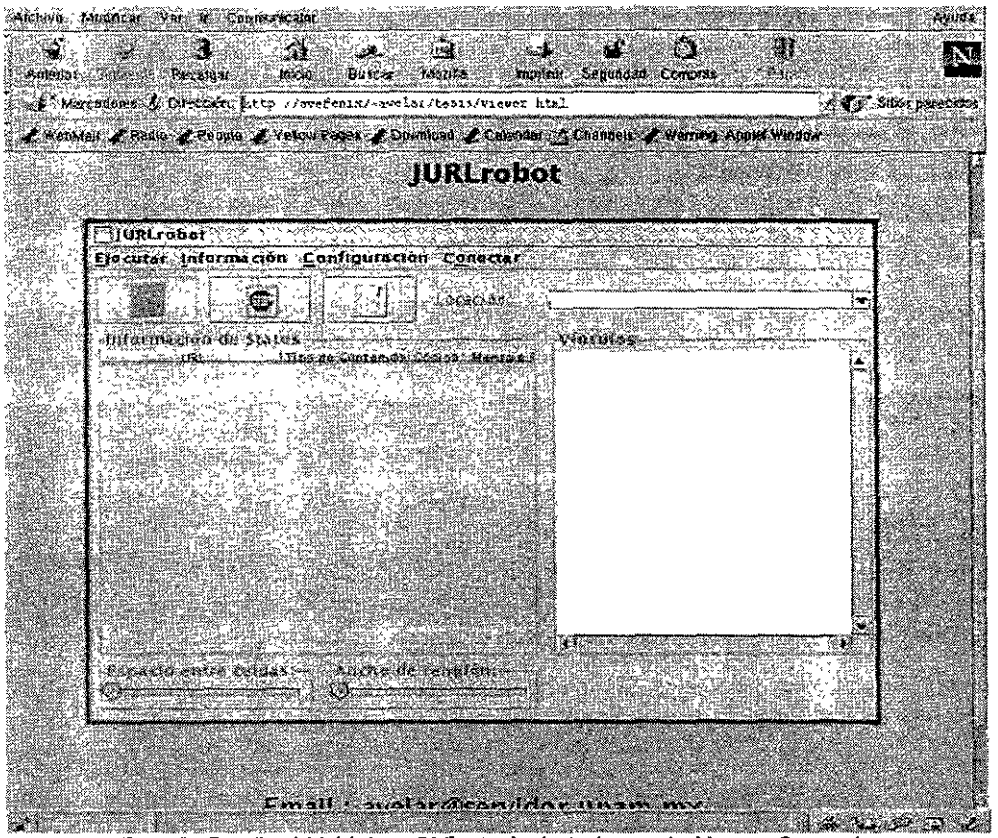


Figura 7 7 Pantalla inicial del cliente RMI – Applet desde el navegador Netscape Communicator

Una vez cargada la aplicación desde el sitio Web donde reside el sistema, se desplegará la pantalla inicial del cliente RMI – Applet. La figura 7.7 muestra la pantalla de inicio que aparecerá dentro del navegador de Web. Como se menciono anteriormente, si aún no está instalado el Plug–in Java 1.2 en el sistema, la primera vez que se conecta un usuario se hará una petición de instalación. Si el usuario decide no instalar el Plug–in, la aplicación RMI – Applet no se ejecutará. Las siguientes ocasiones en que el usuario se conecte desde su maquina al sistema, la aplicación RMI – Applet se ejecutará de manera normal, debido a que el Plug–in Java ya se encontrará instalado en su máquina local.

Para que el usuario pueda conectarse al sistema necesita irse al menú 'Conectar' y escoger la opción de 'Login', entonces aparecerá la pantalla para ingresar al sistema. Ahí el usuario debe teclear un login(puede ser el que guste el usuario) y el passwd(solo hay uno para todos, 'bugaboo').

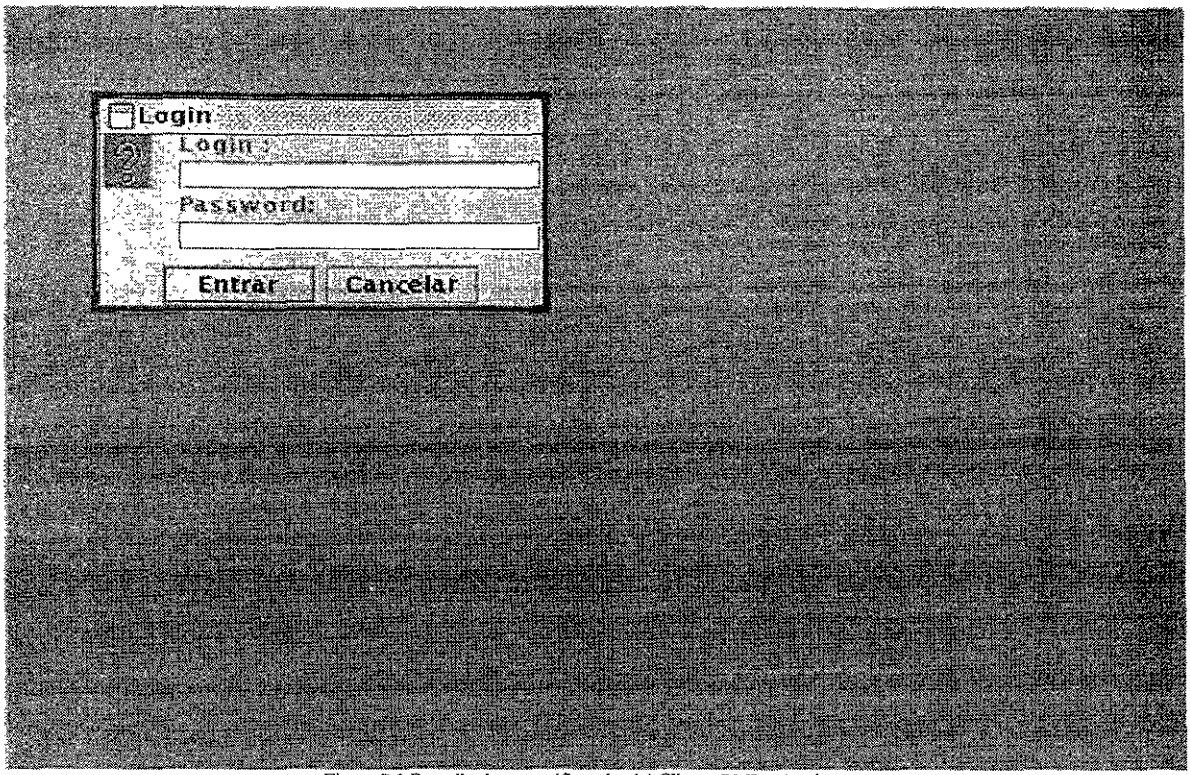


Figura 7.8 Pantalla de autentificación del Cliente RMI – Applet.

- *Permitir la conexión a sitios Web para realizar el análisis de sus vínculos.*

El cliente RMI – Applet desde su interfaz gráfica permite la navegación (conexión) a sitios Web remotos con sólo teclear una dirección URL en el campo 'Locación' de la aplicación. Este campo permite teclear un URL que tenga la estructura básica siguiente:

http://nombre_del_url/ruta

Si no se establece la cadena 'http' al principio el programa no sabrá interpretar el tipo de protocolo usado y por tanto enviará un mensaje de error de protocolo al usuario.

Otra forma de conexión a sitios Web será dando clic con el ratón(mouse) a la lista de URLs que se despliega del lado derecho de la pantalla de la aplicación del cliente RMI – Applet. Esto sólo si ya existe una lista de URLs desplegados ahí.

- *Despliega la lista de todas las ligas contenidas en un sitio Web*

La lista de URLs que se encuentra en el cuadro del lado derecho del cliente RMI – Applet, se despliega una vez que el usuario se conecto a un sitio Web por medio del campo 'Locación', si el sitio Web tecleado en ese campo era válido, se conectará al servidor RMI y éste devolverá una lista de URLs si se trataba de un documento HTML, en caso contrario, sólo devolverá el status del URL. La lista de URLs se desplegarán en el campo llamado 'Vínculos' en la aplicación cliente.

Así una vez que las ligas estén visibles, el usuario puede por medio del ratón(mouse) conectarse a otro sitio Web con solo darle clic a una liga de la lista. Una vez que el usuario da un clic, el cliente RMI – Applet se conecta a ese nuevo URL y se vuelve a desplegar la lista pero ahora con la nueva lista de URLs del nuevo sitio Web al cual se conecto.

Este proceso se puede realizar sucesivamente tantas veces como el usuario desee, pudiendo navegar entre ligas con sólo dar un clic con el ratón(mouse) y internarse en la telaraña de ligas que contiene un sitio Web.

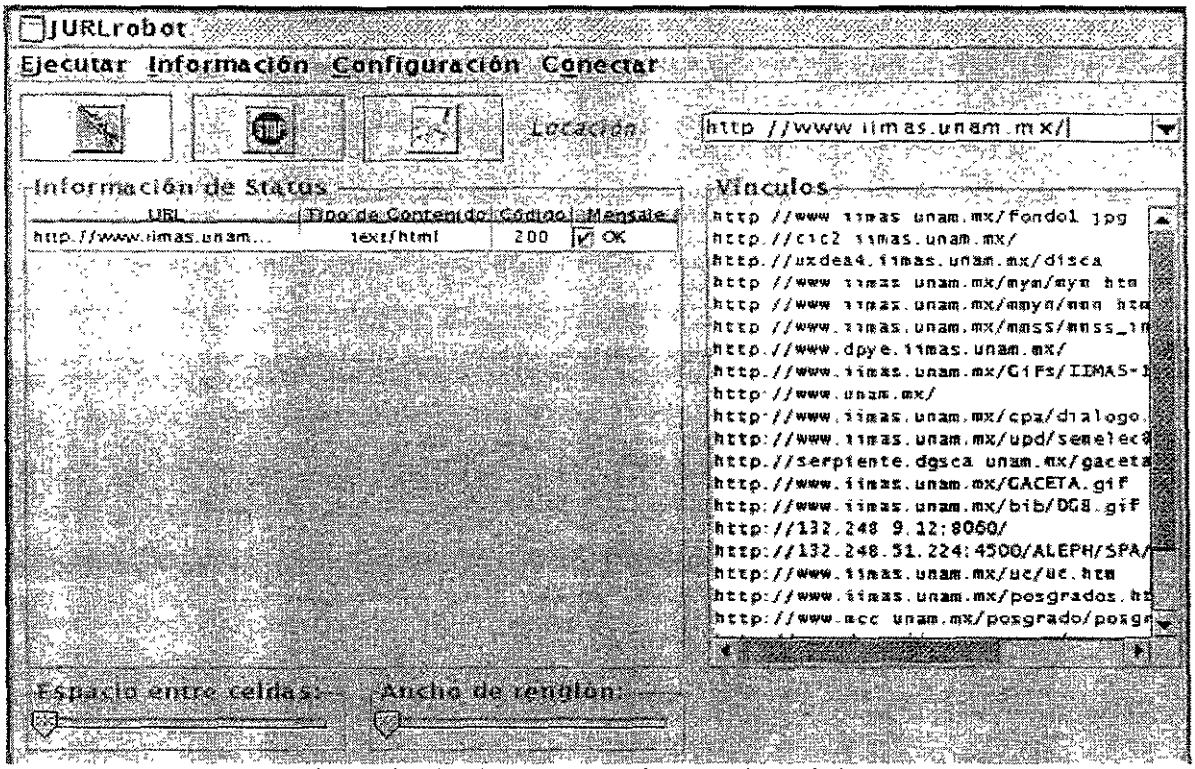


Figura 7 9 Permite la conexión a sitios Web remotos y despliega la lista de todas las ligas contenidas en el sitio

- *Permite el análisis de cada liga y su correspondiente status*

Una vez que la lista de URLs se encuentra ordenada y desplegada en el campo 'Vínculos' de la aplicación cliente; se encuentra preparada para enviarla al servidor RMI para que se verifiquen los URLs que están contenidos en ella. Con la opción 'Verificar' del menú 'Ejecutar' o con el botón de verificar status del cliente RMI- Applet(1er botón de izquierda a derecha dentro de la aplicación), se envía la lista de URLs al servidor RMI para que verifique el status de cada vínculo y despliegue el resultado en la pantalla. El status de cada vínculo es desplegado en el campo 'Información de Status' que se encuentra en el lado izquierdo de la ventana de la aplicación.

La información que nos muestra este campo es:

URL: La dirección URL real del vínculo ya verificado.

Tipo de Contenido: El tipo de contenido que tiene el recurso solicitado. El tipo de contenido puede ser entre otros: text/html, image/jpg, image/gif, etc.

Código: Es el código numérico de status que el servidor HTTP maneja. Este código es un estandar universal para todos los recursos solicitados a través del protocolo HTTP.

Mensaje: Es el mensaje de status generado a partir del código que regreso el servidor HTTP. Éste mensaje dice exactamente y en palabras cuál es el status del vínculo.

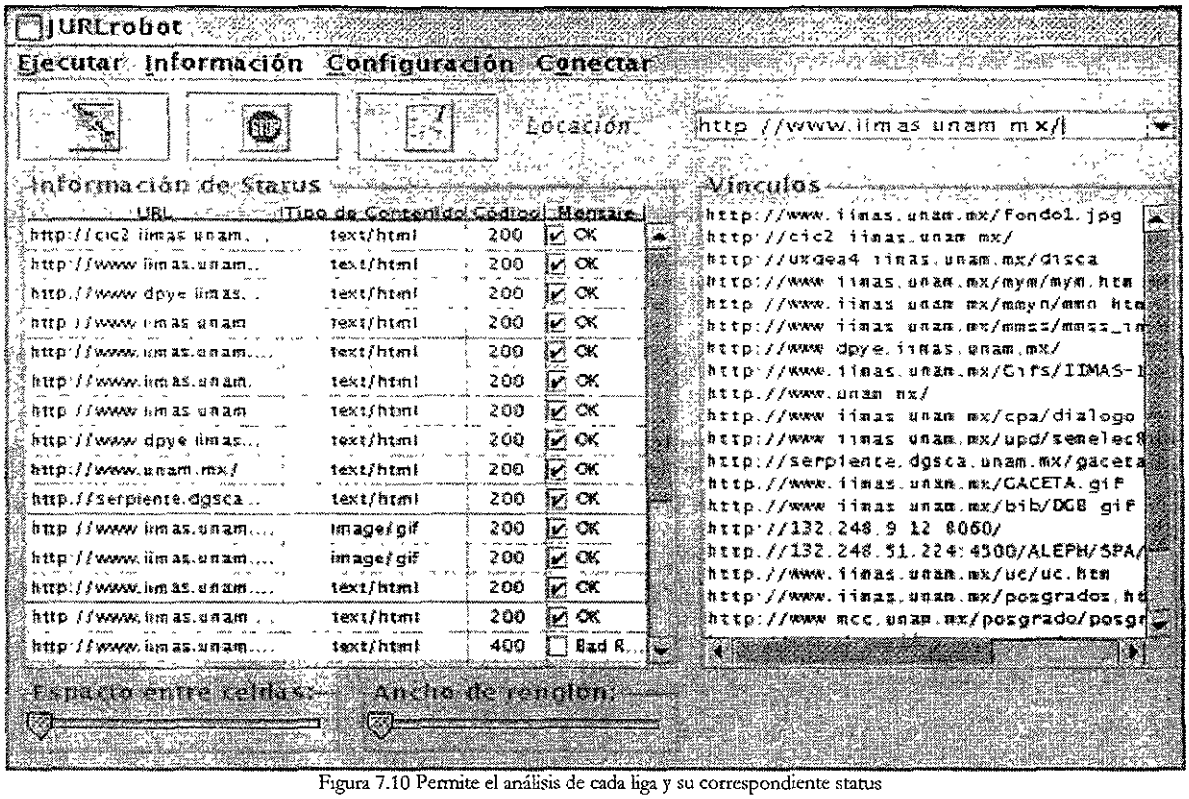


Figura 7.10 Permite el análisis de cada liga y su correspondiente status

Otras opciones que tiene la aplicación son:

- Permite configurar el tiempo para verificación de status de un URL

El tiempo de verificación es el tiempo que se dará para conexiones de verificación. Es tiempo que el sistema utiliza para el proceso de verificación de todos los vínculos. Cuando termina el tiempo asignado a esta variable el servidor RMI termina todos los procesos independientes de verificación individual de vínculos. Regresa todos los vínculos que hayan alcanzado a ser verificados antes de que terminará el tiempo de verificación.

- Permite configurar el tiempo para conexión a un URL solicitado

El tiempo de conexión es el tiempo que se dará para que una sesión con un servidor HTTP permanezca abierta. Es tiempo que el sistema asigna para una sesión con un servidor HTTP cuando un cliente RMI – Applet solicita la conexión hacia algún sitio Web. Una vez que el cliente hizo la petición al servidor RMI, éste intenta la conexión al URL solicitado, una vez realizada la solicitud al servidor HTTP correspondiente, se dispondrá solo del tiempo asignado a esta variable para realizar toda la operación de conexión y parseo del documento.

- Permite seleccionar el tipo de vínculos que serán verificados.

Por default el sistema establece que serán verificados tanto vínculos internos como externos. Pero se puede seleccionar que sólo sean los vínculos internos, o que también, que sólo sean los vínculos externos. Esto se consigue con las opciones de 'Vínculos locales', 'Vínculos remotos', y 'Vínculos ambos' que se encuentran dentro del menú 'Información'.

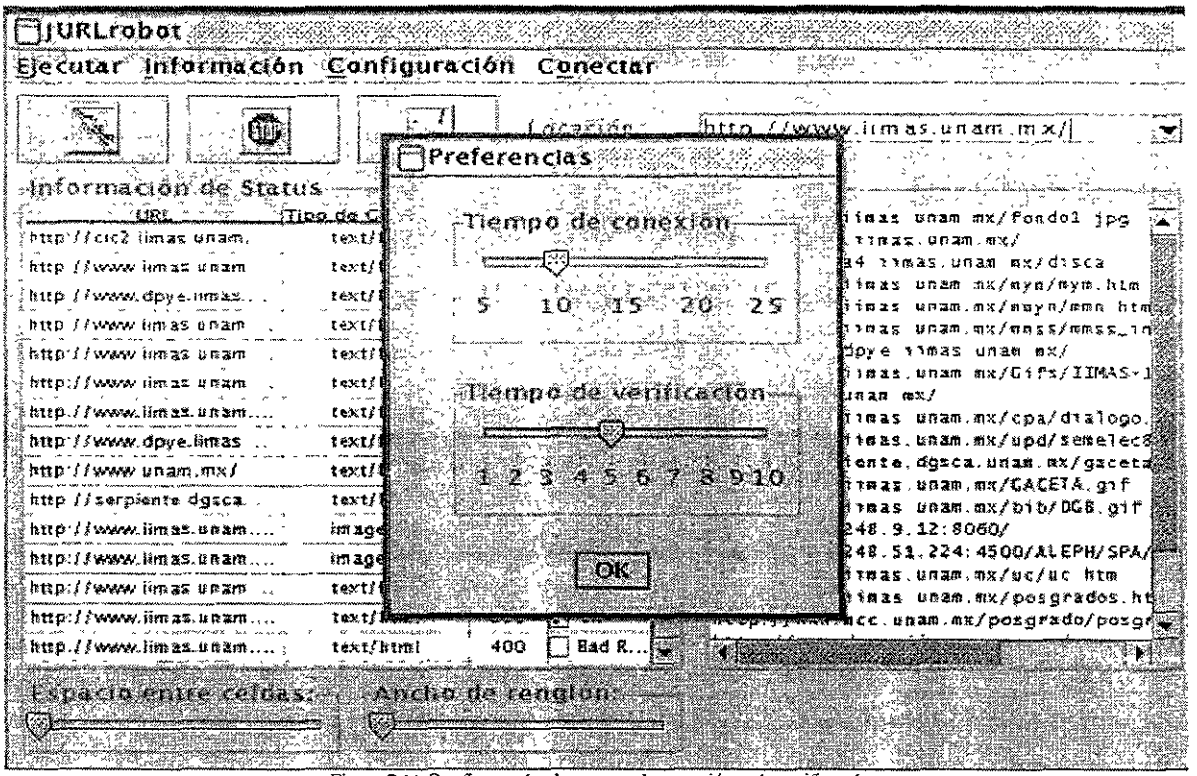


Figura 7.11 Configuración de tiempo de conexión y de verificación

- Da un reporte completo del status de un sitio Web.

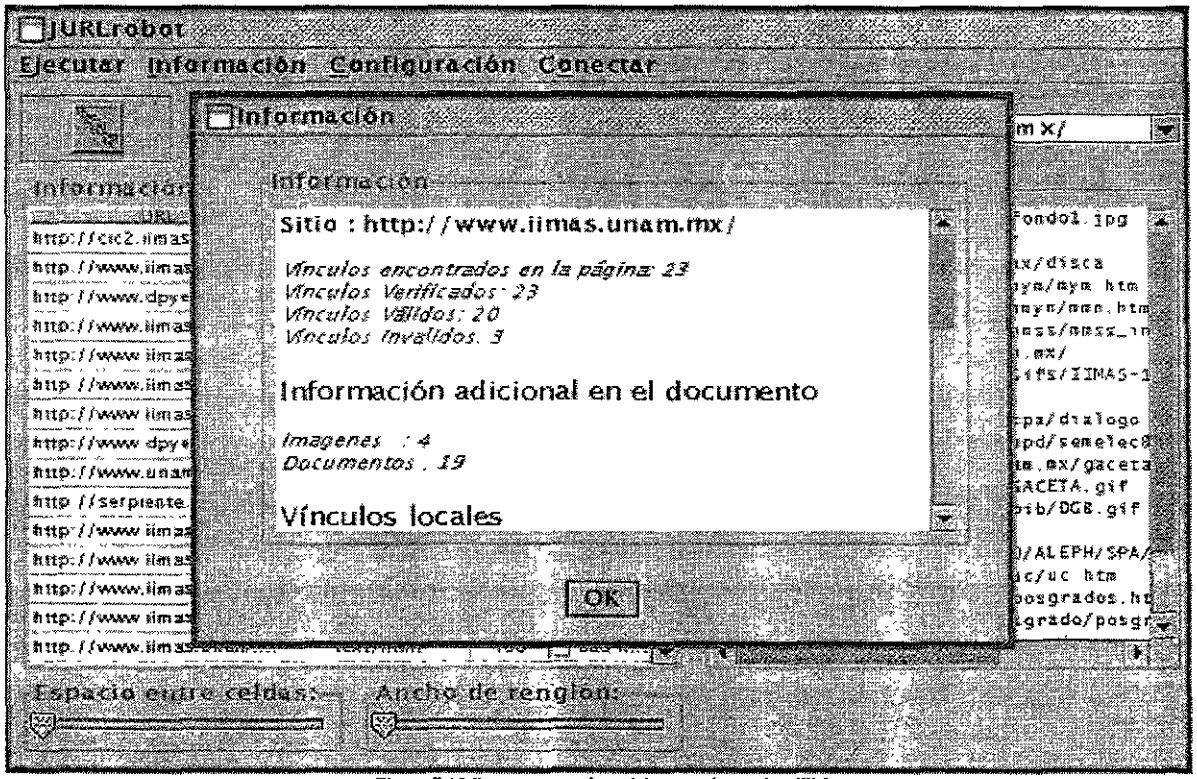


Figura 7.12 Reporte completo del status de un sitio Web

Una vez que se verificaron los vínculos de un URL y se desplegó la información en la tabla, se genera además, un reporte completo con información relevante del sitio Web analizado.

Este reporte nos dice:

- Sitio Web visitado.
- Número de ligas contenidas en el sitio.
- Vínculos verificados
- Vínculos válidos
- Vínculos inválidos
- Número de archivos.
- Número de imágenes.
- Ligas internas al URL solicitado
- Ligas externas al URL solicitado

- Permite desconectarse del servidor RMI en cualquier momento y volver a conectarse sin perder la información de los sitios visitados.

Cada vez que el usuario lo desee puede conectarse con el servidor RMI para iniciar una sesión de verificación. Si al cliente RMI – Applet se le termina el tiempo de sesión, puede en cualquier momento volver a conectarse con el servidor RMI para una nueva sesión. Todos los datos acumulados que tenga el cliente se conservarán. Las opciones 'Login' y '"Logoff' del menú 'Conectar' son las encargadas de abrir o cerrar una sesión con el servidor RMI.

- Permite borrar el contenido del campo 'Vínculos' y el campo 'Información de Status'

Con el botón de borrar (3er botón de izquierda a derecha dentro de la aplicación) o con la opción 'Limpiar' del menú 'Ejecutar' el usuario puede borrar el contenido de la lista de URLs que está en el campo 'Vínculos', y también, el contenido de la tabla de información de los vínculos que se encuentra en el campo 'Información de Status'. Al darle clic al botón o a la opción del menú se borra inmediatamente el contenido de ambos campos.

- Permite detener una conexión o el proceso de verificación de vínculos de algún sitio Web.

Con el botón de detener (2o botón de izquierda a derecha dentro de la aplicación) o con la opción de 'Detener' del menú 'Ejecutar' el usuario puede detener el proceso de conexión a algún sitio Web que haya solicitado. También puede detener el proceso de verificación de vínculos de un sitio Web que haya iniciado. Ambas solicitudes son enviadas directamente al servidor RMI para detener esos procesos en el servidor RMI.

Ya se describió como funciona detalladamente la aplicación distribuida, solo nos faltaría describir el esquema general de clases de Java que son necesarias para que le sistema funcione.

El listado completo del paquete de Java utilizado para este proyecto se muestra a continuación, la salida la se obtuvo directamente del comando 'ls –lR' de Unix:

```
-rwxr-xr-x   1 avelar   avelar     85453 ago 11 12:29 JURLrobot.jar
-rwxr-xr-x   1 avelar   avelar      1892 ago 11 12:28  Makefile
-rwxr-xr-x   1 avelar   avelar       330 ene  3  2000  java.policy
drwxr-xr-x   9 avelar   avelar      4096 jul 17 01:53  jurlrobot
-rwxr-xr-x   1 avelar   avelar      2282 jul 13 13:53  viewer.html
```

```
/jurlrobot:
drwxr-xr-x   2 avelar  avelar     4096 ago 11 12:31 client
drwxr-xr-x   2 avelar  avelar     4096 dic 21  1999 doc
drwxr-xr-x   2 avelar  avelar     4096 ago 11 12 31 gui
drwxr-xr-x   3 avelar  avelar     4096 jul 17 01 53 img2
drwxr-xr-x   2 avelar  avelar     4096 ago 11 12:31 parserh
drwxr-xr-x   2 avelar  avelar     4096 ago 11 12:31 server
drwxr-xr-x   2 avelar  avelar     4096 ago 11 12:31 www
./jurlrobot/client
-rwxr-xr-x   1 avelar  avelar     1093 jul  1 00.25 CheckCellRenderer.java
-rwxr-xr-x   1 avelar  avelar      340 ago 11 11.05 ColumnData.java
-rwxr-xr-x   1 avelar  avelar     5198 ago 11 11:04 JFrameMain.java
-rwxr-xr-x   1 avelar  avelar     3901 ago 11 11:15 JTableModel.java
-rwxr-xr-x   1 avelar  avelar    18929 ago 11 11:36 JURLControl.java
-rwxr-xr-x   1 avelar  avelar     5145 ago 11 11·40 JURLPanel.java
-rwxr-xr-x   1 avelar  avelar     2398 jul 26 02:17 JURLrobot.java
-rwxr-xr-x   1 avelar  avelar      555 jul  1 00:31 MemComboBox.java
-rwxr-xr-x   1 avelar  avelar     8309 jul 26 02:38 PasswdDialog.java
-rwxr-xr-x   1 avelar  avelar     4709 ago 11 11:22 SetupDialog.java
-rwxr-xr-x   1 avelar  avelar     8666 ago 11 11:38 TextView.java
./jurlrobot/gui
-rwxr-xr-x   1 avelar  avelar    32528 jul  1 01:29 HtmlLayout.java
-rwxr-xr-x   1 avelar  avelar     4802 jul  1 01:33 HtmlLayoutMapping.java
-rwxr-xr-x   1 avelar  avelar     4760 jul  1 01:33 Layout.java
-rwxr-xr-x   1 avelar  avelar      577 jul  1 01:34 LayoutException.java
./jurlrobot/img2:
-rwxr-xr-x   1 avelar  avelar     1152 feb  4  2000 clearA.gif
-rwxr-xr-x   1 avelar  avelar      408 ene 28  2000 door_exit.gif
-rwxr-xr-x   1 avelar  avelar      344 ene 28  2000 door_open.gif
-rwxr-xr-x   1 avelar  avelar     1029 feb  4  2000 linkA.gif
-rwxr-xr-x   1 avelar  avelar      300 ene 28  2000 runner.gif
-rwxr-xr-x   1 avelar  avelar      411 ene 28  2000 search_doc.gif
-rwxr-xr-x   1 avelar  avelar      305 ene 28  2000 spider.gif
-rwxr-xr-x   1 avelar  avelar     1100 feb  4  2000 stopA.gif
./jurlrobot/parserh:
-rwxr-xr-x   1 avelar  avelar      788 jun 30 23:43 HtmlException.java
-rwxr-xr-x   1 avelar  avelar    18427 jun 30 23:43 HtmlStreamTokenizer.java
-rwxr-xr-x   1 avelar  avelar    16055 jun 30 23:44 HtmlTag.java
-rwxr-xr-x   1 avelar  avelar      798 jun 30 23:44 Links.java
-rwxr-xr-x   1 avelar  avelar     3746 jun 30 23:46 ParserLinks.java
-rwxr-xr-x   1 avelar  avelar     5993 jul  1 01:25 createLinks.java
./jurlrobot/server:
-rwxr-xr-x   1 avelar  avelar     3456 jul  1 00:13 HttpVerify.java
-rwxr-xr-x   1 avelar  avelar      609 jul  1 02:17 JURLServer.java
-rwxr-xr-x   1 avelar  avelar     5327 jul  2 04:25 JURLServerInit.java
-rwxr-xr-x   1 avelar  avelar     3045 jul  1 01:12 StartVerify.java
-rwxr-xr-x   1 avelar  avelar     4634 jul  1 01:17 UserTimeout.java
-rwxr-xr-x   1 avelar  avelar      983 jul  1 00:23 Verify.java
-rwxr-xr-x   1 avelar  avelar      983 jul  1 01:28 VerifyLink.java
./jurlrobot/www:
-rwxr-xr-x   1 avelar  avelar     1059 jul  1 00:00 HttpClient.java
-rwxr-xr-x   1 avelar  avelar     5607 jul  1 00:03 HttpResponse.java
-rwxr-xr-x   1 avelar  avelar     4619 jul  1 00:04 SocketTimeout.java
```

El código fuente de este paquete irá en un CD aparte que se anexará al presente trabajo de tesis.

Estructura general de clases del paquete jurlrobot.

Clases del Servidor RMI

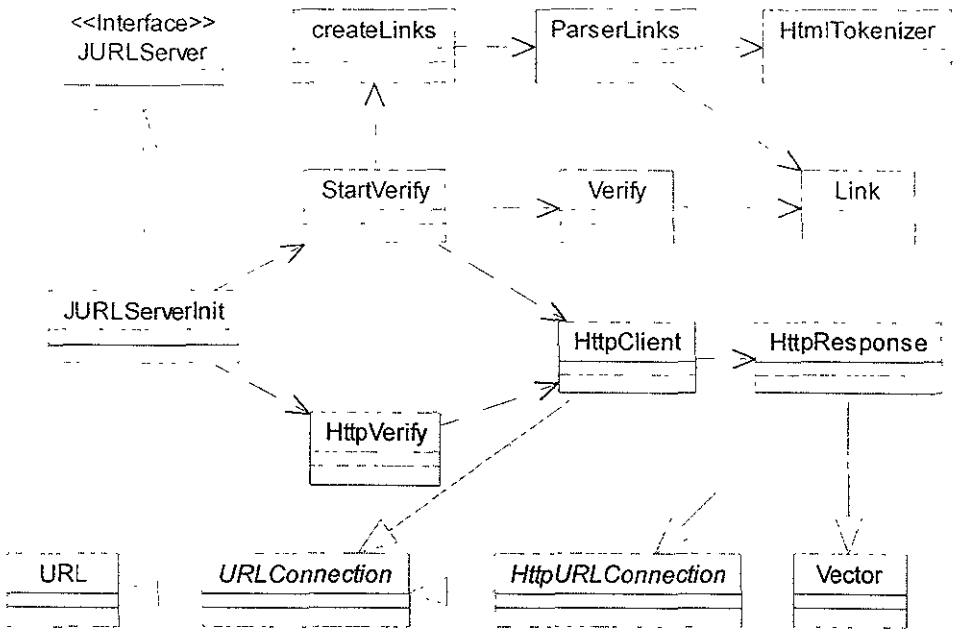


Figura 7.13 Diagrama de clases del servidor RMI

Clases del Cliente RMI - Applet

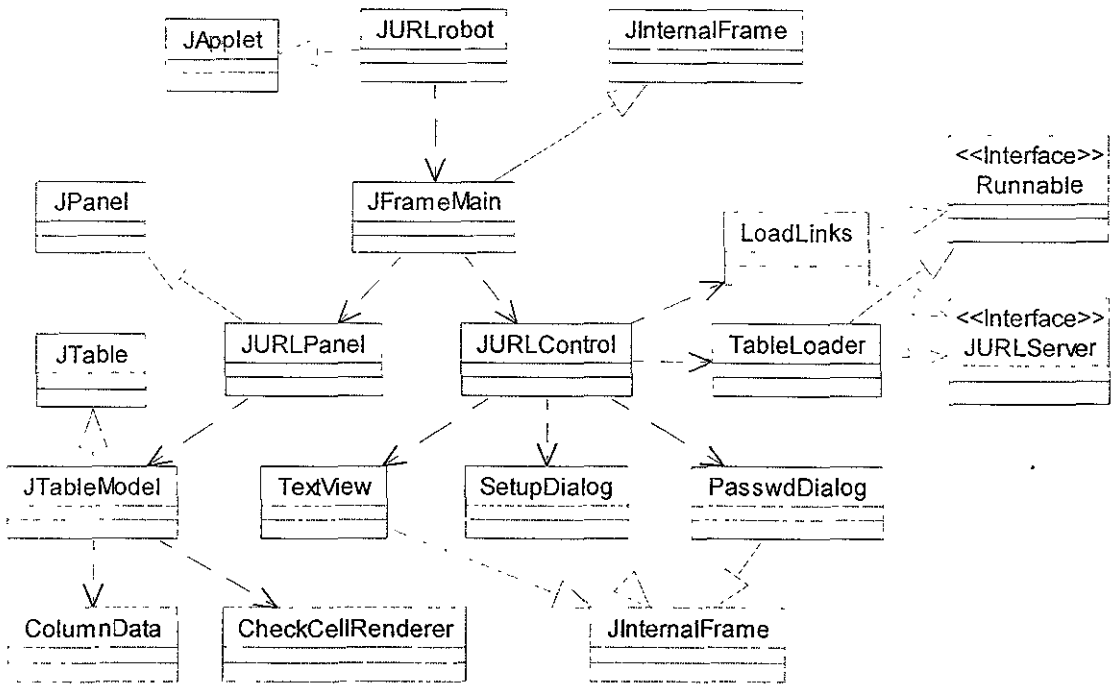


Figura 7 14 Diagrama de clases del cliente RMI - Applet

Mejoras a futuro.

1.- Establecer un sistema de autentificación de usuarios más sofisticado.

El sistema actual se encuentra configurado con un básico sistema de autentificación, por lo que, puede no ser muy seguro. Además, no mantiene un control de usuarios muy elaborado. Sería conveniente tener una base de datos para llenar el control de los usuarios que entran y salen del sistema. Establecer nombres de usuarios permanentes y un password asignado de manera individual a cada usuario. También se puede cifrar está información para que no haya peligro de intromisiones no autorizadas al sistema, debido a robo de claves de usuario. Java, en sus últimas versiones, maneja códigos de encriptación de datos más sofisticados (RSA, DES, etc.) que pueden servir para el cifrado del password

2.- Mejorar la interfaz gráfica de usuario

Agregar más opciones de configuración del sistema. Generar reportes más sofisticados y atractivos visualmente para el usuario. La interfaz gráfica puede ser mejorable agregando más opciones de configuración y ayuda.

3.- Permitir verificar vínculos que manejen otros protocolos

El sistema sólo permite verificar vínculos que manejen el protocolo HTTP. Como una mejora posterior al sistema se pueden agregar otros protocolos como FTP, MAILTO. Claro que todos los protocolos que se quieran agregar deben estar antes soportados por el navegador de Web. Es decir, si el navegador los soporta el sistema puede soportarlos, ya que nuestro sistema está basado totalmente en la iteración entre los navegadores de Web y los servidores HTTP. Además, debe manejar código HTML que utilice vínculos a URLs, de lo contrario no podrá funcionar la parte de parseo de vínculos, ya que el tipo de contenido que la aplicación utiliza y soporta es 'text/html'.

4.- Manejo de más tipos de vínculos

Por default, los vínculos de contenidos en los tags HTML de "A HREF", "IMG SRC", "AREA SRC", "APPLET CODE", "FORM ACTION" y "JAVASCRIPT SRC" son lo que el sistema maneja. Sin embargo, existen otros tipos de vínculos como "BODY BACKGROUND", "EMBED SRC", "TD BACKGROUND", y algunos otros que no fueron incluidos, pero se pueden incluir con sólo editar el archivo HtmlTag.java, HtmlStreamTokenizer.java y ParserLinks java.

5.- Ampliar el sistema para que acepte más usuarios y conexiones simultaneas

El sistema actualmente acepta 20 usuarios conectados simultáneamente. Realizando un mejor esquema de control de peticiones y de autentificación se pueden aceptar mucho más usuarios. Además, estos usuarios podrán tener un mayor tiempo permitido de sesión ya que actualmente el sistema sólo permite 10 minutos por sesión.

6.- Tener mucho mayor control de las fallas

El sistema maneja y controla bastantes casos especiales de errores. Sin embargo, en algunos casos se puede mejorar la solución utilizada. Por ejemplo, cuando hay perdida de conexión a la red, el sistema sólo envía un mensaje de que el servidor RMI no responde. Se puede enviar aquí un mensaje más intuitivo que aparte permita accesar un sistema de ayuda para el usuario.

Otro caso es cuando el servidor RMI se encuentra apagado, el sistema sólo envía un mensaje de que el servidor está abajo, pudiendo tener un servidor RMI alterno al principal para estos casos. Cuando ocurre sobrecarga de trabajo se puede también utilizar la solución de un servidor alterno que comparta las peticiones de los clientes. Esto sería muy eficaz si aumenta el número de usuarios conectados simultáneamente y el tiempo que permanecerán en sesión.

7.- Utilizar compresión de datos para conexiones seguras

Para tener una comunicación de datos entre los clientes RMI – Applet y los servidores RMI se recomienda usar SSL(Secure Socket Layer, Conexiones con sockets seguras) junto con RMI para tener una manejo de la seguridad de los datos. Esto puede ser implementado de una manera sencilla en las nuevas versiones del JDK. Esto es recomendable para cuando se diseñen aplicaciones que requieren un máximo de seguridad en los datos.

Conclusiones

En el presente trabajo se desarrolló e implementó una aplicación distribuida que verifica la validez de los hipervínculos en un sitio WWW. Con los resultados obtenidos se llegó a las siguientes conclusiones:

- La aplicación genera resultados que son confiables y comprensibles para el usuario.

- Muestra buen desempeño para sitios Web que poseen un gran número de vínculos.

- El sistema puede probar la validez de todos los vínculos internos y externos de un URL en algún sitio WWW. Dando la opción de seleccionar el tipo de vínculos(internos, externos o ambos) a verificar.

- Genera reportes con la información de los vínculos ya verificados. Estos reportes dicen que vínculos son válidos y cuáles no lo son. Además, dicen las características de cada vínculo, el tipo de contenido que maneja, y su status en el servidor HTTP. Genera información suficiente para evaluar el estado general de algún sitio Web.

- Para la mejor visualización de los reportes generados por el servidor RMI, se implementó un cliente que tiene una interfaz gráfica amigable para el usuario. Esta interfaz nos permite navegar entre sitios Web, seleccionando y verificando los vínculos con sólo teclear un URL de WWW, o dando clic a algún vínculo ya seleccionado y desplegado en la interfaz gráfica. La interfaz además, permite configurar opciones que dan un análisis más detallado de un sitio Web. El programa aprovecha al máximo las ventajas que ofrece el paquete Swing de Java, creando una interfaz intuitiva, fácil de usar, versátil y sobre todo funcional.

- La aplicación puede ser ejecutada en cualquier momento y en cualquier lugar con sólo accesar una dirección fija de Internet. La aplicación no necesita ser instalada en cada máquina local, el programa se transmite a través de la red Internet y se ejecuta localmente. El sistema hace uso del modelo Cliente/Servidor como forma de funcionamiento entre los clientes RMI – Applet y el servidor RMI. Además, usa el esquema de Invocación de Métodos Remotos(RMI) de Java, como su método de comunicación entre los componentes del sistema. Debido a este esquema de comunicación y a su modelo de funcionamiento la aplicación tiene un comportamiento distribuido.

- Ya que el sistema aprovecha el protocolo HTTP para transportarse y a los navegadores de Web para desplegarse y ejecutarse, no necesita requerimientos de hardware adicionales. La aplicación funciona tanto en máquinas Unix, como sobre arquitecturas PC y Mac. Teniendo las siguientes características:

  - En los sistemas operativos de Windows, MacOs, Linux, Solaris y Irix el cliente RMI – Applet se ejecuta exitosamente en todos ellos, solo necesitando instalar el Java Plug–in 1.2. Para que se pueda ejecutar sobre otras sistemas debe existir ese Plug–in.
  - El servidor RMI se ejecuta exitosamente sobre los sistemas Linux y Solaris. Se puede instalar sobre otros sistemas operativos si existe alguna versión del Java Development Kit (JDK) para esos sistemas.

  Por todas estas características la aplicación tiene un funcionamiento multiplataformas. Ya que se puede utilizar en un gran número de arquitecturas y sistemas operativos.

- La aplicación nos permite comprobar que se pueden construir Java Applets que funcionen como verdaderas aplicaciones corporativas y empresariales, ya que el esquema utilizado para la construcción de la aplicación puede ser utilizado para múltiples usos. Cualquier aplicación que necesite ser distribuida por la red, funcione bajo el esquema de Cliente/Servidor, y utilice una interfaz gráfica del lado del cliente, puede ser construida de esta manera.

- También, permite comprobar que las llamadas a procedimientos remotos(Remote Method Interface - RMI) de Java, pueden ayudar a crear aplicaciones distribuidas que son fáciles de construir y que son totalmente propietarias a Java, ya que se construyen enteramente con este lenguaje de programación, lo que las hace más robustas y controlables por estar programadas en Java puro.

Indice de figuras

7 Aplicación distribuida de diagnostico de vínculos de un URL en un servidor de WWW.

Apéndice A

Manual de uso de la aplicación distribuida de diagnóstico "JURLrobot"

- ¿Qué navegadores lo soportan?

- Netscape Navigator 4.x para cualquier plataforma o sistema operativo.
- Internet Explorer 4.x y 5.x para cualquier plataforma o sistema operativo.

- ¿Qué se necesita?

Los requerimientos mínimos para poder ejecutar y visualizar adecuadamente JURLrobot son:

Para el cliente:

- Conexión a Internet.

Para conectarse al sitio remoto donde reside la aplicación se necesita tener una conexión a Internet. Para conexión por modem(a una velocidad promedio de conexión de entre 30 - 45 Kbytes) el sistema tardará entre 2 - 5 minutos en cargarse. Con una conexión directa conectada a una red local LAN, el tiempo será de unos 10 - 25 segundos.

- Navegador de Web compatible. Se recomienda Netscape Navigator 4.x o Internet Explorer 4.x y 5.x.
- Instalación del Plug-in Java 1.2.

En este caso no hay que preocuparse de donde bajarlo o conseguirlo. La primera vez que se conecta al sitio remoto pide bajar e instalar el Plug-in. En Netscape Navigator despliega una nueva ventana del navegador conectándose al lugar donde debemos bajar el programa. Lo bajamos y luego lo instalamos. En Internet Explorer el proceso mucho más fácil, ya que abre una ventana de instalación automática que solo requiere seguir las instrucciones para poder instalar el Plug- In.

- Pc o Mac un mínimo de 64 MBytes en memoria RAM.

Se recomiendan 128 MBytes. En otras plataformas dependerá de las características propias del Hardware, sin embargo, se recomienda tener el mayor numero de memoria de sistema disponible para ejecutar programas.

- Pc o Mac un mínimo de 250 Mhz.

Se recomiendan 450 MHz o más. En otras plataformas, al igual que la memoria, se recomienda que sea un sistema con un procesador rápido.

Para el servidor.

- Máquina servidor con sistema operativo Unix con conexión permanente a Internet.

Máquina que tenga una dirección IP fija y que este conectada a Internet de forma permanente. Los sistemas Unix recomendados son: Linux Red Hat 6.x, Linux SuSe 6.x, Solaris 2.5.x, Irix 6.x. En todos ellos fue probado y se ejecuto satisfactoriamente.

- Servidor HTTP ejecutándose sobre la máquina Unix

Un servidor de Web o HTTP que esté ejecutándose sobre la máquina Unix. Este servidor estará en espera de peticiones a la pagina de inicio del sistema JURLrobot. Se recomienda Apache versiones 1.3.x.

- Java Development Kit(JDK) 1.1.8 instalado en el servidor UNIX.

Se necesita para compilar la aplicación. Se puede bajar de http://www.javasoft.com

- Java Foundation Classes(JFC) 1.1 instaladas en el servidor UNIX. Se necesitan para el Swing. Se puede bajar de http://www.javasoft.com
- Establecer la variable de ambiente CLASSPATH. Esta debe tener el lugar donde se encuentran las clases de JDK y JFC. Ejemplo en csh y bash:

% export CLASSPATH=/dir_java_clases/classes.zip:/dir_jfc_clases/swingall.jar

- Instalación y configuración

Se necesita tener una cuenta de usuario en el servidor Unix donde se instalara la aplicación. Está máquina Unix será la misma donde debe estar ejecutándose el servidor HTTP. Además, se necesita que el servidor HTTP este configurado para otorgar permiso a usuarios normales de publicar documentos HTML. Si no sabe como hacerlo o tiene dudas, debe preguntar al administrador del sistema Unix la configuración correcta para poder publicar documentos HTML. Se descarga el paquete jurlrobot.tar.gz del CD o del sitio Web (http://www.mcc.unam.mx/~avelar/jurlrobot.tar.gz), en la máquina local donde residirá el sistema. Se debe colocar en un subdirectorio que sea accesible tanto para el servidor de HTTP como para los clientes a dicho servidor.

Se descomprime:
% gzip –d jurlrobot.tar.gz

Se destararea:
% tar –xvf jurlrobot.tar

Esto genera el directorio "jurlrobot". Nos metemos en él.
% cd jurlrobot

Compilamos:
% make

Listo, el servidor debe haberse instalado correctamente. Si genera algún mensaje de error al momento de compilar, se debe verificar que el JDK y las JFC estén instaladas correctamente. También se debe ver si la variable de ambiente CLASSPATH esta correctamente definida.

- Inicio

Se carga el Cliente - Applet RMI desde el sitio remoto.

http://www.mcc.unam.mx/~avelar/jurlrobot/viewer.html

- Si no existe el Java Plug-in 1.2 se debe de instalar.
- Si ya está instalado se ejecuta la aplicación.

- Funcionamiento

Está aplicación llamada JURLrobot es un applet escudriñador de sitios Web. Permite conectarse a sitios Web y escudriñar sus vínculos para conocer el estado de los mismos. Este applet funciona como en software de diagnóstico que verifica todos los vínculos que posee un documento HTML que se encuentra publicado en algún servidor de Web.
Cada vínculo puede ser de caracteristicas distintas y hacer referencia a variados recursos en la red(imagenes, archivos de audio, archivos de sonido, documentos HTML, etc.), el programa puede hacer la distinción entre los distintos formatos y solo analizar los documentos con formato HTML, ya que estos son los que contienen y hacen referencia(por medio de vínculos o hipervínculos) a recursos que se encuentran distribuidos por la red.

- Interfaz Gráfica

Realizada en Java Swing. Es lo último en herramientas para la construcción de GUI´s de Java.

Descripción del Applet:

- **Botones**
  - Verificar

  Inicia el proceso de verificación de vínculos.
  - Detener

  Detiene el proceso de verificación.
  - Borrar

  Borra la tabla de información y la lista de vínculos de la interfaz gráfica y de la memoria del sistema.

- **Campo "Locación"**

Permite teclear un URL de algún sitio de WWW.

- **Campo "Tabla de información de vínculos"**

Despliega los resultados generados del proceso de verificación de vínculos.

- **Campo "Lista de vínculos"**

Despliega todos los vínculos de algún URL en un sitio WWW.

- **Menús**
  - Ejecutar

  Tiene las opciones de : Verificar, Detener, y Borrar. Estas son las equivalentes a los 3 botones de la interfaz.
  - Información

  Tiene las opciones de:
    - Información del sitio. Despliega información mas detallada de un sitio
    - Vínculos locales. Vínculos que ligan al URL y son locales.
    - Vínculos remotos. Vínculos que ligan al URL y son remotos.
    - Vínculos ambos. Vínculos que ligan al URL y son ambos tipos.
  - Configuración

  Permite configurar el tiempo de conexión y el tiempo de verificación.
  - Conectar

  Permite conectarse al servidor RMI para comenzar a navegar los sitios Web.
  Tiene las opciones de:
    - Login. Se conecta al servidor RMI
    - Logoff. Se desconecta del servidor RMI.

Bibliografía

Libros

- Capítulo 1

Redes de computadoras, internet e interredes
Comer, Douglas E.
México · Prentice Hall, 1997

Computer Networks 3
Tanenbaum, A.
Prentice-Hall, 1996

Internetworking with TCP/IP
Comer, Douglas E.
Englewood cliffs, New Jersey : Prentice Hall, 1991-9999

- Capítulo 2

Conéctate al mundo de internet
Krol, Ed
Mexico : McGraw-Hill, 1995

Finding it on the internet : The essential guide to archie, veronica, gopher, wais, www, and other search tools
Gilster, Paul
New york : J. Wiley, 1994

- Capítulo 3

Multimedia and hypertext : the internet and beyond
Nielsen, Jakob
Boston : Ap Professional, 1995

World Wide Web : paso a paso
Eager, William
México : Prentice Hall, 1995

- Capítulo 4

Programando con Java
Ritchey, Tim
España : Prentice Hall International, 1997

Java 1.2. Al descubierto
Jamie Jaworski
España: Prentice Hall International, 1999

- Capítulo 5

Internet agents : spiders, wanderers, brokers, and bots
Cheong, Fah-Chun
Indianapolis, Indiana : New Riders, 1996

Java Network Programming
Harold, Elliotte Rusty
Cambridge : O'Reilly, 1997

Java networking and AWT API superbible : the comprehensive reference to the Java programming language : everything you ever needed to know about Java applets and its networking and windowing
Nagaratnam, Nataraj
Corte Madera, California · Waite Group, 1996

- Capítulo 6

Distributed object-oriented data-systems design
Andleigh, Prabhat K.
Englewood cliffs, New jersey : Prentice Hall, 1992

Java 1.2. Al descubierto
Jamie Jaworski
España: Prentice Hall International, 1999

- Capítulo 7

Designing the user interface : Strategies for effective human-computer interaction
Shneiderman, Ben
Reading, Massachusetts : Addison-wesley, 1992

Applying UML and patterns : an introduction to object-oriented analysis and design
Larman, Craig Upper Saddle River, New Jersey : Prentice Hall PTR, 1997

Referencias en la red

Tipos de redes
http://www.geocities.com/Athens/9105/redes/red_ses_7.html

Redes
http://www.info-ab.uclm.es/sec-ab/plan/redessup.html

Redes LAN
http://194.179.52.102/redeslan

Protocolos de comunicación
http://www.angelfire.com/pa/LilianaP/

¿Sabías que son las redes de computadoras?
http://www.geocities.com/Athens/Olympus/7428/red1.html

Status Codes in HTTP
http://www.w3.org/Protocols/HTTP/HTRESP.html

Webmaestro – Crea tu página de Web
http://wmaestro.com/webmaestro/

¿Qué es WWW?
http://www.supaginaweb.com/queeswww.htm

Conceptos sobre redes
http://www.geocities.com/CapeCanaveral/5312/redes2.htm

Redes de área local. Apuntes
http://tuy.uasnet.mx/prof/cln/ccu/mario/REDES/Lan.html

Curso de redes de computadoras
http://lara.puc.udlap.mx/redes/indexred.htm

Curso WWW
http://ekeko.rcp.net.pe/rcp/tutorial/

Introducción a Internet
http://geminis.adi.uam.es/~pcobo/internet/manual/manual.htm

Arquitectura de redes de área local
http://www.disc.ua.es/asignaturas/rc/trabajos/lan3/rc.html

Familia de Protocolos TCP/IP
http://www.cybercursos.com.ar/tcp-ip.htm

HyperText Transfers Protocol – HTTP 1.0,1.1
http://www.w3.org/Protocols/HTTP/1.0/spec.html
http://www.w3.org/Protocols/HTTP/1.1/spec.html

El protocolo HTTP
http://cdec.unican.es/libro/HTTP.htm

Presente y futuro del Web
http://pages.hotbot.com/sports/factor4/futuroweb.html

Java Development Kit 1.1.x
http://java.sun.com/products/jdk/1.1/index.html

Tutorial de Java
http://maxwell.univalle.edu.co/javatutor/Intro/tabla.html

The Java Tutorial
http://java.sun.com/docs/books/tutorial/index.html

Remote Method Invocation
http://java.sun.com/docs/books/tutorial/rmi/index.html

Comparing IIOP, RMI, HTTP
http://www.npac.syr.edu/projects/cps714fall97/hw4/msen/

Java Foundation Classes
http://java.sun.com/products/jfc/

Distributed Software Systems Development
http://siesta.cs.wustl.edu/~schmidt/cs544/

The Swing Connection
http://java.sun.com/products/jfc/tsc/index.html

Pnuts User's Guide
http://javacenter.sun.co.jp/pnuts/doc/PnutsLayout.html

Ejemplos Java Applet
http://www2.adm.ula.ve/java/

HCI/JAVA Swing Programming
http://www.cm.cf.ac.uk/Dave/HCI/

Getting Started Using RMI
http://www.javasoft.com/products/jdk/1.2/docs/guide/rmi/getstart.doc.html

Java GUI Programming with Swing/JFC
http://www.swtech.com/java/swing/

HtmlStreamTokenizer
http://www.do.org/products/parser/

Sistemas Distribuidos
http://www.inf.utfsm.cl/~rmonge/sd/

Introducción a los Sistemas Distribuidos
http://habitantes.elsitio.com/avilavit/index2.html

Redes de Ordenadores – Procesamiento en paralelo
http://www.ctv.es/USERS/carles/PROYECTO/proyecto.html

Swing Examples
http://www2.gol.com/users/tame/swing/examples/SwingExamples.html

Swing Manual and Advanced Examples
http://manning.spindoczine.com/sbe/

Laboratorio de software. Cursos
http://gilito.lab.dit.upm.es/~labscom/

Java : Serializable
http://tinymac.ecst.csuchico.edu/~kcuncke/CSCIOOPChapters/serial.html

OSIRIS: Laboratorio en Linux para desarrollar Sistemas Distribuidos
http://osiris.uniandes.edu.co/osiris/general/osiris.html

Super Tips Java Swing. Astuces Java
http://www.eteks.com/ups/index.html