18
29

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

TRADUCTOR DE PROTOCOLO

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A N:

LILIA GUADALUPE BEJARANO DIAZ
ARTURO BURGOS BALBUENA
MANUEL CASTILLO CASTAÑON
JAZEEL R. ORTEGA GONZALEZ
IXCOATL PEREZ DE LA BARRERA

DIRECTOR DE TESIS:
M. EN I. JUAN CARLOS ROA BEIZA

MEXICO, D. F. OCTUBRE 1998

Agradecimientos.

Al Ing. Juan Carlos Roa Beiza.
Por su paciencia y sabiduría que amablemente compartió con nosotros.

A la Facultad de Ingeniería y sus profesores.
Por abrirnos las puertas y ofrecernos la posibilidad de forjarnos un futuro.

A la Universidad Nacional Autónoma de México.
Por otorgarnos las herramientas necesarias para mejorar nuestro desarrollo y desempeñarnos como hombres y mujeres capaces de afrontar cualquier reto.

A todos mil gracias...
Lilia, Manuel, Arturo, Ixcoatl y Jazeel.

A Dios y a mis Padres

Ixcoatl

INTRODUCCIÓN

Sobre este documento.

Este documento describe los pasos a seguir para la construcción de un sistema *traductor de protocolo*, término acuñado en el sector financiero que se explicará a detalle más adelante, partiendo desde la teoría básica de redes y el uso del lenguage C++ hasta culminar con el sistema terminado y plenamente funcional.

Este libro se divide en tres capítulos y diez apéndices. El primer capítulo titulado **Teoría Básica** describe los aspectos mas relevantes del protocolo TCP/IP, el lenguage de programación C++, el concepto *plataforma digital*, la comunicación entre equipos usando los puertos incorporados y la aquitectura cliente–servidor. Además en este capítulo se describe el problema a solucionar y, parcialmente, el mecanísmo actual de funcionamiento.

El segundo capítulo titulado **Planteamiento del Problema y Propuestas de Solución** describe a fondo el formato actuál de la información, el uso de las API´s (Advanced Program Interface) de la plataforma digital, los requerimientos del usuario, las situaciones actual y futura, las opciones de solución y los requerimientos de software y hardware para la solución elegida.

El tercer capítulo se titula **Análisis, Diseño e Implementación del Sistema**, cubre todo lo referente al análisis y diseño orientado a objetos, así como el diseño y construcción de la vista final para el usuario, las pruebas realizadas al sistema y su factibilidad técnica y operativa.

Los apéndices detallan una serie de información técnica que se tomó en cuenta durante el proceso de construcción del sistema.

Sobre el problema.

Muy brevemente, nuestro trabajo consistió en construir un sistema traductor que tomara el protocolo de transmisión nativo de una fuente de información y lo transformáse al protocolo común de la plataforma digital solicitada.

Las plataformas digitales surgen en la década de los 70´s junto con el auge de la electrónica digital, pero no es hasta mediados de los 80´s que su uso se vuelve una necesidad imperante. Su trabajo consiste en tomar información de diversas fuentes alrededor del mundo y ponerlas en un formato estándar para que los operadores de las casas de bolsa, bancos y demás instituciones en donde se manejan valores, puedan verla y valerse de ella para operar transacciones. Originalmente cada operador debía tener un número considerable de monitores y computadoras frente a él, cada uno de estos dispositivos le presentaría la información de un servicio individual y su manera de operarlos era comunmente distinta. Con el uso de una plataforma digital el operador gana control, coherencia y homogeneidad en la información al verla y usarla de la misma manera sin importar el servicio del que se trate.

El convertir la información de un formato fijado por los proovedores, al formato estándar de la plataforma involucra tratar con los mas diversos sistemas en un extremo y con una red LAN o WAN en el otro. El dispositivo de software, y en ocasiones hardware, que está en medio se ha dado en llamar **Traductor de Protocolo**. No hay reglas fijas para tratar con la información que llega de las diversas fuentes, porque dicha información puede estar codificada en las mas diversas formas, desde señales analógicas de T.V., hasta sistemas muy avanzados de encriptación digital en fibra óptica o una mezcla de ambas. Sin embargo, en el otro extremo se ha estandarizado el uso de piezas modulares de software (API´s) para automatizar la distribución de información lo más posible.

CAPÍTULO I

TEORÍA BÁSICA

1.1. Características, Ventajas y Desventajas del Protocolo TCP/IP.

1.1.1. *Familia de Protocolos* TCP/IP.

Transporte.

- Protocolo de transmisión TCP (**Transmission Control Protocol**), servicios basados en conexión.
- Protocolo de datagrama de usuario UDP (**User Datagram Protocol**), servicios sin conexión.

Enrutamiento.

- Protocolo Internet IP (**Internet Protocol**), maneja la transmisión de información.
- Protocolo Internet de mensajes de control ICMP (**Internet Control Message Protocol**), maneja mensajes de estado para IP.
- Protocolo de información de enrutamiento RIP (**Routing Information Protocol**), determina enrutamiento.
- Primero abrir la ruta más corta OSFP (**Open Shortest Path First**), protocolo alterno para determinar el enrutamiento.

Direcciones de red.

- Protocolo de definición de direcciones ARP (**Address Resolution Protocol**), determina las direcciones.
- Servicio de nombre de dominio DNS (**Domain Name Service**), determina direcciones para nombres de máquina.
- Protocolo inverso de definición de direcciones RARP (**Reverse Address Resolution Protocol**), determina direcciones.

Servicios de usuario.

- Protocolo de arranque BOOTP (**Boot Protocol**), inicializa una máquina en red
- Protocolo de transferencia de archivos FTP (**File Transfer Protocol**), transfiere archivos.
- **Telnet**, permite registros remotos.

Protocolos Gateway.

- Protocolo externo de gateway EGP (**Exterior Gateway Protocol**), Transfiere información de enrutamiento para redes externas.
- Protocolo de gateway a gateway GGP (**Gateway-to-Gateway Protocol**), transfiere información de enrutamiento entre gateways.
- Protocolo interno de gateway IGP (**Interior Gateway Protocol**), transfiere información de enrutamiento para redes internas.

Otros.

- Sistema de archivos de red NFS (**Network File System**), permite que los directorios de una máquina se monten en otra.
- Servicio de información de red NIS (**Network Information System**), mantiene las cuentas del usuario a lo largo de las redes.
- Llamada de procedimiento remoto RPC (**Remote Procedure Call**), permite que se comuniquen aplicaciones remotas.
- Protocolo simple de transferencia de correo SMTP **(Simple Mail Transfer Protocol**), transfiere correo electrónico.
- Protocolo simple de manejo de red SNMP (**Simple Network Management Protocol**), envía mensajes de estado acerca de la red.

TCP/IP surge de la necesidad de desarrollar un procedimiento estandarizado de comunicaciones que se utilizaría de manera inevitable en una variedad de plataformas,

además de una norma que estuviera disponible para todos. La jerarquia de estos protocolos y servicios se muestra en la figura 1.1.1.

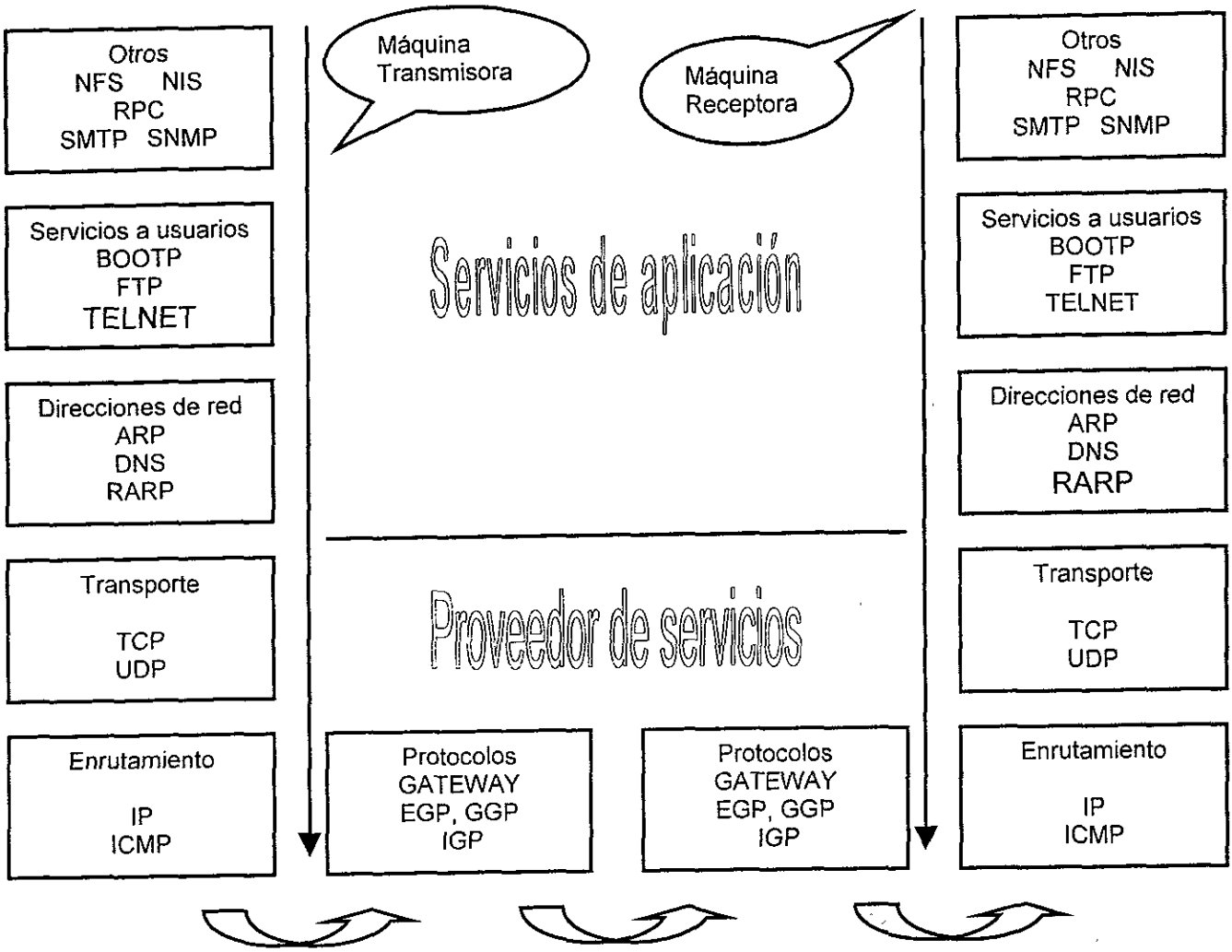


Figura 1.1.1. Esquema de protocolos TCP/IP.

1.1.2. Arquitecturas de Red.

El término red por lo general significa un conjunto de computadora y periféricos, que están conectados por algún medio, esta conexión puede ser directa (por medio de un cable) o indirecta (a través de un módem). Los diferentes dispositivos en la red se comunican entre sí por medio de una serie de reglas predefinidas (protocolos). La estructura de la red, es decir los dispositivos reales y la forma en como están conectados se llama topología de la red.

Si los dispositivos de una red están en una sola ubicación, como un edificio o grupo de habitaciones, se les llama red de área local o LAN (**Local Area Network**). Las LAN por lo general tienen todos los dispositivos en la red conectados por un solo tipo de cable de red. Si los dispositivos están dispersos en forma amplia, como en edificios diferentes o ciudades diferentes, se establecen en varias LAN, unidas por una estructura más grande llamada red de área amplia o WAN (**Wide Area Network**). Una WAN está compuesta por dos a más LAN. Cada LAN tiene su propio cable de red que conecta todos los dispositivos de esta. Las LAN se unen por otro método de conexión, a menudo líneas telefónicas de alta velocidad o cables de red muy rápidos, llamados columnas vertebrales (**BACK BONES**).

Redes de área local.

Aunque existen muchas topologías de LAN, tres son las dominantes: bus, anillo y eje.

La red bus.

Esta red es la más sencilla; comprende una sola ruta de comunicaciones principal en la que cada dispositivo está conectado al cable principal (bus) por medio de un dispositivo llamado tranceptor o caja de empalme, al bus también se le llama columna vertebral. Desde cada tranceptor en el bus, otro cable, corre hasta el adaptador de red del dispositivo en la figura 1.1.2 se muestra un ejemplo de una red de bus.

La ventaja principal de bus es que permite un bus de alta velocidad además de que es inmune a problemas con cualquier tarjeta de red sola, dentro de un dispositivo en la red. Esto se debe a que el tranceptor permite tráfico a través de la columna vertebral, ya sea que un dispositivo esté conectado a la caja de empalme o no. Cada extremo del bus termina en un bloque de resistores u otro dispositivo eléctrico semejante que marque el final del cable desde el punto de vista eléctrico. Cada dispositivo en la ruta tiene un número de identificación especial, o dirección, que le permite al dispositivo saber cuál información recibida es para él.

Una red de bus por lo general se alambra siguiendo los contornos de paredes y edificios conforme se necesita. La mayor parte de los dispositivos en la red de bus pueden enviar o recibir datos a lo largo del bus, empacando un mensaje con la dirección del receptor proyectado.

Una variación de la topología de la red de bus que se encuentra en muchas LAN pequeñas es que utilizan un cable Thin Ethernet o cable de par trenzado. A diferencia de la red de bus no hay transceptores en el bus, en lugar de ellos cada dispositivo está conectado de manera directa en el bus, usando un conector en T en la tarjeta de interfaz de la red, a menudo con un conector llamado BNC. El BNC conecta a la máquina con los dos vecinos, en cada extremo de la red se añade un resistor sencillo del lado desocupado del lado del conector para terminar la red desde el punto de vista eléctrico.

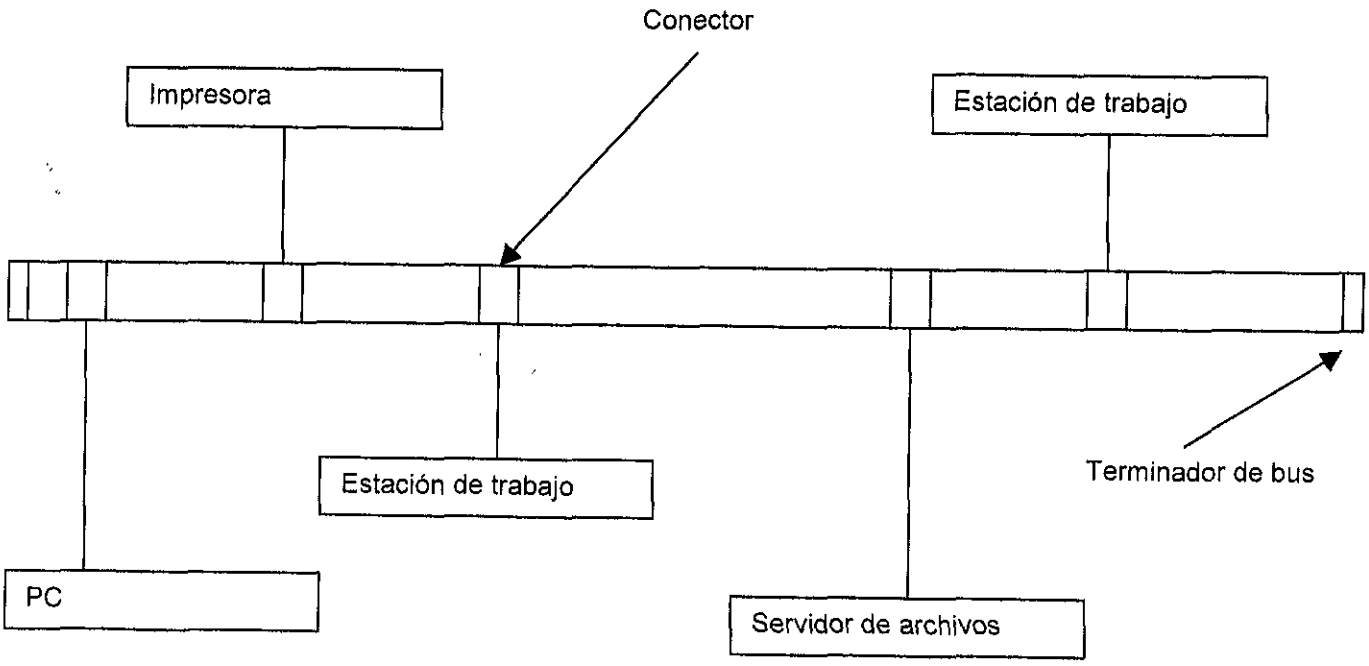


Figura 1.1.2. Esquema de una red de bus que muestra la columna vertebral con transceptores que conducen a dispositivos de la red.

En la figura 1.1.3 Se muestra un esquema de este tipo de red.

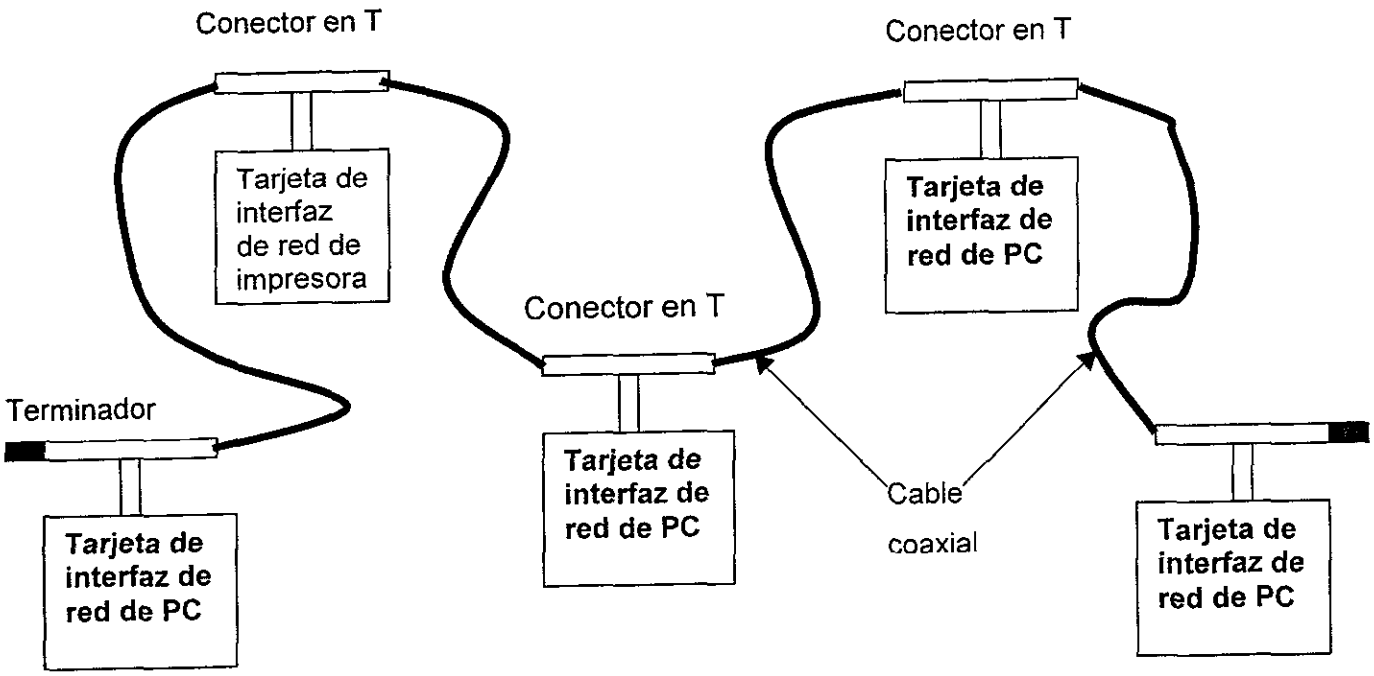


Figura 1.1.3. Esquema de una red de bus máquina a máquina. Cada dispositivo de la red tiene un conector unido en T unido a la tarjeta de interfaz de la red, que conduce a sus dos vecinos, los dos extremos del bus terminan con resistores.

La red de anillo.

Una topología de red de anillo tiene forma de anillo como su nombre lo sugiere. En la figura 1.1.4 se muestra un esquema característico de esta red.

El término anillo es un nombre inapropiado debido a que las redes de anillo no tienen un cable sin fin como en la red de bus con los dos terminadores unidos. En su lugar el anillo se refiere al diseño de la unidad central que maneja el paso de mensajes de la red. En una red de anillo la unidad central de control se llama MAU (**Media Access Unit**) o unidad de acceso a medios. La MAU tiene un circuito de anillo dentro de ella, el anillo dentro de la MAU sirve como el bus para que los dispositivos obtengan los mensajes.

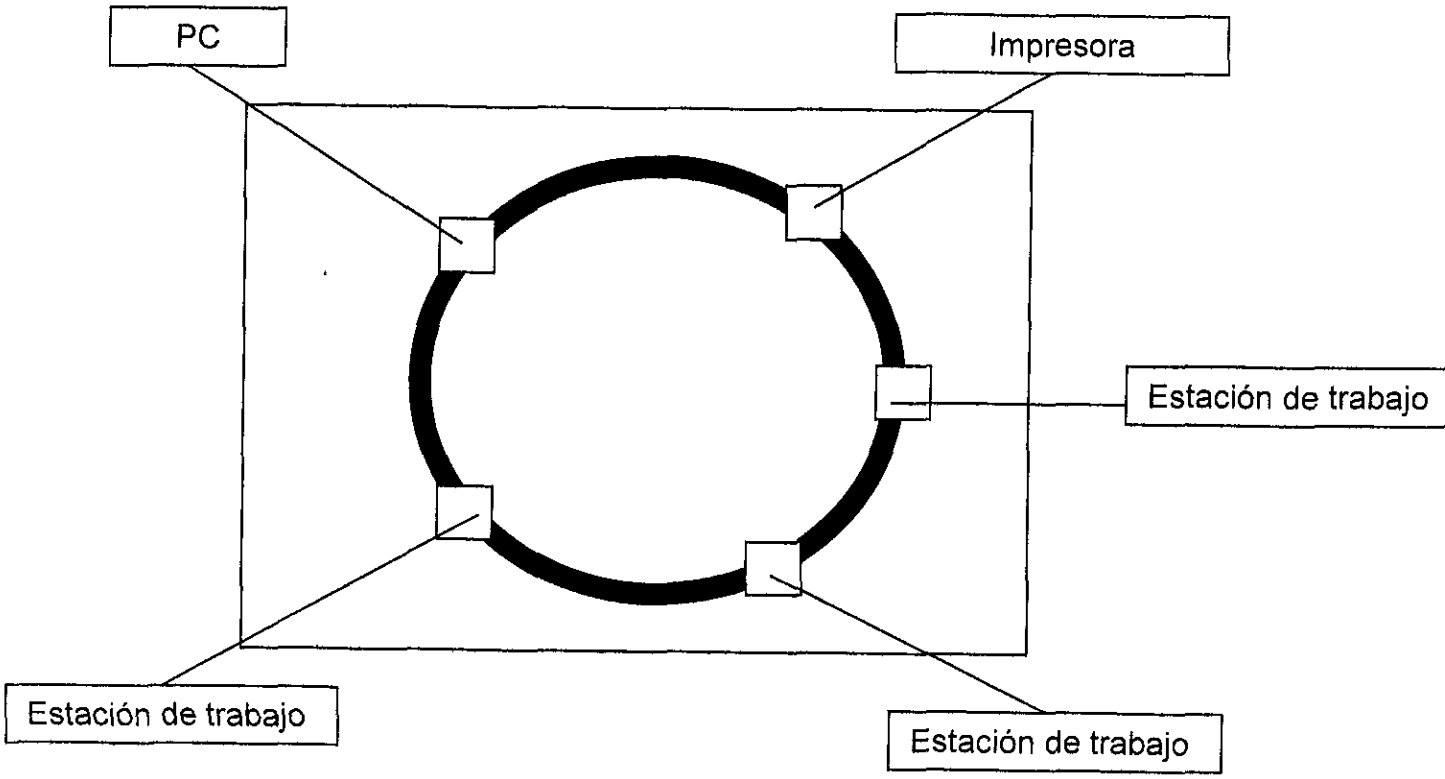


Figura 1.1.4. Esquema de una red de anillo, no existe un cableado de anillo real. El nombre de anillo se deriva de la construcción de la unidad de control central.

Red de eje.

Una red de eje usa un cable principal muy parecido al de la red de bus, el cual se llama plano posterior. La topología de eje se muestra en la figura 1.1.5. Desde el plano posterior, un conjunto de cables conduce a un eje, el cual es una caja que contiene varios puertos en donde se conectan los dispositivos. A menudo los cables hacia un punto de conexión se llaman bajadas porque bajan desde el plano posterior hasta los puertos.

Las redes de eje pueden ser muy grandes, usando un plano posterior de fibra óptica de alta velocidad y bajadas Ethernet un poco más lentas hacia los ejes desde los cuales pueden soportarse un grupo de trabajo.

La red de eje también puede ser pequeña, con un par de ejes soportando unos cuantos dispositivos conectados juntos por cables Ethernet estándar. La red de eje es escalable, lo cual es parte de su atractivo.

El plano posterior puede extenderse a lo largo de una distancia considerable del mismo modo que la red de bus, mientras que los puertos por lo general se agrupan en un conjunto colocado en una caja o tablero. Puede haber muchos tableros o cajas de conexión unidos al plano posterior.

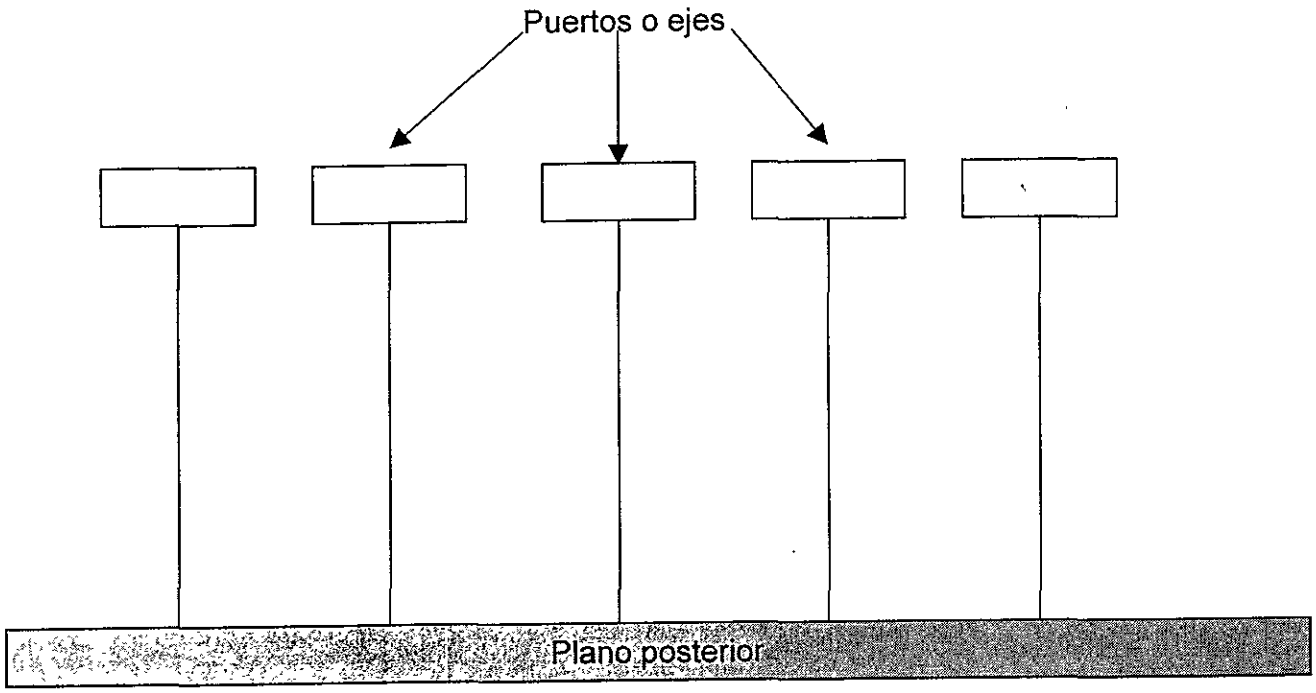


Figura 1.1.5. Esquema de una red de eje.

Red de estrella.

La topología en estrella en una topología punto a punto, ya que los dispositivos se encuentran conectados a un concentrador. Generalmente se le denomina topología de concentradores.

La topología en estrella concentra a todos los dispositivos en una estación centralizada que enruta el tráfico al lugar apropiado. Tradicionalmente esta topología es un acercamiento a la interconexión de dispositivos en la que cada uno de estos se conecta por un circuito separado a través del concentrador.

Esta topología es similar a la red de teléfonos, en donde existe un conmutador y cada llamada que se hace tiene que pasar por él para poder llegar a su destino.

No existe un número máximo de conexiones debido a que los concentradores son cada vez más poderosos y soportan mayor número de dispositivos con un nivel de servicio muy alto. En general el número máximo de estaciones que se pueden conectar al concentrador depende del tráfico que se genere entre ellas, y cuando este es excesivo la red se divide mediante un dispositivo adicional cuya función es aislar el tráfico de un segmento a otro. La figura 1.1.6 muestra una distribución típica con un concentrador.

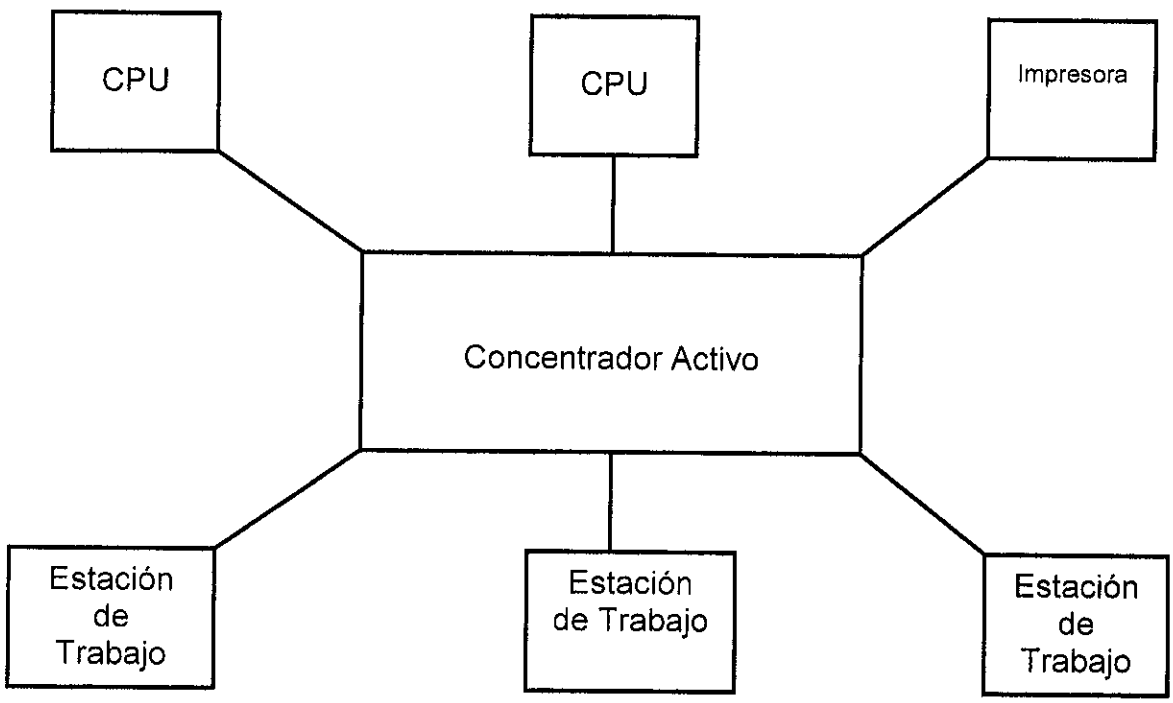


Figura 1.1.6. Esquema de una red en estrella.

Red de malla.

Las redes tipo malla tienen la siguiente característica: todos sus dispositivos están conectados entre sí.

El número de conexiones punto a punto en una red de malla depende del número de dispositivos que integren la red. Tres dispositivos que integren una red necesitan tres conexiones punto a punto, cuatro dispositivos que integren una red de malla necesitan seis conexiones punto a punto.

Cada dispositivo está comunicado con todos los demás dispositivos. Desde un dispositivo en una red de malla existen múltiples rutas de acceso hacia otros dispositivos.

Si la ruta de acceso directo entre dos dispositivos no está disponible, el mensaje puede ser ruteado a través de los otros dispositivos.

Algunas redes híbridas utilizan red de malla entre su estructura.

La figura 1.1.7 muestra un esquema típico de una red de malla.

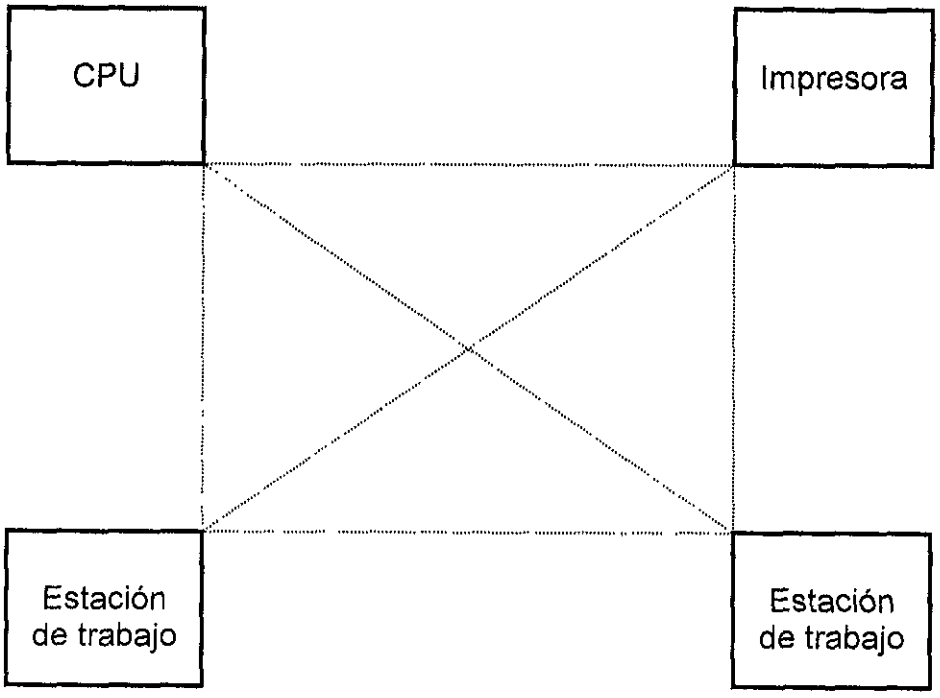


Figura 1.1.7. Esquema de una red de malla.

Redes de área amplia.

Como se mencionó antes, las LAN pueden combinarse en una entidad grande llamada WAN. Las WAN por lo general están compuestas por LAN unidas por una conexión de alta velocidad. En la entrada de cada LAN, una o más máquinas actúan como enlace entre la LAN y la WAN: estos se llaman gateways. Un gateway es la interfaz entre una LAN y una WAN.

Las LAN pueden estar unidas a una WAN por medio de un gateway que maneja el paso de datos entre la columna vertebral de la LAN y la WAN. Otro dispositivo gateway, llamado puente, se usa para conectar las LAN usando el mismo protocolo de red. Los puentes se usan sólo cuando el mismo protocolo de red está en ambas LAN.

1.1.3. Capas del Modelo OSI.

En comunicaciones de sistemas abiertos, dividir todos los requerimientos en grupos de propósito similar es un enfoque razonable, un grupo se encarga de transporte de los datos, otro del empaquetamiento de los mensajes, otro de las aplicaciones de usuario final y así sucesivamente. Cada grupo de tareas relacionadas se llama una capa.

El modelo de referencia OSI (OSI-RM) (Open Systems Interconnection Reference Model) modelo de referencia de interconexión de sistemas abiertos, usa siete capas como se muestra en la figura 1.1.8, la arquitectura TCP/IP es similar pero contiene sólo cinco capas, debido a que combina algo de la funcionalidad OSI de dos capas en una.

Las capas de aplicación, presentación y sesión están orientadas hacia la aplicación, puesto que son responsables de presentar la interfaz de la aplicación al usuario. Las tres son independientes de las capas debajo de ellas y olvidan los medios por los cuales llegan los datos a la aplicación. Estas tres capas se llaman las capas superiores. Las cuatro capas inferiores tienen que ver con la transmisión de los datos, que abarca el empaquetamiento, enrutamiento, verificación y transmisión de cada grupo de datos. Las capas inferiores no se preocupan por el tipo de datos que reciben o envían a la aplicación, sino que sólo tienen que ver con la tarea de enviarlos.

Capa de aplicación.

La capa de aplicación es la interfaz de usuario final para el sistema OSI. Es donde residen las aplicaciones como el correo electrónico, los lectores de noticias de Usenet o los módulos de desplegado de bases de datos. La tarea de la capa de aplicación es desplegar la información recibida y enviar los datos nuevos del usuario a las capas inferiores.

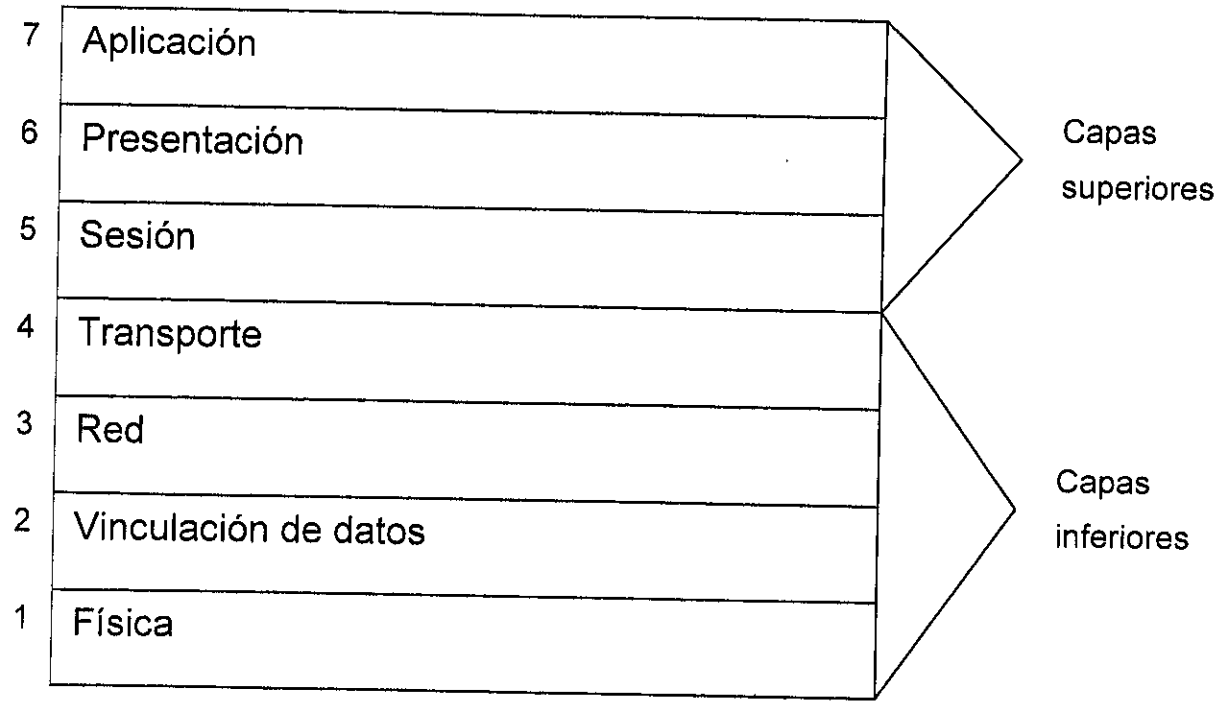


Figura 1.1.8. El modelo de referencia OSI mostrando sus siete capas completas.

La tarea de la capa de presentación es aislar las capas inferiores del formato de los datos de la aplicación. Convierte los datos de la aplicación a un formato común, en ocasiones llamado la representación canónica. La capa de presentación procesa datos dependientes de la máquina, de la capa de aplicación, en un formato independiente de la máquina para las capas inferiores.

La capa de presentación es donde se pierden los formatos de archivo e incluso los formatos de carácter. La conversión del formato de los datos de la aplicación tiene lugar a través de un lenguaje de programación de red común que tiene un formato estructurado. La capa de presentación hace lo inverso para los datos que llegan. Éstos se convierten del formato común a los formatos específicos de las aplicaciones, con base en el tipo de aplicación para que tenga instrucciones la máquina. Si los datos llegan sin instrucciones de reformateo, la información podría no ensamblarse en la manera correcta para la aplicación del usuario.

Capa de sesión.

La capa de sesión organiza y sincroniza el intercambio de datos entre procesos de aplicación. Funcióna con la capa de aplicación para proporcionar conjuntos de datos sencillos llamados puntos de sincronización, que le permiten a una aplicación conocer cómo está progresando la transmisión y recepción de datos. En términos sencillos, la capa de sesión puede pensarse como una capa cronometradora y controladora del flujo.

La capa de sesión interviene en la coordinación de las comunicaciones entre aplicaciones diferentes, permitiendo a cada una conocer el estado de la otra. Un error en una aplicación (sea en la misma máquina o al otro lado del país) lo maneja la capa de sesión para permitirle a la aplicación receptora saber que ha ocurrido un error. La capa de sesión puede resincronizar aplicaciones que están conectadas entre sí en ese momento. Esto puede ser necesario cuando las comunicaciones se interrumpen en forma temporal o cuando ha ocurrido un error que resulta en la pérdida de datos.

Capa de transporte.

La capa de transporte, como su nombre indica, está diseñada para proporcionar la "transferencia transparente de datos de un extremo fuente de un sistema abierto, a un extremo de destino de un sistema abierto", según el Modelo de Referencia OSI. La capa de transporte establece, mantiene y termina comunicaciones entre dos máquinas.

La capa de transporte es responsable de asegurar que los datos enviados correspondan con los datos recibidos. Este papel de verificación es importante para asegurar que los datos se envíen en forma correcta, con un reenvío si se detecta un error. La capa de transporte maneja el envío de datos, determinando su orden y su prioridad.

Capa de red.

La capa de red proporciona el enrutamiento físico de los datos, determinando la ruta entre las máquinas. La capa de red maneja todas estas cuestiones de enrutamiento, liberando las capas superiores de este asunto.

La capa de red examina la topología de la red para determinar la mejor ruta para enviar un mensaje, así como descifrar los sistemas de retransmisión. Es la única capa de red que envía un mensaje de una máquina fuente a una de destino, manejando otros trozos de datos que pasan a través del sistema en su camino hacia otra máquina.

Capa de vinculación de datos.

De acuerdo con el documento de referencia OSI, la capa de vinculación de datos "proporciona el control de la capa física y detecta y posiblemente corrige errores que pueden ocurrir". En la práctica, la capa de vinculación de datos es responsable de corregir errores de transmisión inducidos durante ésta (en oposición a los errores en los datos de aplicación en sí, los cuales se manejan en la capa de transporte).

Por lo general la capa de vinculación de datos está interesada en la interferencia de señales en los medios de transmisión físicos, sean a través de alambre de cobre, cable de fibra óptica o microondas. La interferencia es común, resultado de muchas fuentes, incluyendo rayos cósmicos e interferencia magnética dispersa proveniente de otras fuentes.

Capa física.

La capa física es la capa inferior del modelo OSI y tiene que ver con los "medios mecánicos, eléctricos, funcionales y de procedimiento" requeridos para la transmisión de datos, de acuerdo con la definición de OSI. Esto en realidad es el cableado u otra forma de transmisión.

En la práctica se trata a la capa de vinculación de datos y a la física como una capa combinada, pero la definición OSI formal estipula propósitos diferentes para cada una. TCP/IP incluye las capas de vinculación de datos y física como una capa, reconociendo que la división es más académica que práctica.

1.1.4. Terminología y notaciones.

Paquetes.

En TCP/IP es un término que se refiere a la transmisión de datos entre la capa de Internet y la capa para vínculo de datos. También es un término genérico utilizado para hacer referencia a los datos transferidos a través de una red.

Subsistemas.

Un subsistema es el conjunto de elementos y características de una capa en particular a través de una red.

Entidades.

Una capa puede tener más de una parte en ella, pero no todas estas rutinas están activas a la vez, debido a que podrían no requerirse en ningún momento. Las rutinas activas se llaman entidades.

Notación N.

Las notaciones N, N+1, N+2 etc. Se usan para identificar una capa y las capas que están relacionadas con ella.

N funciones.

Las funciones son las cosas diferentes que hace la capa.

N medios.

Éstos usan la estructura de capas jerárquicas para expresar la idea de que una capa proporciona una serie de medios para la siguiente capa superior.

Servicios.

El conjunto entero de N medios proporcionados a las entidades (N+1) se llama el N servicio, en otras palabras, el servicio es el conjunto entero de N medios proporcionado a la siguiente capa superior.

Encapsulación.

Es la adición de información de control a un paquete de datos. Los datos de control contienen detalles de direccionamientos, verificaciones para detección de errores y funciones de control de protocolo.

La comunicación entre dos aplicaciones OSI en la misma capa es a través de colas hacia la capa que está debajo de ellas. Cada aplicación tiene dos colas, una para cada dirección del proveedor del servicio de la capa inferior. En jerga OSI las dos colas proporcionan operaciones simultáneas entre dos puntos de acción de N servicio.

Un primitivo de servicio puede ser un bloque de datos, un indicador de que algo es requerido o recibido, o un indicador de estado. Un primitivo de solicitud es cuando un servicio envía un primitivo de servicio a la cola (por medio de N-SAP (un punto de acceso a N servicio)) solicitando permiso para comunicarse con otro servicio en la misma capa.

Un primitivo de indicación es lo que envía el proveedor de servicio en la capa debajo de la aplicación transmisora a la aplícación receptora prendida, para permitirle saber que se desea comunicación. Un primitivo de respuesta envía la aplicación receptora al proveedor del servicio de la capa inferior, para conceder el permiso para la comunicación entre los dos usuarios del servicio. Un primitivo de confirmación se envía desde el proveedor de servicio a la aplicación final, para indicar que ambas aplicaciones en la capa de encima se pueden comunicar ahora.

Separación de los datos.

Los datos pueden separarse en secciones pequeñas, o varias secciones pequeñas pueden combinarse en una sección grande para una transferencia más eficiente. Segmentación es el proceso de separar una unidad de datos de N servicio (N-SDU) en varias unidades de datos de N protocolo (N-PDU). Reensamblaje es el proceso de combinar varias N-PDU en una N-SDU. Formación de bloques es la combinación de varias SDU en una PDU más grande dentro de la capa en la que se originó el SDU. División de bloques es la separación de una PDU en varias SDU de la misma capa. Concatenación es el proceso de una capa que combina varias N-PDU de la siguiente capa superior en una SDU. Separación es lo contrario a la concatenación, una capa separa una SDU única en varias PDU para la siguiente capa superior.

Definiciones de conexiones.

Multiplexión es cuando varias conexiones están soportadas por una sola conexión en la siguiente capa inferior. Demultiplexión es lo inverso de la multiplexión, donde una conexión se divide en varias conexiones para la capa que está encima de ella. División es cuando una sola conexión está soportada por varias conexiones en la capa de abajo. Recombinación es lo inverso de la división, de modo que varias conexiones se combinan en una sola para la capa de encima.

Encabezados de protocolo.

La información de control del protocolo es información sobre el datagrama al que está unida. Esta información por lo general está armada en un bloque que está unido al frente de los datos que acompaña y se llama encabezado de protocolo. Los encabezados de protocolo se utilizan para transferir información entre capas, al igual que entre máquinas.

Componentes TCP/IP.

La figura 1.1.9 muestra los elementos básicos de la familia de protocolos TCP/IP. TCP/IP no está comprendido en las dos capas inferiores del modelo OSI(vinculación de datos y física), sino que comienza en la capa de red, donde reside el protocolo Internet. En la capa de transporte intervienen el TCP y el UDP. Las utilerías y protocolos que forman el resto del conjunto TCP/IP se crean encima de estos al usar las capas TCP o UDP e IP para su sistema de comunicaciones. TCP/IP es dependiente del concepto de clientes y servidores. Esto no tiene nada que ver con un servidor al que tiene acceso mediante una estación de trabajo. El término cliente/servidor tiene un significado sencillo en TCP/IP: Cualquier dispositivo que inicie comunicaciones es el cliente y el dispositivo que responde es el servidor. El servidor está respondiendo a las solicitudes del cliente.

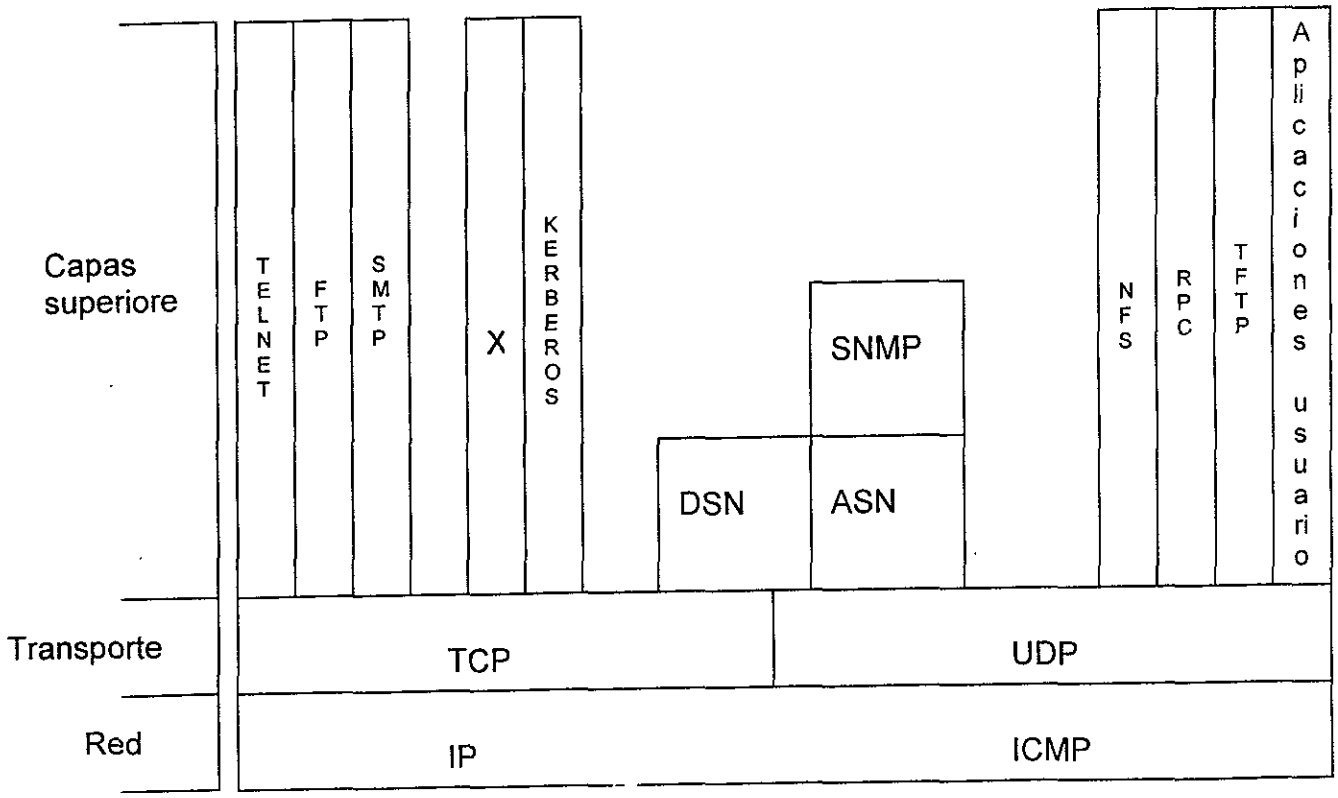


Figura 1.1.9. La coleccion TCP/IP y las capas OSI.

Protocolo Internet (IP).

El protocolo Internet (IP) es un protocolo primario del modelo OSI, así como una parte integral del TCP/IP. El IP define un protocolo no una conexión, el IP es una elección muy buena para cualquier red que necesite un protocolo eficiente para comunicaciones máquina a máquina, aunque enfrenta alguna competencia de protocolo como el IPX de Novell Net Ware.

Sus tareas principales son direccionar los datagramas de información entre computadoras y manejar el proceso de fragmentación de estos datagramas. El protocolo tiene una definición formal de la disposición de un datagrama de información y de la información de un encabezado que se compone de información acerca del datagrama. El IP es responsable del enrutamiento de un datagrama, determinando a dónde será enviado y concibiendo rutas alternativas en caso de problemas.

Otro aspecto importante del propósito del IP tiene que ver con el envío no confiable de un datagrama. No confiable significa que el envío del datagrama no está organizado, debido a que puede demorarse, enrutarse mal o mutilarse en la descomposición y reensamblaje de los fragmentos del mensaje. El IP tiene capacidad inherente para verificar que un mensaje enviado se reciba en forma correcta, las tareas de verificación y control de flujo se dejan a otros componentes en el modelo de capas. Parte del sistema IP define cómo manejan los gateways los datagramas, cómo y cuándo deben producir mensajes de error y como recuperarse del problema que podría surgir. El IP maneja un tamaño máximo de paquete de 65,535 bytes, el cual es mucho mayor de lo que pueden manejar la mayor parte de las redes, de ahí la necesidad de fragmentación. El IP tiene la capacidad de dividir de manera automática, un datagrama de información en datagramas más pequeños si es necesario.

Encabezado de datagrama de IP.

El datagrama es la unidad de transferencia utilizada por el IP. Las especificaciones que definen al IP definen a los encabezados y colas en términos de palabras, donde una palabra es de 32 bits. El encabezado IP tiene una longitud de seis palabras de 32 bits, cuando en el encabezado se incluyen todos los campos opcionales. El encabezado más corto permitido por el IP usa cinco palabras. Para entender todos los campos en el encabezado es útil recordar que el IP depende del hardware, pero debe considerar todas las versiones de software de IP que pueda encontrar. En la figura 1.1.10 se muestra de manera esquemática el diseño del encabezado IP.

Número de versión.

Éste es un campo de 4 bits que contiene el número de versión IP que está usando el software del protocolo. El número de versión se necesita para que el software del IP receptor sepa cómo descifrar el resto del encabezado. Parte de la definición del protocolo estipula que el software receptor primero debe verificar el número de la versión de los datagramas que llegan, antes de proceder a analizar el resto del encabezado y los datos encapsulados. Si el software no puede manejar la versión usada para crear el datagrama, la capa IP de la máquina receptora rechaza el datagrama e ignora el contenido por completo.

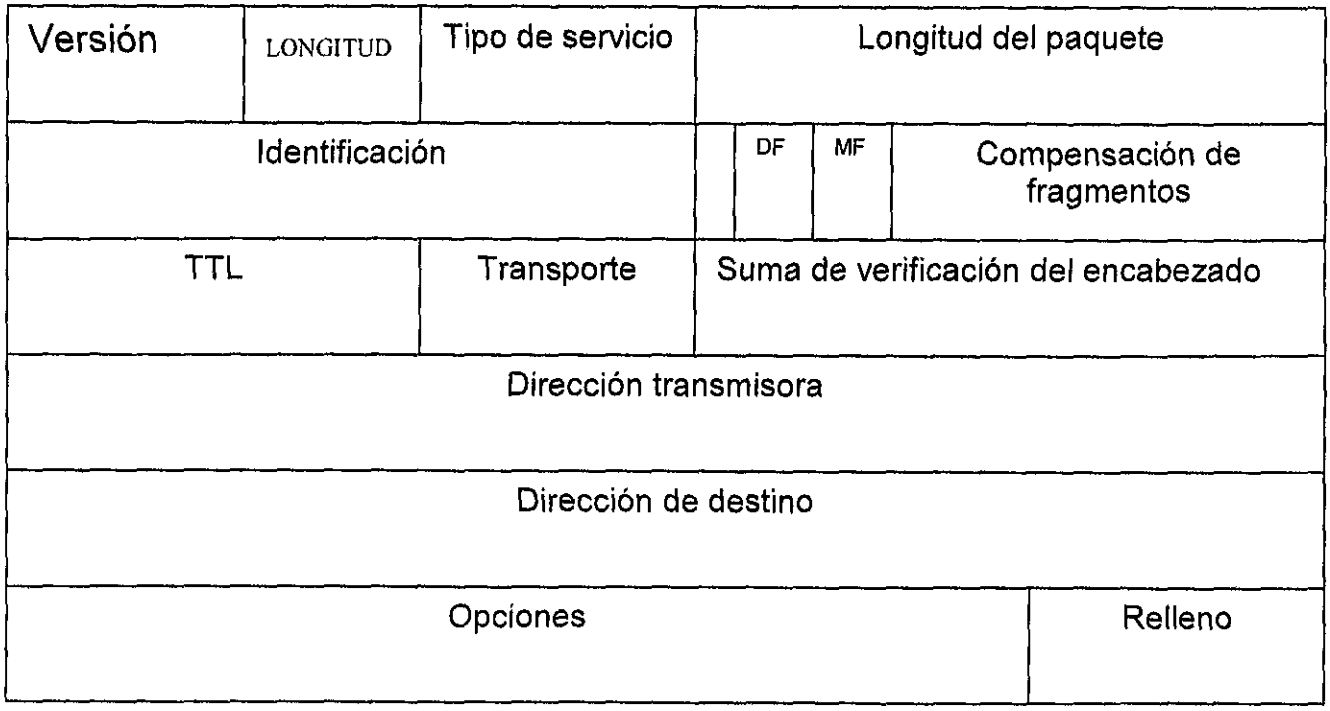


Figura 1.1.10. El diseño del encabezado IP.

Longitud del encabezado.

Este campo de 4 bits refleja la longitud total del encabezado IP creado por la máquina transmisora; se especifica en palabras de 32 bits. El encabezado más corto es de cinco palabras, pero el uso de campos opcionales puede incrementar el tamaño del encabezado hasta su máximo de 6 palabras. Para descifrar de manera correcta el encabezado, el IP debe saber donde termina el encabezado y comienzan los datos, razón por la cual se incluye este campo.

Tipo de servicio.

El campo de tipo de servicio de 8 bits instruye al IP acerca de cómo procesar el datagrama de manera apropiada.

Los primeros tres bits indican la precedencia del datagrama, con un valor de 0 a 7. Entre más grande es el número más importante el datagrama y, al menos en teoría, se enrutará más rápido hacia su destino. Los siguientes tres bits son banderas de un bit que controlan la demora, el paso a través y la confiabilidad del datagrama. Si el bit se fija en cero, el parámetro es normal. Un bit fijado en uno implica una demora lenta, un paso a través alto y una confiabilidad alta para las banderas respectivamente, los últimos dos bits del campo no se usan.

Longitud del datagrama.

Este campo da la longitud total del datagrama, incluye el encabezado en bytes. La longitud del área de datos misma que puede calcularse, restándole a este valor la longitud del encabezado. El tamaño del campo de longitud del datagrama total es de 16 bits, de aquí la longitud máxima 65535 bytes de un datagrama. Este campo se usa para determinar el valor de la longitud que se va a pasar al protocolo de transporte para establecer la longitud total del marco.

Identificación.

Este campo contiene un número que es un identificador único creado por el nodo transmisor. Este número se requiere cuando se reensamblan mensajes fragmentados, asegurando que los fragmentos de un mensaje no estén mezclados con otros. A cada capa de datos recibida por la capa IP de una capa de protocolo más alta, cuando llegan los datos se le asigna uno de estos números de identificación.

Banderas.

El campo de bandera es un campo de 3 bits, el primer bit de los cuales no se usa.
Los dos bits restantes están dedicados a banderas llamadas DF (**Don´t Fragment**) y MF (**More Fragment**), las cuales controlan el manejo de los datagramas cuando la fragmentación es conveniente.

Si la bandera DF está en uno, bajo ninguna circunstancia puede fragmentarse el datagrama. Si la bandera MF se encuentra en uno, al datagrama actual lo siguen más paquetes, los cuales deben reensamblarse para crear el mensaje completo. El último fragmento que se envía como parte de un mensaje tiene su bandera MF fijada en cero de modo que el dispositivo receptor sabe cuando ya no esperar más datagramas.

Compensación de fragmentos.

Si el bit de la bandera MF se fija en uno, el de compensación de fragmentos contiene la posición en el mensaje completo del submensaje contenido dentro del datagrama actual. Esto permite a IP reensamblar los paquetes fragmentados en el orden apropiado. Las compensaciones siempre se dan en relación con el comienzo del mensaje. Éste es un campo de 13 bits, de modo que las compensaciones se calculan en unidades de 8 bits, correspondiendo a la longitud máxima del paquete de 64535 bytes. Si usa el número de identificación para indicar a cuál mensaje pertenece un datagrama recibido, la capa IP en una máquina receptora puede usar la compensación de fragmentos para reensamblar el mensaje completo.

Tiempo de vida (TTL).

Este fragmento de campo da el tiempo en segundos que un datagrama puede permanecer en la red antes de que se deseche. Esto lo establece el nodo transmisor cuando se ensambla el datagrama. Por lo general el campo TTL se fija en 15 o 30 segundos.

Protocolo de transporte.

Este campo contiene el número de identificación del protocolo de transporte al que fue entregado el paquete. Los números los define el NIC (**Network Information Center**), el cual regula Internet. Los dos protocolos más importantes son el ICMP, el cual tiene el número 1, y el TCP, que tiene el número 6.

Suma de verificación de encabezado.

El número en este campo del encabezado IP es la suma de verificación para el campo de encabezado del protocolo, para permitir un procesamiento más rápido. Debido a que el TTL disminuye en cada nodo, la suma de verificación también cambia con cada máquina por la que pasa el datagrama.

Dirección de envío y de destino.

Este campo contiene las direcciones IP de 32 bits de los dispositivos de envío y de destino. Estos campos se establecen cuando se crea el datagrama y no se alteran durante el enrutamiento.

Opciones.

Este campo es opcional, compuesto de varios códigos de longitud variable. Si se usa más de una opción en el datagrama, las opciones aparecen en forma consecutiva en el encabezado IP. Todas las opciones están controladas por un byte que por la general está dividido en tres campos: una bandera de copia de un bit, una clase de opción de dos bits y un número de opción de cinco bits. La clase y número de opción indican el tipo de opción y su valor particular.

Clase de opción	Número de opción	Descripción
0	0	Marca el final de la lista de opciones
0	1	Ninguna opción (usada para relleno)
0	2	Opciones de seguridad (propósitos militares)
0	3	Enrutamiento de fuente holgada
0	7	Activa registro de enrutamiento (agrega campos)
0	9	Enrutamiento de fuente estricta
2	4	Activa el marcador de tiempos (agrega campos)

Tabla 1.1.1. Clases y números de opción validos para encabezados IP.

Relleno.

El contenido del área de relleno depende de las opciones seleccionadas. Por lo general el relleno se usa para asegurar que el encabezado del datagrama es un número redondeado de bytes.

Protocolo Internet de mensajes de control(ICMP).

El ICMP es un sistema de reporte de errores. Es una parte integral del IP y debe incluirse en cada aplicación del IP. Este proporciona mensajes y señales de error consistentes y comprensibles a lo largo de las versiones diferentes del IP y de sistemas operativos distintos. El ICMP es el sistema de comunicaciones de la capa IP. Los mensajes generados por el ICMP los trata el resto de la red como cualquier otro datagrama, pero el software de la capa IP los interpreta diferente. Los mensajes del ICMP tienen un encabezado incorporado de la misma manera que cualquier datagrama IP y, los datagramas ICMP no se diferencian en ningún punto de los datagramas transportadores de datos, hasta que la capa IP de la máquina receptora procesa el datagrama en forma apropiada.

Los mensajes de error enviados por el ICMP se enrutan de regreso a la máquina transmisora del datagrama original. Esto se debe a que sólo las direcciones IP del dispositivo de envío y de destino se incluyen en el encabezado. Debido a que el error no significa nada para el dispositivo de destino, el transmisor es el receptor lógico del mensaje de error. Entonces el transmisor puede determinar, a partir del mensaje ICMP, el tipo de error que ocurrió y establece la mejor forma de enviar de nuevo el datagrama que falló.

Por lo general, cualquier mensaje ICMP que esté reportando un problema de envío, incluye también el encabezado y los primeros 64 bits del campo de datos del datagrama donde ocurrió el problema. Esto es útil para dos cosas.

Primera: permite al dispositivo transmisor igualar el fragmento del datagrama por comparación. Segunda: permite que la máquina receptora del mensaje ICMP realice algún diagnóstico. La figura 1.1.11 muestra el diseño del mensaje ICMP.

El campo Tipo de mensaje de 8 bits en el encabezado ICMP puede tener uno de los valores mostrados en la tabla 1.1.2

Valor	Descripción
0	Eco de respuesta
3	Destino no alcanzable
4	Fuente agotada
5	Se requiere redireccionamiento
8	Eco de solicitud
11	Tiempo de vida excedido
12	Problema con los parámetros
13	Solicitud de marcador de tiempo
14	Respuesta de marcador de tiempo
15	Solicitud de información
16	Respuesta de información
17	Solicitud de máscara de dirección
18	Respuesta de máscara de dirección

Tabla 1.1.2. Valores válidos para el campo ICMP Tipo de mensaje.

El campo código amplía el tipo de mensaje, proporcionando un poco más de información para la máquina receptora. La suma de verificación en el encabezado ICMP se calcula de la misma manera que la suma de verificación del encabezado IP normal.

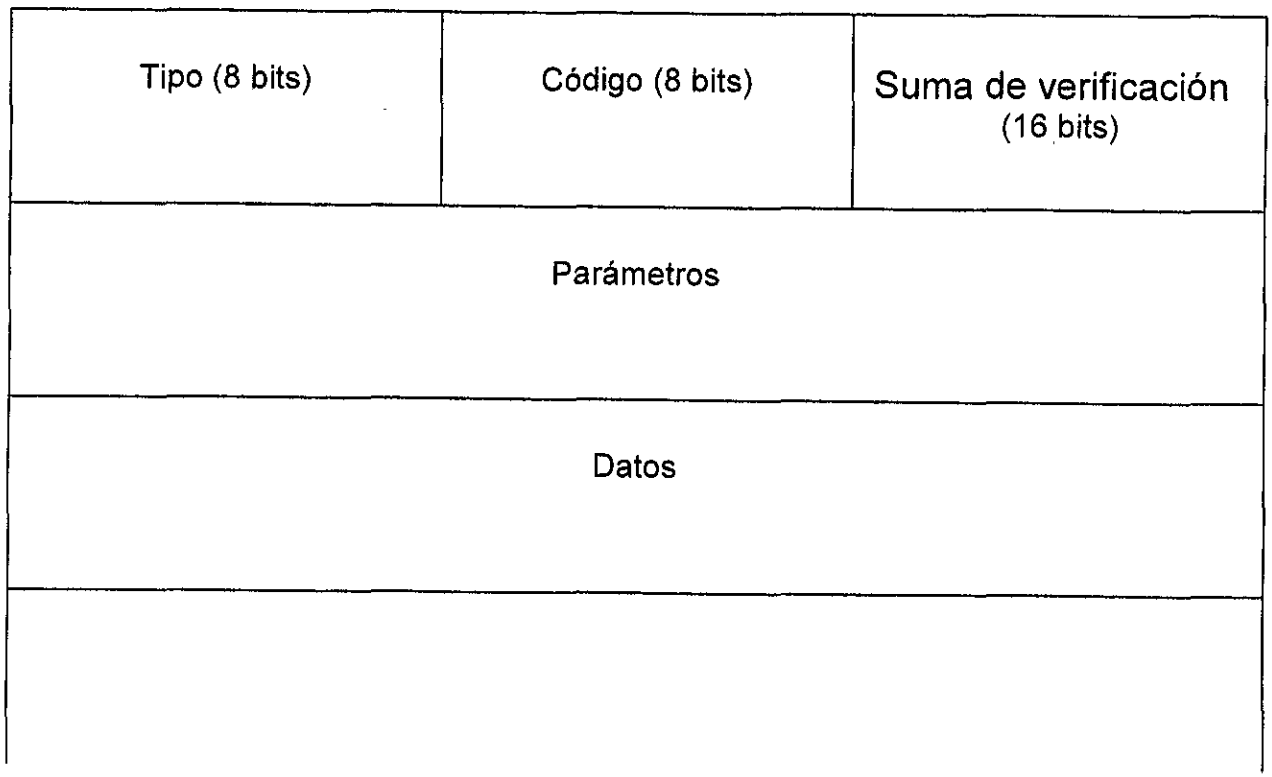
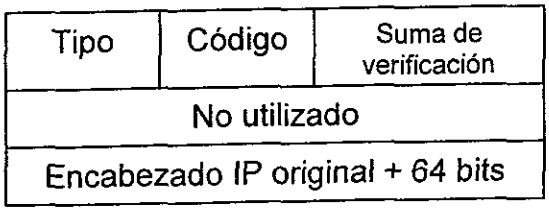
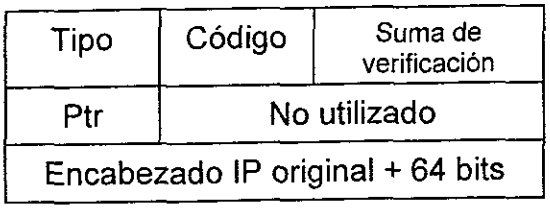


Figura 1.1.11. El diseño de un mensaje ICMP.

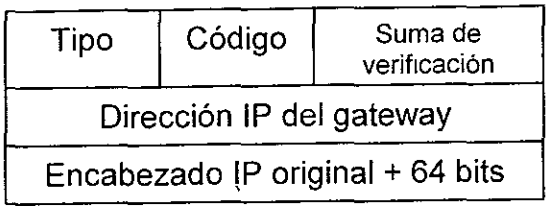
El diseño del mensaje ICMP es ligeramente diferente para cada tipo de mensaje. La figura 1.1.12 muestra los diseños de cada tipo de encabezado de mensaje ICMP.



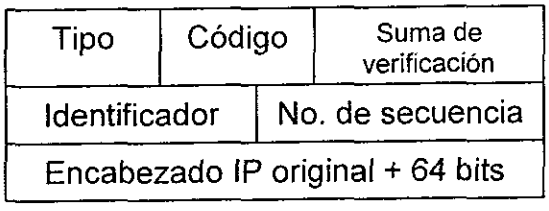
Destino no alcanzable, Fuente agotada, Tiempo excedido



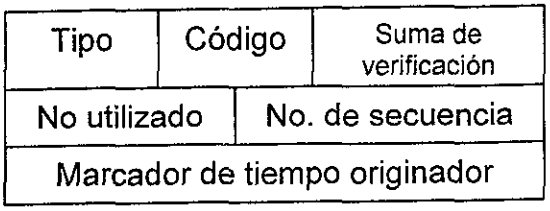
Problema con el parámetro



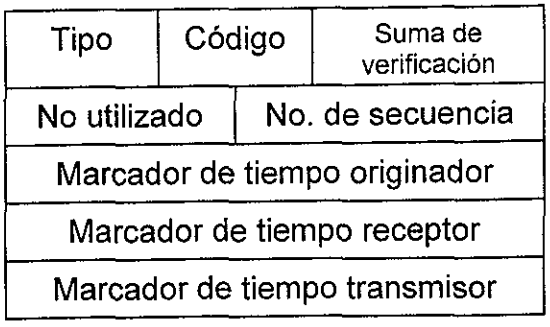
Redireccionar



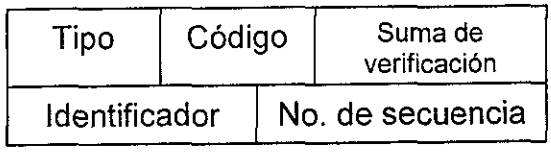
Eco de solicitud y Eco de respuesta



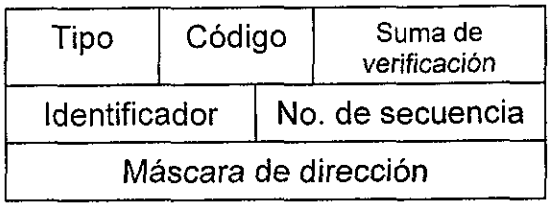
Solicitud de marcador de tiempo



Respuesta de marcador de tiempo



Solicitud y respuesta de información.
Solicitud de máscara de dirección



Respuesta de máscara de dirección

Figura 1.1.12. Diseños del encabezado de mensajes ICMP.

Protocolo de datagrama de usuario (UDP).

El TCP es un producto basado en la conexión. Hay ocasiones cuando se requiere un protocolo sin conexión, de modo que se usa el UDP. El UDP se usa tanto con el protocolo trivial de transferencia de archivos (TFTP) como en el procedimiento de llamada remota (RCP). Las comunicaciones sin conexión no proporcionan confiabilidad, lo que significa que no hay indicación para el dispositivo transmisor de que el mensaje se ha recibido en forma correcta. Los protocolos sin conexión tampoco ofrecen capacidades de recuperación de errores lo cual debe ignorarse, o bien proporcionarlos en las capas superiores o inferiores. El UDP es mucho más sencillo que el TCP. Ocasiona una interfaz con el IP (u otros protocolos) sin la molestia del control del flujo o los mecanismos de recuperación de errores, actuando tan sólo como un transmisor y receptor de datagramas.

El encabezado del mensaje UDP es mucho más sencillo que el del TCP. Los campos son como sigue:

- Puerto fuerte: un campo opcional con el número de puerto. Si no se especifica un número de puerto, el puerto se fija en cero.
- Puerto de destino: el puerto en la máquina de destino.
- Longitud: La longitud del datagrama, incluyendo encabezado y datos.
- Suma de verificación: Un complemento de uno de 16 bits de la suma de complemento de uno del datagrama, incluyendo un pseudoencabezado parecido al del TCP.

El campo de suma de verificación del UDP es opcional, pero si no se usa, no se aplica ninguna suma de verificación al segmento de datos, debido a que la suma de verificación del IP sólo se aplica al encabezado IP. Si no se usa la suma de verificación, el campo debe fijarse en cero. El diseño de las TCP PDU (por lo común llamada encabezado) se muestra en la figura 1.1.13.

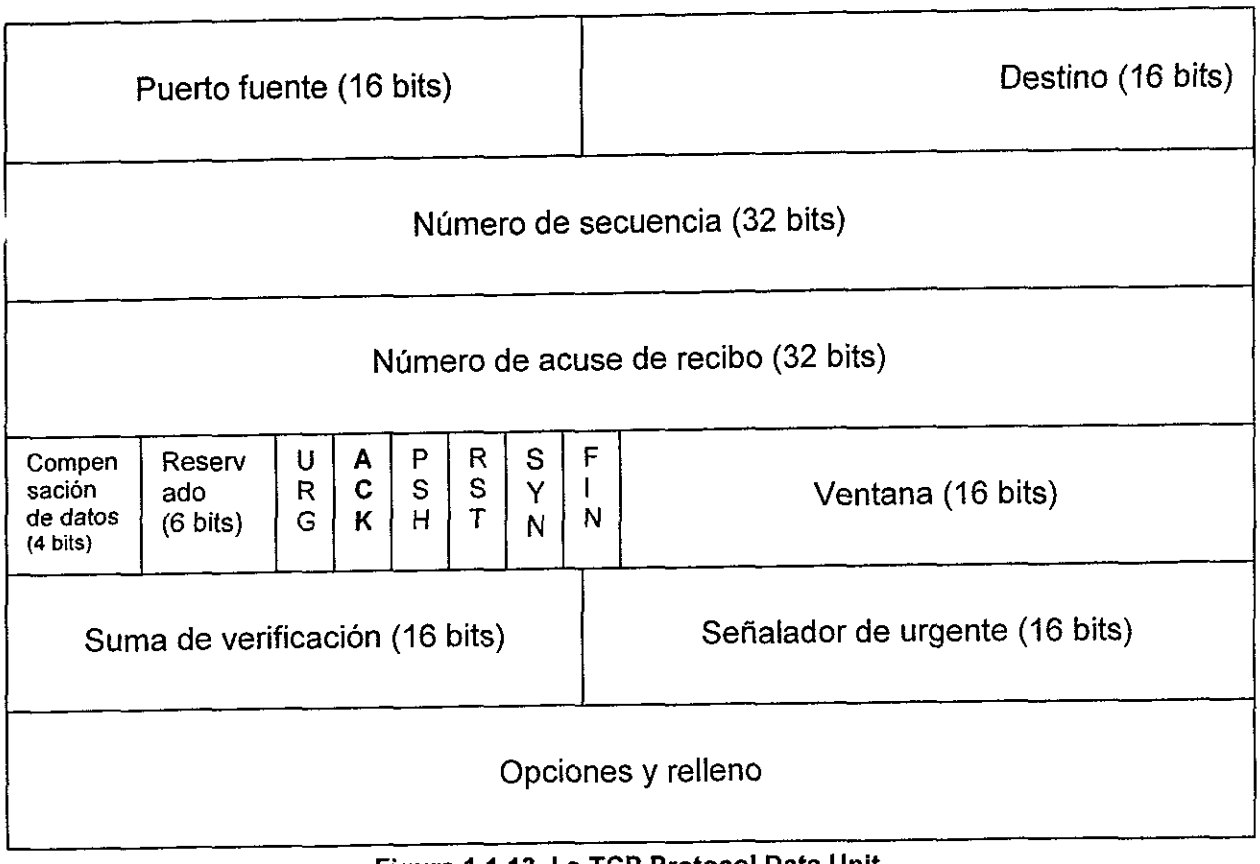


Figura 1.1.13. La TCP Protocol Data Unit.

Protocolo gateway a gateway (GGP).

El GGP se usa para comunicaciones entre gateways con núcleo. El GGP es un protocolo de distancia-vector, lo que significa que los mensajes tienden a especificar un destino (vector) y la distancia hasta ese destino. Para que un protocolo-vector sea efectivo, un gateway debe tener información completa acerca de todos los gateways en la red; de otra manera, calcular una distancia con un protocolo del tipo de menos saltos no puede tener éxito.

Un gateway establece sus conexiones con otros gateways enviando mensajes, esperando las respuestas y luego creando una tabla. Esto se logra al inicio cuando un gateway se instala y no tiene información de enrutamiento en absoluto. Este aspecto de las comunicaciones no se define dentro del GGP sino se basa en mensajes específicos de red. Una vez que la tabla inicial se ha definido se usa el GGP para todos los mensajes.

La conectividad con otro gateway en Internet se determina usando el método K de N. En este procedimiento un gateway envía un mensaje de resonancia a otro gateway y espera la respuesta. Repite esto cada 15 segundos. De acuerdo con las normas Internet, si el gateway no recibe tres (K) respuestas dentro de cuatro (N) solicitudes, el otro gateway se considera caído y los mensajes de enrutamiento no se envían a ese gateway.

Cada mensaje entre gateways tiene un número de secuencia que se incrementa con cada mensaje transmitido. Cada gateway rastrea su propio número de secuencia para enviarlo a todas las otras gateways que estén conectadas a ella, así como los números de secuencia que llegan de ese gateway. No necesariamente son los mismos, debido a que podrían fluir más mensajes en una dirección que en otra, aunque por lo general cada mensaje debería tener un acuse de recibo o una respuesta de algún tipo.

Los números de secuencia tienen significados importantes para los mensajes. Cuando un gateway recibe un mensaje de otro gateway compara el número de secuencia en ese mensaje con el último número de secuencia recibido en sus tablas internas. Si el último mensaje tiene un número de secuencia más alto que el último mensaje recibido, el gateway acepta el mensaje y actualiza su número de secuencia con el último valor recibido. Si el número fue menor que el último número de secuencia recibido, el mensaje se considera obsoleto y se ignora, y se devuelve un mensaje de error conteniendo el mensaje recién recibido. El formato de mensaje GGP se muestra en la figura 1.1.14.

El primer campo es el tipo de mensaje, el cual se fija en un valor de 12 para información de enrutamiento. El número de secuencia constituye un contador creciente para cada mensaje. El campo Update está establecido con un valor de cero, a menos que el gateway transmisor desee una actualización del enrutamiento para la dirección de destino proporcionada, en cuyo caso se fija un valor de 1. El campo número de distancias contiene el número de grupos de direcciones contenido en el mensaje actual.

Tipo (8 bits)	No usado (8 bits)
Número de secuencia (16 bits)	
Actualización (8 bits)	Número de distancia (8 bits)
Distancia 1	No. De redes en 1
Primera red a la distancia 1 (24 bits)	
Segunda red a la distancia 1 (24 bits)	

Etc.....

Figura 1.1.14. El formato del mensaje GGP.

Para cada grupo de distancias en el mensaje se proporciona un valor de distancia y el número de redes que puede alcanzar en ésa, seguidos por todas las identificaciones de la dirección de la red. De acuerdo con la norma GGP, no todas las distancias necesitan reportarse, pero entre más información se proporcione, mayor utilidad tendrá el mensaje para cada gateway.

Protocolo exterior para gateway (EGP).

El EGP se usa para transferir información entre gateways vecinos sin núcleo. Los gateways sin núcleo contienen detalles completos acerca de los vecinos inmediatos y las máquinas enlazadas con ellos, pero carecen de información sobre el resto de la red. Los gateways con núcleo saben todo acerca de otros gateways con núcleo, pero con frecuencia carecen de los detalles de las máquinas que están más allá de un gateway.
El EGP por lo general está restringido a información dentro del sistema autónomo del gateway. Esto impide que pase demasiada información a través de las redes, en especial cuando la mayor parte de la información que se relaciona con sistemas autónomos externos sea utilizable para otro gateway. Por consiguiente, el EGP impone restricciones a los gateways acerca de las máquinas a las que el EGP pasa información de enrutamiento.

Debido a que EGP fue ideado para permitir que sistemas remotos intercambien información de enrutamientos y mensajes de estado, el protocolo se basa, sobre todo, en solicitudes o comandos seguidos por respuestas. Los cuatro comandos EGP y sus respuestas posibles se muestran en la tabla 1.1.3

Nombre del comando	Descripción del comando	Nombre de la respuesta	Descripción de las respuestas
Request	Solicita que un vecino se vuelva un gateway	Confirm/Refuse	Acepta o rechaza solicitud
Cease	Solicita la terminación de un vecino	Cease/Ack	Acepta la terminación
Hello	Solicita confirmación de enrutamiento a un vecino	IHU	Confirma el enrutamiento
Poll	Solicita que el vecino proporcione información de la red	Update	Proporciona información de la red

Tabla 1.1.3. Comandos EGP.

La estructura de los diferentes mensajes usados por el EGP se muestra en la figura 1.1.15

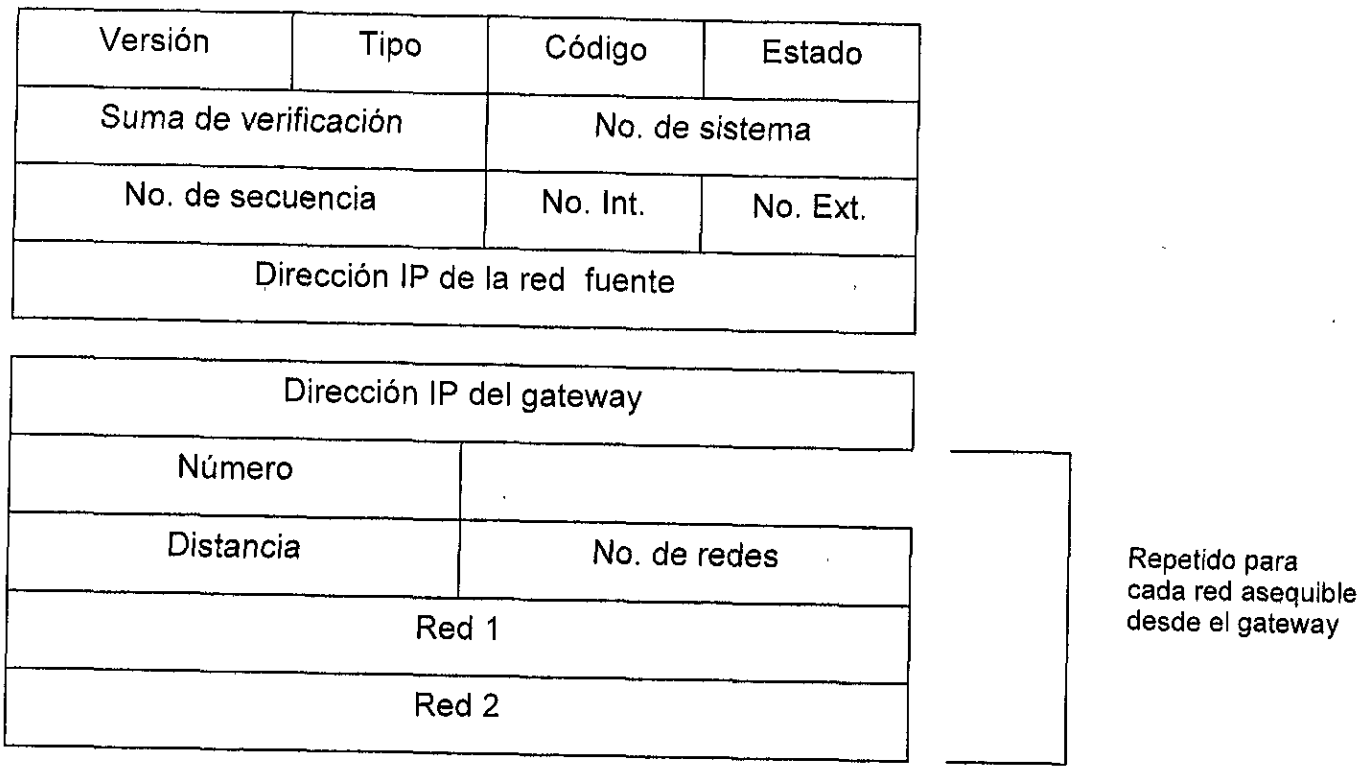


Figura 1.1.15. Formato del mensaje EGP.

Los campos tienen el siguiente significado:

- El campo versión contiene el número de versión del EGP de la máquina transmisora.
- El campo Type identifica el tipo de mensaje EGP. Existen 10 tipos de mensajes en el EGP.
- El campo code contiene un valor que identifica el subtipo del mensaje.
- El campo status se usa con los campos Type y Code para reflejar el estado actual del gateway.
- El campo Checksum se calcula para el mensaje EGP de la misma manera que en otros encabezados TCP/IP.
- El System Number es una identificación del sistema autónomo al que pertenece el gateway transmisor.
- El Sequence Number del mensaje es un contador creciente para cada mensaje, usado también para identificar una respuesta a un mensaje previo.
- El campo Reason del mensaje Error puede contener uno de los siguientes enteros:

  0→Error no especificado

  1→Encabezado EGP malo

  2→Campo de datos EGP malo

  3→Información de asequibilidad no disponible

  4→Polling excesivo

  5→Respuesta a una encuesta no recibida

Protocolo interno para gateway (IGP).

Por lo general la elección de un IGP se hace con base en la arquitectura de la red y en la conveniencia de los requerimientos del software de la red. El RIP y el HELLO son ejemplos de IGP.

Tanto el RIP como el HELLO calculan distancias hacia su destino y sus mensajes contienen un identificador de máquina y la distancia hasta esa máquina.

En general, los mensajes tienden a ser largos, debido a que contienen muchos registros para una tabla de enrutamiento. Ambos protocolos están conectándose en forma constante entre vecinos, para asegurar que las máquinas están activas y comunicándose, lo que puede ocasionar que aumente el tráfico de la red.

Protocolo de información del enrutamiento (RIP).

El RIP encontró un amplio uso como parte de las instalaciones de software LAN. El RIP usa una tecnología de emisión, esto quiere decir que los gateways emiten sus tablas de enrutamiento a otros gateways en la red a intervalos regulares. Esto también es una de las desventajas del RIP, debido a que el aumento de tráfico de la red y la mensajería ineficiente pueden volver lentas a las redes en comparación con otros IGP. El RIP tiende a obtener información acerca de todos los destinos en el sistema autónomo al que pertenece el gateway.

El formato de un mensaje RIP se muestra en la figura 1.1.16. El encabezado del mensaje se compone de tres campos para el comando, el número de versión del protocolo RIP y un campo reservado no se utiliza. El resto del mensaje contiene información de la dirección. Cada conjunto comienza con un identificador de la familia de protocolos usada y un conjunto de identificadores de red. Existen 96 bits disponibles para la dirección de red, de los cuales solo son necesarios un máximo de 32 para una dirección de Internet. El último campo es un valor métrico que por lo general indica el número de saltos hacia la red.

Cada máquina basada en el RIP en la red mantiene una tabla de enrutamiento, con un registro para cada máquina que puede comunicarse a ella. La tabla tiene registros para la dirección IP objetivo, su distancia, la dirección IP del siguiente gateway en el camino hacia el objetivo, una bandera para mostrar si la ruta se ha actualizado en forma reciente y un conjunto de temporizadores que controlan la ruta. La distancia se expresa como un número de saltos requeridos para alcanzar el objetivo y tiene un valor de 1 a 15. Si el objetivo es inalcanzable se establece un valor 16.

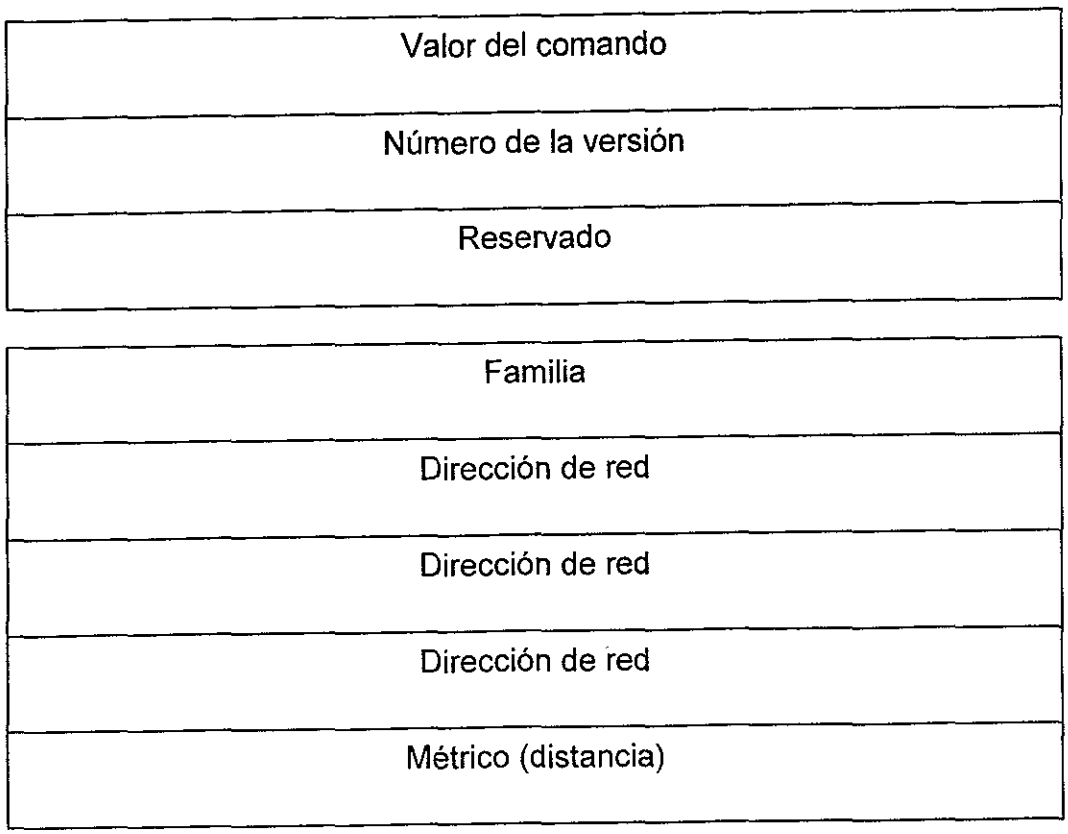


Figura 1.1.16. Formato del mensaje RIP.

Protocolo HELLO.

El protocolo HELLO se usa con frecuencia, en especial donde intervienen instalaciones TCP/IP. Difiere del RIP en que HELLO usa el tiempo en lugar de la distancia como un factor de enrutamiento. Esto requiere que la red de máquinas tenga un cronometraje razonablemente preciso, que se sincroniza con cada máquina. Por esta razón, el protocolo HELLO depende de mensajes de sincronización del reloj.

El formato de un mensaje HELLO se muestra en la figura 1.1.17. Los campos primarios del encabezado son los siguientes:

- Una suma de verificación del mensaje entero.
- La fecha actual de la máquina transmisora.

- La hora actual de la máquina transmisora.
- Un marcador de tiempo usado para calcular demoras en viaje redondo.
- Una compensación que señala los registros siguientes.
- Un número de hosts que siguen como una lista

Suma de verificación
Fecha
Hora
Marcador de tiempo
Compensación
Número de host

Demora para el host 1
Compensación para el host 1

etc.....

Figura 1.1.17. Formato del mensaje HELLO

Protocolo abrir primero la ruta más corta (OSPF).

El OSPF usa la información de la dirección de su destino y el tipo de servicio en un encabezado de datagrama IP para desarrollar una ruta. A partir de una tabla de enrutamiento que contiene información sobre la topología de la red, un gateway OSPF, determina la ruta más corta usando una métrica de costo, la cual considera los factores de velocidad de la ruta, tráfico, confiabilidad, seguridad y diversos aspectos de la conexión. Siempre que las comunicaciones deban dejar una red autónoma, el OSPF llama a ese enrutamiento externo. La información requerida para una ruta externa puede derivarse tanto del OSPF como del EGP.

Existen dos tipos de enrutamiento externo con el OSPF. Una ruta tipo 1 implica los mismos cálculos para la ruta externa que para la interna. En otras palabras, los algoritmos del OSPF se aplican tanto para las rutas internas como para las externas. Una ruta tipo 2 usa el sistema OSPF sólo para calcular una ruta hasta el gateway del sistema destino, ignorando cualquier ruta del sistema autónomo remoto. Esto tiene la ventaja de que puede ser independiente del protocolo usado en la red de destino, lo cual elimina la necesidad de convertir la métrica.

El OSPF permite que una red autónoma grande se divida en áreas más pequeñas, cada una con su gateway y algoritmos de enrutamiento propios. El movimiento entre las áreas es por una columna vertebral o por las partes de la red que enrutan mensajes entre las áreas. El OSPF define varios tipos de enrutadores o gateways:

- Un enrutador interno es aquel para el cual todas las conexiones pertenecen a la misma área, o uno en donde sólo se hacen conexiones de columna vertebral.
- Un enrutador de límite es el enrutador que no satisface la descripción de uno interno (tiene conexiones fuera de un área).
- Un enrutador de columna vertebral tiene una interfaz, hacia la columna vertebral.
- Un enrutador de frontera es un gateway, que tiene una conexión con otro sistema autónomo.

El OSPF está diseñado para permitir a los gateways enviar mensajes entre sí por medio de conexiones de interred. Estos mensajes de enrutamiento se llaman anuncios, los cuales se envían por medio de mensajes de actualización HELLO. En el OSPF se utilizan cuatro tipos de anuncio:

- Un anuncio Router Links (enlaces de enrutado) proporciona información sobre las conexiones de un enrutado local en un área. Este mensaje se emite por toda la red.
- Un anuncio Network Links (enlaces de red) proporciona una lista de enrutadores que están conectados en la red. También se emite por toda la red.
- Un anuncio Summary Links (enlaces de resumen) contiene información acerca de las rutas fuera del área. Se envía por enrutadores de límite a su área completa.

- Un anuncia Autonomous System External Links (enlace externo de sistema autónomo)contiene información sobre rutas en sistemas externos autónomos. Lo usan los enrutadores de frontera, pero abarca todo el sistema. El formato del encabezado del mensaje OSPF se muestra en la figura 1.1.18.

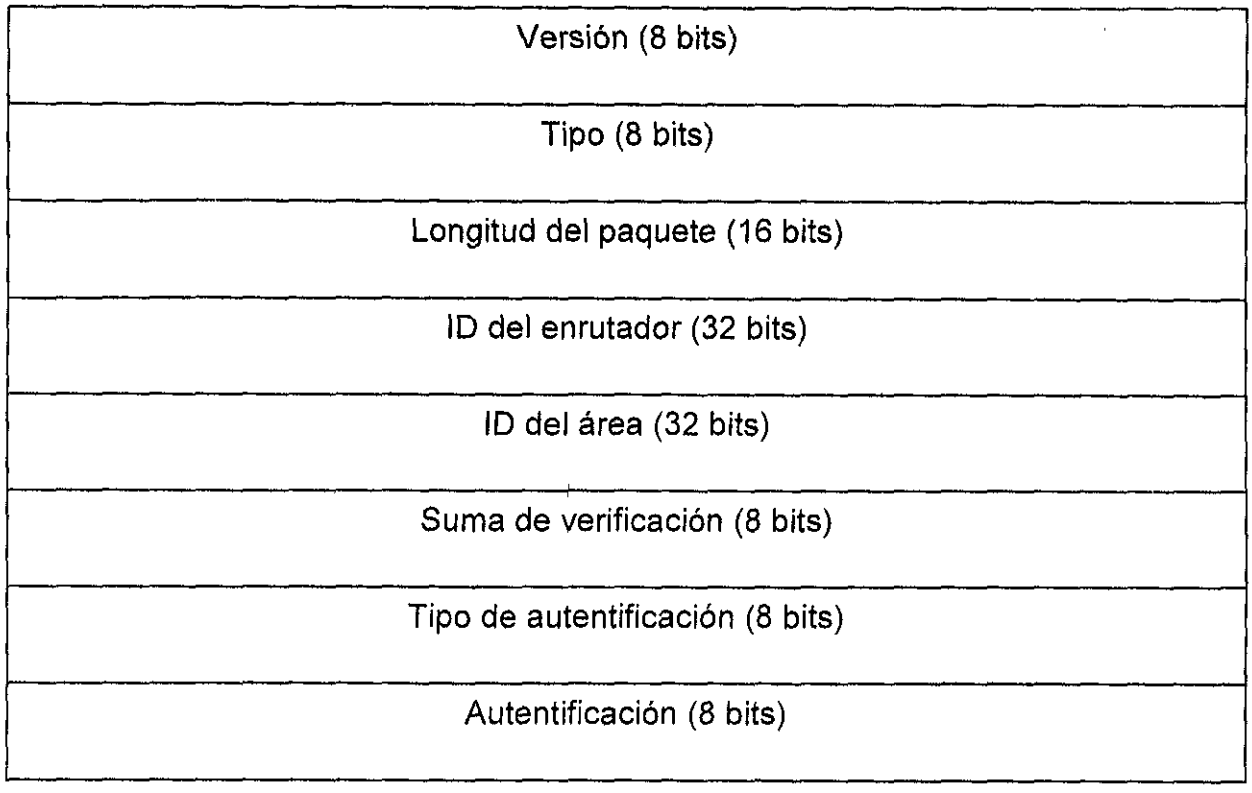


Figura 1.1.18. Formato del encabezado del mensaje OSPF.

El campo número de versión identifica la versión del protocolo OSPF en uso. El campo tipo identifica el tipo de mensaje y podría tener un valor de los mostrados en la tabla 1.1.4.

Tipo	Descripción
1	Hello
2	Descripción de la base de datos
3	Solicitud de estado de enlace
4	Actualización de estado de enlace
5	Acuse de recibo de estado de enlace

Tabla 1.1.4. Valores del campo tipo del encabezado OSPF.

El campo longitud del paquete contiene la longitud del mensaje, incluyendo el encabezado. La ID del enrutador es la identificación de la máquina transmisora y la ID del área identifica el área donde está la máquina transmisora. El campo suma de verificación usa el mismo algoritmo que IP para verificar el mensaje entero, incluyendo el encabezado. El campo tipo de autentificación, identifica el tipo de autentificación que va a usar. El campo autentificación contiene el valor que se usa para autentificar el mensaje, si es aplicable.

Protocolo de definición de direcciones (ARP)

El trabajo del protocolo de definición de direcciones (ARP) es convertir las direcciones IP a direcciones físicas (de red y locales) y, al hacerlo, eliminar la necesidad de que las aplicaciones conozcan las direcciones físicas. El ARP es una tabla con una lista de direcciones IP y sus correspondientes direcciones físicas. La tabla se llama caché ARP. La disposición de un caché ARP se muestra en la figura 1.1.19. Cada fila corresponde a un dispositivo, con cuatro piezas de información para cada dispositivo.

- Índice IF: el puerto físico (interfaz).
- Dirección física: la dirección física del dispositivo.
- Dirección IP: la dirección IP correspondiente a la dirección física.
- Tipo: el tipo de registro en él caché ARP

	ÍNDICE	DIRECCIÓN FÍSICA	DIRECCIÓN IP	TIPO
Registro1				
Registro 2				
Registro 3				
Registro 4				

Figura 1.1.19. La disposición de la tabla de traducción de direcciones caché ARP.

El tipo de mapeo es uno de los cuatro posibles valores indicado en el estado del registro en cada caché ARP. Un valor 2 significa que el registro no es válido; un valor 3 significa que el mapeo es dinámico; un valor 4 significa estático; y un valor de 1 significa que no es ninguno de los anteriores.

Cuando ARP recibe una dirección IP del dispositivo receptor, busca en el caché ARP para realizar una comparación. Si encuentra una igual, regresa la dirección física. Si no encuentra una igual para la dirección IP, envía un mensaje fuera de la red. El mensaje llamado solicitud ARP es una emisión que reciben todos los dispositivos en la red local. La solicitud ARP contiene la dirección IP del dispositivo receptor pretendido, si un dispositivo reconoce la dirección IP como perteneciente a este, el dispositivo envía un mensaje de respuesta que contiene su dirección física de regreso a la máquina que generó la solicitud ARP, la cual coloca la información en su caché ARP para uso futuro. La disposición de la solicitud ARP se muestra en la figura 1.1.20.

<table>
<tr><td colspan="2">Tipo de hardware (16 bits)</td></tr>
<tr><td colspan="2">Tipo de protocolo (16 bits)</td></tr>
<tr><td>Longitud de la dirección de hardware</td><td>Longitud de la dirección de protocolo</td></tr>
<tr><td colspan="2">Código de operación (16 bits)</td></tr>
<tr><td colspan="2">Dirección del hardware transmisor</td></tr>
<tr><td colspan="2">Dirección del IP transmisor</td></tr>
<tr><td colspan="2">Dirección del hardware receptor</td></tr>
<tr><td colspan="2">Dirección del IP receptor</td></tr>
</table>

Figura 1.1.20. La disposición de la solicitud ARP y de la respuesta ARP.

Cuando se envía una solicitud ARP se usan todos los campos en la disposición, excepto la dirección del hardware receptor, a la cual está tratando de identificar la solicitud. En una respuesta ARP, se usan todos los campos. Los campos y sus propósitos son los siguientes:

- Tipo de hardware: el tipo de interfaz de hardware.
- Tipo de protocolo: el tipo de protocolo que está usando el dispositivo de hardware.
- Longitud de la dirección de hardware: la longitud de cada dirección de hardware en el datagrama, dados en bytes.
- Longitud de la dirección del protocolo: la longitud de la dirección de protocolo en el datagrama, dada en bytes.
- Código de operación: el Opcode indica si el datagrama es una solicitud ARP o una respuesta ARP. Si el datagrama es una solicitud, el valor se fija en 1. Si es una respuesta, el valor se fija en 2.
- Dirección de hardware transmisor: la dirección de hardware del dispositivo transmisor.
- Dirección IP del transmisor: la dirección IP del dispositivo transmisor.
- Dirección IP del receptor: la dirección IP del receptor.
- Dirección del hardware receptor: la dirección del hardware del dispositivo receptor.

Un defecto obvio con el sistema ARP es que si el dispositivo no conoce su propia dirección IP no hay manera de generar solicitudes y respuestas. Esto puede suceder cuando se agrega un dispositivo nuevo a la red. La única dirección de la que se percata el dispositivo es la dirección física establecida, ya sea con interruptores en la interfaz de la red o por medio de software. Una solución sencilla es el protocolo inverso de definición de direcciones (RARP), el cual funciona a la inversa del ARP, enviando la dirección física afuera y esperando que regrese una dirección IP. La respuesta que contiene la dirección IP se envía por un servidor RARP, una máquina que puede suministrar la información. Aunque el dispositivo originador envía el mensaje con una emisión, las reglas RARP estipulan que sólo el servidor RARP puede generar una respuesta.

Sistema de nombres de dominio (DNS).

En lugar de usar la dirección IP de 32 bits completa, muchos sistemas adoptan nombres más significativos para sus dispositivos y redes. Los nombres por la general reflejan el nombre de la organización. Los nombres de los dispositivos individuales dentro de una red pueden variar desde nombres descriptivos en redes pequeñas hasta convenciones de denominación más compleja en redes más grandes. Traducir entre estos nombres y las direcciones IP sería imposible en una escala tan amplia como Internet.

Para solucionar el problema de los nombres de las redes, el NIC mantiene una lista de nombres de redes y las correspondientes direcciones gateway de la red. Este sistema creció de una lista de archivos planos hasta un sistema más complicado llamado DNS cuando las redes se volvieron demasiado numerosas para que funcionara con eficiencia el sistema de archivos planos.

DNS usa una arquitectura jerárquica muy parecida al sistema de archivos UNIX. El primer nivel de denominación divide a las redes en categorías de subredes. Debajo de cada una de estas hay otra división que identifica a la subred individual, por lo general una para cada organización. Esto se llama nombre de dominio y es único. El administrador del sistema de la organización puede dividir más las subredes de la compañía conforme se desee, con cada red llamada subdominio.

Hasta ahora se han establecido siete nombres de dominio de primer nivel por el NIC:

- .arp : Una identificación ARPANET - Internet
- .com : Compañía comercial
- .edu : Institución educativa
- .gov : Cualquier cuerpo gubernamental
- .mil : Militar
- .net : Redes usadas por proveedores de servicios Internet
- .org : Cualquiera que no quede dentro de ninguna de las otras categorías.

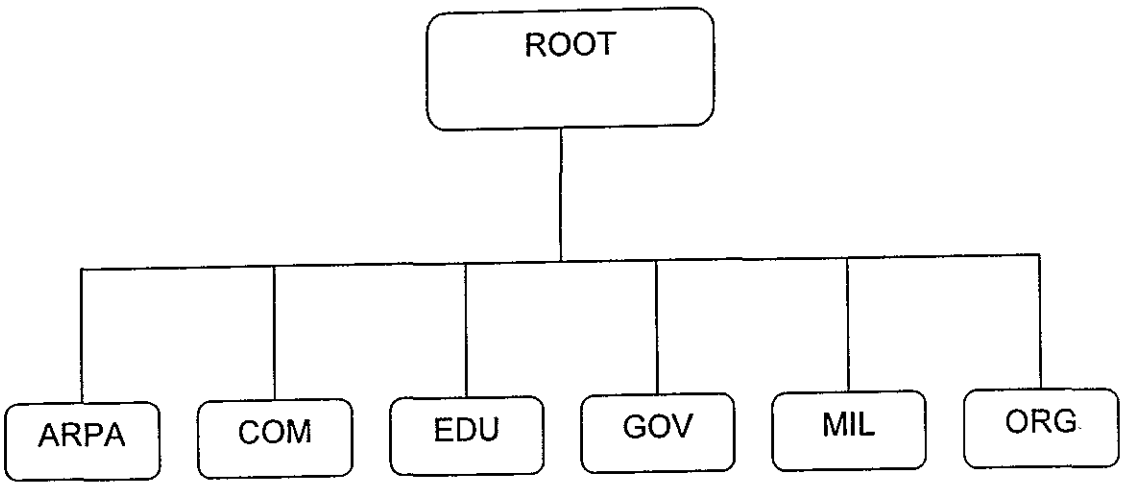


Figura 1.1.21. La estructura de dominios de Internet.

El NIC permite además, que se anexe un designador de país. DSN usa dos sistemas para establecer y rastrear los nombres de dominio. Un solucionador de nombres en cada red examina la información en un nombre de dominio. Si no puede encontrar la dirección IP completa, interroga a un servidor de nombres, el cual tiene la información completa del NIC. El solucionador de nombres trata de completar la información de direccionamiento usando su propia base de datos. Si un servidor de nombres no puede resolver la dirección, interroga a otro servidor de nombres y así sucesivamente, a través de toda la interred.

TELNET

El programa Telnet pretende proporcionar un registro remoto o capacidad de terminal virtual a través de una red. El servicio Telnet se proporciona por medio del puerto número 23 del TCP. El término Telnet se usa para referirse tanto al programa como al protocolo que proporciona estos servicios.

Cuando se comunican dos máquinas utilizando Telnet, este puede determinar y establecer los parámetros de comunicaciones y de terminal para la sesión durante la fase de conexión.

El protocolo Telnet incluye la capacidad de no soportar un servicio que no puede manejar un extremo de la conexión. Cuando una conexión se ha establecido por Telnet, ambos extremos acuerdan un método para que las dos máquinas intercambien información restándole carga a la CPU servidora para evitarle una gran cantidad de trabajo.

Por lo general, Telnet comprende un proceso en el servidor que acepta la llegada de solicitudes por una sesión Telnet. El cliente (el extremo que hace la llamada) ejecuta un programa, por lo general llamado telnet, que intenta la conexión con el servidor. Un pariente del programa telnet, es el programa rlogin, el cual es común en máquinas UNIX.

El protocolo Telnet usa el concepto de una network virtual terminal (terminal virtual de red) o NVT, para definir ambos extremos de una conexión Telnet. Cada extremo de la conexión tiene un teclado y una impresora lógicos. La impresora lógica puede desplegar caracteres y el teclado lógico puede generar caracteres. La impresora lógica es una pantalla terminal, en tanto que el teclado lógico es el del usuario. La figura 1.1.20 ilustra la NVT.

El protocolo Telnet trata a los dos extremos de la conexión como NTV. Los dos programas en cada extremo manejan la traducción de las terminales virtuales hacia los dispositivos físicos reales. El concepto de terminales virtuales permite a Telnet interconectar cualquier tipo de dispositivo, siempre que esté disponible un mapa de los códigos virtuales al dispositivo físico. Una ventaja de este enfoque es que algunos dispositivos físicos no pueden soportar todas las operaciones, de modo que la terminal virtual no tiene esos códigos. Cuando los dos extremos están estableciendo la conexión se nota la falta de estos códigos y, las secuencias que los usarían se ignoran. Este proceso es directo: un extremo pregunta si la función se soporta y el otro contesta, ya sea en forma positiva o negativa. Si se soporta se envían los códigos necesarios. De este modo se cubre con rapidez la lista de funciones soportadas. La figura 1.1.22 muestra una conexión Telnet.

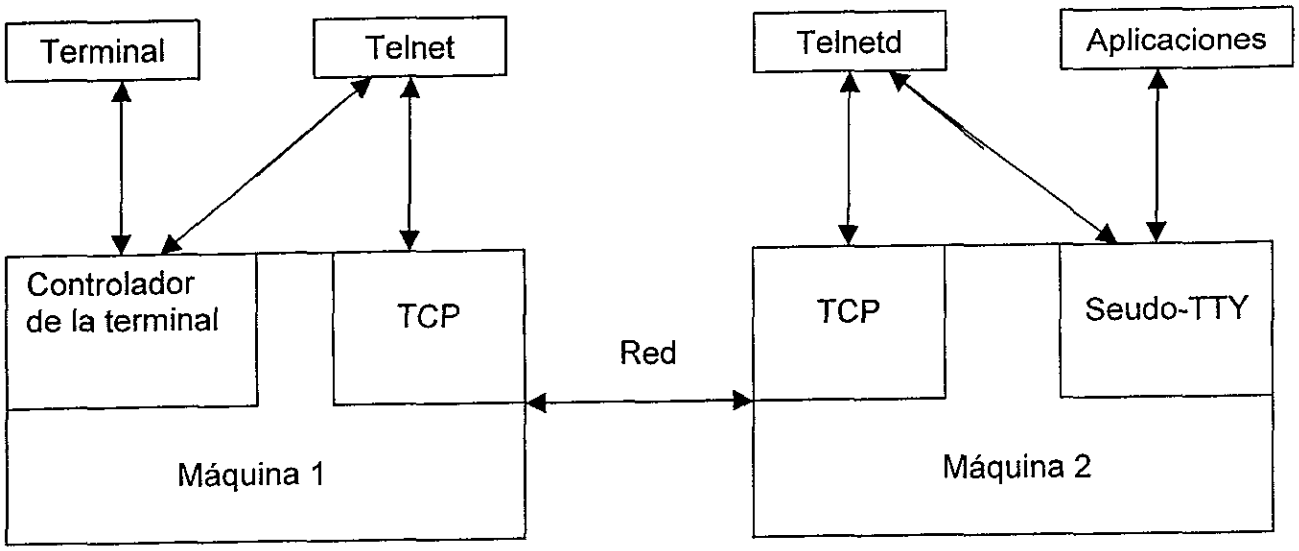


Figura 1.1.22. Una conexión Telnet.

Protocolo para transferencia de archivos (FTP).

El FTP es una utilería para manejar archivos a través de máquinas sin tener que establecer una sesión remota con Telnet, además permite administrar directorios y tener acceso al correo electrónico.

El FTP usa dos canales TCP. El puerto 20 del TCP es el canal de datos y el puerto 21 es el canal de comandos. El FTP observa el proceso de transferencia en tiempo real. AL usar el TCP, el FTP elimina la necesidad de preocuparse por la confiabilidad o la administración de la conexión, debido a que el FTP puede basarse en TCP para ejecutar estas funciones en forma apropiada.

En el lenguaje FTP, los dos canales que existen entre las dos máquinas se llaman protocol interpreter (intérprete de protocolo) o PI, y data transfer process (proceso de transferencia de datos) o DTP. El PI transfiere instrucciones entre las dos aplicaciones usando el canal 21 de comandos TCP y el DTP transfiere datos en el canal 20 de datos TCP.

Los comandos internos del protocolo FTP son secuencias ASCII de cuatro caracteres, terminadas por un carácter de línea nueva. Algunos de los códigos requieren

parámetros después de ellos. Una ventaja principal de usar caracteres ASCII para los comandos es que un usuario puede observar el flujo del comando y entenderlo con facilidad.

Los comandos FTP usados por el protocolo mantienen el proceso de conexión, revisión de contraseña y la transferencia de archivos entre sí. El FTP también usa códigos de retorno sencillos para indicar las condiciones de la transferencia. Cada código de retorno es un número de tres dígitos, el primero de los cuales significa una ejecución exitosa (1, 2 o 3) o una falla (4 o 5). El segundo y tercer digito especifican el código de retorno o condición de error con más detalle.

El FTP permite transferir archivos en varios formatos, los cuales por lo general dependen del sistema. Algunas instalaciones de mainframe agregan soporte para el EBCDIC, en tanto que muchos sitios tienen un tipo local diseñado para transferencias rápidas entre máquinas de red locales. Las transferencias de texto usan caracteres ASCII separados por caracteres de retorno de carro y de línea nueva, en tanto que el binario permite transferir caracteres sin conversión o formato. El modo binario es más rápido que el de texto y, además, permite la transferencia de todos los valores ASCII.

Cuando se establece una conexión por el FTP el proceso es el siguiente:

1. Registro: verifica la ID del usuario y la contraseña.
2. Definir directorio: identifica el directorio de inicio.
3. Definir el modo de transferencia de archivos: define el tipo de transferencia.
4. Iniciar transferencia de datos: permite los comandos de usuario.
5. Detener transferencia de datos: cierra la conexión.

Estos pasos se ejecutan en secuencia para cada conexión. Un usuario tiene varios comandos disponibles para controlar el FTP. La figura 1.1.23 muestra una conexión del canal FTP.

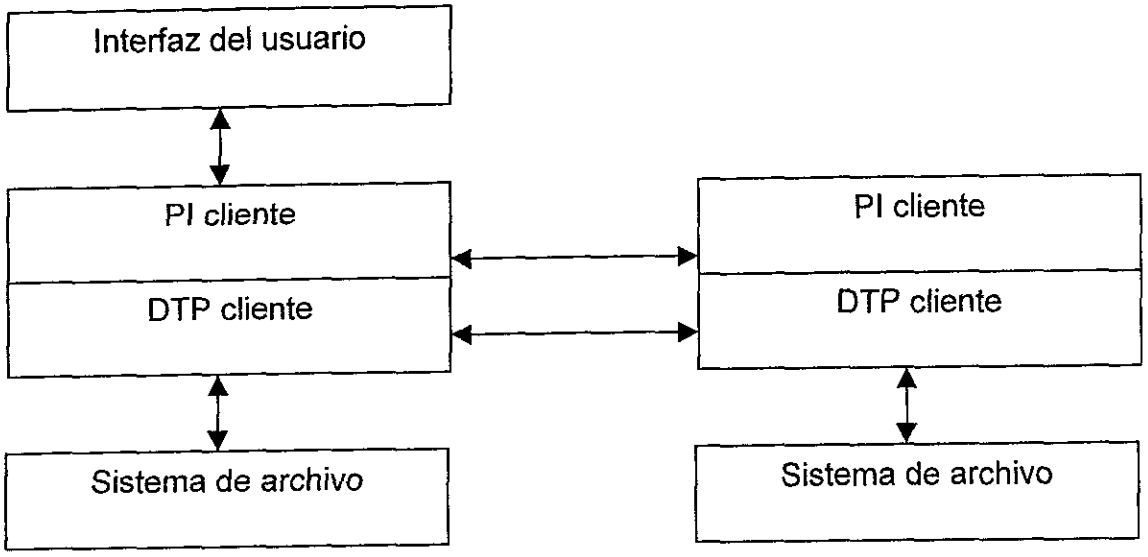


Figura 1.1.23. Conexiones del canal FTP.

Protocolo trivial para transferencia de archivos (TFTP).

El TFTP es uno de los protocolos de transferencia de archivos más sencillo. Diferente del FTP de dos maneras principales: no se registra en la máquina remota y usa un protocolo de transporte sin conexión UDP en lugar del TCP. Al usar UDP el TFTP no supervisa el progreso de la transferencia de archivos, aunque tiene que emplear algoritmos más complejos para asegurar la integridad de los datos. Al evitar el registro remoto se evita los problemas de autorización de acceso y de archivo. El TFTP usa el identificador de puerto TCP número 69, aun cuando el TCP no participa en el protocolo.

El TFTP tiene pocas ventajas sobre el FTP. Por lo general, no usa transferencia de archivos entre máquinas donde se podría usar el FTP en su lugar, aunque el TFTP es útil cuando comprende una terminal o estación de trabajo sin disco. De manera común, el TFTP se usa para cargar aplicaciones y fuentes en estas máquinas, así como para el arranque.

El TFTP es necesario en estos casos debido a que las máquinas sin disco no pueden ejecutar el FTP hasta que están cargadas por completo con un sistema operativo. Los

requisitos de tamaño de ejecución y memoria pequeños del TFTP lo hacen ideal para incluirlo en el arranque, donde el sistema sólo requiere el TFTP, el UDP y un controlador de red, todo lo cual puede proporcionarse en un EPROM pequeño.

El TFTP maneja las autorizaciones de acceso y archivo imponiendo restricciones por sí mismo. Debido a las reglamentaciones de acceso vagas, la mayor parte de los administradores del sistema imponen más control sobre el TFTP; de otra manera es bastante fácil para un usuario tener acceso a archivos o transferir archivos que constituye una violación de seguridad.

Las transferencias TFTP pueden fallar por varias razones, debido a que prácticamente cualquier clase de error encontrado durante una operación de transferencia origina una falla completa. El TFTP soporta algunos mensajes de error básicos, pero no puede manejar errores sencillos como recursos insuficientes para una transferencia de archivos o incluso una falla para localizar un archivo solicitado.

El conjunto de comandos TFTP se parece al del FTP, pero difieren en varios aspectos importantes debido al aspecto de conexión del protocolo. El más notable es el comando connect, el cual simplemente determina la dirección remota en lugar de iniciar conexión.

El TFTP permite transferencias de texto y binarias, como el FTP. Como ocurre con Telnet y FTP, el TFTP usa un proceso servidor y uno ejecutable, por lo general llamado tftp.

Código	Descripción
0	No definido
1	Archivo no encontrado
2	Las autorizaciones impiden el acceso
3	Disco lleno o límite de autorización excedido
4	Operación TFTP ilegal solicitada
5	Número de transferencia desconocido

Tabla 1.1.5. Códigos y mensajes de error del TFPT.

Protocolo simple de transferencia de correo (SMTP).

El SMTP es el método Internet definido para transferir correo electrónico. El SMTP se parece al FTP en muchas formas, incluyendo la misma sencillez de operación. El SMTP usa el puerto TCP 25.

El programa sendmail, por ejemplo, actúa como un cliente y como un servidor, por lo general ejecutándose en segundo plano como un daemon. Los usuarios no interactúan de manera directa con sendmail sino que usan un programa de correo de front-end como mail, mailx o Mail. Estas interfaces de sistema de correo pasan el mensaje a sendmail para su envío.

El SMTP usa spoolers o colas. Cuando un mensaje se envía al SMTP, lo coloca en una cola. El SMTP intenta enviar el mensaje desde la cola siempre que se conecta con máquinas remotas. Si no puede enviar el mensaje dentro de un límite de tiempo especificado, el mensaje se regresa al transmisor o se elimina.

Las transmisiones de datos SMTP usan un formato sencillo. Todo el texto del mensaje se transfiere como caracteres ASCII de 7 bits. El fin del mensaje lo indica un solo punto en una línea. Si por alguna razón una línea en el mensaje comienza con un punto, el protocolo agrega un segundo punto para evitar confusión con el indicador de fin de mensaje.

Cuando se establece una conexión, los dos sistemas SMTP establecen códigos de autentificación. Después de esto, un sistema envía un comando mail al otro para identificar al transmisor y proporcionar información acerca del mensaje. El SMTP regresa un acuse de recibo, después del cual envía un RCPT para identificar al receptor. Si se identifica más de un receptor en la ubicación del receptor, se envían varios mensajes RCPT, pero el mensaje en sí sólo se transmite una vez. Después de cada RCPT hay un acuse de recibo. A un comando DATA le siguen las líneas de mensaje hasta que un solo punto en una línea indica el fin del mensaje. La conexión se cierra con el comando QUIT.

COMANDO	Descripción
DATA	Texto del mensaje
EXPN	Expansión de una lista de distribución
HELO	Se usa en el establecimiento de la conexión para intercambiar identificadores
HELP	Solicita ayuda
MAIL	La dirección del transmisor
NOOP	Ninguna operación
RCPT	La dirección de destino del mensaje (puede proporcionarse más de una)
RSET	Termina la transacción actual
SAML	Envía un mensaje a la terminal de usuario y envía correo
SEND	Envía un mensaje a la terminal de usuario
SOML	Envía un mensaje a la terminal de usuario o envía correo
TURN	Cambia la dirección de envío (invierte los papeles de transmisor y receptor)
VRFY	Verifica el nombre de usuario

Tabla 1.1.6. Conjunto de comandos del protocolo SMTP-

Protocolo BOOTP.

BOOTP utiliza UDP para permitir que una máquina sin disco determine su dirección IP sin necesidad de utilizar RARP. BOOTP supera algunos de los problemas de RARP. RAPR requiere de acceso directo al hardware de la red, lo cual puede ocasionar problemas cuando se está tratando con servidores. Asimismo, RARP suministra sólo una dirección IP. Cuando deben enviarse paquetes grandes, esto ocupa una gran cantidad de espacio que podría utilizarse para información valiosa.

BOOTP fue elaborado para utilizar UDP y requiere de sólo un paquete de información para brindar toda la que se requiera, para que comience a operar una estación de trabajo sin disco.

Para determinar la dirección IP de una estación de trabajo sin disco, BOOTP utiliza las capacidades de transmisión del IP. Esto permite que la estación de trabajo envíe un mensaje aun cuando no conozca la dirección de la máquina de destino o incluso la suya.

BOOTP coloca todas las tareas de comunicaciones en la estación de trabajo sin disco. Éste especifica que todos los mensajes UDP enviados por la red utilicen sumas de verificación y que este establecido el bit no fragmentar. Esto tiende a reducir el número de datagramas perdidos, mal interpretados o duplicados.

Para manejar la perdida de un mensaje, BOOTP utiliza temporizadores. Cuando se ha enviado un mensaje, arranca un temporizador. Si no se recibe respuesta cuando se termina el tiempo, se envía de nuevo el mensaje. El protocolo estipula que el temporizador se coloque en un valor aleatorio que se incrementa cada vez que vence el tiempo, hasta que alcanza un valor máximo, después de lo cual vuelve a ser aleatorio.

BOOTP utiliza los términos cliente servidor para hacer referencia a las máquinas. El cliente es la máquina que inicia una consulta y el servidor es la máquina que responde a esa consulta. Los mensajes BOOTP se mantienen en formatos fijos por sencillez y, para permitir que el software BOOTP se ajuste a un pequeño espacio dentro de un PROM.

Protocolo de tiempo para red (NTP).

El protocolo que mantiene los estándares de tiempo se llama NTP y puede utilizar tanto TCP como UDP; el puerto 37 está dedicado a este protocolo.

La operación del NTP se basa en la obtención de un tiempo exacto a partir de una consulta hecha a un servidor de tiempo primario, el cual obtiene su propia información de tiempo a partir de una fuente de tiempo estándar. El servidor de tiempo consulta el reloj estándar y ajusta sus propios tiempos con el estándar.

Una vez que el servidor de tiempo primario tiene uno exacto, envía mensajes NTP a los servidores secundarios que utilizan NTP; aunque la exactitud se pierde con cada comunicación, debido al estado latente de las redes. Estos mensajes de tiempo pueden enviarse a los gateways y a las máquinas individuales que se encuentren dentro de la red, si el administrador así lo decide. Generalmente, cada red tiene por lo menos un servidor de tiempo primario y uno secundario, aunque las redes extensas pueden tener varios de cada uno.

El formato de los mensajes NTP es sencillo. Se utilizan varios campos de control para los procedimientos de sincronización y actualización. La figura 1.1.24 muestra el formato del mensaje NTP.

Campos de control
Sincronizar la distancia con el primario
ID de la red para el primario
Reloj de tiempo local actualizado
Originando pulsos de tiempo
Recibiendo pulsos de tiempo
Transmitiendo pulsos de tiempo
Autentificación

Figura 1.1.24. El formato del mensaje NTP.

Buscador Internet de paquetes (PING).

PING se usa para consultar otro sistema y asegurar que una conexión todavía está activa. El comando PING está disponible en la mayor parte de los sistemas operativos que aplican el TCP/IP.

El programa PING opera enviando una solicitud de reflejo del ICMP. Si el software IP de la máquina de destino recibe la solicitud ICMP, emite de inmediato una respuesta de reflejo. La máquina transmisora continúa enviando una solicitud de reflejo hasta que el programa PING termina, PING despliega una serie de estadísticas.

Usar PING para enviar paquetes grandes es un método para determinar el comportamiento de la red con tamaños de paquetes grandes, en especial cuando debe ocurrir fragmentación. El programa PING es útil también para supervisar los tiempos de respuesta de la red, observando el tiempo de respuesta en paquetes enviados conforme cambia la carga de la red. Esta información podría ser muy útil para la optimización del TCP/IP.

El programa PING es útil para diagnóstico debido a que proporciona cuatro piezas importantes de información: si el software TCP/IP está funcionando en forma correcta, si un dispositivo de la red local puede alcanzarse, si se puede tener acceso a una máquina remota y la verificación del software en la máquina remota.

Protocolo simple para manejo de redes (SNMP)

SNMP no es un protocolo, sino tres que juntos forman una familia; todos diseñados para trabajar en pro de las metas de la administración. Los protocolos que conforman la familia SNMP y sus papeles se muestran a continuación:

- Base de información de la administración (MIB): una base de datos que contiene información del estado.

- Estructura e identificación de la información sobre la administración (SMI): una especificación que define las entradas en una MIB.
- Protocolo simple para la administración de red (SNMP): el método de comunicación entre los dispositivos administrados y los servidores.

Los dispositivos administrados por el SNMP se comunican con el software servidor SNMP que está localizado en cualquier parte de la red. El dispositivo habla con el servidor de dos formas: por sondeo o interrupción. Un dispositivo sondeado hace que el servidor se comunique con el dispositivo, preguntándole sobre su condición o estadísticas actuales. Un sistema SNMP de interrupción hace que el dispositivo administrado envie mensajes al servidor cuando algunas condiciones lo garanticen. De esta forma el servidor conoce inmediatamente cualquier problema.

Cada dispositivo administrado por SNMP mantiene una base de datos que contiene estadísticas y otro tipo de información. Esta base de datos se llama MIB. Las entradas MIB tienen cuatro datos: un tipo de objeto, una sintaxis, un campo de acceso y un campo de estado. Las entradas MIB generalmente las estandarizan los protocolos y siguen reglas estrictas para el formateo.

El tipo de objeto es el nombre de la entrada específica. La sintaxis es el tipo de valor. El campo de acceso se usa para definir el nivel de acceso de la entrada, comunmente está definida por valores de sólo lectura, lectura-escritura, solo escritura y no accesible. El campo de estado contiene un indicación de que la entrada de la MIB es obligatoria, opcional u obsoleta.

Sistema de archivos de red (NFS)

El protocolo NFS comprende varios protocolos, de los cuales sólo uno se llama protocolo NFS. Los protocolos del producto NFS están diseñados como un conjunto de capas, similares al modelo OSI. Las capas del producto NFS pueden compararse con las del modelo OSI en la figura 1.1.25.

El producto NFS se basa en el modelo de capas OSI, dando como resultado protocolos que son independientes entre sí y protocolos que se encuentran en diferentes capas.

El protocolo NFS se compone de un conjunto de procedimientos RPC. Éste no es un protocolo en el sentido convencional de definir un complejo y desesperante proceso que se lleva acabo entre dos máquinas. Más bien es un método de comunicar información acerca del procedimiento que va a correr. NFS utiliza UDP y tiene un número de puerto asignado el 2049.

NFS fue diseñado para ser un protocolo sin estado, lo que significa que las máquinas que utilizan NFS no tendrían que mantener las tablas de estado para utilizar el protocolo. Asimismo, se diseño para ser consistente, lo que quiere decir que después de una falla el sistema podría recobrarse fácil y rápidamente.

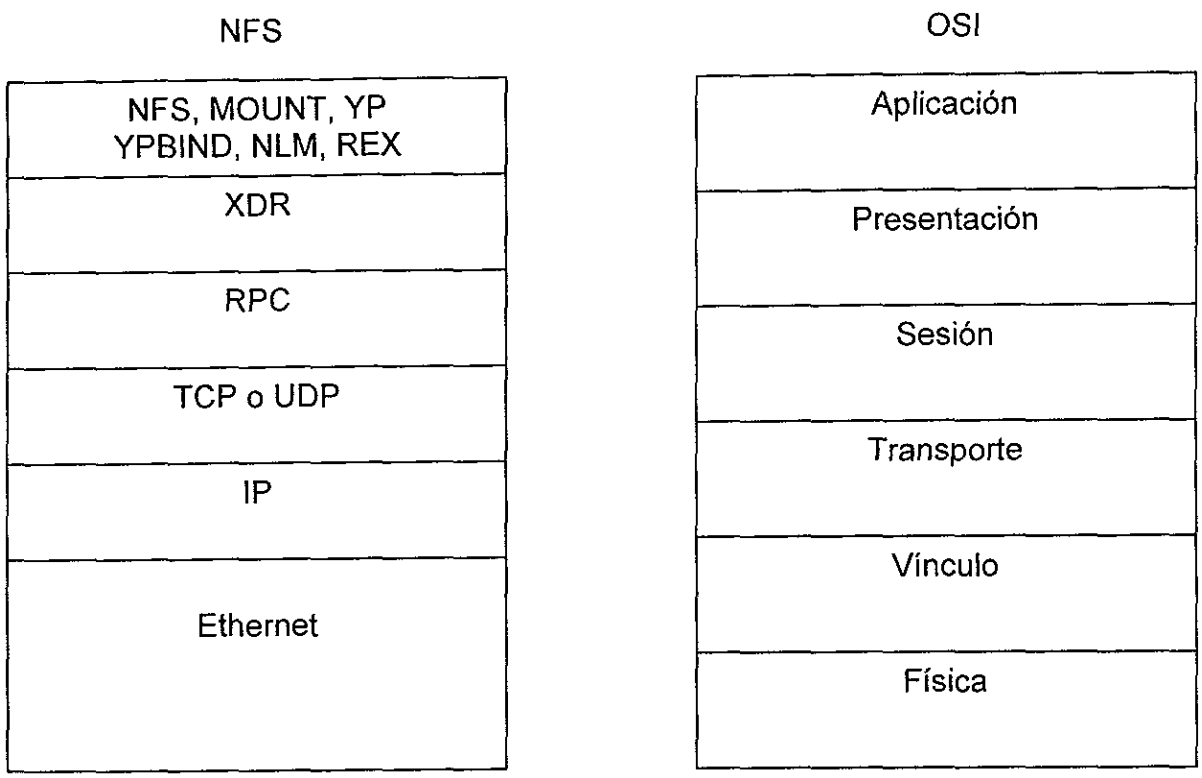


Figura 1.1.25. Las capas del protocolo de NFS.

NFS define un conjunto de constantes que se utilizan para establecer varios parámetros, como el número de bytes que hay en el nombre de una ruta de acceso, el número máximo de bytes de una solicitud de lectura o escritura o el tamaño de un puntero NFS. Éstas se llaman constantes del protocolo y deben ser las mismas para todas las aplicaciones de NFS.

Varios objetos de datos los utiliza el NFS para definir los archivos y sus atributos. Dado que NFS trata específicamente con archivos, estos objetos son importantes para el protocolo. Un objeto de datos es un manipulador de archivos, el cual únicamente identifica un archivo del servidor. Los manipuladores del archivo se proporcionan en todos los mensajes NFS que hacen referencia al archivo. Al igual que la mayor parte de los tipos de datos NFS, el manipulador del archivo es un campo de 32 bytes de formato libre que entiende el servidor.

Un objeto de datos se utiliza para el tipo de archivo, que define todas las clases de archivo conocidas por el NFS. Éstas imitan a los tipos de archivos UNIX, incluyendo un archivo regular, un directorio, los vínculos y los archivos, tanto en modo de bloques como de caracteres.

También se utiliza una estructura de datos para los atributos de los archivos, la cual define los permisos del archivo, los tiempos de acceso, el propietario y otros parámetros. Esto es necesario cada vez que se realiza una lectura o escritura de un archivo, puesto que los atributos deben ser los correctos para permitir que el procedimiento continúe.

Estos objetos de datos pueden combinarse dentro de una entidad más grande utilizando una unión discriminatoria. Una unión discriminatoria es una combinación de varios objetos de datos a los que se les da una sola etiqueta. Este tipo de estructura se utiliza para simplificar la programación.

Llamada de procedimiento remoto (RPC).

El RPC actúa como la capa de sesión y como el intercambiador de mensajes para todas las aplicaciones basadas en NFS. RPC está compuesto por un conjunto de procedimientos que pueden incorporarse dentro de las aplicaciones de alto nivel para manejar cualquier tipo de acceso a la red requerido.

Los desarrollares de aplicaciones pueden crear sus propios procedimientos RPC entre un cliente y un servidor. Un grupo de procedimientos se llama servicio. Cada servidor puede utilizar un solo servicio, por lo que a cada servicio se le asigna un número de programa para identificarse a sí mismo, tanto ante el cliente como ante el servidor.

RPC funciona sobre la red que se encuentra entre un cliente y un servidor. Éste comienza con la activación del procedimiento por parte del cliente, desde el cual se envía un mensaje de solicitud hacia el servidor. Después de recibir el mensaje y de extraer la solicitud, el servidor ejecuta el procedimiento solicitado y sobrepone un mensaje de respuesta con los resultados. Al recibir la respuesta, el cliente quita el mensaje y continua con la ejecución normal de la aplicación. Cada paso lo controlan las rutinas que se encuentran en el interior de la biblioteca RPC.

Los mensajes de RPC pueden enviarse utilizando TCP o UDP. Comúnmente, RPC se utiliza con UDP, debido a que el protocolo basado en la conexión no es necesario y a que UDP por lo general es más rápido. Sin embargo, UDP impone un tamaño máximo para el paquete, lo cual puede ocasionar algunos problemas con los procedimientos. Asimismo, UDP no garantiza la entrega, por lo que una aplicación que utiliza UDP debe manejar aspectos de confiabilidad. La utilización de TCP ofrece la posibilidad de no sólo ignorar las cuestiones de confiabilidad, sino también de agrupar las solicitudes. Con una conexión en grupo, el cliente y el servidor acuerdan que el cliente pueda enviar varias solicitudes RPC una después de la otra, sin tener que esperar un reconocimiento o una respuesta para cada una.

El protocolo RPC se utiliza para enviar solicitudes y respuestas. El formato del encabezado del paquete del protocolo RPC contiene campos codificados en el formato de representación de datos externos o XDR. El campo ID de la transacción se utiliza para combinar las solicitudes y las contestaciones y lo modifica el cliente con cada nueva solicitud. El campo indicador de dirección muestra el mensaje originado en el cliente (un valor 0) o en el servidor (un valor 1). El primer número de versión es la versión de RPC que se utiliza y el segundo número de versión identifica la versión de programa. El número de programa identifica el procedimiento particular del servicio.
Algunos procedimientos pueden requerir que un cliente se autentifique así mismo frente al servidor, tanto para propósitos de identificación como por razones de seguridad. El encabezado para el protocolo contiene dos campos para la autentificación. El campo de información de la autorización es para su propia información y el campo de verificación de la autorización se utiliza para la validación. La figura 1.1.26 muestra la ejecución de un RPC.

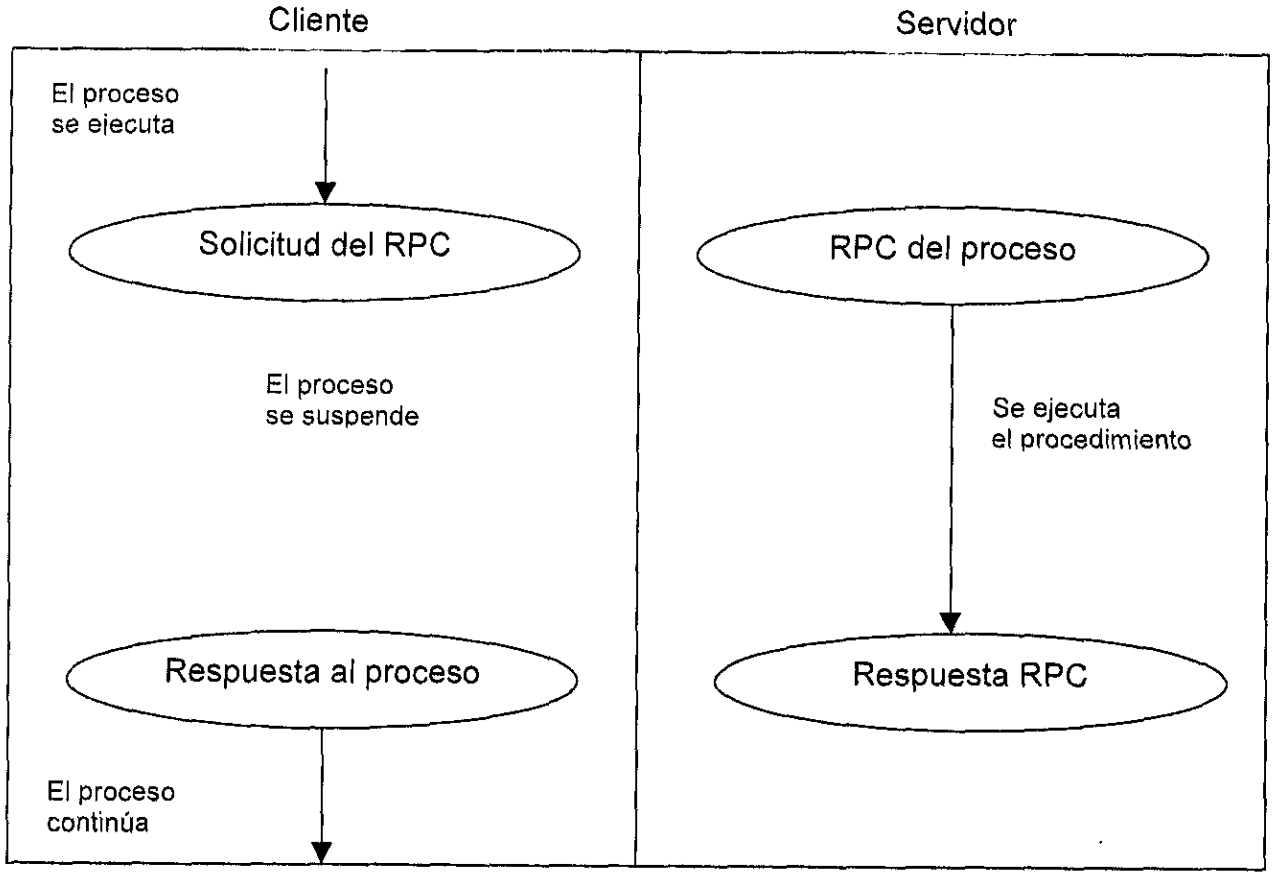


Figura 1.1.26. Ejecución de un RPC.

Servicio de información para red (NIS)

El protocolo Yellow Pages (YP) es un servicio de capa de aplicación que brinda un servicio de directorio versátil. Debido a restricciones de derechos de autor este fue renombrado como NIS. NIS fue elaborado por varias razones pero la que afecta principalmente a los usuarios es la de los permisos de acceso. El efecto que tienen estos permisos en los usuarios generalmente es transparente, excepto por una ventaja principal.

Si un usuario de una red extensa suele conectarse a otras máquinas, debe mantener una cuenta en cada máquina a la que se conecte. De tal forma, necesitaría cuentas de usuario en cada máquina a la que concebiblemente quiera tener acceso. Mantener las claves de acceso en un gran número de máquinas es complicado, puesto que debe registrarse en cada una y realizar cambios a las claves de acceso. NIS se elaboró para permitir que un solo archivo central de usuario pueda compartirse en toda la red, requiriendo únicamente una sola entrada para permitir el acceso a todas las máquinas (a menos que se impongan restricciones específicas), y simplificar a un solo paso el cambio de la clave de acceso en todas las máquinas.

En términos de RPC, esta combinación de ID de usuario y clave de acceso trabaja sobre los procedimientos de autentificación de RPC. Para garantizar el acceso a los archivos RPC utiliza las ID del grupo y del usuario, por lo que es necesario que coincidan las ID del usuario y del grupo para el cliente y el servidor. Sin NIS esto sería muy difícil de aplicar, puesto que el archivo de un usuario en cada máquina podría tener los mismos nombres, pero sus ID de usuario podrían no coincidir. Lo que es peor, un usuario que se encuentre en otra máquina y que tenga una ID de usuario que coincida podría tener acceso a los archivos de su máquina como si estuviera registrado.

NIS es un sistema de acceso distribuido en el cual cada máquina de la red que utilice NIS tiene acceso a un servidor central, llamado NIS maestro o verificador (dependiendo de la versión), para obtener información sobre el acceso. En las redes grandes, para repartir la carga, y en todas las redes como una contingencia de respaldo, se designan varias máquinas más como esclavas o ypslaves y mantienen actualizada la información del acceso. En caso de falla del servidor maestro, una esclava asume las funciones. NIS utiliza tanto TCP como UDP para las comunicaciones.

El protocolo NIS tiene un conjunto de procedimientos definidos dentro de la RPC. Éstos permiten las funciones de búsqueda de servidores maestros, acceso a los archivos del usuario y administración del sistema. Se utiliza otro procedimiento para transferir copias de los archivos maestros. Varias máquinas están agrupadas en la subred NFS, llamada dominio Internet. Cada dominio tiene una máquina maestra y una esclava.

NIS mantiene la información del acceso en un conjunto de mapas, cada mapa corresponde a un área o dominio especifico de una red. Esto permite que varios grupos utilicen el mismo NIS maestro, pero con diferentes permisos de acceso. Los mapas NIS no tienen que corresponder con los dominios DSN, permitiendo mayor versatilidad en la configuración. Los mapas consisten en un conjunto de registros que están en formato ASCII, cada uno con una clave de índice para una búsqueda rápida.

NIS no está restringido a los usuarios del sistema. Cualquier archivo puede configurarse para utilizar NIS, como la lista de máquinas de una red. De esta forma, sólo tiene que hacer un cambio a estos archivos para cualquier red. Un conjunto de alias también puede administrarlo NIS.

Protocolo de control de transmisión (TCP)

El TCP proporciona una cantidad considerable de servicios a la capa IP y a las capas superiores. Proporciona un protocolo orientado hacia la conexión para las capas superiores, que permite a una aplicación estar segura de que un datagrama enviado por la red fue recibido íntegro. En este papel, el TCP actúa como un protocolo de validación del mensaje que proporciona comunicaciones confiables.

El TCP maneja el flujo de datagramas desde las capas superiores hasta la capa IP, así como los datagramas que llegan desde la capa IP hacia los protocolos de niveles superiores. El TCP tiene que asegurar que las prioridades y la seguridad se respeten de manera apropiada. El TCP debe ser capaz de manejar la terminación de una aplicación superior, que estaba esperando la llegada de datagramas, al igual que fallas en las capas inferiores. El TCP también debe mantener una tabla de estado de todas las series de datos que entran y salen de la capa TCP. El aislamiento de todos estos servicios en una capa separada permite que las aplicaciones se diseñen sin importar el control del flujo o la confiabilidad del mensaje. Sin la capa TCP, cada aplicación tendría que aplicar los servicios por sí misma, lo cual es un desperdicio de recursos.

Debido a que el TCP es un protocolo orientado hacia la conexión responsable de asegurar la transferencia de un datagrama, desde la máquina fuente hasta la máquina de destino, el TCP debe recibir mensajes de comunicaciones desde la máquina de destino para acusar recibo del datagrama.

El TCP debe comunicarse con aplicaciones en la capa superior y con un sistema de red en la capa de abajo. Varios mensajes están definidos para el protocolo de capa superior para comunicaciones TCP, pero no existe un método definido para que el TCP hable con las capas inferiores. El TCP espera que la capa de abajo de él defina el método de comunicación. Por lo general se supone que el TCP y la capa de transporte se comunican de manera asincróna.

El método de comunicación del TCP con el protocolo de capa superior está bien definido, consistiendo en un conjunto de primitivos de solicitud de servicio.

El TCP también debe comunicarse con otras implementaciones TCP a través de las redes. Para hacer esto, usa unidades de datos de protocolo o PDU, las cuales se llaman segmentos en el lenguaje TCP.

El diseño de las TCP PDU se muestra en la figura 1.1.27.

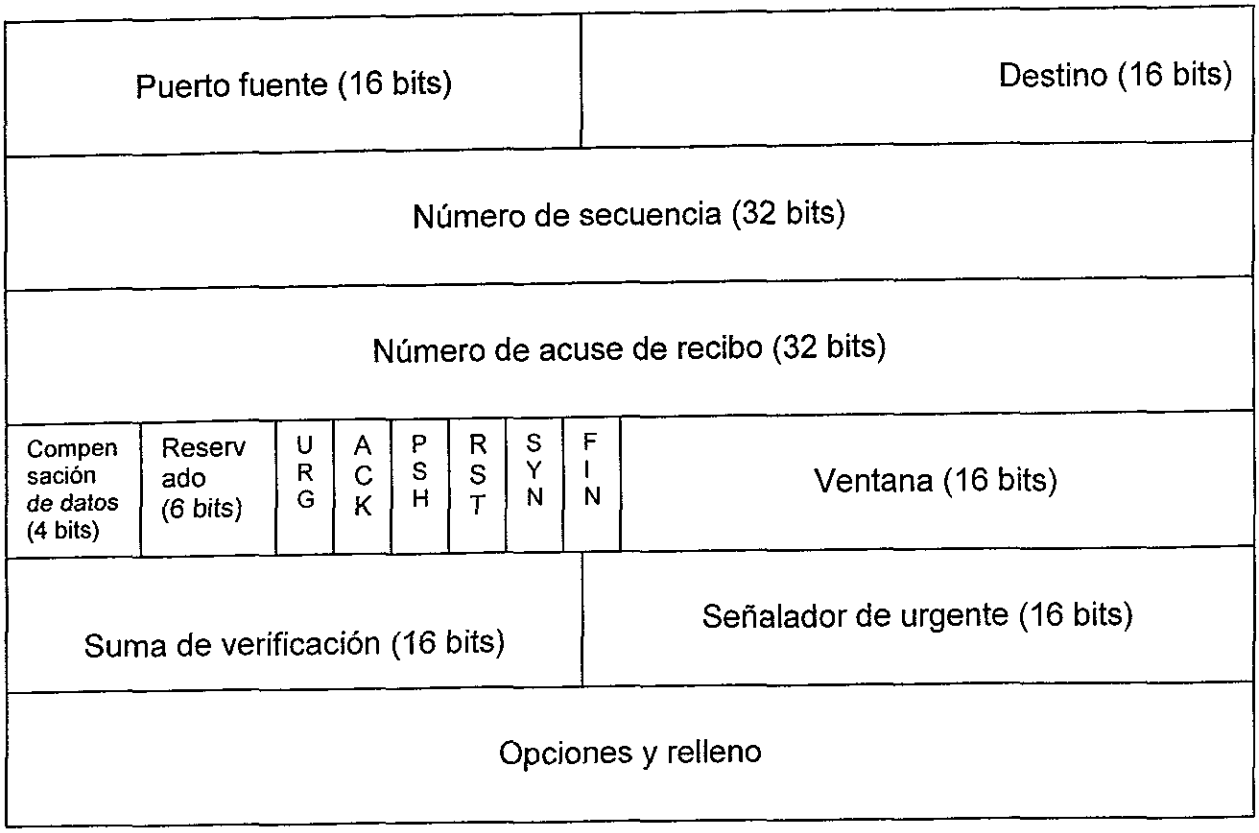


Figura 1.1.27. La TCP PDU.

Los diferentes campos son:

- Puerto fuente: un campo de 16 bits que identifica al usuario TCP local.
- Puerto destino: un campo de 16 bits que identifica al usuario TCP de la máquina remota.
- *Número de secuencia*: un número que indica la posición del bloque actual en el mensaje total. Este número se usa también entre dos implementaciones TCP para proporcionar el número de secuencia de envío inicial.
- Número de acuse de recibo: un número que indica el siguiente número de secuencia esperado.

- Compensación de datos: el número de palabras de 32 bits que están en el encabezado TCP. Este campo se usa para identificar el inicio del campo de datos.
- Reservado: un campo de 6 bits reservado para uso futuro. Los 6 bits deben fijarse en 0.
- Bandera Urg: si está activa (un valor de 1), indica que el campo del señalador urgente es significativo.
- Bandera Ack: si está activa, indica que el campo acuse de recibo es significativo.
- Bandera Psh: si está activa, indica que la función push debe ejecutarse.
- Bandera Rst: si está activa, indica que la conexión debe reiniciarse.
- Bandera Syn: si está activa, indica que los números de secuencia deben sincronizarse. Esta bandera se usa cuando se está estableciendo una conexión.
- Bandera Fin: si está activa, indica que el transmisor no tiene más datos que enviar. Éste es el equivalente de un marcador de fin de la transmisión.
- Ventana: un número que indica cuántos bloques de datos puede aceptar la máquina receptora.
- Suma de verificación: Calculada tomando el complemento de uno de 16 bits, de la suma del complemento de uno de las palabras de 16 bits en el encabezado y el texto juntos.
- Señalador urgente: usado si se estableció la bandera urg; indica la parte del mensaje de datos que es urgente al especificar la compensación del número de secuencia en el encabezado.
- Opciones: Similar al campo opciones del encabezado IP, éste se usa para especificar opciones del TCP. Cada opción consta de un número de opción (un byte), el número de bytes en ésta y los valores de la opción. Están definidas tres opciones para el TCP.

  0 Fin de la lista de opciones

  1 No-operación

  2 Tamaño máximo del segmento
- Relleno: Rellenado para asegurar que el encabezado es un múltiplo de 32 bits.

1.2. Características, Ventajas y Desventajas del C++.

1.2.1. Introducción.

En este capítulo, la intención es mostrar y discutir algunas bondades y limitantes de un lenguaje de programación que hoy por hoy tiene una aceptación muy amplia, ya sea dentro de el ámbito profesional, para usarse dentro del desarrollo de sistemas de información, o bien, dentro del ámbito académico.

Comenzaremos por mostrar el lugar que ocupa C++ en la historia de los desarrollos de los lenguajes, así como también los dos diferentes tipos de lenguajes de programación Orientados a Objetos, para de esta manera ubicar a C++ entre ellos. Posteriormente se expondrá una visión general de la POO (**Programación Orientada a Objetos**) y las características ventajas y desventajas que son inherentes a C++. Y como punto final el estilo de su sintaxis, así como la interpretación y utilización de los conceptos básicos de la Orientación a Objetos tomado a C++ como referencia.

1.2.1.1. Antecedente Histórico.

Smalltalk es conocido como el primer lenguaje que incluyó conceptos de la programación con OO (**Orientación a Objetos**), lo que no es tan cierto. El abuelo de los lenguajes OO es Simula, el cual es acrónico de "Lenguaje de Simulación". Simula fue desarrollado en Noruega durante la década de los sesentas, para resolver complejos problemas de simulación y modelaje.

A partir de Simula empezó la evolución de los lenguajes de programación con OO. Estos lenguajes existen desde hace 25 años aproximadamente (figura 1.2.1).

Smallltalk fue un buen descendiente de Simula; siendo realmente el primer lenguaje con OO reconocido como tal, desarrollado en el centro de investigaciones de Xerox en Palo Alto (PARC) en la década de los setentas.

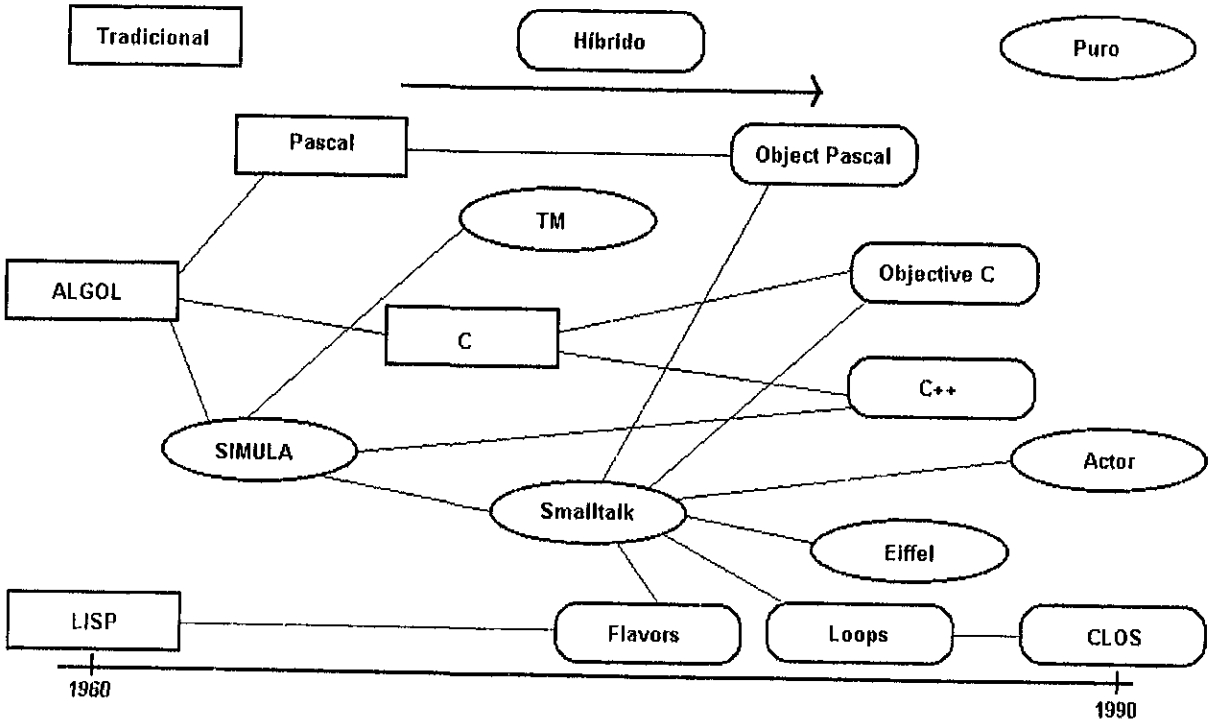


Figura 1.2.1. Evolución de los lenguajes con Orientación a Objetos.

A partir de Smalltalk surgieron muchos otros lenguajes puros, como por ejemplo Eiffel y Actor. Esto también motivó la adición de características de los objetos a una variedad de lenguajes tradicionales como LISP, C y Pascal.

1.2.1.2. Lenguajes Puros y Lenguajes Híbridos.

Desde la aparición de Simula hace 25 años, empezó el desarrollo de los lenguajes de programación Orientada a Objetos, los que han evolucionado de tal manera que hoy podemos identificar dos tendencias: puros e híbridos. C++ es un lenguaje híbrido que actualmente cuenta con una gran aceptación tanto en el ámbito académico como el desarrollo de aplicaciones.

Existen básicamente dos diferentes tipos de lenguajes OO: puros e híbridos. Un lenguaje puro es construido a partir de agrupar todas las características de una orientación a objetos. Cualquier cosa, en él, consiste de clases, objetos y métodos.

Por otra parte, un lenguaje híbrido se construye con base en un lenguaje de programación existente, tal como LISP, C o Pascal, teniendo las características de ser funcional, procedural o estructurado, *más las características* adicionales de objetos (figura 1.2.2).

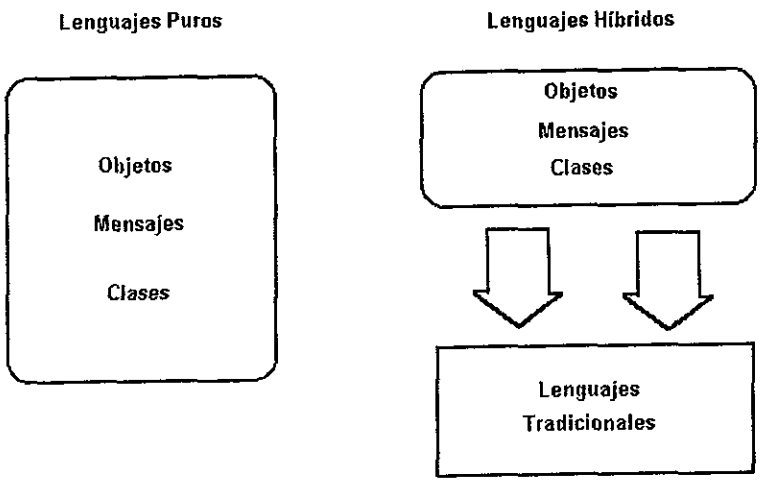


Figura 1.2.2. Lenguajes puros y lenguajes híbridos.

Tanto los lenguajes puros como los híbridos tienen pros y contras, de la comparación entre estos dos tipos diferentes, podemos concluir que:

- En los lenguajes híbridos no sólo se tiene la posibilidad de hacer una abstracción de los datos por medio de objetos y clases, sino que también podemos pensar en apuntadores o cualquier otro concepto heredado de los lenguajes tradicionales. Por otra parte, en los lenguajes puros estamos restringidos a manejar la información a través de objetos, clases y mensajes.
- Debido al uso de un intérprete, el desempeño de los lenguajes puros es menor que el de los lenguajes híbridos.
- Los lenguajes puros e híbridos también difieren en la facilidad de aprendizaje.

1.2.2. La Programación Orientada a Objetos.

La POO (**Programación Orientada a Objetos**), es una nueva forma de pensar cómo resolver algunos de los problemas planteados en el mundo de la Informática. En lugar de tratar de adaptar el problema a algún aspecto familiar al computador, el computador se adapta al problema. En la POO, se examina el problema en "entidades" independientes que se relacionan con otras partes del problema. Estas entidades no se eligen por sus características en el ámbito computacional, sino porque tienen algún límite físico o conceptual que los separa del resto del problema. Las entidades son representadas como objetos en el programa para computadora. El objetivo es tener una correspondencia, una a una, entre entidades en el problema físico y objetos en el programa. Se puede tener dificultad en elegir objetos porque el proceso sea demasiado obvio, ya que se está acostumbrado a forzar problemas en el espacio de solución arbitraria de una programación tradicional.

El más fundamental de los propósitos que subyacen a la POO es el de permitir a los programadores gestionar y comprender programas más grandes y más complejos. La clave para alcanzar esta meta es el Objeto. En esencia, la POO implica la creación de objetos, los cuales combinan y encapsulan tanto los datos como el código que opera sobre estos datos. Los objetos pueden contener tanto elementos públicos como elementos privados. Cuando un elemento de un objeto es privado, solamente los demás elementos de ese objeto pueden tener acceso a él. Los elementos públicos son de libre acceso para cualquier otra parte del programa. Al usar elementos privados es posible controlar estrictamente la forma en que se accede un objeto. De esta manera, un objeto proporciona un nivel significativo de protección contra alguna otra parte del programa que no esté relacionada con él, y que pudiera modificar o utilizar incorrectamente las partes privadas del objeto. El enlazado de código y de datos de esta manera suele denominarse encapsulación. La ventaja de utilizar objetos es que, cuando se aplica correctamente, un objeto es una única entidad lógica que resulta más sencilla de entender y de manejar que los distintos elementos que forman el objeto.

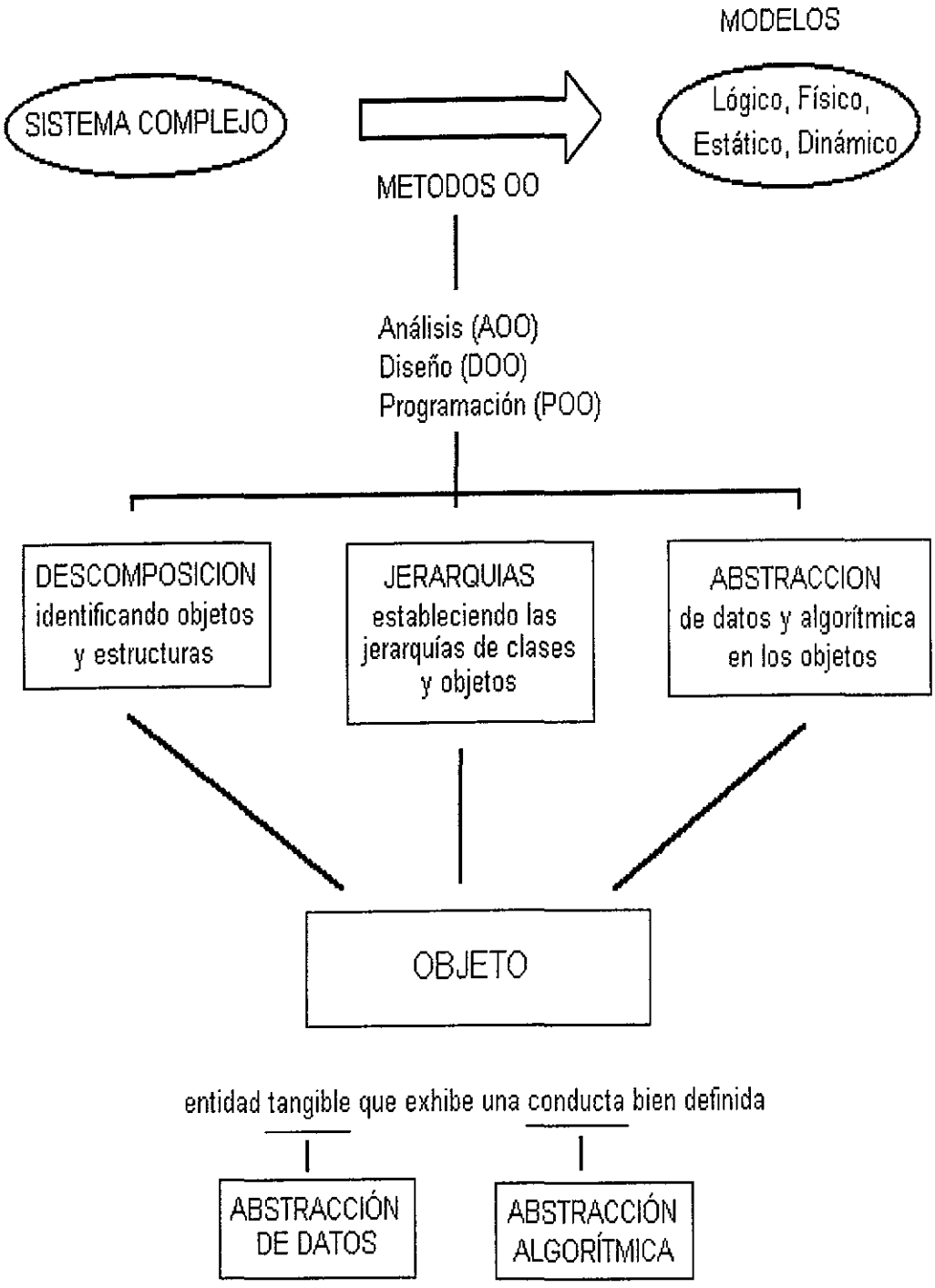


Figura 1.2.3. *Proceso de desarrollo para Sistemas Orientados a Objetos.*

La POO encubre la complejidad del programa en sí mismo, así como la ocultación de los datos, enfatiza los tipos de datos y las operaciones intrínsecas que pueden desarrollarse en aquellos tipos de datos. En la POO, los datos no fluyen libremente por el sistema, ya que están protegidos de alguna modificación accidental. Son los mensajes, más que los datos, los que se pueden mover en el sistema. A diferencia del enfoque de procedimientos que "invocaba una función en un dato", en la POO se "envía un mensaje a un objeto".

Un lenguaje orientado a objetos apoya la experimentación de dos formas:

- Los objetos son paquetes compactos que no se rompen simplemente porque se añadan otros objetos. El programador puede introducir nuevos objetos y codificarlos, sabiendo que los errores del nuevo código serán aislados del resto del sistema.
- Los tipos de objetos nuevos, pueden derivarse de los viejos. Esto ahorra al programador tiempo y permite exploraciones más rápidas. También localiza errores en el código para el tipo de objeto derivado, ya que se supone que el tipo original funciona correctamente.

El "empaquetamiento" natural de objetos tiene otro efecto importante. Los programadores eligen estructurar un programa en un lenguaje orientado a objetos porque les ayuda a codificarlo, no porque tenga un beneficio posterior.

Otra característica de la POO es que permite crear una jerarquía de objetos, pasando de lo más general a lo más específico. En esta jerarquía, cada objeto hereda las características de los que le anteceden. La posibilidad de crear estructuras jerárquicas permite al programador organizar cuidadosamente las distintas partes del programa en unidades claras e independientes unas de otras.

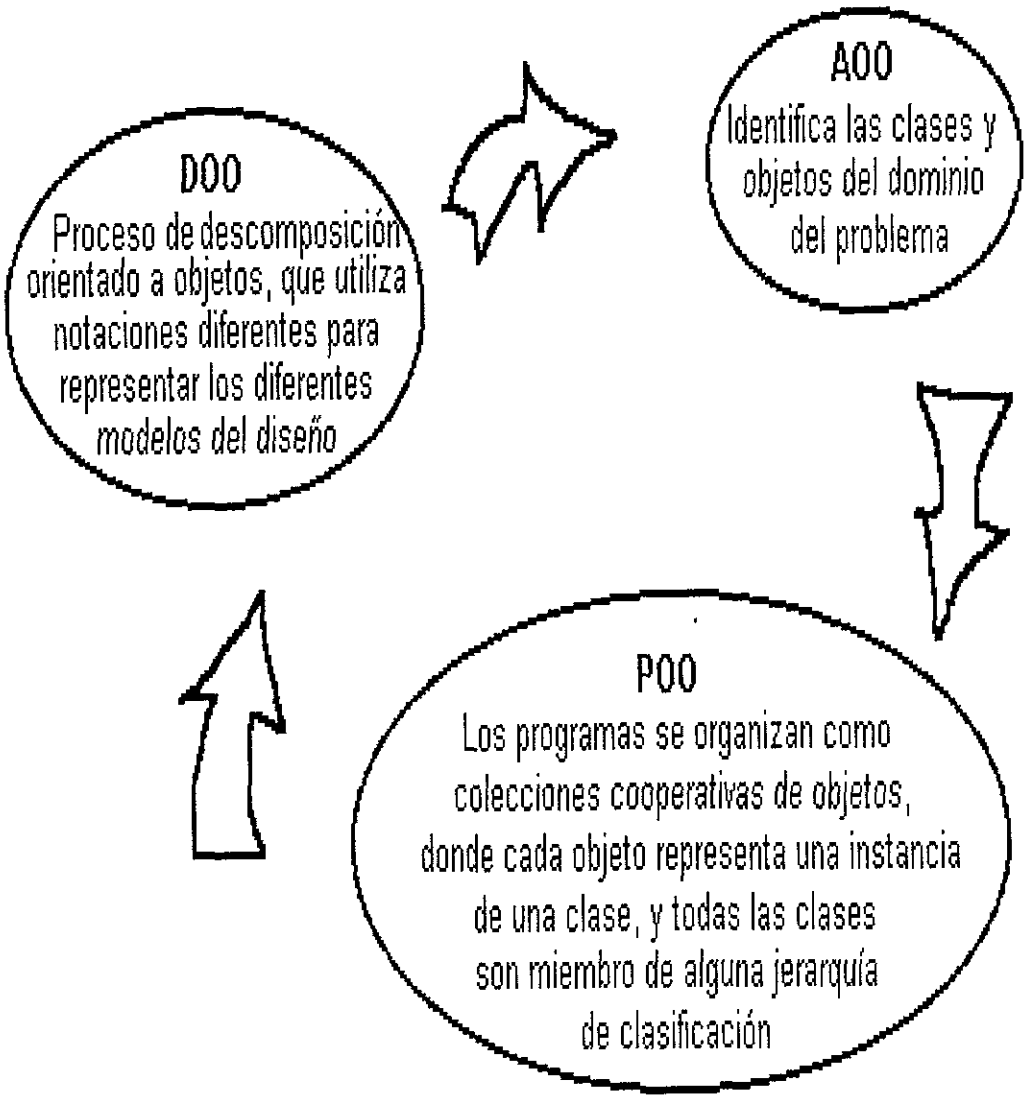


Figura 1.2.4. Análisis, Diseño y Programación Orientada a Objetos.

1.2.3. Orígenes y Evolución de C++.

Se suele decir que el C es un *leguaje de nivel intermedio*, a medio camino entre el ensamblador (bajo nivel) y el Pascal (alto nivel). En parte, la razón por la cual se creó C fue la de dar al programador un lenguaje de alto nivel que pudiese utilizarse como sustituto del *ensamblador*. Como es sabido, el ensamblador utiliza la representación simbólica de las instrucciones reales que ejecuta la computadora.

Existe una relación uno a uno entre cada instrucción del lenguaje ensamblador y la correspondiente instrucción para la máquina. Aunque esta relación permite escribir programas sumamente eficientes, hacerlo suele ser bastante tedioso, y proclive a errores. Por otra parte, los lenguajes de alto nivel, como Pascal, están muy alejados de la máquina. Una sentencia en Pascal no tiene relación con la secuencia de instrucciones de máquina que se ejecutan en última instancia. Sin embargo, aunque C sigue disponiendo de estructuras de control de alto nivel como las que se encuentran en Pascal, permite al programador manipular bits, bytes y direcciones de una forma que está relacionada más directamente con la máquina que la abstracción que ofrecen otros lenguajes de programación. Por esta razón, a veces se dice que C es un "código ensamblador de alto nivel". Como consecuencia de su naturaleza dual, C permite a los programadores crear programas muy rápidos y eficientes sin tener que recurrir al lenguaje ensamblador. La idea que anima a C es la de que el programador sabe lo que está haciendo. Por esta razón, el lenguaje C casi nunca "estorba" al programador, y éste es libre para usar el lenguaje (o abusar de él) como le parezca oportuno. Hay muy pocas comprobaciones (si es que las hay) de errores en el momento de la ejecución. Por ejemplo, si por alguna extraña razón se escribe encima de la zona de memoria en la cual reside el programa, el compilador de C no hace nada para evitarlo. La razón de esta aproximación consistente en que "el programador manda" es que esto permite al compilador de C crear un código muy rápido y eficiente, porque la responsabilidad de comprobar errores y similares recae sobre el propio programador. En resumidas cuentas, C supone que uno es lo suficientemente inteligente como para añadir su propia comprobación de errores donde sea necesaria.

El lenguaje C++ es un lenguaje OO híbrido que surgió como una extensión del lenguaje C. Hoy en día éste domina el mercado de lenguajes para construir aplicaciones, tanto en computadoras personales como en estaciones de trabajo, teniendo un sinnúmero de compiladores, herramientas y bibliotecas para lograr un desarrollo rápido y eficiente.

C++ fue diseñado por Bjarne Stroustrup en los laboratorios Bell de AT&T a principios de la década de los ochenta. Este nuevo lenguaje fue resultado de extender al lenguaje C con característica esencial de la programación OO: encapsulamiento a partir de clases y objetos. Inicialmente se le denominó como "C con clases" en el que se incluyeron clases, objetos, validación y conversión de tipos de datos de los argumentos.

Cuando se inventó C++, Bjarne Stroustrup sabía que era importante mantener el espíritu original de C, incluyendo su eficiencia, su naturaleza de lenguaje de nivel medio, y la idea de que el programador, y no el lenguaje, es quien está al mando, al mismo tiempo que se añadía el apoyo para la programación orientada a objetos. Este objetivo fue alcanzado. C++ sigue proporcionando al programador la libertad y el control de C junto con la potencia de los objetos. Las características orientadas a objetos de C++, usando las palabras de Stroustrup, "Permiten que los programas se estructuren para ser claros, extensibles y de fácil mantenimiento, sin pérdida de eficiencia".

A finales de 1983, "C con clases" fue rediseñado, extendido y reimplementado dando como resultado C++, teniendo como nuevas características funciones virtuales y la sobrecarga de operadores.

A principios de 1990, se le agregaron nuevas extensiones, principalmente herencia múltiple, abstracción de clases y mecanismos refinados para la resolución de sobrecarga.

Además de las características puramente OO que el lenguaje reúne, se le han incluido características de abstracción, manejo dinámico de memoria y estructuras que cumplan con las condiciones y restricciones especificadas durante el diseño.

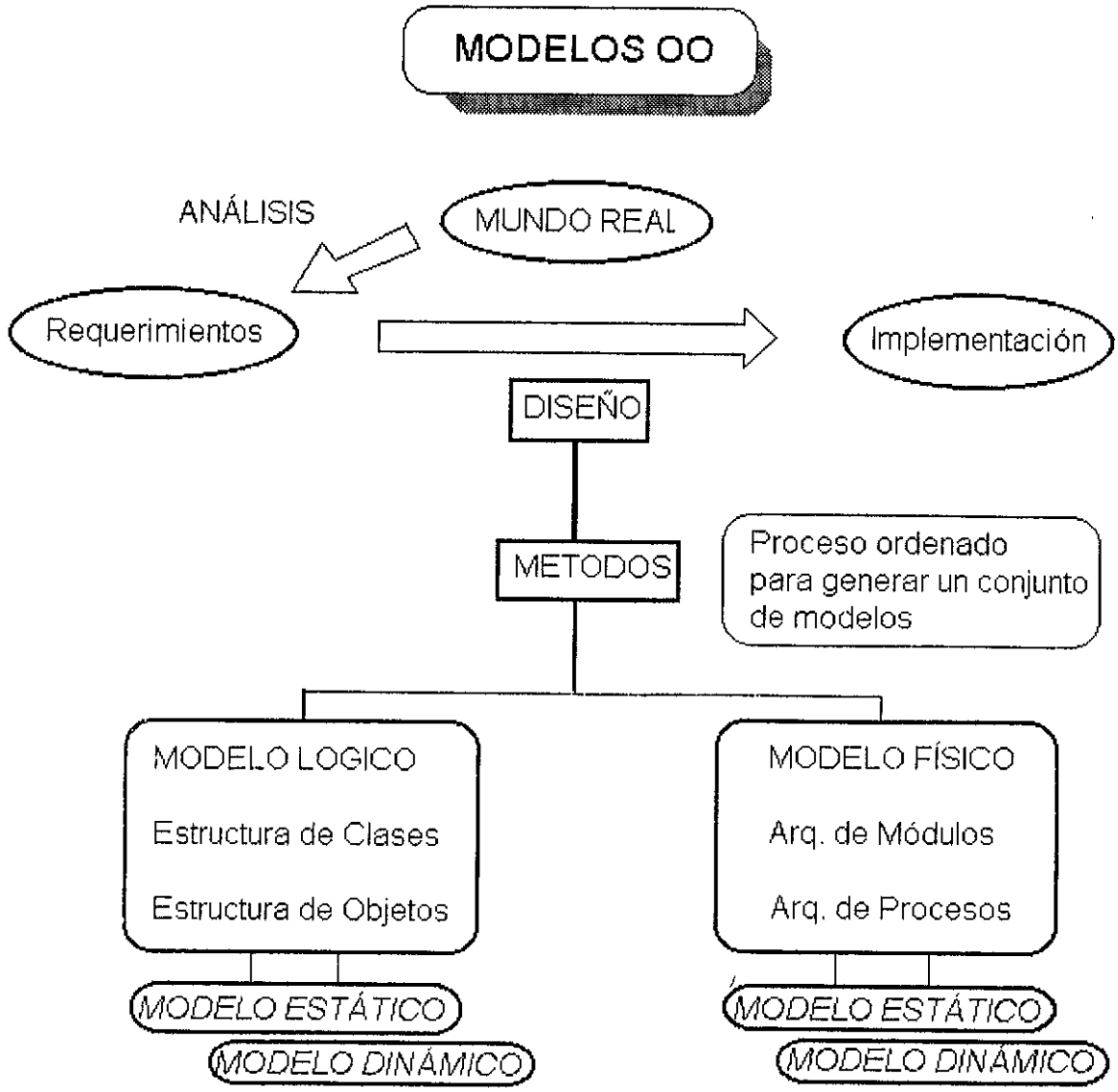


Figura 1.2.5. Diagrama esquemático del modelo Orientado a Objetos.

1.2.4. Extensión del Lenguaje C++.

Debido a que C++ está basado en el lenguaje C desarrollado por Kernighan y Ritchie, la mayor parte de los cambios especificados en el ANSI C se ven reflejados en él; por lo que cualquier compilador de C++ podrá aceptar programas escritos en el C estándar (ANSI C). En cuanto a las reglas sintácticas, podemos decir que éstas son muy parecidas en ambos lenguajes.

Algunas de las extensiones de C++ con respecto a C son:

- Creación de Entidades Múltiples
- Abstracción de Datos
- La Herencia
- Polimorfismo
- Prototipos de Funciónes
- Sobrecarga de Funciónes
- Valores por Defecto
- Referencias
- Sobrecarga de Operadores
- Ligado al Tiempo de Ejecución
- Memoria Dinámica

1.2.4.1. Creación de Entidades Múltiples.

Cuando se utiliza un lenguaje de procedimiento tradicional (figura 1.2.6), el programador crea a menudo, funciones que no se adaptan fácilmente a nuevas situaciones. La idea de "solución a una función" indica que solamente se puede utilizar para un caso concreto y que el dato está delimitado al sistema. Cuando se produce una situación donde se necesita más de una función, el programa debe diseñarse de nuevo. Por ejemplo, cuando se visualiza una información, se puede programar como si la pantalla fuera la única forma de comunicarse con el usuario.

Sin embargo, si se utiliza un sistema de ventanas cambia el planteamiento, ya que se puede disponer de muchas pantallas que pueden tratarse como entidades separadas. Para manejar entidades múltiples en un lenguaje de procedimiento tradicional, debe declararse una estructura de datos capaz de soportar toda la información necesaria para cada entidad. Por ejemplo, una estructura de ventanas tiene una posición X, Y, un tamaño, colores de fondo y de primer plano, un sitio para almacenar los datos, etc. Las funciones se definen para inicializar, limpiar y manipular la entidades. Lleva tiempo pensar en la visualización como una única pantalla y luego cambiar al nuevo planteamiento de ventanas múltiples. Si se escribe el código utilizando las características orientadas al objeto de C++, la creación de entidades múltiples es bastante sencilla: se declara una estructura de datos para cada entidad, y se llama a las funciones con la dirección de una estructura como parámetro. Es tan fácil crear una entidad como varias entidades.

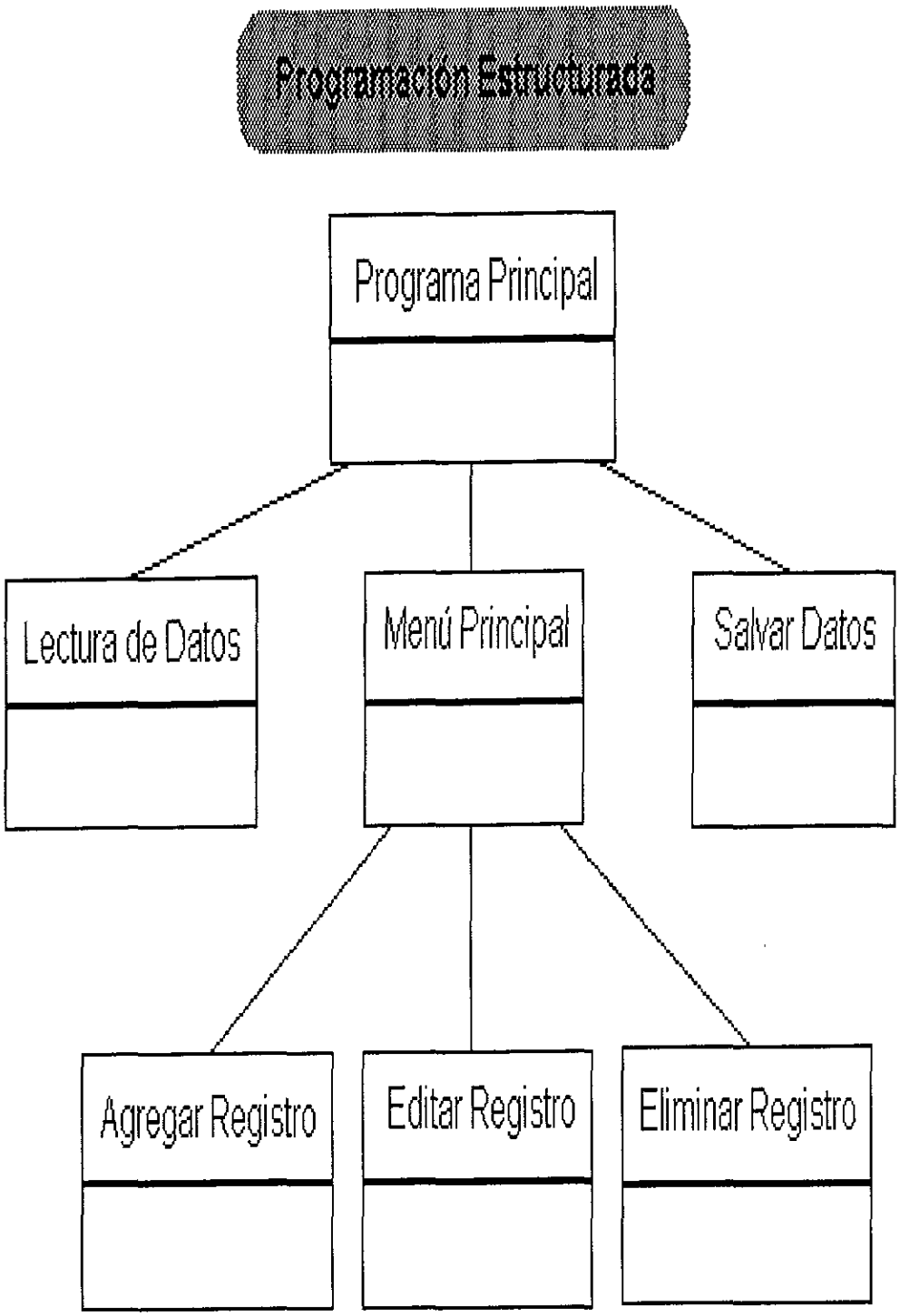


Figura 1.2.6. Ejemplo de programación estructurada tradicional.

El sistema funciona bien, pero hay mucho espacio para los errores de contabilidad. El usuario de la biblioteca debe recordar inicializar y limpiar las estructuras. A veces el programador obliga al usuario a hacer esto por medio de la manipulación directa de la estructura, más que a través de funciones. Esto puede ser confuso, ya que hay normalmente una estructura de elementos que se utilizan internamente por el sistema, aunque son visibles para el usuario. La visibilidad de los elementos de la estructura ocasiona otro problema: las funciones del usuario pueden también manipular los elementos de estructura, a veces intencionadamente y otras por accidente. El usuario puede cambiar elementos de estructura sin darse cuenta, lo que puede provocar resultados imprevistos. Cualquier persona que haya intentado utilizar la biblioteca de otra, se habrá dado cuenta que es mucho más complicado de lo que parece inicialmente. La mayor o menor dificultad, dependerá del cuidado de la persona que haya documentado el sistema.

1.2.4.2. Abstracción de Datos.

C++ soporta entidades múltiples y bibliotecas mediante la utilización de la abstracción de datos. La abstracción de datos significa que se puede combinar la estructura de datos y las operaciones de esa estructura en un nuevo tipo de dato abstracto. Un tipo de dato abstracto se comporta como los tipos de datos que son incorporados al lenguaje, con la única excepción de que no son parte de la definición del lenguaje, sino de algo que ha creado el programador.

Cuando se crea una estructura de datos en el lenguaje procedural tradicional, se crean normalmente funciones para manipular la estructura, que pueden utilizarse únicamente en esa estructura. Otras funciones no sabrían cómo manipular la estructura, ya que ésta no es un tipo de dato incorporado. Es muy lógico, por tanto, vincular a una unidad la estructura de datos y las funciones que la manipularán. En C++ esta unidad se denomina class (**clase** también conocida como tipo definido por el usuario). Las variables, o instancias de esa class se denominan objetos.

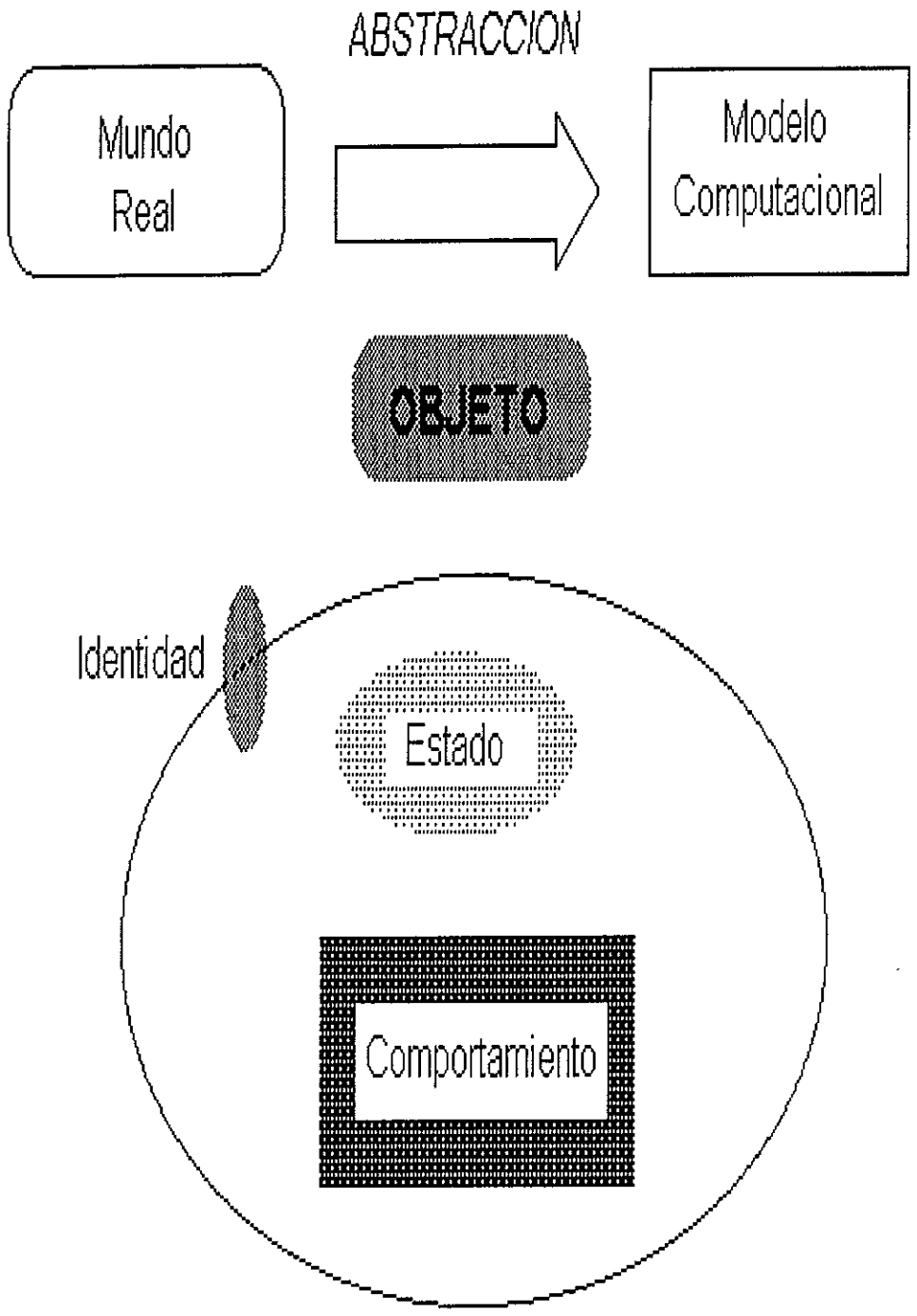


Figura 1.2.7. Diagrama conceptual de un Objeto.

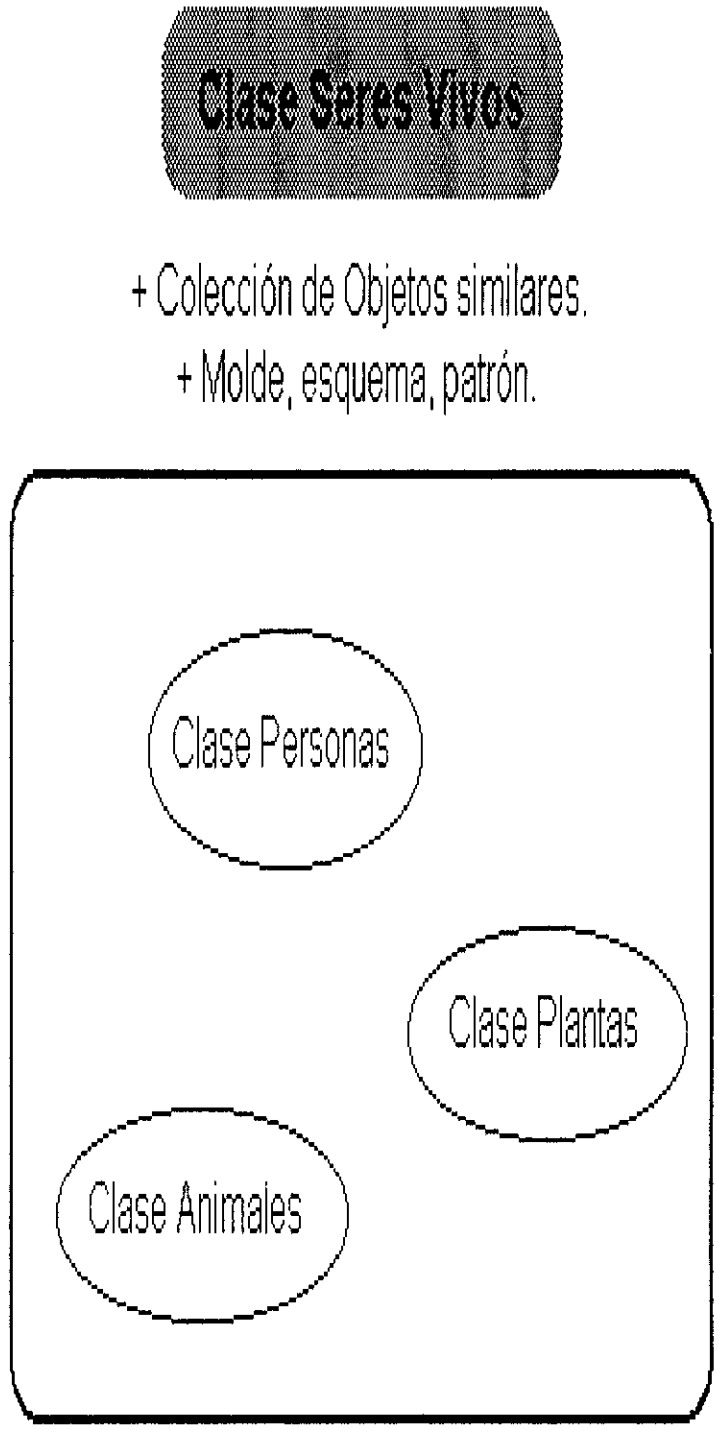


Figura. 1.2.8. Ejemplo de una clase.

Escribir datos abstractos funciona de la misma forma. Los elementos de datos de una estructura pueden ser privados, para que el usuario no pueda manipularlos directamente. Para hacer públicas ciertas partes de una class (esto es, para que puedan acceder a ellas otras partes del programa), es preciso declararlas después de la palabra reservada public (**pública**). Todas las variables o funciones que se definen después de public están al alcance de todas las demás funciones del programa. En esencia, el resto del programa accede a un objeto a través de sus funciones y datos public. Es necesario indicar ahora que, aunque se pueden tener variables public, esta métodología implica que se debe intentar limitar o eliminar su utilización. En lugar de utilizarlas, hay que hacer que todos los datos sean privados, y hay que controlar el acceso a ellos mediante funciones public. Los elementos de datos Private (**privados**), pueden únicamente manipularse por medio de funciones especiales que son parte de la class (funciones miembro) o funciones no miembro que tienen un permiso especial (funciones amigas). Esto evita una modificación accidental de los datos, y hace más fácil encontrar los errores.

Cada class tiene dos tipos especiales de función miembro. La primera, denominada constructor y que tiene el mismo nombre que la class, se encarga de la inicialización cuando se crea un nuevo objeto. La segunda, el destructor, salvaría automáticamente un objeto cuando no se necesitara más tiempo. En el caso de un objeto ventana, un constructor podría crear espacio en memoria y pintar la ventana en la pantalla. El destructor eliminaría la ventana desde la pantalla y liberara la memoria. Los constructores y destructores impiden la existencia de objetos que no han sido inicializados adecuadamente, y de este modo, elimina otras fuentes de error.

Como el requisito de iniciación es tan frecuente, C++ permite que los objetos se den a sí mismos valores iniciales cuando se crean. Esta iniciación automáticamente se lleva a cabo mediante el uso de una función de construcción. La función de construcción de un objeto se invoca cuando se crea el objeto. Esto significa que se invoca cuando se ejecuta la declaración del objeto. Además para los objetos locales, el constructor se invoca cada vez que se llega a la declaración del objeto.

El complemento de un constructor es el destructor. En muchos casos, los objetos necesitan hacer una o más cosas cuando son destruidos (hay que tener en cuenta que los objetos locales se crean al entrar al bloque y se destruyen al salir del bloque). Por ejemplo, un objeto puede tener la necesidad de liberar la memoria que hubiese reservado anteriormente. En C++, lo que gestiona la desactivación es la función de destrucción o destructor. El destructor tiene el mismo nombre que el constructor, pero va precedido por un ~.

1.2.4.3. La Herencia.

En un lenguaje orientado al objeto, se pueden heredar las características de un tipo definido por el usuario (class) en otro. Cuando se hereda se dice, "esta clase es la antigua, con algunas modificaciones", o "con algunas limitaciones".

Estructura jerárquica de clases.

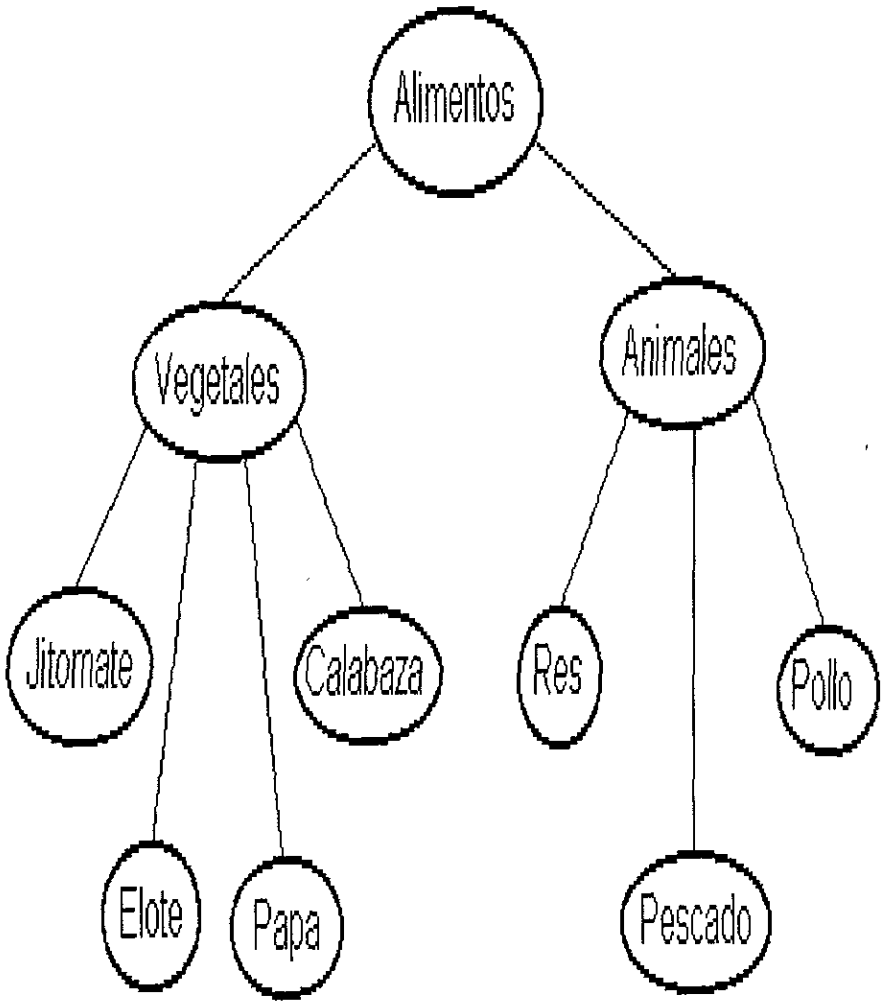


Figura 1.2.9. Ejemplo de la Herencia entre clases.

La herencia es el proceso mediante el cual un objeto puede adquirir las propiedades de otro objeto. Esto es importante porque sirve de base para el concepto de clasificación. Si piensa uno sobre esto, la mayor parte del conocimiento resulta tratable gracias a una clasificación jerárquica. Por ejemplo, una manzana golden es parte de la clase manzanas, que a su vez forma parte de la clase frutas, que pertenece a una clase mas grande, comida. Si se utilizan las clasificaciones, cada objeto tendría que definir explícitamente todas sus características. Sin embargo, cuando se usan las clasificaciones, los objetos sólo necesitan definir aquellas características que los hacen únicos dentro de su clase. El mecanismo de herencia es el que hace posible que un objeto sea un caso concreto de un caso más general.

La herencia es útil por dos motivos. El primero es conservar el soporte de codificación. Si tiene un class operacional (depurado) que otra persona ha escrito, la herencia ayuda a reutilizar el código en el class. No necesita recurrir al código fuente y entender la implementación; se hacen cambios donde se considera necesario y se reutiliza el viejo código. Si esto no fuera posible sería preferible empezar desde cero, más que tratar de adivinar cómo funciona el código.

Otro motivo por el que la herencia es útil es más sutil y eficaz. El concepto de subclase orientado al objeto, ayuda al programador a crear una solución que sea más fácil de mantener y ampliar.

Cuando una clase derivada hereda de una clase base, los objetos de la class derivada retienen, todavía, el número de miembros en la class base. Mediante la derivación de muchas classes a partir de la misma class base, se pueden crear un grupo de classes que tengan el mismo interfaz, pero diferentes implementaciones. El programa principal maneja un grupo de esos objetos. Puede enviar cualquier mensaje a cualquier objeto, pero el efecto será diferente dependiendo de cada subclase.

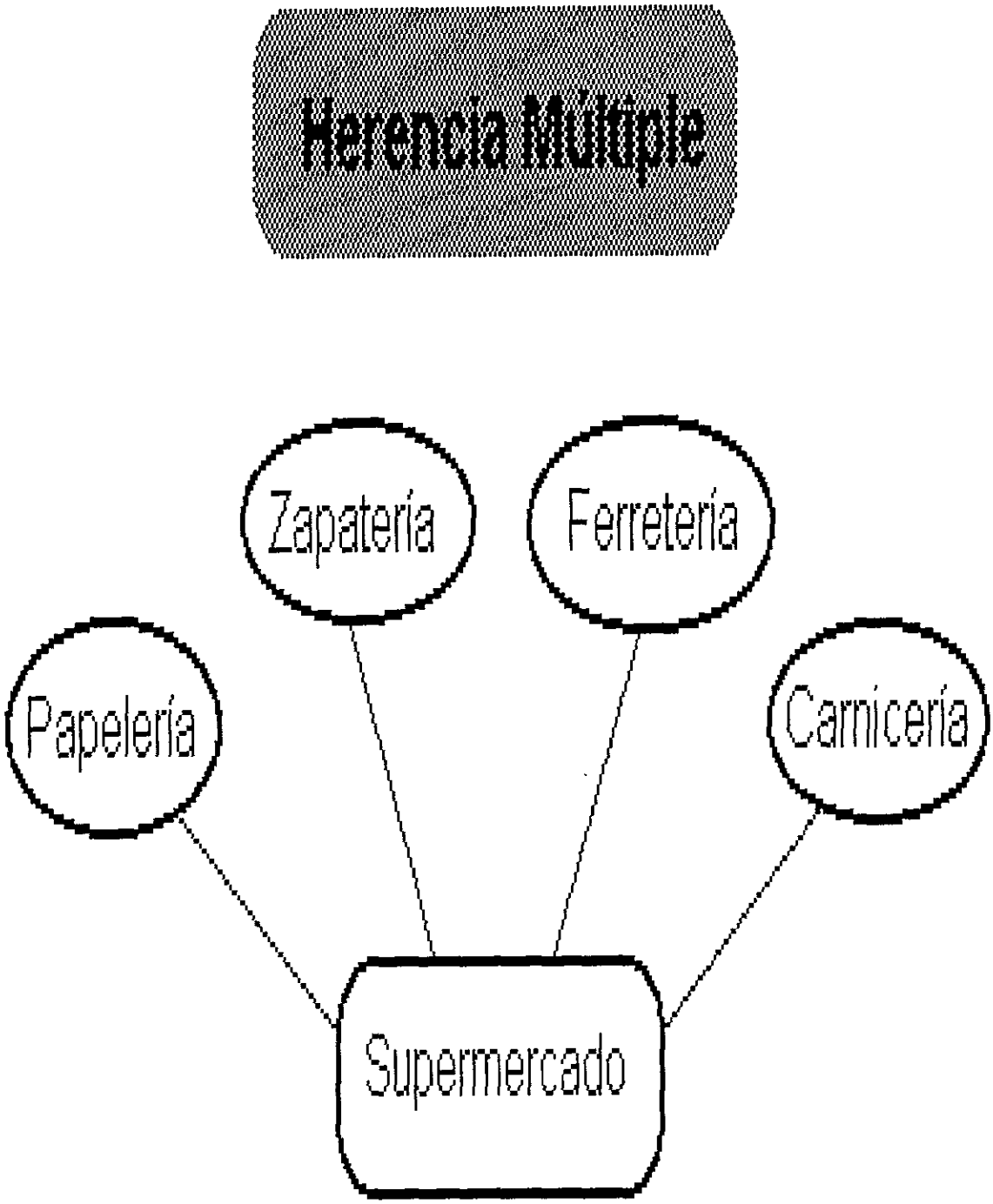


Figura 1.2.10. Ejemplo de Herencia Múltiple entre clases.

1.2.4.4. Polimorfismo.

La utilización de interfaces idénticas con diferentes implementaciones se denomina polimorfismo. Los lenguajes de programación orientados a objetos admiten el polimorfismo, que en esencia significa que un nombre se puede utilizar para especificar una clase genérica de acciones. Sin embargo, y dependiendo del tipo de datos con que esté tratando, se ejecuta una variante concreta del caso general. Como muchas de las características de C++, el polimorfismo mejora la calidad del programa. Un programa diseñado en torno al polimorfismo, es fácil de mantener y ampliar. Para ampliar un programa polimórfico, simplemente hay que derivar una nueva subclase a partir de la misma class base, que heredaron los otros objetos genéricos. La nueva subclase puede manejarse por el mismo programa sin modificación. Como el programa es únicamente un gestor para un conjunto de objetos genéricos, los errores se aíslan automáticamente en los mismos objetos. Una vez que una class base es depurada, cualquiera de los errores en una nueva class derivada es consecuencia del nuevo código en la clase derivada. Los primeros lenguajes de programación orientados a objetos eran intérpretes, así que el polimorfismo se le permitía en el momento de la ejecución. Sin embargo, C++ es un lenguaje compilado. Por tanto, en C++ se admite tanto el polimorfismo en ejecución como en compilación.

1.2.4.5. Prototipos de Funciónes.

En C++, todas la funciones deben tener un prototipo (ésta es una característica que ha sido heredada a C, y es muy saludable como estilo de programación). El prototipo de una función permite la especificación formal de entradas (tipos de parámetros) y las salidas (tipo de valor de retorno) de la función. Los prototipos tienen como finalidad poder realizar una verificación de los tipos de datos de manera más estricta; de esta forma, el compilador puede encontrar y reportar cualquier conversión ilegal de tipos entre los argumentos utilizados en la llamada a la función y sus parámetros, así como también permite conocer si la función ha sido llamada con un número distinto de argumentos.

1.2.4.6. Sobrecarga de Funciónes.

El concepto de un "objeto inteligente" que decida que hacer con un mensaje tiene un rasgo adicional en C++. La sobrecarga significa que un nombre de una función puede utilizarse de diferentes formas, siempre y cuando las declaraciones de sus parámetros sean diferentes. En esta situación, se dice que las funciones que comparten el mismo nombre están sobrecargadas y el proceso se denomina sobrecarga de funciones.

La ventaja de sobrecargar las funciones es que permite acceder a conjuntos de funciones que están relacionadas utilizando un sólo nombre. En cierto sentido, la sobrecarga de funciones permite crear un nombre genérico para alguna operación, y el compilador resuelve que función es exactamente la que realmente se necesita para llevar a cabo la operación.

La sobrecarga significa que dos o más funciones diferentes tengan el mismo nombre, diferenciándose unas de las otras en el tipo y número de parámetros. Desde el punto de vista tradicional de programación, para definir dos funciones que calculen el área de un rectángulo y de un círculo, respectivamente, éstas se deben definir con nombres y parámetros diferentes, ya que la estructura de un círculo y la de un rectángulo no son las mismas. En C++, a través de la sobrecarga, estas dos funciones se pueden llamar igual, ya que estamos representando la misma abstracción: calcular el área de una figura geométrica. De esta forma no tenemos que aprender muchos nombres para una operación que hace lo mismo. Con la sobrecarga de funciones lo único en lo que hay que tener cuidado, es en conocer qué argumentos deben ser enviados cuando es un rectángulo y cuando es un círculo (figura 1.2.11).

```
//área de un rectángulo
int area(int x1, int y1, int x2, int y2){
return x2-x1*y2-y1;
//área de un círculo (sobrecarga de función area)
float area(float radio){
return PI*(radio*radio);
}
```

Figura 1.2.11 Sobrecarga de funciones.

1.2.4.7. Valores por Defecto.

C++ permite dentro de la definición de una función asignarle a los argumentos valores a priori, conocidos como valores por default, de esta manera cuando la función es llamada con un número menor de argumentos que los especificados, el valor por default es utilizado. La sintaxis usada para la definición de un valor por default es similar a la utilizada cuando inicializamos una función en C++, permiten representar una gran variedad de situaciones en las que una función frecuentemente trabaja de la misma manera (figura 1.2.12).

En el listado de la figura 1.2.12 se puede ver que en la llamada la función mensaje, el segundo y tercer argumento no se incluyen, entonces el valor que toma por default es uno para ambos parámetros. Los valores por default son valores que se calculan al tiempo de ejecución, por ello es posible incluir en este valor expresiones que incluyan la llamada a una función o cualquier operación aritmética. En otras palabras, puede decir que los valores por default representan la manera en que una función trabaja la mayoría de las veces.

```
//Prototipo de la función mensaje
int mensaje(char texto[], int posX, int posY);

//Implementación
int mensaje(char *texto, int posX=1, int posY=1){
   gotoxy(posX, posY);
   return x2-x1*y2-y1;
}
void main(void){
   mensaje("C++"); // Llamada ala función mensaje
                   // con valores por default
}
```

Figura 1.2.12. Utilización de valores por default.

1.2.4.8. Referencias.

Tanto C como C++ realizan el paso de parámetros por valor a una función. En ambos lenguajes, cuando es necesario que una función altere los valores de las variables utilizadas como argumento, los parámetros deben ser explícitamente declarados como apuntadores (usando el operador *). C++ introduce el concepto de paso de parámetros por referencia, de esta manera la modificación de los valores de los argumentos, al llamar a una función, son manejadas de un modo más transparente. Con el paso de parámetros por referencia el compilador se encarga de ligar las direcciones de memoria, en lugar de que el programador lo tenga que decir explícitamente (figura 1.2.13).

Algunas de las restricciones que se le aplican al uso de argumentos por referencias son:

- Las variables que son referencias no pueden ser referenciadas. Esto quiere decir que no se puede obtener su dirección y asignarse a un apuntador.
- No se pueden crear arreglos de referencias.
- Las referencias no son permitidas sobre campos de bits.

```
//Definición de la función intercambio
void Intercambio(char texto[], int posX, int posY);

//Implementación
int mensaje(int &a, int &b){
   int temporal;
   temporal = a;
   a = b;
   b = temporal;
}
void main(void){
   int x=1, y=2;
   Intercambio(x,y); //Llamada a la función Intercambio
}
```

Figura 1.2.13. Paso de parámetros por referencia.

1.2.4.9. Sobrecarga de Operadores.

Una de las grandes ventajas de la programación OO está en poder utilizar los operadores, que antes se restringían al uso exclusivo de operaciones aritméticas, lógicas, relacionales, etcétera, construidas internamente en el compilador. C++, como otros lenguajes, permite la redefinición o sobrecarga de operadores, dando la habilidad al programador de nombrar a sus funciones con alguno de los operadores existentes.

Otra forma en que se logra el polimorfismo en C++ es mediante la sobrecarga de operadores. Como es sabido, en C++ se pueden utilizar los operadores << y >> para llevar a cabo operaciones de E/S en la consola. La razón de esto es que estos operadores están sobrecargados en el archivo de encabezado IOSTREAM.H. Cuando se sobrecarga un operador, el operador adquiere un significado adicional con respecto a una cierta class. Sin embargo, sigue conservando todos sus significados anteriores. En general, se puede sobrecargar cualquiera de los operadores de C++ definiendo lo que significa con respecto a una cierta class.

La palabra reservada operador es usada en estos casos (figura 1.2.14).

```
class vector{ //Definición de estructura vector
        float x,y;
    public;
        vector (float xx, float yy){
            x = xx;
            y = yy;
        }
        vector operator + (vector v1){
            return vector(x+v1.x, y+v1.y);
        }
}v1, v2, resultado; //Declaración de las variables por tipo vector
void main(void){
    ...
    resultado = v1 + v2;
}
```

Figura 1.2.14. Sobrecarga de Operadores.

1.2.4.10. Ligado a Tiempo de Ejecución.

El ligado a tiempo de ejecución, o funciones virtuales, significa que la dirección del código a la que tiene que saltar la computadora cuando se envía un mensaje a un objeto (llamada a una función miembro de la clase), se determina cuando el programa está corriendo. Esto es útil cuando no se conoce la clase o el tipo de objeto al que se está enviando el mensaje en el momento que se está compilando el programa, ya que su tipo depende de la ejecución. Para poder efectuar esta operación, el compilador de C++ debe incluir código que genere una tabla de apuntadores a funciones virtuales, y cada objeto de la clase debe tener una referencia a esa tabla y determinar cuál de ellas es la que debe llamar en el momento de su ejecución.

Ligadura Tardía o Dinámica.

La resolución de una llamada de función es el proceso de insertar la dirección (o cualquier otra referencia) de la definición de función al punto donde se llama a ésta. Cuando el programa se ejecuta, la función se lleva acabo mediante una llamada del lenguaje ensamblador a esa dirección. Un lenguaje típico compilado resuelve las llamadas de función en el tiempo de compilación.

Un programa diseñado en torno al polimorfismo maneja una colección de objetos class base. El resultado concreto de un mensaje que envía a uno de esos objetos, no puede determinarse en tiempo de compilación, ya que únicamente se conoce la class base y no la subclase del objeto. La función específica que es llamada, debe determinarse en tiempo de ejecución más que en tiempo de compilación. La posibilidad de retrasar la resolución de la función hasta el tiempo de ejecución se denomina late bindig (**ligadura tardía o dinámica**). La ligadura dinámica es una característica esencial de un lenguaje orientado al objeto porque es el mecanismo que implementa el polimorfismo.

1.2.4.11. Memoria Dinámica.

El manejo de memoria dinámica en C++ se puede hacer de dos formas. La primera es por medio de las funciones convencionales de C malloc y free, y la segunda a través de las funciones new y delete las cuales hacen a C++ más poderoso que a C. La principal característica de new y delete es que no fragmentan la memoria a diferencia de malloc y free, de ahí que se tiene un aprovechamiento total de la memoria disponible. Además, new y delete eliminan la necesidad de utilizar enmascaramiento.

En C, la definición de un arreglo de 10 enteros en memoria dinámica se podría escribir como se especifica en el *listado de la figura 1.2.15.*

```
int *pi;
pi = (int *)malloc(sizeof(int) * 10);
...
free(pi); ...
```

Figura 1.2.15. Manejo dinámico de memoria en C.

Mientras que en C++ la misma operación se puede escribir como se muestra en el listado de la figura 1.2.16.

```
int *pi;
pi = new int [10];
...
delete pi;
```

Figura 1.2.16. Manejo dinámico de memoria en C++.

Como podemos darnos cuenta la notación de C++ es mucho más simple que la de C.

1.2.5. Desempeño, Restricciones y Recomendaciones para el uso C++.

Una de las mayores desventajas originales de los lenguajes puros de programación OO es un desempeño, ya que para ellos es necesario contar con un intérprete que realice la ejecución de las instrucciones. C++ no sufre problemas de desempeño, debido a que la generación de código ejecutable se realiza a través de un proceso de compilación. La ejecución por un intérprete tiene la desventaja de correr un programa, en el mejor de los casos, cuatro veces más lento que la ejecución directa de un programa compilado. El problema está en que la llamada a todos los mensajes (por ejemplo, en Smalltalk) se resuelve a tiempo de ejecución, teniendo que buscar su dirección en un diccionario de símbolos; mientras que en un leguaje compilado como C++, la mayor parte de dichas búsquedas se resuelven a tiempo de compilación (excepto las funciones virtuales).

1.2.5.1. Portabilidad y Bibliotecas.

Debido a que existe una gran gama de compiladores de C++ para diferentes plataformas, desde computadoras personales hasta estaciones de trabajo, es posible la portabilidad del código sin realizar ningún cambio en él (siempre y cuando se use un C++ estándar) para generar código ejecutable y de esta manera correr el mismo programa en diversas plataformas. Por otro lado, la gran cantidad de bibliotecas de clases existentes hoy en día en el mercado del software, que van desde clases para el manejo de estructuras dinámicas de datos; clases que permite la manipulación de objetos gráficos como rectángulos, botones y ventanas; clases para la manipulación de bases de datos; también existen clases para el manejo extendido de la memoria o incluso para el control de los dispositivos de salida, sean impresoras, graficadores o cualquier otro, que permiten ahorrar tiempo en el desarrollo.

1.2.5.2. Recomendaciones para el Empleo de C++.

C++ es un leguaje grande y complejo que puede ser fácilmente abusado si no se usa con precaución, pero también incluye características que pueden ayudar a los programadores a hacer programas de mayor calidad y más rápido.

Cuando se desarrollan proyectos grandes, es importante que la programación se reduzca al máximo. El área de programación es la cantidad de cosas que deben ser memorizadas y comprendidas para que el código de un programador funcione correctamente en combinación con el código de otro. Cuando se da mantenimiento a un sistema, también es importante que la persona que de mantenimiento a una parte del programa no tenga que conocer el más mínimo detalle de cómo funciona el proyecto.

Entre las prácticas de programación que extienden el área están, por ejemplo, la secuencia en la que deben ser llamadas diversas rutinas para funcionar y la recolección de basura. En el primer caso, el programador debe estar consciente de las dependencias que existen dentro de un conjunto de rutinas para evitar cometer errores, y, en el segundo caso, es importante que el programador esté al tanto de los recursos que deben ser liberados.

Algunos toolkits (**utilerias**) para programar en C++ tratan de lidiar con el problema de la recolección de basura (por ejemplo Borland cuenta con una clase TsholdDelete), pero tal vez el mejor de los esquemas propuestos es el que desarrollaron John R. Ellis y David L. Detlefs, de Xerox Parc y DEC, respectivamente. El área de programación se puede extender por varias razones y C++ es un lenguaje que, desafortunadamente, promueve las practicas que aumentan el área.

1.2.5.3. Restricciones. ¿Qué cosas evitar?.

Inicialmente es muy difícil resistir la tentación de utilizar todas las características que ofrece el lenguaje C++: es necesario contenerse. Se ha probado en múltiples ocasiones que algunas de las novedosas características de C++ lo único que logran a la larga es entorpecer el desarrollo de las aplicaciones.

A continuación se listan las características que deben evitarse:
La herencia múltiple pocas veces soluciona problemas que pueden ser solucionados utilizando herencia normal con un buen diseño, y tiene como desventaja que suele complicar el trabajo del diseñador, del creador del programa y de todos aquellos que van a dar mantenimiento al código.

Las rutinas friend (**rutinas amigas**), son una mala manera de acabar con el encapsulamiento; cuando un proyecto requiere de los friend, quiere decir que hay algo mal en el diseño del sistema, es mejor atacar el problema desde el fondo: más vale esto que lidiar con ese error por años.

Los streams (**manejador de entradas y salidas para archivos, con la consola y con entradas y salidas estándar**) de C++ no hacen nada que no pueda ser hecho con la biblioteca estándar de entrada y salida de C. Y no sólo eso, sino que en la práctica los streams son mucho más lentos. Por ejemplo, un simple chequeo de archivos de encabezados para las rutinas de entrada y salida de C contra los streams de C++ nos hace ver la triste realidad: el stdio.h de C con 122 líneas, 3,410 caracteres; el stream.h y compañía de C++ con 946 líneas, 23,317 caracteres.

El uso normal de los streams genera muchas llamadas de subrutinas que se mantienen pasando una enorme cantidad de apuntadores, mientras que la versión del mismo programa que hace uso de la biblioteca estándar de entrada y salida normalmente requiere sólo una llamada a la subrutina que hace todo.

Además de que realizan muchas llamadas con los streams, cada llamada pasa una cantidad exagerada de información (como ejemplo esta el apuntador this para cada stream).

La sobrecarga de operadores es una de las características que entusiasma a muchos programadores que vienen de C. El problema fuerte que presenta la sobrecarga de operadores es que son una estupénda manera de extender el área del programa (el programador deberá saber mucho más). En general, la sobrecarga de operadores sólo debe usarse para redefinir el sentido del operador con otros tipos de datos sin cambiar el significado natural de estos. Los programas que no siguen esta sencilla regla, además de ser estéticamente inadecuados, son difíciles de mantener y comprender.

Otra característica que no debe ser usada son los parámetros por omisión de una función: sólo sirven para evitar la notación de el fuente que se está usando; sin embargo, el programador debe saber que hay parámetros que van a tomar valores por omisión y debe estar consciente de ello (incluso hay quien debe poner comentarios al lado para indicarlo), esto, como la sobrecarga de operadores, sólo extiende la cantidad de información que el programador tiene que recordar en todo momento.

La sobrecarga de funciones presenta el mismo problema que los parámetros por omisión, y no sólo eso sino que además se necesita que el programador tenga un control sumamente desgastante para saber que tipos de datos va a pasar a la función sobrecargada y que valor desea recibir. Esto por supuesto es un problema terrible cuando hay constructores que convierten entre tipos.

Los constructores en objetos globales, aunque es una característica que cuenta con muy buena propaganda del autor de C++, tiene la desventaja de que no especifican el orden en que estos son llamados, lo que impide que el programador pueda confirmar el orden de ejecución del programa (esto dependerá de la implementación del compilador).

1.2.5.4. Características de C++ que se pueden Usar y sus Limitaciones.

Aunque C++ ha probado ser un lenguaje sobrecargado de virtudes que lo hacen el PL/1 de los 90's (o el Ada, dependiendo del punto de vista y la experiencia personal), hay algunas cosas del lenguaje que se salvan. Por un lado están los prototipos de funciones, indispensables para evitar horas de tediosa depuración. Desde los programadores de Pascal hasta los programadores de C aprecian en sobremanera esta característica del C++ (afortunadamente, el ANSI C tiene algo equivalente).

Por otro lado están las class (u objetos); ahora sí con C++ se puede comenzar a desarrollar programas con toda la métodología de la programación orientada a objetos (**Programación Orientada a Objetos**),

C++, al igual que la POO, sufre de ser la novedad del momento. Ha llegado incluso al punto en que se considera retrógrada a la gente que no ve el beneficio de usar C++ y la POO como la solución a todos los problemas. Y es por esto que fácilmente suele caerse en el desacierto de generar en una forma descontrolada objetos que en realidad no tienen una función que no pueda realizarse con código de el lenguaje C estándar.

El uso de las funciones inline (**en línea**) permite mantener un estilo de programación limpio mientras mantiene la eficiencia del programa. Esta característica, aunque es útil, tiene la desventaja de que para poder ser usada óptimamente requiere que el código inline termine en los archivos de encabezados y cada vez que hace un cambio a estos, se deben recompilar todos los módulos dependientes. Por si fuera poco, si la función inline sólo se usa en un módulo, los compiladores optimizadores normalmente hacen un mejor trabajo que el programador en decidir qué funciones se convierten en inline y hacen esto sin intervención del programador.

Por otro lado, a menos que el diseño de toda la jerarquía de objetos haya sido muy bien planeada, entonces no tiene sentido separar los métodos y datos de las clases en privados, públicos y protegidos.

Un ejemplo de lo anterior es la biblioteca ObjectWindows versión 1, de Borland (la versión 2 ya corrige estos problemas) en la que existían métodos que se habían hecho protegidos y no quedaba otra manera de usar estos métodos más que derivando una nueva clase, forzando al programador a darle las vueltas al programa. Es muy recomendable que en la etapa inicial del diseño de la aplicación, se dividan los métodos y datos de las class en privados y públicos.

1.2.6. Conclusiones.

C++ extiende al lenguaje C en muchas formas, el uso inadecuado de algunas de estas extensiones puede entorpecer el desarrollo de un programa. Antes de usar alguna complicada o rebuscada característica de C++, se debe pensar si esa es la mejor solución y no un programa para que el parser del compilador recorra todos sus estados. La programación orientada a objetos no es un sustituto para la inteligencia y el diseño: al contrario, cuando se hace desarrollo de aplicaciones con objetos, es necesario contar con un buen diseño antes de empezar a codificar.

1.2.6.1. Características Generales de la POO y C++.

Una vez tocados los conceptos previos, se puede dar una definición completa.

- Un lenguaje orientado al objeto o a objetos, soporta tres características básicas: escritura de datos abstractos, herencia y polimorfismo.
- La actividad principal en la POO es la creación de nuevos tipos de datos abstractos (classes). Estos tipos de datos pueden crearse por medio de la herencia y pueden tener características polimórficas.
- El usuario de una class crea objetos y envía mensajes a los objetos. Un objeto sabrá que hacer con los mensajes.

- La abstracción y ocultación de datos reduce la dependencia entre los módulos. Estos significa que la modificación del módulo de una sección no afectará al código de otra sección, debido a que los cambios no repercuten en todo el sistema.
- La herencia ayuda a la reutilización del código. Cuando se utiliza con el polimorfismo, la herencia es útil para reutilizar diseños del programa y crear programas extensibles. Esto no es automático, por lo que debe incorporarlo al sistema.

1.2.6.2. ¿Cómo soporta C++ la POO?

- C++ soporta la escritura de datos abstractos con el constructor class. Los elementos de la class (miembros) pueden incorporarse en tipos de datos, objetos o funciones. Los miembros pueden ser public (disponibles para cualquier persona), private (únicamente disponibles para algunos miembros) o protecte (disponibles para los miembros class y los miembros de las clases heredadas). La combinación de, por ejemplo, funciones y datos se denomina encapsulación. La posibilidad de impedir que los datos sean vistos por personas no autorizadas se denomina ocultación de los datos.
- C++ soporta la herencia. Se puede derivar un nuevo tipo definido por el usuario a partir de uno ya existente, y hacer cambios únicamente cuando se necesiten. La herencia contribuye a una reutilización del código más fácil y es fundamental para implementar el polimorfismo.
- C++ soporta el polimorfismo con la palabra clave virtual. Cuando se crea una función virtual en una class base puede redefinirse en una class derivada. Cuando se llama a la función como un miembro de la class base se llama al código apropiado mediante late binding.
- La sobrecarga de función proporciona el soporte para el concepto general de "enviar un mensaje a un objeto e indicarle que hacer con el".
- C++ soporta la POO sin perder la eficacia de C.

1.2.7. Compiladores de C/C++ para DOS/Windows.

Con la creciente popularidad de Windows y de C++, el mercado se ha llenado de compiladores y de bibliotecas de clases que hacen uso de las obscuras virtudes del lenguaje.

Quizá, lo mejor de C++ es su nombre y aunque tiene algunas características que entusiasman a muchos programadores, C++ es un lenguaje que trata de ser el lenguaje que tiene todo lo que el programador pudo haber soñado o deseado y mucho más. No podemos negar que C++ tiene algunas características muy útiles, como son las clases y los prototipos de funciones, pero también tiene muchas extensiones que son muy rebuscadas o simplemente, que entorpecen el desarrollo de programas y su mantenimiento.

Uno de los mayores atractivos para usar un compilador de C++ es que hay toda una gama de bibliotecas de clases para desarrollar aplicaciones con interfaces muy pulidas.

Las dos bibliotecas más conocidas para compiladores de C++ son la ObjectWindows Library de Borland, y el Microsoft Foundation Classes que se distribuye con los compiladores de Microsoft, Watcom y Symantec. Ambas bibliotecas han pasado una serie de revisiones que han hecho el código fuente de programas viejos incompatibles con las nuevas versiones. Afortunadamente, las nuevas interfaces de las bibliotecas se han estabilizado y ambas compañías han pulido sus productos.

Muchas veces se dice que con C++ es posible desarrollar aplicaciones en menor tiempo gracias a su naturaleza orientada a objetos. Esto es cierto sólo en algunos casos, ya que casi siempre es posible conseguir bibliotecas que hagan tareas similares sin hacer uso de objetos.

En el caso específico de C++ como una herramienta de desarrollo de programas para Windows, el problema reside en que las bibliotecas de funciones son tan complejas que es más sencillo desarrollar con generadores de código que usando las clases de la biblioteca. Y es que el programador se enfrenta a aprender el lenguaje C++, la programación orientada a objetos, la biblioteca de clases y funciones para desarrollar en Windows y por último en tener conocimientos del Application Programming Interface (API **Interface de Programación Aplicativa**) de Windows. Escribir aplicaciones pequeñas con estas bibliotecas puede ser fácil, pero cualquier aplicación mediana presenta estos problemas al programador, ya que a fin de cuentas tendrá que aprender el funcionamiento de Windows.

Tan no son sencillas las clases de ambas compañías, que los compiladores vienen ahora con generadores de código que al correr crean entre ocho y doce archivos fuentes con un código muy difícil de leer, con definiciones de nuevas clases. Por todo lo anterior es importante realizar con gran cuidado la elección del compilador que se pretende ocupar, se debe verificar que el software elegido cubra el mayor número de requerimientos del desarrollador, para que de esta forma el compilador se convierta en una solución y no en un problema.

1.2.7.1. Borland C++ 4.0.

La versión 4.0 del compilador de C++ de Borland tiene muchas características como la biblioteca de clases OWL 2.0, así como un agradable ambiente de desarrollo en Windows desde el cual es posible depurar aplicaciones sin tener que usar el debugger (**depurador**) interno.

El compilador viene con un generador de aplicaciones para Windows (AppExpert) y una utilería para editar clases (ClassExpert) que interactúa junto con el pulido editor de recursos de Windows, el Resource Workshop. El Resource Workshop permite editar los recursos de archivos ejecutables, bibliotecas dinámicas, recursos de Windows y fuentes de recursos, entre otros.

Esta versión del compilador viene con un excelente soporte para el programador en Windows, sin embargo, los programadores de DOS extrañarán el ambiente integrado de las versiones anteriores, aunque todavía es posible usar el Borland C++ 4.0 para crear aplicaciones de DOS por medio de una versión del compilador que corre desde la línea de comandos.

El Integrated Development Environement, IDE (**Ambiente Integrado de Desarrollo**) en Windows viene con un manejador de proyectos que se encarga del proceso de recompilación de la aplicación que se esté escribiendo. El manejador de proyectos no es muy flexible, lo que obligará a los programadores de proyectos grandes a simplificar su sistema de mantenimiento de proyectos o a utilizar un manejador de proyectos con make y perder parte de la funcionalidad que proporciona el IDE de Borland.

Desde esta versión Borland ya no incluye el Turbo Assembler (**Ensamblador Turbo**), lo cual hace necesario obtener una copia del ensamblador de la versión 3.0, ya que será necesaria su utilización.

El compilador puede generar código de 16 bits para DOS y Windows y código de 32 bits para Windows. No hay soporte para ejecutar aplicaciones en modo protegido sin usar Windows. El optimizador genera un buen código, pero no hace un trabajo tan bueno como el que hace el de Microsoft C/C++.

1.2.7.2. Borland C++ Development Suite 5.0.

Es el paquete integrado más completo de desarrollo de C++, que ahora incluye el compilador AppAccelerator TM Just-in-Time (**justo a tiempo**) para Java.

Incluye un conjunto de cinco herramientas esenciales para el desarrollo rápido de aplicaciones. El paquete incluye Borland C++ 5.0, CodeGuard 32/16, PVCS Version Manager, InstallShield Express, y el compilador AppAccelerator Just-in-Time para Java.

Característica Básicas de Borland C++ 5.0:

- Soporte multiplataforma de 16 y 32-bit para Windows 95, NT, 3.1 y DOS.
- Nueva versión de ObjectWindows Library (**OWL Librería de Objetos Ventana**) 5.0
- Soporte para Microsoft Foundation Classes (**MFC Clases Fundación Microsoft**).
- ObjectScripting (**Escritura de Objetos**) TM un nuevo sistema programable y sencillo que permite al desarrollador modificar y configurar fácilmente el premiado entorno de desarrollo (IDE) de Borland C++.
- Visual Database Tools (**Utileria de Bases de Datos Visual**) que permite crear fácil y rápidamente aplicaciones con bases de datos.
- Borland C++ también incluye librerías estándar ANSI/ISO C++.
- Soporte OCX.
- Editor de recursos integrado de 16 y 32-bit.
- Depurador de 32-bit.
- Para facilitar el desarrollo Internet, ahora incluye herramientas de desarrollo compatibles con Java.
- Incluyendo Sun`s Java Development Kit (JDK **Paquete de Desarrollo Java en Sun**).
- El JDK está integrado en el IDE de Borland C++ y permite a los desarrolladores, crear código para plataformas cruzadas.
- Las completas funciones de Borland C++ soportan el desarrollo de aplicaciones Java, incluyendo soporte de dirección de proyectos.
- Acceso a opciones de depuración y compilación de Java a través de cajas de diálogo multipágina del IDE.
- Resaltado de sintaxis en color para el código fuente de Java.

Otras herramientas incluidas en Borland C++ Development Suite:

- **AppAccelerator TM para Java.** Borland C++ Development Suite 5.0 introduce a los programadores de Borland C++ en el desarrollo Java con el AppAccelerator TM para Java, un nuevo compilador Just-in-Time. Con AppAccelerator los programadores pueden incrementar el rendimiento de sus aplicaciones Java. El AppAccelerator trabaja con todas las aplicaciones Java cualquiera que sea la herramienta que se utilizó para crearlas. Incluso un alto incremento en velocidad puede ser conseguido cuando se ejecuta código calculado intensivo. El AppAccelerator es crítico para las aplicaciones Java de alto rendimiento, El estándar Java es un lenguaje interpretado, por ello es natural que una porción intensiva de cálculo, de las aplicaciones Java corran significativamente más despacio que su código equivalente compilado en C++. Los desarrolladores de Java encuentran que el AppAccelerator puede proporcionar un empuje en velocidad de ejecución.
- **CodeGuard 32/16.** CodeGuard (**Depurador de Código**) 32/16 es la última versión de CodeGuard, la mejor herramienta de depuración automática para Windows de Borland. Con CodeGuard 32/16 los desarrolladores de software pueden detectar, localizar y diagnosticar automáticamente bugs (**errores en la programación**) en aplicaciones Windows de 32 y 16 bits. Desde que CodeGuard 32/16 está plenamente integrado en el IDE de Borland C++, los desarrolladores no tienen que hacer el cambio de herramientas ni modificar el código fuente para usar el CodeGuard. CodeGuard detecta ahora más corrupciones de memoria de manera más rápida porque valida todos los indicadores e indicadores aritméticos de puntos de ejecución en el código de la aplicación. CodeGuard 32/16 permite al desarrollador soportar aplicaciones Windows 32-bit;
- Verificar los indicadores. Depurar múltiples módulos al mismo tiempo, incluyendo aplicaciones multi-threaded (**Multi fibra**).
- Validación de recursos de Windows usados en llamadas a funciones.

- **PVCS Version Manager.** PVCS Version Manager (**Manejador de Versiones PVCS**) es una versión especial de la galardonada versión del software de control de Intersolv. Plenamente integrado en el IDE, PVCS Version Manager permite a los desarrolladores que trabajan en grupo, seguir la pista a los cambios hechos en los ficheros de un proyecto durante el ciclo de desarrollo. Guardando los cambios en un archivo especial, los programadores pueden checar y modificar el código fuente, ejecutables, utilidades y archivos de documentación. Como se mantiene el archivo histórico del proyecto continuamente, los desarrolladores pueden asegurar que todas las versiones de su código están a salvo. Como está plenamente integrado en el IDE, los desarrolladores pueden ahora sacar ventaja de las funciones de la versión de control sin tener que aprender nuevas herramientas.
- **InstallShield Express.** InstalllShield Express (**Protector de Instalación Explícito**) es una versión visual especial de la tecnología InstallShield, basada en la última versión del InstallShield3. Diseñada para completar el ciclo de Borland Development Suite. Simplifica la instalación de aplicaciones Windows y proporciona una excelente primera impresión a las aplicaciones que están siendo instaladas. Permite a los desarrolladores crear instalaciones profesionales para aplicaciones en Windows 95 y Windows NT a través de un interface visual point-and-click. Incluye el InstallShield Objects, que instala automáticamente los archivos de runtime (**tiempo de corrida**) necesarios usados en la tecnología Borland C++, como OWL 5.0 y VDBT. Los desarrolladores pueden rápidamente especificar componentes y archivos del producto: Hacer archivos de sistema y registrar cambios. Seleccionar objetos Installshield para incluir en archivos terceras partes. También incluye soporte para desinstalación automática para aplicaciones compiladas para Windows 95.

1.2.7.3. Microsoft Visual C++ 1.5 .

Los programadores de *Microsoft elaboraron* el Visual C++, con buen ambiente de desarrollo y con buenas utilerias. Esta versión del producto incluye las Microsoft *Foundation* Classes (MFC: **La biblioteca de clases para programar en Windows de Microsoft**) y al igual que Borland C++ tiene un debugger que puede ser usado desde el ambiente integrado.

Microsoft distribuye con su Visual C++ dos Wizards (**Magos**): AppWizard y ClassWizard. Los generadores de aplicaciones de Visual C++, así como las MFC 2.5 están pensados para soportar el complejo protocolo OLE 2.0 y además crean machotes para aplicaciones que quieran usar el *Open Database Connectivity* (ODBC **Conectividad de Bases de Datos Abierta**). El problema son los 12 archivos fuentes que genera el programa para una aplicación mínima que quiera soportar OLE 2.0 y usar bases de datos por medio de ODBC.

Visual C++ viene con el compilador optimizador de su C/C++ versión 8.0, con éste es posible generar código altamente optimizado que compite con la optimización del Watcom C/C++ y la optimización del GNU C. Visual C++ puede ser usado para escribir aplicaciones de DOS. También se distribuye una versión recortada del extensor de DOS de Phar Lap, lo cual permite escribir aplicaciones para DOS que ocupen hasta 2 mega bytes de RAM, aunque también esta disponible la versión que puede usar hasta 16 megas de RAM.

Un problema que tiene Visual C++ es que no genera código de 32 bits, para eso es necesario adquirir la versión que genera código de 32 bits y tener ambas versiones instaladas en caso de que sea necesario, ocupando con esto un gran espacio en disco.

1.2.7.4. Watcom C/C++ 8.0.

Watcom C/C++ es famoso por soportar varios sistemas operativos en la plataforma Intel y por contar con uno de los mejores optimizadores del mercado. Antes de la versión 8.0 del compilador de Microsoft y de la aparición del GNU C para DOS, era posiblemente el mejor optimizador de código disponible para PC's. Watcom C/C++ puede generar código de 16 y 32 bits, para DOS y para Windows y de 32 bits para OS/2. Incluye el extensor de DOS de Phar Lap, lo cual permite que sus aplicaciones hagan uso de toda la memoria disponible en el sistema. Watcom C/C++ también se puede usar para compilar los programas NLM de Novell.

Watcom, al igual que Symantec C++, licenció las MFC de Microsoft y las distribuye con su compilador, así que es posible usar las herramientas de programación de Visual C++ y luego recompilar las aplicaciones con este excelente optimizador. Otra característica muy agradable de Watcom es que cuenta con ambientes de desarrollo integrados para DOS, Windows y OS/2. La versión de DOS incluso se distribuye con un clon del editor 'vi' de Unix, con ventanas, bastante útiles y vistosas. Cuenta también con un agradable debugger que corre con un sistema de ventanas similar al Turbo Vision de Borland.

1.2.7.5. Symantec C++.

El Symantec C++ es tal vez el que compila aplicaciones más rápido, pero no tiene optimizador de código, lo cual puede resultar frustrante para los usuarios de aplicaciones finales. Este compilador puede generar código para DOS y para Windows, pero sólo es posible desarrollar desde el ambiente integrado de Windows. Este ambiente está muy pulido, la interfaz es muy intuitiva y tiene características que lo hacen muy cómodo para los entusiastas del mouse, ya que los programadores acostumbrados a usar el teclado para manipular su proyecto se sentirán un tanto indefensos, ya que la interfaz carece de secuencias de teclas para las acciones más comunes.

Si no se tiene el inconveniente de desarrollar en el frágil y lento ambiente de Windows, los Workspace (**espacios de trabajo)** del Symantec pueden ahorrar mucho tiempo. El compilador viene con algunos Workspace predefinidos para depurar la aplicación, para trabajar en el proyecto, para controlar el proyecto, y es posible crear nuevos espacios de trabajo, donde define la colocación de las ventanas y qué es lo que éstas muestran, y puede cambiar rápidamente de un Workspace a otro con sólo un click del mouse.

Un problema fuerte de este compilador es que algunas funciones macros no están declaradas en los archivos de encabezados correctos, lo que puede complicar el proceso de convertir sus fuentes de otros compiladores del PC al Symantec o en el portado de programas de Unix a DOS.

1.2.8. Requerimientos de Hardware para utilizar C++.

Todos los compiladores que se analizaron en el punto 1.2.7 tienen la desventaja de ser lentos en caso de utilizar computadoras 486 con recursos limitados y en algunos casos sólo tienen ambiente de desarrollo para Windows.

Para poder instalar y correr las versiones más populares y potentes de C++, se recomienda contar con las siguientes características de Hardware:

Requerimientos Mínimos en el Sistema.

- 16 MB en RAM (Se recomiendan 24).
- Procesador Pentium / 166 Mhz, similar o mejor.
- Windows 3.1 (No aplica para Borland C++ 5.0), Windows 95, Windows NT 3.51 o 4.0
- De 25 a 230 MB de espacio libre en disco.
- Monitor VGA, SVGA ó UVGA.
- Mouse

1.3. Conceptos de la Plataforma Digital.

1.3.1. ¿Para qué Sirve una Plataforma Digital?

Durante la década de los 70's las computadoras se hicieron mas económicas bajo la forma de micro computadoras. Para mediados de los 80's este reciente tipo de sistemas había encontrado su lugar en muchas empresas publicas y privadas en todo el mundo. Se convirtieron en una excelente alternativa para el tareas cotidianas en el mundo de los negocios, desde trabajos de oficina clásicos como la composición de documentos, hasta tareas complejas para registrar grandes volúmenes de información en bases de datos.

En el sector financiero, la capacidad de la electrónica digital para transportar mucha información de manera rápida y confiable se hizo evidente casi de inmediato. Surgieron empresas cuyo único servicio era el de proveer a la gente interesada con información de precios de divisas, tipos de cambio, posturas de valores y todo tipo datos valiosos en el mundo del dinero. Las operaciones en casas de bolsa se automatizaron permitiendo a los operadores hacer transacciones millonarias en una fracción del tiempo que invertían antes, el tiempo de respuesta del sistema para llevar la información a un operador se volvió crucial para el éxito de las transacciones. Se definió una premisa que hasta ahora es valida: Quien mas rápido y seguro tiene acceso a la información, gana mas dinero.

Estas empresas se valieron de todo tipo de tecnología disponible para distribuir su información sin ningún tipo de estándar, y lo que es más, considerando su información muy valiosa para ser robada, la encriptaron y encapsularon en sistemas propietarios de los que solo unos cuantos conocían su funcionamiento.

Veinte años mas tarde, un operador debe tener tantos monitores frente a él como fuentes de información desee consultar y operar con ellas de manera distinta en todos los casos. De ahí surge la necesidad de estandarizar estos sistemas.

Pasada la esquizofrenia inicial, hoy en día los distribuidores de información financiera se están abriendo y ahora casi todos ellos proveen su información tanto en su formato nativo como bajo protocolos bien definidos como TCP/IP e IPX. Si bien no existen muchas aplicaciones que se le puedan dar a la información en estos formatos además de la inyección a bases de datos, se abren las puertas a sistemas que liberen la información de sus sistemas nativos para darle un mejor tratamiento. Esta es la función de una plataforma digital y sus características más importantes se describirán a continuación.

1.3.2. Características de una Plataforma Digital.

Una plataforma digital integra, simplifica y homogeneiza el tratamiento de la información y trae orden al caos que usualmente reina en las salas de producción. Permite al usuario comparar entre la información de varias fuentes y verificar lo confiable que es un dato. El usuario puede reducir costos de hardware y software, además de los costos mismos de la información por que la mayoría de las plataformas permiten la distribución selectiva de los datos a través de la red. Si una cierta casa de bolsa necesita que cinco de sus usuarios vean distinta información financiera contenida en un mismo servicio, no será necesario contratar mas que un solo servicio por que la plataforma digital permitirá seleccionar que datos van a que usuario. Si esta funcionalidad no estuviera presente, cada usuario debería contratar un servicio individual.

Como siempre hay mas de una fuente de información funcionando, el usuario corre menos riesgo de dejar de operar por la caída de los servidores.
Hay que hacer notar que cada fuente de información tiene un servidor independiente y los problemas en uno de ellos no afectarán a los demás. Una funcionalidad adicional permite poner mas de un servidor para cada fuente redundando la información. En estas configuraciones aún la falla de un servidor para un servicio no afecta a los

usuarios porque las terminales son lo suficientemente inteligentes para 'ver' que el servidor original no esta respondiendo y automáticamente de cambiaran al servidor (o servidores) auxiliares.

Otra ventaja es que para el usuario todas las fuentes tendrán una apariencia similar y se podrán usar las herramientas de la plataforma para todas ellas. De esta manera el usuario podrá interactuar con distintos servicios de forma idéntica y aprovechar la integración de todos en la plataforma y con el sistema operativo.

1.3.2.1. Capacidad de Incorporar Muchas Fuentes de Información.

Esta tal vez esta sea la característica más importante de una plataforma digital para pisos financieros. Mediante diversas técnicas se rescata la información de su formato original y se traduce a un protocolo común para la plataforma, después se distribuye en una red. Las estaciones de trabajo obtendrán de la red la información que necesiten de las distintas fuentes dándoles un tratamiento idéntico. Generalmente se usan servidores especializados para la traducción de la información conectados a la misma red local que las computadoras cliente. La figura 1.3.1 muestra un diagrama típico de plataforma digital.

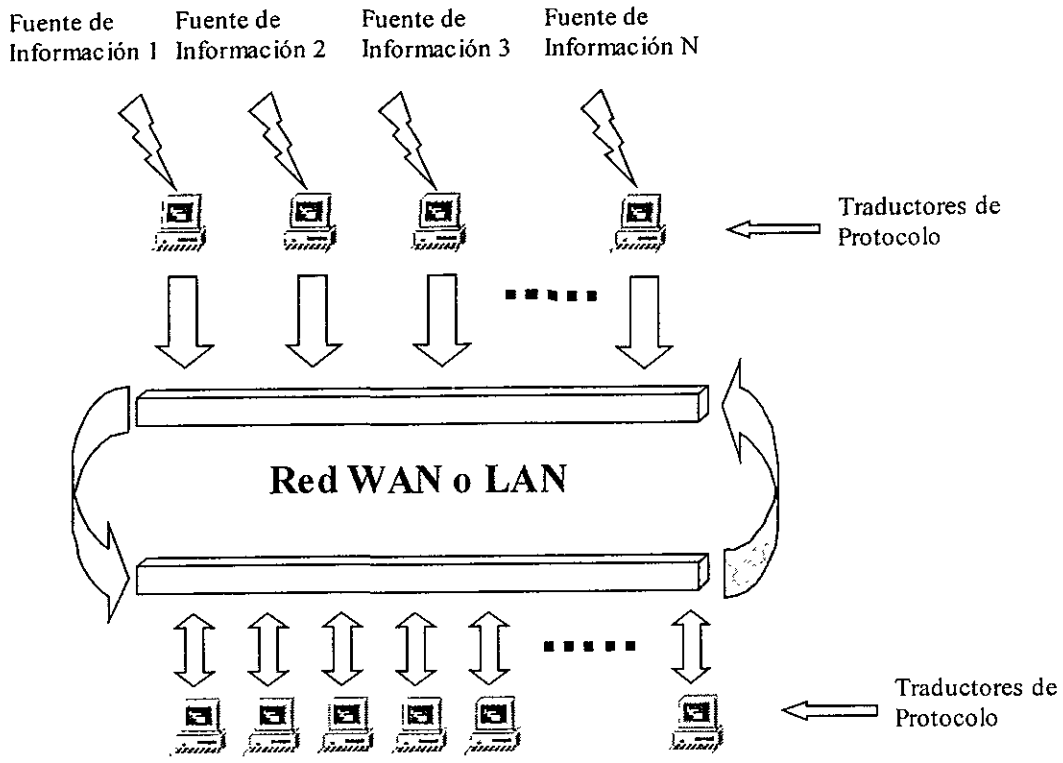


Figura 1.3.1. Diseño básico de una Plataforma Digital.

Por lo regular, la fuente de información se monta sobre algún protocolo común. El flujo de información ser rápido y confiable durante todo el proceso, para este fin se incorporan medidas de seguridad como el conocido **checksum** y tiempos limites de respuesta por parte del servidor.

La plataforma debe garantizar que todos los usuarios de todos los servicios tengan la información al mismo tiempo, es decir con un retraso no detectable para el ser humano. También debe informar al usuario del mal funcionamiento de alguna parte del sistema de manera oportuna. En algunos casos es conveniente que el servidor encargado del servicio de información que falló conserve la ultima actualización de los datos, de manera que el usuario pueda consultarlos en caso de ser necesario.

El fallo en uno de los servicios no debe afectar a los demás en forma alguna ni a la información de estos, aunque se tengan asociaciones entre ellos. El tratamiento y uso de los servicios dentro de la plataforma debe ser congruente con el tipo de información presentada buscando homogeneizar el sistema lo más posible. Tampoco las actividades de un usuario deben afectar las actividades de otros usuarios, ni el fallo de la terminal de un usuario debe afectar el buen desempeño del resto de la plataforma.

Por lo general no todos lo usuarios tienen acceso a cualquier tipo de información. Es responsabilidad de la plataforma restringir el acceso de los usuarios al tipo de información que les es permitida.

1.3.2.2. Múltiples Herramientas Incorporadas.

El usuario final de información financiera suele tener necesidades especificas del ramo. Necesita no solo ver la información, sino también ser alertado de cambios en ella y capacidad de ordenarla en una forma conveniente en el momento adecuado y de la forma adecuada. Para esto se han incorporado herramientas básicas que manejan la información controladas por el usuario final.

Las cotizaciones (Quotes).

Es la forma básica para la visualización, sirven para mostrar al usuario un cierto tipo de información en un formato preestablecido. El nombre parece haber sido elegido a razón de que la información mas deseable para un corredor son las cotizaciones de valores. Con el tiempo este nombre se ha hecho común a cualquier tipo de información con una distribución visual fija. La figura 1.3.2.2 muestra una ventana típica de cotizaciones.

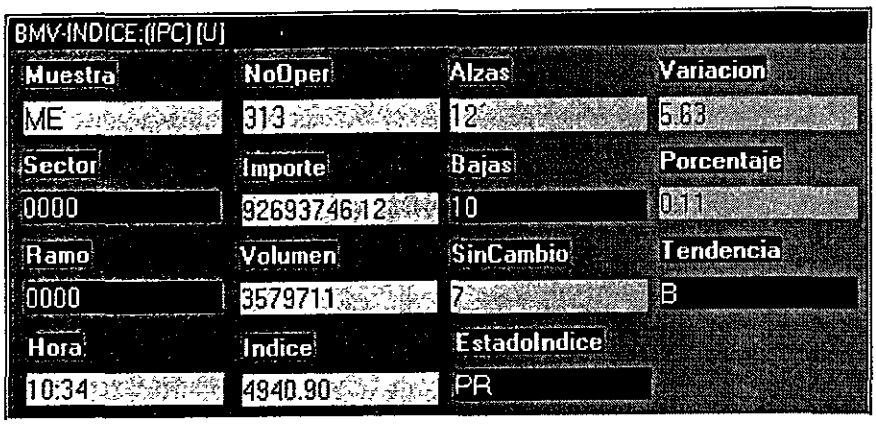


Figura 1.3.2. Ventana típica de cotizaciones.

En la figura los valores resaltados están siendo actualizados. El usuario no puede cambiar la distribución de los datos en la ventana, pero puede seleccionar los valores como elementos independientes y usarlos dentro de la plataforma ó en aplicaciones especializadas.

Las ventanas de cotización son la plataforma de salida al elegir la información útil, después los números, como elementos discretos en formato digital, podrán ser usados por hojas de calculo, procesadores de palabras y otras herramientas compatibles.

Despliegues por actualización (Tickers).

Los despliegues por actualización muestran la información cuando cambia garantizando que el usuario vea siempre el último valor emitido. La finalidad de esta herramienta es que el usuario se informe de manera visual del cambio en algún dato.

Hay dos tipos de tickers horizontales y verticales, que se describen en detalle a continuación.

Tickers horizontales.

Figura 1.3.3. Ticker horizontal .

Este es el tipo mas popular, es muy común en los techos de los pisos de remates de las casas bolsa. La información recorre el ancho de la ventana de derecha a izquierda mostrando los valores más recientes de un campo a la vez con forme se actualizan.

Tickers verticales.

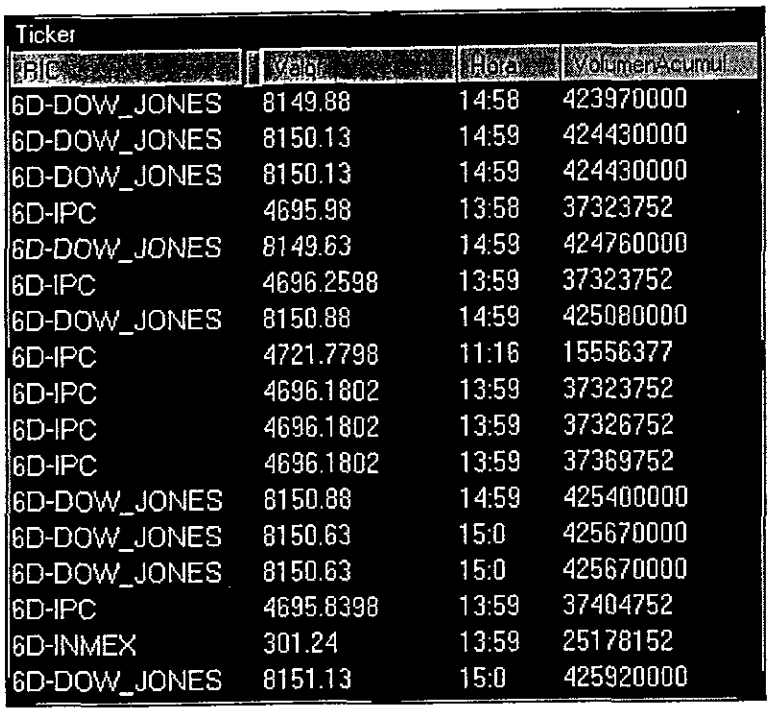


Figura 1.3.4. Ticker Vertical.

Un ticker vertical muestra varios campos de un solo objeto a la vez. Los valores mas antiguos se van recorriendo hacia arriba para dejar espacio a los nuevos en la parte inferior.

Gráficas.

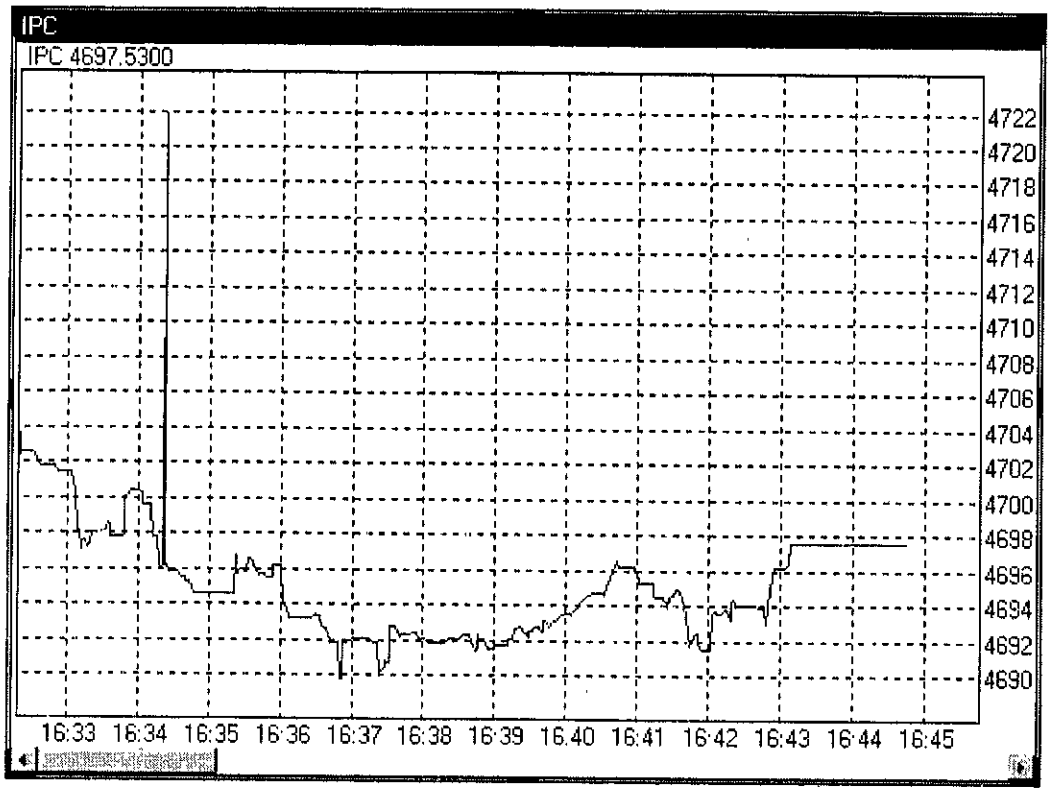


Figura 1.3.5. Gráfica.

Las gráficas permiten dar seguimiento al comportamiento de uno o mas valores. Son una herramienta muy utilizada por los operadores actuales ya que una gráfica puede dar muchisima información sobre los mecanismos que gobiernan a los mercados actuales.

Las plataformas mas avanzadas permiten hacer una serie de análisis de sencillos con los puntos representados en una gráfica. A demás el usuario podrá hacer acercamientos, alejamientos, cortar secciones rectangulares, mover los datos hacia aplicaciones especializadas, etc.

Alarmas o alertas.

En el caos de información que rodea al operador de los pisos financieros es difícil mantener la atención en cada uno de los posibles eventos. Las alarmas están diseñadas para alertar al usuario sobre cambios significativos en los valores que haya decidido vigilar.

Las alarmas se pueden programar para activarse cuando la cifra cambie un cierto porcentaje o cantidad con respecto a un valor de referencia. Cuando un alarma se dispara puede mostrar una caja de dialogo con algún mensaje, emitir sonidos, disparar acciones como el envío de un mensaje de correo electrónico, etc.

La figura 1.3.2.5 muestra un caja de dialogo en la que el usuario programa que una alarma dispare varios acciones

El usuario puede programar que varios eventos se disparen

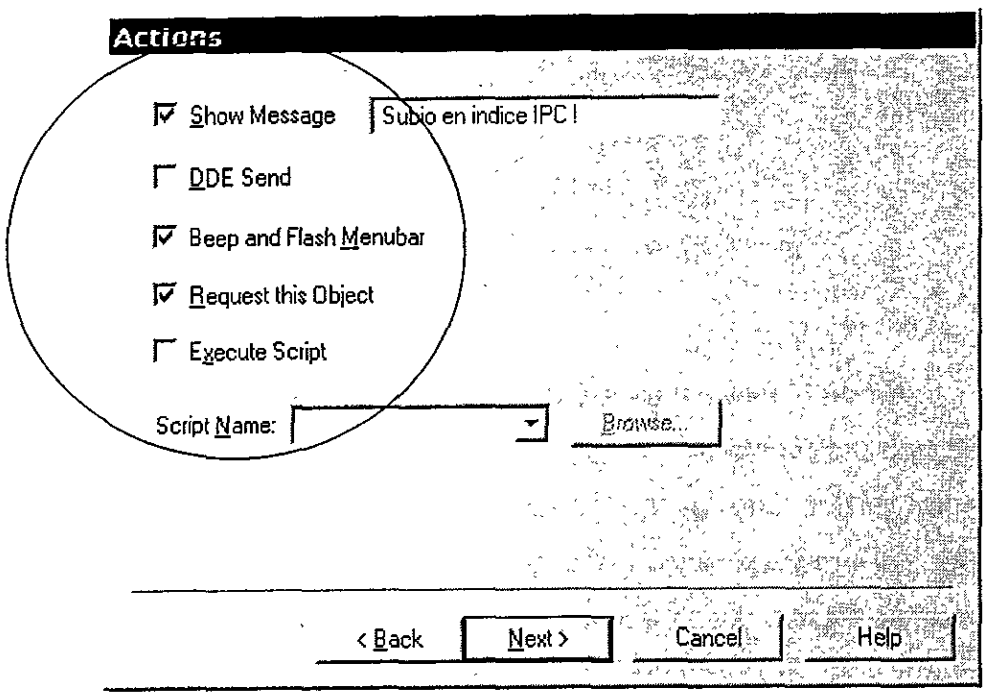


Figura 1.3.5. Programación de las acciones realizadas al dispararse una alarma.

En la figura 1.3.2.6 se muestra la caja de dialogo de una alarma que permite al usuario programar una alarma en la que la cifra vigilada se acota por cuatro extremos. De esta manera el operador podrá determinar los valores máximos y mínimos permitidos y dentro de estos los mínimos y máximos para cada uno.

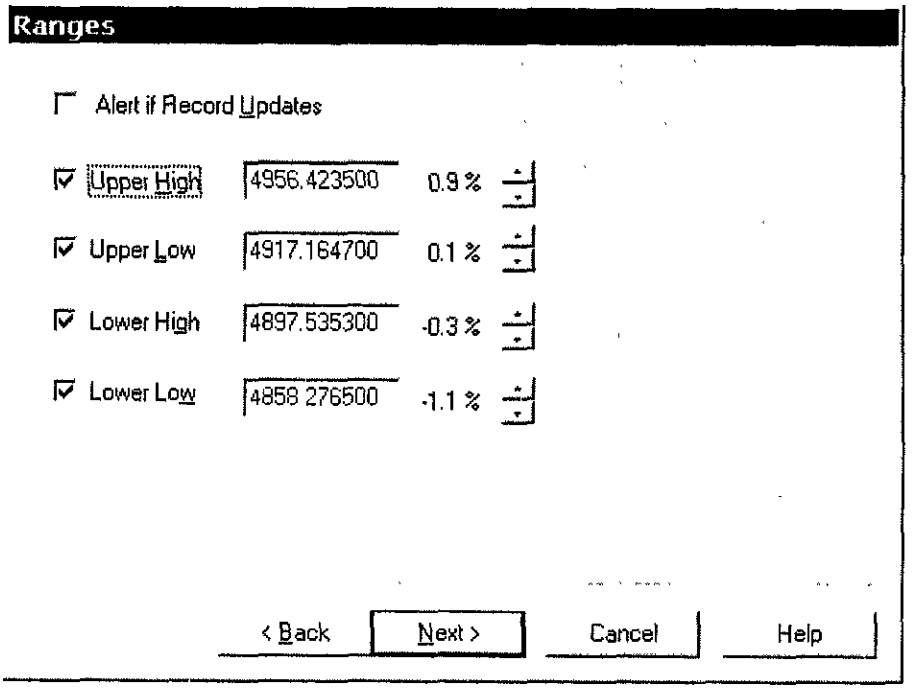


Figura 1.3.2.6. Acotación de los valores máximos y mínimos permitidos para el valor determinado en una alarma.

Las implementaciones más avanzadas permiten que una alarma se active a intervalos de tiempo predefinidos, lo cual permite que el usuario solo reciba notificaciones en los horarios en los que los mercados operan. No es difícil pensar que un operador en Nueva York maneje acciones en Europa a seis horas de diferencia, con la capacidad de ajustar intervalos de tiempo, el operador podrá activar las alarmas para valores europeos solo en la mañana y operar con valores locales por la tarde.

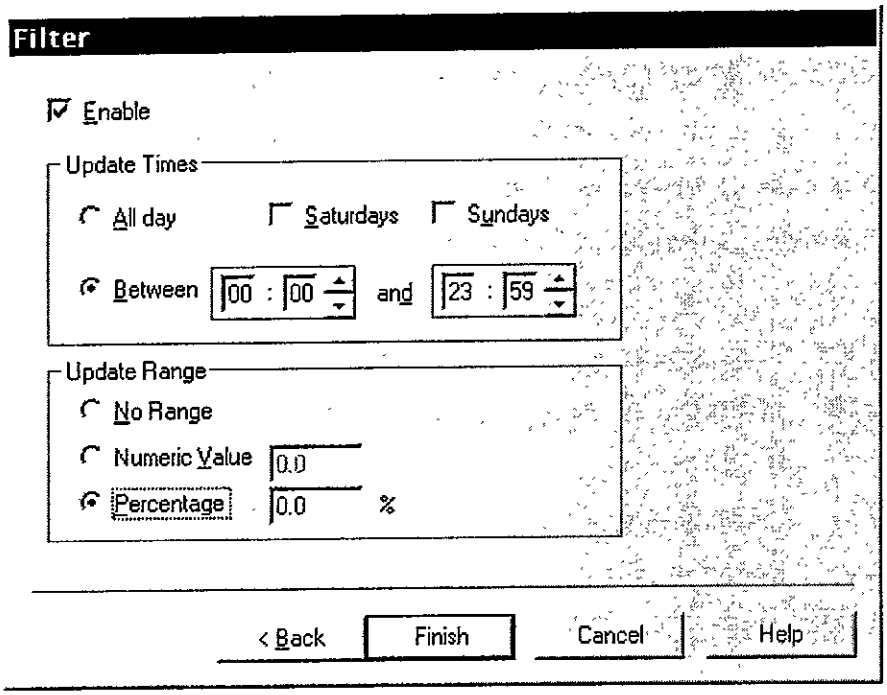


Figura 1.3.2.7. Programación de alarmas en intervalos de tiempo predefinidos.

Comodidad de la interface.

A demás de todas estas herramientas que ofrecen funcionalidad, el usuario precisa de toda la simplicidad de uso disponible. Características como el *arrastre y suelte*, menús descendentes, diálogos sencillos de interpretar y otros avances de simplificación son imprescindibles para controlar cómodamente el sistema.

1.3.2.3. Conectividad con Otros Sistemas.

La estandarización es la clave para el éxito de una plataforma digital y en los bordes de lo posible debe ser capaz de importar y exportar información hacia y desde otros sistemas. Pero esta labor no es responsabilidad únicamente de la plataforma digital, también lo es, y en gran medida, del sistema operativo.

En la actualidad los sistemas mas complejos usan ligas dinámicas, controles extendidos, etc. Como ya se menciono en párrafos anteriores, es deseable automatizar una serie de operaciones tediosas o cuya velocidad de ejecución es critica, pero cuando la información que se recibe es en tiempo real, también es deseable procesarla y tomar decisiones en base a ella lo mas rápido posible.

Una de las herramientas mas utilizadas por gente que trabaja con números son las hojas de calculo. Las más poderosas pueden ser programadas en lenguajes sencillos pero muy completos, permiten al usuario simplificar enormemente la implementación de cálculos matemáticos complejos. La terminal de una plataforma digital deberá poder exportar de manera sencilla y rápida la información a hojas de calculo para darle procesarla.

En la figura 1.3.2.8 se muestra un caso típico en el que dos valores son exportados a una hoja de calculo usando una liga dinámica. En la hoja de calculo se realiza una división entre ellos. Para colocar los datos en las celdas adecuadas basto arrastrarlos desde sus ventanas de cotización a demás los valores se actualizan automáticamente sin tener que mantener abiertas las ventanas originales. El documento con la hoja de calculo fue insertado en la terminal como un objeto *encimado* y como tal permanecerá cuando se salve el archivo a disco, al recuperarlo todas los valores en las celdas serán solicitados a la fuente de información automáticamente.

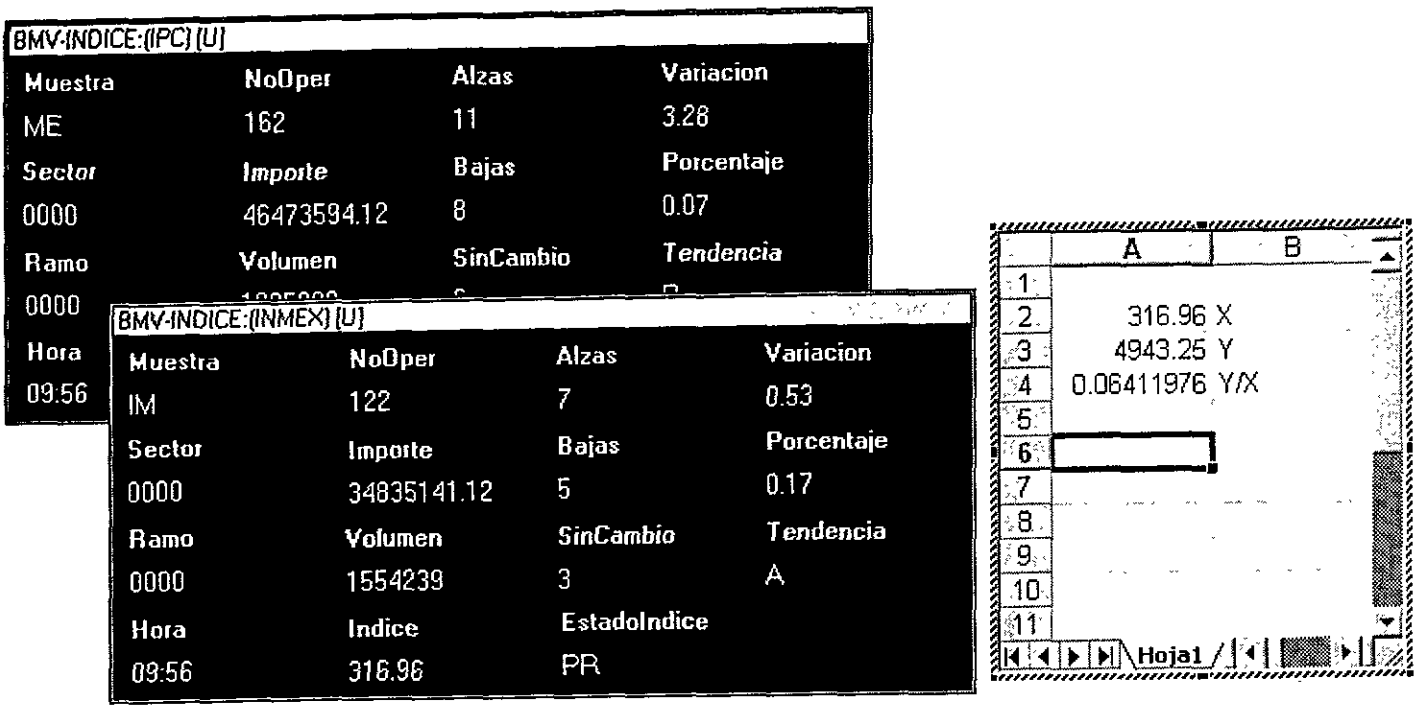


Figura 1.3.2.8. Exportación de información entre ventanas de cotizaciones y un ahoja de calculo.

Toda esta exportación de datos no sería posible si el sistema operativo no estuviera diseñado para transportar la información en forma de objetos entre aplicaciones. La mayoría de los sistemas operativos actuales tienen funcionalidad similar a este respecto.

De esta manera se pueden realizar tareas como inyectar a bases de datos con solo arrastrar y soltar los campos que nos interesan en aplicaciones cliente sin tener que programar nada de código.

1.3.3. Tecnología de Conexión.

1.3.3.1. El Servidor.

Desde el punto de vista arquitectónico, un servidor es un proceso lógico que provee de servicios a peticiones informáticas realizadas desde un cliente el cual inicia la interacción en un modelo Cliente/Servidor.

Funciónes.

Las funciones que el servidor deberá llevar a cabo son determinadas, en gran medida, por los diversos tipos de solicitudes de los clientes. A la inversa, un servidor que no es capaz de llevar a cabo una petición de un cliente no puede participar en un modelo Cliente/Servidor. En general, desde el momento en que los clientes y los servidores son interconectados dentro de una red, las siguientes funciones son requeridas por usuarios:

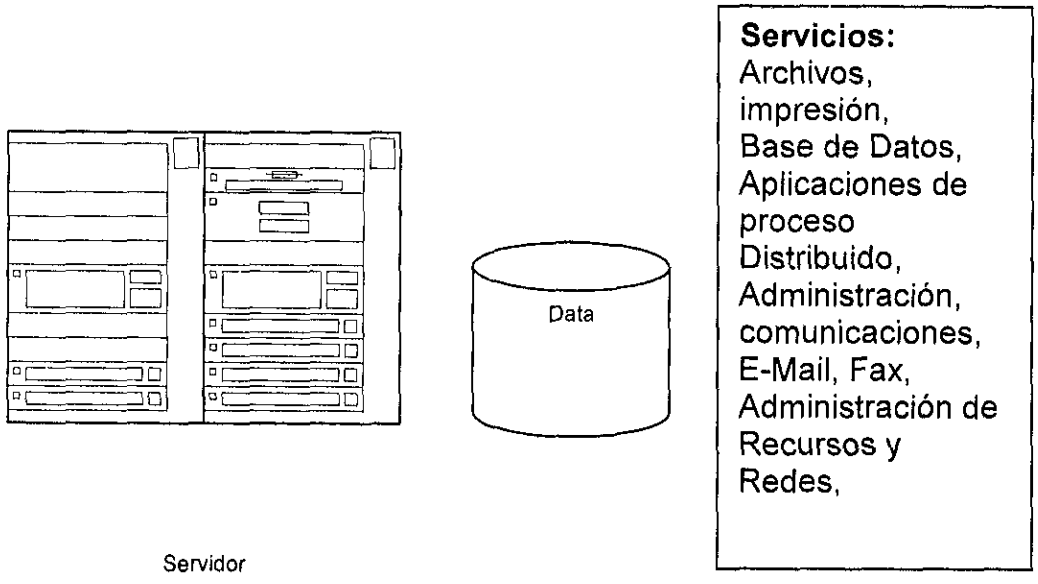


Figura 1.3.1. El Servidor.

- **Servicio de archivos**. En un ambiente de *trabajo en grupo, los clientes* tienen la necesidad de compartir los mismo archivos de datos (por ejemplo, el archivo de *tarifas en una aseguradora*) . El archivo de tarifa es puesto disponible en un servidor que tiene la característica de compartir archivos, este provee de acceso al archivo en forma completa; también se encarga de resolver el conflicto de cuando un cliente actualiza un archivo compartido negando el acceso al mismo mientras este esta siendo actualizado a otros clientes que lo estén solicitando. Otro uso típico de un servidor de archivos es la transferencia de archivos entre clientes.

- **Servicio de impresión**. *En un ambiente* de trabajo en grupo, una impresora de alta capacidad puede reemplazar a todas las impresoras individuales de los clientes. Entonces todos los clientes podrán enviar una petición de impresión de archivos a un servidor de impresión. Este servidor mantiene una cola de todos los archivos que serán impresos, enviando cada uno a impresión, en turno, hacia una impresora compartida (usualmente de alta calidad y alto volumen de impresión).

- **Servicio de base de datos**. *En un ambiente Cliente*/Servidor, el cómputo de aplicaciones es dividido entre los sistemas cliente y servidor. El servidor puede ejecutar una porción de la lógica del negocio debido a la incorporación de lógica en la base de datos. En forma similar al servidor de archivos, el de base de datos provee a los clientes de acceso a los datos que residen en él. Sin embargo, los DBMS's (**Database Managment System -Manejadores de bases de datos**) son mas sofisticados que los métodos básicos de acceso de entrada/salida utilizados por los servidores de archivos. Los DBMS's *proporcionan acceso concurrente* a los datos con varios niveles de candados e integridad de datos. Los DBMS's eliminan *redundancia* en los datos distribuidos en forma transparente a los usuarios y regularmente permite que aplicaciones especificas de acceso lógico a datos sean incorporados dentro del DBMS por el mismo. Las peticiones de los clientes accesan a los datos deseados (contrario a los servidores de archivos quienes accesan al archivo completo), y permite la manipulación necesaria sobre los datos requeridos.

Por consiguiente, múltiples clientes pueden accesar simultáneamente a una base de datos.

- **Servicio de comunicación**. En un ambiente de trabajo en grupo, que es conectado hacia un servidor remoto, todo el software y hardware de comunicación pueden ser concentrados sobre un servidor especial de comunicaciones, hacia el cual los clientes envían sus peticiones para su proceso.

- **Servicio de fax**. El servicio de fax, requiere equipo y software especial y en la actualidad estos existen y son ahora más confiables dentro de un servidor. Los clientes envían y reciben documentos de fax mediante peticiones a servicios apropiados a un servidor de fax.

Otras funciones que resuelven peticiones de clientes, como por ejemplo, correo electrónico y servidores de objetos, la administración de los recursos, configuración y administración, han sido encargados en la actualidad al ambiente Cliente/Servidor.

Características.

Un Servidor dentro del modelo Cliente/Servidor puede estar especializado en llevar a cabo una particular función para hacer mas eficiente el modelo. Sin embargo, por el lado individual, la especialización en funciones específicas, los servidores como una clase de sistema pueden especializarse en satisfacer los siguientes requerimientos de propósito general.

- **Proceso multiusuario**. Regularmente dentro de un ambiente de trabajo en grupo, un servidor dispone de servicios a múltiples clientes concurrentes. Los clientes corriendo diferentes tareas requieren de un servidor que soporte proceso multitarea. Nótese que multitarea puede ser implementado dentro de un sistema de monousuario (como OS/2) , y es un requerimiento necesario pero no suficiente para soportar multiusuario.

- **Escalabilidad.** La escalabilidad es la propiedad de un sistema la cual permite el crecimiento programado en capacidad, desempeño, proceso de información, incremento en el número de usuarios soportados, etc.; adicionando recursos de computo como fuera necesario sin cambiar las aplicaciones. Así como en las aplicaciones y sus recursos requeridos, un servidor deberá estar disponible a satisfacer esas demandas de incremento, y por consiguiente deberá proveer de escalabilidad en el desempeño. La escalabilidad no significa que los usuarios deberán comprar un servidor con una mayor capacidad a un costo mayor. Por el contrario, los sistemas deberán satisfacer los actuales requerimientos, y al mismo tiempo, deberán ser fácilmente expandidos. Esta expansión puede ser un escalamiento vertical del servidor (adición o sustitución de periféricos o unidades de disco duro o actualización de CPU), o escalamiento horizontal, en el cual múltiples servidores cooperan en forma transparente para compartir cargas de trabajo.

- **Desempeño** . Un sistema servidor deberá proporcionar niveles de desempeño y proceso de computo satisfactorios a las necesidades de cualquier negocio y a los requerimientos del ambiente Cliente/Servidor multiusuario. Por ejemplo, si la carga de trabajo aumenta con la adición de nuevos usuarios, el desempeño y el proceso de computo es afectado, en forma similar, las demandas de otras aplicaciones de cualquier otro negocio se verán afectados rápidamente, entonces el servidor deberá estar disponible a proveer escalabilidad y fácil ajuste en el desempeño y el proceso de computo.

- **Capacidad de almacenamiento.** Como el número de usuarios y aplicaciones corriendo sobre el servidor se incrementan y con avances en la tecnología del almacenamiento que reducen el costo físico de los mismos, la demanda para almacenamiento extra y tiempo de acceso son mas mayores, estos factores juntos llegan a ser un requerimiento critico para un sistema servidor. A nivel sistema, la demanda de almacenamiento llega del sistema operativo quien requiere de espacio adicional para implementar nuevas características avanzadas,

Desde usuarios que desean almacenar varios archivos de datos sobre el servidor, y desde aplicaciones como DBMS's o herramientas CASE's (**Computer Aided Software Engineering**), que son algunos de los mayores consumidores de espacio. Por ejemplo, si una estación de trabajo que esta corriendo una herramienta CASE requiere al menos de 32 MB de RAM (**Random Access Memory**) y 300 MB de Disco Duro, un servidor puede necesitar de 128 a 512 MB de RAM y de 2 a 5 GB de espacio en Disco Duro para soportar muchas de esas estaciones de trabajo.

- **Disponibilidad**. Muchas aplicaciones de misión critica son migradas o desarrolladas en un ambiente Cliente/Servidor , la disponibilidad del servidor llega a ser un requerimiento esencial del Negocio. Similar a los datos centralizados en un ambiente mainframe, hoy se espera de los servidores que estén levantados y corriendo al 24 X 7 (24 horas , 7 días a la semana) .

  Los factores clave que sostienen la disponibilidad de un servidor son la robustez y la administración en línea. La robustez implica que el sistema servidor reduce la importancia de cualquier falla particular de alguno de sus dispositivos y recobra esta en forma transparente y automática. El hardware y software son tolerantes a fallas incluyendo características como servidores stanby hot y warm, discos duplex y espejos, y el uso de subsistemas de discos RAID (**Redundant Array of Inexpensive Disks**), son todos diseñados para proveer robustez a un servidor.

  La administración en línea, esta diseñada para proveer de operación continua , donde las funciones de administración planeadas y no planeadas son reducidas o casi eliminadas. Operaciones como reorganización de bases de datos, respaldo y recuperación, levantar o terminar procesos del servidor, monitoreo y configuración del sistema, la administración de los usuarios, la actualización y de alguna aplicación o del sistema deberán ser llevados a cabo en línea, sin necesidad de echar abajo el servidor.

- **Multimedia**. Como una nueva aplicación tecnológica, la demanda para soporte de almacenamiento de multimedia será incrementada conforme a la disponibilidad que tenga. Imagen , Video y aplicaciones de Sonido serán cada día mas populares. Esto implica, que los requerimientos a los servidores no solo serán de disponibilidad de almacenamiento de imágenes digitalizadas sobre disco, también de hipertexto sobre dispositivos de almacenamiento óptico (WORM write once-read many), CD-R (CD Recordable) y DVD (Diversified Video Disk) y datos de video y sonido sobre vidocassetes, discos compactos y discos de video.

- **Redes y comunicaciones**. Las comunicaciones dentro de un ambiente *Cliente/Servidor* pasan sobre una red de transmisión/recepción. Ambos sistemas, cliente y servidor deberán tener capacidades de enlaces de todo tipo para comunicarse (si no existen los medios de comunicación no existe interacción en el modelo Cliente/Servidor).

Arquitectura de un servidor.

Los principios básicos en el diseño de computadoras describen, entre otras cosas, los factores que afectan el rendimiento del equipo. En general, estos factores incluyen la arquitectura del CPU y el tamaño e implementación de funciones dentro del conjunto de instrucciones. Debemos considerar otros factores como la habilidad de los compiladores de optimizar y proveer de un mayor desempeño, la tecnología del circuito del servidor y el sistema operativo.

En términos de aplicación, la arquitectura del CPU, la tecnología y el conjunto de instrucciones son los recursos de los cuales los compiladores y los sistemas operativos deberán ser capaces de explotar para proporcionar el mas alto desempeño. Idealmente, la arquitectura de hardware de los sistemas, los sistemas operativos y los programas habilitados (por ejemplo los compiladores) son todos balanceados y afinados de acuerdo a la aplicación en orden de alcanzar el mayor rendimiento y proceso de datos como sea posible.

Por ejemplo, aplicaciones que tienden a ser fácilmente vectorizadas, pueden ser beneficiadas si son ejecutadas sobre computadoras equipadas con facilidad de proceso vector. Muchas aplicaciones no son vectorizadas, y pueden operar mejor sobre sistemas con proceso escalar.

Otras arquitecturas incluyen memoria distribuida , sistemas paralelamente masivos y sistemas cluster. En ningún caso, el orden de utilizar totalmente las soluciones de hardware son muy costosos, los sistemas operativos deberán asegurarse de la configuración del hardware, y los compiladores deberán estar disponibles a producir código de maquina optimo para esas plataformas. De antemano, el conjunto de instrucciones llega a ser uno de los factores críticos de desempeño .

El rendimiento de una computadora puede ser descrito por la siguiente formula simbólica:

Desempeño = 1 / (ciclo de reloj) * (longitud) * (ciclos / instrucción)

El ciclo de reloj es lo opuesto a la velocidad del reloj del sistema, y esta limitada mas de las veces por la tecnología del circuito. Muchas de las actuales computadoras personales y estaciones de trabajo pueden operar a velocidades en los rangos de 75 a 200 MHz.

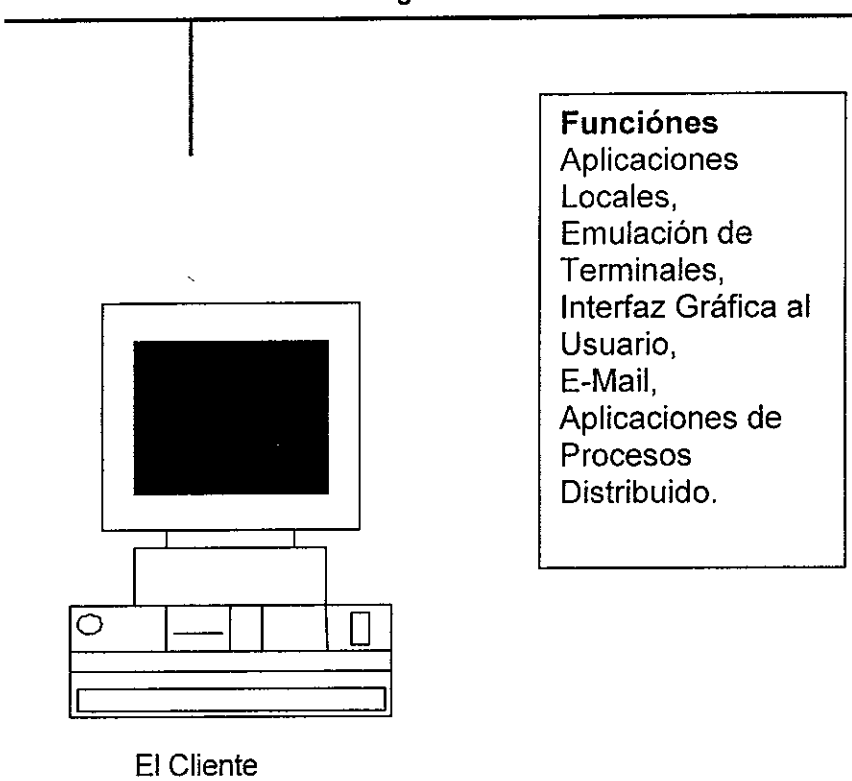
El Path Length describe el número de instrucciones de maquina necesarias para ejecutar un comando. El Path Lenght mas corto provee el mas alto desempeño resultante. La arquitectura del CPU, el conjunto de instrucciones y los compiladores optimizados son los factores que pueden reducir el path lenght. El factor de ciclos por instrucción describe cuantos ciclos de una computadora son necesarios para ejecutar una instrucción. el número puede variar desde menos que una (dentro de arquitecturas RISC) hasta mas de 1 en arquitecturas CISC.

1.3.3.2. El Cliente.

El cliente es una entidad por medio de la cual un usuario solicita un servicio, realiza una petición o demanda del uso de recursos. El cliente se encarga de realizar el FRONT-END que es la parte de la aplicación que interactúa con el usuario, básicamente es la presentación de los datos y/o información al usuario, en un ambiente gráfico (terminales gráficas y uso de iconos).

Podemos agregar que un cliente es la combinación de software y hardware que invoca los servicios de uno o varios servidores, e incluso de otro cliente. El método mas común por el cual el cliente solicita los servicios a un servidor es por medio de RPC's (**Remote Procedure Call, Llamada de Procedimientos Remotos**). Un RPC es un procedimiento que se ejecuta en otra computadora diferente a la que hizo la invocación del procedimiento, solo se invoca en el servidor.

Figura 1.3.2. El Cliente.



Funciónes del cliente.

A continuación se muestran las funciones del Cliente

- Inicia y termina solicitudes.
- Permite el manejo de la pantalla y ventanas.
- Presenta la información y/o datos
- Interpreta los comandos.
- Maneja procesos de ayuda.
- Recibe las entradas provenientes del mouse y/o teclado.
- Permite controlar cajas de dialogo.
- Habilita el manejo de multimedia (si es el caso).

Características del cliente.

A continuación se mencionan las características del cliente:

- Es el medio de enlace y/o comunicación entre el usuario y los servicios y/o recursos; por medio del cual se requieren los servicios o recursos disponibles en la red para realizar alguna actividad determinada.
- Por medio del cual se reciben los resultados y notificaciones del servidor o servidores que interactúan en el modelo; siendo el responsable de mantener el dialogo con el usuario mediante una interfaz que posiblemente sea gráfica.
- El proceso del cliente es proactivo, es decir, previo al procesamiento de datos o uso de los recursos, el cliente envía una solicitud y esta es canalizada directamente al servidor respectivo.

Tipos de clientes.

Los clientes conectados a la red pueden variar dependiendo del tipo de aplicación y de la naturaleza de la tarea a realizar.

Computadoras personales. Cuando se ha seleccionado un sistema operativo DOS, OS/2 o Windows como un estándar para la empresa, se tiene la flexibilidad y facilidad para correr aplicaciones en la misma PC (personal computer) dando por consiguiente una mayor portabilidad y compatibilidad de software con otras. La ventaja principal de utilizar una PC como cliente es que por medio de ellas se pueden expandir las redes de área local a un bajo costo y debido a que tanto las mismas PC's como sus aplicaciones se pueden migrar hacia las nuevas tecnologías para incrementar su rendimiento.

Macintosh. A diferencia de las computadoras compatibles con IBM que utilizan tecnología Intel, las macintosh de Apple Computer emplean tecnología Motorola para sus aplicaciones. Estas computadoras son las pioneras en utilizar una interfaz gráfica en el sistema operativo ya que en su arquitectura esta involucrada una GUI haciéndola mas eficiente para el manejo gráfico que otras computadoras. En la actualidad, este tipo de computadoras están mas difundidas porque el fabricante ha introducido en el mercado productos para que sean abiertas y ahora pueden ser considerados para ambientes distribuidos.

Estación de trabajo. Se utiliza como cliente cuando se maneja un sistema operativo de red como Unix o VMS en donde las aplicaciones requieren de pantallas grandes con un manejo de gráficos de alta resolución como son aplicaciones científicas y de ingeniería con procesadores de alto rendimiento. Ejemplos de estas computadoras son equipos IBM, Digital SUN, HP, etc. Recientemente se comienza a utilizar el término de estación de trabajo a todas las computadoras que están conectadas a las redes.

1.3.3.3. El Intercambio de Información.

El modelo Cliente/Servidor empieza de la idea de que todas las computadoras sobre una red podrán compartir información. A diferencia de la vieja métodología maestro/esclavo, la nueva métodología Cliente/Servidor sugiere que los datos y la potencia de proceso es compartido a lo largo de equipos independientes que forman parte del modelo.

Las redes forman en esencia mucha parte del middleware , permitiendo comunicación entre las computadoras cliente, el software y los servidores complementarios, en el procesamiento compartido y el intercambio de datos.

La columna vertebral de una red esta hecha primordialmente de varias redes de área local (LAN's). Los protocolos que corren sobre las redes transportan información entre las maquinas. Sobre cada servidor reside algún tipo de software servidor, un servidor de archivos esta formado por un sistema operativo de red. Las LAN's que permiten la comunicación entre computadoras por si mismas son usadas para comunicar con otras LAN's convirtiéndose en una red de área geográfica amplia.

Redes de area local. Estas han crecido en número y tamaño a una velocidad fenomenal, conectando un gran variedad de computadoras: PC's estaciones de trabajo, mainframes (servidores multiproceso) y supercomputadoras (Numerosos procesadores en paralelo). Como esta tecnología ha crecido , otras como Cliente/Servidor y el computo distribuido proveen de mas caminos económicos para compartir información que los sistemas mainframe. Las LAN's se han extendido a lo largo de pequeñas oficinas, escuelas, gobierno y universidades.

Con el incremento del uso de LAN's para uso personal y de negocios, ha sido crucial para los sistemas Cliente/Servidor pues se ha provisto la capacidad de interfaz a los usuarios de la red en forma fácil y eficiente.

Términos básicos de redes.

Nodo. Un dispositivo conectado a una red. Usualmente una computadora , estación de trabajo o un servidor. Ocasionalmente, un ruteador, un gateway, un repetidor, bridge o un hub.

Paquete. Es el bloque electrónico de datos básicos construidos para una red.
Un paquete es por si mismo un contenedor de estructuras de datos que son enviados sobre la red. Un paquete consiste de dos piezas: la cabecera y los datos. La cabecera contiene información de la dirección origen, destino y el tipo de paquete. El paquete de datos es la información misma que viaja sobre la red.

Un paquete es enviado uno por una computadora (o servidor) y es pasado a lo largo de la red. Cada nodo checa la información del paquete cabecera revisando si su dirección es igual a la dirección destino. Si la dirección checa, el paquete pertenece al nodo y acepta los datos. Si no, el nodo no leerá los datos y el paquete pasara al siguiente nodo.

Protocolos de transporte.

Los protocolos de transporte trabajan sobre la capa física, permitiendo la transmisión y manipulación de datos sobre la red, teniendo como principal función la transmisión de paquetes, colisión de datos y sensado de la línea.

Ethernet. Es el mas común de los protocolos de red . Cada paquete ethernet consiste de un preámbulo, una dirección destino, una dirección fuente, un tipo de campo y un CRC checksum. Soporta una industria estándar llamada Carrier Sense Multiple Access with Collision Detection (CSMA/CD). Este protocolo asegura que la colisión de datos sea manipulada rápida y de forma confiable. Por ejemplo, si dos nodos están trasmitiendo paquetes al mismo tiempo, esto causa colisión de datos sobre la red.

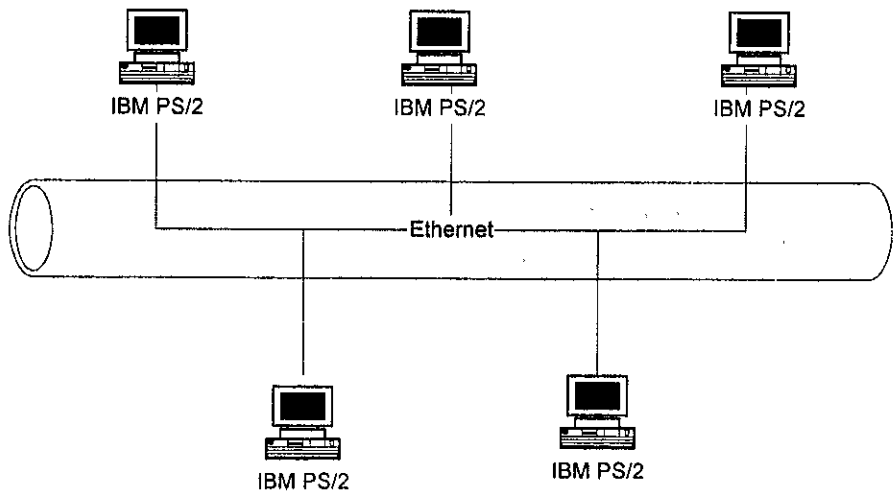


Figura 1.3.11. Protocolo de Transporte Ethernet.

Todos los nodos sobre la red paran inmediatamente de transmitir datos y cada uno de los nodos que trasmitieron datos colisionados esperan un tiempo aleatorio antes de difundir nuevamente los paquetes (esto es llamado protocolo no-persistente).

Por este camino, el nodo con el tiempo de espera mas corto empieza a difundir y los otros checan el bus, si se encuentra ocupado, esperan hasta que este se desocupe. Un gran número de colisiones de datos decrementan la velocidad de la red.

Ethernet, dentro de la implementación estándar, puede correr a velocidades de 10 MB/segundo. Debido al incremento en la demanda sobre las redes, la velocidad de transferencia de la red alarga sus limites y es incapaz de manipular una gran carga de datos eficientemente. Esto es visto como un decremento en el desempeño y como consecuencia incrementa el número de colisiones. Para resolver este problema, ha sido creada, una Ethernet rápida, IEEE 802.3u. La ethernet rápida tiene velocidades de transmisión de 100 MB/Segundo, 10 veces mas rápida que la implementación Ethernet normal. Esta nueva implementación retiene el protocolo original CSMA/CD.

Token Ring. Una red token ring usa una topología anillo. La información es pasada usando un protocolo de marca de paso. Hay dos tipos de paquetes (llamados frames en la terminología Token-Ring): un frame de mensaje y frame token. Un frame es pasado de nodo a nodo alrededor del circulo. Cada nodo checa los frames para ver cuando es un mensaje o un token. Si el frame es un token, esto significa que la red esta disponible para enviar un frame mensaje. Si el frame es un mensaje, este checa la dirección para ver si el mensaje es para este nodo. Si no, el mensaje es enviado al siguiente nodo. Los avances en la tecnología en redes Token-Ring se reflejan en la creación de los switches Token-Ring disponibles para operar dentro de la capa de liga de datos. Estos switches ofrecen una ayuda en velocidad aprovechando el ancho de banda a través de un puerto de alta densidad que provee un incremento en la velocidad de la red basada en el número de puertos activos. A través del uso del protocolo de ruteo fuente, los datos son enviados de acuerdo a la ruta localizada en el frame cabecera.

ARCnet. Es un protocolo desarrollado en los años 1970 por Datapoint Coorporation. En este protocolo se usa un "bus" con tecnología igual a la Ethernet, pero usa un token que determina cual nodo esta controlando el bus (igual que token ring). cada nodo ARCnet es asignado a un número. El token es pasado consecutivamente de nodo en nodo. El nodo que actualmente tiene el token puede entonces enviar datos sobre la red.

FDDI. Esta basado sobre un estándar Token-Ring. FDDI usa un token de control de acceso a la red. Este protocolo es primordialmente para altas velocidades, muy confiable. Una topología de red FDDI es instalada igual a un anillo dual. El anillo extra provee redundancia. Si una parte del anillo se deshabilita o desconecta, el protocolo automáticamente se reconfigura por si mismo para pasar los datos alrededor del área deshabilitada. Un concentrador adiciona protección a la red habilitando nodos que serán conectados y desconectados desde el sistema sin interrumpir la operación de la red. FDDI son implementados con fibra óptica y soporta transmisiones de velocidades alrededor de 100 MB/segundo.

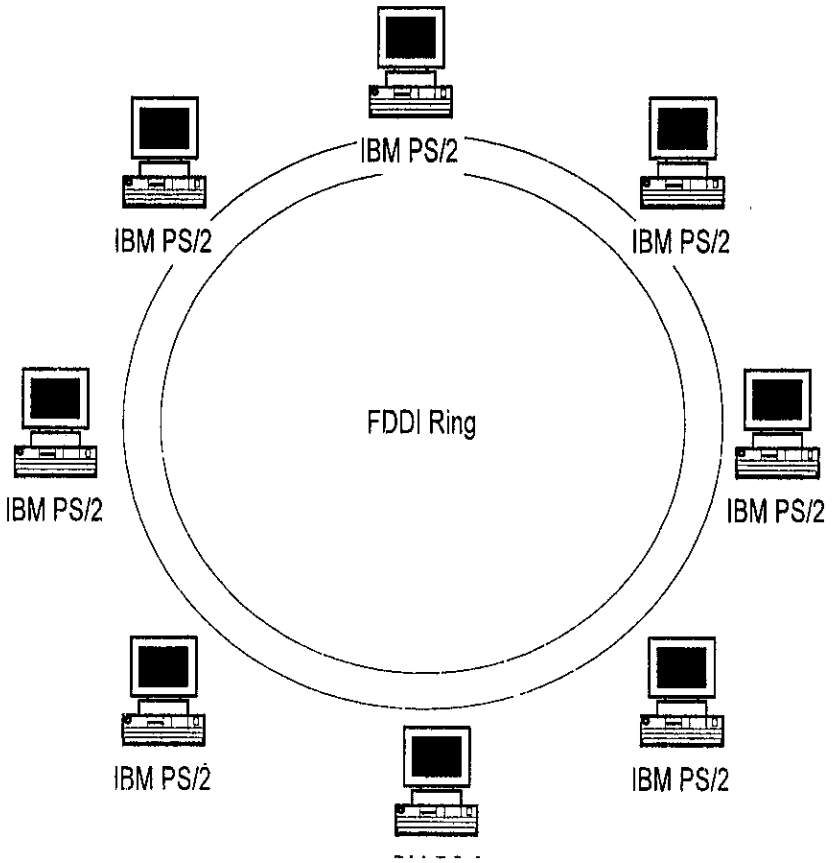


Figura 1.3.12. Red FDDI Token Ring.

Hardware de Redes

El hardware a continuación descrito es utilizado normalmente para conectar una LAN con otra o una LAN con una WAN. Este hardware tiene funciones como la de un simple amplificador de señal, extender la longitud de una LAN existente o unirla con otra. El hardware de red incrementa la capacidad y complejidad empezando con repetidores, bridges, ruteadores, gateways y hubs.

Estas capacidades son implementadas a través de circuitos electrónicos internos y software adicionales que proveen de características como regeneración de paquetes, administración de la red, ruteo y control de flujos.

Repetidores. Estos operan en la capa física del modelo de red, permiten extender la distancia de paso de los paquetes, sin embargo inducen tiempos de retardos a los equipos receptores.
Algunos ruteadores tiene la capacidad de retransmitir y regenerar señales; estas características aseguran que los paquetes podrán ser usados cuando lleguen a su destino.

Bridges. Este opera en la capa física y en la sub capa MAC (Media Access Control) de la capa data-link del modelo OSI. Este hardware habilita la interconexión de redes que tienen el mismo sistema operativo pero diferente hardware. Por ejemplo, un bridge se pone entre una red Ethernet que usa paquetes IPX y otra red Token Ring que también usa paquetes IPX. Un bridge tiene la habilidad de aprender cual dirección de nodo reside sobre una red. Esto provee a la red la capacidad de control de trafico sobre la misma permitiendo el acceso o denegandolo a través de procesos llamados de filtrado y reenvío. Este proceso transparente a protocolos de alto nivel, es una característica básica diseñada dentro de un bridge. Un bridge generalmente tiene un alto desempeño en el proceso de datos, requiere menos tiempo de mantenimiento, administración y son menos costosos que los ruteadores y los gateways.

Algunos de los bridges avanzados tienen características como encerrar y liberar direcciones (la cual provee privacía y restringe acceso a usuarios), protocolos basados en filtrado/reenvío (la cual filtra el trafico sobre la red basado en los requerimientos de los usuarios), respaldo de ruteo (la cual establece rutas de respaldo automáticas activadas solo cuando una liga falla), y SNMP (Simple Network Management Protocol).

Ruteadores. Estos son el siguiente paso de los bridges, proveyendo a los usuarios con capacidades de expansión a través de programación. Los ruteadores normalmente operan en la capa de red dentro del modelo OSI, haciéndolos mas costosos y difíciles de instalar y administrar Son usados para controlar y dirigir el trafico de una red a través de la mejor ruta, como consecuencia de esto se incrementa el desempeño en proceso de información y reduce la congestión. Algunos ruteadores y protocolos no

son compatibles (como DECnet Local Area Trasport LAT e IBM Netbios), por lo tanto eliminan la habilidad de rutear paquetes con esos protocolos.

En una red de muchas computadoras existe mucho trafico propiciado por las mismas provocando transferencia lenta de paquetes y mayor colisión de datos. Usando un ruteador, la red puede ser partida en pequeñas redes llamadas zonas. Si un ruteador recibe un paquete que procede a otra zona, este lo envía a esa zona. Si el destino del paquete es en la misma zona el paquete nunca será expuesto a otras zonas.

Gateway. Son similares a los ruetadores excepto que ofrecen capacidades adicionales mas complejas el cual dispone de mayor desempeño en el protocolo de traslado. Pueden conectarse a redes que usan diferentes sistemas operativos y pueden o no tener hardware similar. La transmisión de gateway vincula el uso de tres protocolos: El protocolo fuente, el protocolo de transmisión y el protocolo de red destino.

Por ejemplo, un gateway puede habilitar una red token ring corriendo paquetes IPX y una ethernet corriendo paquetes TCP/IP para comunicar libremente. El gateway traduce los paquetes al Sistema Operativo nativo de la red antes de pasarlos a través de ella. Pueden también proporcionar interfaces entre varios sistemas y servidores de bases de datos.

Hubs. Con el rápido crecimiento de las redes y de los requerimientos de integrar a mas usuarios y estaciones de trabajo, el primer concentrador o hub fue desarrollado. Un hub es un punto central donde computadoras o redes son interconectadas. Tradicionalmente para una topología estrella. Este hub puede ser implementado físicamente de repetidores, bridges, ruteadores o un gateway.

Existen dos tipos de hubs: los pasivos y los inteligentes. Un hub pasivo solo se utiliza como centro de una estrella, conectando varios nodos. Un hub inteligente, de cualquier modo, puede ser controlado en forma remota por el sistema operador y puede retornar información de paquetes de trafico para su análisis.

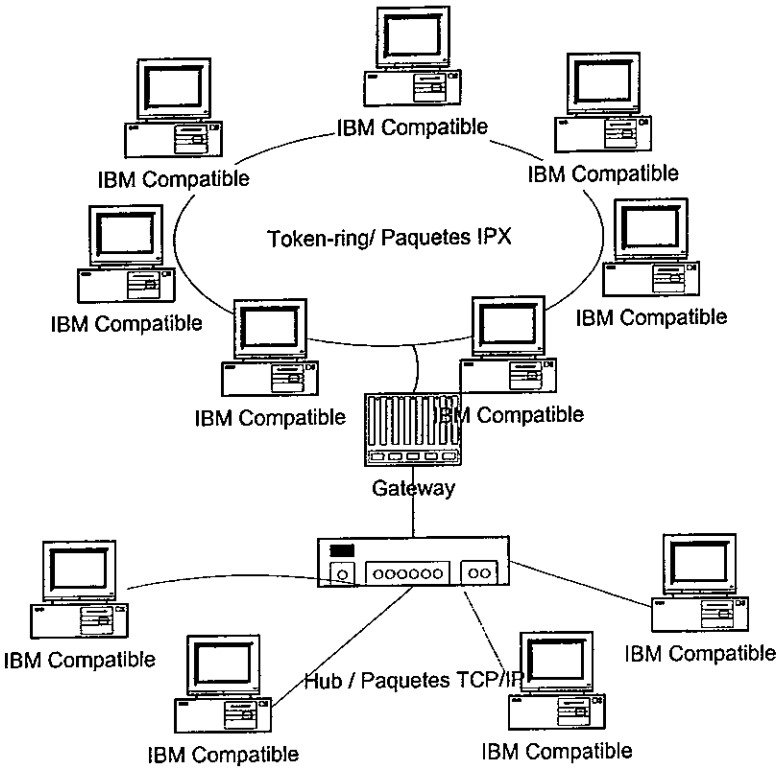


Figura 1.3.13. Combinación de Hardware de comunicaciones (Gateway y Hub).

Middleware.

El middleware entre el cliente y el servidor es la parte mas difícil y compleja de interactuar en el modelo Cliente/Servidor. El término middleware es generalmente usado para referir al software que habilita la comunicación entre el cliente y el servidor, en forma transparente al usuario y consiste de una serie de drivers o programas que apilados en muchas capas y protocolos que pueden ser utilizados en forma simultanea.
El middleware degrada el desempeño (cada driver apilado requiere tiempo adicional y ciclos de proceso), pero a cambio permite establecer comunicación entre sistemas heterogéneos, con la posibilidad de comprar piezas adicionales de middleware e integrarlas en aplicaciones brindando una mayor capacidad de conectividad.
El middleware se puede clasificar en tres tipos: Llamadas a procedimientos remotos, (rpc), sistemas con paso de mensajes y conversaciones. Algunos ejemplos de middleware son: ODBC (**Open Data Base Conectivity**); DRDA (**Distribuited Relational Database Architecture**) de IBM; SqlNet de Oracle.

1.4. Conceptos Relevantes de la Arquitectura Cliente/Servidor.

Los sistemas Cliente/Servidor se han convertido en la parte mas importante en la estrategia tecnológica de eficiencia y productividad de muchas organizaciones, lo que explica su éxito. El diseño Cliente/Servidor ha dominando todas las formas de computo sobre una red, provocando que la implementación con esta tecnología sobre todo en redes de área local se hayan convertido en un estándar dentro de la comunidad de los negocios. Actualmente, los sistemas Cliente/Servidor pueden compartirlo todo, desde los datos hasta la potencia del proceso distribuyendo la responsabilidad de procesar e integrar los datos, el software y hardware en ambos el Cliente y el Servidor.

Concepto.

Cliente/Servidor es un modelo para sistemas de información con tecnología de procesamiento distribuido el cual funciona en una red de computadoras ya sea en una red local o una red de área amplia. Su funcionamiento se inicia con un proceso que hace un requerimiento de servicios (cliente) y otro proceso respondiendo a estos (servidor) los cuales pueden correr en la misma o en diferentes computadoras dentro de la red. En Cliente/Servidor existe la separación de procesos o división de esfuerzos de un proceso de datos entre dos distintos componentes, en este modelo el servidor tiene la tarea especifica de atender peticiones de los clientes en la red. Cuando una petición es recibida, el servidor responde a las necesidades del cliente, este recibe el resultado y lo presenta al usuario en un ambiente gráfico, como si la tarea hubiese sido realizada localmente por el cliente.

Evolución.

La mayoría de los ambientes de computo actuales incluyen soluciones tradicionales que permiten el acceso a aplicaciones residentes de un sistema central (host) y/o PC's que se han conectado en forma serial a un servidor de terminales sin infraestructura de red alguna.

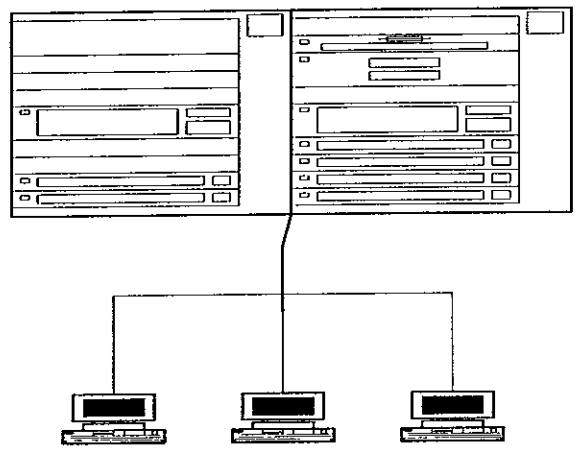


Figura 1.4.1. Sistema centralizado.

Una organización con este tipo de configuración no puede adoptar un ambiente Cliente/Servidor de manera inmediata. Sin embargo, la correcta instrumentación de un ambiente Cliente/Servidor se basa en un proceso evolutivo y no en la generación de una revolución dentro de la organización. El primer paso lógico para las empresas es aprovechar el potencial del que ya disponen en sus equipos de escritorio, integrando estas maquinas en un ambiente de red.

Las ventajas de integración de estos ofrece la posibilidad de optimizar todos los recursos de la organización: la información, la tecnología y lo mas importante: el personal. Mediante un acceso funcional cruzando hacia los datos de la empresa, el personal se vuelve mas productivo. Esta flexibilidad mantiene a las empresas en óptimos niveles de competencia. En años anteriores las implementaciones Cliente/Servidor fueron teniendo diferentes procesos de evolución. A continuación se describen las características de los estados que ha tenido este proceso.

Primera etapa.

Se utiliza el término Cliente/Servidor para la descripción de la interconexión de estaciones de trabajo programables (PWS Programable Workstations) en la red de tal forma que comparten recursos como archivos y/o impresora. Las estaciones de trabajo fueron aumentando en las empresas, corriendo en esas software productivo, como hojas de calculo., procesadores de texto, etc. Al mismo tiempo, algunas compañías enlazaron estaciones de trabajo al "mainframe", accesando sus aplicaciones emulando una terminal, pero todavía con la distribución de aplicaciones entre el "hosts" en un proceso cooperativo muy estrecho. Estas implementaciones son usadas cuando las estructuras son homogéneas y compatibles en su ambiente de comunicación. El control de la aplicación y gran parte de la lógica de la aplicación se realiza en el "host".

Segunda etapa.

Este nivel se caracteriza por la proliferación de herramientas que se dedican al desarrollo para aplicaciones Cliente/Servidor. Herramientas inicialmente orientadas a diseño de interfaces gráficas de usuario, se expanden a herramientas de desarrollo mas globales. Muchos proveedores de sistemas manejadores de bases de datos (DBMS) proporcionan métodos y herramientas que abarcan todos los aspectos de desarrollo de aplicaciones. Sistemas para grupos de trabajo evolucionan de aplicaciones de productividad a sistemas que ahora ejecutan muchas aplicaciones criticas del negocio en grupos de trabajo. Muchos de los proveedores recurren a soluciones basadas en redes de área local eliminando sistemas basados en procesamiento centralizado.

Presente.

Típicamente las estaciones ahora se ejecutan en un ambiente multi aplicaciones y los usuarios controlan la ejecución desde sus interfaces gráficas (GUI), que son en ocasiones orientadas a objetos.

La integración de aplicaciones de negocios desarrolladas dentro de la organización con aplicaciones comerciales estándar disponibles que (hojas de calculo, procesadores de texto, correo electrónico, etc.) se realizan por medio de la interfaz gráfica de la instalación, que permite la compartición de datos entre las aplicaciones; la comunicación directa y el intercambio de datos entre programas de aplicación utilizando tecnologías como DDE y OLE. Esta es la era de la computación en red, donde las computadoras de las compañías ("mainframes", "mini-computadoras" y pc´s) están conectadas entre si. Las aplicaciones y los datos están distribuidos mas allá de la red. El objetivo es tener aplicaciones ejecutándose a través de múltiples plataformas, con acceso transparente a funciones distribuidas, datos y recursos del sistema donde quiera que se encuentren.

Infraestructura.

Todos los sistemas Cliente/Servidor están divididos en tres partes: El cliente, el middleware y el servidor. Ya hicimos un análisis de las funciones y características de los elementos que componen el modelo, sin embargo a manera de resumen, presentamos el siguiente esquema.

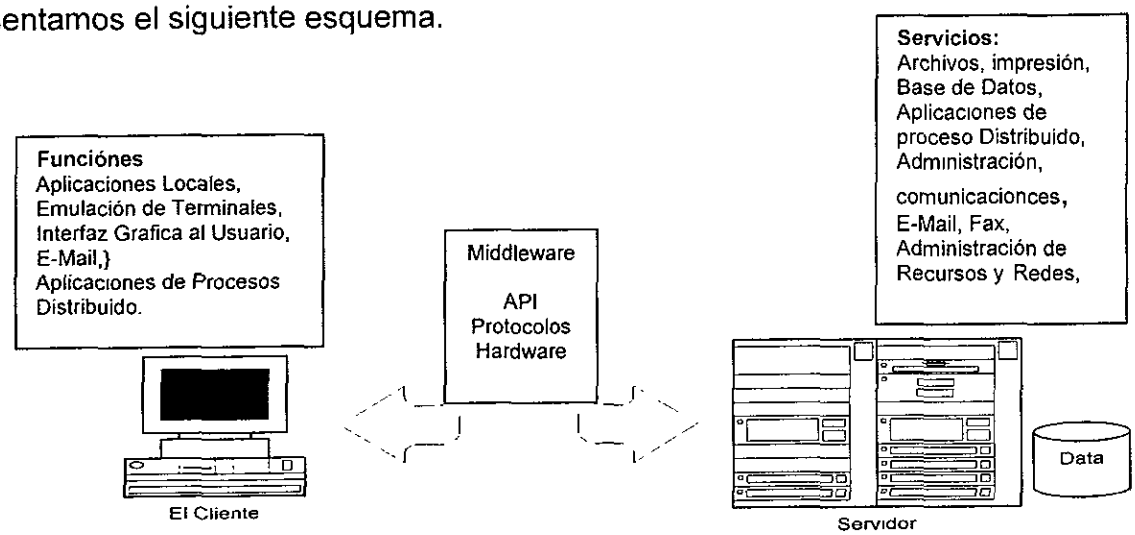


Figura 1.4.2. Infraestructura Cliente/Servidor.

Capas en aplicaciones Cliente/Servidor

Las aplicaciones Cliente/Servidor pueden ser divididas en aplicaciones de dos y tres capas. Esta es una forma de diferenciar como una aplicación puede ser dividida entre clientes y servidores. El middleware empleado y las herramientas de desarrollo de aplicaciones son frecuentemente los que determinan el nivel de flexibilidad en la distribución de datos y funciones.

Aplicaciones de dos capas.

La aplicación es dividida en dos partes, ejecutándose en una estación de trabajo (cliente) y un servidor. Los servicios de presentación, la lógica de la misma, la lógica del negocio y de acceso a los datos son usualmente una pieza monolítica que se ejecuta en la estación de trabajo cliente, por lo que la base de datos esta físicamente localizada en el servidor, junto con el DBMS. Estas implementaciones están frecuentemente construidas con lenguajes de cuarta generación y sistemas manejadores de bases de datos El formato de mensajes entre el cliente y el servidor es invariablemente alguna forma de SQL. Algunos DBMS ofrecen procedimientos almacenados que se prestan para implementar parte de la lógica del negocio en el servidor. La infraestructura Cliente/Servidor frecuentemente es una implementación propietaria y cerrada. La integridad de los datos es confiada al DBMS. Una desventaja potencial puede ser contar con soluciones propietarias basadas en la arquitectura de un vendedor, no escalables, perdiendo flexibilidad en la distribución de funciones y datos.

Aplicaciones de tres capas.

Las funciones de las aplicaciones (lógica de presentación, del negocio y de datos), pueden ser distribuidas en múltiples procesadores a lo largo de la red. Con esto se obtienen servidores de aplicaciones que proporcionan funciones disponibles a múltiples procesos cliente usando diferentes formas de interactuar con el usuario.

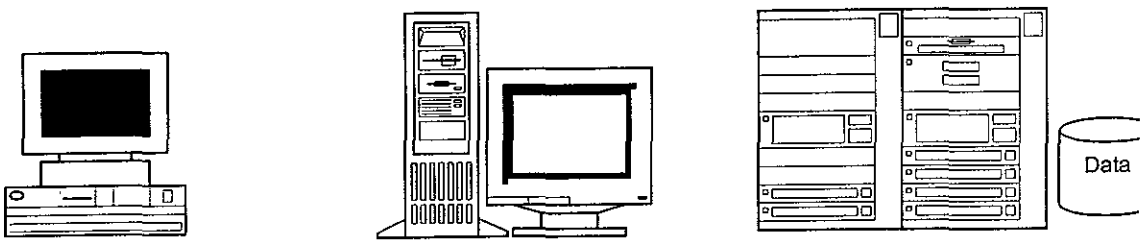


Figura 1.4.3. Esquema Cliente Servidor de tres capas.

Los recursos de procesamiento están distribuidos verticalmente. Normalmente el primer nivel es asignado al sistema de computo mas poderoso y controla los datos corporativos. El segundo nivel contiene servidores de redes locales con doble función: cliente del sistema en el primer nivel y servidor de las estaciones de trabajo de la red local asociada, las cuales ocupan el tercer nivel de la aplicación. Este modelo de aplicación es sumamente flexible ya que puede crecer tanto horizontal como verticalmente, añadiendo sistemas de computo al primer nivel en el primer caso, o redes locales y servidores al segundo nivel. La aplicación en tres capas proporciona un alto grado de flexibilidad en la distribución de datos y funciones. Las aplicaciones pueden ser escalables y abiertas (de acuerdo a las herramientas de desarrollo seleccionadas). Un cambio gradual a nuevas tecnologías puede realizarse sin deshacerse de los recursos existentes. Sin embargo, esta arquitectura de sistema es mas compleja de construir y administrar, y se requiere de una mayor inversión en sistemas al hacer su implementación inicial.

Estructura de los sistemas Cliente/Servidor.

Hay dos estructuras comunes de un Sistema Cliente/Servidor: Soporte a Toma de Decisiones y Proceso de Transacciones en Línea. La división de estos tipos de sistemas es simple de diferenciar.

Sistemas de soporte a toma de decisiones.

Los sistemas de soporte a toma de decisiones son regularmente usados por administradores,. Estos sistemas son diseñados para obtener información periódicamente (por ejemplo, cada trimestre) de un sistema independiente. Desde esta fotografía o marco congelado de datos periódicos, un administrador puede entonces generar un resumen del lugar que ha tomado su negocio y de los juicios que puede hacer sobre la siguiente dirección que debe tomar. No es necesario tener la colección de datos en forma instantánea.

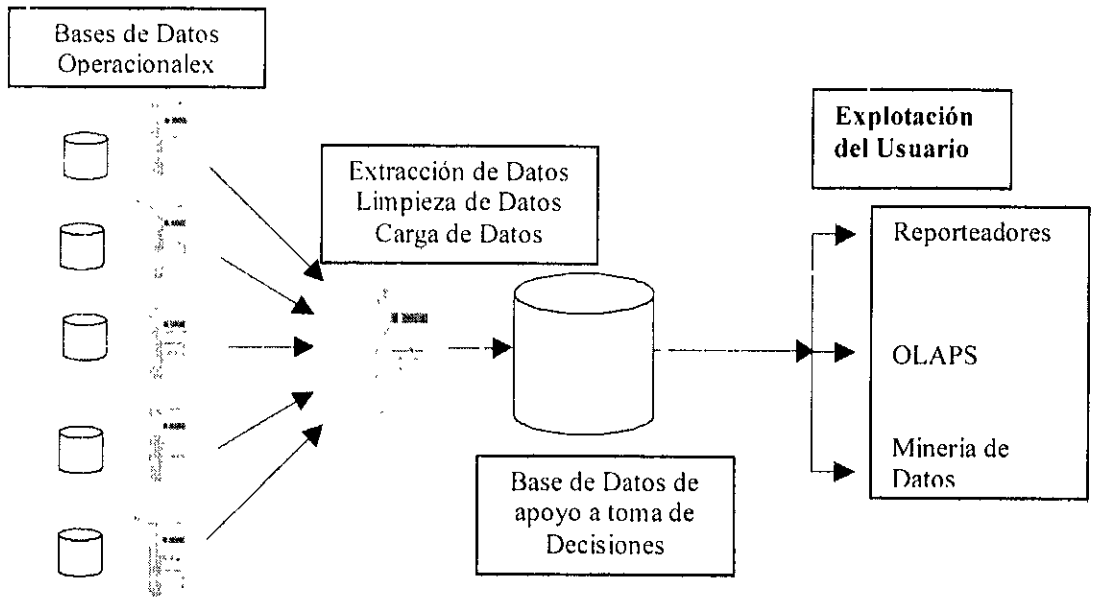


Figura 1.4.4. Esquema General de una Base de Datos para Apoyo a Decisiones.

Por ejemplo, cuando los contadores hacen cierre anual, a menudo ellos agotan muchas semanas coleccionando y analizando los datos apropiados para el resumen final.

Debido a que el reporte es generado en horas extras , es importante que los datos hayan sido trabajados desde una fotografía. Por ejemplo, cuando se reporta sobre niveles de inventario, si el reporte es generado de datos en producción, entonces mostrara diferentes figuras cuando este se genere en diferentes tiempos.

Sistemas de procesos de transacciones en línea.

Estos sistemas son usados para controlar la operación día con día. Tiene la característica de responder al instante (o al menos responder en unos pocos segundos) y deberá ser construido bajo este principio. Un sistema punto de venta es un excelente ejemplo. Cuando un vendedor hace su labor y vende algún producto en el punto de venta, cuando este es entrado al sistema en forma automática sobre la pantalla, entonces el producto es descontado del inventario de la tienda. La computadora deberá reaccionar segura e instantáneamente.

Clasificación Cliente/Servidor según sus peticiones.

Esta se considera una clasificación informal, que surge de las posibles interacciones lógicas que proporcionan las peticiones compartir/ejecutar y en consecuencia el grado de inteligencia que muestra tanto el cliente como el servidor. se parte de la formulación de sistemas simples, hasta sistemas complejos, pasando por una diversidad de sistemas intermedios.

- **Sistemas de peticiones básicas.** En este sistema el cliente genera peticiones simples y básicas para la evolución de su tarea, un ejemplo típico de estos sistemas son los que constituyen los sistemas de archivos, para los cuales las peticiones se refieren a operaciones básicas de archivos: abrir/cerrar, leer/escribir, crear/borrar archivos o subdirectorios, etc. Los sistema como Netware, NFS, PC LA, LAN Manager, etc., mantienen este esquema de operación.
- **Sistema de petición de servicios específicos.** Estos sistemas son mas claros y convincentes del modelo Cliente/Servidor, el cliente solicita un servicio especifico y

el servidor se esmera en proporcionar tal servicio con la mejor funcionalidad y desempeño. Los servidores de fax, telex, correo y módem son ejemplos cotidianos de este sistema. Es importante mencionar que estos servidores específicos hacen uso del servidor de archivos, comportándose como cliente que generan peticiones básicas para lograr su cometido, podemos decir que son proveedores de terceros quienes ofrecen estos servicios.

- **Sistemas de peticiones reiterativas.** Aquí, para que el cliente pueda obtener un servicio, necesita generar y remitir al servidor un conjunto finito de peticiones (cada petición posterior a la primera, causada por respuestas sugerentes). Esta categoría tienen los sistemas servidores de bases de conocimientos son sus mas genuinos representantes, los fabricantes de esqueletos (shells) comerciales ofrecen esta funcionalidad.

- **Sistemas de peticiones complejas**. El cliente formula la petición con un alto grado de abstracción, lo cual proporcionara un procesamiento complejo por parte del servidor, los sistemas servidores de base de datos se adecuan perfectamente a esta categoría. Las peticiones típicas se refieren a consultas/actualizaciones (query/update) especifica de alguna base de datos.

- **Sistema de peticiones cooperativas**. Una petición determinada desencadena un conjunto de peticiones cooperativas entre los diversos proveedores. Esta situación de interacción múltiple requiere de un comportamiento Cliente/Servidor en cada sitio servidor y de un mayor grado de inteligencia en el sitio cliente. Este esquema funcional es el indicado para sistemas de base de datos distribuidas, base de datos multi dimensionales y automatización de oficinas.

Ventajas y desventajas del modelo Cliente/Servidor.

La implementación del Modelo Cliente/Servidor presenta una serie de características que pueden considerarse ventajas comparadas con los modelos tradicionales de computo. Sin embargo, también presenta desventajas en la etapa actual de su evolución . La consideración de ventaja o desventaja de estas características depende de la situación de la empresa y del modelo Cliente/Servidor que se haya implementado.

A continuación se presenta un resumen de ambas características, aun cuando ya se haya hecho mención de algunas a lo largo de este capítulo:

Ventajas.

- Aumento de la productividad y disminución de tiempos y costos de entrenamiento de usuarios al usar estaciones de trabajo con interfaz gráfica.
- Integración de aplicaciones desarrolladas internamente con programas comerciales.
- Flexibilidad en la organización al distribuir datos y aplicaciones en la red, ponerlos mas cerca del usuario y poder realizar funciones sin necesidad de modificar las aplicaciones.
- Mejor aprovechamiento de los recursos de computo ya que cada tarea se ejecuta en la plataforma mas apropiada.
- Acceso a la información desde y cuando el cliente la necesite.
- Facilidad de modificar la operación de la organización sin necesidad de hacer grandes inversiones y grandes planes.
- Flexibilidad de uso y migración de plataformas de hardware, herramientas de desarrollo y sistemas operativos.

Desventajas.

- Las desventajas principales del modelo Cliente/Servidor se basan en que actualmente se encuentra en una etapa de madurez no totalmente establecida.

Así como tampoco son del dominio de la comunidad informática que en su mayoría trabaja utilizando estructuras organizacionales y métodologías tradicionales para desarrollo de aplicaciones.

- Las tecnologías base de Cliente/Servidor son menos maduras en comparación de las utilizadas en el modelo centralizado y el de redes locales, en los aspectos de herramientas de desarrollo, sistemas operativos, protocolos de comunicación, protocolos de bases de datos y software que permita a convivencia de aplicaciones en un ambiente heterogéneo. (middleware).
- El área de desarrollo de aplicaciones tiene que cambiar su enfoque de sistemas basados en un host así como sus métodologías de desarrollo, procesos de administración de proyectos acompañado de programas continuos de capacitación en las nuevas tecnologías.
- La administración del servicio de proveedores externos (de equipos , programas comerciales, desarrolladores de aplicaciones) puede ser mas compleja al incrementarse las plataformas de hardware y software disponibles no propietarios y aunando a la novedad de las tecnologías involucradas, el control del servicio se dificulta.
- El cambio organizacional podría generar conflictos en las áreas involucradas al tratar de deshacerse nichos de poder y posiciones políticas. La común resistencia al cambio es uno de los principales factores a considerar.

EL futuro del modelo Cliente/Servidor.

Aun cuando el modelo Cliente/Servidor se ha convertido en un modelo usual de proceso de información, los avances tecnológicos en diversos aspectos relacionados con Cliente/Servidor y los resultados obtenidos por las empresas que lo han implementado harán crecer la confianza de los directivos de sistemas para adoptar esta tecnología como la base para el desarrollo de sus aplicaciones. Sin embargo, los alcances de este modelo no están limitados al ámbito de la empresa.

No solo se espera que los usuarios disfruten de las ventajas de interfaces gráficas, y opcionalmente orientados a objetos, servidores distribuidos a lo largo de la red de la empresa que proporcionen un servicio determinado en un ambiente abierto, aplicaciones desarrolladas con lenguajes orientados a objetos, bases de datos distribuidas utilizando lenguajes estándares para su acceso.

El alcance que se lograra en un futuro con el modelo Cliente/Servidor va mas allá de las fronteras de la organización. Se pretende crear un universo de la información sin barreras tecnológicas, ni organizacionales, en donde, se pueda tener acceso a la información sin importar donde resida esta, el formato que tenga, ni el método como se ingreso al sistema desde cualquier punto de la red. Esto implica el uso de tecnología de redes de área amplia (WAN) que permita conectar a miles de usuarios, donde cada uno puede ser un cliente y un servidor a la vez, accesando información de distintas organizaciones y empresas, solicitando diversos servicios, solamente limitados por las restricciones de seguridad. Todo esto desde la comodidad de la estación de trabajo.

La "imagen única del sistema" se volverá una realidad a nivel global. Los bajos precios y el alto desarrollo tecnológico de las estaciones de trabajo las hará accesibles para cualquiera. El volumen y la complejidad de transacciones, además del valor agregado de los datos sobre los cuales se operara, crea la necesidad de habilitar nuevas tecnologías como:

Procesos de transacciones sofisticadas. Se requerirá de transacciones anidadas que puedan cruzar la red, transacciones que se ejecuten durante largos periodos de tiempo, mientras viajan de un servidor a otro, transacciones en cola para operaciones negocio a negocio seguras.

Agentes viajeros. Los consumidores tendrán agentes personales que velaran sus intereses. Los negocios contaran de agentes para vender sus productos/servicios por la red. Los agentes rastrearan las estadísticas, analizaran las tendencias del mercado.

Esta tecnología incluye mecanismos de escritura de plataforma cruzada, mecanismos de flujos de trabajo y una infraestructura que permita al agente residir en cualquier parte de la red.

Administración de datos sofisticados. Desde cualquier sitio de la red podremos crear, almacenar, visualizar y editar documentos compuestos (texto, imágenes, hojas de calculo). Los super servidores proporcionaran depósitos para el almacenamiento y distribución de cantidades masivas de documentos compuestos sin olvidar por supuesto información con estructura de base de datos. Existen cuatro aspectos tecnológicos en que se basaran las aplicaciones Cliente/Servidor para lograr esto:

- **Bases de datos SQL.** Actualmente los servidores SQL dominan el campo de Cliente/Servidor. Aun cuando existen diversos distribuidores de SQL, el uso (en un ambiente heterogéneo de base de datos) de un común denominador para manipulación de información como SQL, facilitara enormemente la convivencia entre aplicaciones de diferentes compañías.

- **Monitores TP.** Las aplicaciones de misión critica (transferencias de dinero, actualización de inventarios, etc.) por la importancia de su naturaleza requieren la administración de los procesos que operan sobre la información mejor conocidos como transacciones. Las transacciones se han convertido en una de las partes fundamentales en el diseño de Sistemas Operativos distribuidos ya que el monitor de transacciones asegura la consistencia de la información en un ambiente altamente concurrente. X/Open y OMG (Object Managment Group) han creado normas para definir la forma en que los monitores de transacciones interactúan con las aplicaciones, los administradores de recursos y con otros monitores.

- **Software para grupos (groupware).** Este tipo de software tiene cinco componentes unidos para apoyar el trabajo en grupo, los cuales son: Administración de documentos multimedia, flujo de trabajo, correo electrónico, Conferencias y programación de agenda.

La importancia de estas herramientas en el modelo Cliente/Servidor es que en su conjunto proporcionaran el ambiente para el desarrollo de aplicaciones y procesos del negocio que se adaptan mejor a la naturaleza de los procesos y de la organización, para obtener los mejores resultados del modelo para Cliente/Servidor. Y mas aun, en algunos productos ya en el mercado se han implementado características para interactuar con bases de datos vía SQL.

Objetos distribuidos. La tecnología de objetos distribuidos promete hacer los sistemas Cliente/Servidor mas flexibles. Esto es debido a que, por la propia naturaleza de los objetos, estos pueden viajar a través de la red ocultando aspectos como datos, lógica del negocio, manejo de los mismos y de sus recursos. Para que los objetos de diferentes vendedores puedan interactuar a lo largo de la red y en los distintos sistemas operativos , el consorcio OMG (Object Managment Group) formado por mas de 400 compañías, comercializadoras de tecnología de objetos, ha trabajado en la especificación de la arquitectura de un software abierto que permita que los componentes de objetos hechos por diferentes compañías puedan convivir. Es importante resaltar que a diferencia con lo ocurrido con otras tecnologías, en donde son las fuerzas del mercado las que dirige su desarrollo y normatividad, en la tecnología de objetos son los acuerdos salidos de OMG los que guían su desarrollo. Hace algún tiempo la OMG aprobó una serie de especificaciones llamados CORBA 2.0 (Common Object Request Broquer) que definen una estructura basada en TCP/IP para la interacción entre ORB's (Objects Requests Broker). También especifica opcionalmente un servicio de comunicación entre ORB's basado en DCE entre otras especificaciones.

La OMG ha creado importantes alianzas para asegurar la aceptación general de sus estándares. Algunas de las instituciones con las que tiene acuerdos o esta en vías de tenerlos son:

- ODMG (Object Database Managment Group). La definición del servicio de persistencia de CORBA esta alineado a las especificaciones de ODMG para bases de datos orientadas a objetos.

- X/Open. Los servicios de transacción de objetos de CORBA pueden interoperar con las definiciones de transacciones orientadas a procedimientos de X/Open. Colaboración en la definición de interfaces de administración de sistemas basados en ORB.
- CI Labs (consorcio de compañías responsables de OpenDoc) selecciono a CORBA como su modelo objeto para la intercomunicación de componentes.
- Taligent y Open Step tienen puertas de acceso CORBA para sus comunicaciones externas con objetos.
- Microsoft propuso a OMG la creación de una compuerta de comunicación OLE-CORBA.

Por las características de las tecnologías mencionadas anteriormente al parecer el candidato mas viable a alcanzar los objetivos de un ambiente Cliente/Servidor mundial es la tecnología de distribución de objetos, que una vez extendida puede absorber las otras y lograr la administración lógica (entidades de software) que viajaran en la red.

1.5. Descripción del Problema.

Se pretende incorporar un servicio de información financiera a una plataforma digital estandarizada. Para este efecto se debe desarrollar un **traductor de protocolo** que traduzca la información de su formato original al formato estandarizado. Esta fuente de información transmite tres tipos básicos de datos:

- Noticias
- Páginas
- Información financiera

Las noticias transmitidas son principalmente sobre temas financieros, una vez que se recibe una noticia se guarda en una base de datos hasta que se solicita. Las noticias nunca cambian. Las páginas transportan información que no suele cambiar muy frecuentemente tal como el precio y cotizaciones de monedas. La información financiera es constituida por hechos y posturas de valores de todo tipo. Esta información se caracteriza por ser sumamente cambiante e impredecible.

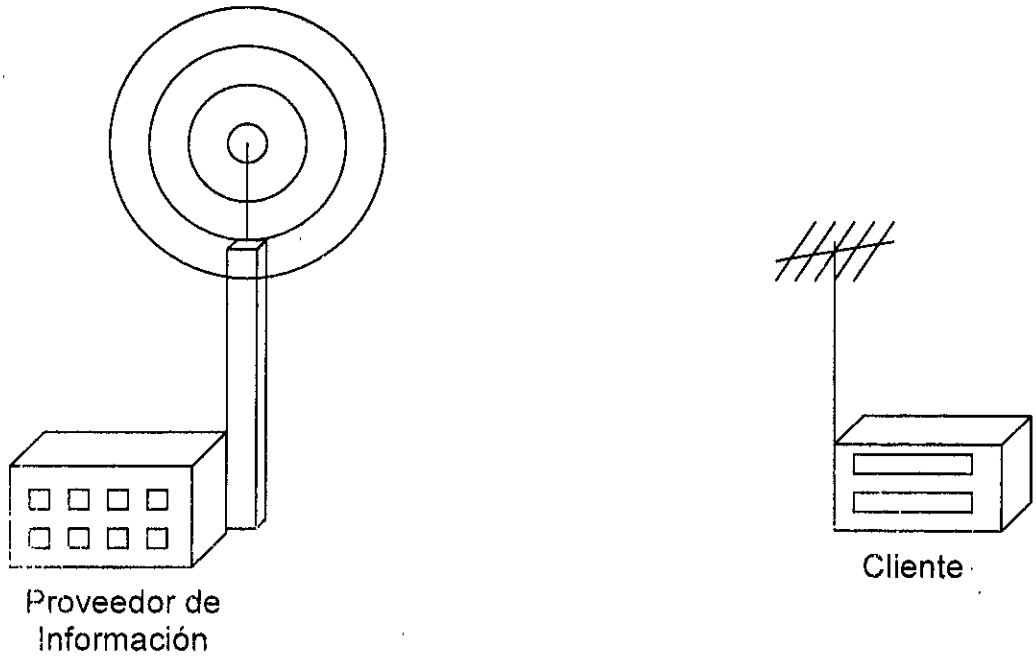
El detalle de los formatos de transmisión se tratará mas adelante.

1.5.1. Formas de Distribución de la Fuente de Información.

Actualmente la fuente de información que se pretende incorporar a la plataforma digital distribuye su información usando dos tecnologías: radio y líneas directas. Cada una de ellas tiene ventajas y desventajas asociadas. A continuación se describirán en detalle.

1.5.1.1. Distribución Vía Radio.

En este caso la información llega en una señal de FM que es captada por una antena aérea.



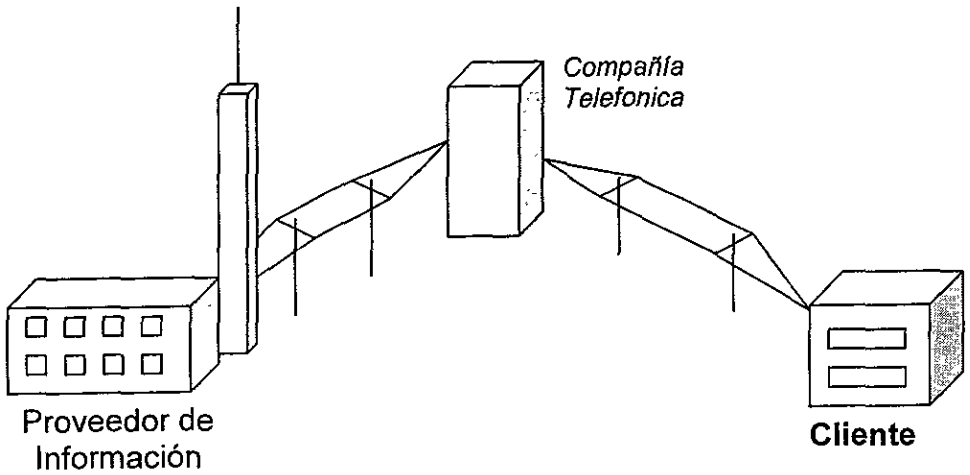
Ventajas de la distribución vía radio.

- El usuario final no necesita rentar una línea privada que suele ser costosa.
- Se puede ocupar una sola antena de FM para un número muy grande de usuarios.
- En algunos casos es posible recibir la señal en puntos apartados de grandes núcleos urbanos sin necesidad de hacer grandes inversiones en infraestructura.

Desventajas de la distribución vía radio.

- La calidad de la señal varia de acuerdo a factores topográficos y atmosféricos. Estos factores están completamente fuera del control del usuario y puede, en algunos casos, impedir que la instalación sea viable.
- La potencia de la señal puede no ser constante.
- Hay una gran probabilidad de error.

1.5.1.2. Distribución Vía Línea Directa.



En este caso el usuario debe rentar una línea directa de alguna compañía telefónica.

Ventajas de la distribución vía línea directa.

- La calidad de la señal transmitida es homogénea y casi completamente independiente de factores atmosféricos y topográficos.
- La potencia de la señal siempre es constante.
- La probabilidad de tener errores durante la recepción es menor.

Desventajas de la distribución vía línea directa.

- El precio de una línea directa suele ser muy elevado.
- Se necesita una infraestructura muy especializada lo cual impide el acceso a zonas apartadas de núcleos urbanos importantes.
- La línea directa solo puede transmitir el volumen de información que su ancho de banda le permita.

En ambos casos se usa un módem decodificador que traduce la señal de su formato original y la hace disponible usando un puerto serie incorporado en el aparato. El puerto también permite la programación del dispositivo para adecuarlo a varias frecuencias de FM. Cada usuario individual debe tener uno de éstos módems conectado a un puerto serie de su computadora. La velocidad de transmisión máxima es de 9600 bits por segundo.

Los proveedores de información ofrecen un sistema que configura y recupera la información del módem.

El sistema en general tiene varias desventajas que lo hacen poco rentable desde el punto de vista del usuario y de la compañía distribuidora.

Como cada usuario debe tener un módem separado, la falla de este aparato hace indispensable que un representante de la compañía distribuidora lo reemplace haciendo una visita al sitio. La probabilidad de fallo individual aumenta en instalaciones con muchos usuarios, por esta razón, la compañía debe mantener un departamento especializado en la atención a usuarios que resulta sumamente costoso. Este departamento no solo debe estar a cargo de la configuración de hardware, también debe atender llamadas por fallo en la configuración del software.

Comó es evidente, este costo de un departamento como el descrito debe ser absorbido por la compañía distribuidora quien mas tarde lo refleja en el precio de la información para sus clientes.

1.5.2. Plataforma Suministrada por el Fabricante.

Este sistema permite la visualización de la totalidad de información contenida en la fuente a partir de menús descendentes, cajas de texto y otros elementos comunes en las GUI ***(Graphical Users Interface – Interface Gráfica de Usuario)*** actuales. Con esta herramienta el usuario puede consultar datos en tiempo real o históricos sobre información financiera y noticias, las páginas únicamente en tiempo real.

La principal desventaja de esta terminal es que la información no es transportable fuera de ella. El usuario se limita a *verla* y no puede comunicarla a otras aplicaciones especializadas en su análisis.

Otra desventaja es que la aplicación mantiene bases de datos locales en las maquinas del usuario, lo que provoca que la información no sea homogénea en toda una red de usuarios. Si un usuario mantiene su terminal apagada durante un tiempo, es casi seguro que al prenderla nuevamente no tendrá la capacidad de acceder al mismo volumen de información a la que podría acceder un usuario que la ha mantenido encendida todo el tiempo.

Por otro lado, este sistema consta de múltiples procesos corriendo simultáneamente. Estos procesos mantienen la comunicación con el módem, acceden a las bases de datos locales, procesan las instrucciones del usuario y despliegan la información. Tanto movimiento hace que la plataforma sea relativamente inestable. Conversaciones con los usuarios han dejado en claro que el sistema adolece de inestabilidad al correrse simultáneamente con otras aplicaciones.

1.5.3. La Fuente de Datos.

Para extraer información de la fuente y darle un tratamiento especial, el proveedor de información ofrece un programa llamado **Fuente de Datos**. Este programa se comunica con la terminal, también proporcionada por el proveedor, y hace disponible la información por uno de los puertos serie de la computadora. Este programa permite configurar todos los parámetros comunes de un puerto serie estándar tales como velocidad, paridad, bits de paro y bits de datos. También lleva un registro de cuantas noticias, páginas, información financiera y errores se han recibido.

Este programa será la base de nuestro sistema porque nos permitirá la extracción de la información que de otra manera seria imposible. En platicas anteriores, se exploro la posibilidad de conectar a nuestro sistema directamente con el módem decodificador para hacer mas simplificar el diseño en general y quitar del esquema la computadora que mantiene ejecutándose la terminal suministrada por el fabricante. Desafortunadamente los representantes de la fuente de información consideraron que hacer publico su formato de distribución comprometería la seguridad de su información y dieron esta opción como no valida.

1.5.4. Conclusiones.

En resumen el sistema actual adolece de graves desventajas que se reflejan en costos para los usuarios de la información. Nuestra labor consiste en trasladar dicha información a un formato seguro y eficiente para que después sea transmitida a terminales de nuestra plataforma digital para que el usuario obtenga provecho de ella.

Se considera que el volumen de información total que se debe procesar excede los 10 millones de Bytes y la variabilidad de esta cifra es grande. Dependiendo de la actividad económica en el país, la cantidad de información recibida puede ser muy pequeña o enorme. Además el retraso sufrido en la traducción desde su formato nativo al formato de la plataforma digital debe ser imperceptible para el ser humano.

Estas premisas influyen directamente en el diseño. Lo primero en considerar es que no se pueden usar estructuras de datos de tamaño fijo para almacenar la información. La razón es que, por ejemplo, si se usara un arreglo de X elementos para guardar las actualizaciones de un cierto grupo de valores, habrían días en los que la mayoría de los elementos estarían vacíos porque la actividad económica fue pequeña. Por el otro lado, no podemos arriesgarnos a que en un período de gran actividad económica, alguna información valiosa se pierda por no tener suficiente espacio para almacenamiento.

Además el sistema de almacenamiento de información debe ser sumamente rápido para guardar y recuperar la información. Un retraso de un segundo se considera inaceptable al hacer actualizaciones. Esto hace muy poco viable el uso de dispositivos de almacenamiento masivo como discos duros, al menos para el almacenamiento de la información financiera mas valiosa.

Dadas estas condiciones se hace evidente que se debe trabajar en la memoria RAM en la mayoría de los casos. También que se debe ocupar memoria dinámica que se ajuste automáticamente al volumen de información recibida.

1.6. Características, Ventajas y Desventajas de la Comunicación Serie y Paralela.

Interconexión de comunicación en serie.

Un dispositivo I/O puede transferir la información binaria en paralelo o en serie. En la transmisión en paralelo, cada bit de información usa una línea separada de manera que los n bits de un componente pueden ser trasmitidos simultáneamente. Por ejemplo, un dispositivo periférico paralelo puede trasmitir una palabra de 16 bits, todos al mismo tiempo, a través de dos buses de 8 bits de la interconexión periférica en paralelo.

En la transmisión en serie, los bits de una palabra son trasmitidos en secuencia, bit a bit a través de una sola línea. La transmisión en paralelo es más rápida pero requiere muchas líneas. Esta se usa para distancias cortas y donde la velocidad es importante. La transmisión en serie es lenta pero menos costosa ya que solamente requiere dos líneas. La información binaria trasmitida desde terminales remotos a través de cables telefónicos u otro medio de comunicación es del tipo serie porque sería muy costoso suscribir o rentar un gran número de líneas. Ejemplos de terminales de comunicación son los teletipos, los terminales de CRT y los dispositivos de cómputo remoto.

La información binaria en serie trasmitida a una terminal consiste de caracteres de códigos binarios. Los caracteres pueden representar información alfanumérica o caracteres de control. Los caracteres alfanuméricos son trasmitidos como un texto e incluyen las letras del alfabeto, los dígitos decimales y un número de símbolos gráficos tales como el punto, el más y la coma. Los caracteres de control se usan para la distribución de la impresión o para especificar el formato del mensaje trasmitido. El número de bits asignados a cada código de caracteres puede estar entre cinco y ocho dependiendo de la terminal.

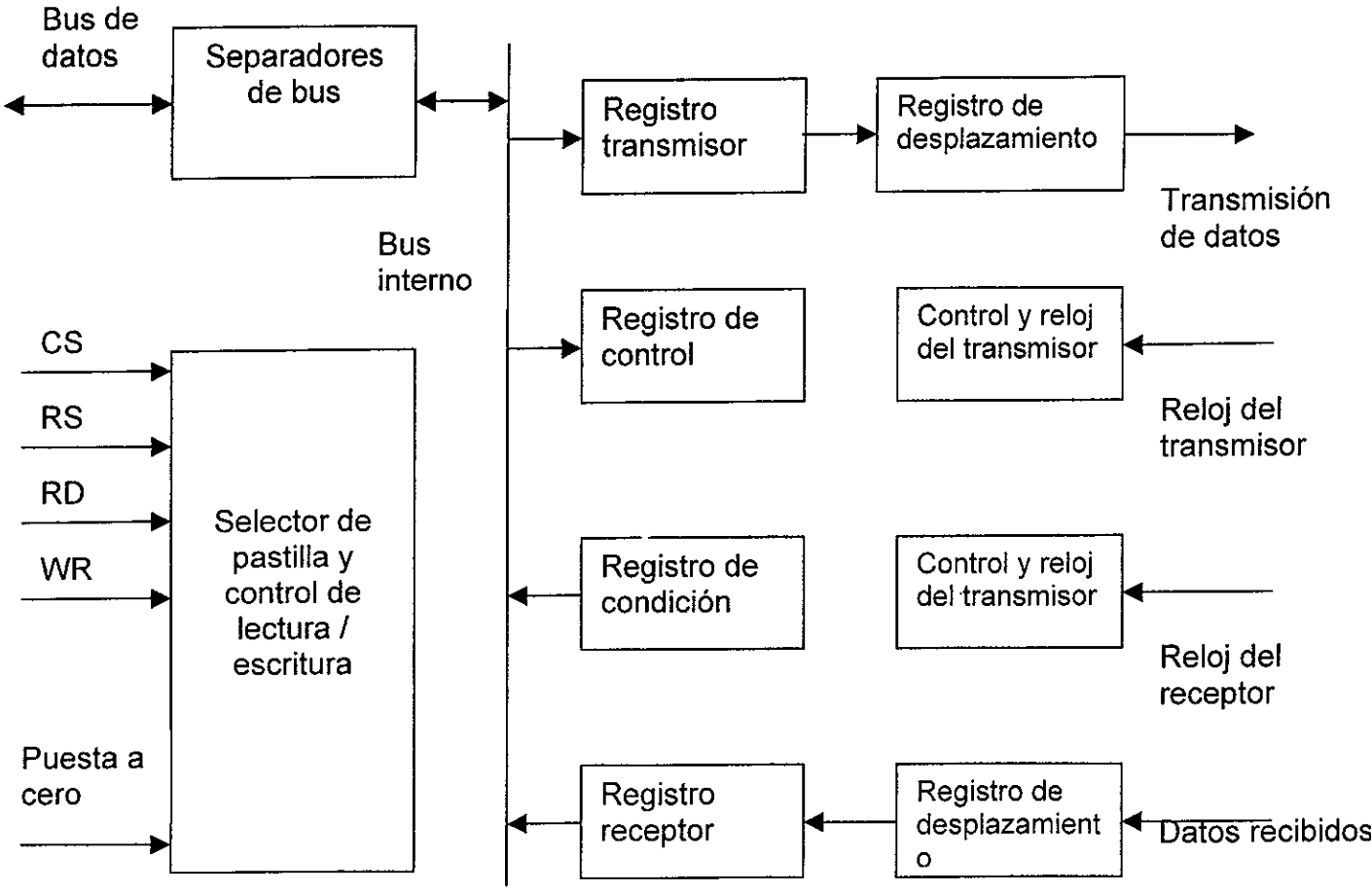


Figura 1.6.1. Diagrama de bloque típico de una interconexión de comunicación en serie.

El diagrama de bloque de una interconexión de comunicación en serie se muestra en la figura 1.6.1. Este funciona como un transmisor o como receptor y puede ser programado para operar en una variedad de modos de transmisión. La interconexión se inicia para un modo de trasferencia en serie particular por medio de un byte de control, el cual se carga a su registro de control. El registro de transmisión acepta un byte de datos del microprocesador a través del bus de datos. Este byte se transfiere a un registro de desplazamiento para una transmisión en serie. La parte de recepción recibe la información de serie en otro registro de desplazamiento y cuando se acumula un byte de datos completo, éste se transfiere al registro receptor. El microprocesador puede seleccionar el registro receptor para leer el byte por medio del bus de datos. Los bits del registro de condición se usan para poner a uno los indicadores de entrada y salida y para detectar ciertos errores que pueden ocurrir durante la transmisión.

El microprocesador puede leer el registro de condición para constatar el estado de los bits indicadores y para determinar si cualquier error puede ocurrir.

Las líneas de selección de lectura/escritura, se comunican con el microprocesador. El terminal de entrada de selección (CS) se usa para seleccionar la interconexión. El selector de registro (RS) se asocia con los controles RD y WR. Dos registros aceptan información durante una operación de escritura y los otros dos suministran información durante la operación de lectura. El registro seleccionado es entonces una función de la condición de RD y WR como se muestra en la tabla que acompaña el diagrama.

El transmisor y receptor tiene una entrada de reloj para sincronizar la razón de los bits al cual se transfiere la información en serie. La línea de datos de transmisión se conecta a un receptor remoto y la línea de datos recibidos vienen de un transmisor remoto Si el reloj está conectado al terminal remoto, se dice que la transmisión es sincrónica. Si el reloj no está compartido con el terminal remoto se dice que la transmisión es asincrona.

En el modo serial sincrónico de transmisión el transmisor remoto y local y el receptor comparten el reloj común. Los bits son enviados desde el transmisor a intervalos iguales de tiempo determinados por el ritmo de los pulsos de reloj. Como el receptor comparte un reloj común con el transmisor, éste acepta los bits al mismo ritmo del reloj. En la transmisión asíncrona, las dos partes no comparten un reloj común. Los pulsos de reloj del transmisor de interconexión y del receptor son alimentados al ritmo del reloj local que especifica la ruta de trasferencia del terminal de comunicación remoto al cual está conectada la interconexión.

Un problema común asociado con una transmisión en serie trata de la demarcación de caracteres en una cadena continua de bits. El transmisor y receptor pueden estar programados para reconocer el número de bits en cada carácter en el terminal remoto. Permanece allí el problema de detectar el primer bit en cada carácter de manera que una cuenta pueda comenzar a demarcar el siguiente carácter.

La forma en que los caracteres estén demarcados en la transmisión en serie, depende de si el modo de trasferencia es sincrónico o asincrónico.

En la transmisión en serie sincrónica, un carácter de control de comunicación, llamado carácter de sincronismo, se escoge para que sirva como agente de sincronización entre el transmisor y el receptor. Por ejemplo, cuando un código ASCII de 7 bits se usa con un bit de paridad impar en la posición más significativa, el carácter sincrónico asignado tiene el código de 8 bits éste envía varios caracteres sincrónicos y luego envía el mensaje actual. La cadena continua inicial de bits aceptada por el receptor acepta un bit más, rechaza el bit anterior de mayor orden y comprueba de nuevo los últimos ocho bits recibidos por un carácter de sincronismo. Esto se repite después de cada pulso de reloj y bit recibido hasta que se reconozca un carácter de sincronismo. Una vez que se haya detectado un carácter de sincronismo, el receptor habrá demarcado un carácter. De aquí en adelante el receptor cuenta cada ocho bits y los acepta como un solo carácter. Comúnmente el receptor comprueba dos caracteres de sincronismo consecutivos para evitar cualquier duda de que el primer carácter de sincronismo no ocurrió como un resultado de una señal de ruido en la línea. Sin embargo, cuando el transmisor está inactivo y no tiene ningún mensaje que enviar, éste envía una cadena continua de caracteres de sincronismo. El receptor reconoce todos los caracteres de sincronismo como una condición para sincronizar la línea y pasa a un estado latente sincrónico. En este estado, las dos unidades mantienen sincronismo mientras no se esté comunicando ningún mensaje.

El procedimiento normal antes descrito indica que el transmisor en una interconexión de comunicación sincrónica se ha diseñado para enviar caracteres de sincronismo al comienzo de la transmisión y también cuando no hay caracteres disponibles del microprocesador. El receptor en una interconexión de comunicación sincrónica debe demarcar ocho bits consecutivos en caracteres y debe poder identificar ciertos códigos de caracteres tales como el carácter de sincronismo. Cuando el receptor reconoce los caracteres de sincronismo, se usan éstos para mantener el sincronismo con el transmisor, pero los caracteres de sincronismo no se envían al microprocesador.

El procedimiento normal para demarcar caracteres durante la transmisión asincrónica es enviar al menos dos bits adicionales con cada carácter. Estos bits adicionales son llamados bits de paro y de comienzo. Por ejemplo, una unidad de teletipo usa un código de carácter de 8 bits pero envía 11 bits por cada carácter trasmitido. El primer bit es el bit de comienzo. Este está seguido por los 8 bits del carácter y luego por los dos bits de parada. La convención en este terminal es que permanece en el estado 1 cuando no se trasmiten caracteres. El primer bit es siempre 0 y representa el bit de comienzo para indicar el principio de un carácter. El receptor puede detectar el bit de comienzo cuando la línea va de 1 a 0. Un reloj en el receptor conoce la razón de trasferencia y el número de bits del carácter que se esperan. Después de que se reciban los 8 bits de caracteres, el receptor comprueba los dos bits que están siempre en el estado 1. La longitud de tiempo que la línea permanece en el estado de 1 (parada) depende de la cantidad de tiempo requerido para que la terminal se resincronize. Otros terminales usan justamente un bit de paro y algunos uno y medio tiempo de bit para el período de parada. La línea permanece en el estado 1 hasta que se trasmita otro carácter. La figura 1.6.2 muestra los 11 bits del carácter típico del teletipo, que requiere dos bits de paro. Después de que los dos bits de parada han sido trasmitidos, la línea puede ir a 0, indicando un bit de comienzo para un nuevo carácter. La línea permanecerá en el estado 1 si no sigue otro carácter inmediatamente.

El procedimiento normal antes descrito indica que el transmisor en una interconexión de comunicación asincrónica agrega los bits de comienzo y parada antes de la transmisión en serie. El receptor debe reconocer los bits de comienzo y parada para demarcar el carácter. El receptor puede aislar los bits de información para transferir al microprocesador.

Los procedimientos de demarcación normalizados son incorporados con una interconexión de comunicación en serie. La interconexión de comunicación en serie puede ser solamente asincrónica, sólo sincrónica o ambas cosas.

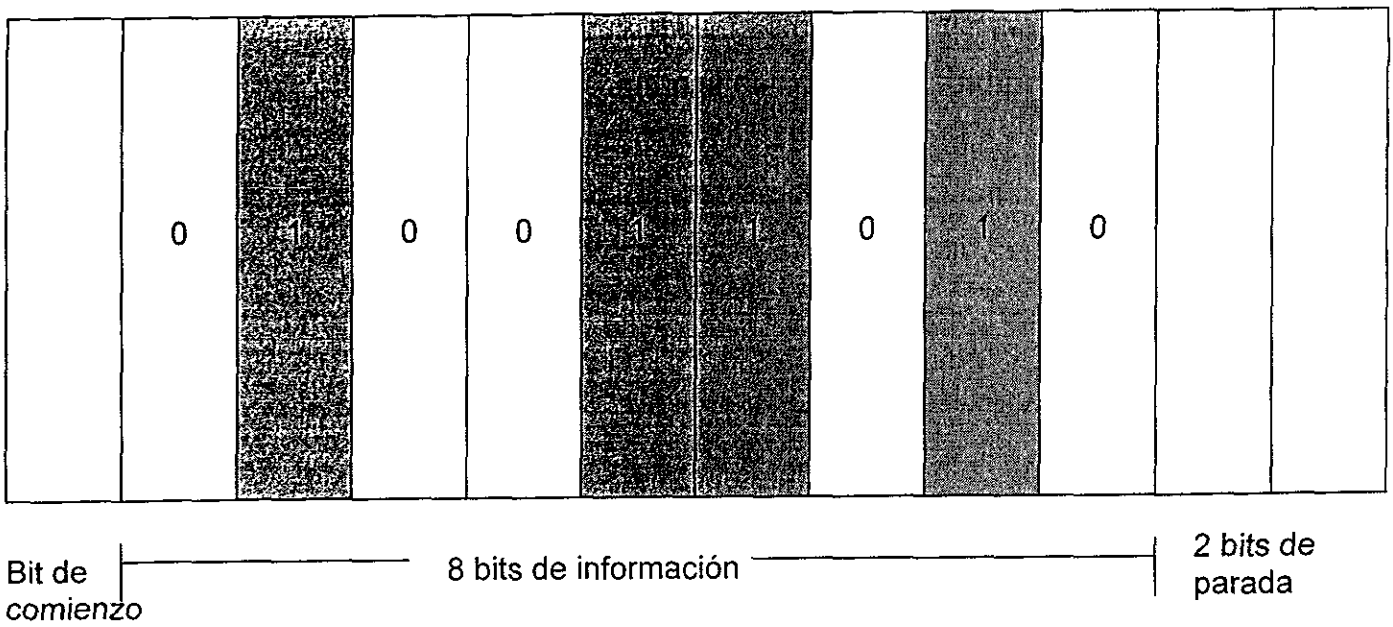


Figura 1.6.2. Transmisión asincrónica en serie de un carácter.

Software de comunicaciones asíncronas.

La mayoría de los grandes sistemas de computadoras constan de un CPU conectado a varias terminales. A la hora de utilizar una aplicación en la computadora, un usuario se comunica con ella mediante una terminal. En este caso a la computadora se le denomina host.

Una terminal es un dispositivo no inteligente que consta de una pantalla, un teclado y los circuitos necesarios para comunicarse con el host. Sólo puede realizar dos funciones: enviar al host los caracteres introducidos desde el teclado, y mostrar en la pantalla los caracteres que recibe del host.

Las terminales en muchos casos han sido sustituidas por computadoras personales que ejecutan software de emulación de terminales. Este sistema ofrece muchas ventajas, la capacidad de capturar los datos recibidos del host, intercambiar información y emular varios tipos de terminales sin tener que realizar ni una sola modificación en los equipos. Permite descargar información desde el host y procesarla en forma local.

Hardware de comunicaciones asíncronas.

El hardware de comunicaciones asíncronas para PC puede estar incluido en el equipo o ser incorporado mediante una placa de ampliación. El tipo de interfaz ofrecido por estos dispositivos se denomina interfaz RS-232, denominada también habitualmente puerto, puerto COM o puerto serie.

El estándar original del PC define dos puertos de comunicaciones, COM1 y COM2. La mayoría de las placas de expansión para PC pueden configurarse como uno de estos dos puertos. Los equipos PC que incorporan un puerto serie lo tienen definido como COM1.

Instalación de módems.

Las telecomunicaciones mediante PC requieren el uso de un módem interno o externo. Un módem realiza la función de convertir la información almacenada en el PC, en un formato que pueda ser transmitido a través de una línea telefónica.

La mayoría de los módems tienen dos conectores telefónicos; uno para la línea que se conecta en la caja de la pared y otro que se conecta a un teléfono. Generalmente, los módems son suministrados con un cable para conectar a la caja de la pared. Si tiene un teléfono se puede conectar al otro conector de módem. Esto permite utilizar la misma línea para transmisiones telefónicas de voz y de datos, aunque no puedan realizarse de forma simultánea.

Todos los módems incorporan un altavoz que permite monitorear el desarrollo de la llamada. Los módems externos disponen de indicadores luminosos de estado, que visualizan la función que están realizando en cada momento.

Módems internos.

Un módem interno es una placa de expansión que ocupa un conector adicional en el PC. Se alimenta de la fuente de poder del PC. Los módems internos no requieren cableado. Basta con disponer del cable telefónico que conecta el módem a la línea telefónica.

Como muchos equipos PC ya disponen de un puerto serie, que generalmente es el COM1, la mayoría de módems internos vienen configurados de origen como COM2. La instalación de cada módem viene descrita detalladamente en el manual suministrado con cada uno de ellos.

Módems externos.

Los módems externos están protegidos por una carcasa. El uso de un módem externo requiere la existencia de un puerto serie en el PC, un cable del puerto al módem y una fuente de alimentación.

Cableado.

Un cable es un conjunto de hilos conductores que, en este caso, se utiliza para transmitir señales eléctricas desde el PC hacia el módem y viceversa.

El tipo de conector usado en el puerto serie de un PC, y en la mayoría de módems se denomina conector tipo D. Existen dos tipos de conectores D: macho y hembra. Los conectores tipo D están rotulados, de modo que cada patilla o ranura tiene un número, del 1 al 9 o del 1 al 25. Generalmente, el conector serie situado en el PC es hembra. Algunos tienen 9 patillas y otros 25.

Instalación de software.

La primera cuestión a determinar, es el número del puerto de comunicaciones correspondiente al módem interno o el puerto serie que se va a utilizar. Al instalar el software de comunicaciones en el PC, el programa debe ser configurado para el tipo de módem conectado y el puerto COM correspondiente.

Direccionamiento de los puertos COM.

Podemos ver la definición de un puerto de comunicaciones COM como la identidad de un puerto. Esta identidad se obtiene de la combinación de una dirección y de un IRQ (número de petición de interrupción). Cada puerto de un PC debe tener una identidad única, es decir, no debe haber más de un puerto con la misma combinación de dirección e IRQ. La dirección le indica al software donde está el puerto.

Dentro de un PC hay varios dispositivos que utilizan señales llamadas interrupciones o IRQ cuando tienen algo que decir. Por ejemplo, un puerto genera una interrupción cuando ha recibido un carácter. Esto le indica al software que hay que dirigirse a ese puerto para recoger el carácter y procesarlo.

La siguiente tabla muestra las direcciones estándar utilizadas para los puertos serie de los equipos PC y PS2.

Puerto	Dirección IRQ para PC	Dirección IRQ para PS2
COM1	03F8h/4	03F8h/4
COM2	02F8h/3	02F8h/3
COM3	03F8h/4	3220h/3
COM4	02F8h/3	3228h/3
COM5		4220h/3
COM6		4228h/3
COM7		5220h/3
COM8		5228h/3

Tabla 1.6.1. Puertos serie para los equipos PC y PS2.

La UART.

La UART (transmisor-receptor asíncrono universal) es un circuito integrado especializado que controla las comunicaciones asíncronas. Se encarga de enviar y recibir los datos, así como del protocolo de comunicaciones con el dispositivo conectado.

La interacción entre el software de comunicaciones y la UART presenta un alto nivel de transparencia. El software le indica a la UART que envíe o reciba caracteres, y que inicie o detenga el flujo de datos.

Mnemónico	Nombre	Origen	Propósito
TXD	Datos transmitidos	PC	Transmite caracteres del PC al módem
RXD	Datos recibidos	Módem	Transmite caracteres del módem al PC
RTS	Solicitud de envío	PC	Usada para control de flujo
CTS	Listo para enviar	Módem	Usada para control de flujo
DSR	Conjunto de datos separado	Módem	Usada para control de flujo
SG	Señal de tierra		Referencia eléctrica para las señales restantes
CD	Detección de portadora	Módem	El módem activa esta señal cuando ha conectado con otro módem.
DTR	Terminal de datos preparado	PC	Algunos programas activan esta señal cuando van a efectuar alguna llamada o están en modo local.
RI	Indicador de llamada	Módem	Se activa cuando se detecta una llamada.

Tabla 1.6.2. Cableado de un puerto serie.

Cableado de un puerto serie.

Aunque los conectores serie tienen 25 patillas, sólo se utilizan 9 de ellas para transmitir señales. Estas 9 señales están relacionadas con la tabla anterior:

Hay dos tipos de dispositivos serie: ETD (equipos terminales de datos) y ECD (equipos de comunicación de datos). Las terminales y los PC son dispositivos ETD, los módems son dispositivos ECD.

Señales de un conector de 25 patillas

Los cables de este tipo están conectados en forma directa. Es decir, la patilla 2 de un extremo se conecta a la patilla 2 del otro, la patilla 3 con la 3, y así sucesivamente. La mayoría de los cables utilizados para conectar un puerto de 25 patillas de un PC a un módem están configurados de la siguiente forma:

Señal	Patilla del PC	Patilla del módem
TXD	2	2
RXD	3	3
RTS	4	4
CTS	5	5
DSR	6	6
SG	7	7
CD	8	8
DTR	20	20
RI	22	22

Tabla 1.6.3. Señales de un conector de 25 patillas.

Señales de un conector de 9 patas.

La configuración de un puerto serie tipo AT de 9 patillas es bastante distinta de la correspondiente a los puertos estándar ETD. Las señales se corresponden con patillas distintas de las que cabría esperar. La siguiente tabla muestra la configuración correcta de un cable para conectar un puerto tipo AT de 9 patillas con un módem con conector de 25 patillas.

Señal	Patilla del PC	Patilla del módem
TXD	3	2
RXD	2	3
RTS	7	4
CTS	8	5
DSR	6	6
SG	5	7
CD	1	8
DTR	4	20
RI	9	22

Tabla 1.6.4. Señales de un conector de 9 patas.

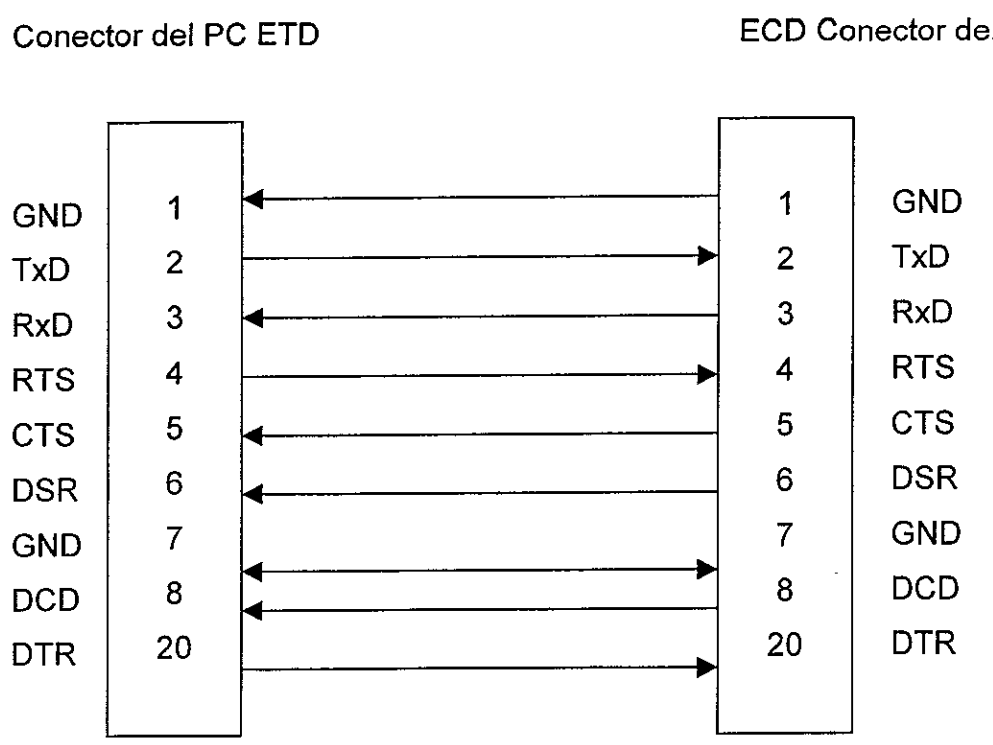


Figura 1.6.3. Cable de módem estándar RS-232 para PC con conector "D" de 25 patas.

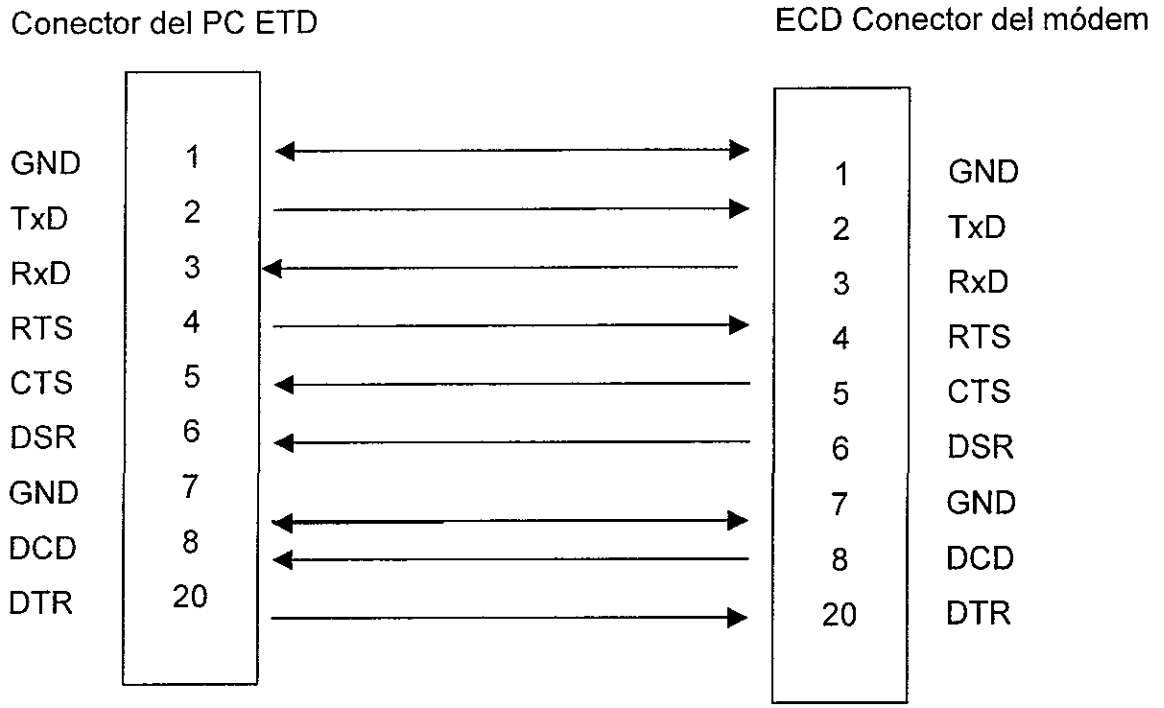


Figura 1.6.4. Cable de módem estándar para PC con conector DB-9 de 9 patas.

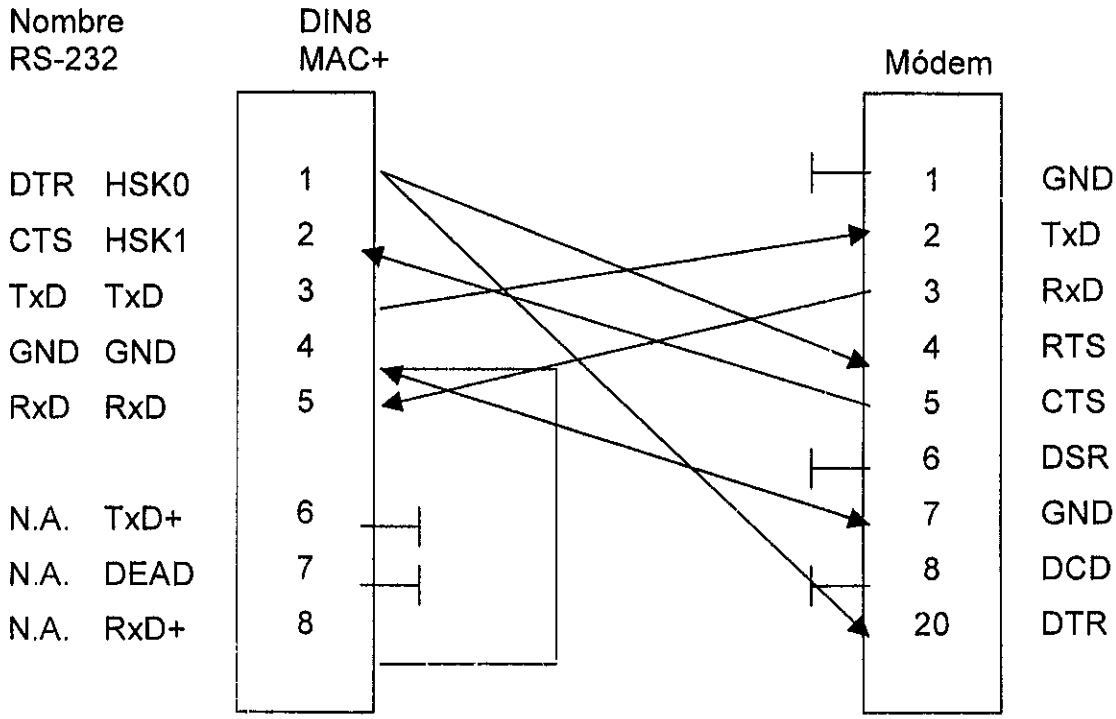


Figura 1.6.5. Cable para módem Machintosh Plus, conector DIN 8 patas.

Cables para conexión directa entre equipos

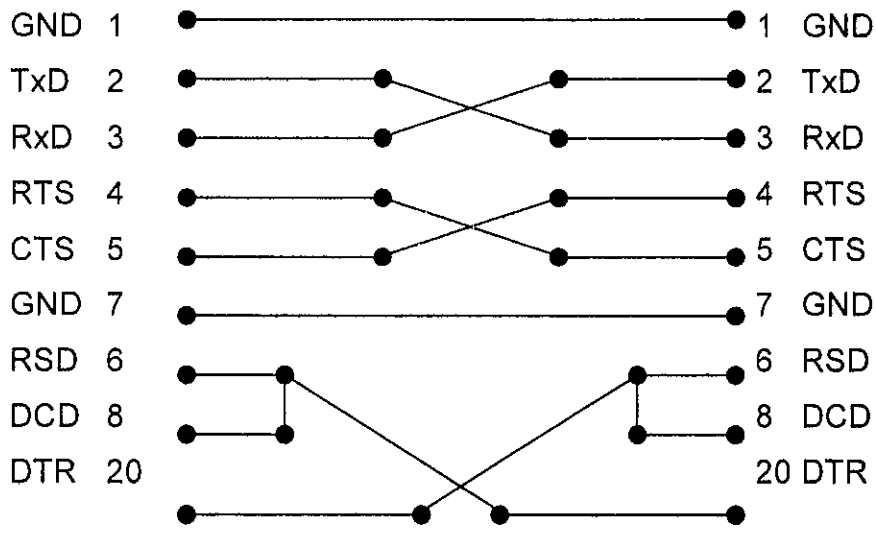


Figura 1.6.6. Cable sin módem, DB-25 patillas a DB-25 Patas.

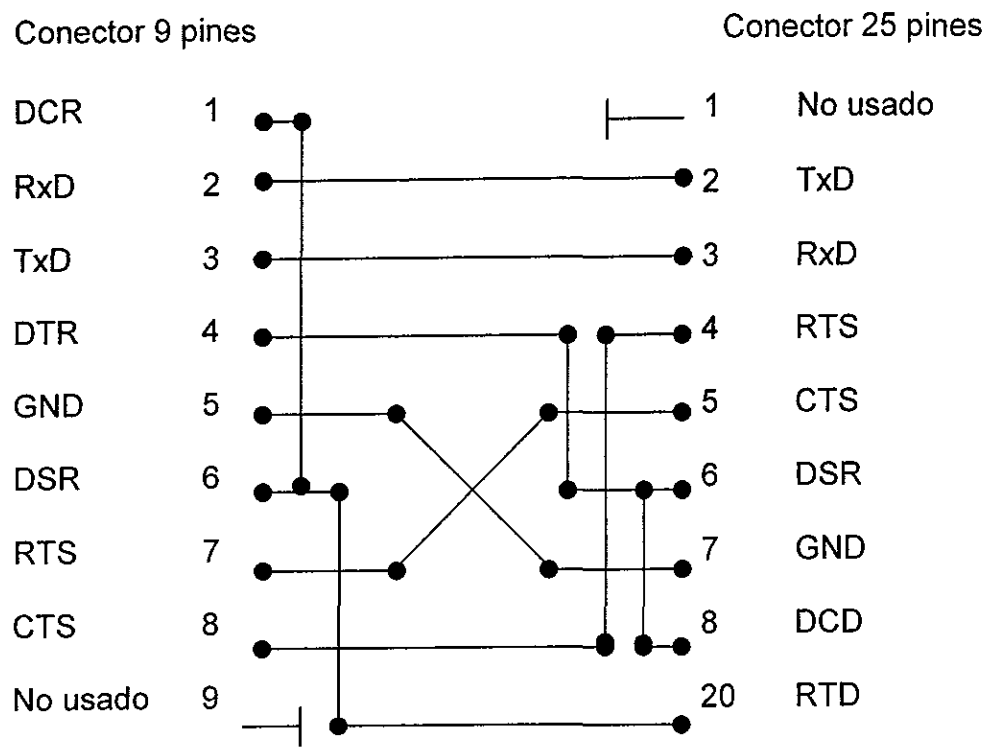


Figura 1.6.7. Cable sin módem. DB-9 patillas a DB-25 patas.

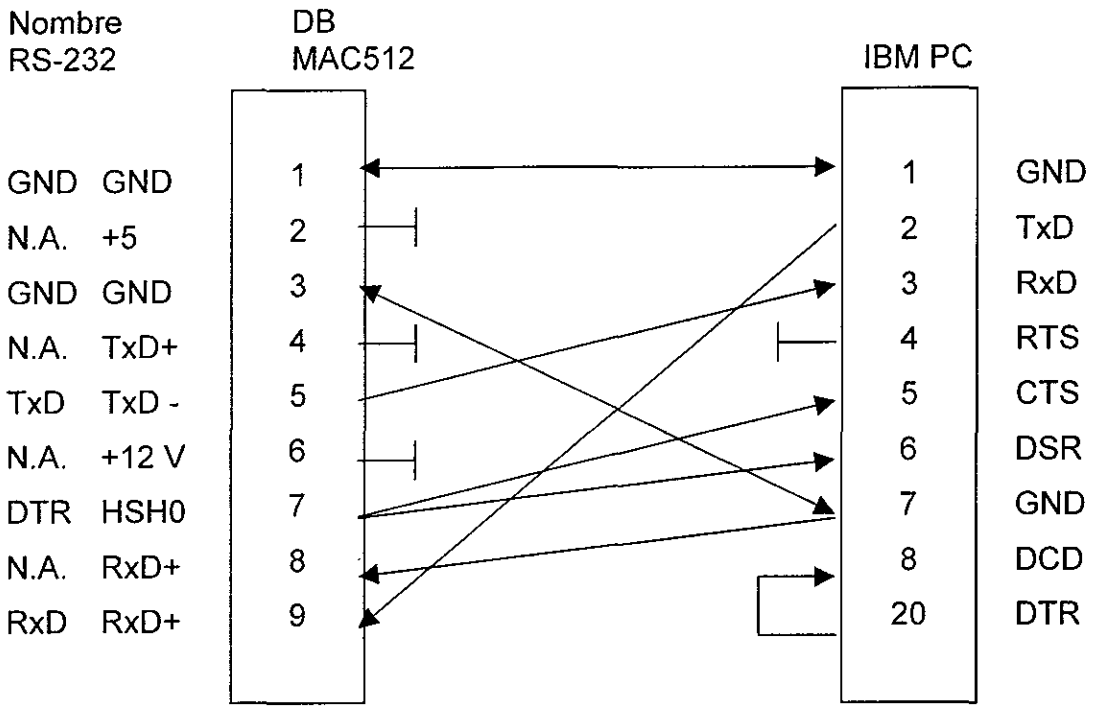


Figura 1.6.8. Cabe sin módem, de Machintosh DB-9 patillas a IBM DB-25 patas.

Interfaz RS-232, RS-422 y AS-485C.

Las interfaces de nivel físico se utilizan para conectar los dispositivos de usuario a los circuitos de comunicaciones. Para realizar esta importante función, la mayoría de los interfaces de nivel físico describen cuatro atributos. Los atributos eléctricos describen los niveles de tensión (o de corriente) y la temporalización de los cambios eléctricos que representan los valores binarios 0 y 1. Los atributos funcionales describen las funciones realizadas por el interfaz físico. Muchos protocolos de nivel físico clasifican esas funciones como control, temporalización, datos y tierra. Los atributos mecánicos describen los conductores y los hilos del interfaz. Todos los hilos de control, señalización y datos se encuentran generalmente en un mismo cable y unidos a conectores en ambos extremos del mismo. Los atributos de procedimiento describen la función de los conectores y la secuencia de eventos que es necesario efectuar para la transmisión real de datos por el interfaz.

Interfaz RS-232.

La transmisión serial de datos ha sido utilizada en gran medida como medio eficiente de transmisión de información digital a larga distancia. El convenio de la interfaz RS-232 se desarrollo para estandarizar la interfaz entre el ETD (Equipo Terminal de Datos) y los ETCD (Equipo Terminal de Comunicación de Datos) utilizando un intercambio de datos a través de datos seriales binarios.

Los ETD y ETCD se conectan a la interfaz estándar RS-232. Un ETD es normalmente un equipo de usuario final, como una terminal o computadora. El ETCD proporciona al ETD la conexión con el circuito de comunicaciones. La RS-232 describe cuatro funciones del interfaz:

- Definición de las señales de control del interfaz
- Movimientos de los datos del usuario a través del interfaz
- Transmisión de señales de reloj para sincronizar el flujo de datos
- Formación de las características reales del interfaz

RS-232 envía los datos por el interfaz mediante cambios de niveles de tensión. Un 0 binario se representa con una tensión en un rango de +3 a +12 voltios. Un 1 binario se representa con un –3 a –12 voltios. La longitud real del cable RS-232 depende de las características eléctricas del cable, aunque los vendedores no permiten una longitud mayor a los 50 pies o su equivalente de 15 metros.

Los circuitos de RS-232 constan de 25 conexiones (canales). Se presenta en la figura 1.6.9. No se utilizan los 25 canales. Un interfaz entre dos ETCD normalmente necesita entre cuatro y ocho canales.

Las funciones de los 25 canales son como sigue:

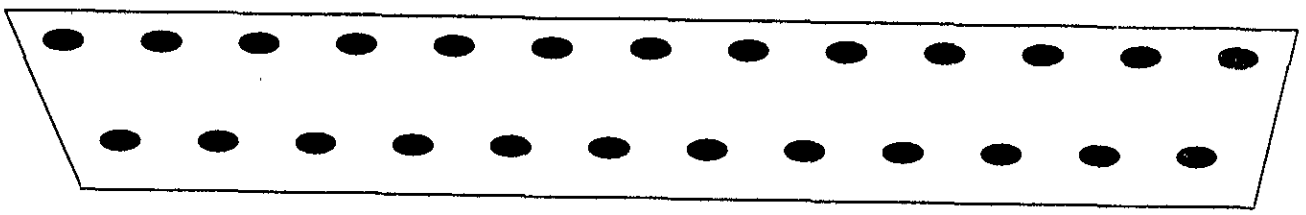
Terminal 1: Circuito AA- Tierra de protección. El conductor se conecta eléctricamente al chasis del equipo.

Terminal 7: Circuito AB- Tierra de señal. Tierra común para todos los circuitos. Así se establece la tensión de referencia para las otras líneas. Es un circuito de referencia común.

Terminal 2: Circuito BA- Datos de transmisión. Señales de datos transmitidos desde el ETD hacia el ETCD. Generalmente representan datos de usuario.

Terminal 3: Circuito BB- Datos de recepción. Señales de datos de usuario transmitidas desde un ETCD hacia un ETD.

Terminal 4: Circuito CA- Solicitud de envío. Señal de ETD a ETCD. Este circuito sirve para notificar al ETCD que la terminal tiene datos para transmitir. El circuito CA se utiliza también en las líneas semiduplex para controlar la dirección de la transmisión de datos. La transición de inactivo a activo notifica al ETCD que debe tomar las medidas necesarias para prepararse para la transmisión.



1.TIERRA DE PROTECCION
2.DATOS TRANSMITIDOS
3.DATOS RECIBIDOS
4.SOLICITUD DE TRANSMISION
5.PEMISO PARA TRANSMITIR
6.EQUIPO DATOS PREPARADO
7.TIERRA DE SEÑAL
8.DETECCION DE PORTADORA
9.RESERVADO. NO ASIGNADO
10.RESERVADO. NO ASIGNADO
11.RESERVADO. NO ASIGNADO
12.DETECCION DE PORTADORA
13.PERMISO PARA TRANSMITIR
14.DATOS SECNDARIOS
15.RELOJ DE TRANSMISION
16.DATOS SECUNDARIOS
17.RELOJ DE RECEPCION
18.NO ASIGNADO
19.SOLICITUD DE TRANSMISION
20.TERMINAL DE DATOS PREPARADA
21.DETECTOR DE CALIDAD DE SEÑAL
22.TIMBRE INDICADOR
23.SELECTOR DE VELOCIDAD DE DATOS
24.RELOJ DE TRANSMISION
25.NO ASIGNADO

Figura 1.6.9. Conexión del circuito RS –232.

Terminal 5: Circuito CB- Permiso para transmitir. Señal de ETCD que indica al ETD que puede transmitir datos. La señal de permiso puede activarse tras recibir una señal de portadora precedente del módem remoto. La temporalización del CB varía de un módem a otro.

Terminal 5: Circuito CC- Equipo de datos preparado. Señal precedente del ETCD, con lo que indica una de las siguientes condiciones:

- La máquina está descolgada, es decir conectada a la línea conmutada
- El ETCD ha completado las funciones de sincronización y responde con tonos

Terminal 20: Circuito CD- Terminal de datos preparados. Señal procedente del ETD que indica que la terminal está encendida, que no se detecta ningún indicio de mal funcionamiento y que no se encuentra en modo de pruebas. Por lo general la línea CD permanecerá activa siempre que el equipo este listo para transmitir o recibir datos.

En una configuración conmutada, una señal de timbre del nodo remoto activará generalmente el CD. CD mantiene el canal en condición de conectado.

Terminal 22: Circuito CE- Indicador de timbre. Señal procedente del ETCD que indica que se está recibiendo una señal de llamada por un canal conmutado.

Terminal 8: Circuito CF- Detección de señal de recepción en línea. Señal procedente del ETCD, con la que se indica que este está detectando la señal portadora generada por el módem remoto. Se denomina también detección de portadora en línea (DCD).

Terminal 21: Circuito CG- Detector de calidad de la señal. Señal procedente del ETCD, con la que se indica que la señal recibida tiene la calidad suficiente para suponer que no ha aparecido ningún error.

Terminal 23: Circuito CH y CI- Selector de velocidad binaria de la señal. Señales procedentes del ETD y ETCD, respectivamente, que indican la velocidad de señalización de los datos en las máquinas dotadas de velocidad dual. Algunos dispositivos son capaces de transmitir a velocidades binarias variables.

Terminal 24: Circuito DA- Temporalización del elemento de señal del transmisor. Señales procedentes del ETD que proporcionan la temporalización a as señales de datos que estén siendo transmitidas por el circuito o BA(Datos transmitidos) hacia el ETDC. El ETD se encarga de generar la señal; si es el ETDC el que genera el sincronismo, el circuito empleado es DB.

Terminal 15: Circuito DB- Temporalización del elemento de señal del transmisor. Señales procedentes del ETDC que proporcionan la temporalización a las señales de datos que estén siendo transmitidas por el circuito o BA(Datos transmitidos) hacia el ETCD. El ETD se encarga de generar la señal; si es el ETD el que genera el sincronismo, el circuito empleado es DA.

Terminal 17: Circuito DD- Temporalización del elemento de señal del receptor. Señales procedentes del ETCD que proporcionan el ETD la temporalización necesaria para las señales de datos que estén siendo recibidas por el circuito BB(Datos recibidos).

En RS-232 los niveles de tensión son detectados en el receptor mediante la diferencia relativa de tensión ente el circuito de señal y el circuito de tierra (Circuito AB), sin embargo, las estaciones transmisora y receptora generalmente tienen diferentes tierras debido a las diferentes características eléctricas de sus componentes. El voltaje aplicado por el transmisor y recibido por el receptor pueden ser diferentes. Si la diferencia de potencial es pequeña no se producirán errores.

Uno de los usos más comunes para RS-232 es la conexión de terminales con computadoras. Esto se realiza a través de módem tanto directa como indirectamente.

Interfaz RS-422.

Este circuito está diseñado de modo equilibrado, donde un circuito toma como referencia otro circuito y no una determinada tierra.

El circuito de interfaz de voltaje balanceado normalmente se utiliza para líneas de datos temporizando o control, donde las velocidades de la señal están entre 100 Kbps a 10 Mbps. Las especificaciones del RS-422 no ponen restricciones en la frecuencia de operación mínima o máxima pero sí en la relación de velocidades de transición de un intervalo unitario.

Aunque los circuitos de transmisión simple son normalmente utilizados a bajas frecuencias, la transmisión diferencial en líneas balanceadas pueden ser preferidas bajo las siguientes condiciones:

- Líneas de interconexión demasiado largas para operación desbalanceada efectiva.
- Líneas de transmisión expuestas a los niveles de ruido electrostático o electromagnético.

- Donde se desee una simple inversión de señal (obtenida a cambio de las líneas balanceadas).

Un circuito de interfaz digital balanceado básico consta de 3 partes y un ejemplo se muestra en la figura 1.6.10.

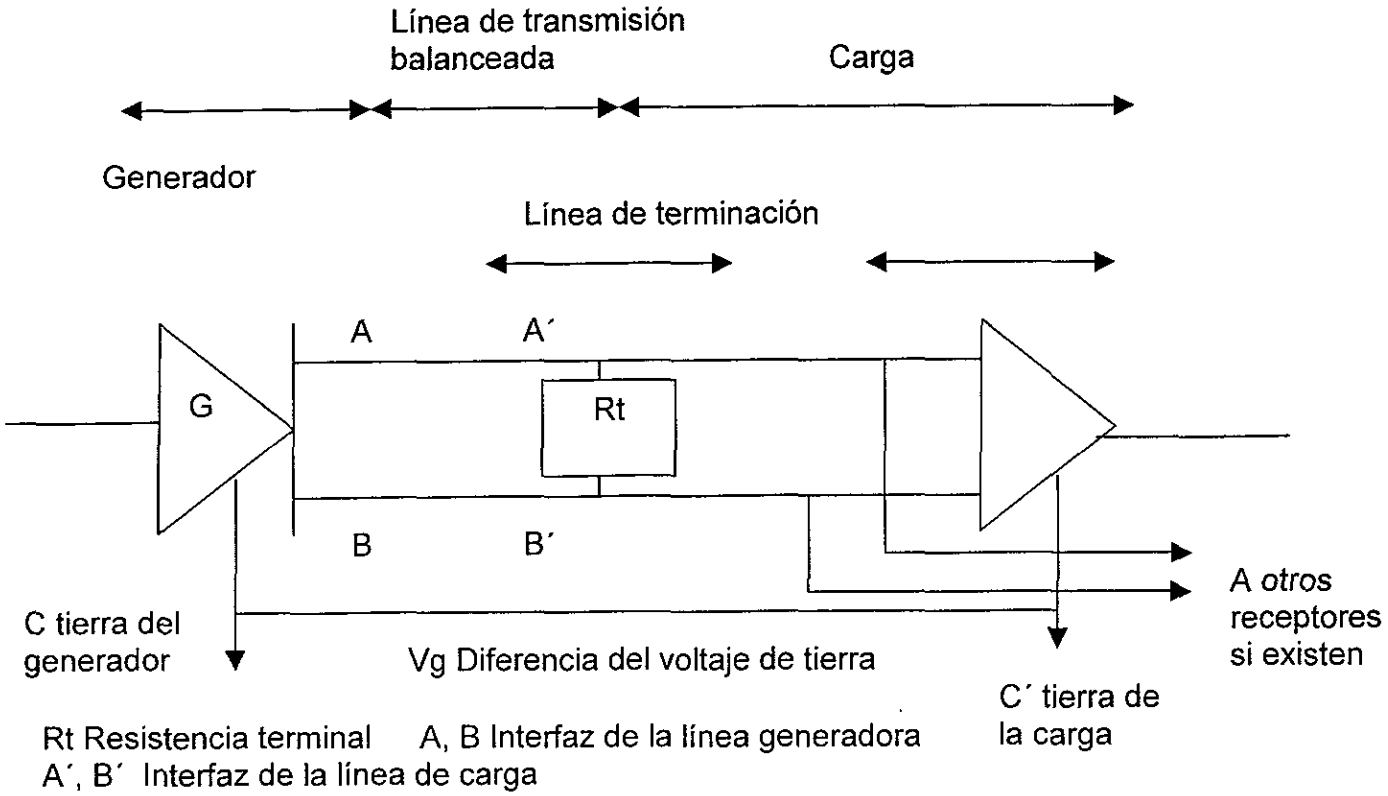


Figura 1.6.10. Interfaz digital balanceada.

- El generador (G) o manejador de línea de datos.
- Una línea de transmisión balanceada.
- Las cargas, donde una carga puede consistir de uno o más receptores (R) y la línea de resistencia terminal de la línea (R_T).

El tipo de manejador RS-422 tiene una fuente de voltaje de salida balanceada (diferencial) con una impedancia de 100Ω menos. Su salida de voltaje diferencial esta en un rango de 2 V mínimo y como máximo 6 V. Adicionalmente, el voltaje de salida con respecto a tierra no debe exceder los 6 V.

El balance de voltaje de salida (V_{OD}) no debe ser menor a 2 V con resistencias terminales de 50Ω (± 1 %) en serie entre salidas. La diferencia entre la polaridad opuesta del voltaje de salida debe ser menor a 0.4 V. El voltaje de offset de salida del manejador (V_{OS}), medido de la unión de dos resistencia terminales de 50Ω y la tierra del manejador, no debe exceder a los 3 V (en cualquier polaridad). La magnitud del cambio en (Vos) debe ser menor a 0.4 V para voltajes diferenciales de salida de la polaridad opuesta.

La corriente de salida del manejador con cualquier salida en corto con tierra, no debe exceder 150 μA. La corriente de fuga en apagado, con cualquier voltaje entre –0.25 y 6 V aplicado a cualquier salida no debe exceder 100 mA.

Los requerimientos de entrada básicos del receptor son como siguen:

- Sensibilidad del umbral de entrada de datos diferencial de ±200 mV, sobre un rango de modo común (V_{CM}) de –7 a 7 V. Impedancia de entrada mayor o igual a 4Ω.
- Las características de voltaje-corriente de entrada del receptor deben ser balanceadas de tal forma que su salida permanezca en el estado binario deseado con una entrada diferencial aplicada de 400mV (a través de 500Ω ±1 % en cada terminal de entrada).

El estándar EIA RS-485, introducido en 1983, es una versión mejorada del RS-422. Incrementa el uso de transmisiones balanceadas en la distribución de datos a varios sistemas, componentes y periféricos sobre las líneas relativamente largas, teniendo la necesidad de múltiples combinaciones manejador/receptor en una línea simple de par trenzado.

El estándar EIA RS-485 toma los requerimientos del RS-422 para la transmisión en líneas balanceadas más las características adicionales para manejadores y receptores múltiples.

El estándar EIA RS-485 difiere del RS-422 principalmente en las características que permiten comunicaciones multipunto confiables. Para los manejadores, estas características son:

- Un manejador puede controlar hasta 32 cargas unitarias y un total de resistencia en la línea terminal de 60Ω o más (una carga unitaria es típicamente un manejador pasivo o un receptor).
- La corriente de fuga de salida del manejador, en "apagado" debe ser de 100μA o menor con cualquier voltaje entre –7 y 7 V.
- El manejador debe de ser capaz de proporcionar un voltaje de salida diferencial ente 1.5 y 5 V con voltajes de línea a modo común de –7 a 12 V.

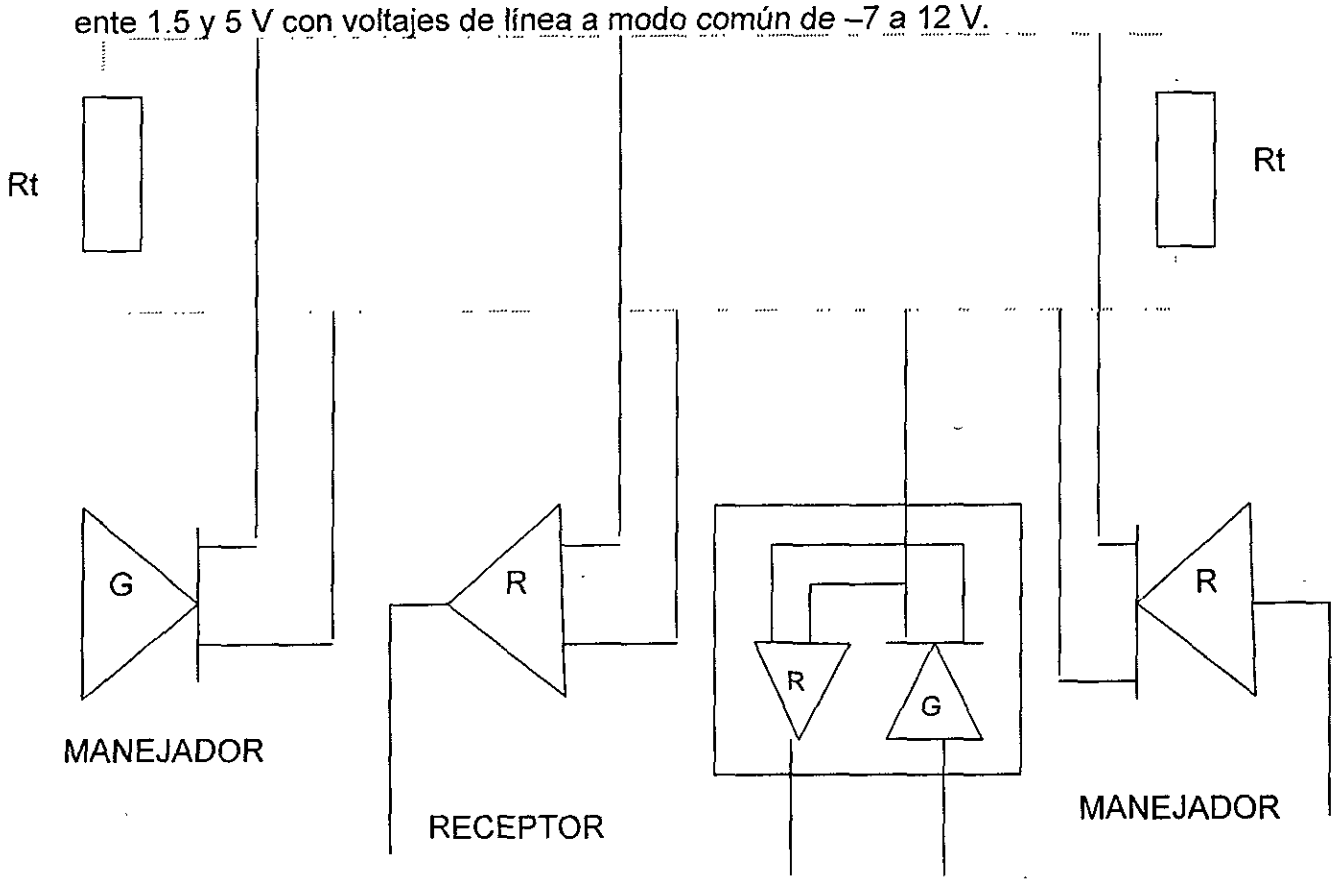


Figura 1.6.11. Interfaz digital balanceada multipunto.

Los manejadores deben tener protección propia contra cualquier contención (los manejadores múltiples pelean por la líneas de transmisión al mismo tiempo), esto es, no debe ocurrir daño en el manejador cuando sus salidas están conectadas a una fuente de voltaje de –7 a 12 V, si su estado de salida es 1 binario o pasivo.

Para los receptores las características son:

- Alta resistencia de entrada del receptor, 12Ω mínimo.
- Un rango de voltaje de entrada a modo común entre –7 y 12 V.
- Sensibilidad de entrada diferencial de ±200 μV sobre un rango de –7 y 12 V.

Funciónamiento de un módem.

Los módems (**modulador/demodulador que permite la comunicación entre dos computadoras**) siguen siendo un misterio para muchos de los millones de individuos que los han adquirido, inclusive muchos piensan que los módems representan la única forma de comunicación variable hoy en día para enviar y recibir información entre computadoras. La tecnología de los módems ha avanzado de forma impresionante en los últimos años. Los nuevos módems son más fáciles de usar, funcionan mayor velocidad, son más baratos, más fiables y consumen menos electricidad ocupando menos espacio. Los módems ofrecen prestaciones estándar que anteriormente eran consideradas "un lujo", como el marcado automático, respuesta automática, capacidades de diagnóstico, ROM programable y conjuntos de órdenes avanzadas.

El módem es un dispositivo para convierte los pulsos eléctricos ON/OFF generados por una computadora en sonidos audibles, y luego los reconvierte en impulsos eléctricos ON/OFF (**activo/inactivo**) al otro extremo de la línea. Todos los módems incluyen componentes comunes, como un transmisor y un receptor. El transmisor modula la señal digital a analógica (tonos y sonido), y el receptor demodula la señal analógica recibida y la convierte de nuevo en digital. La figura 1.6.12 muestra ambos tipos de señales, analógica y digital.

Señal portadora y onda senoidal.

Cuando dos módems se comunican, intercambian tonos audibles continuos denominados señales portadoras. Cada señal portadora tiene una frecuencia establecida por los fabricantes de módems o un estándar publicado. Si un módem detecta la ausencia de portadora durante un intervalo superior a pocos milisegundos, interrumpe la conexión (el módem cuelga). Es como una conversación entre dos personas; si la persona 1 no escucha nada procedente del otro extremo de la línea (ruido de fondo, música ni respiración procedente de la persona 2), cuelga. El módem de la persona 1 envía un mensaje de "portadora perdida" al usuario.

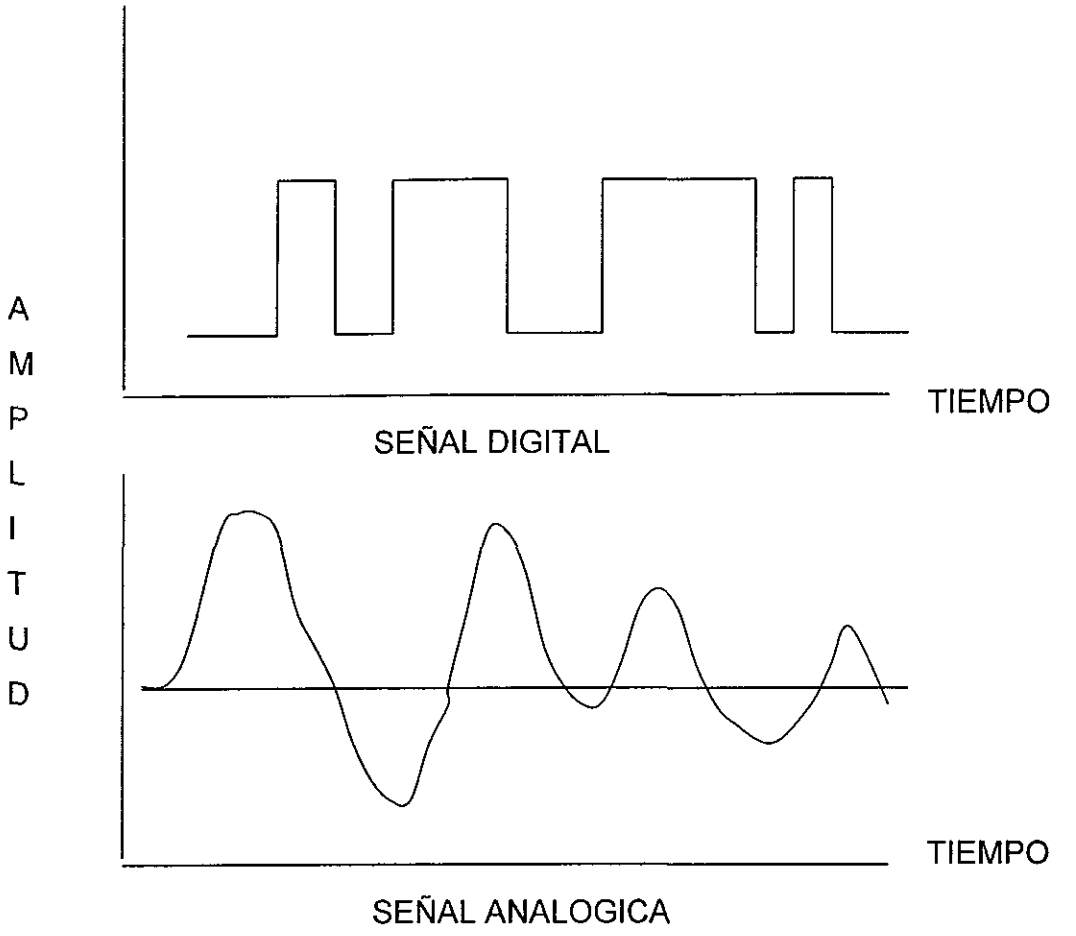


Figura 1.6.12. Señales analógicas y digitales.

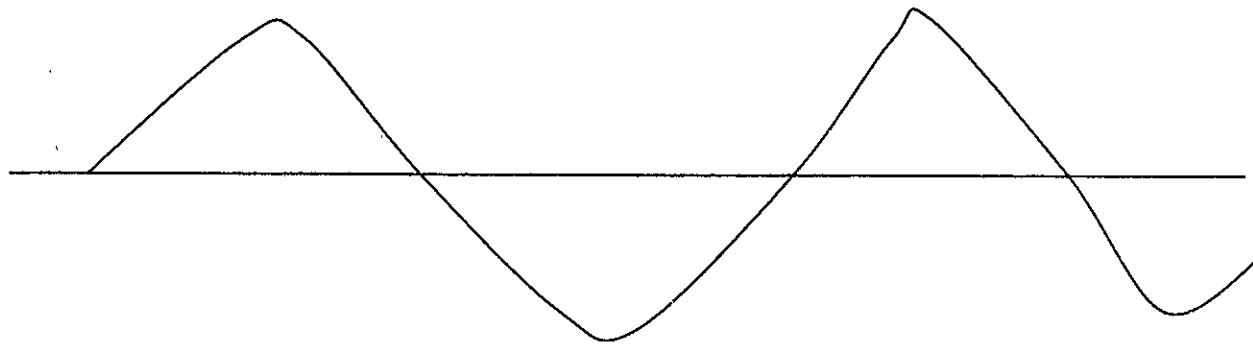


Figura 1.6.13. Onda Senoidal.

Las señales portadoras son generadas como ondas senoidales como se aprecia en la figura 1.6.13. Las ondas senoidales comienzan con un voltaje cero y suben hasta llegar a un cierto valor positivo, luego vuelvan a cero, y luego al mismo valor pero negativo, y luego a cero. Cuantos más ciclos produzcan en una unidad de tiempo, mayor será la frecuencia de la señal.

Transmisión mediante un módem.

-Comprensión de las señales analógicas digitales.

El sistema de numeración binario utiliza 1s y 0s, y permite expresar cada número, letra y símbolo como una secuencia especial de estos dos dígitos. Ejemplo:

A = 01000001
B = 01000010
C = 01000011

Cada grupo de ocho bits (1s y 0s) representa y byte (**un carácter**).

Generalmente, dentro de una computadora la transferencia de información se realiza mediante un bus paralelo (**conjunto de cables**). Si un sistema ON/OFF puede ofrecer dos estados de información, podemos ver un bus paralelo como ocho sistemas ON/OFF, que ofrecen ocho canales simultáneos de datos. Los 1s y 0s de los ocho sistemas pueden ofrecer esta información a cualquier observador. Un canal, o circuito ON/OFF, nos da un bit de datos en cada instante: un bit 0 o un bit 1. Por lo tanto agrupando los ocho bits, se tendrá un byte de información.

Como los ocho circuitos ofrecen datos, podemos denominarlos correctamente canales de datos. La información de estos canales de datos (numerados del 1 al 8) se mantiene sincronizada por un reloj común que permite entrar y salir simultáneamente del canal a los ocho bits.

Combinando ocho canales de dos estados, se obtiene un total de 256 posibles combinaciones.

00000000 00000100
00000001 00000101
00000010 00000110
00000011 00000111... hasta 11111111

Se ha creado un método estandarizado para utilizar estos 1s y 0s como información (dos bytes de cuatro bits por byte). Este estándar se denomina el American Standard Code for Information Interchage (Código estándar americano para intercambio de información), conocido también como ASCII.

Transmisión de información digital usando modulación.

Para representar información digital se necesitan como mínimo dos estados. Estos estados se representan por la alteración de la señal portadora para representar el dígito binario 1. La modificación de la señal portadora se denomina modulación.

La modulación puede emplear la variación de un grupo cualquiera de estos atributos de la portadora:

- Amplitud : Magnitud de voltaje de pico.
- Frecuencia : Número de oscilaciones completas de las señal por la unidad de tiempo
- Fase : Posición en que la señal pasa por cero, relativa a la señal anterior.

Distintos módems, velocidades y frecuencias.

Los módems usan distintas formas de modulación dependiendo de la velocidad correspondiente. Por ejemplo:

- **Modulación por desplazamiento de frecuencia (FM o FSK).** Se utiliza para velocidades inferiores a 1200 bps (**bits por segundo**). La modulación FM es una técnica en dos niveles que representa los cambios en el patrón binario de bits mediante cambios en la frecuencia de un tono de audio. Se supone que la línea está en reposo con un valor 1, representado por un tono de frecuencia determinada. El módem cambia a un tono de otra frecuencia cuando se envía un bit de datos 0 (véase figura 1.6.14). Aunque los módems de mayor velocidad (véase la modulación PSK descrita posteriormente) incluyen soporte para transmisión FSK, la modulación FSK a baja frecuencia se utiliza en contadas ocasiones. Aunque extremadamente fiable, la transmisión a 300 bps limita la transmisión a unos 30 caracteres por segundo. La mayoría de usuarios pueden leer texto visualizado en la pantalla a velocidades dos o tres veces superiores, y se impacientan ante una trasferencia a 300 bps.

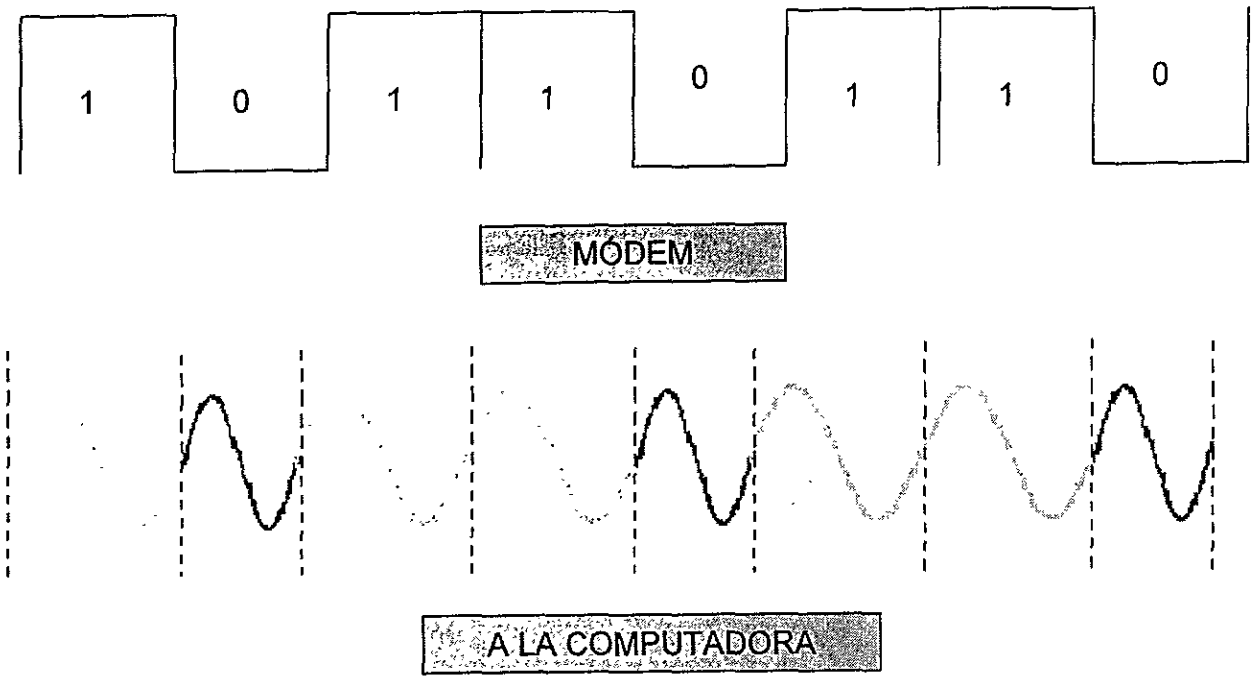


Figura 1.6.14. Modulación por desplazamiento de frecuencia (FSK o FM).

- **Modulación por desplazamiento de fase (PSK).** Esta modifica la fase de una señal, es decir, su sincronización respecto a una referencia fija, para representar cambios en el patrón de bits. Para medir el desplazamiento de fase de la señal recibida y determinar si es 0 ó 1, se utiliza un oscilador de referencia (figura 1.6.15).

- **Modulación por desplazamiento de fase diferencial (DPSK).** Se utiliza un módems de 1200 y 2400 bps para PC, y compara el ángulo de fase de la señal recibida con la señal recibida anteriormente. Un cambio de fase se interpreta como un 0 binario si la fase anterior se interpretaba como 1, y así sucesivamente. Este método no requiere una señal de referencia, por lo tanto necesita menos circuitería (figura 1.6.16).
- **Modulación de amplitud (AM).** Es la técnica de modulación más sencilla. Las ondas de amplitud grande se asignan al 1 binario, y las de amplitud pequeña al 0 binario. La AM es muy susceptible a las interferencias de las líneas, y en la practica no se utiliza de forma aislada.

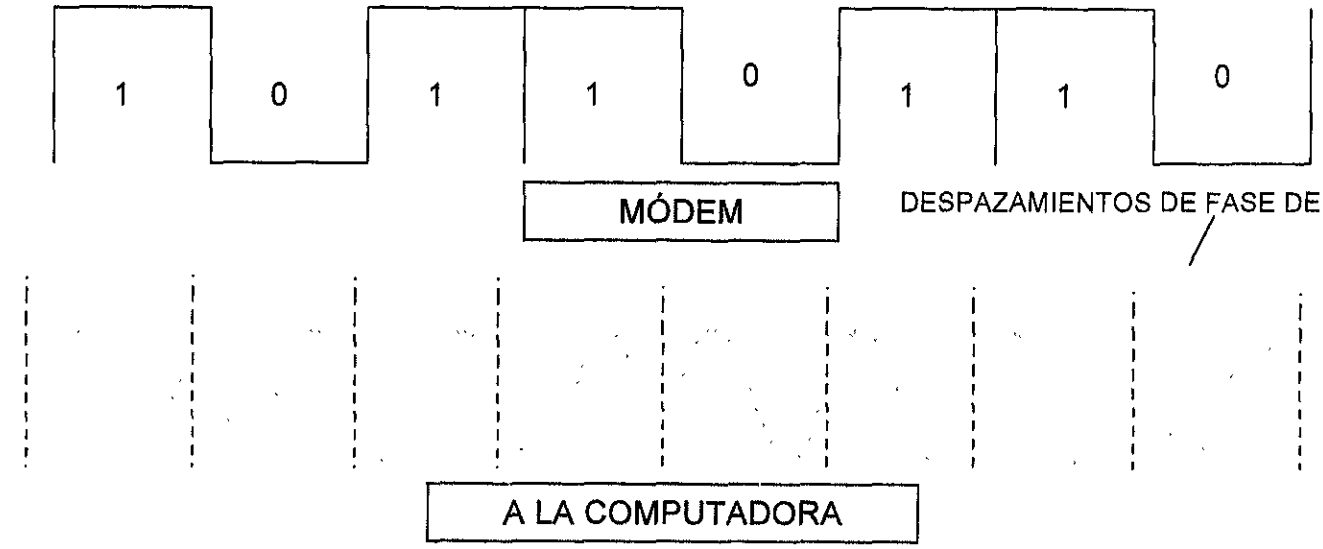


Figura 1.6.15. Modulación por desplazamiento de fase (PSK).

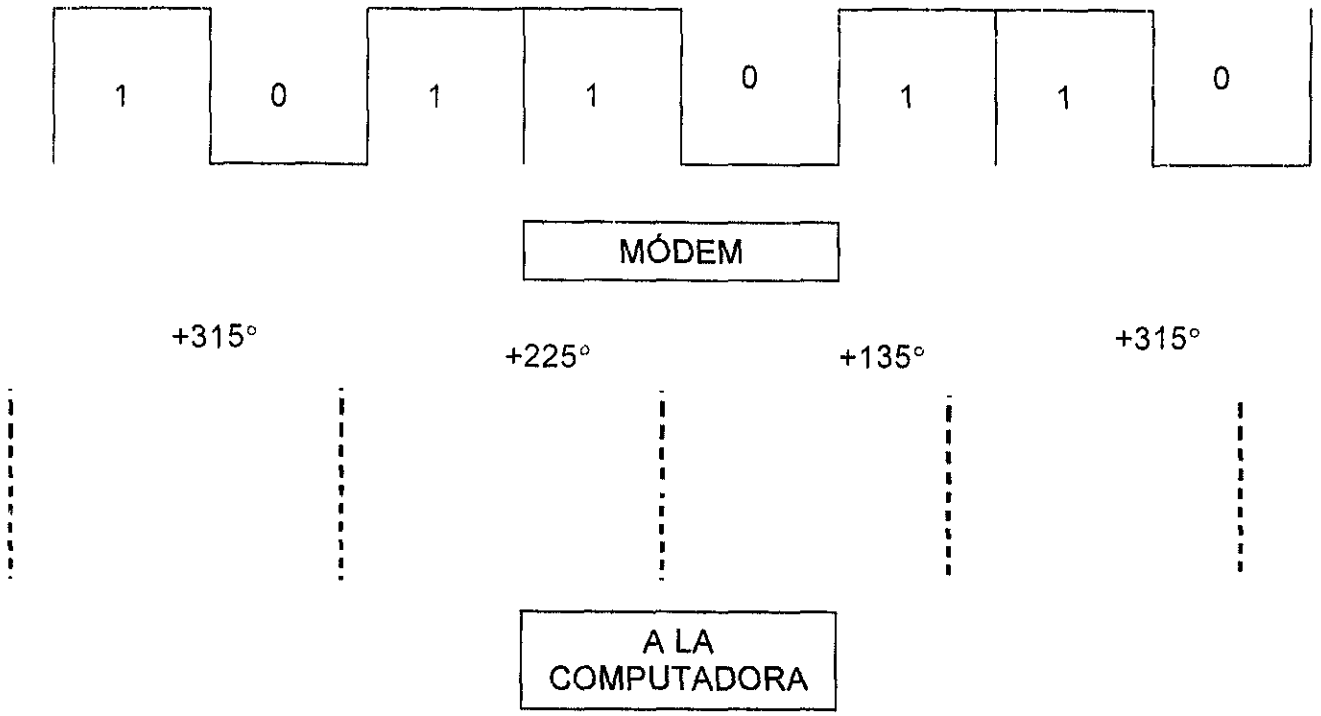


Figura 1.6.16. Modulación por desplazamiento de fase diferencial (DPSK).

La velocidad de transmisión se controla con la configuración de los interruptores del módem, o por órdenes de software. La velocidad de transmisión aparece a menudo, aunque erróneamente, denominada como la velocidad en baudios.

Velocidad en baudios y velocidad de transmisión no significan lo mismo, aunque a menudo se utilizan como sinónimos. La diferencia es la siguiente: El término <<bps>> expresa la velocidad de cambios en la señal; es una medida de señal por segundo (baudios). Los módems de alta velocidad codifican dos o más bits en cada cambio de señal. La velocidad en bps corresponde al número de bits de datos por señal, multiplicado por los baudios.

Baudio = Cambios de la señal por segundo

Bps = (Grupo de bits de datos) x Velocidad en baudios

Grupos de un bit	Grupos de dos bits
00	10
01	11

Con grupos de dos bits, podemos representar cuatro patrones de bits (00,01,10,11). Por ejemplo, en la modulación por desplazamiento de fase, un desplazamiento de 90 grados podría representar el grupo 01, 180 grados el 10, 270 el 11, y un desplazamiento nulo 00. Transmitiendo cuatro bits de datos en cada señal, un módem puede soportar una velocidad de 9,600 bps. Como todos los módems de media y alta velocidad agrupan los bits de datos para transmitir, actualmente es raro que la velocidad en bits de un módem coincida con su equivalente en baudios.

Condiciones de las líneas.

La red telefónica no es un medio de transmisión perfecto, incluso para la comunicación oral. El incremento en la velocidad de datos exige métodos de transmisión cada vez más precisos. Problemas tales como fallos en la línea, interferencias eléctricas y ruidos aleatorios pueden interrumpir la transmisión.

El ruido aleatorio es una disminución de la relación señal/ruido; es decir, la proporción entre la potencia de la señal y la amplitud de los pulsos de ruido. La distorsión de amplitud es otro problema enorme aún para los expertos en telecomunicaciones. Los módems deben poder adaptarse a las propiedades de la línea de comunicaciones, para evitar deformaciones excesivas de la señal. Otra causa de problemas en los circuitos es la propagación desigual de las frecuencias altas y bajas. Esta condición, denominada vibración de fase, afecta severamente a los módems de alta velocidad, cuyas técnicas de modulación utilizan desplazamientos de fase para representar patrones de bits.

Algunos expertos en telecomunicaciones aconsejan a los de computadoras que se pongan en contacto con su compañía telefónica local y obtengan líneas especialmente acondicionadas. Las líneas acondicionadas requieren la conexión de amplificadores de señal y ecualizadores de atenuación o retraso. Estas líneas suponen una fuerte mejora, pero siguen sin garantizar la obtención de unas tasas óptimas de eficiencia.

Los últimos modelos de módems ofrecen una prestación útil denominada <<fallback>>. Esta consiste en la capacidad de detectar condiciones desfavorables en la línea y utilizar velocidades de transmisión más bajas para evitar errores en los datos.

Conexión de los módems a líneas telefónicas.

Existen dos formas de conectar un módem a una línea telefónica: de forma acústica o directa. El acoplamiento acústico requiere colocar un microteléfono (el auricular/micrófono) en un par de receptáculos que albergan el altavoz y micrófono del módem.

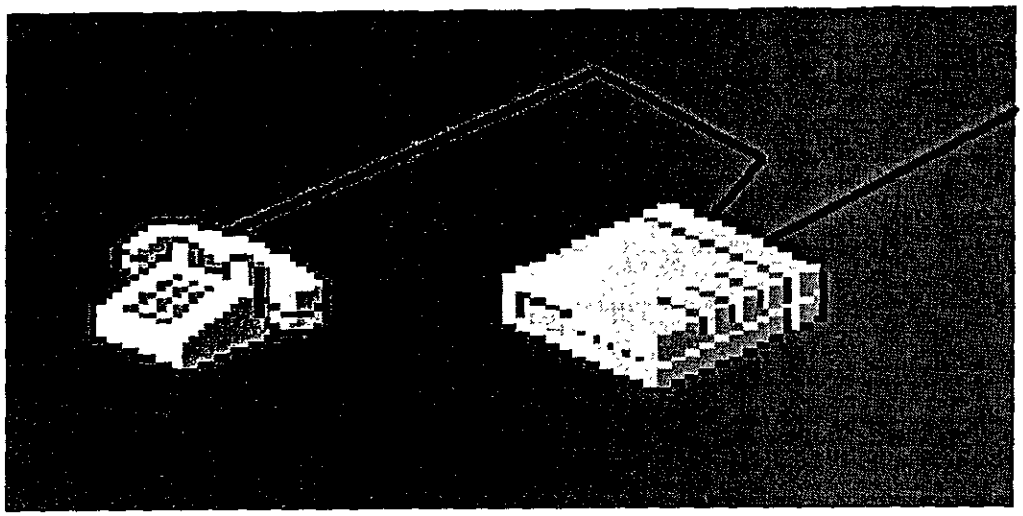


Figura 1.6.18. Módem con acoplamiento acústico.

Los antiguos dispositivos con acoplador acústico continúa siendo útiles si queremos conectar rápidamente un módem y no disponemos de conectores modulares, como en los teléfonos públicos, habitaciones de hotel, edificios de oficinas o teléfonos con varias líneas. Resulta simple poner el microteléfono sobre el módem acústico Se debe evitar mover o golpear el módem durante la transmisión, ya que se podría interrumpir la comunicación. Muchas de las antiguas computadoras portables incluían un módem acústico.

Los módems de conexión directa se conectan a las líneas telefónicas utilizando el habitual conector modular telefónico RJ11 (figuras1.6.21). Son menos sensibles al ruido, y resultan fáciles de conectar.

La mayoría de los módems utilizan conexión directa. Se han vuelto tan populares que ya en muchas oficinas e incluso hoteles se instalan conectores RJ11 para que las personas puedan conectar sus computadoras personales fácilmente.

Leyenda de los diagramas de módem

CCITT	EIA	MNEMONICO	DESCRIPCION
101	AA	GND	Tierra de protección
103	BA	TxD	Datos transmitidos
104	BB	RxD	Datos recibidos
105	CA	RTS	Petición de envío
106	CB	CTS	Listos para enviar
107	CC	DSR	Datos preparados
102	AB	GND	Tierra de referencia de la señal
109	CF	DCD	Detección de portadora
108	CD	DTR	Terminal de datos preparado

Figura 1.6.19. Leyenda de los diagramas de módem.

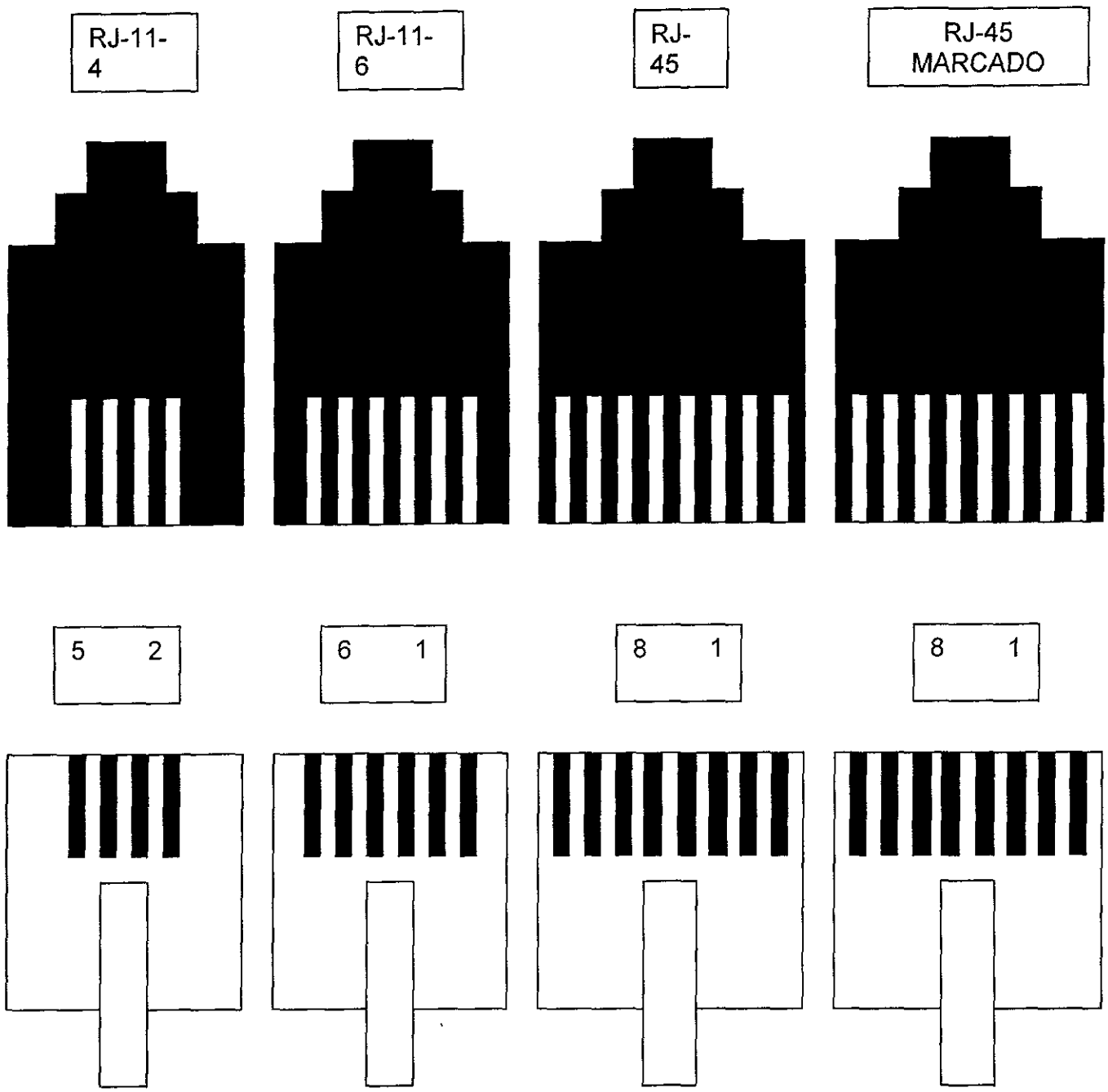


Figura 1.6.20. Cajetín telefónico RJ11 asignación de patillas en los conectores.

Modems de llamada y respuesta automática.

Prácticamente todos los módems de conexión directa pueden contestar automáticamente a una llamada telefónica. También pueden realizar una llamada abriendo electrónicamente la línea y marcando un número. El uso ideal de un módem de respuesta automática es dejarlo conectado a la línea telefónica, para que responda cuando reciba la llamada de otro módem. El problema consiste en que el módem no puede detectar cuándo recibe una llamada de otro módem u otra persona, y puede que no deseemos que todas las llamadas que recibamos sean respondidas por un tono agudo. Los diseñadores de varios productos han prometido que distinguirán entre voz, módem y fax en una línea telefónica y luego dirigirán la llamada al dispositivo adecuado.

También hay programas que convierten el módem de llamada y respuesta automática en un marcador de teléfonos. Estos programas de marcado recuerdan los números, buscan uno y lo marcan usando el módem. Todo lo que se tiene que hacer es seleccionar la persona a la que se desea llamar y esperar. El modulador/demodulador de módem se desconecta para que no exista ninguna señal portadora molesta. Si un número está ocupado, el programa indica al módem que siga marcando hasta establecer la conexión.

Conexión de un módem a una PC.

Después de conectar satisfactoriamente el módem a la línea telefónica, el paso siguiente es conectar el módem a la computadora persona (figura 1.6.21).

Los módems externos se conectan con un cable DB-25 o DB-9 al puerto de comunicaciones RS-232 de la computadora. Los módems de tarjeta o internos se conectan instalando una placa en uno de los conectores de expansión de la computadora (figura 1.6.22).

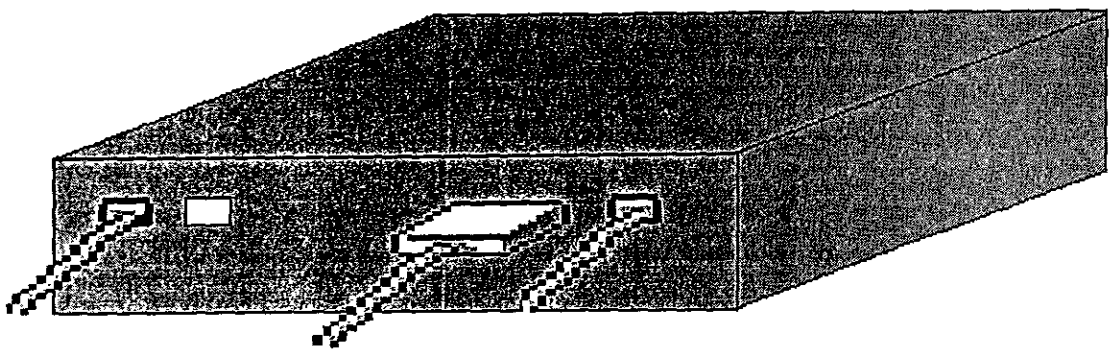


Figura 1.6.21. Conexiones de la parte trasera de *un módem externo.*

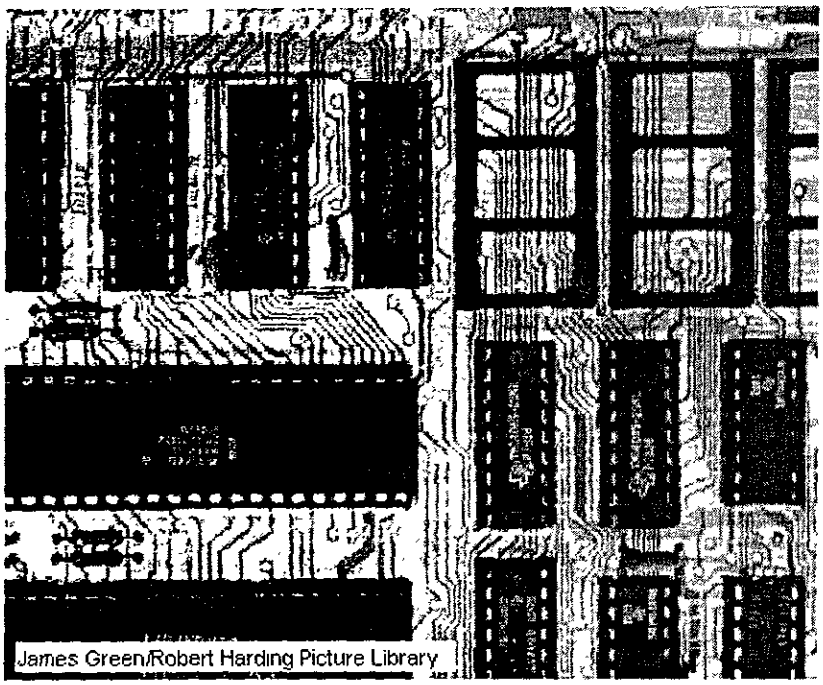


Figura 1.6.22. Un módem interno se conecta en un conector de *expansión.*

CAPÍTULO II

PLANTEAMIENTO DEL PROBLEMA Y PROPUESTAS DE SOLUCIÓN

2.1 Recopilación y Clasificación de la Información.

Introducción.

La Fuente de Información Financiera recibe su alimentación desde varias fuentes generadoras, ubicadas en diversos puntos geográficos. Fuentes como bolsas de valores, salas de redacción, bancos, entre otras, son recolectadas y transmitidas a los ticker plants (**centros de procesamiento**) estratégicamente ubicados para que la información sea procesada y distribuida en tiempo real.

La información aquí descrita es una guía para entender el protocolo del Feed (**alimentación**) de la Fuente de Información Finaciera, así como para obtener información del catálogo de instrumentos y mercados que provee.

Objetivo del Sistema.

El objetivo del sistema es sustraer la información que recibe la Fuente de Información Financiera, procesarla para obtener un filtro ASCII y enviarla a través de un puerto serie de máquina.

Información que es extraida.

La información que se puede extraer por medio del Sistema de Alimentación es la siguiente:

- Páginas
- Notas
- Hechos de acciones de Mercado de Capitales
- Posturas de acciones de Mercado de Capitales
- Hechos de Warrants
- Posturas de Warrants
- Indices

- Indice IPC
- Hechos de acciones de USA
- Posturas de acciones de USA
- Hechos de Opciones
- Posturas de Opciones

2.1.1 Formato de la información.

Se cuenta con un formato general y a partir de este se derivan el resto de los formatos. A continuación se explican tanto el formato general como los cambios en cada uno de los restantes, según el tipo de información a extraer.

Formato General

[SOH][Tipo][Datos][EXT][CRC]

en donde:

SOH	0X01 (inicio del paquete)
Tipo	Tipo de mensaje (ver APÉNDICE C)
Datos	Contenido de la información, este dato varía dependiendo del Tipo
EXT	0x03 (fin de paquete)
CRC	Detección de errores (ver APÉNDICE B)

Nota:

De momento, la implementación cumple con los requerimientos que exige un Feed Serial independiente del medio de transmisión (RS232, TCP/IP, etc...). En el futuro se podría alterar los headers (cabeceras) y footers (pies de página) para no incluir ciertos campos que pudieran ser intrascendentes en un medio de transmisión que provea una capa de transporte con facilidades para establecer sesiones 100% confiables (streams (flujos) de TCP/IP por ejemplo). Lo que si es igual en todos los casos es la descripción de los campos [Tipo][Datos].

2.1.1.1 Formato de Páginas (TIPO ´4´).

Este tipo de mensajes contiene la información que maneja las páginas. Para este tipo hay 2 subtipos de mensajes diferentes: el del encabezado y el del contenido o cuerpo de la página.

Existen 2 clases de envío:

Envío de la página completa: Cuando se realiza este envío aparecen los 2 subtipos de páginas, es decir, el encabezado y el contenido completo; siempre apareciendo primero el encabezado.

Envío de la información cambiante: Sólo se envía la información que haya cambiado en ese momento puede ser desde un renglón en adelante.

Subtipos de páginas:

A	Encabezado de la Página
B	Contenido o cuerpo de la Página

Nota:
Cuando el texto contenga doble comilla, la información deberá contener pares de éstas para diferenciarlas; las que se usan como delimitadores.

Subtipo ´A´.- Encabezado de la Página.

Contiene la información del encabezado de la página. Para que una página pueda comenzar a recibirse es indispensable haber recibido previamente este registro.

Formato del paquete
[Tipo][Subtipo][Fuente][,][Clasificación][,][Número de Página][,]["Nombre de

página"][,][Total de Renglones][Total Columnas]

Tipo	Tipo de mensaje (ver APÉNDICE C); en este caso es ´4 ´.
Subtipo	Clase de información que contiene el paquete; en este caso es ´A´.
Fuente	Fuente que envía la información (ver APÉNDICE E).
Clasificación	Valor reservado. Este campo siempre toma valor de cero "0".
Núm. Página	Número entero único que junto con la fuente identifica de manera única la página.
Nom. Página	Alfanumérica que describe una página. Este puede estar compuesto de cualquier secuencia de caracteres ASCII enmarcados dentro de doble comillas como delimitador.
Total Renglón	Número de renglones de la página.
Total Columna	Número de columnas de la página.

Subtipo ´B´.- Contenido o Cuerpo de la Página.

Este paquete contiene el texto, cuerpo o información de la página, especificando la posición que ocupa dentro de la misma.

Formato del paquete

[Tipo][Subtipo][,][Número de Página][,][Renglón][,][Columna[,]["Texto ASCII"]

Tipo	Tipo de mensaje (ver APÉNDICE C); en este caso es ´4 ´.
Subtipo	Clase de información que contiene el paquete; en este caso es ´B´.
Núm. Página	Número entero único que junto con la fuente identifica de manera única la página (Debe coincidir con el número enviado en el paquete de encabezado de la página).
Renglón	Posición inicial del texto en lo que se refiere a renglones.

	Número de renglón en que iniciará el texto, comienza en 1.
Columna	Posición inicial del texto en lo que se refiere a columnas. Número de columnas en que iniciará el texto, comienza en 0.
Texto ASCII	Es el texto, cuerpo o información de la página. Este puede estar compuesto de cualquier secuencia de caracteres ASCII enmarcados dentro de doble comillas como delimitador.

Ejemplos:

Se envía la página completa del Dólar interbancario, la cual consta de 18 renglones y 157 columnas, su origen es la Fuente de Información Financiera. Como el primer paquete que se envía es un encabezado entonces el primer paquete queda como sigue:

[SOH]4AINFOFIN,0,1004,"Dólar intervancario",18,157[EXT][CRC]

El siguiente paquete es:

[SOH]4B,1004,0,0," BMV-MD ----- ------ ------- -------"[EXT][CRC]

Se envía 3 renglones de la página del dólar en 3 paquetes diferentes:

[SOH]4B,1004,1,0," BMV-MD ----- ------ ------- -------12:52 22-Nov"[EXT][CRC]

[SOH]4B,1004,2,0," BMV-24 ------ ------ ------- ------ 12:52 22-Nov"[EXT][CRC]

[SOH]4B,1004,3,0," BMV-48 ------ ------ ------- ------ 12:52 22-Nov"[EXT][CRC]

2.1.1.2 Formato para Noticias (TIPO´5´).

En este tipo de mensaje envía las noticias bajo un formato estándar de la Fuente de Información Financiera. Como se detalla más adelante, las noticias (al igual que las páginas) pueden ser enviadas en bloques de textos con el fin de permitir el envío de la información más relevante en lo que se transmite el resto de la noticia, es decir, una vez comenzado el envío de bloques de una noticia es posible enviar índices o información del mercado (o incluso otras noticias), sin tener que esperar

la transmisión de todo el texto de la noticia. Para el caso de las noticias hay 2 subtipos de paquetes, el del encabezado y el del contenido o cuerpo de la nota.

Subtipos de noticias:

A Encabezado de la Nota

B Contenido o cuerpo de la Nota

NOTA:

Cuando el texto contenga doble comillas, la información deberá contener pares de éstas para diferenciarlas de las que se usan como delimitadores.

Subtipo ´A´.- Encabezado de la nota.

Este mensaje sirve para enviar la descripción y encabezado de una noticia. Es indispensable que éste se reciba previamente para que se pueda formar la noticia completa.

Formato del paquete

[Tipo][Subtipo][Fuente][,][Clasificación][,][Número de Nota] [,][Prioridad][,][hora][,][Num Bloques][,]["Encabezados"]

Tipo	Tipo de mensaje (VER apendice C); en este caso es ´5´.
Subtipo	Clase de información que contiene el paquete; en este caso es ´B´.
Num. Nota	Número que junto con la fuente identifican de manera única una noticia.
Prioridad	Este número indica la importancia para que se le pueda dar un trato especial; 0 (cero) para noticias de alta prioridad y 1 (uno) para noticias normales.
Hora	Es la hora en que se generó originalmente la noticia.

Núm. Bloques Este número indica el total de bloques de textos en los que se divide la noticia, sin incluir el encabezado.

Encabezado Es el encabezado de texto noticioso delimitado por doble comillas.

Subtipo 'B'.- Contenido o cuerpo de la Nota

Este mensaje sirve para enviar bloques de texto de la noticia indicada. Es necesario haber recibido previamente el mensaje de encabezado de la nota respectiva para que se pueda procesar.

Formato del paquete

[Tipo][Subtipo][,][Número de Nota][,][Número de Bloque][,]["Texto de la nota"]

Tipo Tipo de mensaje (VER apendice C); en este caso es '5'.

Subtipo Clase de información que contiene el paquete; en este caso es 'B'.

Num. Nota Número que junto con la fuente identifican de manera única una noticia, (debe coincidir con el número enviado en el paquete de encabezado de la noticia).

Núm. Bloque Este es un número consecutivo que indica el número de bloque de texto de la noticia referenciada a través del número de noticia. La concatenación de todos los bloques según el orden consecutivo, da como resultado el texto completo de la noticia.

Texto de Nota Este es el texto integro de un bloque de noticia. El texto puede contener cualquier carácter ASCII delimitado por doble comillas.

Ejemplos:

Se envía una nota cuya clasificación es Noticias Generales, América, la cual está

dividida en 2 bloques, su prioridad es normal y el encabezado es "(EU) APRUEBAN A WESTINGHOUSE COMPRA CBS". Los mensajes quedan como sigue:

Paquete 1:

[SOH]5AINFOFIN,151,197986,1,Mu,2,"(EU) APRUEBAN A WESTINGHOUSE COMPRA CBS"[EXT][CRC]

Paquete 2:

[SOH]5B,197986,1"(EU) APRUEBAN A WESTINGHOUSE COMPRA CBS WASHINGTON, Nov. 22.- La Comisión Federal de Comunicaciones de EU aprobó la compra de la empresa CBS Inc., la tercera mayor compañía de televisión estadounidense, por parte del consorcio Westinghouse Electric Corp., quien pagó 5,400 millones "[EXT][CRC]

Paquete 3:

[SOH]5B,197986,2,"de dólares. La dependencia federal señaló que Westinghouse necesita además la autorización especial de la comisión para poner en funcionamiento su red ampliada de televisión y radio.

"[EXT][CRC]

2.1.1.3 Formato de Información Financiera.

Formato para información de la BMV (TIPO ´6´).

Este tipo de mensajes corresponde a aquellos que provengan de información de la bolsa Mexicana de Valores; pueden contener varios de los siguientes campos a la vez. El listado muestra todos los campos que pueden existir dentro de los paquetes para transacciones de acciones de Renta Variable, posturas de acciones de Renta Variable, evolución de índices, transacciones de acciones de Warrants, posturas de acciones de Warrants.

Campo	Contenido	Descripción
A	Tipo Valor [,]	Clasificación que le es otorgada a la acción por la BMV. Este es un campo tipo alfanumérico y termina con una coma como delimitador.
B	Hora (*)	Hora en que se generó la última operación de la acción, ya sea postura, transacción o status.
C	Status [,] Códigos de Status: A C O R P	Este campo contiene una secuencia de caracteres en donde cada carácter indica un status de la acción. Es de tipo alfanumérico y termina con una coma como delimitador. Alta: Con este código se envía una transacción o hecho de la acción. Cierre: Con este código se envían el último valor de la acción cuando cierra el mercado. Operatividad: Con este código se envían los datos del último estado de la acción. Reestructuración: Con este código se envían los datos de una transacción o hecho enviado anteriormente con el "folio de envío" y actualización del último estado de la acción. Prueba: Indica que el mensaje es de prueba y no debe ser procesado para usuarios finales.
D	Valor (*)	Precio o valor actual de la acción o del índice. Este es un campo de tipo flotante.
E	Valor de Referencia (*)	Precio o valor de referencia para vacaciones. Este es un campo de tipo flotante.
F	Fecha del Valor de Ref (*)	Fecha en que sucedió el precio de referencia.
G	Valor de Apertura (*)	Precio o valor con que dio inicio la cotización de la acción o del índice. Este es un campo de tipo flotante.
H	Valor Máximo del día (*)	Precio o valor máximo de la acción o del índice durante el día. Este es un campo de tipo flotante.
I	Valor Mínimo del día (*)	Precio o valor mínimo de la acción o del índice durante el día. Este es un campo flotante.
J	Importe Acumulado (*)	Importe acumulado durante el día de la acción o índice. Este es un campo de tipo entero.
K	Número de Operaciones (*)	Total de operaciones efectuadas de la acción en el día. Este es un campo de tipo entero.
L	Volumen (*)	Volumen de acciones de la última operación. Este es un campo de tipo entero.
M	Volumen Acumulado (*)	Volumen acumulado en el día. Este es un

		campo de tipo entero.
N	Valor Máximo 12 Meses (*)	Precio de la acción los últimos 12 meses. Este es un campo de tipo flotante.
O	Valor Mínimo 12 Meses (*)	Precio mínimo de la acción en los últimos 12 meses. Este es un campo de tipo flotante.
P	Nombre Comprador [,]	Nombre del comprador en la última operación. Este es un campo tipo alfanumérico y termina con una coma como delimitador.
Q	Nombre Vendedor [,]	Nombre del vendedor en la última operación. Este es un campo tipo alfanumérico y termina con una coma como delimitador.
R	Acciones a la Alza (*)	Número de acciones a la alza durante el día. Este es un campo tipo entero.
S	Acciones a la Baja (*)	Número de acciones a la baja durante el día. Este es un campo tipo entero.
T	Acciones sin Cambio (*)	Número de acciones sin cambio durante el día. Este es un campo tipo entero.
U	Mercado [,] (**)	Mercado en que cotiza la acción. Este es un campo tipo alfanumérico y termina con una coma como delimitador.
V	Bolsa [,]	Bolsa donde se efectuó la transacción, en este caso es la BMV. Este es un campo tipo alfanumérico y termina con una coma como delimitador.
W	Folio BMV	Folio de la Bolsa Mexicana de Valores. Este es un campo de tipo entero.
X	Volumen de Venta	Volumen de la postura de venta. Este es un campo de tipo flotante.
Y	Precio de Venta (*)	Precio de la postura de venta. Este es un campo de tipo entero.
Z	Volumen Compra	Volumen de la postura de compra. Este es un campo de tipo entero.
a	Precio de Compra (*)	Precio de la postura de compra. Este es un campo de tipo flotante.
b	Folio de envío	Folio de envío siempre ascendente, reinicializado diario. Este es un campo de tipo entero.
c	Fecha Actual (*)	Fecha del día en que está cotizando la acción.
d	Fecha de caducidad	Fecha de expiración de una acción, provoca baja de la acción. Maa La acción aún está vigente, es decir, no tiene caducidad. Mab Este código causa baja.

e	Tasa Precio Códigos de Tasa Precio 1 2 4 8	Este campo indica la unidad de medida de la tasa o precio con que se va a regir la acción. Precio en pesos Tasas en porcentaje Índice Precio en dólares
g	Strike Price	Precio que se va a ejercer a la fecha de vencimiento, ya sea compra o venta. Este es un campo de tipo flotante.
i	Valor Suby [,]	Es la acción a que se refiere el warrant. Este es un campo tipo alfanumérico y termina con una coma como delimitador.
j	Agente Colocador [,]	Es la casa de bolsa que puso warrant. Este es un campo tipo alfanumérico y termina con una coma como delineador.
k	Precio Valor Suby	Es el precio que está cotizando la acción a que se refiere el warrant. Este es un campo de tipo flotante.
z	Fuente (**)	Quien envía la información.

(*) Ver APÉNDICE D

(**) Ver APÉNDICE E

Estos paquetes también están divididos en subtipos dependiendo de la información que se envíe.

Subtipos de BMV:

A Transacciones de Acciones de Renta Variable

B Posturas de Acciones de Renta Variable

D Evolución de Indices

F Transacciones de Warrants

G Posturas de Warrants

Subtipo A.- Transacciones de Acciones de Renta Variable.

Estos mensajes dan información de los hechos de las acciones.

Formato del paquete

[Tipo][Subtipo][Nombre del Indice][,][Campo1][Campo2].....[CampoN]

Tipo	Tipo de mensaje (ver APÉNDICE C); en este caso es '6'.
Subtipo	Clase de información que contiene el paquete; en este caso es 'A'.
Nombre de la Acción	Nombre de la acción de renta variable.
Campo	Los campos posibles en este mensaje: b,B,c,A,C,D,E,F,H,I,J,K,L,M,N,O,P,Q,d,U,W

Ejemplo del paquete expresado con variables:

[Tipo][Subtipo]['Nombre de la Acción][,][b][B][c][A][,][C][,][D][E][F][G][H][I][J][K][L][M][N][O][P][,][Q][,][d][U][,][W]

Subtipo B.- Posturas de Acciones de Renta Variable.

Estos mensajes envían información referente a posturas de acciones de renta variable.

Formato del paquete

[Tipo][Subtipo][Nombre de la Acción][,][Campo1][Campo2].....[CampoN]

Tipo	Tipo de mensaje (ver APÉNDICE C); en este caso es '6'.
Subtipo	Clase de información que contiene el paquete; en este caso es 'B'.
Nombre de La Acción	Nombre de la acción de renta variable.
Campo	Los campos posibles en este mensaje: A,B,c,D,U,X,Y,Z,a.

Ejemplo del paquete expresado con variables:

[Tipo][Subtipo][´Nombre de la Acción][,][A][,][B]c][U][,][X][Y][Z][a]

Subtipo D.- Evolución de Índices.

Estos mensajes envían información referente a la evolución de los índices de la Bolsa Mexicana de Valores e índices de Estados Unidos.

Formato del paquete

[Tipo][Subtipo][Nombre del Indice][,][Campo1][Campo2].....[CampoN]

Tipo	Tipo de mensaje (ver APÉNDICE C); en este caso es ´6´.
Subtipo	Clase de información que contiene el paquete; en este caso es ´D´.
Nombre del Indice	Nombre del Indice de renta variable. Actualmente los índices disponibles son: IPC, Índices Sectoriales e Índices de Estados Unidos.
Campo	Los campos posibles en este mensaje: B,b,C,D,M,J,R,S,T,E,F.

Ejemplo del paquete expresado con variables:

[Tipo][Subtipo][´Nombre de la Acción][,][B][b][C][,][D][M][J][R][S][T][E][F]

Subtipo F.- Transacción de Warrants.

Estos mensajes dan información de los hechos de los warrants. Los warrants son opciones de compra y venta de acciones que cotizan en la BMV.

Formato del paquete

[Tipo][Subtipo][Nombre de la Acción][,][Campo1][Campo2].....[CampoN]

Tipo	Tipo de mensaje (ver APÉNDICE C); en este caso es ´6´.
Subtipo	Clase de información que contiene el paquete; en este caso es ´F´.
Nombre de La Acción	Nombre del Warrant.
Campo	Los campos posibles en este mensaje son: b,B,c,A,C,D,E,F,H,I,J,K,L,M,N,O,P,Q,d,U,W,g,i,j,k

Ejemplo del paquete expresado con variables:

[Tipo][Subtipo][´Nombre de la Acción][,][b][B][c][A][,][C][,][D][E][F][G][H][I][J][K][L][M][N][O][P][,][Q][,][d][U][,][W][g][i][,][j][,][k]

Subtipo G.- Posturas de Warrants.

Estos mensajes envían información referente a posturas de warrants.

Formato del paquete

[Tipo][Subtipo][Nombre de la Acción][,][Campo1][Campo2].....[CampoN]

Tipo	Tipo de mensaje (ver APÉNDICE C); en este caso es ´6´.
Subtipo	Clase de información que contiene el paquete; en este caso es ´G´.
Nombre de La Acción	Nombre del warrant.
Campo	Los campos posibles en este mensaje: A,B,c,U,X,Y,Z,a.

Ejemplo del paquete expresado con variables:

[Tipo][Subtipo][´Nombre de la Acción][,][A][,][B]c][U][,][X][Y][Z][a]

Ejemplos:

Transacciones (hechos) de acciones de renta variable:

6AGFINBURB,b8638BMFcFAACC IND. COM. Y SER,CA,Da252FAVFHA255la252JB22808K11 LC11MD9Na2550F8357476PINTERACCIONES,QINVERLAT,sMaae1UCapitales Mex,W38

6AAEROMEXCPO,b8647BMEcAWFAACC IND. COM. Y SER,CA Db45Eb42FAVFHb46ib42JA1441393K167 LC19MC32701Ng66248230g1858182POPERADORABOLSA,QIXE,Dmaae1Ucapitales Mex,W12558

6AGFBB,b8658BMHcAWFAACC IND. COM. Y SER,CA,Db303Eb297FAVFHb311lb299JA5369167 K292LC4MC17536Nb3120g8582143PARKA,QBURSAMEX,dMaae1Ucapitales Mex, W18652

Posturas de acciones de renta variable:

6BAPASCO*,AACC IND. COM. Y SER,BMtcKVEUCapitales Mex,XC24Ya286ZC25aa283

6BBANACCIA,AACC IND. COM. Y SER.VMHcAWFUCapitales Mex,XD1Ya158ZE2aa157

6BVITRO*,AACC IND. COM. Y SER,BMHcAWFUCapitales Mex,XC69YA155ZC5ab146

Evoluciones de índices:

6DCONSTRUC.,BmuDb13166MC718JB361055Ea1282FKUE

6DSERVICIOS,BMtDb24392M36229554Jb6748189438Eb23396FKUE

6DIPC,BmuDb253707M110282095Jc1145973701013RA7S12Eg24484299316 FKUE

6DMEX,BNtDb6866M0J0.0Eb662FKUE
6DSPX,BnuDb60011M0J0.0Eb60024FKUE

Transacciones (hechos) de warrants:

6FCON605A,b10294BMpcAWFACANASTA DE ACC, CO, De4363486Ee4387199FAVFHe4424926le4311343
J0.0K84LOMONe443906500.0dMaae1UWarrants,W0G0.0I,j,k0.0
6FMSC606ADC004,b10361BMpcAWFAWARRANT DE ACC,CA,Db158Eb 126FAQFHb158lb154JC31K2LC5
MD2Nb158Ob105PABACO,QOPERADORABOLSA,dFDFe1UWarrants,W20023gb 476IMASECAB,Jobsa,KB562
6FCON605A,b9539BM^cANASTA DE ACC,CO,De4359529Ee4387199FAVFH e4424926Le4311343
J0.0K79L0M0Ne4439065O0.0dMaae1UWarrants,W0G0.0I,J,K0.0

Posturas de warrants:

6GCFR607ADC009,AWARRANT DE ACC,BMecAWFUWarrants,XC7Ya34ZC 1aa32
6GCFR607ADC009,AWARRANT DE ACC,BMjcAWFUWarrants,XD2Ya35ZC 1aa32
6GCFR607ADC009,AWARRANT DE ACC,BMOcAWFUWarrants,XC6Ya32ZC 5a3

Formato para información internacional (TIPO ´ 7´).

Este tipo de mensajes corresponde a aquellos que provengan de información de las bolsas de Estados Unidos (USA); pueden contener varios de los siguientes campos a la vez. El listado muestra todos los campos que pueden existir dentro de los paquetes para transacciones de acciones de USA, posturas de acciones de USA y transacciones de Opciones.

Campo	Contenido	Descripción
A	Tipo Valor [,]	Clasificación que le es otorgada a la acción. Este es un campo de tipo alfanumérico y termina con una coma como delimitador
B	Hora (*)	Hora en que se generó la última operación de la acción u opción, ya sea postura, transacción o status.
C	Status [,] Código de Status: C O R	Este campo contiene una secuencia de caracteres en donde cada carácter indica un status de la acción u opción. Es de tipo alfanumérico y termina con una coma como delimitador. Cierre: Con este código se envía el último valor de la acción cuando cierra el mercado. Operatividad: Con este código se envían los datos del último estado de la acción. Restauración: Con este código se envían los datos de una transacción o hecho de la acción, indicando que se debe corregir el hecho enviado anteriormente con el "folio de envío" y actualizarlo con los datos que se envían es este paquete. Si el folio se envío es igual a cero indica solamente una actualización del último estado de la acción.
D	Valor (*)	Precio de referencia para variaciones. Este es un campo de tipo flotante.
E	Valor de Referencia (*)	Precio actual de la acción o de la opción. Este es un campo de tipo flotante.
F	Fecha del Valor De Ref (*)	Fecha en que sucedió el precio de referencia.
H	Valor Máximo Del día (*)	Precio máximo de la acción o de la opción durante el día. Este es un campo de tipo flotante.
I	Valor Mínimo del día (*)	Precio mínimo de la acción o de la opción durante el día. Este es un campo de tipo flotante.
J	Importe Acumulado (*)	Importe acumulado en el día. Este es un campo de tipo entero.
K	Número de Operaciones (*)	Total de operaciones efectuadas de la acción u opción en el día. Este es un campo de tipo entero.
L	Volumen (*)	Volumen de la opción. Este es un campo de tipo entero.
M	Volumen Acumulado (*)	Volumen acumulado en el día. Este es un campo de tipo entero.

P	Bolsa [,] (***) Comprador	Bolsa donde se realiza la postura de compra. Este es un campo tipo alfanumérico y termina con una coma como delimitador
Q	Bolsa [,] (***) Vendedor	Bolsa donde se realiza la postura de venta. Este es un campo tipo alfanumérico y termina con una coma como delimitador.
U	Mercado [,] (**)	Mercado en que cotiza la acción u opción. Este es un campo tipo alfanumérico y termina con una coma como delimitador.
V	Bolsa [,] (***)	Bolsa donde se efectuó la transacción. Este es un campo de tipo alfanumérico y termina con una coma como delimitador.
X	Volumen de Venta	Volumen de la postura de venta. Este es un campo de tipo entero.
Y	Precio de Venta	Volumen de la postura de venta. Este es un campo de tipo entero.
Z	Volumen Compra	Volumen de la postura de compra. Este es un campo de tipo flotante.
a	Precio de Compra	Precio de la postura de compra. Este es un campo de tipo flotante.
b	Folio de envío	Folio de envío siempre ascendente, reinicializado diario. Este es un campo de tipo entero.
c	Fecha Actual (*)	Fecha del día en que está cotizando la acción u opción.
d	Fecha de Caducidad	Fecha de expiración de una acción u opción, provoca baja de la misma. Maa La acción aún esta vigente, es decir, no tiene caducidad. Mab Este código causa baja de la acción.
e	TasaPrecio Códigos de TasaPrecio 1 2 4 8	Este campo indica la unidad de medida de la tasa o precio con que se va a regir la acción. Precio en pesos Tasas en porcentaje Índice Precio en dólares
f	Fecha de Vencimiento (*)	Fecha de expiración de una opción.
G	Precio del Ejercicio (*)	Valor del Strike Price (Precio del Ejercicio).Este es un campo de tipo flotante.
H	Tipo de Opción [,] C P	Tipo de operación realizada Call Compra de la opción. Put Venta de la opción.

(*) Ver APÉNDICE D.

(**) Ver APÉNDICE E.

(***) Ver APÉNDICE H.

Estos paquetes también están divididos en subtipos dependiendo de la información que se envíe.

Subtipos de Información Internacional (USA):

A	Transacciones de Acciones de Renta Variable
B	Posturas de Acciones de Renta Variable
F	Transacciones de Opciones
G	Posturas de Opciones

Subtipo A.- Transacciones de Acciones de Renta Variable.

Estos mensajes dan información de los hechos que se están realizando de las acciones.

Formato del paquete:

[Tipo][Subtipo][´Nombre de la Acción][,][Campo1][Campo2].....[CampoN]

Tipo	Tipo de mensaje (ver APÉNDICE C); en este caso es ´7´.
Subtipo	Clase de información que contiene el paquete; en este caso es ´A´.
Nombre de la Acción	Nombre de la acción de renta variable.
Campo	Los campos posibles en este mensaje son los que tienen las siguientes variables: b,B,c,A,C,D,E,F,H,I,K,M,V,d,e,U.

Ejemplos del paquete expresado con variables:

[Tipo][Subtipo][´Nombre de la Acción´][,][c][B][f][h][,][a][P][,][Y[q][,]

Ejemplos:

Transacciones(hechos) de acciones USA:

7ATMX,b1BJ_cALFAUSA,CO,Db342Eb3475FAKHb3425lb3425K1MC2VT,dMaae8UADR,

7AKOF,b2BJ_cALFAUSA,CO,Da235Ea235FAKFHa235la235K1MB35VN,dMaae8UADR,

7AGTR,b5BJ_cALFAUSA,CO,Dc8625Ea85FAKFHc8625lc8625K1MB12VN,dMaae8UADR,

Posturas de acciones USA:

7BKOF,AUSA,BJ_cALFUADR,XA7Yc23625ZA5ac23375PN,QN,

7BDIS,AUSA,BJ_cALFUADR,XB1Yb6075ZA4aa605PN,QN,

7BSDK,AUSA,BJ_cALFUADR,X206Yb275Z258ac2625PN,QN,

Transacciones de Opciones:

7FGTRAU,b1cALFBJdfATFga75hC,CO,Dd10625LC2VC,Hd10625ld10625K1M0J0.0FAKFEc1125dMaa

7FGTRU,b1cALFBJdfGTFga75hC,CO,Dd23125LB4VA,Hd23125ld23125K1M0J0.0FAKFEd23125dMaa

7FGTRAU,b2cALFBJgfATFga75hC,CO,Dc1125LB17VC,Hc1125lc1125K2MB37Ja41625FAKFEc1125dMaa

7FVTOQA,b1cALFBJOfERFg5H5P,CO,Db755LC2VC,Hb75lb75K1MC2JB15FAKFEb75dMaa

Posturas de Opciones:

7GTVAB,cALFBJafATFgA1hC,ac16125PC,Yc15875QC,
7GTVAC,cALFBJafATFg15hC,ac11125PC,Yc10875QC,
7GTVMC,cALFBJafATFg15hP,ac125PC,Y0.0Q,
7GTVMW,cALFBJafATFgA175hP,ac16125PC,Y0.0Q,

2.1.1.4 Consideraciones Importantes.

Las siguientes consideraciones deberán ser tomadas en cuenta para el buen funcionamiento del sistema:

- Las aplicaciones de la Fuente de Información Financiera y el Sistema de Alimentación deberán estar ejecutándose permanentemente con el fin de que la tabla de acciones se mantenga actualizada, así se evitará que el Sistema de Alimentación filtre paquetes que no encuentren la referencia adecuada.

- En caso de que un paquete no encuentre la referencia adecuada se marcará como error. En el archivo LOG se encontrará la leyenda del error, la hora en que ocurrió y su descripción.

- Aproximadamente cada 3 meses, o antes si es necesaria la actualización de las tablas de Mercados, Tipos de Valor y Casas de Bolsa.

- Es recomendable depurar frecuentemente los archivos .LOG generados por el Sistema de Alimentación, ya que estos pueden llegar a medir hasta 8Mb por día si se almacena toda la información, y pueden ocasionar que se agote el espacio libre en disco duro provocando que la Fuente de Información Financiera no pueda escribir y se dañen sus bases de datos.

2.2. Uso de las API's de la plataforma digital

Las APIS's le dan la capacidad al programador de desarrollar sus propios sistemas distribuidores de información en tiempo real en una red local. Da acceso a los mecanismos de distribución en la red y controla el flujo de la información. Permite a los desarrolladores independizar la información de sus proveedores originales y darle su propio valor agregado a los sistemas sin tener grandes conocimientos sobre comunicaciones en tiempo real y redes.

Las API's para la publicación están escritas para usarse en programas en C y los sistemas actualmente soportados incluyen a MS-Windows NT, OS/2, SunOS, HP-UX y Solaris. Las API's manejan todas las comunicaciones del sistema con aplicaciones cliente en una red de esta plataforma digital. Manejan funciones tales como seguridad y verificación, distribución de información y control de recursos y agnación. Las funciones en el modulo publicador son usadas igualmente en toda la plataforma, de manera se trabaja en forma consistente en todo el sistema.

La información publicada usando las API's del publicador podrá ser usada por estaciones de trabajo y aplicaciones construidas con las API's del solicitador. Cuando una aplicación actúa como traductor de protocolo, su función es monitorear los puertos seriales de la computadora esperando la llegada de información de la fuente y cuando esta se recibe, se traduce al formato de la plataforma digital y la suministra al publicador para su distribución. Se usan las API's del publicador para poner la información disponible al usuario y para que sus requisiciones sean satisfechas.

Este tipo de sistemas se desarrollan cuando se desea hacer disponible información en tiempo real. La información puede ser generada dentro de la compañía o bien por servicios externos.

El término *servicio* es usado para describir cualquier suministro de información que es comunicado en formato electrónico. La conexión entre una cualquier servicio de

información externo o interno es manejada por una aplicación servidora referida como *servidor de información,* dedicado únicamente a su servicio.

El servidor también controla cómo será distribuida la información dentro de una red, por ejemplo, cuando una aplicación cliente desea accesar la información de un servicio, debe primero registrarse en él y ser validada. Una vez que el usuario se ha registrado, el servidor maneja toda la distribución de datos hacia él. También se monitorean requisiciones del usuario para asegurarse que sus limites no se han excedido y para detectar y resolver restricciones.

Una vez que el usuario se ha registrado en un servicio, puede tener acceso a la información en tiempo real en forma de objetos. Para la plataforma digital, un objeto es un espacio de presentación de datos que es controlado por un servidor.

Descripción de las funciones incorporadas en las APIs para Publicación

A continuación se describirán las sintaxis, los parámetros y valores de retorno para las rutinas incorporadas en las API de desarrollo para publicación. A si mismo se da un breve ejemplo de como usar estas rutinas con comentarios asociados, de ser necesarios. La siguiente es una lista de todas las funciones disponibles:

InvStartPublisher	Arranca el publicador
InvStopPublisher	Detiene el publicador
InvCreateObject	Crea un objeto en la memoria
InvDestroyObject	Destruye un objeto
InvPublishObject	Publica la información de un objeto en memoria.
InvSetOnLine	Pone el estado del servicio como disponible.
InvSetOffLine	Pone el estado del servicio como no disponible.
InvGetMessage	Obtiene el siguiente mensaje del publicador.

InvStartPublisher
Arranca el publicador

Declaración
*Unsigned short InvStartPublisher (char *nombre,short int servicios, unsigned long opciones, unsigned short slots, unsigned short snapslots)*

Descripción de los parámetros

Nombre
Es el nombre del servicio que esta por ser arrancado.

Servicios
Para el caso de las API's de publicación este es siempre .

Opciones
Son un conjunto de banderas que configuran la sesión que se arrancará:
IRO_NO0RMAL. - Operación estándar no interactiva. El publicador no aceptara solicitudes de objetos que no estén en el cache ni solicitudes de inserción.
IRO_INTERACTIVE - El publicador aceptara y comunicará solicitudes de objetos que no estén en el cache así como solicitudes de inserción. En este modo también se eliminaran los objetos que ya no estén siendo utilizados de manera dinámica.
IRO_ENABLE_INSERTS – El publicador comunicara a la aplicación cualquier mensaje de inserción.
IRO_ENABLEREFRESH – El publicador comunicara un mensaje a la aplicación cuando el cliente solicite un refresco manual de información.

Slots
Numero total de espacios disponibles en el cache, cada espacio puede alojar un objeto.
Snapslots
Numero de espacios para objetos instantáneos en el cache. Un objeto instantaneo es aquel que es publicado una sola vez.

Códigos de retorno

IRE_PAPI_ALREADYSTARTED
IRE_INTERNALERROR
IRE_NOMEMORY

Esta función debe ser llamada antes de que cualquier otra función relacionada con el servidor. El número total de slots debe ser menor a 32767.

InvSetOnLine
Esta rutina pone al servicio en modo disponible.

Declaración

unsigned short InvSetOnLine(short int servicio);

Descripción de parámetros

Servicio
Para la API del publicador este siempre es 1.

Códigos de retorno

IRE_PAPI_ALREADYSTARTED
IRE_INTERNALERROR
IRE_NOMEMORY

InvGetMsg
Obtiene el siguiente mensaje de la cola de mensajes del publicador.

Declaración

Unsigned int InvGetMsg(PTYPMSG msg);

Descripción de los parámetros

Msg
Es un apuntador a la estructura TYPMSG. Esta estructura contiene un campo llamado usMsgType cuyos posibles valores son:

INV_OBJ_INSERT_MSG	Solicitud de Inserción de objeto
INV_OBJ_REQUEST_MSG	Solicitud de publicación objeto
INV_OBJ_RELEASE_MSG	Solicitud de liberación de objeto.

Las estructuras posibles para estos tres tipos de mensaje son:

INV_OBJ_INSERT	Insertar mensaje
INV_OBJ_REQUEST	Publicar mensaje
INV_OBJ_RELEASE	Liberar objeto

El código de estas estucturas se muestra a continuacion:

INV_OBJ_INSERT

```
typedef struct INV_INSERT_MESAGE
{
        short usMsgType;
        short idSvc;
        long lSeq;
        CHAR64 szName;
        CHAR16 szOrig;
        unsigned short fsUpdType;
        unsigned long flType;
        short iRow;
        short iCol;
        short iCRow;
        short iCCol;
        short fsC;
        unsigned short cbData;
        unsigend char *pbData;
};
```

INV_OBJ_REQUEST

```
Typedef struct
{
        unsigned short usMsgType;
        unsigned short idSvc;
        long lSeq;
        CHAR64 szName;
        unsigned short fsReqType;
        INVPARAM invParam;
}
```

INV_OBJ_RELEASE

```
typedef struct INV_OBJ_RELEASE
{
        unsigned short usMsgType;
        unsigned short idSvc;
        long lSeq;
        CHAR64 szName;
        unsigned short fsReqType;
        INVPARAM invParam;
};
```

InvCreateObject
Crea un objeto en la memoria del publicador.

Declaración

unsigned short InvCreateObject(long *plSeq, INV_OBJ_DEFN * pInvObjDefn);

Descripción de los parámetros

plSeq
Numero único de secuencia generado por esta función.

PInvObjDef
Esta es la definicion del objeto que sera creado. El código de la estructura INV_OBJ_DEFN se muestra a continuación.

INV_OBJ_DEFN

```
typedef struct INV_OBJ_DEFN
{
        short idSvc;
        BOOL fPermanent;
        unsigned short fsObjType;
        CHAR64 szName;
        unsigned short cRow;
        unsigned short cCol;
        unsigned short fsPageType;
        unsigned short usFore;
        unsigned short usBack;
        unsigned short idFidList;
        long lPE;
        short sPriority;
        CHAR szText[80];
};
```

Esta estructura es de particular interés porque define los dos tipos de objetos básicos que usa la plataforma digital: las páginas y los registros. En las siguientes líneas se describen algunos de los parámetros mas importantes.

idSvc
Es un identificador que para motivos de publicación vale siempre uno.

fPermanent
Indica si un objeto es permanente o no. Un objeto permanente permanece en la memoria hasta ser destruido por la rutina InvDestroyObject() , mientras que un objeto no permanente puede ser borrado por el API cuando lo considere conveniente.

fsObjectType
Indica el tipo de objeto que estamos creando puede ser:

INV_OBJECT_TYPE_PAGE	Crea una página
INV_OBJECT_TYPE_RECORD	Crea un registro

szName
Es una cadena terminada con cero con el nombre del objeto que se creará.

CRows
Si el objeto creado es una página, este parametro define el número de renglones.

CCols
Si el objeto creado es una página, este parametro define el número de columnas.

fsPageType
Si el objeto creado es una página, este parametro define que tipo de página se creará. La plataforma digital soporta dos tipos de páginas: de solo caracteres o en color (también llamada *mezclada*).

usBack y *usFore*
Definen el color de fondo y de texto para el objeto creado.

idFidList
Se usa si el objeto que se esta creando es un registro y define el número de la mascara que se desea usar.

IPE
Determina los derechos usuario. Por lo regular se mantiene el valor en cero.

sPriority
Asigna un valor de prioridad al objeto. Este valor se usa para decidir si será eliminado o no de la memoria del publicador en condiciones de recursos bajos.

szText
Es un parámetro reservado.

En general esta función crea el espacio de presentación para el objeto en la memoria del publicador.

Sus códigos de retorno en caso de error son:

IRE_NO_FREE_SLOTS	No hay espacio disponible para mas objetos.
IRE_OBJECT_ALREADY_EXISTS	El objeto que se desea crear ya existe.

InvPublishObject
Publica datos de un objeto.

Declaración

unsigned short InvPublishObject(unsigned short usMsgType, void *pMsg);

Descripción de los parámetros

usMsgType
Define el tipo de mensaje que será publicado. Hay dos tipos de mensaje:
INV_OBJ_DATA_MSG – Contiene información de objeto.
INV_OBJ_STATUS-MSG – Indica un cambio en el estado de un objeto.

pMsg
Es un apuntador al mensaje. El mensaje deberá contener un apuntador a alguna de las siguientes estructuras, dependiendo del tipo de mensaje, como se indica en usMsgType.

INV_OBJ_DATA – Contiene información de objeto.
INV_OBJ_STATUS – Indica un cambio en el estado de un objeto.

El detalle de estas estructuras se muestra a continuación.

typedef struct INV_OBJ_DATA

```
{
        short usMsgType
        short idSvc;
        long lseq;
        CHAR64 szName;
        CHAR16 szOrig;
        unsigned short fsUpdType;
        unsigned long flType;
        unsigned short iRow;
        unsigned short iCol;
        unsigned short iCRow;
        unsigned short iCCol;
        short cbData;
        unsigned char *pbData
}
```

A continuación se describen los parámetros mas importantes de esta estructura:

usMsgType
Define el tipo de mensaje, para un objeto de datos es del tipo INV_OBJ_DATA_MSG

idSvc
Es un identificador unico para el servicio. Para un servicio publicador basado en API's es siempre 1.

lseq
Es un identificador único para el objeto, se obtiene como valor de retorno de la función InvCreateObject.

szName
Es una cadena terminada con nulo conteniendo el nombre del objeto. Asegurese de que no existen espacios en ella.

szOrig
Es el origen del objeto. Debe ser apuntado a nulo.

fsUpdType
Indica el tipo del objeto como sigue:

INV_OBJECT_TYPE_PAGE	Es un objeto basado en páginas.
INV_OBJECT_TYPE_RECORD	Es un objeto basado en registros.

pbData
Es un apuntador al la información que se va a enviar. El largo de la cadena es especificado en pbData. El formato de información esta especificado de acuerdo al tipo de información que se desea enviar.

Si se trata de un registro, la información debe ser encapsulada en una cadena de caracteres con el siguiente formato:

0x1F*NumeroFID1*0x1Edato10x1F*NumeroFID2*0x1Edato2 .... 0x1C

Nótese que el byte 0x1F inicia un dato seguido del número FID que identifica al campo en la plataforma digital. Después un byte 0x1F indica el principio de los datos asociados con este campo. El tren de datos se termina con un byte 0x1C.

```
typedef struct INV_OBJ_STATUS
{
        short usMsgType
        short idSvc;
        long lseq;
        BOOL FChange
        CHAR64 szName;
        unsigned short fsStatus;
        unsigned short fsRspType;
        short sRspType
        short sRspCode
        CHAR szRspText [ INV_MAX_RESPONSE_TEXT ];
        INVPARAM invParam;
}
```

usMsgType
Define el tipo de mensaje, para un objeto de datos es del tipo INV_OBJ_DATA_MSG

idSvc
Es un identificador único para el servicio. Para un servicio publicador basado en API's es siempre 1.

lseq
Es un identificador único para el objeto, se obtiene como valor de retorno de la función InvCreateObject.

fChange
Indica si el nombre del objeto ha cambiado o no.

szName
Es una cadena terminada con nulo conteniendo el nombre del objeto. Asegurese de que no existen espacios en ella.

InvDestroyObject
Destruye a un objeto.

Declaración

unsigned short InvDestroyObject(PSZ pszName, short sRc, PSZ pszErrorMsg);

Descripción de los parámetros

pszName
Es una cadena terminada en nulo con el nombre del objeto que será destruido.

sRc
Sirve para especificar un código de error, si no se necesita se puede hacer 0.

PszErrorMsg
Manda un mensaje de error a la terminal que solicito el objeto, este mensaje se desplegara en la barra de eventos en la parte inferior de la estación de trabajo.

Hay otras dos funciones que merecen ser mencionadas:

InvSetOffLine(1);
InvStopPublisher(void);

La primera pone al servicio fuera de línea y recibe un parámetro que siempre es 1. La segunda detiene todas las funciones del publicador. Aunque formalmente deberían invocarse al final del programa, en realidad son mas frecuentemente usadas por sistemas tolerantes a fallas, cuando se detectan errores y se desea re iniciar el proceso.

Cabe mencionar que no es estrictamente necesario que estas rutinas se usen al finalizar el programa y su uso es opcional.

2.3. Formato de la Información Emitida por el Proveedor.

El sistema recibe información desde varias fuentes generadoras de información ubicadas en diversos puntos geográficos. El traductor de protocolo se encarga de separar perfectamente la información y de darle un formato para que el usuario a través de una terminal especializada pueda solicitar la información requerida.

Información que es extraída

- Registros de Noticias
- Registros de Páginas
- Registros de Información Financiera

El sistema Windows es un ambiente eficiente de trabajo para una PC, en el cual es posible ejecutar más de una aplicación a la vez además de transferir información entre ellas. A través de Windows el usuario puede visualizar la información en un formato agradable.

2.3.1. Formato de despliegue para las noticias.

Las noticias al igual que las páginas pueden ser enviadas en bloques de textos con el fin de permitir el envío de información más relevante en lo que se transmite el resto de la noticia, es decir una vez comenzado el envío de bloques de la noticia es posible enviar Información Financiera (o incluso otras noticias), sin tener que esperar la transmisión de todo el texto de la noticia. Existen dos subtipos de noticias.

Subtipos de noticias.

- Encabezado de la noticias
- Cuerpo de la noticia

Encabezado de la noticia: Realiza una descripción de la noticia. Es indispensable que este se reciba previamente para que se pueda formar la noticia.
Cuerpo de la noticia: Contiene los bloques de texto de la noticia indicada. Es necesario haber recibido previamente el mensaje del encabezado de la nota respectiva para que se pueda procesar.

El usuario puede escribir la palabra noticias solicitando así todas las noticias que arriben de la fuente de información

En la figura 2.3.1 se muestra la forma en la que será desplegada la lista de noticias para que el usuario pueda visualizarlas y desplazarse a través de ellas y seleccionar la que desee consultar.

Figura 2.3.1. Formato de despliegue de Noticias.

Al elegir la deseada el usuario deberá oprimir la tecla control y accionar dos veces el botón derecho del ratón para que aparezca el cuerpo de la noticia, la figura 2.3.2 muestra el cuerpo e la noticia seleccionada.

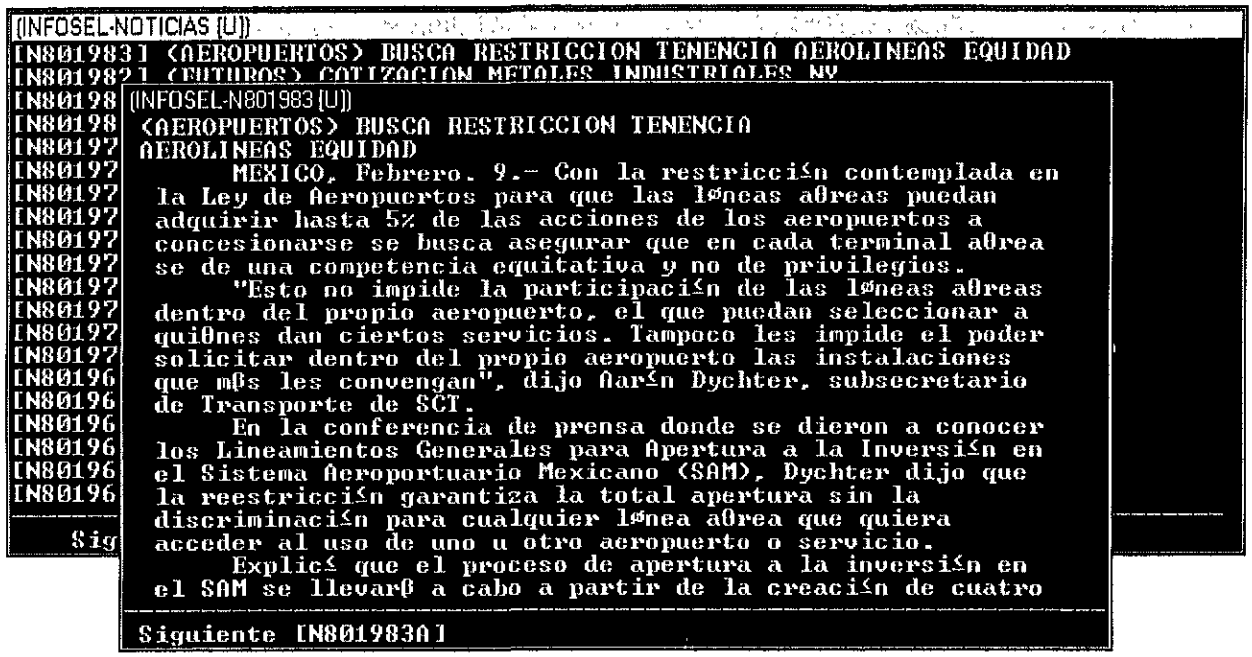


Figura 2.3.2. Formato de despliegue para el cuerpo de las noticias.

2.3.2. Formato de despliegue para páginas.

Para las páginas existen también dos subtipos de ellas: el encabezado y el cuerpo.

Existen dos clases de envío:

- Envío de la página completa: Cuando se realiza este tipo de envío aparecen los dos subtipos de páginas, es decir, el encabezado y el cuerpo, siempre apareciendo primero el encabezado.
- Envío de la información cambiante: Sólo se envía la información que haya cambiado en ese momento.

Subtipos de páginas:

- Encabezado de la página
- Contenido o cuerpo de la página

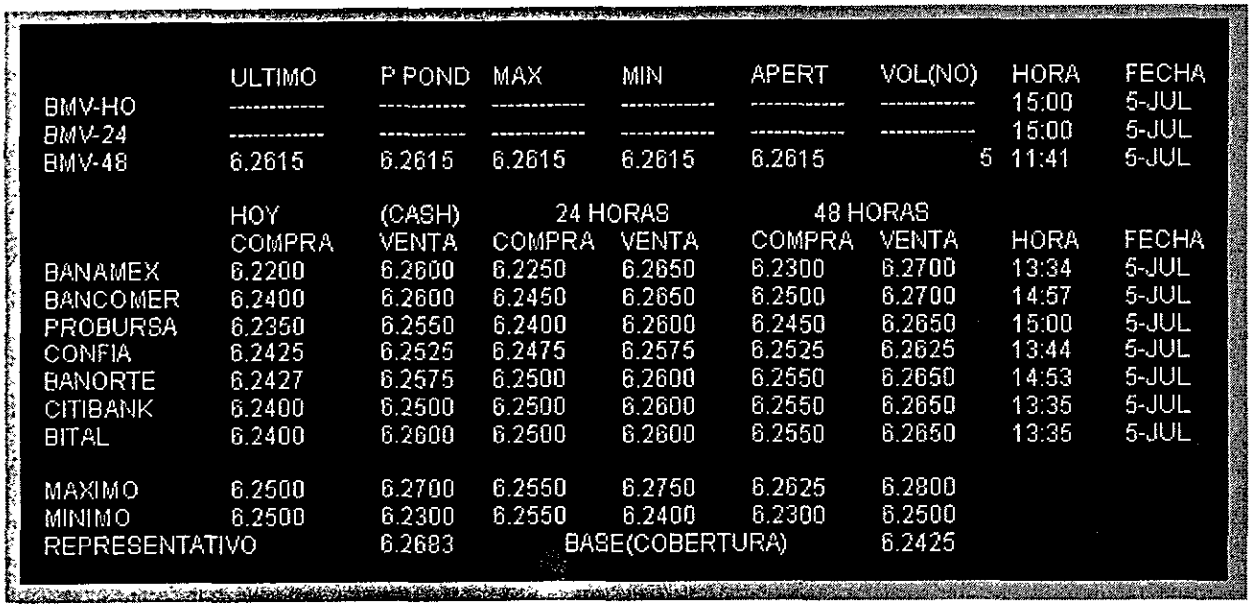
Encabezado de la página: Contiene la información del encabezado de la página. Para que una página pueda comenzar a recibirse es indispensable haber recibido previamente este registro.

Cuerpo o contenido de la noticia: Contiene el texto, cuerpo o información de la página especificando la posición que ocupa dentro de la misma.

Los formatos de despliegue de las páginas presentan información en tiempo real, de una forma accesible y en una agrupación lógica por tipo de información, éstas presentan la última cotización que se haya recibido, es decir, no manejan datos históricos.

Las páginas completas pueden ser copiadas a otra aplicación de Windows para realizar estadísticas y gráficas de los datos extraídos de estas.

- Texto ASCII: Es el texto, cuerpo o información de la página. Este puede estar compuesto de cualquier secuencia de caracteres ASCII.

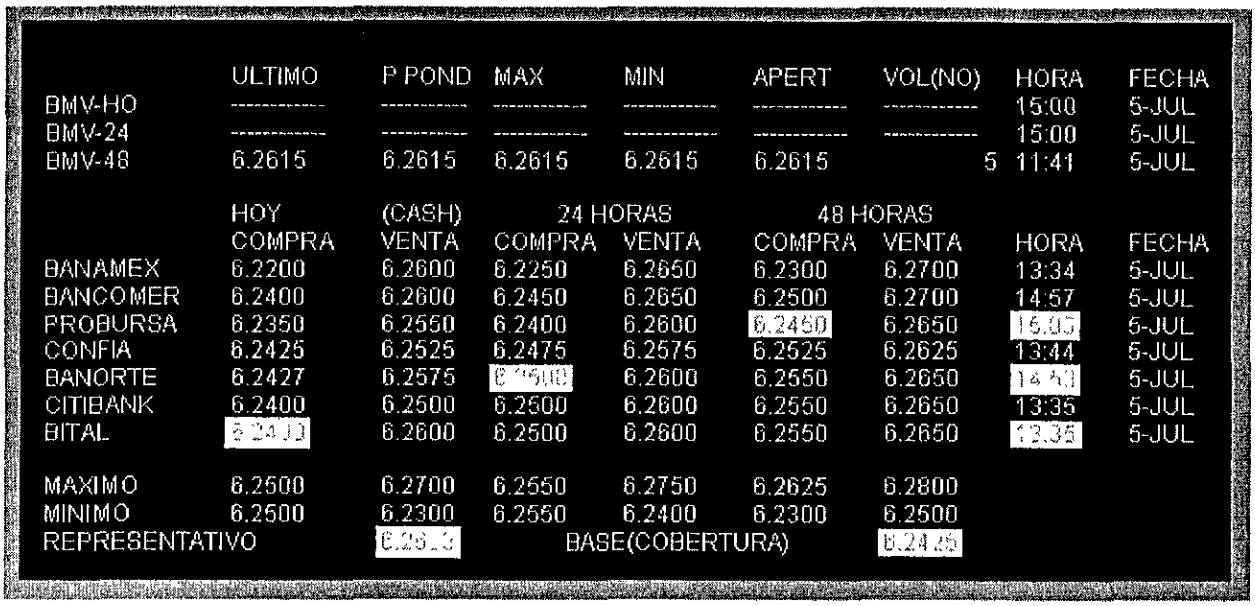


	ULTIMO	P POND	MAX	MIN	APERT	VOL(NO)	HORA	FECHA
BMV-HO	----------	----------	----------	----------	----------	----------	15:00	5-JUL
BMV-24	----------	----------	----------	----------	----------	----------	15:00	5-JUL
BMV-48	6.2615	6.2615	6.2615	6.2615	6.2615	5	11:41	5-JUL

	HOY	(CASH)	24 HORAS		48 HORAS			
	COMPRA	VENTA	COMPRA	VENTA	COMPRA	VENTA	HORA	FECHA
BANAMEX	6.2200	6.2600	6.2250	6.2650	6.2300	6.2700	13:34	5-JUL
BANCOMER	6.2400	6.2600	6.2450	6.2650	6.2500	6.2700	14:57	5-JUL
PROBURSA	6.2350	6.2550	6.2400	6.2600	6.2450	6.2650	15:00	5-JUL
CONFIA	6.2425	6.2525	6.2475	6.2575	6.2525	6.2625	13:44	5-JUL
BANORTE	6.2427	6.2575	6.2500	6.2600	6.2550	6.2650	14:53	5-JUL
CITIBANK	6.2400	6.2500	6.2500	6.2600	6.2550	6.2650	13:35	5-JUL
BITAL	6.2400	6.2600	6.2500	6.2600	6.2550	6.2650	13:35	5-JUL
MAXIMO	6.2500	6.2700	6.2550	6.2750	6.2625	6.2800		
MINIMO	6.2500	6.2300	6.2550	6.2400	6.2300	6.2500		
REPRESENTATIVO		6.2683		BASE(COBERTURA)		6.2425		

Figura 2.3.2. Formato de despliegue de un registro de una página financiera.

La figura 2.3.2 muestra el formato de despliegue de las páginas financieras. Pero como esta información se actualiza en tiempo real el usuario puede observar como se realiza esta actualización por medio de un encendido luminoso del campo que se actualiza. Lo cual se muestra en la figura 2.3.3.



	ULTIMO	P POND	MAX	MIN	APERT	VOL(NO)	HORA	FECHA
BMV-HO	----------	----------	----------	----------	----------	----------	15:00	5-JUL
BMV-24	----------	----------	----------	----------	----------	----------	15:00	5-JUL
BMV-48	6.2615	6.2615	6.2615	6.2615	6.2615	5	11:41	5-JUL

	HOY	(CASH)	24 HORAS		48 HORAS			
	COMPRA	VENTA	COMPRA	VENTA	COMPRA	VENTA	HORA	FECHA
BANAMEX	6.2200	6.2600	6.2250	6.2650	6.2300	6.2700	13:34	5-JUL
BANCOMER	6.2400	6.2600	6.2450	6.2650	6.2500	6.2700	14:57	5-JUL
PROBURSA	6.2350	6.2550	6.2400	6.2600	6.2450	6.2650	15:05	5-JUL
CONFIA	6.2425	6.2525	6.2475	6.2575	6.2525	6.2625	13:44	5-JUL
BANORTE	6.2427	6.2575	[illegible]	6.2600	6.2550	6.2650	14:43	5-JUL
CITIBANK	6.2400	6.2500	6.2500	6.2600	6.2550	6.2650	13:35	5-JUL
BITAL	6.2400	6.2600	6.2500	6.2600	6.2550	6.2650	12:35	5-JUL
MAXIMO	6.2500	6.2700	6.2550	6.2750	6.2625	6.2800		
MINIMO	6.2500	6.2300	6.2550	6.2400	6.2300	6.2500		
REPRESENTATIVO		6.26[illegible]		BASE(COBERTURA)		6.24[illegible]		

Figura 2.3.3. Actualización a los campos de la página financiera.

2.3.3. Formato de despliegue para información financiera.

Es importante la definición de los principales conceptos financieros involucrados en las finanzas, la diferencia principal entre los hechos y las posturas, así como la descripción general de la información que conforma cada uno de los registros transmitidos para contar con un mejor entendimiento de la información manejada por los usuarios, y así proponer los formatos de despliegue de la información financiera. Empecemos por las definiciones importantes.

Definiciones.

Acciones. Título nominativo que representa una de las partes iguales en que se divide el capital social de una empresa, e incorpora los derechos y obligaciones de los socios. Los recursos se destinan para el financiamiento a largo plazo de las empresas, y su vencimiento es indefinido, es decir, el plazo de las acciones depende de la permanencia de la empresa. El rendimiento se obtiene de dos maneras:
Por diferencia de capital y Dividendos.

Warrants. títulos opcionales de compra o de venta emitidos por intermediarios bursátiles o empresas. A cambio del pago de una prima, el tenedor adquiere el derecho opcional de comprar o vender al emisor un determinado número de valores a los que se encuentran referidos, a un precio de ejercicio y dentro de un plazo estipulado en el documento.

Indices. La finalidad del cálculo o construcción de índices de precios accionarios es obtener el valor representativo de un conjunto de acciones, en un momento especifico de tiempo. Los índices calculados y publicados son:

IPC Indice de Precios y Cotizaciones. Es el principal indicador del comportamiento del mercado accionario en su conjunto y expresa un valor en función de los precios de una muestra balanceada, ponderada y representativa del total de acciones cotizadas en la BMV. La muestra se revisa bimestralmente y se integra alrededor de 35 emisoras de distintos sectores de la economía. Aplicado en su actual estructura desde 1978, el IPC expresa en forma fidedigna la situación del mercado bursátil y constituye un indicador altamente confiable.

Indices Sectoriales. Son el conjunto de índices que presentan el comportamiento de ciertas áreas de actividad económica o sectores, que la BMV define convencionalmente. El método utilizado para el cálculo de los índices sectoriales es el mismo que para el IPC, variando únicamente el tamaño de la muestra y los valores que la integran.

INMEX, Indice México. Este índice es utilizado como un subyacente, en la medida en que representa un factor básico para la emisión de títulos derivados. La muestra empleada en su construcción abarca de 20 a 25 emisoras en sus series más representativas y con los niveles más altos de bursatilidad. Para su selección se toman también en cuenta su liquidez, representatividad sectorial y valor de mercado. La ponderación de cada emisora no puede exceder del 10%, y la muestra se revisa semestralmente.

IP-MMEX. Indice de Precios del Mercado para la mediana empresa Mexicana. Es un indicador diseñado para reflejar el comportamiento de la mediana empresa. Al cierre de octubre de 1997, la muestra del IP-MMEX se conformaba por 23 series correspondientes a las 18 emisoras que habían efectuado oferta pública en este mercado. La muestra se revisa bimestralmente, y la fórmula empleada para su cálculo es la misma que para el IPC.

Clasificación de las Operaciones.

Posturas. Ofertas de compra/venta de acciones y warrants en volumen y precio.

Hechos. Ofertas concretadas de compra/venta de acciones y warrants.

Características comunes para el despliegue de la Información.

Es deseable cumplir algunas características para el despliegue de los datos ya que éste se convertirá en un punto de acceso a todos los servicios de información externa proporcionando un valor agregado a la aplicación.

Servicio de Ventanas. Los datos que son requeridos de algún servicio de información deberá ser desplegada en su propia ventana. Cualquier número de ventanas que puedan ser abiertas simultáneamente , podrán desplegar páginas, registros y sesiones de diferentes servicios. Es importante que exista la posibilidad de personalizar las ventanas en términos de posición, color, tipo de letra utilizada, títulos, etc. También es importante el poder ajustar el tamaño de las ventanas para desplegar solo los datos relevantes al usuario.

Capas. Una vez que pone un número de ventanas dentro de la pantalla y son personalizadas de acuerdo a las preferencias de los usuarios, es deseable que estas preferencias puedan ser almacenadas como una capa para uso futuro. El usuario en cualquier momento podrá obtener estas capas y desplegarlas sobre la pantalla como cuando esta fue salvada. Si un número de capas regularmente son usadas, podrá existir la posibilidad de alternar entre ellas en forma fácil y rápida. Es importante que exista la posibilidad de verlas simultáneamente.

2.3.3.1. Formato para hechos de capitales.

La información que acompaña a los registros de hechos capitales es la siguiente:

Nombre de la Acción, Folio de Envío, Hora, Fecha Actual, Tipo Valor, Status, Valor, Valor de Referencia, Fecha de Valor de Referencia, Valor máximo del día, Valor mínimo del día, Importe acumulado, Numero de operaciones, Volumen, Volumen Acumulado, Valor máximo a 12 meses, Valor mínimo a 12 meses, Nombre del comprador, Nombre del vendedor, Fecha de caducidad, Mercado y Bolsa.

A continuación se propone el formato de despliegue para los Hecho Capitales:

Figura. 2.3.3.1.1. Formato de Despliegue de Hechos Capitales

2.3.3.2. Formato para hechos de warrants.

La información que acompaña a los registros de hechos de warrants es la siguiente:

Nombre de la Acción, Folio de Envío, Hora, Fecha Actual, Tipo Valor, Status, Valor, Valor de Referencia, Fecha de Valor de Referencia, Valor máximo del día, Valor mínimo del día, Importe acumulado, Numero de operaciones, Volumen, Volumen Acumulado, Valor máximo a 12 meses, Valor mínimo a 12 meses, Nombre del comprador, Nombre del vendedor, Fecha de caducidad, Mercado, Bolsa., strike price, valor suby, agente colocador y precio valor suby.

A continuación se propone el formato de despliegue para los Hecho de warrants:

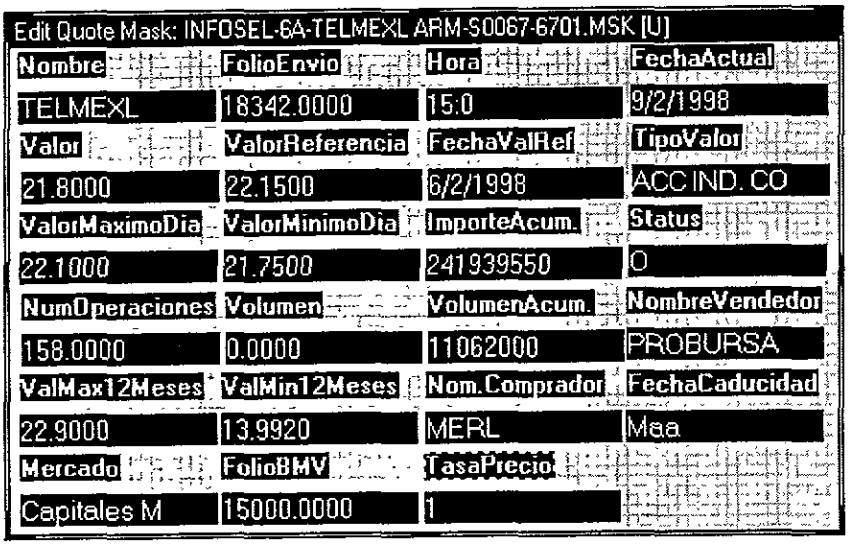


Figura. 2.3.3.2.1. Formato de Despliegue de Hechos Warrants

2.3.3.3. Formato para posturas de capitales.

La información que acompaña a los registros de posturas capitales es la siguiente:

Nombre de la Acción, Tipo Valor, Hora, Fecha Actual, Mercado, Volumen de Venta, Precio de Venta y Volumen de Compra y Precio de Compra.

A continuación se propone el formato de despliegue para las posturas Capitales:

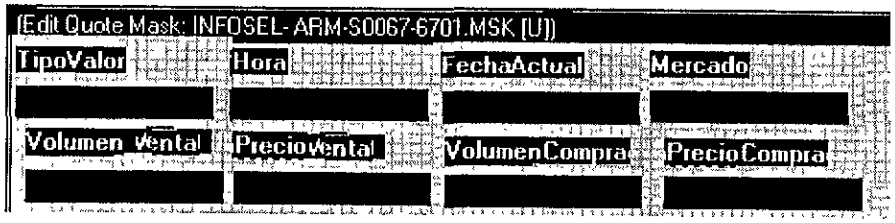


Figura. 2.3.3.3.1. Formato de Despliegue de Posturas capitales

2.3.3.4. Formato para posturas de warrants.

La información que acompaña a los registros de posturas de warrants es la siguiente:

Nombre de la Acción, Tipo Valor, Hora, Fecha Actual, Mercado, Volumen de Venta, Precio de Venta, Volumen de Compra y Precio de Compra.

A continuación se propone el formato de despliegue para las posturas warrants:

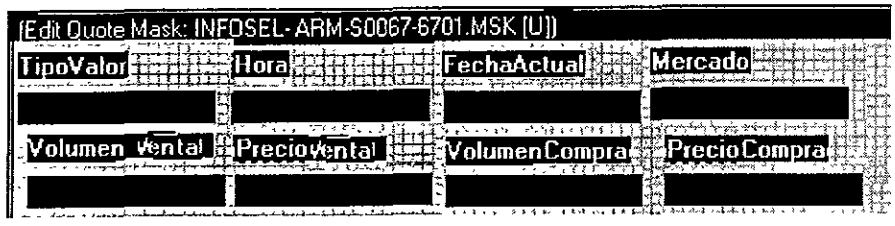


Figura. 2.3.3.4.1. Formato de Despliegue de Posturas Warrants

2.3.3.5. Formato para índices.

La información que acompaña a los registros de índices la siguiente:

Hora, Folio de Envío, Status, Valor, Volumen acumulado, Importe Acumulado Acciones a la Alza, Acciones a la Baja, Acciones sin cambio, Valor de Referencia y Fecha de Valor de Referencia.

A continuación se propone el formato de despliegue para los Indices.

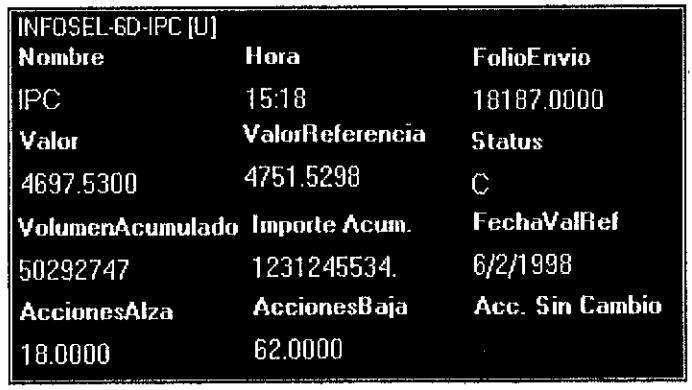


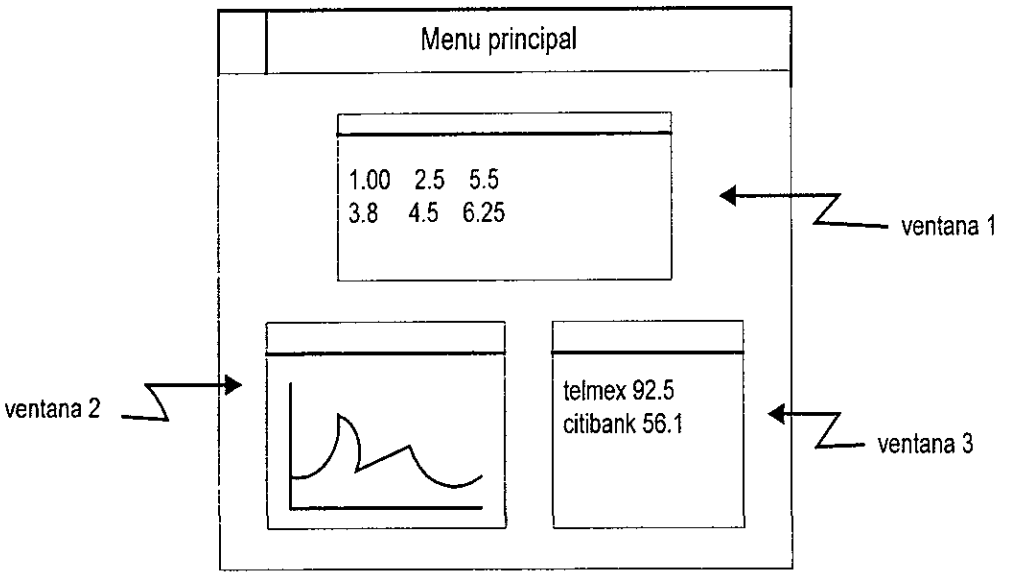
Figura. 2.3.3.5.1. Formato de Despliegue de Indices.

2.4. Requerimientos.

Toda solicitud que haga un usuario al proveedor es un requerimiento el cual va a resolver una necesidad que se tenga referente al sistema para su automatización. Los requerimientos se obtienen como consecuencia de la interacción con los usuarios de nivel operativo de la organización cliente.

Interfaz con el Usuario.

La interfaz con el usuario deberá ser amigable y fácil de manejar, así como intuitiva y debe de ser una interfaz gráfica para el usuario (GUI). La información tiene que desplegarse simultáneamente desde diferentes servicios de información y el usuario podrá solicitar registros, páginas de diferentes fuentes de información y visualizarla en una sola pantalla como se muestra en la figura 2.4 Una vez desplegadas las ventanas el usuario podrá personalizarlas cambiando la posición, el tamaño, etc; esto debido a que el sistema correrá en un ambiente Windows, que nos proporciona una interfaz eficiente e intuitiva para trabajar con una PC. En un ambiente como este se puede ejecutar más de una aplicación a la vez, información entre ellas.



- Debido a que los clientes manejan diferentes plataformas para la operación de este, el sistema esta diseñado para trabajar en cualquiera de las que pertenecen en la industria estándar GUI como son Windows, Windows NT, OS/2 Presentation Manager.

- El soporte de múltiples Tickers verticales como horizontales es otro requerimiento del usuario para poder visualizar de la manera más cómoda en la pantalla las noticias e información nueva que vaya llegando.

- Así como el tener la facilidad de un histórico de los datos en forma gráfica y en tiempo real , la actualización y visualización de los mismos.

- Los filtros deberán de aplicarse a tickers, alertas, gráficas y ligas DDE. El sistema tiene que soportar la técnica de drag-and-drop manejada por Windows.

- Toda la información debe de presentarse en tiempo real debido a que su uso depende de la toma de decisiones criticas sobre la compra y venta de capitales muy grandes; por lo tanto el tiempo de retraso deberá ser mínimo.

Dynamic Data Exchange (DDE).

El sistema debe de contar con la funcionalidad de poder exportar los datos con otras aplicaciones como Excel, Word, y Visual Basic; para esto se utilizará DDE. El intercambio entre una aplicación y la otra deberá de ser en tiempo real así como cuando en el sistema exista un cambio en los datos deberá de reflejarse igual al momento de accesar la otra aplicación la cual tendrá que guardarse. Una vez que se tiene la liga entre el sistema y la aplicación (por ejemplo Excel) por medio de DDE se actualizarán los datos cada vez que se accese la aplicación. La copia de estos datos se harán por medio del clipboard de Windows, esto para que sea más compatible con todos el software que corre bajo esta plataforma.

Previamente para cualquier instrumento que se requiera exportar a otra aplicación debe de estar definido en los monitores del sistema de información financiero, es decir pertenecer a cualquiera de los monitores: Capitales, USA, Warrants y Páginas. La información que se tiene que exportar de acuerdo *al instrumento en uso deben* de ser el precio, la hora, precio y hora, posturas ó todas las opciones.

Cuando se desee exportar una página debe de aparecer una lista de las páginas disponibles que se encuentren abiertas dentro del sistema, así como otra lista de las páginas que se han exportado.

Al exportar instrumentos del mercado de Capitales , Warrants, acciones USA e índices del mercado; al seleccionar *la opción de exportar todo dentro* del cuadro de diálogo de exportar, las columnas que se exportarán por default y el orden que deben llevar es el siguiente:

- Instrumento/Indice
- Hora hecho
- Volumen
- Precio / Indice máximo del día
- Precio / Indice mínimo del día
- Precio / Indice Actual
- Precio / Indice Anterior
- Variación en porcentaje
- Variación unitaria
- Fecha

Si un instrumento que ya se encuentre configurado es dado de baja de los monitores, y la liga con la aplicación no se borra; los datos aparecerán con N/A.

Exportar Información.

De acuerdo a las necesidades del cliente para poder manipular la información dentro de otra aplicación , toda la información contenida dentro del sistema puede ser exportada a un editor de texto u hoja de cálculo según la necesidad del usuario. Dentro de la información que se debe de exportar aparte de los registros y las páginas, es toda la información contenida dentro de la BD Noticiosa y la BD Intradía Bursátil.

La información también tendrá que poder ser exportada a un ambiente MS-DOS para su uso en un editor de texto, ó en una hoja de cálculo de Lotus 123.

Páginas y Registros.

Al hacer una petición de una página o un registro, el sistema debe de contar con la opción de poder ver la información de un registro ó varios en específico dentro de otra ventana para su monitoreo.

Manejo de Ventanas.

Las ventanas deben de tener la funcionalidad con la que cuentan las ventanas de Windows como es poder minimizarse, maximizarse, cerrarse, restaurarse y deben de tener un scroll-bar (barra de desplazamiento) horizontal y vertical; esto es para cuando los datos excedan el tamaño de la ventana, se pueda ir desplazándose por la ventana por medio de los scroll-bar.

También debe de tener la opción de acomodar las ventanas en forma de mosaico, cascada y organizar los iconos (una vez que las ventanas se hayan minimizado); es decir que cuando tengan varias ventanas abiertas a la vez se puedan acomodar en una forma más visible y cómoda.

Como parte del manejo de ventanas el usuario solicitó poder contar con una opción que permita tener disponibles las distribuciones de los monitores más utilizados en la pantalla principal, que previamente se hayan salvado con anterioridad y poderlos accesar por medio de una opción.

Gráficas en Tiempo Real.

El sistema tiene que contar con una aplicación para el análisis técnico de la información, que permita un estudio basado en gráficas del comportamiento de los diferentes instrumentos con la intención de predecir el futuro de sus precios. La información presentada en los monitores gráficos debe ser en tiempo real y los parámetros de las gráficas estarán definidos en cuanto a un rango de precios y horas; es decir desde que precio hasta que precio se va a graficar; de igual forma para las horas que se van a graficar. Las acciones que se pueden graficar son las que pertenezcan a los monitores de registro como son: Capitales, Indices, USA, Warrants y BMV Tasas Promedio.

Ticker de Alarma.

Dentro de esta opción se debe de diferenciar por medio de un símbolo y un color el arribo de una alarma noticiosa , como el de una alarma bursátil. La definición de un ticker de alarma debe de contener los siguientes instrumentos: Monitor Mercado de Capitales BMV, Hechos de Mercado de Dinero, BMV Tasas promedio (1 día), Monitor personal de Indices, Monitor Personal USA, Ticker Noticioso, Monitor Mercado de Warrants. El formato de las columnas que debe de llevar cada monitor de ticker de alarmas debe de estar definido por una lista de las columnas disponibles para el monitor seleccionado. Esto es con la finalidad de que se pueda personalizar la salida del ticker de alarma. EL tipo de formato a desplegar para las columnas que contengan datos alfanuméricos será un string, para las horas será hh:mm (horas, minutos) y para los campos numéricos se pueden desplegar con el formato siguiente:

#,###
#,###.0
#,###.00
#,###.000
#,###.0000
#,###.00000
#,### ##/##
#.### (millones)
#,### (miles)

Las columnas se deben de poder borrar así como modificarse.

Calculadora Bursátil.

Dentro de esta aplicación se debe de contar con las siguientes opciones:

- Un Directorio en el cual se podrán dar de alta las casas de bolsa., la comisión que se cobra, los datos del promotor y un campo para comentarios importantes.
- Selección. Que servirá para agregar un nuevo inventario de acciones, eliminar y activar uno que ya exista.
- Una lista de movimientos para agregar, eliminar ó modificar los movimientos de compra de instrumentos . Los datos que deben de seleccionarse dentro de esta pantalla son: casas de bolsa, Tipo de Valor, Valor, Fecha, Precio y Volumen, Importe Total.
- Generación de un reporte en donde se indique el valor de la selección especificada en el momento de la consulta; así como otro reporte donde se compare el valor de la adquisición de los valores contra el precio, en el mismo instante en que se genera el reporte con el fin de obtener el rendimiento o pérdida neta en el tiempo transcurrido.

Ayuda .

En el menú de ayuda se debe de proporcionar información acerca de:

- El contenido informativo del servicio de información bancaria
- Los pasos a seguir para ejecutar alguna acción
- Un glosario de términos financieros

La localización de un tema en específico, regresar al último tema consultado, mostrar el siguiente tema de una serie de temas relacionados así como el anterior, y un historial de los nombres de los últimos 40 temas consultados.

La información de la BD Noticiosa se debe de poder almacenar en el disco duro dentro de un lapso de tiempo, evitando así la saturación del sistema. Los días de almacenamiento podrán ser configurados y estarán comprendidos entre 1 y 365 días.

EL sistema tiene que ser capaz de distribuir la información por una LAN a la que se puedan conectar hasta 100 usuarios a la vez. Así como se podrán conectarse varias redes de área local por medio de gateways para la distribución de los datos en diferentes plataformas.

2.5 Situación Actual y Futura.

2.5.1 Situación Actual.

La distribución de información financiera actualmente se hace utilizando una línea directa ó vía radio y en cualquiera de los medios de distribución se usa un decodificador que se conecta a uno de los puertos de la computadora. Se tienen dos aplicaciones de software que permiten obtener la información y extraerla para su uso posterior en donde será procesada en el formato correcto por medio del PT (**Traductor de Protocolo**) para ser enviada a la terminal que hizo la petición en la red local.
Actualmente cada usuario que desee tener acceso a la información deberá tener un decodificador (receptor de información) conectado a la antena. Una vez que la información ya tiene el formato correspondiente su salida es por medio de un puerto serie y deberá ser organizada y traducida al formato estándar para distribuirla dentro de una red local. Ver figura 2.5.1

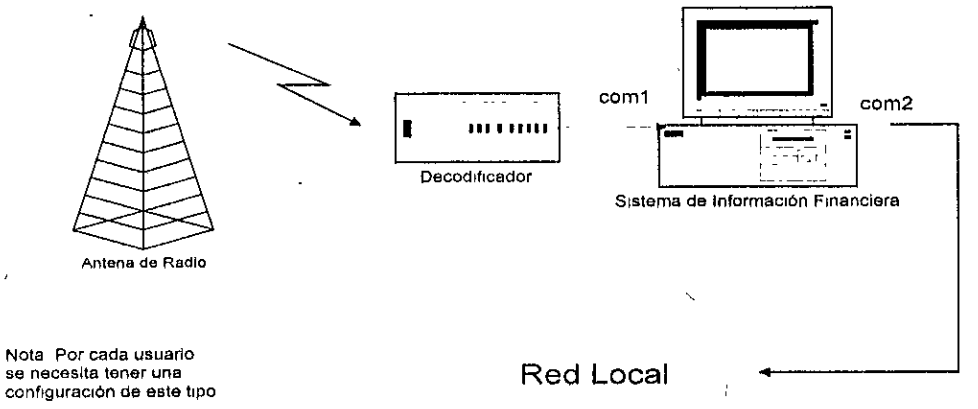


Figura - 274 - 2.5.1. Situación Actual.

El receptor de información (decodificador) es el dispositivo por el cual se comunica la computadora con la fuente de información a través del sistema por medio de una antena como se muestra en la figura anterior.

Entre las partes que lo componen tiene una señal de potencia que refleja el nivel de recepción de información, un indicador en línea muestra que ha encontrado la frecuencia por la cual esta recibiendo la señal, dos indicadores de los puertos Com1 y Com2 que se encienden cada vez que se recibe un bloque de información y por último los indicadores de error que sirven para indicar que no existe una recepción constante. Ver figura 2.5.2

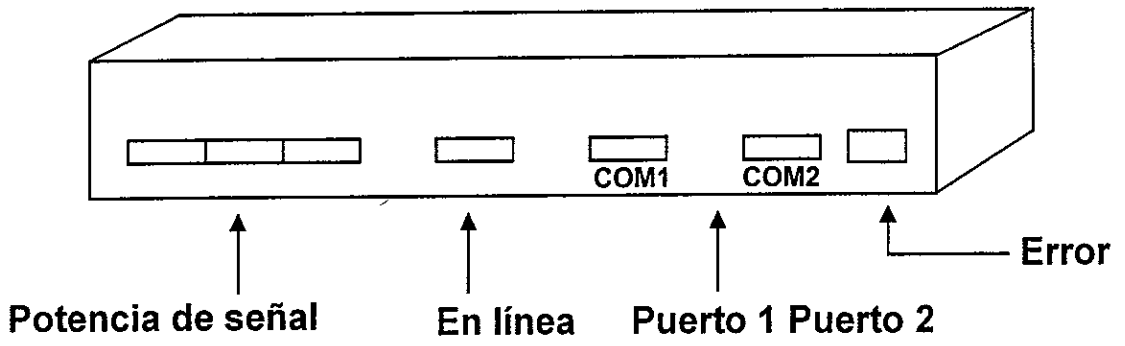


Figura 2.5.2. Receptor de Información – Decodificador.

2.5.1.1. Información en Tiempo Real

Se cuenta con una aplicación que nos provee la información en tiempo real en la cual es posible consultar la información que se esta recibiendo en el sistema a través de monitores y alarmas definidas especialmente para ello.

Monitores.

Toda la información que es recibida en tiempo real puede ser consultada de una forma ágil y selectiva a través de monitores. Sí definimos monitorear un instrumento como podría ser una acción, la información se va a ir actualizando conforme se obtenga la información de la fuente de información y se puede tener un monitoreo de está. Actualmente se cuenta con varios tipos de monitores: monitores de registro, monitores gráficos, de páginas, de eventos y tickers de alarmas.

Monitores de Registro. Se caracterizan principalmente por que cada registro de información es único y sobre esté se actualiza la información según va llegando.
La información que se puede manejar dentro de estos monitores de registro tenemos las siguientes : Capitales BMV (Bolsa Mexicana de Valores), BMV Tasas promedio (1 día), Indices, USA, Más cambios, Warrants y Ticker personal; para todos estos monitores se puede recibir información en tiempo real y se puede activar una alarma para indicarnos cuando un nuevo movimiento ha cambiado dentro de los monitores sin necesidad de cerrarlos.

Una alarma nos puede mostrar como van cambiando las tasas, precios, alza y baja de acciones en el mercado mundial, posturas, monitor de pisos ó volumen, etc.

Monitores Gráficos. Presentan información en tiempo real a través de una gráfica de líneas, los movimientos del precio para la acción definida. Las acciones que se pueden graficar son las que pertenecen a los monitores de registro, como los capitales, índices, Warrants, y BMV Tasa promedio. La configuración dentro de estos monitores gráficos puede ser de manera automática según varían los movimientos ó se puede definir un rango de tiempo y precios según sea el monitor de registro que este activado.

Monitores de Página. El uso de los monitores de página nos presentan la información de una manera agrupada por tipo de información que se maneje, en una forma accesible y en tiempo real. Pueden presentar la última cotización que se haya recibido es decir, no se manejan datos históricos. La información que manejan es de tipo financiero como pueden ser información referente al Mercado de Dinero Internacional, dólar interbancario, Futura USA, Mercado de Dinero Mexicano, e Indicadores Económicos Mexicanos.

Monitores de Eventos. Con este tipo de monitores el usuario puede definir el tipo o rango de información que desea recibir y al llegar la nueva información será agregada al monitor como un nuevo registro.

Monitor de Opciones. A través de este monitor es posible dar seguimiento en tiempo real a los listados en bolsas norteamericanas, ADR´s mexicanos, así como también es posible dar seguimiento a las transacciones en el piso de remates de instrumentos del mercado de dinero en tiempo real.

Ticker de Alarmas. El ticker de alarmas filtra la información Noticiosa y Bursátil a través de alarmas, desde este monitor se puede obtener el texto de la nota; si se trata de una alarma noticiosa ó del detalle de la acción, ó instrumento si se trata de una alarma bursátil. La información noticiosa es un texto en donde se indica el cambio de un índice, una tasa, un precio de una acción, etc.

Base de Datos Noticiosa.

El sistema provee una aplicación que presenta las noticias de los mercados financieros del mundo, empresas, rumores, comentarios y análisis de las operaciones financieras dentro de los mercados de dinero cambiario y bursátil, así como del entorno económico general.. La información dentro de la base de datos noticiosa esta clasificada por categorías como se muestra en la figura 2.5.3. Información BD Noticiosa.

Base de Datos Intradía Bursátil.

La base de datos intradía bursátil es otra aplicación que viene con el sistema que sirve para la presentación de reportes y consultas que permiten obtener el desglose de operaciones generadas en el piso de remates de la B. M. V. durante la jornada. El sistema permite configurar el almacenamiento de información bursátil de los valores de interés propios del usuario evitando de esta forma la acumulación de información en la base de datos. Con esta aplicación se pueden almacenar todos los movimientos referentes a un mercado de capitales ó solo los valores que se hayan seleccionado; así como ningún movimiento referente al mercado de capitales. Al igual que en la aplicación de la BD Noticiosa la información se divide en categorías las cuales se clasifican con la siguiente información de la figura 2.5.4. Información de la BD Intradía Bursátil.

Categoría	Clasificación
Capitales	Perspectivas, Reportes financieros, Razones Financieras, Información Oficial, Derechos, Múltiplos, B.M.V. Internacional, Fondos de Inversión, Noticias, Rumor.
Dinero	Subastas Banco de México, Mesas de Dinero Fondeo, Mesas de Dinero Plazo, Tasas Bancarias, Información Oficial, Colocaciones Primarias, Análisis, Comentarios B.M.V., Noticias, Rumor
Cambios	Dólar Interbancario, Dólar Menudeo, Coberturas, Otras divisas, Noticias, Metales Nacionales.
Noticias Generales	Prensa Nacional, Flash Noticioso, México en el mundo, Agenda Financiera, Prensa Internacional, América, Europa, Análisis semanal, Resto del mundo.
Internacional	Bolsas del mundo, Mercado de Dinero, Mercado de Divisas, Metales, Futuros, Petróleo.
Mercancías	Nacionales, Internacionales.

Tabla 2.5.3. Información BD Noticiosa.

Calculadora Bursátil.

Como una herramienta útil el sistema nos provee de una aplicación en la que se permita definir los inventarios, las acciones, con la finalidad de evaluar los rendimientos y la posición financiera desde que se hace la compra de una acción dentro de una casa de bolsa hasta el momento de realizar la consulta, para posteriormente generar un reporte.

Categoría	Clasificación
Capitales	Valores a la alza, Valores a la baja, Movimientos de Valores, Operaciones de Casas de Bolsa, Concentrado de Operaciones, Participación en el Mercado, Valores más Negociados, Boletín Bursátil, Evolución del Indice IPC, Indice IPC Histórico.
Dinero	Movimientos de valores, Evolución de Tasas, Evolución de Tasas Promedio 1 día, Resumen de Operaciones, Boletín Bursátil.

Figura 2.5.4. Información BD Intradía Bursátil.

Por otra parte se puede tener una consulta rápida de una acción ó instrumento que permite el detalle de la operación, así como consultar a los proveedores que están enviando la información. Además se tiene una herramienta que nos facilita el uso del manejo de ventanas cuando se tienen varias ventanas abiertas; ya que por cada aplicación se tiene una ventana y por cada monitor. La ayuda en línea es otra herramienta dentro del sistema y puede ser accesada desde cualquier cuadro de diálogo del sistema.

Toda la información que ha sido procesada dentro del sistema es almacenada en un registro que lleva el traductor de protocolo en tiempo real. El paso siguiente es traducir toda esa información en un formato gráfico y fácil de usar por medio de otro software especializado para después enviarla al cliente que hizo la petición. Este software es otra aplicación que sirve para sustraer la información que se recibe de todo el sistema de información financiera y procesarla para obtener un filtro y enviarla a través del puerto serie de la máquina hacia la red local. EL puerto serial se configura también a través de esta aplicación.

2.5.2. Situación Futura.

El sistema estará corriendo bajo la arquitectura Cliente-Servidor y bastará con solo una conexión al decodificador, una fuente de información en donde residirá el sistema de información financiera que se conectará por el puerto hacia otra computadora la cual fungirá como servidor (Traductor de Protocolo), para mandar la información a las terminales conectadas a una red LAN (**Local Area Network**) así con este tipo de conexión se enviará la información a todos los usuarios conectados a la red para que tengan acceso al sistema sin estar conectados directamente al decodificador y a la fuente de información financiera como se hacia anteriormente. Esto con el fin de disminuir los costos de instalación y mantenimiento para la compañía que ofrece el servicio y para el cliente. Ver figura 2.5.5 Situación Futura.

Toda la información estará centralizada en el servidor PT el cual mantendrá la información en memoria para que los tiempos de retraso entre la recepción y el envío al usuario sea mínimo, esto con la finalidad de asegurar la confiabilidad de la información ya que toda la información que se maneja es de tipo financiero y la toma de decisiones sobre la compra o venta de una acción ó instrumento de capital debe de ser en el momento exacto es por esto que el tiempo de respuesta debe ser mínimo. Así mismo toda la información en tiempo real debe de mantenerse en memoria para que el acceso sea más rápido y no tener que saturar los discos duros locales ó remotos y las bases de datos.

El servidor se comunicará con las terminales a través del protocolo TCP/IP conectadas en red, también podrá atender hasta 100 usuarios simultáneamente con un solo servidor siendo esta una característica de las Redes de Area Local (LAN).

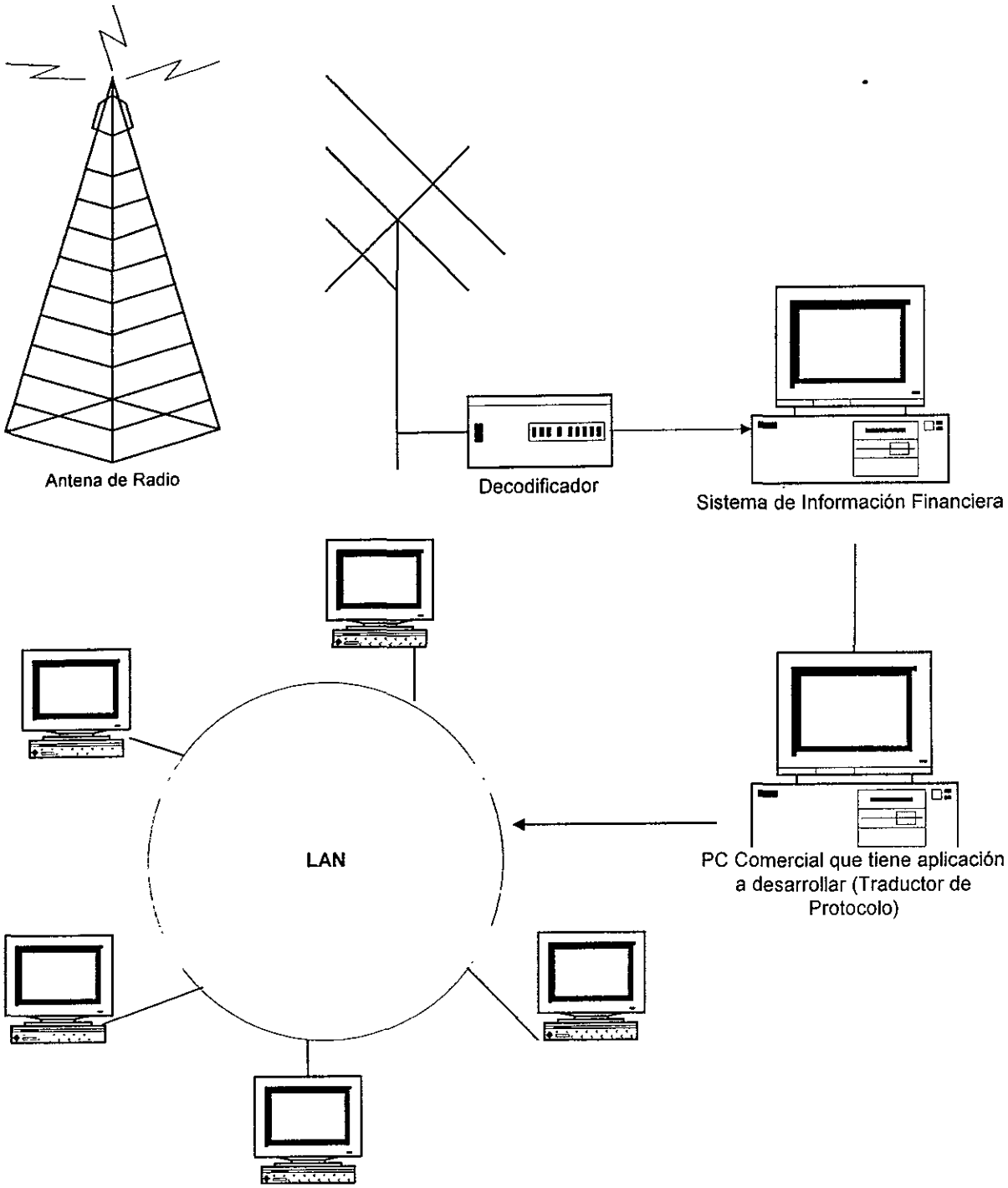


Figura 2.5.5. Situación Futura.

Arquitectura.

El sistema estará diseñado para poder atender los requerimientos de usuarios ya sea hacia una red de área local con pocos usuarios, así como conectarse y enviar la información a otra red externa más grande. Se basará en una red LAN con un servicio de Gateways (**Compuertas**) y Workstations (**Terminales**).

Gateway. El servicio de gateways manejará todas los conexiones con las fuentes de información y distribuirá los datos desde estás hacia toda la red. Las workstations proporcionarán al usuario la interface para desplegar y procesar la información requerida. Se pueden conectar varias redes de tipo LAN por medio de un bridge (**puente**) y a su vez conectadas a un servicio de gateway como se muestra en la figura 2.5.6

Como se mencionó anteriormente el sistema se basará en la arquitectura cliente-servidor usando procesamiento distribuido siendo así un sistema modular y flexible, capaz de tener muchas extensiones físicas con diferentes configuraciones. Cada instalación tendrá que adaptarse a los requerimientos y recursos específicos de la organización cliente. Será totalmente confiable y tolerante a fallas, esto es que cualquier falla que ocurra en algún componente de el sistema no tendrá efecto sobre los demás componentes. La pérdida de un servicio no afectará el acceso del usuario a los demás servicios disponibles, de la misma forma la falla de una workstation solo afectará al usuario de está y se podrá accesar al sistema por medio de otra terminal y a todos los recursos. La recuperación de datos perdidos estará comprendida dentro de un intervalo de 5 segundos después de la perdida de la información, la información recuperada se desplegará de inmediato en la workstation. Si el servicio de gateway no puede responder a alguna petición esto se deberá seguramente a que todos los recursos están agotados ó porque los límites habrán sido excedidos por los usuarios, en este caso un mensaje será enviado a el usuario y la petición se enviará en cuestión de segundos lo más pronto posible.

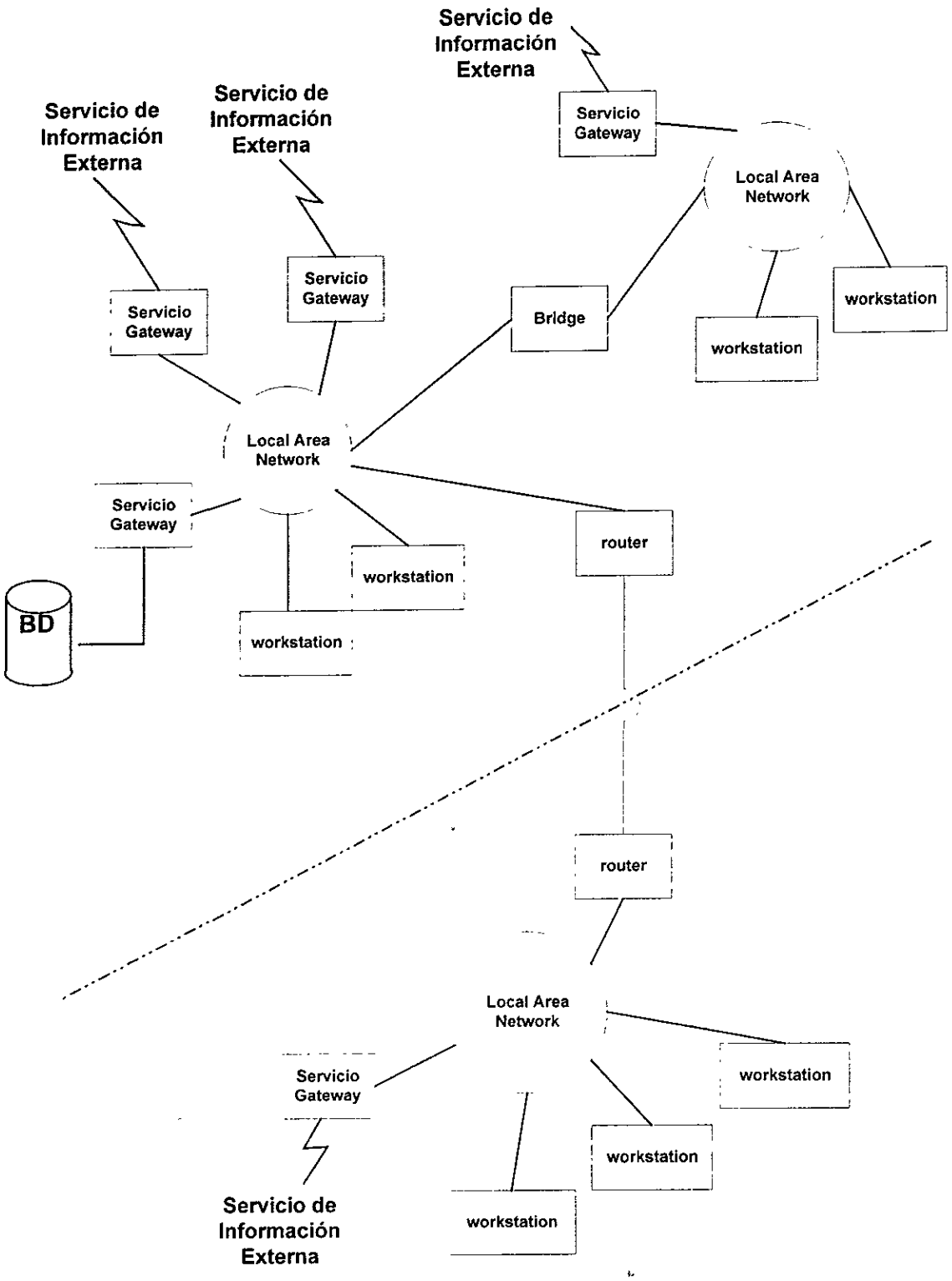


Figura 2.5.2.3 Conexiones de Red con Gateway

Cuando el usuario llegue a exceder el máximo de requerimientos del sistema los datos no se perderán ya que el servidor estará configurado de tal forma que la información se mantendrá siempre en un buffer de memoria, hasta que puedan ser procesados y se liberen los recursos suficientes en la workstation; por esta razón los datos nunca se perderán ya que se mantienen en memoria.

El servicio de las gateways se encargará del procesamiento de los datos cuando lleguen desde el sistema de información financiera, así como la distribución y actualización en la red. Podrá tener varias conexiones externas del servidor de información soportada por el sistema, también se encargará de controlar los datos y distribuirlos.

Cada gateway será implementada como un servidor de aplicación que se compone de dos principales componentes:

- PT (Traductor de Protocolo)
- ODS (Object Distribution System)

El traductor de protocolo tomará el papel de back-end de la aplicación manteniendo la comunicación con el servicio de información con el que el gateway estará conectado.
El back-end son todos los módulos que residen en el servidor (Traductor de Protocolo) y estará orientado a recibir solicitudes del cliente con capacidad de procesamiento y de regresar la información solicitada a esté. El PT administrará la distribución de los datos en la red para lograr un procesamiento más rápido y eficiente.

ODS. El ODS será el front-end de la aplicación cuya función es la de controlar la distribución de los datos desde el servicio del gateway por toda la red. Este es un componente del gateway que sirve para aumentar la capacidad del cliente en la red, es decir el sistema podrá atender hasta 100 usuarios conectados con un solo servidor, si se necesitará atender a más usuarios conectados a otra red se tendrá que instalar un sistema de distribución en cascada; en donde la última terminal de la primera red servirá para conectarse a un ODS que podrá atender a los demás usuarios de la red.

El ODS administrará el uso de los recursos de el PT para múltiples aplicaciones clientes, así como será el responsable de los servicios del gateway, como son la seguridad, la verificación, localización y control de los recursos.
Todo el sistema será un sistema abierto capaz de funcionar en diferentes sistemas operativos, en diferentes ambientes de red con diferentes configuraciones. Las aplicaciones con las que contará el sistema para accesar a la información financiera, además de las que se mencionaron en el punto 2.5.1 Situación Actual; se describen a continuación.

Dynamic Data Exchange (DDE).

El software que residirá en la terminal soportará DDE que es conocido como el intercambio de datos dinámico lo cual nos facilita el utilizar la información bajo los diferentes sistemas operativos como MS-Windows NT, MS-Windows, ó OS/2 Presentation Manager para el intercambio de datos en tiempo real. Se exportarán los datos por medio de esta aplicación DDE hacia hojas de cálculo, procesadores de texto, reportes, etc. Los datos exportados podrán mantenerse en constante cambio en tiempo real, es decir los cambios se verán reflejados en el objeto exportado por DDE.
Con DDE se pueden crear ligas entre el objeto y por ejemplo una hoja de cálculo de Excel y se irá actualizando de forma automática cada vez que se accese a esta forma en Excel.

Application Programing Interfaces (API).

Un API servirá para desarrollar una aplicación en tiempo real y hacerla disponible dentro del sistema, ya sea como una aplicación aparte o como parte del propio sistema. Se podrá desarrollar un API que administre el servicio del gateway con un nuevo servicio de información, así como también se podrá desarrollar una aplicación final para el usuario que pueda accesar los datos del sistema.
Peticiones API. EL uso de las API's en la plataforma digital en C proporcionarán un acceso a los datos desde todos los gateways dentro de la red, habilitando nuevos clientes y desarrollando sus propias aplicaciones para las terminales.

Publisher API. Servidor de API, con estas API´s se puede desarrollar un servicio de gateway para manejar información de una nueva fuente de información, también se puede desarrollar algún servicio que se requiera como un aplicación agregada. El uso de una API nos proporcionará una nueva aplicación como un servicio de gateway con acceso a los mecanismos estándares del sistema , para la distribución de los datos a través de la red, controlando el acceso de los datos. Los datos del nuevo servicio se presentarán de la misma manera como se presentaban en el sistema.

Interface con el Usuario.

La interface con el usuario podrá ser soportada en diferentes plataformas como MS-Windows NT, MS-Windows, MS-Windows Workgroups, OS/2 Presentation Manager.
Será una interface gráfica y fácil de usar, los datos podrán visualizarse desde todos los servicios simultáneamente. Toda la información presentada será en tiempo real, y las aplicaciones que el usuario podrá utilizar serán Tickers, alarmas, DDE, filtros de datos, Macros, utilidades de Excel, Ayuda en línea, más todas con las que contaba anteriormente el sistema.

2.6. Opciones de Solución.

Existen diversas opciones de solución para la distribución de información financiera y la manipulación por parte del cliente. Entre las descritas a continuación podemos encontrar que van desde un desarrollo completo con tecnología de vanguardia, pasando por otras un poco mas tradicionales que consisten de programas comerciales para la distribución de la información ya existente en el mercado en combinación con desarrollo en la parte del cliente hasta llegar a una en la cual se combinan un desarrollo en la parte del servidor para la distribución de información y para conseguir la total funcionalidad requerida por el usuario la utilización de un programa comercial ya existente en el mercado. Todas ellas incorporan diversas tipos de tecnologías en los procesos de traducción y distribución de la información teniendo como principio básico la disponibilidad en tiempo real de la misma por parte del usuario.

Debemos considerar que las características que debemos cumplir en la parte del cliente son: manipulación de los datos recibidos por el usuario (despliegue gráfico de la información) , la posibilidad de integración con otras aplicaciones de gran utilidad al usuario y la posibilidad de personalizar la aplicación a todos aquellos usuarios que desean incorporar mayor funcionalidad a su aplicación.

Primera Opción de Solución.
Propuesta de Desarrollo utilizado Tecnología Internet.

La primer propuesta consiste en hacer un desarrollo partiendo de las especificaciones comentadas en los puntos anteriores y que involucran algunas tecnologías como son: **DCOM** para el desarrollo del Servicio de Traductor de Protocolo (la programación puede realizarse en Visual Basic, Visual Fox o Visual C++), además de una **Base de Datos** para el almacenamiento de la información financiera (como Microsoft Access o SQL Server).

Para la distribución de la información financiera sobre la red de área local se pretende involucrar **Internet** (desarrollando una **Intranet**) en combinación con **Tecnología de Push** (la cual puede distribuir información respondiendo a disparos de una base de datos).

En la parte del cliente con la ayuda de un browser podremos desplegar la información en páginas a los usuarios, éstas deberán contener algunos pequeños programas ya sea desarrollados por nosotros o incluir programación de terceros (**programas con ocx**), los cuales servirán de contenedores de la información con las característica de tecnología DDE y OLE automatización.

A continuación se muestra un diagrama el cual contempla todos los elementos involucrados en el desarrollo de esta solución.

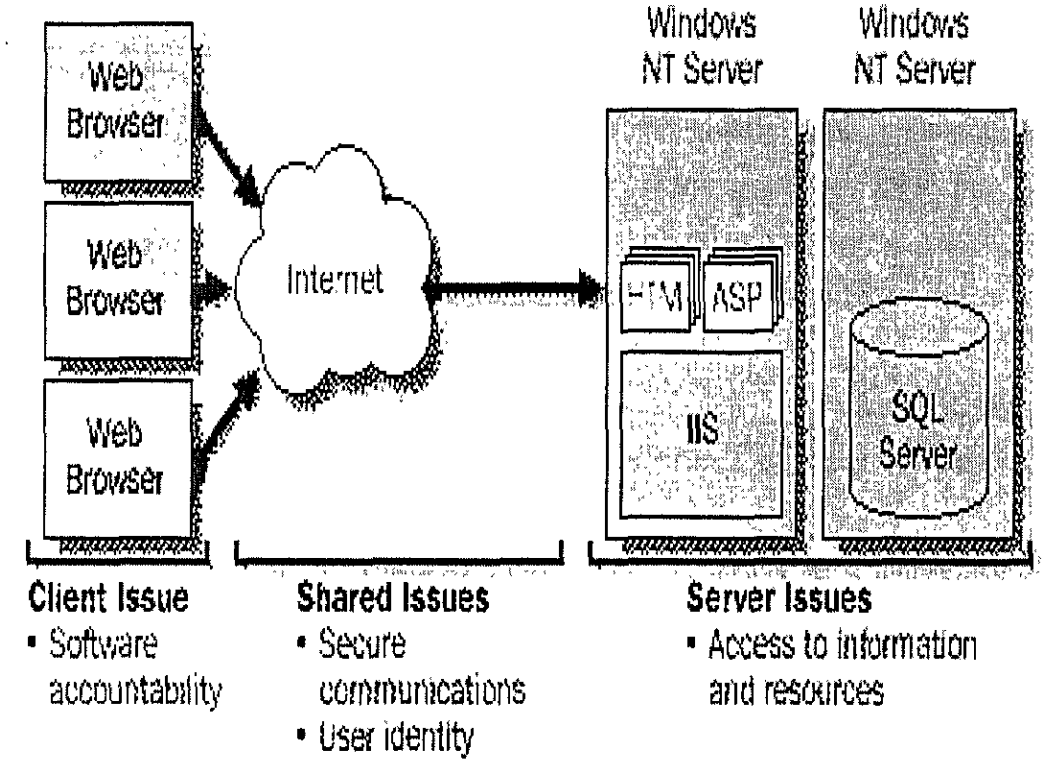


Figura 2.6.1. Opción de solución con Tecnología Intranet.

Para un mejor entendimiento de la propuesta creemos necesario hacer algunas definiciones.

DCOM. Es la base tecnológica de los producto Microsoft, podemos decir que es la evolución de la tecnologías DDE (Dinamic Data Exchange, una forma de mensajería entre programas Microsoft), OLE (Object Linking and Embeding), COM (Component Objetc Model, usado como base para la intercomunicación entre objetos) y ActiveX (COM habilitado para Internet). Esta es una arquitectura que marca los estándares para la intercomunicación entre objetos distribuidos a lo largo de una red. Entre las características que podemos encontrar tenemos: esta basado sobre la mas amplia tecnología de componentes, es la mejor tecnología de red para extender aplicaciones a lo largo de Internet y por ultimo podemos decir que es una tecnología abierta que corre sobre múltiples plataformas.

Internet. Partiremos de la definición de Internet la cual es conocida como la red de redes. Esta actualmente conecta miles de redes permitiendo a los usuarios compartir en forma global información y recursos de computadoras. Con Internet, los usuarios pueden compartir cualquier cosa que pueda ser almacenada en un archivo. La comunicación Internet es posible entre redes sobre diferentes plataformas y dentro de diferentes ambientes. Este dinámico intercambio de datos es debido al desarrollo de protocolos de comunicación particulares para esto. Ya hemos discutido el papel que juegan los protocolos en el intercambio de información y en la habilidad de comunicar a los usuarios entre diferentes redes.

Intranet. Una Intranet es una red privada de computadoras que utilizan estándares de Internet y protocolos para habilitar a los miembros de una organización comunicarse y colaborar mas eficientemente unos con otros, obteniendo como consecuencia una mayor productividad.

Algo de tomarse en cuenta es que no hay que deshacerse de toda la infraestructura de una red de área local o de una red de área amplia para tener una intranet.

A diferencia de la redes tradicionales, una intranet cuenta con una tecnología especial abierta y estándar.

Browser. Browser es un programa del lado del cliente Internet o Intranet que comunica los requerimientos del cliente hacia el servidor y despliega la información recibida al usuario. Un browser es la interfaz entre el usuario y el contenido almacenado sobre la red. Este es el universal front end para maquinas clientes y muchos tipos de software servidor.

Software Especial. Una variedad de controles, scripts, plug-in y otros pequeños programas o applets caen bajo la cabecera de software especial. Algunos programas especiales se encuentran disponibles sin costo alguno en el mundo Internet, mientras otros pueden ser obtenidos a un costo muy bajo. Por otra parte, existe software especial que podrá ser programados con relativa facilidad a través de tecnología de componentes. (Como sería nuestro caso).

Páginas Estáticas y Dinámicas. En los principios de la programación HTML , todas las páginas Web fueron estáticas. Cuando se teclea una dirección URL (Uniform Resource Locator) o escoge una liga (la cual apunta a una URL), el browser envía la petición (a través del motherboard, la tarjeta de red, o el módem) hacia el servidor, el cuál entonces sirve una página de retorno hacia el browser sobre su computadora. El contenido de los datos de la página es pre-especificado, esto es conocido con el concepto de código duro, estos es, es escrita dentro de archivos texto, lo cual puede ser apropiado para información que no cambia frecuentemente.

Las páginas dinámicas mantienen la estructura de la información sobre si mismas, los datos están almacenados en una Base de Datos o un archivo de fácil administración. Las páginas pueden ser creadas con CGI (common gateway interface), Scripts Java, servidores de páginas activas (ASP Active Page Server) o controles ActiveX. En otras palabras las páginas actúan como un cascarón de los datos, que son obtenidos de una Base de Datos puestos dentro y devueltos al cliente.

Cuando una página Web ofrece una forma que se puede usar para hacer una petición específica, o provee de una liga para datos especializados, instrucciones especializadas que son codificadas dentro de la petición y esta es enviada hacia el servidor. El servidor interpreta la instrucción especial y procesa la petición, usualmente con la ayuda de una rutina script que permita accesar los datos y selecciona los que cumplan con la petición. Por último el servidor retorna los datos a la medida del cliente sobre su browser.

Tecnología Push. Es una tecnología de Internet la cual habilita a los dueños de sitios web (por ejemplo una intranet) sus propios canales (Internet o Intranet) para la entrega de paquetes de información en forma personalizada directamente a usuarios finales. Este consta de un producto en el servidor (el cual provee la entrega de paquetes), software cliente el cual establecerá la necesidad de información en algún canal, y algunas utilerías para la administración del servicio. La información que puede ser entregada puede ser Screen savers, wallpapers, mensajes de audio, descargo de páginas Web/HTML, archivos de datos, Java applets, ligas a otros sitios en Internet.

Evaluación de la Solución.

Existen alguna ventajas de esta solución:

- Los usuarios no necesariamente podrían usar esta solución desde su escritorio, pues con los ajustes pertinentes, sobre todo en la parte de infraestructura y seguridad dentro de la intranet, podrían estar accesando a esta información desde cualquier parte del mundo. Para lograrlo es necesario que el usuario cuente con una laptop con el software desarrollado para la manipulación de la información financiera.
- Es una solución que implica un reto interesante para la empresa y para el despacho que se encargara de su desarrollo, pues se esta involucrando toda la tecnología de punta que predominará en el Modelo Cliente/Servidor por bastante tiempo.

Sin embargo también existen desventajas:

- Implica un mayor tiempo de desarrollo e implementación pues los elementos necesarios que soportan la infraestructura no se encuentran la mayoría disponibles.
- Implica una mayor inversión económica pues tenemos que adicionar piezas de software y hardware para otorgar la funcionalidad deseada. En la parte de Hardware la inversión posiblemente sea mayor debido a que se tiene que garantizar disponibilidad de información en tiempo real lo que implica un mayor poder de computo.
- Entre más elementos estén involucrados en la solución, mayor es el riesgo que se corre de no poder brindar la funcionalidad antes planteada.
- Existe dependencia tecnológica hacia Microsoft.
- Se requiere establecer un plan de mantenimiento preventivo y correctivo al software desarrollado, así como mantenimiento perfectivo en el caso de puntos de mejora propuestas por los usuarios.

Segunda Opción de Solución.
Utilización de Programa Comercial SIVA 25.

El "Sistema Integral de Valores Automatizado X.25" (SIVA 25) -Sistema de Transmisión de datos en Tiempo Real- se emplea en la **Bolsa Mexicana de Valores S.A. de C.V.** para la diseminación de la información general diaria que se utiliza en la comunidad Bursátil y Financiera, para obtener, oportunamente, la información que en esta institución se genera, ya sea para llevar a cabo las operaciones rutinarias del medio, efectuar análisis y estudios del comportamiento del Mercado de Valores, o difundir la información a través de los medios masivos de comunicación.

Beneficios propuestos por el vendedor.

Este **Sistema de Transmisión de Datos en Tiempo Real**, pone a disposición de los usuarios de la BMV, la información que se genera en ésta en el menor tiempo posible, de forma que se puedan tomar acciones y decisiones a la velocidad que se requiere en la actualidad, buscando utilizar, en la medida de lo posible, los adelantos tecnológicos disponibles tanto en los equipos de cómputo como de comunicaciones que al mismo tiempo sean accesibles para todos los usuarios, sin importar la plataforma de cómputo instalada. El sistema es autorregulable, de forma que detecta fallas o interrupciones en la transmisión de datos y recupera las mismas de forma automática sin necesidad de la intervención humana.

"El usuario receptor de la información, debe desarrollar programas de cómputo que interactúen dinámicamente con la computadora de BMV, controlando el flujo de los registros que le son enviados, validando la información que le llega y solicitando la retransmisión de los datos en caso de alguna falla, de forma automática. Sin el desarrollo de estos programas el ***SIVA25*** *no podrá funcionar como fue planeado."*

Protocolo de Comunicaciones X.25.

El protocolo de comunicaciones **X.25** fue escogido debido a que se rige por estándares que son aceptados mundialmente, ya que se han revisado y mejorado de manera continua a lo largo de los años con el fin de hacerlo más seguro y confiable, esto ha permitido el que se facilite su implementación ya que prácticamente todos los proveedores de equipos de cómputo cuentan con el software y hardware necesario para su instalación y uso. Otra razón para seleccionar este protocolo es la estabilidad que ha mostrado a lo largo de los años en que se ha usado así como su capacidad para detectar y recuperar errores, propios de los sistemas de conducción de señales, de forma automática y que forman parte del mismo código de comunicaciones.

Diagrama del SIVA25.

El diagrama siguiente muestra los elementos que forman parte del **SIVA25** y la interrelación que existe con el **SIVA**.

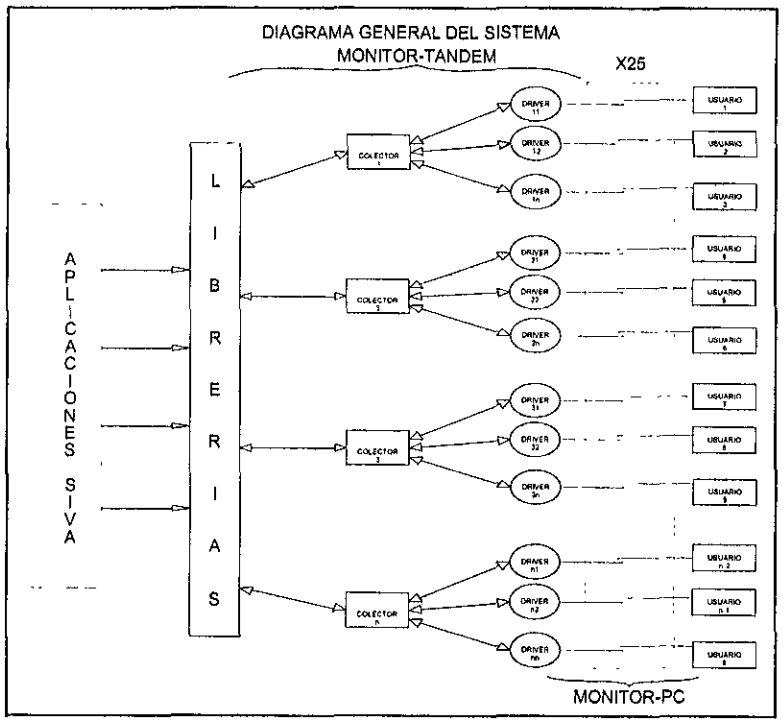


Figura 2.6.2. Esquema de Operación de SIVA 25.

El Colector.

Los programas de los diferentes subsistemas del **SIVA** que actualizan a las bases de datos, al mismo tiempo que están grabando la información, activan las rutinas del **SIVA25** que validan, formatean y envían los registros a cada uno de los usuarios del sistema. Así tenemos que, inmediatamente después de que un registro ha sido aceptado por la base de datos, el programa encargado de grabar estos datos, los envía a un programa llamado **Colector**, el cual se encarga de recibir el registro, tipificarlo de acuerdo a la información que contiene y darle un formato estándar para que pueda ser transmitido.

Driver en BMV.

Después de que el registro ha sido grabado en la base de datos, se envía al programa que transmite la información al usuario (Driver BMV), para que, a su vez, sea enviado al programa que recibe la información (Driver Usuario) que reside en el equipo de cómputo del usuario. Con el fin de poder dar un mejor servicio a cada uno de los usuarios y adaptar el ritmo de la transmisión a las necesidades de cada uno de ellos, se activan tantos Drivers en BMV como usuarios hay definidos en el sistema, de forma que puedan ser atendidos de forma dedicada y exclusiva.

Todos los Drivers de BMV son, sin embargo, iguales en su funcionamiento y sólo varían en cuanto al ritmo en que están efectuando la transmisión al usuario al que se encuentran asignados. Cada Driver de BMV mantiene actualizado un registro de usuarios donde se especifican las características de ese usuario y el último número de secuencia que fue enviado correctamente. Este registro sirve para verificar que los datos se estén enviando en la secuencia correcta y para asegurar que no haya faltantes en la información. El mismo Driver de BMV se encarga de filtrar la información que se envía de forma confidencial a los diferentes usuarios del sistema así como los organismos reguladores del gobierno de forma que sólo sean los directamente interesados quienes reciban los registros que van dirigidos a ellos.

Es el Driver de BMV el que se encarga de manejar las solicitudes de retransmisión solicitadas por los usuarios o las que genera el Colector de forma automática cuando se detecta algún problema en el flujo normal de los datos.

Tanto el Driver de BMV como el Colector son programas únicos que pueden por sí solos controlar el flujo de datos hacia los usuarios, sin embargo, para evitar problemas de contención o la saturación de los programas, se optó por tener trabajando, concurrentemente, varias copias del Colector.

Cada uno de estos Colectores puede alimentar de información hasta diez copias del Driver de BMV el cual atiende, de forma exclusiva, a un sólo usuario del sistema. De esta forma se minimiza el impacto que pueda tener el que un Driver de BMV se deje de ejecutar por cualquier problema, ya sea en BMV o en el equipo del usuario.

Monitores.

El **SIVA25** cuenta con dos programas para el monitoreo y control automático del sistema. Uno de ellos reside en la computadora Tandem y su función es la de detectar y corregir las caídas del Colector y los Drivers de BMV de forma que cuando se presente una de estas, el programa en cuestión sea reactivado de inmediato. El otro monitor reside en una PC y su función es la de detectar cualquier interrupción en el flujo de datos que se pueda dar en la red de teleproceso. En este caso, el monitor no ejecuta ninguna acción correctiva, limitándose a informar, con despliegues visuales y auditivos, de la falla que ha detectado.

Adicionalmente a estos dos programas, existe todo un sistema de vigilancia y control de la red digital de comunicaciones, independiente de la computadora central de la BMV el cual fue instalado para monitorear el estatus de la red de comunicaciones así como su rendimiento de forma que se pueden hacer ajustes en los parámetros de configuración de las líneas para mantenerlas trabajando con la máxima eficiencia y calidad en la transmisión de datos.

Envío de la Información.

El sistema envía todo lo que se genera en la BMV sin distinción alguna. Como se apuntó anteriormente, es el Driver de BMV el que se encarga de seleccionar la información confidencial para cada uno de los usuarios. En el caso de la información confidencial, se envía el registro con datos al interesado y, para evitar que haya saltos en el número de secuencia, se envía el Header de control y el resto del registro en blanco al resto de los usuarios. De esta forma se asegura la confidencialidad de la información sin perder la posibilidad de detectar fallas en la transmisión por el simple hecho de contar con un número de secuencia consecutivo.

El **SIVA25** efectúa las funciones de enlace y transmisión de datos de forma automática, en el momento que detecta que un usuario está listo para recibir la información. En el momento en que el Driver de BMV detecta que el controlador de comunicaciones del usuario está listo para recibir los datos:

- Verifica cual fue el último número de secuencia que envió correctamente.
- Se comunica con el Colector para solicitar el siguiente número de secuencia.
- Si el número de secuencia solicitado es el que tiene listo para enviarse, lo transfiere al Driver de BMV y éste le da salida.
- Si el número de secuencia solicitado es mayor al último enviado, descarta la solicitud y se queda en espera de una nueva solicitud por parte del Driver de BMV.
- Si el número de secuencia solicitado es menor al último enviado, el Colector accesa su base datos en busca del registro solicitado e inicia una retransmisión a partir de ese punto.
- El driver transmisor recibe el registro solicitado y lo envía al usuario.
- Dependiendo del equipo de cómputo, el controlador de comunicaciones del usuario enviará una confirmación de recepción de los registros por cada uno de los registros recibidos o por cada grupo de 7 registros (**X.25** envía bloques de 7 registros en cada transmisión).

- Si el Driver de BMV recibe estas confirmaciones, continúa con el proceso normal de envío.

El que el envío de información se inicie al momento en que los controladores de comunicaciones se encuentren listos para la transmisión de datos, implica que, en algunos equipos, el flujo de información se inicie antes de que los programas aplicativos que van a manejar la información se encuentren listos para leer los registros.

En estos casos, la primera instrucción del programa aplicativo será la de pedir una retransmisión a partir del último registro recibido para asegurar que toda la información generada llegue completa a las aplicaciones de los usuarios.

Evaluación de la Solución.

Las ventajas de esta solución son:

- Los usuarios tienen un menor tiempo de implementación y operación de acuerdo al plan de instalación y pruebas del producto.
- Es un programa comercial con experiencia en el mercado.

Las desventajas son:

- La solución es parcial lo que implica un tiempo de desarrollo en la parte de manipulación de la información financiera del lado del cliente.
- Se requiere la inversión en el hardware que permita el manejo de X.25 como protocolo de transporte
- El sistema SIVA tiene un costo elevado por terminal.
- Se requiere establecer un plan de mantenimiento preventivo y correctivo al software desarrollado, así como mantenimiento perfectivo en el caso de puntos de mejora propuestas por los usuarios.

Tercera Opción de Solución.

Combinación de Software Comercial para la administración de peticiones sobre Plataforma Digital con desarrollo de proceso de traducción y distribución de Información.

¿Que es el Software Comercial para la administración de la Plataforma Digital?

Este es un software orientado a objetos para un ambiente de escritorio el cual permite la recuperación, despliegue y análisis de información financiera y noticias. Incorpora todas las herramientas estándares de la presente generación de estaciones de trabajo financieras (por ejemplo páginas, cotizaciones, cadenas de despliegue, lista de cotizaciones, teletipo y alertas). En suma ofrece un conjunto de herramientas avanzadas, funcionalidad OLE, facilidades de desarrollo, soporte para desarrollos y fácil integración de programas adicionales y desarrollos de terceros. Juntas todas estas características proveen unas herramientas de escritorio fácil de usar, versátil y abierta a conocer los actuales y futuros requerimientos de información.

El software corre bajo Microsoft Windows NT con la interfaz estándar de Microsoft. Si el *usuario esta familiarizado con otras aplicaciones* Windows podrá encontrar facilidad en su aprendizaje y uso. Para los usuarios que no están familiarizados con la ultima interfaz de Windows, ambos sistema operativo y el software fueron diseñados para ser de fácil aprendizaje y uso, al tiempo que proveen una sofisticada funcionalidad.

Frames y Espacios de Trabajo.

Los frames son los elementos básicos de despliegue para datos sobre la plataforma digital. Cada objeto que es requerido o creado (por ejemplo una página, cotización, gráfica o lista de cotizaciones) es desplegado dentro de un frame es normalmente coloreado su borde y la barra de títulos. Estos frames son desplegados en áreas de espacios de trabajo

Cooperación con Otras Aplicaciones.

La software para el control de peticiones sobre la plataforma digital tiene una completa funcionalidad OLE, habilitando la posibilidad de interactuar y compartir información con otras aplicaciones. OLE provee principalmente a la plataforma digital la posibilidad de integración y administración con una amplia variedad de aplicaciones comerciales y de oficina con la posibilidad de utilizarlas sin la necesidad de abandonarla. Cuando un usuario desea trabajar con un objeto de una aplicación de terceros, su menú puede estar disponible dentro de la aplicación. En el mismo caso, puede interactuar como un contenedor de objetos de una gran variedad de aplicaciones de terceros, Los frames de la plataforma digital pueden ser insertados en otras aplicaciones y actualizadas en tiempo real.

Algunas de las aplicaciones externas que pueden ser integradas dentro del ambiente de Arena son OLE Servers como Microsoft Excel y Microsoft Word. Otros son presentados como controles OLE (pequeños componentes de software reusables que proveen una funcionalidad especializada a cualquier aplicación).

Este software también provee soporte para la tecnología estándar DDE de Microsoft (Dinamic Data Exchange). DDE es usado para pasar información en tiempo real desde la plataforma digital a otra aplicación donde estos son tratados como datos nativos y que pueden ser incluidos dentro de un análisis y cálculos. DDE también es usado para pasar información de otras aplicaciones dentro del software.

Las capacidades de desarrollo dentro de la plataforma digital.

El software para control de la plataforma digital cuenta con un lenguaje script que provee una flexible y estable herramienta de desarrollo para aquellas instituciones y usuarios que desean personalizar o adicionar funcionalidad a la plataforma digital. Este lenguaje script, esta basado en Visual Basic for Aplications, puede accesar, controlar y manipular cualquier aplicación, componente de software u objeto con una interfaz OLE.

Esto incluye a todos los objetos de la plataforma digital, objetos OLE y controles incluidos en una área de trabajo del software, y otros muchos objetos externos dentro de aplicaciones servidoras de OLE Automation como Microsft Excel y Word. El lenguaje script puede ser usado para escribir programas que manipulan e integran todos los objetos para una gran diversidad de propósitos.

El Servicio de Traducción de Protocolo y Distribución de Información.

Debemos considerar que la fuente de información es proporcionada por el usuario mediante la renta de un servicio y que para su manipulación nos apoyamos de un software llamado el cual sustrae la información que es recibida de la fuente, la procesa y obtiene un filtro ASCII, el cual es enviado a través de un puerto serie de la Computadora.

La información que es extraída y filtrada (los tipos más importantes que tenemos son: Páginas, Noticias, Hechos de acciones de Mercados de Capitales, Posturas de acciones de Mercados de Capitales, Hechos de Warrants, Posturas de Warrants, Indices, Indices IPC, etc.) cuenta con un formato general y a partir de este se derivan el resto de los formatos de cada uno de los tipos de información (estos formatos fueron descritos en el punto 2.1. Recopilación y clasificación de la información).

La información que es distribuida a los usuarios mediante las peticiones que realizan desde su aplicación Arena deben cumplir con cierto formato (el cual hemos discutido en el punto 2.3.3. Formato de despliegue de la información financiera) para que ésta pueda ser procesada y la cual es diferente al formato entregado por la fuente de información. Por lo tanto, es necesario desarrollar un servicio que corra en el Sistema Operativo Windows NT Server que básicamente realice dos funciones: una es la traducción de protocolo (permite la traducción del formato entregado por la fuente de información al requerido por la aplicación del usuario) y la otra es la poder atender todas las peticiones de clientes en una ambiente de computo Cliente/Servidor sobre una plataforma digital.

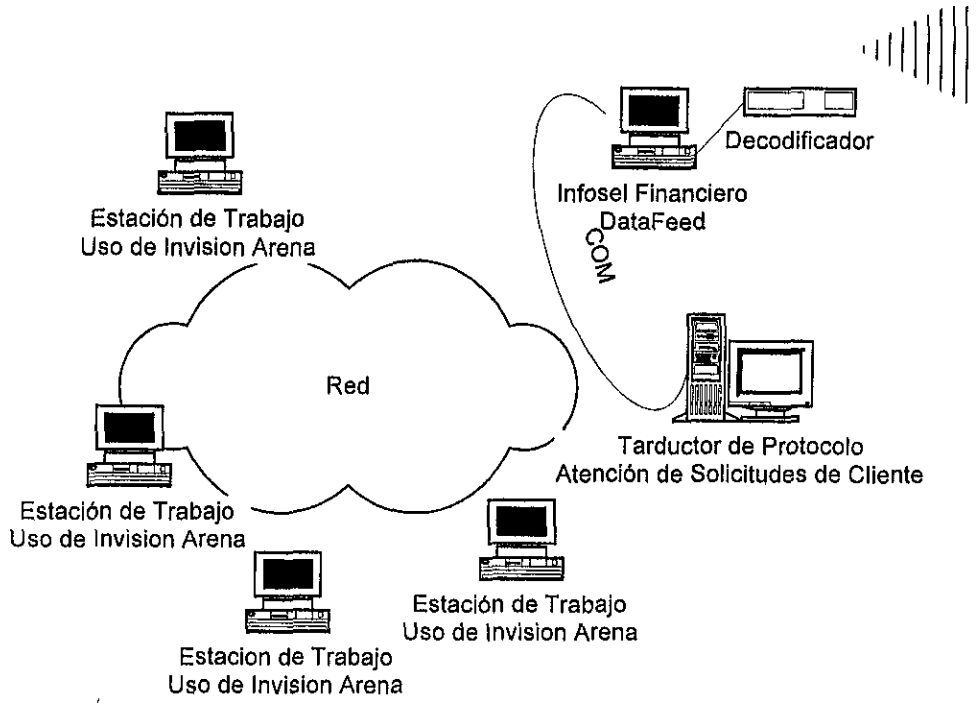


Figura 2.6.3. Esquema de Funciónamiento con Traducción de Protocolo.

La programación de este servicio proponemos sea realizada en el Lenguaje C++ y así aprovechar todas la bondades que esto conlleva, entre la más importante es la teoría de orientación a objetos, entre las características de su funcionamiento, toda la información publicada debe encontrarse en la memoria principal del servidor. El esquema anterior representa toda la funcionalidad propuesta en esta solución.

Evaluación de la Solución.

Las ventajas de esta solución son:

- El usuario cuenta con las licencias de uso de los programas que se proponen en la solución, además de contar con la infraestructura de hardware requerida.
- El desarrollo e implementación de la solución es rápida en comparación a las propuestas anteriores, garantizando toda la funcionalidad requerida por el usuario.

- El Lenguaje de programación propuesto permite independencia tecnológica y de infraestructura dentro de la solución, además de garantizar total aprovechamiento de los recursos del servidor.
- No existen demasiados elementos dentro de la solución lo que permite menos puntos de ruptura dentro de la misma.

Las desventajas son:

- El sistema Arena tiene un costo elevado por licencia sin embargo, no tanto como el de SIVA25.
- Se requiere establecer un plan de mantenimiento preventivo y correctivo al software desarrollado, así como mantenimiento perfectivo en el caso de puntos de mejora propuestas por los usuarios.

Elección de Solución.

Los elementos considerados para la elección de la solución fueron:

- Inversión del usuario en la adquisición de todos los elementos involucrados en la solución.
- Tiempo de Implementación y Operación de la solución.
- Robustez y funcionalidad de la solución. Entendemos por robustez los puntos de ruptura que llegara a tener el sistema debido a la interface de los diferentes elementos que componen la solución.

Características de la Solución	**Desarrollo usando Tecnología Internet**	**Programa Comercial SIVA 25**	**Desarrollo de Programa de Traducción de Protocolo y Sotware de la Plataforma Digital**
Inversión del usuario	Alta	Alta	Mínima
Tiempo de Desarrollo e Implementación	8 meses	4 meses	3 meses
Integración de los elementos para brindar la funcionalidad	Baja	Alta	Alta

Consideramos que la mejor alternativa para el usuario es el desarrollo del servicio protocolo de traducción entre la Fuente de Información y el Sistema Administrador de la Plataforma Digital, además de la funcionalidad de distribución a los usuarios. Es una solución sencilla que no involucra demasiados elementos dentro su y no requiere grandes montos de inversión debido a se que cuenta con la infraestructura necesaria para su implementación.

2.7 Requerimientos de Hardware y Software para la integración de la Aplicación.

2.7.1 Componentes de Hardware.

El sistema podrá operar en diferentes plataformas de sistemas operativos así como tendrá la capacidad de una amplia conexión en red. Todos los componentes de hardware, sistema operativos y de red en la industria estándar que existen en el mercado, pueden ser utilizados por el sistema, no es necesario un equipo sofisticado como se muestra en la figura 2.7.1 Plataformas de conexión.

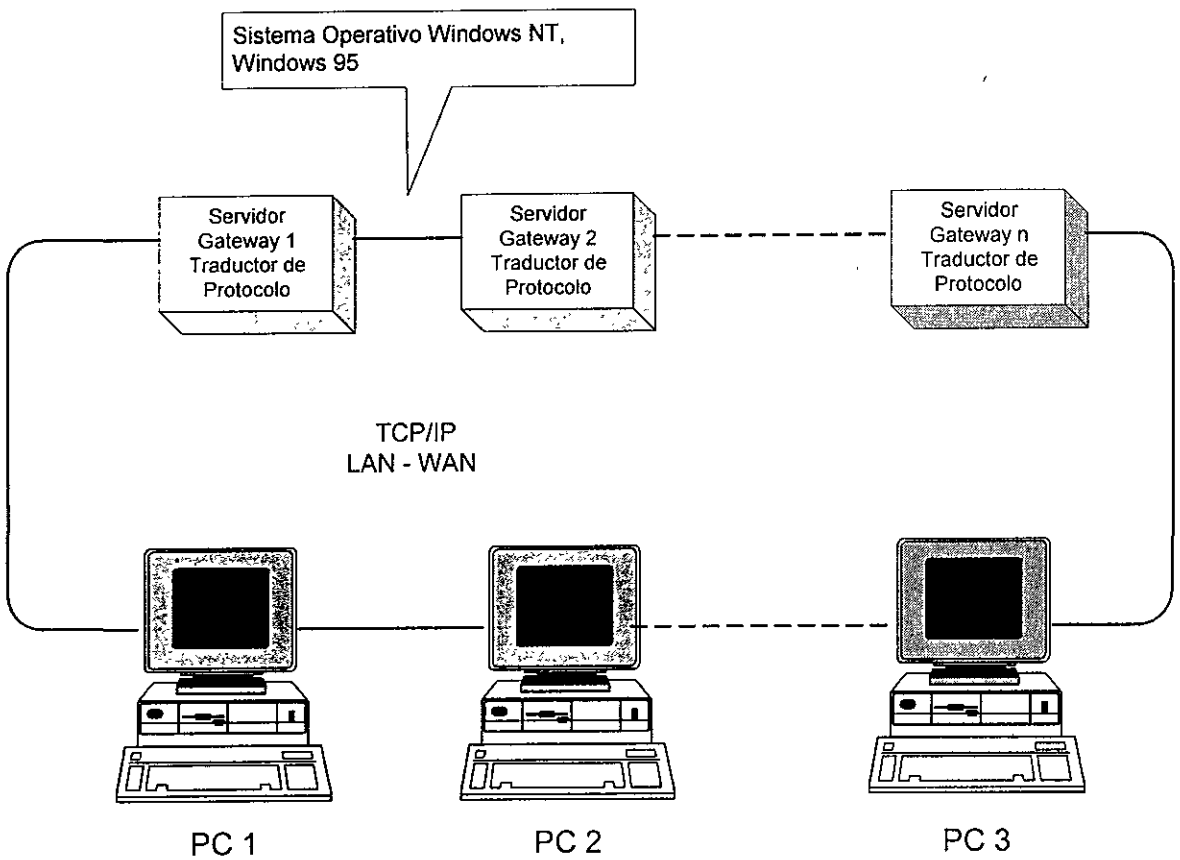


figura 2.7.1. Plataformas de conexión.

Se necesitará un cierto número de PC´s conectadas a una LAN en donde una computadora estará configurada como un servicio de gateway ó servidor , para la distribución de los datos a través de la red. Los sistemas operativos en los que puede correr el sistema deben de tener la característica de multitarea ya que el servicio que proporcionará el gateway será para una serie de PC´s que tienen que accesar al sistema simultáneamente. Windows NT cumple con esta característica dentro de los estándares.

Los requerimientos de hardware varían dependiendo del sistema operativo que se utilice, la aplicación cliente puede instalarse en diferentes sistemas operativos como son Microsoft Windows 95, Windows for Workgroups, Windows NT. Cada PC puede estar configurada para correr en la red con diferente plataforma, lo único que se necesita es tener una tarjeta de red para conectarse a esta. Las especificaciones mínimas de hardware y sistema operativo para que la aplicación pueda correr en Microsoft Windows 95, Windows for Workgroups y Windows NT son:

- una PC Pentium
- 133 MHz o con mayor velocidad
- 16 MB en RAM
- 1 GB como mínimo de espacio en disco duro.

Las configuraciones para el monitor que soporta el sistema son los estándares del mercado como son Super VGA, XGA; también puede soportar los formatos de pantallas más largas de 17" y 20".

Configuración del Sistema en RED.

El sistema funcionando en red puede soportar a la vez varios gateways conectados entre sí atendiendo a varias PC´s en dos ó mas redes de área local.

Debido a que un gateway nos permite conectarnos con otras redes que utilicen diferentes sistemas operativos así como en diferentes protocolos de transporte, utilizaremos un gateway para conectarnos de una red en la que se tenga un sistema operativo como Windows NT con otra red que utilice diferente sistema operativo aunque el protocolo de transporte sea el mismo TCP/IP. El servidor y las PC´s se conectan a la misma LAN y pueden accesar al servicio de gateway . Si se requiere conectar dos o más redes LAN con diferente hardware ó protocolo de comunicación se puede hacer utilizando *bridges* **(puentes)**; ya que estos manejan los paquetes de datos basándose en la dirección de la que proviene el mensaje y en la dirección a la que se dirige por lo tanto se requiere que las direcciones estén en el mismo formato.
Además una LAN se puede configurar para atender hasta 100 usuarios sin que se vea afectado el rendimiento de esta; ya que los tiempos de respuesta son mínimos tanto para el que hace la petición como para el que la recibe.

Los protocolos de red que se pueden usar son los estándares utilizados en la industria, como pueden ser NetBIOS o TCP/IP, incluyendo Ethernet y Token Ring. El protocolo que se utilizó para esta aplicación es el TCP/IP, debido a que es un protocolo estandarizado de comunicaciones que se puede utilizar en diferentes plataformas, con equipos que tengan diferente hardware; además de que permite el ruteo hasta el destino final y una conectividad de manera universal debido a que cada computadora dentro de una red con TCP/IP es reconocida por medio de una dirección que la identifica. Una LAN puede estar configurada para diferentes tipos de protocolos y conectarse por medio de bridges para formar una red lógica simple. Ver fig . 2.7.2 .

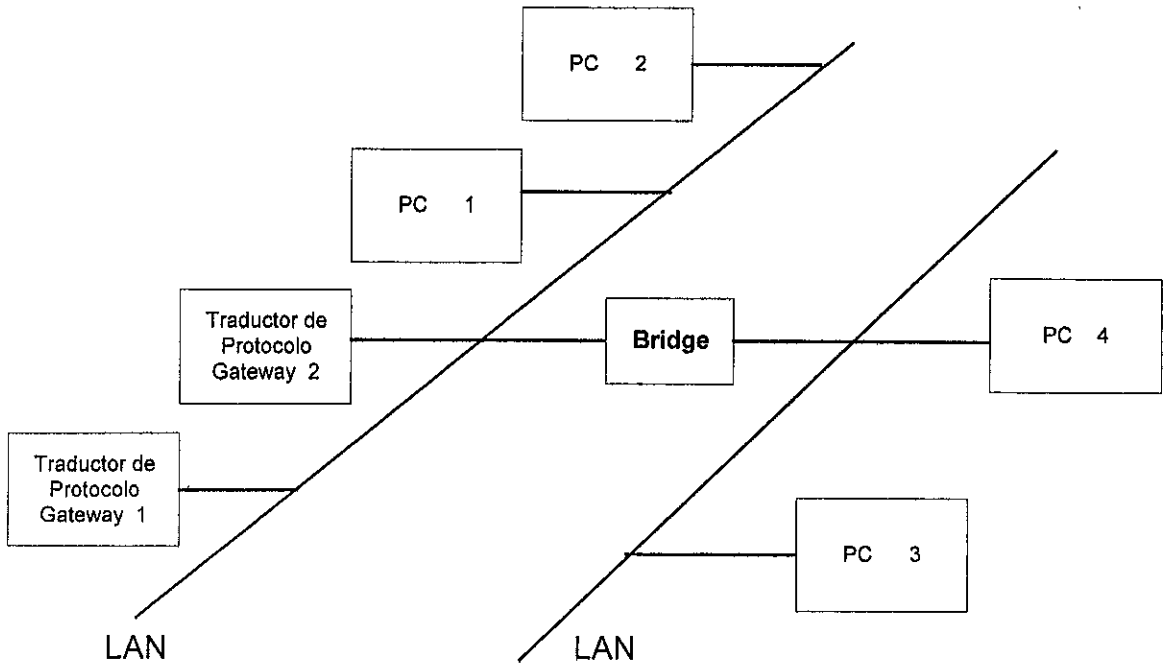


figura. 2.7.2. Configuración del Sistema conectado a dos LAN.

Diferentes tipos de protocolos pueden ser utilizados en la misma red; esto es con un solo gateway (que es el servicio del Traductor de Protocolo) se pueden comunicar dos usuarios con diferente protocolo.

La comunicación entre una PC y el servicio de gateway es punto a punto, esto quiere decir que los datos recibidos están libres de error. No se requiere ningún sistema operativo de red especial para la aplicación ya que esta puede residir en cualquiera de los siguientes : Novell Netware, IBM OS/2 LAN server, ó Microsoft Windows for Workgroups.

Si es necesario atender a más usuarios (mas de 100) y se requiere una configuración muy grande de red, es decir que se necesite conectar varias redes LAN esto puede hacerse segmentando una LAN o instalar un sistema de distribución de cascada. Ambas configuraciones se caracterizan por que balancean la sobrecarga en la red cuando se han hecho varias peticiones al mismo tiempo pueden ser entregadas y actualizadas simultáneamente.

Una LAN segmentada es la división de una red muy grande en varias subredes más pequeñas, con la finalidad de dividir el ancho de banda del tráfico de la red en las subredes garantizando que no se sobrecarguen ninguna de las subredes LAN. Cada división de la LAN forma otra red más pequeña en la cual solo hasta 64 PC´s pueden conectarse en cada subred. Ver figura 2.7.3.

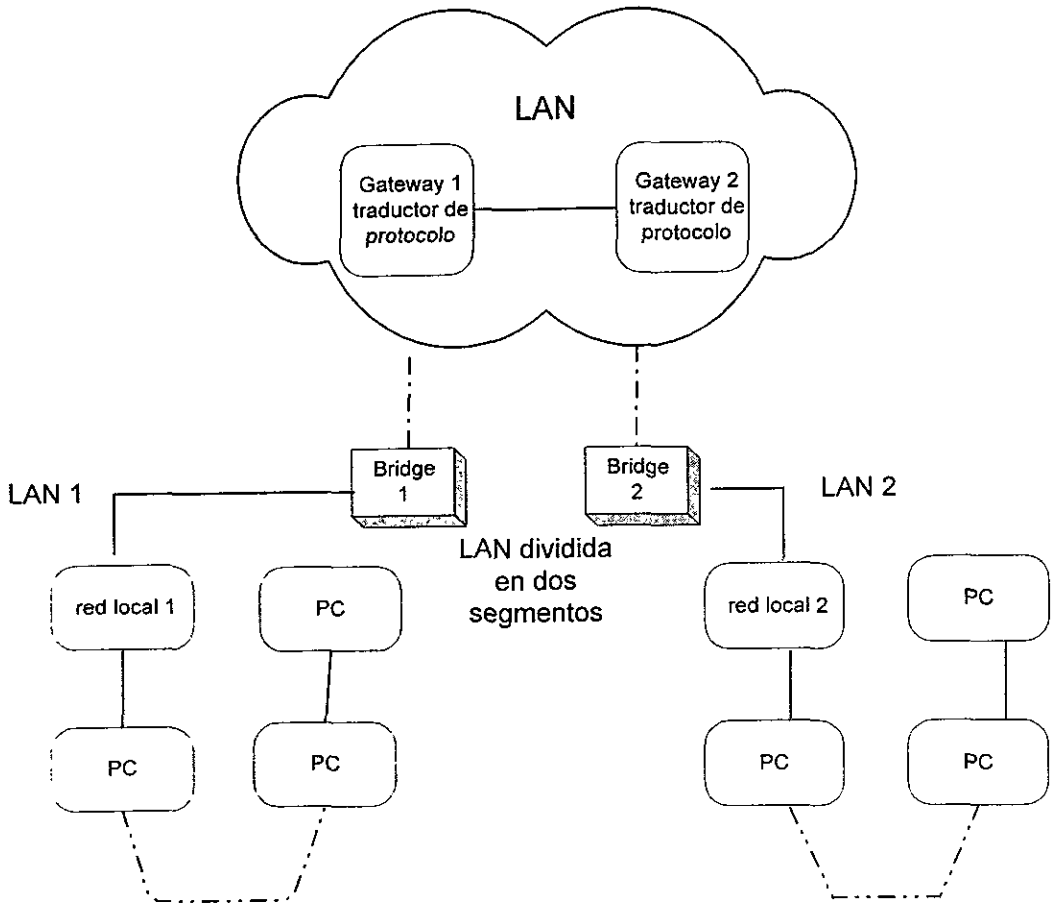


Figura 2.7.3. Configuración de un LAN (dividida).

2.7.2 Componentes de Software.

El sistema se compone de una aplicación cliente y una aplicación servidor y ambas funcionan sobre una LAN usando el procesamiento distribuido. La aplicación del servidor es el Traductor de Protocolo (servicio de gateway) que manipula y procesa los datos desde las diferentes fuentes de información para ser enviados al cliente. Los principales componentes de software para que el sistema sea utilizado son: el Traductor de Protocolo (Servidor), el ODS (Objeto para la distribución del sistema) y el cliente. Por otra parte los servicios fundamentales del sistema son el MP (Multi Procesador) y el CP (Procesos de Conexión) . El Traductor de Protocolo y el ODS forman el servicio del gateway, mientras que el cliente es la aplicación que se encuentra en la PC. El MP y CP nos facilitan el poder desarrollar en diferentes sistemas operativos así como el soporte para diferentes conexiones; ya que permiten tener un procesamiento multitarea y tener un servicio de conectividad remota tanto para el servicio del gateway como para la aplicación cliente en la PC. Ver fig 2.7.4 Estructura Lógica de la aplicación.

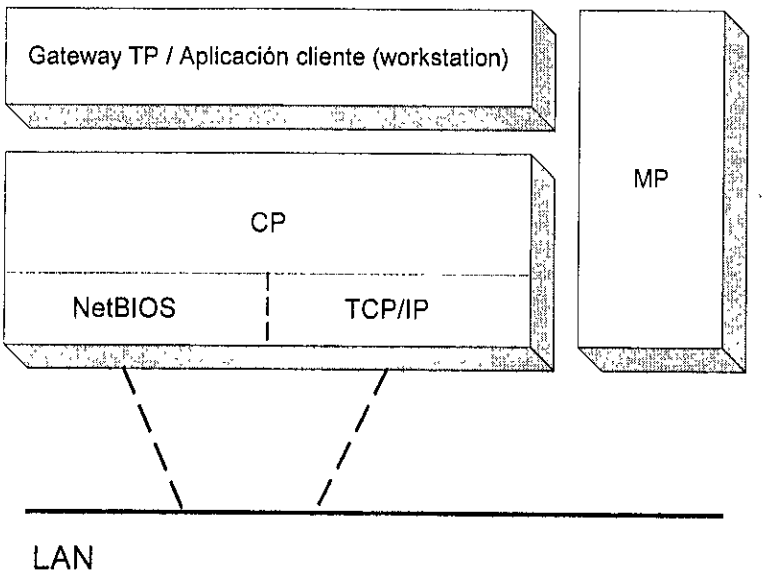


Figura 2.7.4. Estructura Lógica de la aplicación.

El MP nos proporciona una mayor portabilidad del sistema es decir, se puede correr en diferentes ambientes y plataformas independientes, teniendo el manejo de la memoria, el procesamiento, etc. La aplicación puede ser implementada en diferentes sistemas operativos con un mínimo de modificaciones ya que la interface para el MP es la misma en cualquier ambiente.

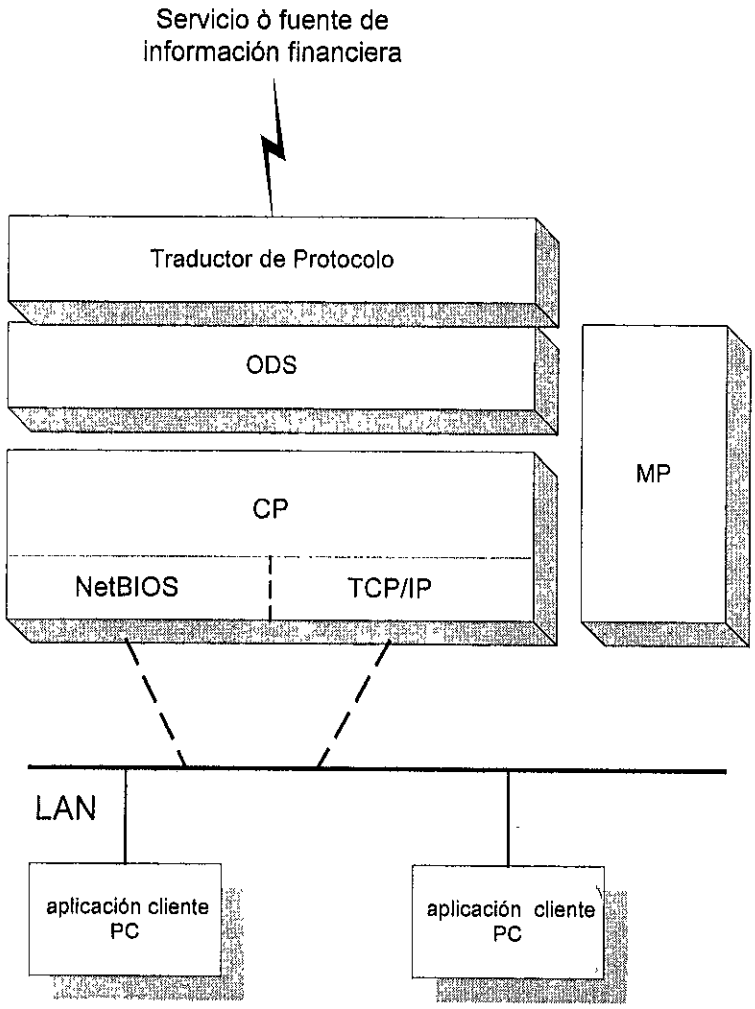
El CP es el encargado de la comunicación entre los procesos como son el servicio de gateway (servidor, traductor de protocolo) y la aplicación cliente, soportando el procesamiento distribuido y administrando la comunicación entre los procesos remotos sobre la red. También maneja la comunicación entre los procesos locales corriendo en el mismo nodo. Si la comunicación es local o remota es transparente para la aplicación del sistema, ya que está se realiza por medio del protocolo, el CP maneja la comunicación utilizando el mecanismo de transporte más apropiado según sea la llamada remota ó local. Puede soportar diferentes protocolos siempre que estos corran en el mismo nodo de la red es decir, el servidor de gateway TP puede comunicarse con una aplicación que utilice NetBIOS y se puede comunicar con otra aplicación cliente simultáneamente que este utilizando el protocolo TCP/IP. Cuando la PC inicia la comunicación con el servidor del gateway , el CP inmediatamente identifica el protocolo de comunicación con el que se esta enviando el mensaje, entonces usa el mismo protocolo para el siguiente mensaje que le sea enviado desde la misma PC.

La aplicación servidor que reside en el gateway es la que administra la información de la fuente de información financiera y controla la distribución de los datos hacia toda la red. Cada gateway en una red es un servidor de aplicación el cual consta de dos componentes principales: el Traductor de Protocolo y el ODS. El PT es el que administra la comunicación entre el servicio o fuente de información para el que el gateway es dedicado. El ODS es el que controla la distribución de los datos desde el servicio a través de la red. Ver figura 2.7.5 Diagrama del servicio de gateway.

Se requerirá de un software adicional como son las API´s que proporcionan la facilidad de desarrollar una aplicación que administre un nuevo servicio de gateway (servidor TP), y distribución de su propia información en tiempo real.

Aplicación Cliente. La aplicación cliente reside en las PC´s y es la que se encarga de procesar y desplegar la información en la pantalla que se encuentra disponible en el servidor del gateway.

Además puede correr en diferentes sistemas operativos como son Windows 95, Windows NT, Windows for Workgroups, por medio de un API.



2.7.5. Diagrama del servicio de gateway.

A continuación se describen los requerimientos mínimos de las plataformas en las que puede ser instalado el sistema.

Windows for Workgroups.

- Sistema operativo MS-DOS versión 6.0 ó mayor
- Una PC con microprocesador Pentium como mínimo
- 166 Mhz.
- 16 MB en RAM
- 1 GB en disco duro como mínimo
- Tarjeta de vídeo SVGA

Windows NT

- Sistema operativo Windows NT versión 3.5 ó mayor
- PC con microprocesador intel 32-bit Pentium
- 166 Mhz.
- 85 MB de espacio libre en disco duro
- 16 MB en RAM.
- 1 GB en disco duro como mínimo.

Windows 95

- Sistema operativo MS-DOS versión 3.2 (mínimo) se recomienda la 5.0
- PC con microprocesador Pentium.
- 16 MB en RAM
- 133 Mhz.
- 55 MB de espacio libre en disco duro
- 1 GB en disco duro como mínimo.

CAPÍTULO III

ANALISIS, DISEÑO E IMPLEMENTACIÓN DEL SISTEMA

3.1 Análisis y Diseño del Sistema Traductor de Protocolo.

Análisis.

El objetivo del análisis orientado a objetos es desarrollar una serie de modelos que describan la aplicación de computadora al trabajar para satisfacer un conjunto de requisitos definidos por el cliente. El análisis orientado a objetos se basa en un conjunto de principios básicos, estos son:

- Se modela el dominio de la información
- Se describe la función del modelo.
- Se representa el comportamiento del modelo.
- Los modelos se dividen para mostrar mas detalles.
- Los modelos iniciales representan la esencia del problema mientras que los últimos aportan detalles de la implementación.

El propósito del análisis orientado a objetos es definir todas las clases (y las clases y comportamientos asociados a ellas) que son relevantes al problema que se va a resolver. Para cumplirlo hemos ejecutado las siguientes tareas:

- Los requisitos básicos del usuario fueron comunicados entre el cliente y el desarrollador de la aplicación.
- Identificamos las clases (es decir, definimos atributos y métodos).
- Especificamos una jerarquía de clases.
- Representamos las relaciones objetos a objeto (conexiones de objetos).
- Modelamos el comportamiento de los objeto.

La popularidad de las tecnologías de objetos ha generado docenas de métodos de análisis orientados a objetos. Cada uno de ellos introduce un proceso para el análisis de un producto o sistema, un conjunto de modelos que evoluciona fuera del proceso, y una notación que posibilita al desarrollador de la aplicación crear cada modelo de una

manera consistente. El método utilizado por nosotros es el de Rumbaugh ya que se apega a la métodología utilizada por nuestro cliente.

Rumbaug y sus colegas desarrollaron la Técnica de Modelado de Objetos para el análisis, diseño del sistema y diseño del nivel de objetos. La actividad de análisis crea tres modelos: el modelo de objeto (una representación del comportamiento, clases, jerarquías y relaciones), el modelo dinámico (una representación del comportamiento de sistema y de los objetos) y el modelo funcional (una representación a alto nivel del flujo de información a través del sistema similar al Diagrama de Flujo). A continuación se muestra una breve descripción del proceso de análisis orientado a objetos de Rumbaugh.

- Desarrollar una declaración del ámbito del problema.
- Desarrollar un modelo de objetos.
  - Identificar clases relevantes al problema.
  - Definir atributos y asociaciones.
  - Definir enlaces de objetos.
  - Organizar las clases de objetos usando una jerarquía.
- Desarrollar un modelo dinámico.
  - Preparar escenarios.
  - Definir eventos y desarrollar una traza de eventos para cada escenario.
  - Construir un diagrama de flujo de eventos.
  - Desarrollar diagramas de estados.
  - Revisar el comportamiento para comprobar consistencia.
- Desarrollar un modelo funcional para el sistema.
  - Identificar entradas y salidas.
  - Usar diagramas de flujo de datos para representar transformaciones del flujo.
  - Desarrollar especificaciones del proceso para cada función.
  - Especificar criterios de restricciones y optimización.

La primera parte del método consiste de desarrollar la declaración del ámbito del problema, el cual ya ha sido tratado en el tema *requerimientos del usuario* (Capitulo 2.4), donde se desarrollaron los escenarios de uso básico para el sistema.

Diseño.

El *diseño orientado a objetos* (DOO) transforma el modelo de análisis creado usando el análisis orientado a objetos en un modelo de diseño que sirve como un anteproyecto para la construcción del software, el DOO esta constituido por un cierto número de diferentes niveles de modularidad así que los principales componentes del sistema están organizados en módulos denominados subsistemas. Los datos y las operaciones que se manipulan se encuentran encapsulados en los objetos, en una forma modular formando el bloque de construcción de un sistema orientado a objetos (OO). Por lo tanto el DOO debe de describir la organización de datos específicos, de atributos y los detalles de las operaciones en cada procedimiento.

El diseño orientado a objetos esta basado en cuatro importantes conceptos del diseño de software que son la abstracción, ocultamiento de la información, independencia funcional y modularidad; los cuales le permiten al diseñador menor complejidad en el desarrollo del sistema.

Para entender un poco más acerca del DOO podemos definir un diseño en pirámide el cual consta de cuatro capas como se muestra en la siguiente fig. 3.1.1 Pirámide del Diseño Orientado a Objetos.

Capa del Subsistema. En esta capa se aloja una representación de cada uno de los subsistemas que le permiten al software conseguir que los requisitos definidos en la etapa del análisis e implementar la infraestructura técnica.

Capa de clases y objetos. Contiene clases y sus jerarquías entre ellas las cuales nos permiten crear el sistema usando generalizaciones mejor definidas. También en esta capa se representa el diseño para cada objeto.

Capa de mensajes. Contiene los detalles que le permiten a cada objeto tener una comunicación con los demás y es aquí en donde se establecen las interfaces externas e internas para el sistema.

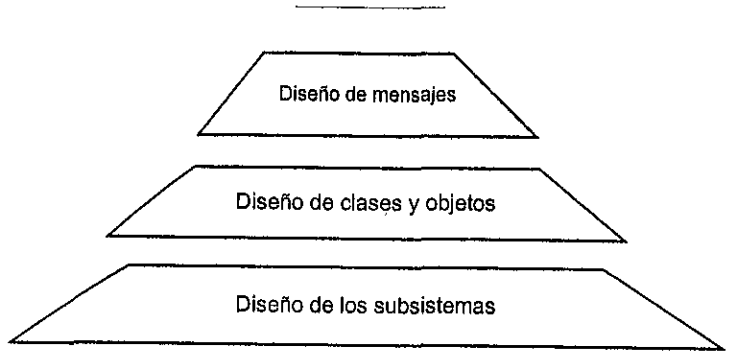


Figura. 3.1.1 Pirámide del Diseño Orientado a Objetos.

A continuación se describirán algunas características de los métodos de diseño orientado a objetos, con la finalidad de aportar una visión rápida del DOO propuesto por los autores del método.

Método de Coad y Yourdon [COA91].

Componentes del dominio del problema:

- Agrupar las clases específicas al dominio
- Diseñar una jerarquía de clases apropiada para las clases de aplicación
- Simplificar la herencia
- Desarrollo de una interface con el componente de gestión de datos
- Revisar el diseño y proponer lo que sea necesario para el análisis.

Componente de interacción humana:

- Definir los actores humanos.
- Desarrollar escenarios para las tareas.
- Refinar la secuencia de interacción del usuario.

- Diseñar clases relevantes y la jerarquía de clases e integrar las clases de GIU (Interfaz Gráfica de Usuario).

Componente para la gestión de tareas:

- Identificar tipos de tareas (eventos)
- Establecer prioridades
- Identificar la tarea que servirá de coordinadora para otras tareas.
- Diseñar objetos apropiados para cada tarea

Componente para la gestión de datos:

- Diseñar las estructuras de datos y su distribución.
- Diseñar servicios necesarios para manejar las estructuras de datos
- Identificar las herramientas para la implementación de la gestión de datos.

Método de Jacobson.

La actividad de diseño para la ISOO (Ingeniería de Software Orientada a Objetos) [JAC92] es una versión simplificada del método desarrollado por Jacobson, el modelo de diseño hace hincapié en el seguimiento al modelo del análisis. Jacobson nos presenta el proceso de DOO como sigue:

- Considerar adaptaciones para hacer que el modelo de análisis ideal cumpla con el entorno del mundo real.
- Crear bloques como objetos del diseño primario. Definir bloques de control, de entidades, de interfaces, así como un bloque para implementar el análisis de objetos relacionados, para describir como los bloques se comunican durante la ejecución.
- Crear un diagrama de bloque de interacción entre los estímulos de cada bloque.
- Organizar los bloques en subsistemas y revisar el trabajo de diseño.

Método de Booch [BOO94].

Planificación arquitectónica:

- Agrupar objetos similares en particiones arquitectónicas separadas.
- Distribuir los objetos en capas según los niveles de abstracción.

- Crear un prototipo de diseño.

Diseño Táctico:

- Definir políticas independientes del dominio (es decir, las reglas que gobiernan el dominio del uso de operaciones y atributos).
- Definir políticas específicas al dominio para la administración de memoria, la gestión de errores y otras funciones de la infraestructura.
- Crear un prototipo para cada política.

Método de Rumbaugh.

Mejor conocido como TMO [RUM91] Técnica de Modelado de Objetos. A continuación se describen algunos puntos importantes del diseño orientado a objetos de Rumbaugh.

- Realizar un diseño del sistema, dividir el modelo de análisis en subsistemas, así como identificar la concurrencia dictada por el problema.
- Elegir una estrategia básica para la implementación de la gestión de datos.
- Identificar los recursos globales y los mecanismos de control requeridos para acceder a ellos.
- Construir un diseño de control, seleccionar las operaciones del modelo de análisis, definir algoritmos para cada operación, seleccionar las estructuras de datos apropiadas para los algoritmos, definir todas las clases internas, diseñar los atributos de las clases, revisar la organización de clases para optimizar el acceso a los datos y mejorar el rendimiento computacional.
- Implementar mecanismos de control definidos en el diseño del sistema.
- Diseñar el intercambio de mensajes para implementar relaciones entre objetos (asociaciones).
- Empaquetar las clases y asociaciones en módulos.

La diferencia entre los métodos mencionados anteriormente se basa en el enfoque de cada uno de ellos. El método de Coad y Yourdon se enfoca en el diseño que se dirige no solamente a la aplicación, sino también a la infraestructura para la aplicación.

El método de Booch abarca un proceso de desarrollo mínimo en el cual se definen el conjunto de tareas de diseño que se reaplican en cada etapa del proceso de macro-desarrollo; manteniéndose así un enfoque evolutivo.
El método de Rumbaugh abarca la actividad de diseño del sistema centrándose en la distribución de los componentes necesarios para construir un producto completo.
En este tipo de diseño se enfatiza más a la distribución detallada de un objeto individual.

Una de las diferencias entre el análisis orientado a objetos (AOO) y el diseño orientado a objetos (DOO) es que el AOO es una actividad de clasificación, es decir se analiza un problema para determinar las clases de objetos que serán aplicables al desarrollo de la solución, así como el determinar las relaciones y el comportamiento del objeto. El DOO nos ayuda a descubrir que objetos se derivan de cada clase y como estos objetos se interrelacionan unos con otros, así como es el comportamiento y cómo se comunican entre objetos. El diseño orientado a objetos sigue un flujo genérico desde el análisis hasta el diseño como se muestra en la siguiente figura 3.1.2.

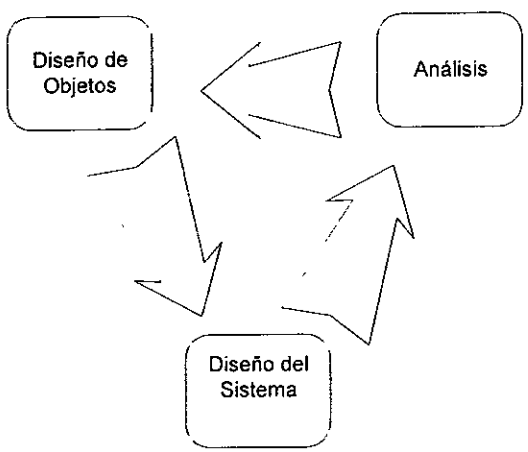


Figura. 3.1.2. Proceso del Diseño Orientado a Objetos.

El diseño del sistema se realiza a través de la descripción de los subsistemas necesarios para implementar los requisitos del cliente y el entorno de soporte requerido para la realización. Dentro del diseño de los subsistemas se definen los siguientes componentes:

- Dominio del problema: los subsistemas responsables de la implementación de los requisitos del cliente.
- Interacción humana: los subsistemas que implementan la interfaz de usuario.
- Gestión de tareas: los subsistemas responsables de control y coordinación de tareas concurrentes que pueden empaquetarse dentro de uno o varios subsistemas.
- Gestión de datos: el subsistema que es responsable del almacenamiento y recuperación de objetos.

El diseño de cada uno de estos componentes se traslada al diseño de objetos como se muestra en la siguiente figura 3.1.3.

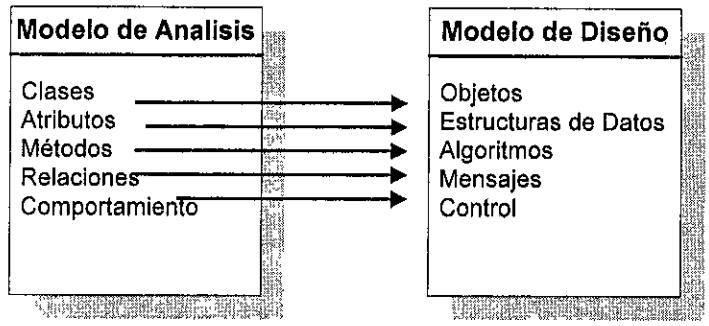


Figura. 3.1.3. Modelo de Análisis - Modelo del Diseño.

El DOO se divide en el modelado de tres partes básicas:

- El Modelo de Objetos
- El Modelo Dinámico y
- El Modelo Funciónal

El método utilizado para la realización de este trabajo fue el método de Rumbaugh debido a que es el método estándar utilizado por el usuario.

3.1.1. Procesos del Sistema.

Todo análisis debe de llevarse a través de un método como se explicó en la sección anterior, y una de las herramientas básicas del análisis y el diseño son las gráficas que nos ayudan a entender el comportamiento del flujo de la información como el funcionamiento del sistema por medio de procesos. El diagrama de procesos nos permite conocer a una determinada organización así como los datos que viajan a través de ellos así como las operaciones. La simbología de un diagrama de procesos es la siguiente: se utiliza un rectángulo para representar una entidad externa, esto es, un elemento del sistema o puede ser otro sistema que produce información para ser transformada por el nuevo sistema, el círculo representa un proceso o transformación que se aplica a los datos y los cambia de alguna forma; la flecha representa uno ó más elementos de datos. La línea doble representa un almacenamiento de datos. Ver fig. 3.1.4 Simbología para el Diagrama de Procesos.

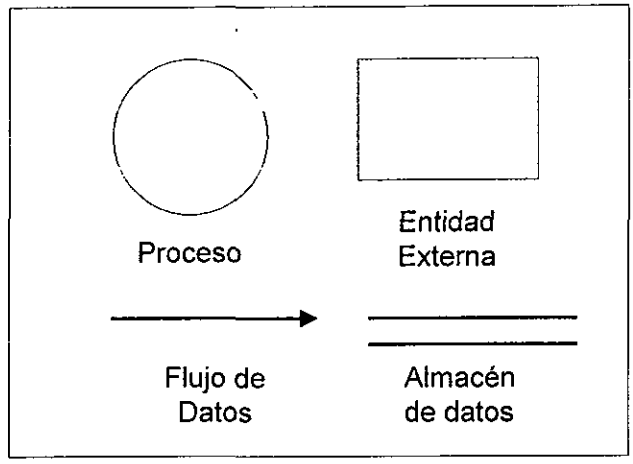


Figura 3.1.4. Simbología para el Diagrama de Procesos.

Diagrama del proceso principal.

Siguiendo la anterior sismología podemos realizar el diagrama del proceso principal del sistema como se muestra en el siguiente diagrama de la figura 3.1.5. Proceso Principal.

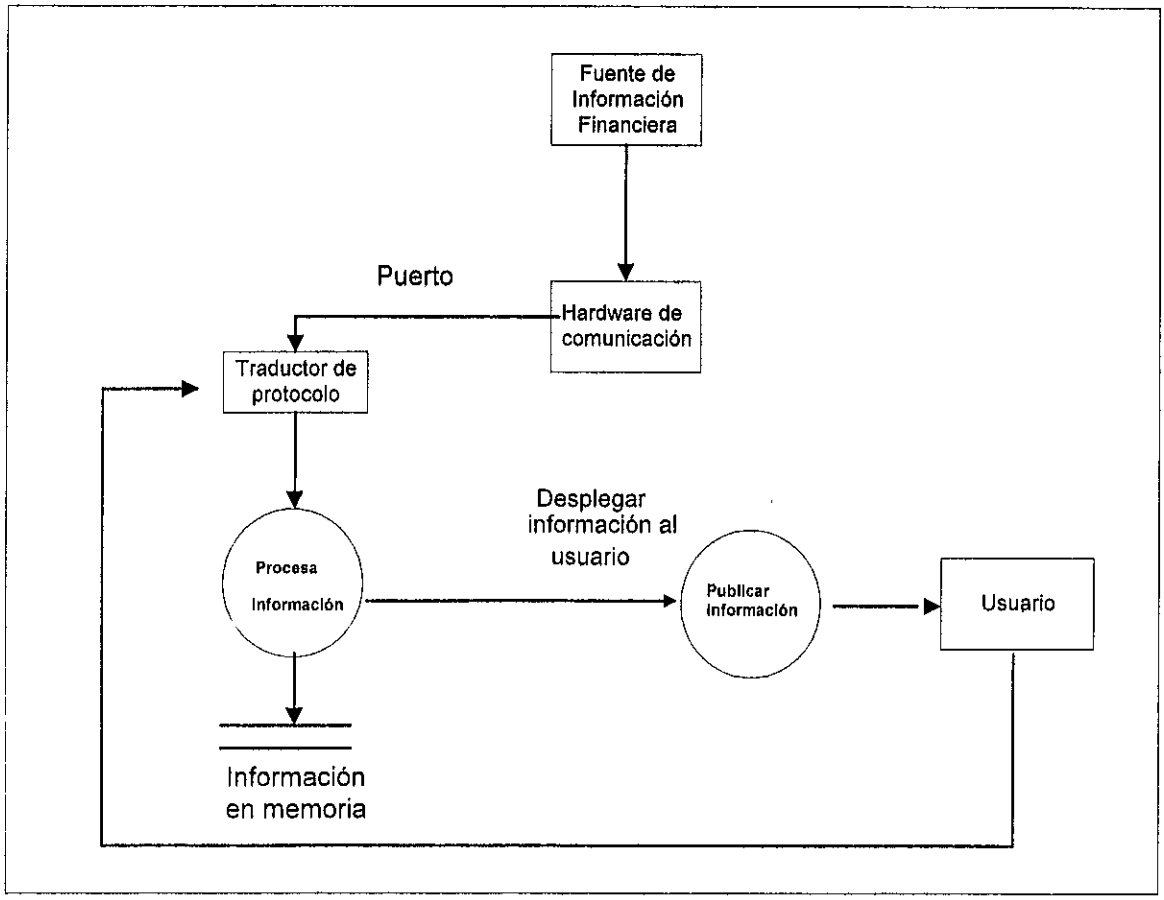


Figura 3.1.5. Proceso Principal.

Proceso Procesar Información Solicitada.

Este proceso se encarga de procesar el registro que cumple con la información solicitada por el usuario . Figura 3.1.5 Procesar Información Solicitada.

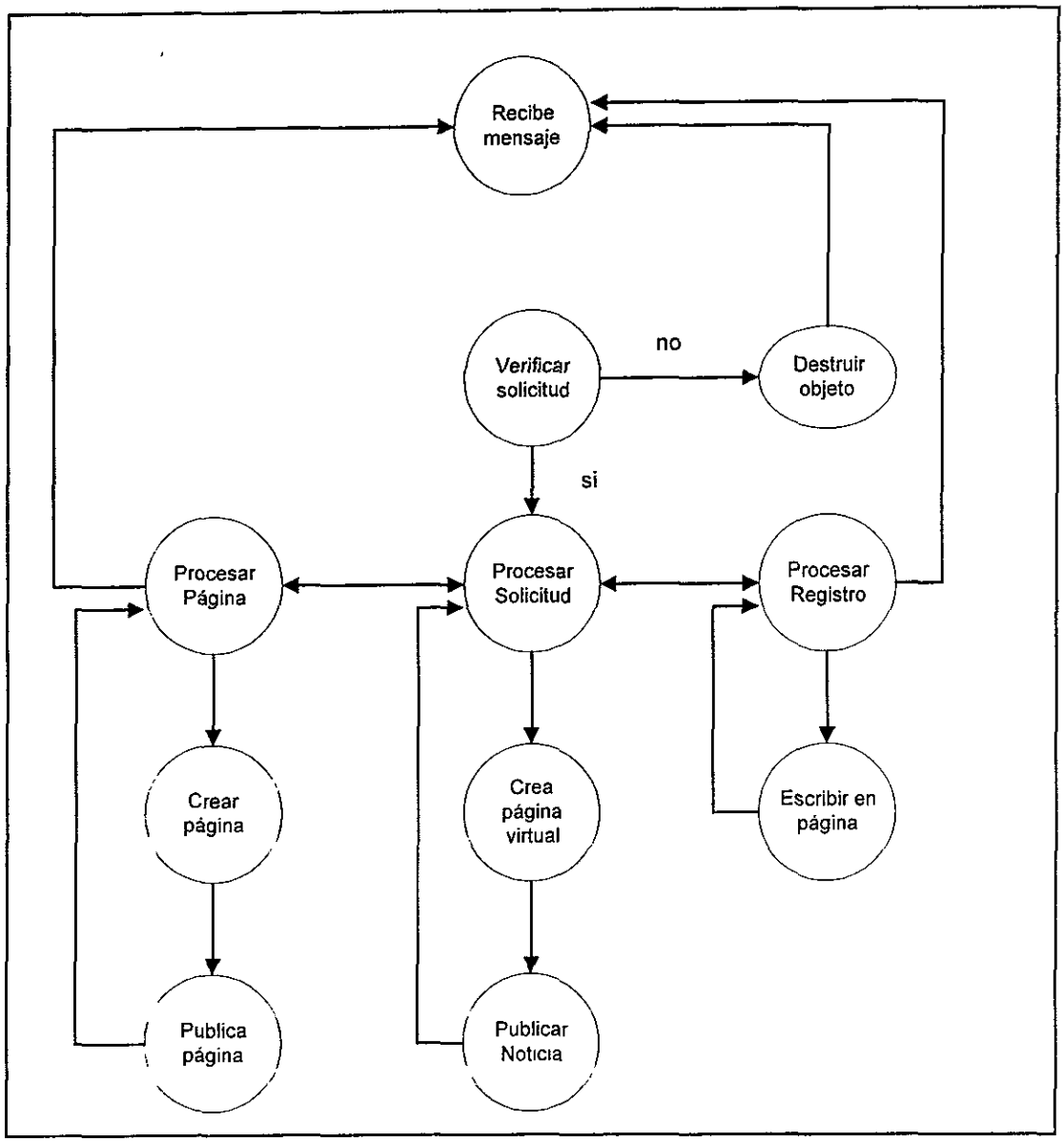


Figura 3.1.6. Procesar Información Solicitada.

Proceso que verifica la llegada de información por el puerto

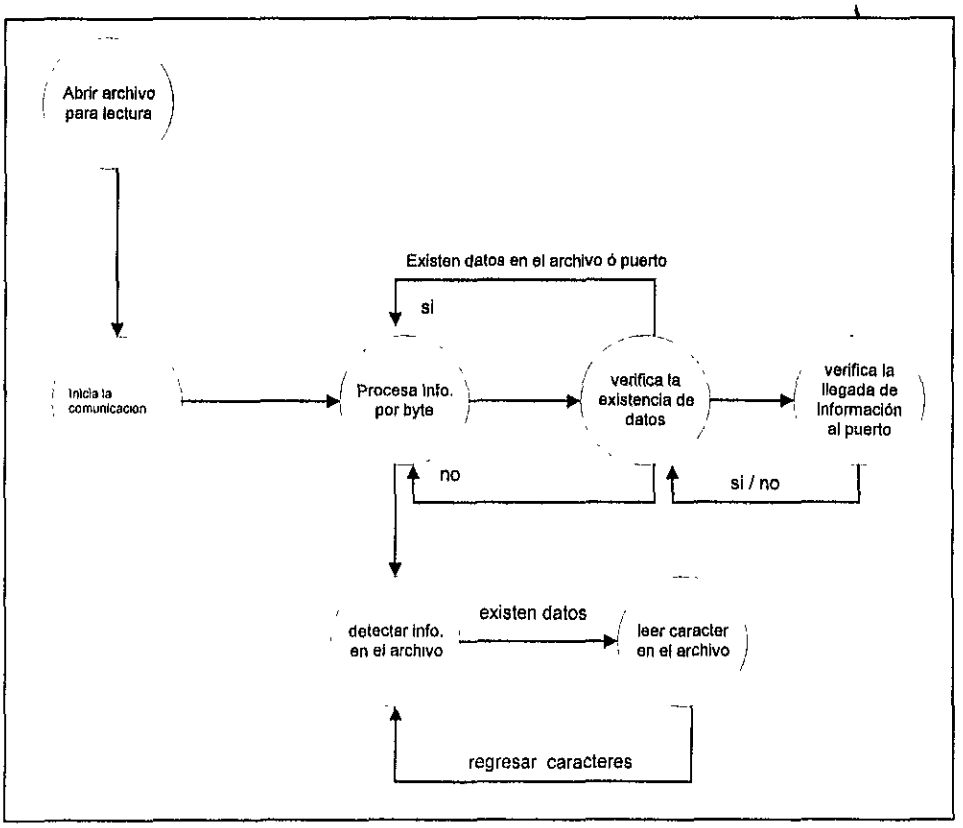


Figura 3.1.7. Arribo de información.

Proceso de lectura de un registro.

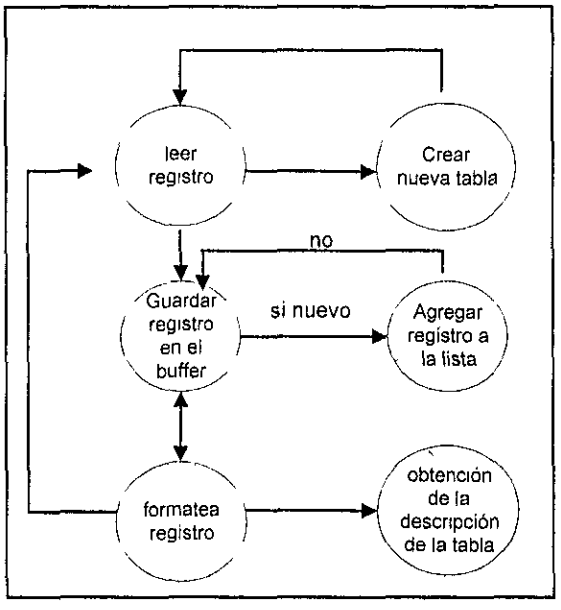


Figura 3.1.8. Registro.

3.1.2. Modelo de Objetos.

3.1.2.1. Modelado de clases-responsabilidades-colaboraciones.

En este punto se deben identificar las clases candidatas, e identificar sus responsabilidades y colaboraciones. El modelado de clases-responsabilidades-colaboraciones aporta un medio sencillo de identificar y organizar las clases que resulten relevantes al sistema.

A continuación hacemos una descripción de todas las clases involucradas en el sistema traductor de protocolo que fueron resultado de análisis hecho, esperando cumplir con requerimientos del usuario.

1. **Nombre de la Clase:** Información
   Tipo de la Clase: Interacción
   Características de la Clase: Persistencia
   Responsabilidades: Administrar los recursos involucrados en la recepción de la información fuente que se recibe de la Bolsa Mexicana de Valores a través del puerto serial o de un archivo generados ambos de la computadora que contiene la información fuente. Cuenta con funciones que le permiten configurar a la aplicación para trabajar con puerto serial o archivo, leer un byte, una cadena de caracteres o un registro.
   Colaboradores: Colabora con la clase Puerto y la clase General para alcanzar su objetivo.

2. **Nombre de la Clase:** Puerto
   Tipo de la Clase: Dispositivo
   Características de la Clase: Inclusividad

Responsabilidades: Administra el Puerto Serie, con funciones que permiten configurar, abrir, probar si hay datos, esperar datos , leer , escribir y cerrar el puerto,.

Colaboradores: Tiene la característica de ser una clase atómica, es decir, no incluye otras clases.

3. **Nombre de la Clase:** Tabla

   Tipo de la Clase: Interacción

   Características de la Clase: Tangibilidad

   Responsabilidades: Esta es una de las clases más importantes del sistema, ya que su principal responsabilidad es la administración de las reglas del negocio utilizadas para la traducción de la información fuente en los formatos utilizados por la plataforma digital de la información financiera (páginas, registros y noticias). Esta traducción se efectúa en forma dinámica en la memoria de la computadora. La funcionalidad de esta clase esta dividida en dos partes, la primera, dirigida hacia el usuario, pues permite la posibilidad de guardar , buscar, reemplazar o borrar los registros en la memoria. La segunda está dirigida a la manipulación de la información por parte del sistema, por ejemplo, comparar campos, comparar registros, obtener nombre y datos de un campo, insertar datos en un campo o en un registro entre otras.

   Colaboradores: Colabora con la Clase ListaLigada para el manejo dinámico de la memora.

4. **Nombre de la Clase:** ListaLigada

   Tipo de la Clase: Interacción

   Características de la Clase: Secuenciabilidad

   Responsabilidades: Brinda servicios de administración de la información dentro de las registros, por ejemplo, agregar campo, limpiar campo, borrar campo. Otra responsabilidad es la búsqueda de información en los diferentes registros que componen las tablas que contienen la información manipulada por el

Sistema (páginas, registros y noticias); como característica la búsqueda de información debe ser muy rápida y de diferentes formas.

Colaboradores: Colabora con la Clase ElementoLista para el manejo dinámico de la memora.

5. **Nombre de la Clase:** ElementoLista
   Tipo de la Clase: Interacción
   Características de la Clase: Inclusividad
   Responsabilidades: Su responsabilidad es el control de los diferentes elementos que componen cada una de los registros que de una lista ligada, permitiendo las operaciones básicas sobre ellos: creación, borrado y consulta.
   Colaboradores: Tiene la característica de ser una clase atómica, es decir, no incluye otras clases.

6. **Nombre de la Clase:** ControlInvision
   Tipo de la Clase: Interacción
   Características de la Clase: Secuenciabilidad
   Responsabilidades: Este objeto tiene la responsabilidad de administrar las funciones que controlan las librerías de la plataforma digital, así como la administración de las peticiones de los usuarios y la publicación de las noticias, registros ó páginas que satisfacen dichas peticiones.
   Colaboradores: Las clases que colaboran son la de Invision, ObjetoInfosel y Tabla.

7. **Nombre de la Clase:** Invision
   Tipo de la Clase: Interacción
   Características de la Clase: Secuenciabilidad
   Responsabilidades: Tiene la responsabilidad de administrar los servicios que controlan las librerías de la Plataforma Digital (Arrancar , probar y

detener el servicio) , así como la creación de los objetos (Noticia, Página y Registro) para su publicación.

Colaboradores: Api´s de la Plataforma Digital.

8. **Nombre de la Clase:** IndiceNoticias

   Tipo de la Clase: Interacción.

   Características de la Clase: Secuenciabilidad

   Responsabilidades: Permite el control de las Noticias provenientes de la Fuente de Información, con funciones como, el dar el alta de noticias con los objetos asociados, así como su consulta para los casos en los cuales son requeridas por los usuarios.

   Colaboradores: Invision y ListaLigada para el control de los objetos asociados para el manejo de noticias.

9. **Nombre de la Clase:** ListaPáginas

   Tipo de la Clase: Interacción

   Características de la Clase: Secuenciabilidad

   Responsabilidades: Permite el control de las Páginas provenientes de la Fuente de Información, con funciones como, como el dar el alta de páginas con los objetos asociados, así como su consulta y despliegue en pantalla para los casos en los cuales son requeridas por los usuarios.

   Colaboradores: ListaLigada, Tabla y PáginaVirtual para el control de los elementos asociados para el manejo de noticias y su despliegue en pantalla.

10. **Nombre de la Clase:** Publicables

    Tipo de la Clase: Interacción

    Características de la Clase: Secuenciabilidad

    Responsabilidades: Permite el control de cualquier elemento que tenga la característica de poder ser publicado provenientes de la Fuente de

Información, con funciones como, agregar, eliminar y definir cuando un registro es publicable.

Colaboradores: ListaLigada para el control de los objetos asociados para el manejo de noticias.

11. **Nombre de la Clase:** PáginaVirtual

Tipo de la Clase: Dispositivo

Características de la Clase: Inclusividad

Responsabilidades: Tiene la responsabilidad del despliegue de las páginas mediante el mapeo de un registro de longitud variable en un formato de despliegue de n columnas por m renglones. Esto lo realiza mediante funciones que permiten la conceptualización de una página virtual.

Colaboradores: Tiene la característica de ser una clase atómica, es decir, no incluye otras clases.

12. **Nombre de la Clase:** ObjetoInfosel

Tipo de la Clase: Interacción

Características de la Clase: Secuenciabilidad

Responsabilidades: Este es uno de los objetos con mayor responsabilidad pues tiene la habilidad de obtener la información fuente en su formato nativo y traducirlo en un formato consistente dentro de la plataforma digital. Este objeto tiene la habilidad de procesar los registro de la fuente de información y transformarlos en los formatos manejados por la plataforma digital publicando todas aquellas solicitudes provenientes del usuario con funciones que permiten la manipulación de los diferentes tipos manejados en el ambiente financiero (Cadenas de caracteres, números, fechas, horas).

Colaboradores: Esta clase colabora con la clase de Información (determina si la fuente de información proviene de un puerto serial o de un archivo), ListaLigada donde tenemos almacenado todos los índices de los objetos publicables (Noticias, Páginas y Registros)

3.1.2.2. Definición de los Objetos.

IndiceNoticias

Atributos:
Invision *Inv;
ListaLigada lista;

Servicios:
IndiceNoticias();
void AgregarNoticia(unsigned long numero,char *nombre);
};

ControlInvision

Atributos:
Tabla *Pubs;
Invision *Inv;
char *ObjInfo;

Servicios:
ControlInvision();
void ProcesarSolicitud(char *sucio);
void PublicarNoticia(char *nombre);
void ProcesarRegistro(char *nreg);
void ProcesarPagina(char *nreg);
void ProcesarPendientes();
void Prueba(char *nreg);
void ProcesarIndice(char *sucio);

ListaPaginas

Atributos:
char *oi,mensajes;
ListaLigada *PendPub,datos;
Tabla *Pubs;

Servicios:
ListaPaginas();
int GuardarPagina(ListaLigada *l);
ListaLigada *ExisteLista(char *nombre);
void EncontrarDiferencias(ListaLigada *lorig,ListaLigada *l,ListaLigada *difs);
void BuscarDifCadenas(char *mem1,char *mem2, unsigned int nreng, ListaLigada *Dif);
ListaLigada *Existe(char *nombre) { return(ExisteLista(nombre)); }
void Listar();

Invision

Atributos:
IINV_OBJ_REQUEST *obj;
unsigned char buf[500],mensaje;
char mensajes;

Servicios:
Invision()
int EsperarMensaje();
void ArrancarPublisher(char *nombre);
void DetenerPublisher();
void ServicioEnLinea();
void ServicioFueraDeLinea();
long CrearPagina(char *nombre,int c,int r);
long CrearPaginaColor(char *nombre,int x,int y);
void DestruirObjeto(char *nombre,int coderror,char *mensaje) {
InvDestroyObject(nombre,coderror,mensaje); }
int PublicarPagina(char *nombre, long numero,unsigned int tam,unsigned char *txt);
int PublicarPagina(char *nombre, long numero,unsigned int tam,unsigned unsigned int x,unsigned int y,unsigned char *txt
long CrearRegistro(char *nombre,unsigned long numero_de_mascara);
void PublicarPrimerRegistro(char *nombre, long numero,unsigned int tam,unsigned char *txt);
void ActualizarRegistro(char *nombre, long numero,unsigned int tam,unsigned char *txt);

PaginaVirtual

Atributos.
int posx,posy,
int tamx,tamy,
char TAB,RC,AL,ncars,color;
char *buf,traduccion,MCCon,tab,mostrar;

Servicios:
PaginaVirtual();
void operator << (char *txt);
void Escribir(char c);
void Escribir(char *txt);
void Escribir(char *txt,int l);
void Geometria(int lx,int ly,char nc);
void Geometria(int lx,int ly);
void Pos(int x,int y);
void Cls();
int ProcesoDeControl(char c);
void Escribir(long num);

```
Attributos:
unsigned long nt, tamtabmax;
char separador;
ListaLigada nombres,datos;

Servicios:
Tabla();
int NuevaTabla(char *descrip);
void ListarDescripciones();
void ListarDatos();
char *DesTabla(char *nombre);
char *DesTabla(unsigned long n);
int ProducirRegistro(char *datos, char *regtot, unsigned long maxregtot);
int GuardarRegistro(char *reg);
int ReemplazarRegistro(char *donde, char *con, char modo);
int BorrarRegistro(char *donde );
char *BuscarUltimoRegistro(char *donde );
int CompararCampos(char *reg1, char *reg2);
int ContarCamposReg(char *reg);
int ContarCamposDes(char *buf);
int ContarDatosInser(char *buf);
void SacarCampoReg(char *reg,char *buf,unsigned int maxbuf, unsigned int
*pos1);
void SacarNombreCampoDes(unsigned char *buf,unsigned int n,unsigned char
*buf2, int maxbuf2);
void SacarDatoInser(unsigned char *buf,int ndato,unsigned char *nombre,int
maxbufn,unsigned char *dato,int maxbufd);
void SacarDato(unsigned int nc, char *reg,char *buf,unsigned int n);
void FormatearRegistro(char *reg);
```

ElementoLista

```
Atributos:
unsigned long num;
unsigned long tam;
unsigned char tipo;
ElementoLista *sig,*ant,
unsigned char *Mem,*ll;

Servicios:
ElementoLista();
~ElementoLista();
ElementoLista
*ElementoLista::Nuevo(unsigned long
tam_, unsigned char *copia,unsigned char
*l);
ElementoLista *Nuevo(unsigned long tam,
unsigned char *copia);
ElementoLista *Nuevo(unsigned long tam) {
return(Nuevo(tam,NULL)); }
void Borrar();
void ImpInfo();
unsigned char * operator = (char *texto);
```

ListaLigada

```
Attributos:
unsigned long total,indice,totalbytes;
        ElementoLista *primero,*ultimo;

Servicios
ListaLigada(),
~ListaLigada();
void Inic();
unsigned char * Agregar(unsigned long tam,unsigned char
*copia);
unsigned char * operator + ( char *texto );
void Borrar(unsigned char *ap);
void Borrar(unsigned long num);
void Borrar(ElementoLista *ap);
void BorrarPrimero();
void Listar();
void Limpiar();
unsigned long QueNumero(unsigned char *ap);
unsigned long QueNumero(ElementoLista *ap);
unsigned char *QueMemoria(unsigned long num);
unsigned char *QueMemoria(ElementoLista *ap);
ElementoLista *QueElemento(unsigned long num);
ElementoLista *QueElemento(unsigned char *ap);
char *operator [ ] (unsigned long num);
```

ObjetoInfosel

Attributos:
char hayinfo,estado,mensajes,noticias,bmv;
char buftitulos[_MaxBufTit_],bufdatos[_MaxBufDatos_];
char dirnoti[100],actindicenot,actindicepag;
Informacion info;
Tabla ECampos,Datos,*Pubs;
ListaLigada PendPub,INoticias,IPaginas;
ListaPaginas LP;
unsigned int rpnot;

Servicios:
ObjetoInfosel();
void ProcesarInfo();
int LeerObjeto(char *nombre);
char *EncontrarCaracter(char *buf,char c,unsigned long num);
void SacarCadena(char *salida,int tambuf,char *buf,char fin);
void GuardarNoticia(char *nombre,char *RegListo);
int ProcesarReg(char *buftitulos);
void ProcesarPagina(char *cab,char *RegListo);
void TransformarAInvision1(char *reg,char *invreg);
void TransformarAInvision2(char tipo,char *reg,char *invreg);
void CrearRegisroVacio(char tipo,char stipo,char *salida,unsigned int maxs);
int LeerConfExt(char *nombre);
int SacarCadena(unsigned char *buf,unsigned char *destino,unsigned int max,char terminacon);
int SacarNumero(unsigned char *buf,double &val);
int SacarHora(unsigned char *buf,unsigned char *destino);
int SacarFecha(unsigned char *buf,unsigned char *destino);
void ActualizarRegistro(char *nompub,char *reginv);
void QuitarCeros(unsigned char *buf);
void ProcesarIndice(char *nombrebase,ListaLigada *l);
void PublicarIndiceNoticias();
void PublicarIndicePaginas();
void GuardarIndicePagina(char *tit);

Información

Attributos:
Puerto puerto;
FILE *archivo;
char modolectura,mensajes;
char buf[_MaxBufInf_],RegListo[_MaxBufInf_],estado,hayregistro;
unsigned int posbuf,CRC,lonreg;

Servicios:
Informacion();
~Informacion();
void UsarPuerto(char *com);
void UsarArchivo(char *nombre);
void UsarNada();
char HayDatos();
int HayRegistro();
char LeerByte();
void LeerRegistro(char *buf);
void ProcesarByte();
void MeterByte(char c);
void SacarRegistro(char *buf_);
unsigned int QueCRC(unsigned char *buf);

Puerto

Atributos:
unsigned long id;
HANDLE com,h;
char mensajes,activo;
char nombrep[50];

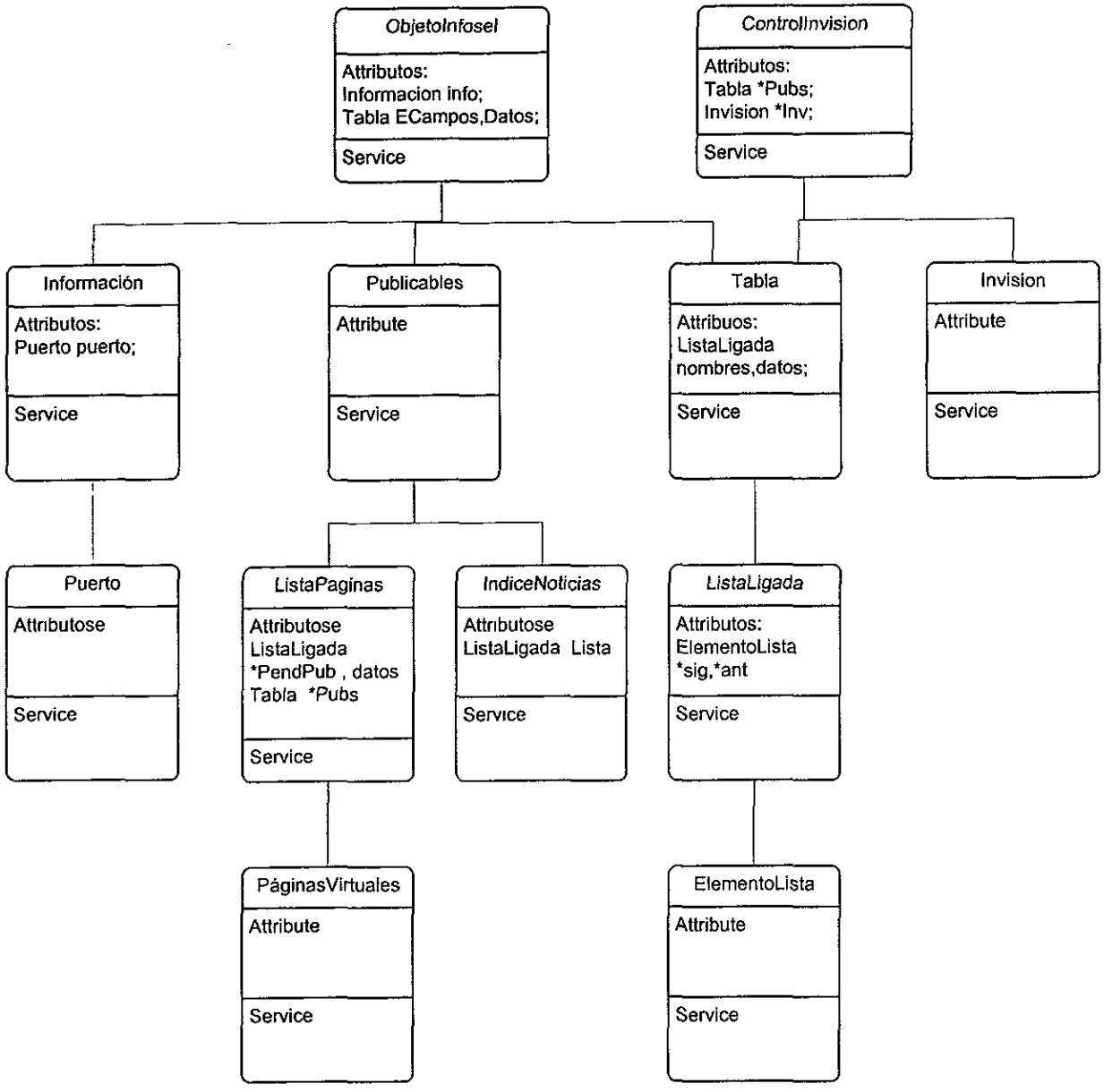
Servicios:
Puerto();
~Puerto() ;
int AbrirPuerto(char *_nombre);
void Configuracion();
void Escribir(char c);
char Leer();
char EsperarCar();
unsigned int HayDatos();
void CerrarPuerto();

Publicables

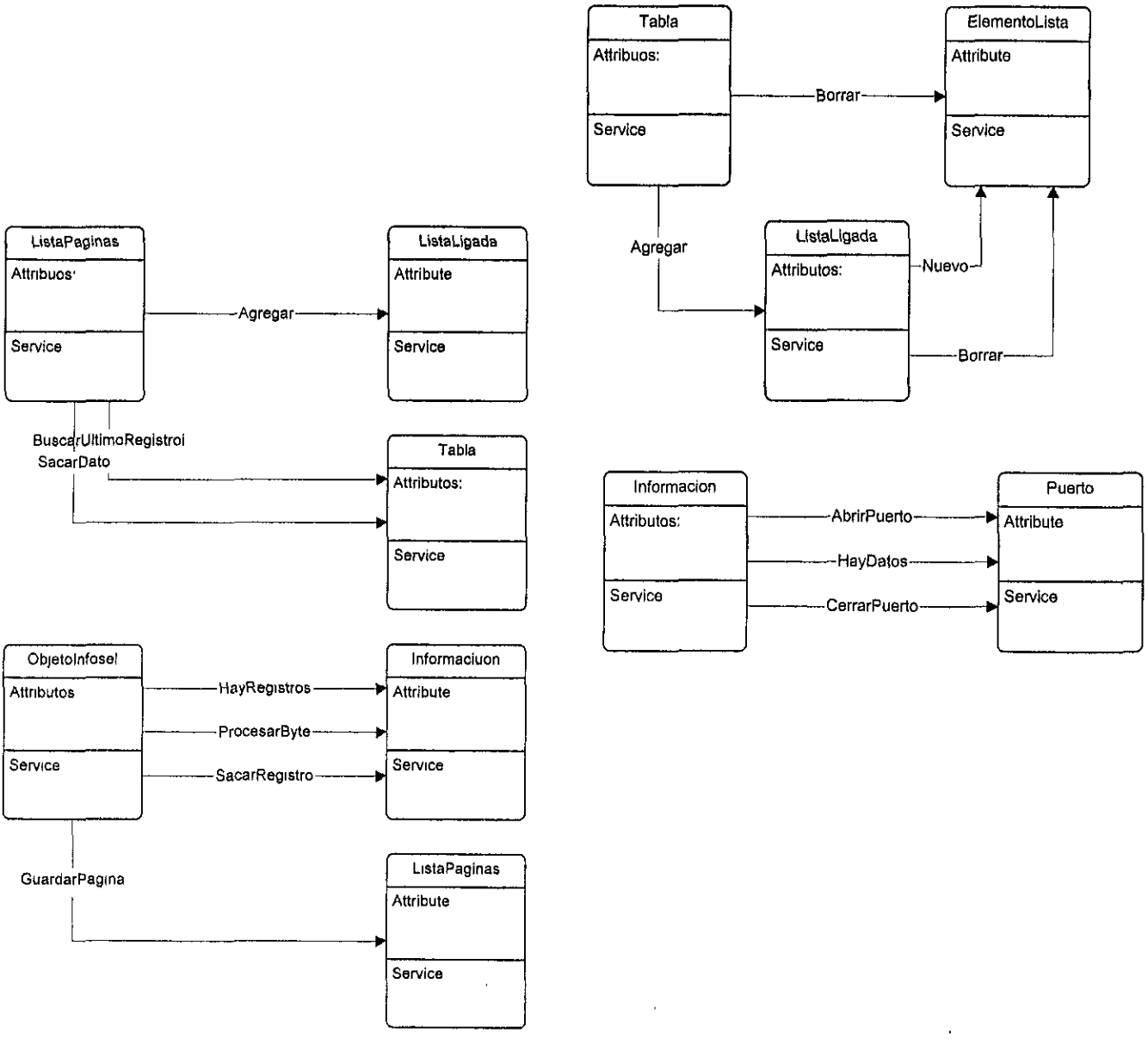
Atributos:
ListaLigada lista;

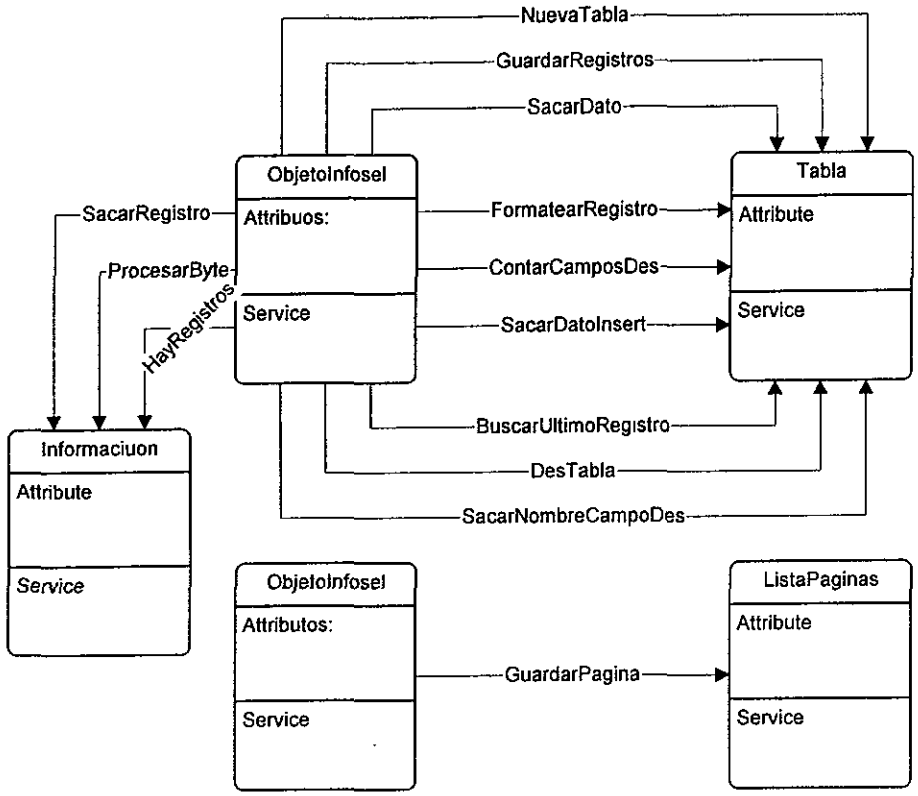
Servicios:
void AgregarPublicable(unsigned long num,char *nombre);
void QuitarPublicable(char *nombre);
void SacarNombre(char *reg,char *salida);
unsigned long EsPublicable(char *nombre);

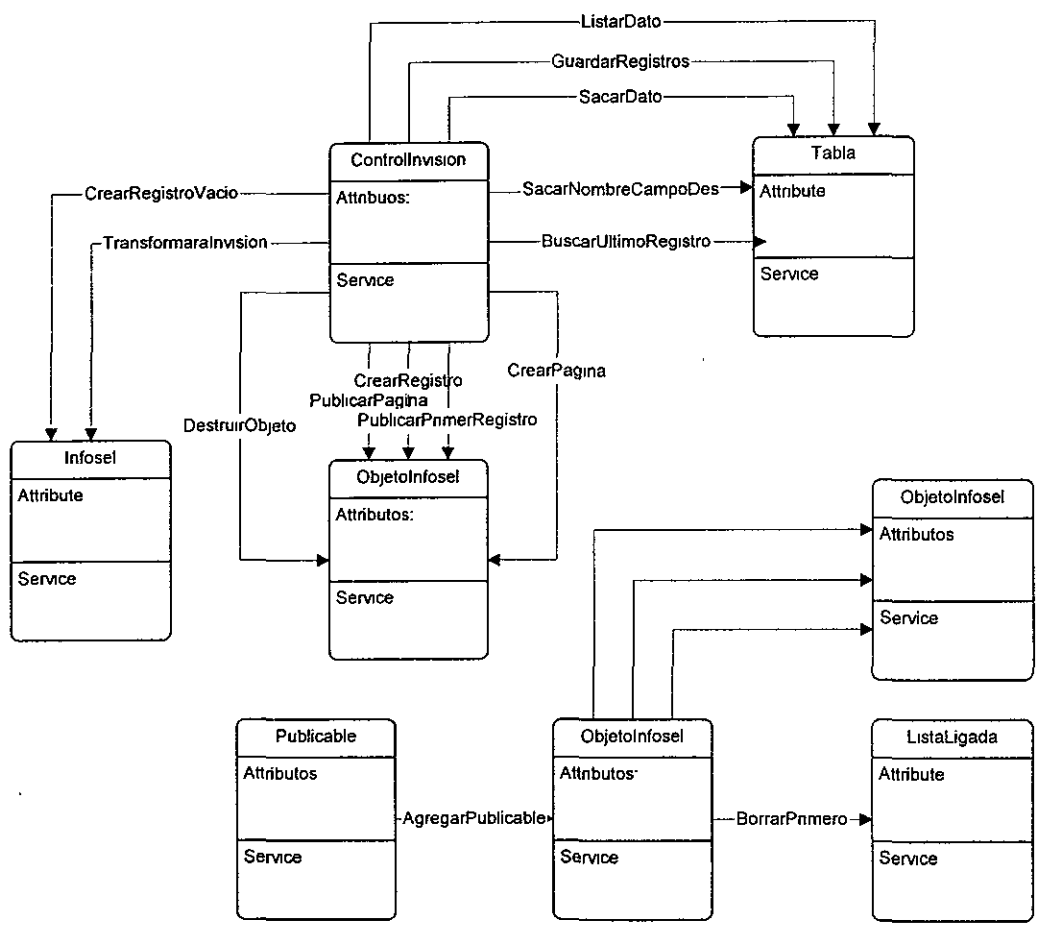
3.1.2.3. Jerarquía de los Objetos.



3.1.2.4. Relación y Mensajes entre los Objetos.




NuevaTabla
GuardarRegistros
SacarDato
ObjetoInfosel
Attribuos:
Service
Tabla
Attribute
Service
SacarRegistro
ProcesarByte
HayRegistros
FormatearRegistro
ContarCamposDes
SacarDatoInsert
BuscarUltimoRegistro
DesTabla
SacarNombreCampoDes
Informaciuon
Attribute
Service
ObjetoInfosel
Attributos:
Service
GuardarPagina
ListaPaginas
Attribute
Service


ListarDato
GuardarRegistros
SacarDato
ControlInvision
Attnbuos:
Service
Tabla
Attnbute
Service
CrearRegistroVacio
TransformaraInvision
SacarNombreCampoDes
BuscarUltimoRegistro
CrearRegistro
PublicarPagina
PublicarPnmerRegistro
CrearPagina
DestruirObjeto
Infosel
Attribute
Service
ObjetoInfosel
Attributos:
Service
ObjetoInfosel
Attributos
Service
Publicable
Attributos
Service
AgregarPublicable
ObjetoInfosel
Attnbutos
Service
BorrarPnmero
ListaLigada
Attribute
Service

3.1.3. Modelo Dinámico.

Un sistema puede ser comprendido de mejor manera si primero se examina como una estructura estática, la estructura de objetos y sus relaciones con otros en un instante de tiempo. Cuando se examinan los cambios en los objetos y sus relaciones a través del tiempo se establece lo que se conoce como modelo dinámico. El control es un aspecto del sistema que describe la secuencia de operaciones que ocurren como respuesta a un estímulo externo, sin considerar que hacen las operaciones, que manejan o como son implementadas.

3.1.3.1. Eventos y Estados.

Un modelo de objetos describe los posibles patrones de objetos, atributos y ligas que pueden existir en un sistema. Los valores de atributos y ligas retenidas por un objeto son llamadas *Estados*. A través del tiempo, los objetos son estimulados por diferentes valores, resultando una serie de cambios en sus estados. Un estimulo individual de un objeto a otro es un *Evento*. La respuesta a un evento depende del estado en que se encuentre el objeto que lo esta recibiendo, y puede incluir un cambio de estado o un envío de cualquier evento al objeto original o a un tercer objeto. Los patrones de eventos, estados y transiciones de estado pueden originar clases que son abstraídas y representadas como un diagrama de estados. Un *Diagrama de Estados* es una red de estados y eventos. Es como un diagrama de objetos que esta en una red de clases y relaciones. El modelo dinámico consiste de múltiples diagramas de estados, para cada clase con un importante comportamiento dinámico, y muestra el patrón de actividad para un sistema completo.

3.1.3.2. Eventos.

Un evento es algo que sucede en un punto determinado de tiempo y que no tiene duración. Por supuesto, nada es realmente instantáneo, un evento es simplemente

una ocurrencia que es rápida comparada con la precisión de la escala de tiempo de una abstracción.

Un evento puede lógicamente preceder o seguir a otro, o los dos eventos pueden no estar relacionados. Los eventos pueden ser concurrentes cuando estos no influyen en otros. En el modelado de un sistema no se puede tratar de establecer un orden entre eventos concurrentes ya que estos pueden ocurrir en cualquier orden. Cualquier sistema realístico de un sistema de distribución debe incluir eventos concurrentes y actividades.

Un evento es una forma de transmisión de información de un objeto a otro. Esto no es como una subrutina que es llamada y regresa un valor. En el mundo real, todos los objetos existen concurrentemente. Un objeto envía un evento a otro objeto que puede esperar una respuesta, pero la respuesta es un evento separado bajo el control del segundo objeto, el cual puede o no enviarlo.

Cada evento es una ocurrencia única, pero son agrupados dentro de clases de eventos que dan un nombre a la clase del evento para indicar estructuras comunes y comportamiento. Esta estructura es jerárquica.

Algunas clases de eventos pueden transmitir simples señales de que algo a ocurrido, mientras que otras clases de eventos pueden llevar valores de información retenida por un objeto. Los atributos son mostrados dentro de paréntesis después de cada nombre de la clase de eventos.

El término evento es a menudo usado ambiguamente. Algunas veces cada evento se refiere a una instancia evento, y en otras ocasiones a una clase evento. En la práctica esta ambigüedad usualmente no es un problema y el significado preciso es aparentemente de contexto. Los eventos incluyen también condiciones de error como ocurrencias normales.

3.1.3.3. Escenarios y Trazo de Eventos.

Un escenario es una secuencia de eventos que ocurren durante un ejecución particular de un sistema. La vista de un escenario puede variar; este puede incluir todos los eventos dentro del sistema, o puede incluir sólo algunos eventos generados por ciertos objetos dentro del sistema.

El usuario está listo para realizar la llamada
Pulsa tono de marcación
El usuario pulsa un 5
El tono de marcación termina
El usuario pulsa un 5
El usuario pulsa un 5
El usuario pulsa un 1
El usuario pulsa un 2
El usuario pulsa un 3
El usuario pulsa un 4
El timbre del teléfono comienza a sonar
El tono de llamada aparece en el teléfono que realiza la llamada
El usuario que recibe la llamada contesta
El teléfono deja de sonar
El tono de llamada desaparece en el teléfono que llama
Los teléfonos están comunicados
El usuario que llamó cuelga el teléfono
Los teléfonos se desconectan
El usuario que recibió la llamada cuelga

Figura 3.1.8. Escenario para una llamada telefónica.

Cada evento transmite información de un objeto a otro. El siguiente paso después de escribir un escenario es identificar los objetos transmisores y receptores de cada evento. La secuencia de eventos y los objetos intercambiando eventos pueden ser mostrados en un escenario aumentado llamado *Diagrama de Trazos de Eventos*. Este diagrama muestra cada objeto como una línea vertical y cada evento como una flecha horizontal que va del objeto emisor al objeto receptor. El tiempo se incrementa

desde la parte alta hasta la parte baja, pero el espaciado es irrelevante; es sólo la secuencia de eventos que son mostrados, no su tiempo exacto.

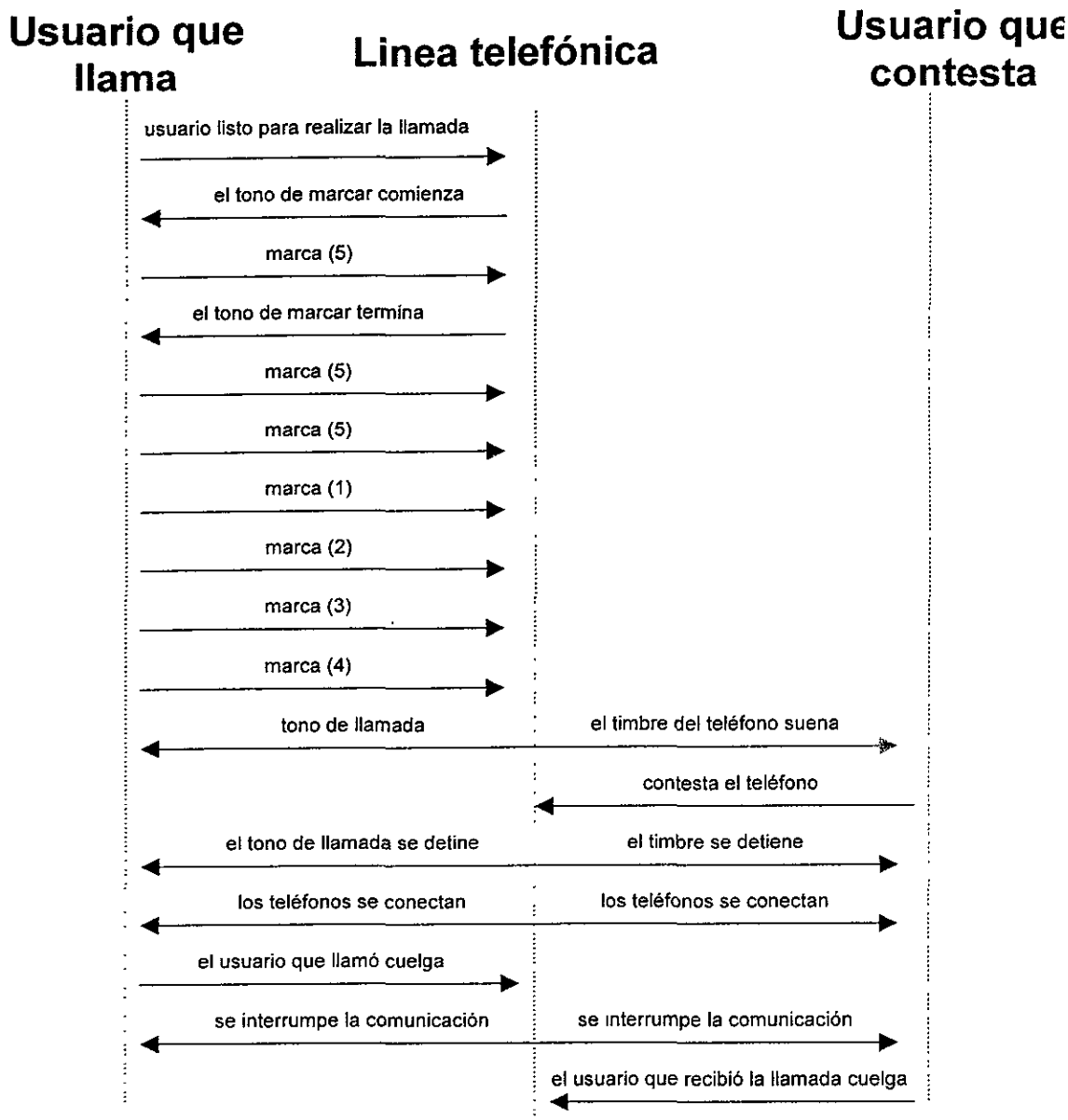


Figura 3.1.9. Trazo de eventos para una llamada telefónica.

3.1.3.4. Estados.

Un estado es una abstracción de un valor del atributo y ligas de un objeto. El conjunto de valores son agrupados dentro de un estado acordado para propósitos que afectan el grueso del funcionamiento del objeto. Un estado especifica la respuesta de un objeto a eventos entrantes. La respuesta a la recepción de un evento por un objeto puede variar cuantitativamente dependiendo de valores exactos de estos atributos, pero la respuesta es cualitativamente la misma para todos los valores dentro de el mismo estado y puede ser cualitativamente diferente para valores dentro de diferentes estados. La respuesta de un objeto a un evento puede incluir una acción o cambio de estado para el objeto.

Un estado corresponde al intervalo entre dos eventos recibidos por un objeto. Los eventos representan puntos en el tiempo; los estados representan intervalos de tiempo. El estado de un objeto depende del paso de la secuencia de eventos que ha recibido, pero en la mayoría de los casos el paso de eventos es eventualmente ocultado por los eventos subsecuentes.

Un estado tiene duración, ya que este ocupa un intervalo de tiempo. Un estado es a menudo asociado con una actividad continua. Los eventos y estados son complemento uno del otro; un evento separa dos estados, y un estado separa dos eventos. Un estado también es asociado con un valor de un objeto que satisface la misma condición; en el simple caso, de que cada valor enumerado de un atributo defina a un estado separado. Dentro de la definición de estado, se ignoran los atributos que no afectan el funcionamiento de un objeto, y se deben unir en un simple estado todas la combinaciones de valores de atributos y ligas que tienen la misma respuesta para los eventos. Por supuesto, cada atributo tiene algún efecto significativo en el funcionamiento, pero regularmente algunos atributos no afectan el patrón de control y pueden ser considerados como un simple valor de parámetro que esta dentro de un estado dado.

Estado: Sonido de Alarma

Descripción: La alarma se enciende en la pantalla y suena para indicar una hora programada.

Secuencia de eventos que produce el estado:

programación de la alarma (hora programada)

cualquier secuencia no incluida *borra alarma*

hora actual = *hora programada*

Condiciones que caracterizan al Estado:

alarma = encendida, hora programa =< hora actual =< hora programada + 20 seg, y ningún botón ha sido pulsado desde la hora programada

Eventos aceptados en el Estado:

Evento	Acción	Siguiente Estado
hora actual = hora programada + 20	inicializa alarma	normal
botón pulsado (cualquier botón)	inicializa alarma	normal

Figura 3.1.10. Caracterizaciones de un Estado.

3.1.3.5. Diagramas de Estado.

Un diagrama de estados relaciona eventos y estados. Cuando un evento es recibido, el siguiente estado depende del estado presente, así como el evento; un cambio de estado causado por un evento es denominado *Transición*. Un diagrama de estados es una gráfica en la cual los nodos son estados y los arcos direccionados son las transiciones etiquetadas por los nombres de eventos. Un estado se representa por medio de una caja redondeada que contiene opcionalmente un nombre. Una transición se representa como una flecha desde el estado destino hasta el estado origen; la etiqueta de la flecha es el nombre del evento que causa la transición. Todas las transiciones abandonan un estado que debe corresponder a los diferentes eventos.

El diagrama de estados especifica la secuencia del estado causada por una secuencia de eventos. Si un objeto esta dentro de un estado y un evento identifica la ocurrencia

de una de estas transiciones, el objeto ingresa el estado del destino final de la transición. Si mas de una transición abandona un estado, entonces el primer evento que ocurra provocará que la transición correspondiente se dispare. Si un evento no ha abandonado la transición del presente estado, provoca que el evento sea ignorado. Una secuencia de eventos representa una ruta a través de la gráfica.

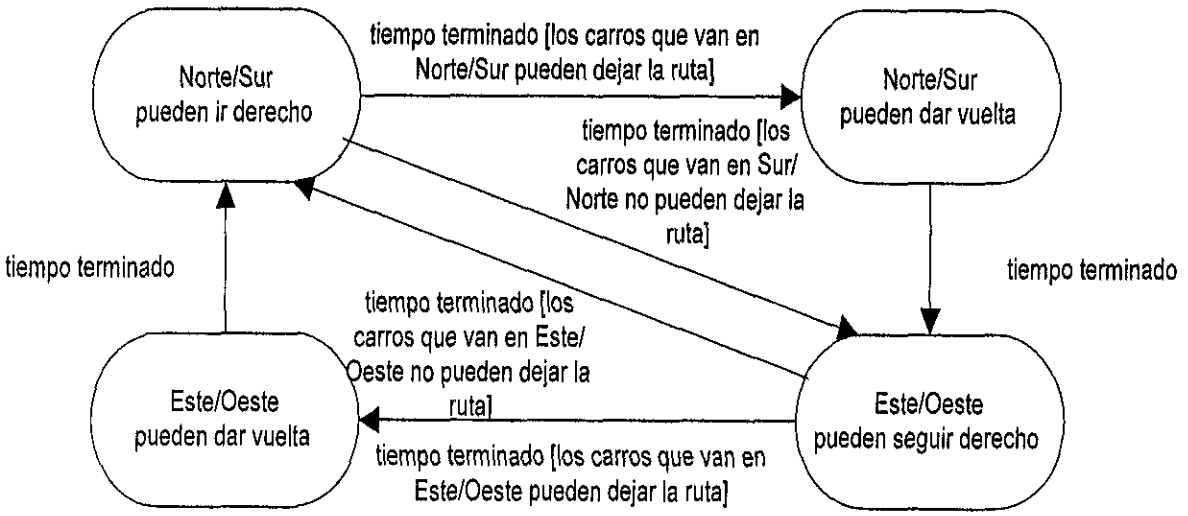


Figura 3.1.11. Diagrama de estado con transiciones.

3.1.3.6. Control de Operaciones.

El diagrama de estados, también describe el patrón de eventos y estados para una simple clase de objetos.

El diagrama de estados puede ser de poco uso si sólo se describen los patrones de eventos. Una descripción del funcionamiento de un objeto debe especificar que hace el objeto en respuesta a los eventos. Las operaciones asociadas a los estados o transiciones son diseñadas en respuesta a los estados o eventos correspondientes.

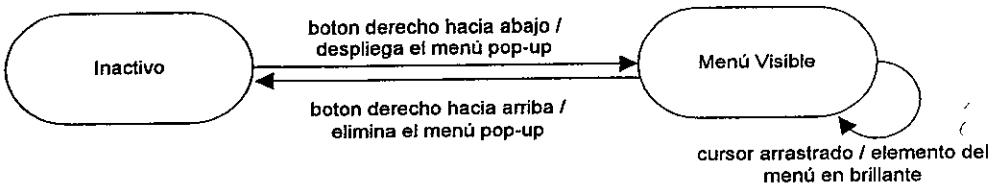


Figura 3.1.12. Acciones para un menú de ventana.

Una *Actividad* es una operación que lleva tiempo completar, y esta asociada con un estado. Las actividades incluyen operaciones continuas, que se pueden ver como operaciones secuenciales que terminan por si solas después de un intervalo de tiempo, las cuales se cierran como una compuerta. Un estado puede controlar actividades continuas que persisten hasta que un evento lo termina por causa de la transición de un estado. La notación "do: A" (**Efectúa: A**) dentro de una caja de estado indica que la actividad A comenzará a la entrada del estado y se detendrá a la salida. Un estado puede además controlar una actividad secuencial, que avanzará hasta que esta se complete o hasta que sea interrumpida por un evento que ha terminado prematuramente. La misma notación "do: A" indica que la actividad secuencial A comienza al inicio del estado y se detiene cuando se ha completado. Si un evento causa una transición de un estado antes de que la actividad sea terminada, entonces significa que la actividad término prematuramente. Los dos usos no son realmente diferentes; una actividad continua puede ser vista como una actividad secuencial que dura indefinidamente.

Una acción es una operación instantánea y que esta asociada con un evento. Una acción representa una operación cuya duración es insignificante comparada con la resolución de el diagrama de estado. En el mundo real las operaciones no son instantáneas, pero al modelarlas como una acción nos permiten olvidarnos de la estructura interna para propósitos de control dentro de una implementación. Una acción puede además representar operaciones internas de control, las cuales son como un conjunto de atributos o generadores de otros eventos.

La notación para una acción sobre una transición es un slash ('/') (**diagonal invertida**) y el nombre (o descripción) de la acción, seguida del nombre del evento que la causa.

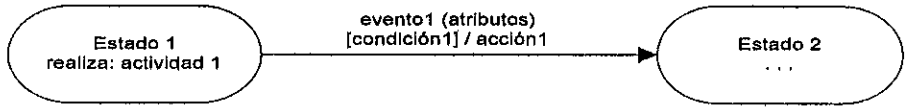


Figura 3.1.13. Notación para una estructura de diagramas de estado.

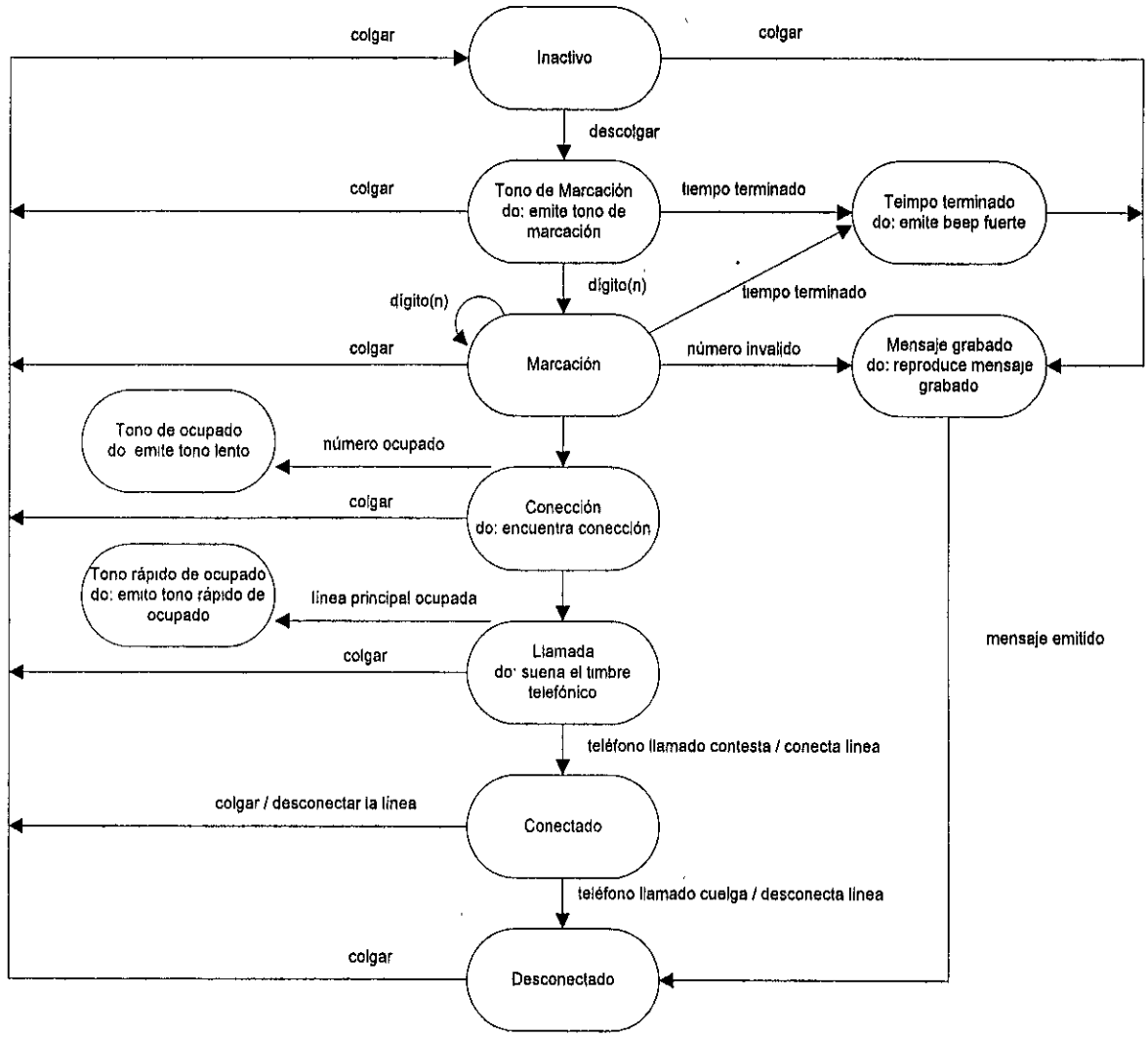


Figura 3.1.14. Diagrama de estados para una línea telefónica.

3.1.3.7. Diagrama de Estados Anidados.

Los diagramas de estados pueden ser estructurados para permitir descripciones concisas de sistemas complejos. La manera de estructurar estados es similar a la manera de *estructurar objetos*: generalización y agregación. La generalización es equivalente a desglosar actividades anidadas. Esto significa que una actividad debe ser descrita a un alto nivel, y entonces expandir a bajo nivel por medio del detalle, similar a las llamadas de procedimientos anidados. En suma, una generalización involucra estados y eventos que deberán ser contemplados dentro de la generalización jerárquica con la inherencia de estructuras y funciones comunes, similar a la inherencia de atributos y operaciones dentro de las clases. Una agregación involucra estados que serán fragmentados en componentes ortogonales, con acción limitada entre ellos, similar a una agregación jerárquica de objetos. Una agregación es equivalente a la concurrencia de estados. La concurrencia de estados generalmente corresponde a la agregación de objetos, *posiblemente un sistema entrante*, que tiene partes que interactúan. Una actividad en un estado puede ser expandida como un diagrama de estado a bajo nivel, donde cada estado representa un paso de la actividad. Las actividades anidadas son diagramas de estado con transiciones entrantes y salientes, similar a las subrutinas. El conjunto de diagramas de estado anidados forman una malla.

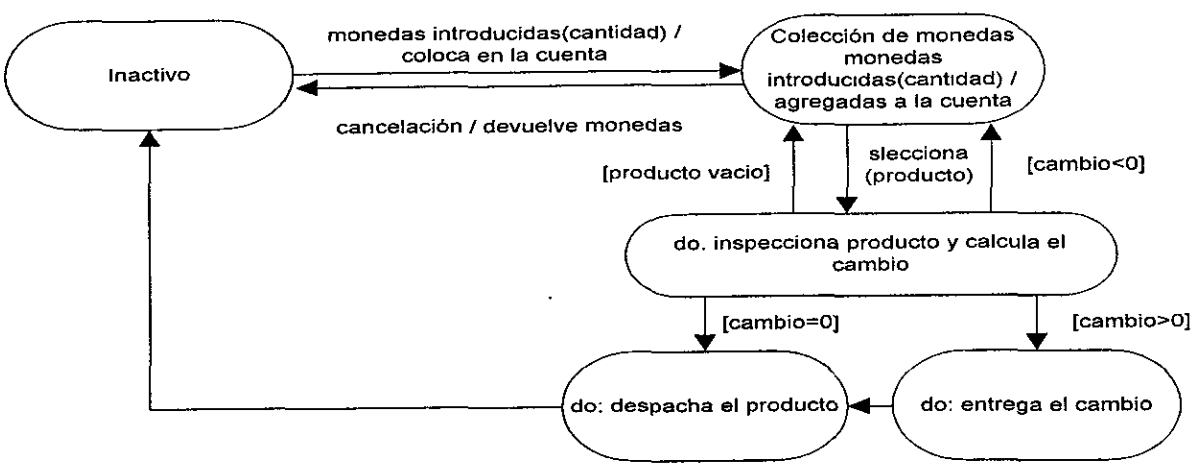


Figura 3.1.15. Modelo de una máquina despachadora.

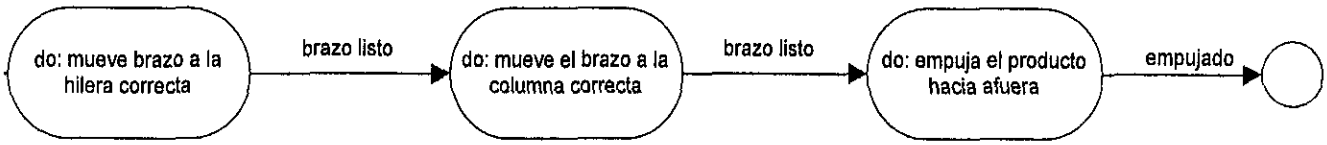


Figura 3.1.16. Actividad para entrega de productos.

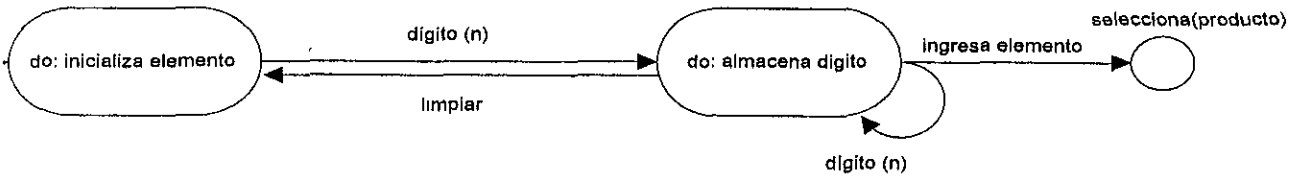


Figura 3.1.17. Transición para la selección de productos.

3.1.3.8. Generalización de Estados.

Un diagrama de estados anidados es actualmente una forma de generalizar los estados. La generalización es una relación disyuntiva. Un objeto dentro de un estado en un diagrama de alto nivel debe ser exactamente un solo estado dentro de un diagrama anidado. Debe estar en el primer estado, o en el segundo, o dentro de ambos. Los estados en el diagrama anidado son todos los refinamientos de un estado en el diagrama de alto nivel y pueden interactuar con otros estados.

Los estados pueden tener subestados que son inherentes a las transiciones de sus superestados, como clases que tienen subclases que adquieren los atributos y operaciones de su superclase. Cualquier transición o acción que aplique para un estado debe aplicar para todos lo subestados, a menos que sea sobreentendida por una transición equivalente en un subestado.

La notación de la generalización para estados es diferente de la que se usa para las clases, por causa del gran número de líneas que pueden confundirse con transiciones. Un superestado es un dibujo como de una caja redondeada que engloba todos sus subestados. Los subestados pueden contener a su vez otra caja redondeada que englobe más subestados, ya que la caja redondeada representa la variedad de estados que están anidados, y esto se conoce como contour (**contorno**).

Es posible representar situaciones mas complicadas, como una transición explícita de un subestado a un estado fuera del contour, o una transición explícita dentro del contour. En cualquier caso, todos los estados deben aparecer en un diagrama usando la notación contour. En casos simples donde no hay interacción excepto por la iniciación y la terminación, los estados anidados pueden dibujarse simplemente como diagramas separados y referenciados por el nombre en una cláusula "do: A".

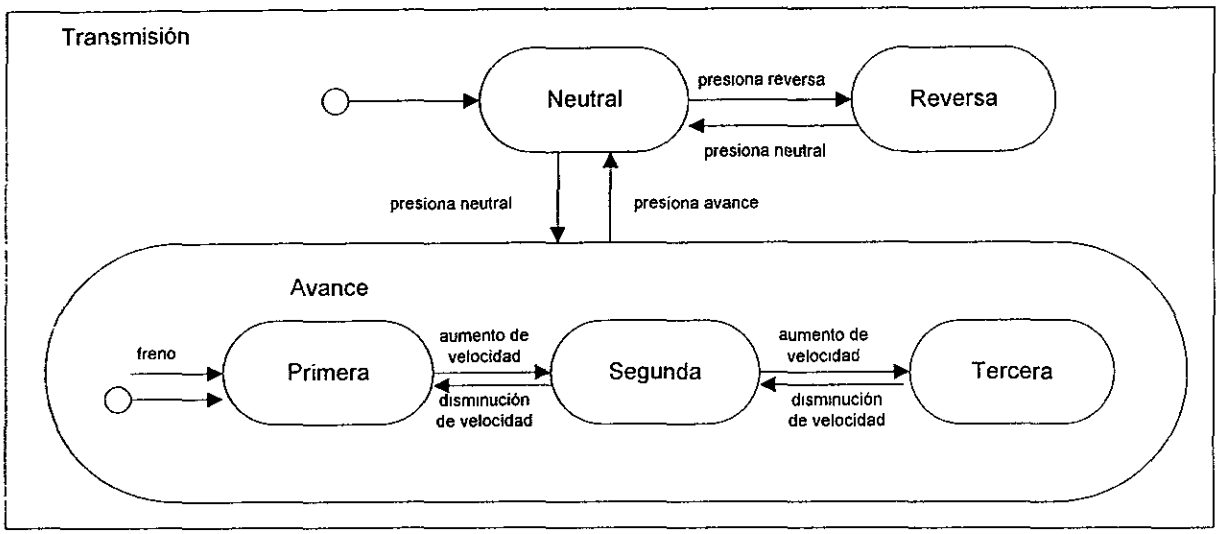


Figura 3.1.18. Generalización de un diagrama de estados para una transmisión de auto.

3.1.4.9. Generalización de Eventos.

Los eventos pueden estar organizados dentro de una generalización jerárquica con la inherencia de atributos de eventos. Esto puede ser representado como un diagrama de árbol, donde existe un evento raíz, que puede tener "n" ramas, las cuales a su vez se convierten en una nueva raíz que puede derivar a otras "n" ramas y así sucesivamente, hasta lograr desglosar completamente todos los eventos que integran la generalización jerárquica. La elaboración de eventos jerárquicos permite obtener diferentes niveles de la abstracción que serán usados en distintas partes dentro del modelo.

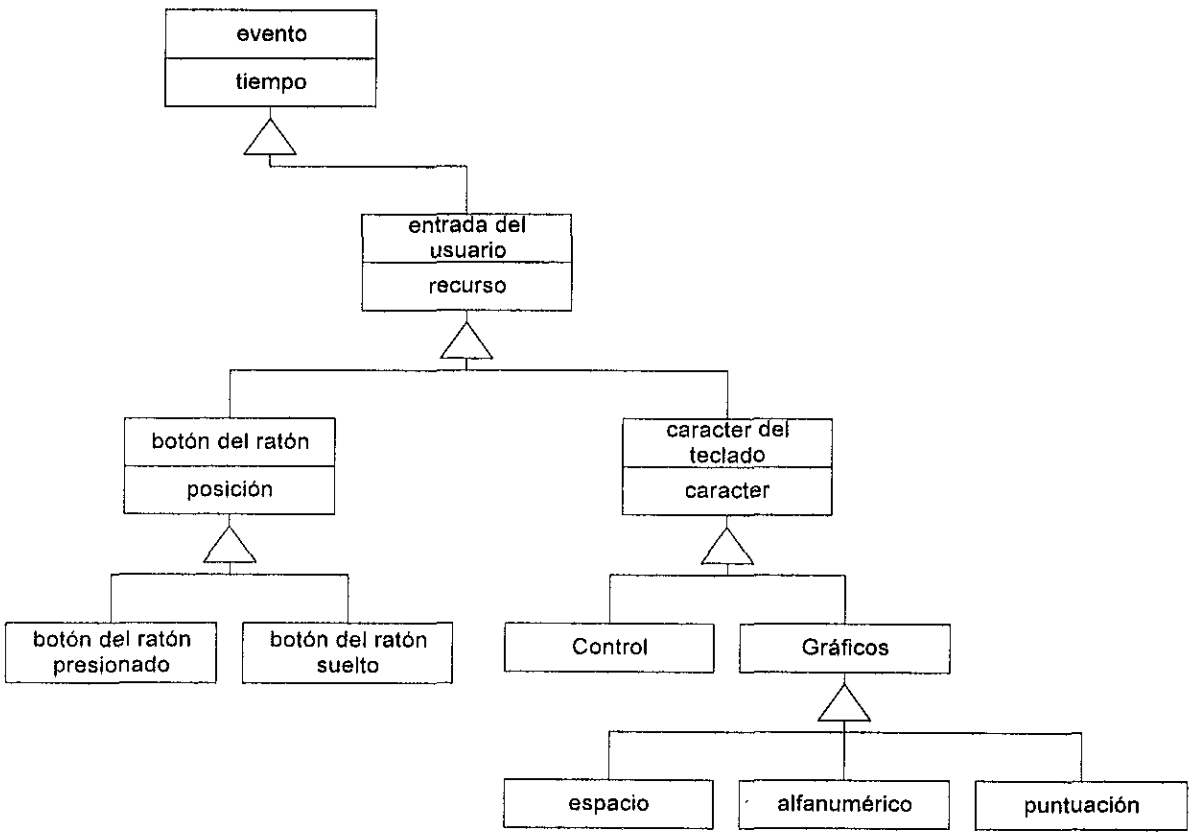


Figura 3.1.19. Eventos jerárquicos para un teclado.

3.1.3.10. *Concurrencia* de Agregación.

Un modelo dinámico describe un conjunto de objetos concurrentes, y con esto estados propios y un diagrama de estados. Los objetos dentro de un sistema son inherentemente concurrentes y pueden cambiar estados independientemente. El estado de un sistema entrante no puede ser representado por un simple estado dentro de un solo objeto; es el producto de estados de todos lo objetos en él. En muchos sistemas, el número de objetos puede cambiar dinámicamente con mucha facilidad.

Un diagrama de estado para una concurrencia es un conjunto de diagramas, uno para cada componente. La agregación implica concurrencia. El agregado de estados corresponde a la combinación de todos los componentes del diagrama. La agregación es relación conjuntiva. El agregado de estados es un estado del primer diagrama y un estado del segundo diagrama, o un estado de cualquier diagrama. El guardado de transiciones para un objeto puede depender de otro objeto que comienza en un estado dado. Esto permite la interacción entre diagramas de estado, mientras se conserva la modularidad (ver figura 3.1.20).

3.1.3.11. Concurrencia dentro de un objeto.

La concurrencia dentro de un estado en un simple objeto se eleva cuando el objeto puede ser particionado en subconjuntos de atributos o ligas, donde cada una de ellas puede tener su propio subdiagrama. El estado del objeto comprende un estado de cada subdiagrama. Los subdiagramas no necesitan ser *independientes: el mismo* evento puede causar transiciones en más de un subdiagrama. La concurrencia dentro de la composición de estados de un objeto esta reflejada en una porción de la composición de estados dentro de los subdiagramas mediante líneas punteadas. El nombre de la composición de estados global puede ser escrita en una región aparte, *mediante una caja,* y separada por una línea gruesa de los subdiagramas concurrentes.

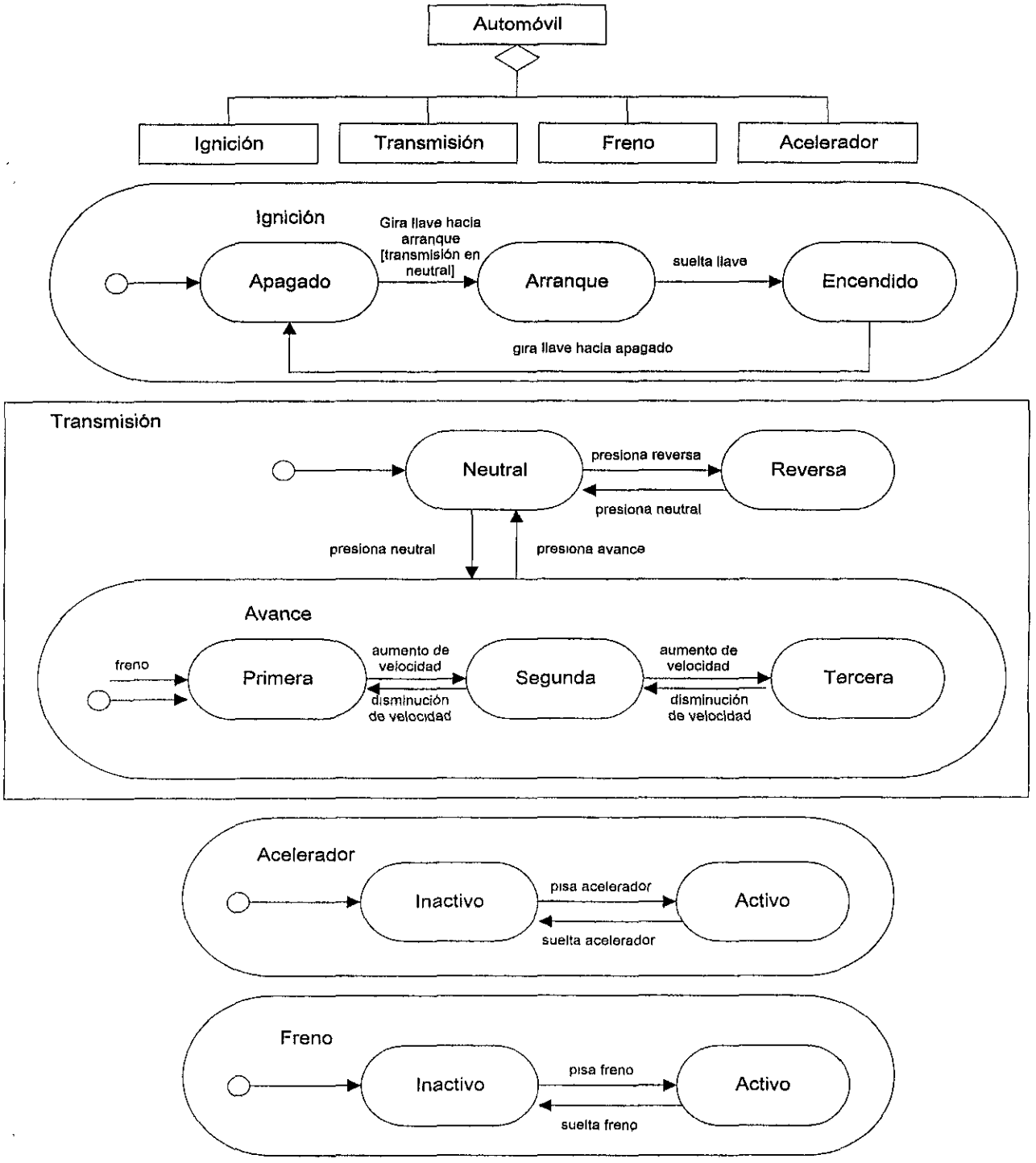


Figura 3.1.20. Agregación y diagramas de estado concurrentes.

3.1.3.12. Conclusiones del Modelo Dinámico.

El Modelo Dinámico representa el control de la información: la secuencia de eventos, estados y operaciones que se presentan en un sistema de objetos. Como el Modelo de Objetos, el Modelo Dinámico es un patrón que especifica los posibles escenarios que se pueden ocurrir.

Un evento es una señal de que algo a ocurrido. Un estado representa el intervalo entre eventos y especifica el contexto en el cual un evento es interpretado. Una transición entre estados representa la respuesta a un evento, incluyendo el siguiente estado, posibles acciones y eventos enviados a otros objetos. Una condición es una función Booleana que controla una transición que esta próxima a ocurrir. Un diagrama de estados es una gráfica de estados y transiciones etiquetadas por transiciones.

Una acción es una operación instantánea en respuesta a un evento. Una clase de acción está enviando un evento a otro objeto. Las acciones pueden ser adicionadas como entradas o salidas de un estado. Una actividad es una secuencia de acciones que lleva tiempo completar. Una actividad puede ser reflejada con un estado o un diagrama de estados. El resultado de una actividad puede ser usada como una decisión para realizar la transición al siguiente estado. Los subeventos toman las mismas transiciones que sus supereventos.

Los estados y eventos pueden ser ampliados en el diagrama de estado para mostrar un mayor detalle, además de que pueden ser organizados dentro de herencias jerárquicas. Los subestados son inherentes a las transiciones de sus estados.

Los objetos son concurrentemente inherentes. Cada objeto es una colección que tiene sus propios estados. Los diagramas muestran la concurrencia como un agregado de los estados concurrentes, para cada operación independiente. Los objetos concurrentes interactúan por intercambio de eventos y por las condiciones de prueba

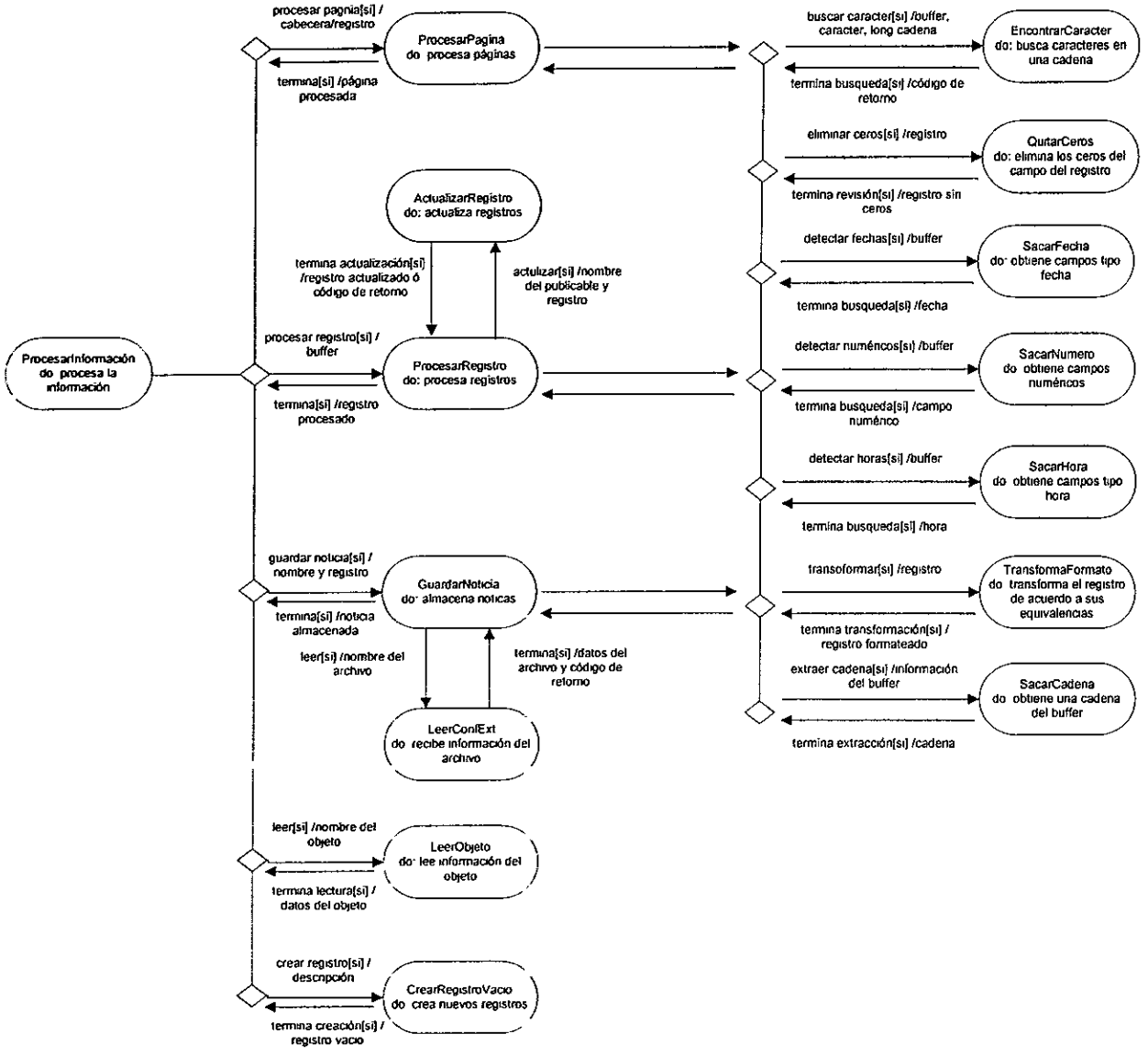
de otros objetos, incluyendo los estados. Las transiciones pueden separar o integrar el diagrama de control.

Las acciones de entrada y salida permiten acciones que pueden ser asociadas con un estado, para indicar todas las transiciones entrantes o salientes del estado. Estas poseen sus propios diagramas de estado posibles para ser usados en contextos múltiples. Las acciones internas representan transiciones que no abandonan el estado. Las transiciones automáticas se disparan cuando sus condiciones son satisfechas y cualquier actividad en el estado origen ha terminado.

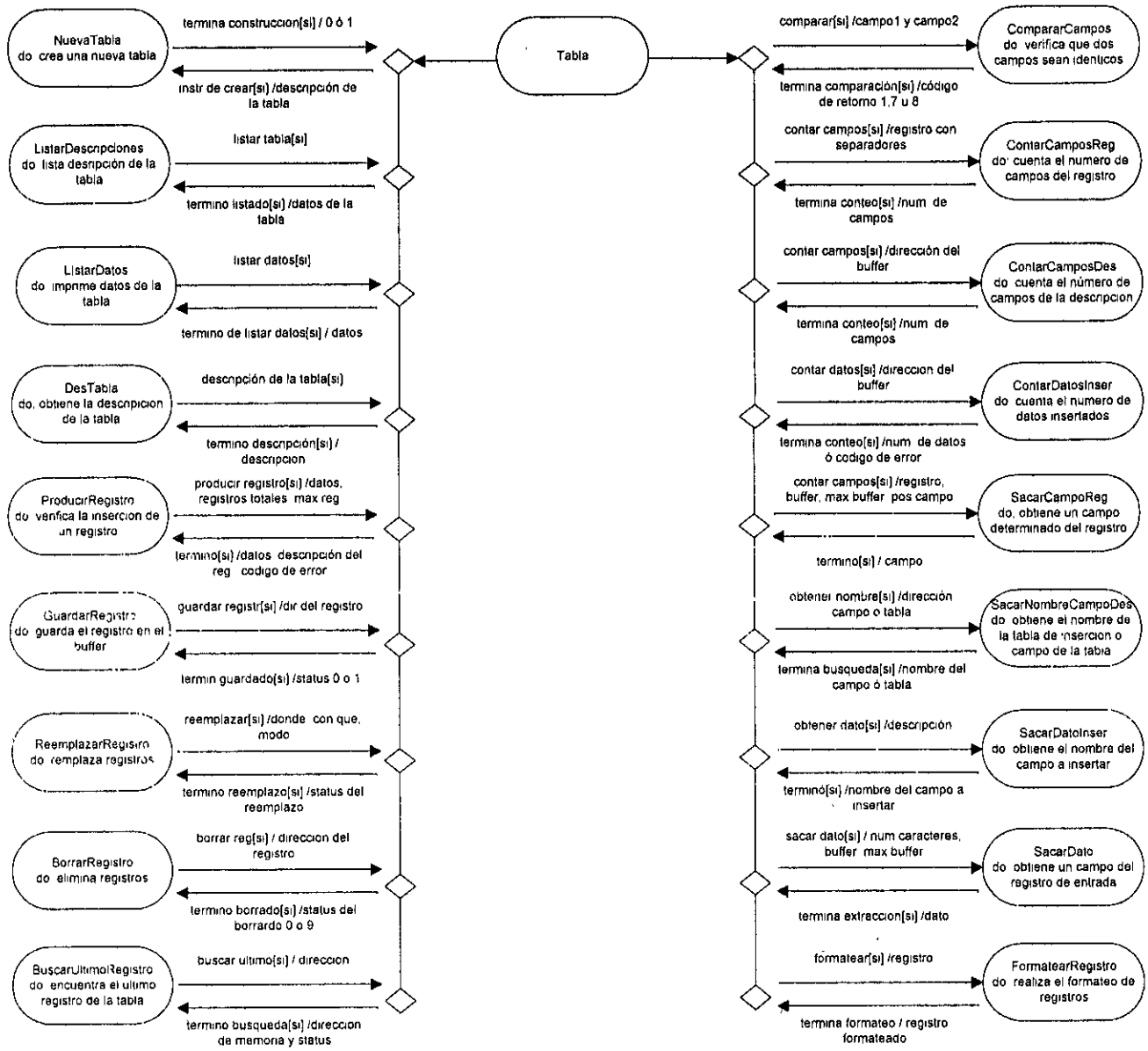
Los estados son realmente restricciones generalizadas dentro de una clase y son complementarias a la generalización de extensiones ordinarias. Una subclase inherente al diagrama de estado de este, puede ser concurrente con cualquier diagrama de estado que este definido. Esto es posible para encontrar un diagrama de estado inherente por medio de la expansión de estados dentro de subestados o subdiagramas concurrentes.

3.1.3.13. Diagramas del Modelo Dinámico para el Sistema Traductor de Protocolo.

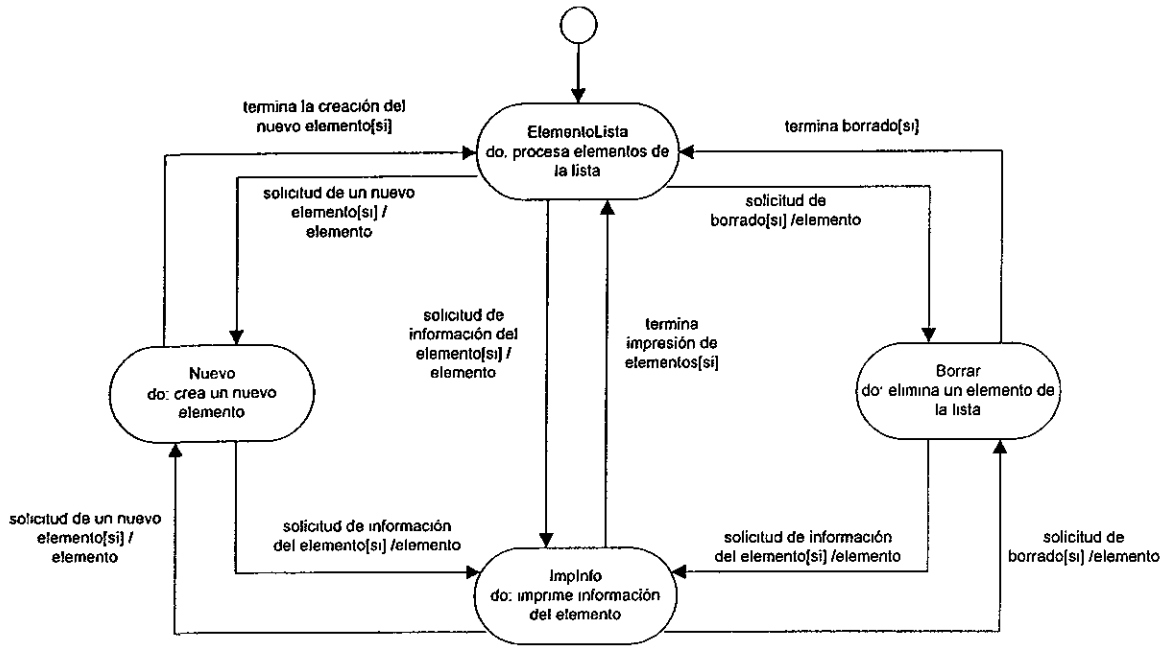
Objeto FuenteInfo



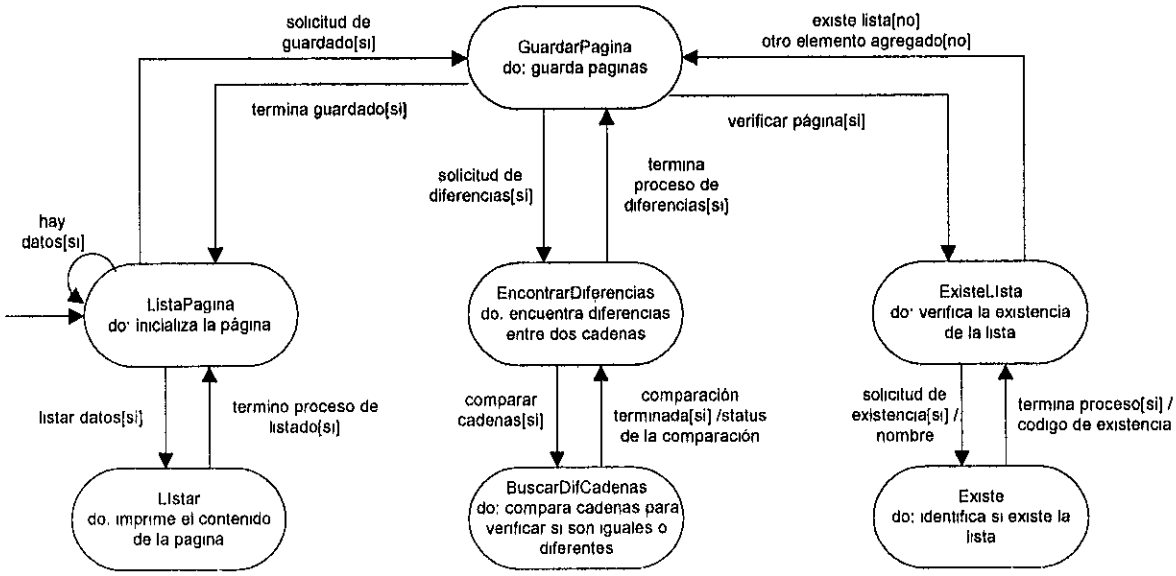
Objeto Tabla



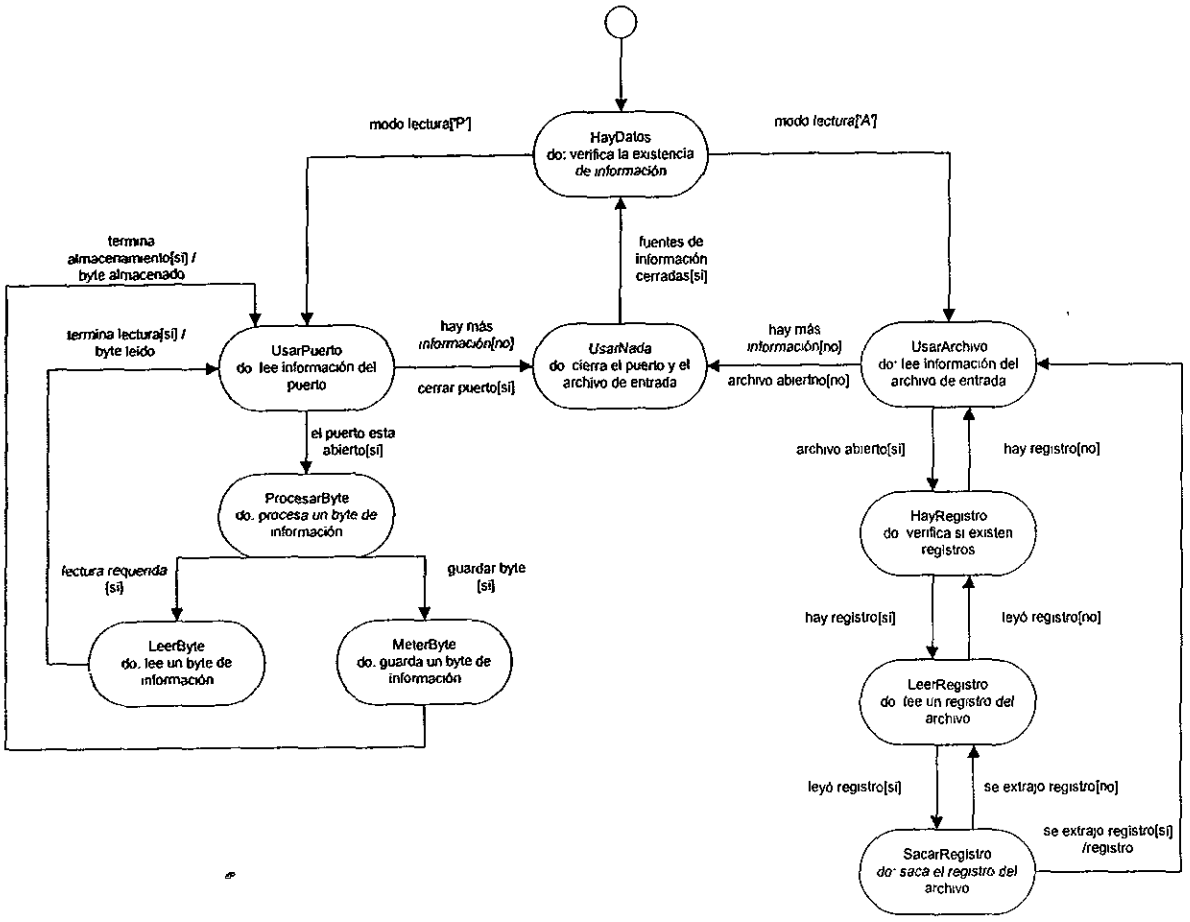
Objeto Elemento Lista



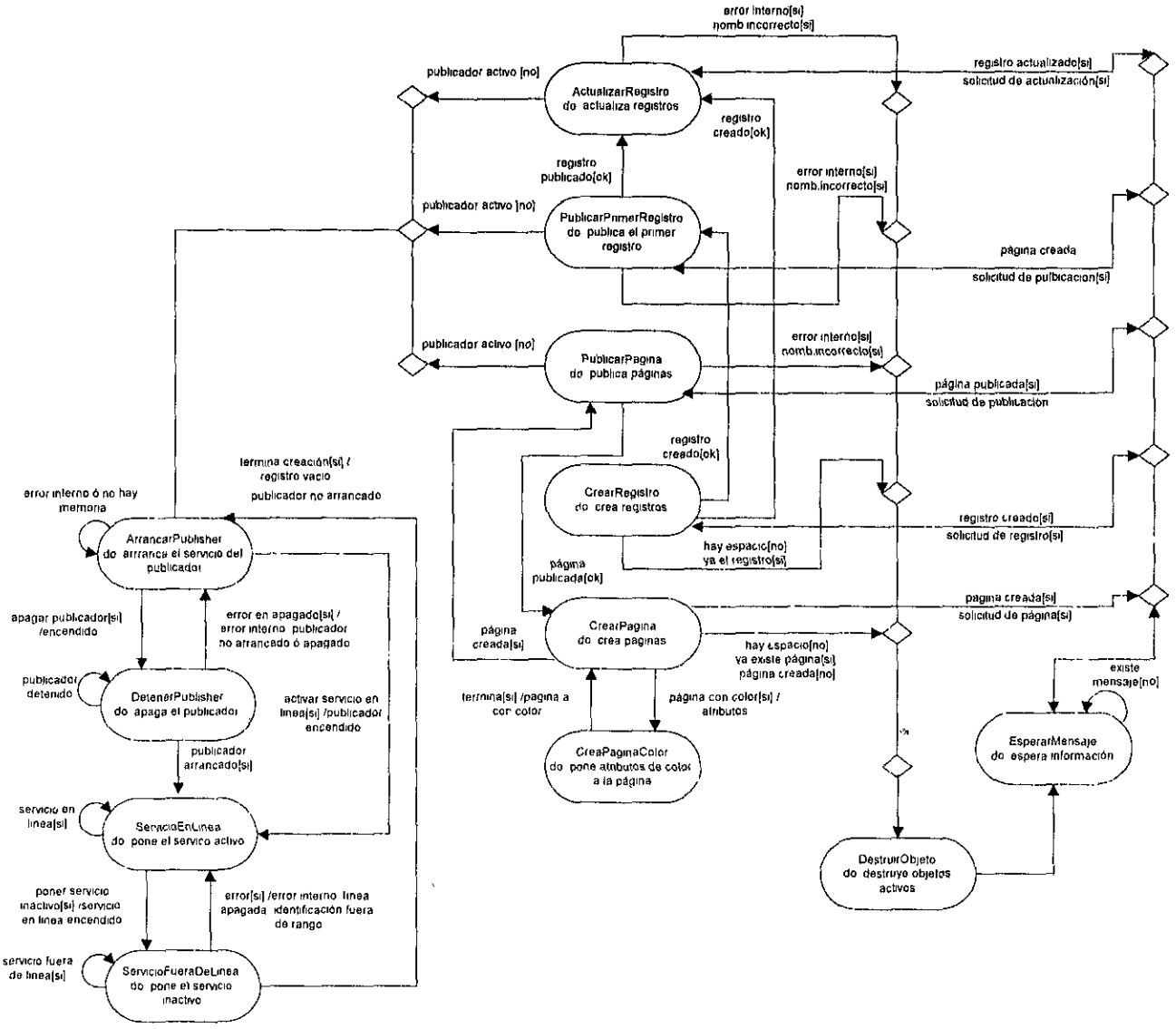
Objeto Lista Página



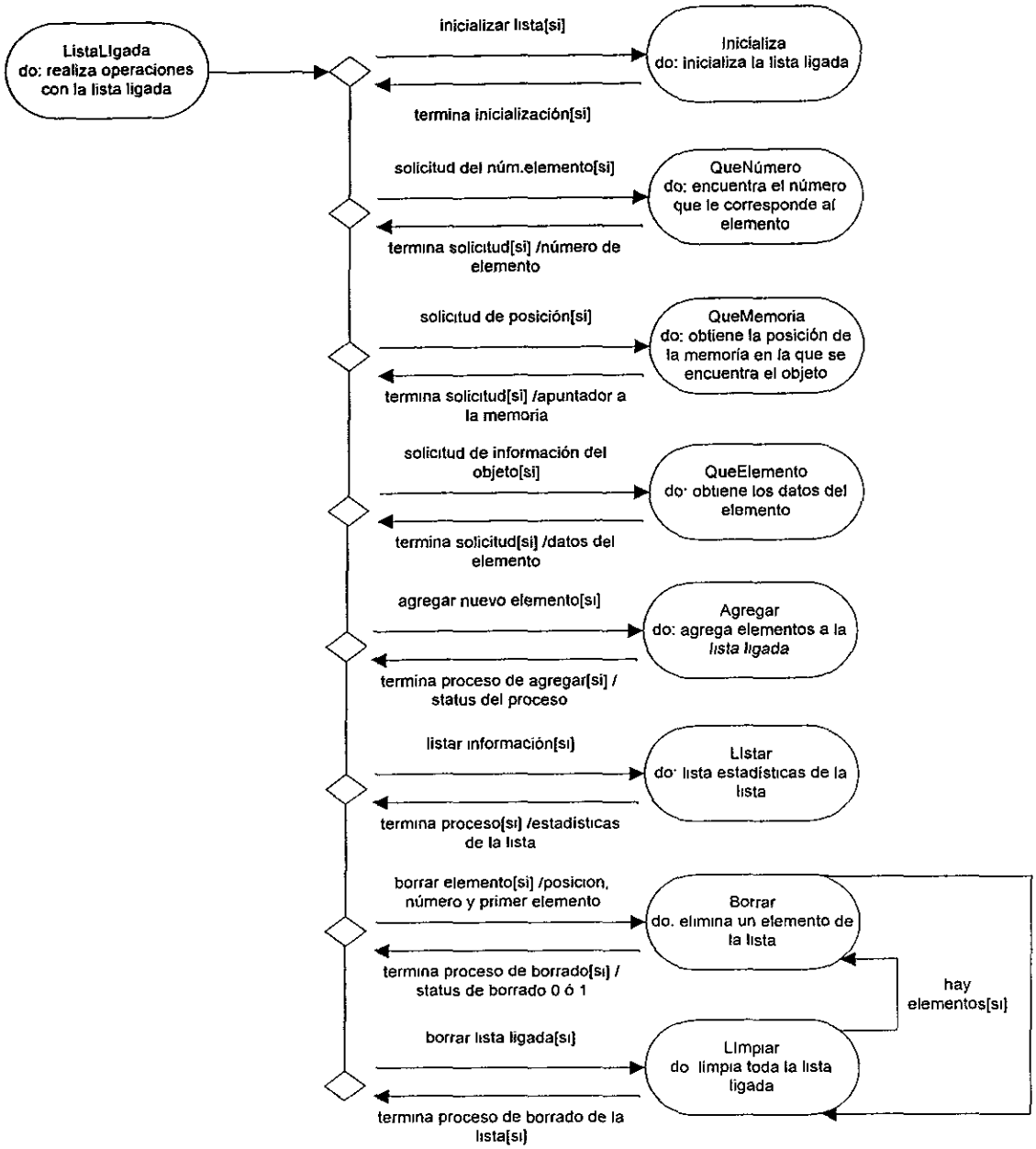
Objeto Información



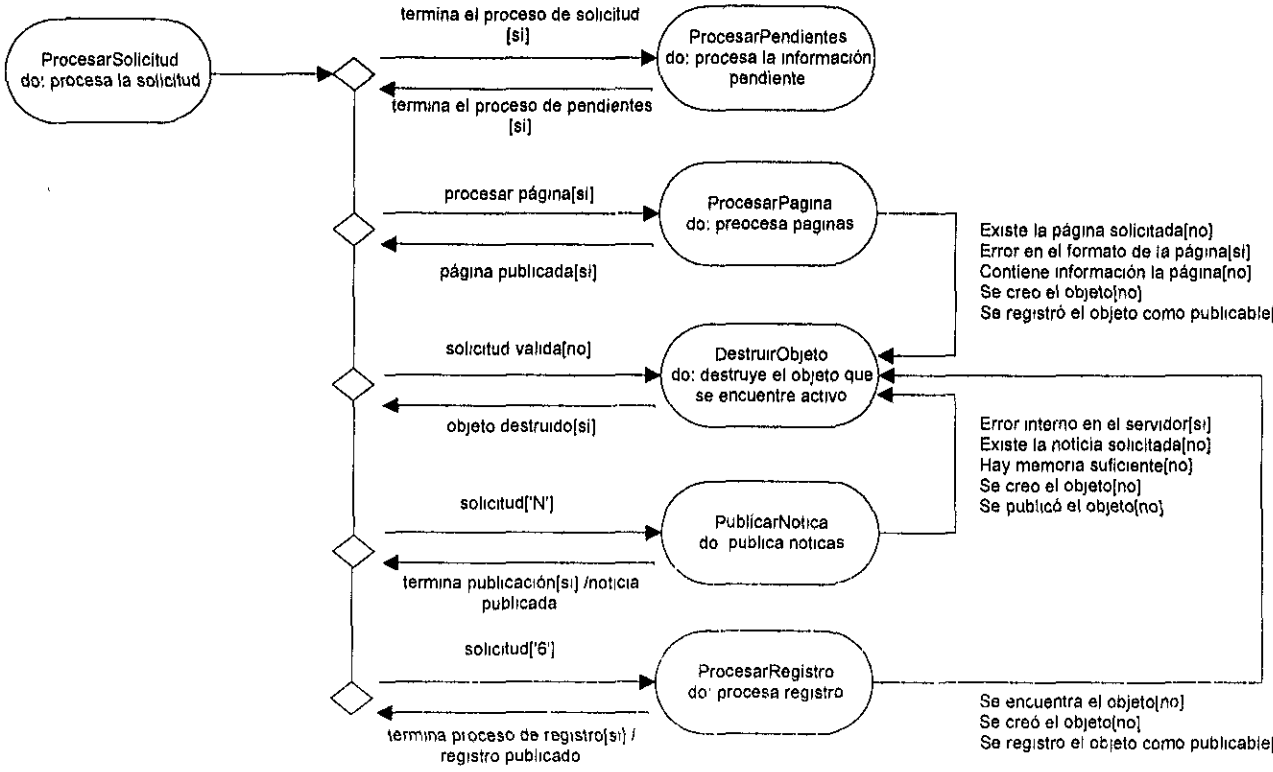
Objeto Plataforma Digital



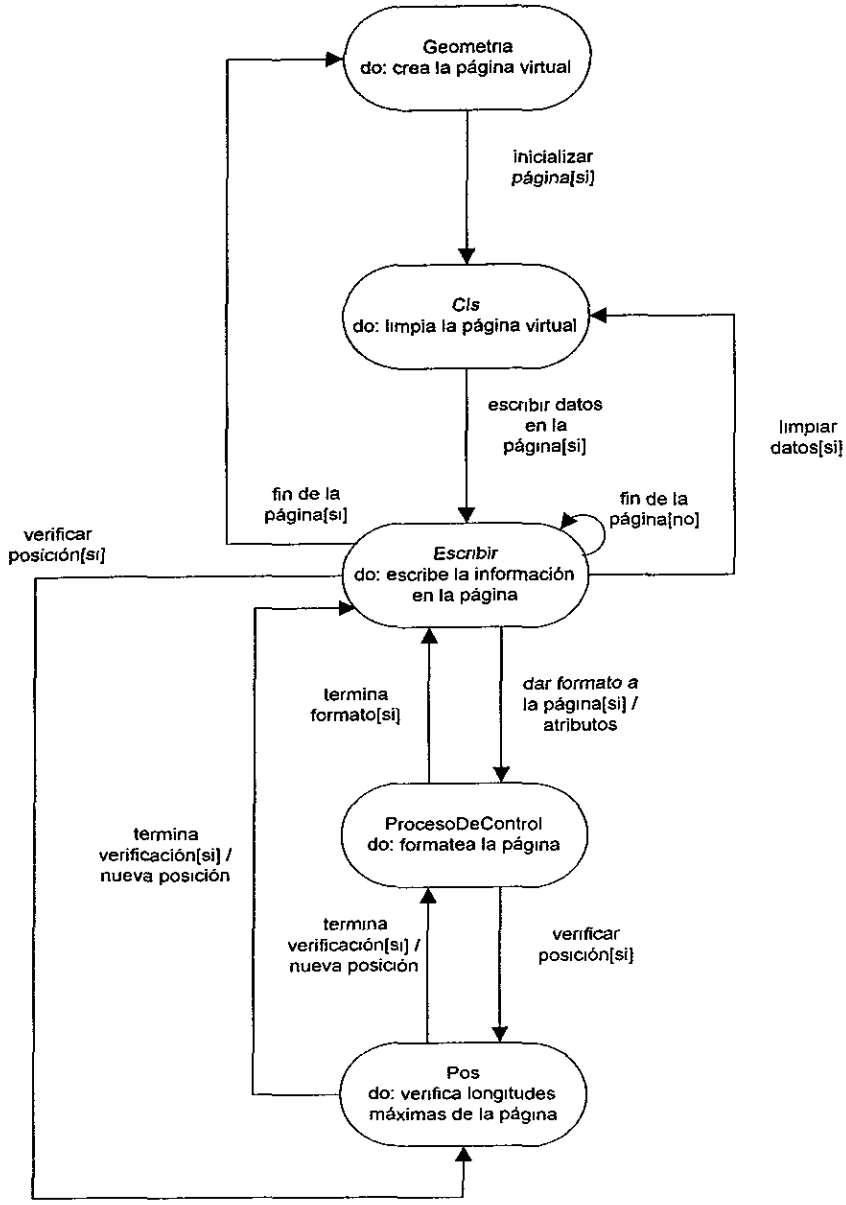
Objeto Lista Ligada



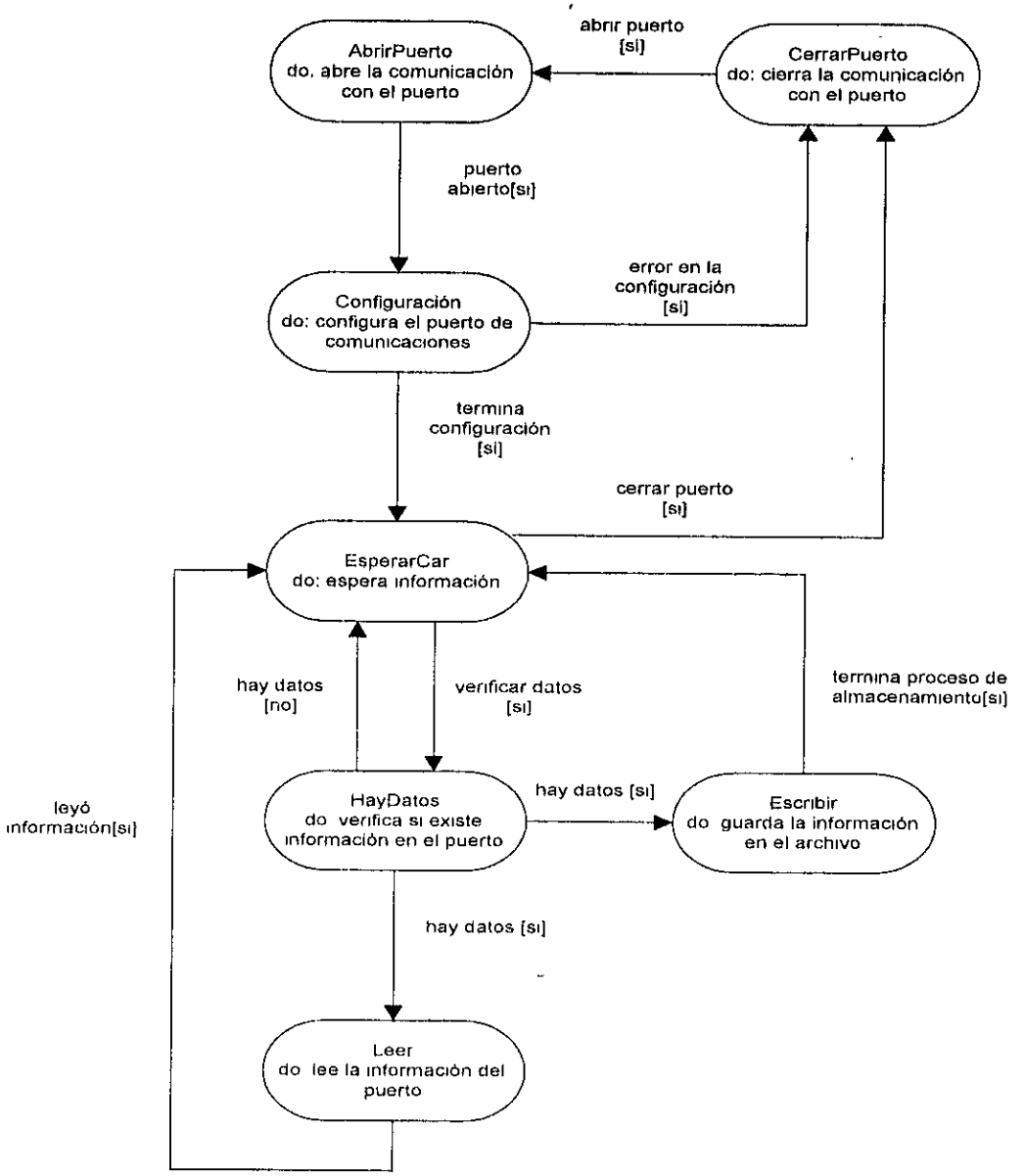
Objeto Control Plataforma Digital



Objeto Página Virtual



Objeto Puerto



3.1.3.14. Diseño de Algoritmos.

Una variedad de representaciones incluidas en el modelo de análisis y el diseño del sistema aportan una especificación de todas las operaciones y atributos. Los algoritmos orientados a objetos se diseñan usando un enfoque que difiere poco del enfoque del diseño de datos y procedimientos utilizados para la ingeniería del software convencional.

Se crea un algoritmo para implementar la especificación de cada operación. En muchos casos, el algoritmo es una simple secuencia computacional o procedural que puede implementarse como un módulo del software autocontenido. Sin embargo, si la especificación de la operación es compleja, pudiera ser necesario la modularización de la operación. Para la realización de esta tarea pueden usarse las técnicas convencionales de diseño procedural.

Las estructuras de datos se diseñan concurrentemente con los algoritmos. Debido a que las operaciones manipulan invariablemente los atributos de una clase, el diseño de estructuras de datos que mejor refleje los atributos tendrá una fuerte influencia sobre el diseño algorítmico de las operaciones correspondientes.

La asignación de operaciones se realiza durante el análisis, pero durante el diseño de objetos será refinada en un número de operaciones más especificas, necesarias para configurar el sistema. Cada una de estas nuevas operaciones se convierte en parte del objeto, y tiene el conocimiento de las estructuras de datos internas que implementan los atributos del objeto, y se invoca a través del envío de mensajes de objeto de la forma:

MENSAJE (operación) -> ACCION: descripción de la acción (verbo)

Los verbos se asocian a acciones u ocurrencias. En el contexto de la formalización del diseño de objetos, consideramos no solamente verbos sino también frases verbales descriptivas (p. ej: "es igual a") como operaciones potenciales. Se aplica

recursivamente el análisis gramatical hasta que cada operación haya sido refinada hasta su nivel más detallado.

Una vez que se a creado el modelo de objetos básico, debe ocurrir la optimización. La métodología de Rumbaugh sugiere tres etapas principales para la optimización del DOO:

- Revisar el modelo objeto-relación para asegurarse que el diseño implementado lleva a una utilización eficiente de los recursos y a una facilidad de implementación. Se debe añadir redundancia donde sea necesario.
- Revisar las estructuras de datos atributos y las operaciones algorítmicas correspondientes para aumentar el rendimiento.
- Crear nuevos atributos para preservar información derivada, evitando para esto la repetición de computaciones previas.

Métodos - Algoritmos.

Objeto Información.

Algoritmo para utilizar el puerto.

```
si (modolectura) UsarNada()
AbrirPuerto(com)
modolectura='p'
```

Algoritmo para abrir un archivo:

```
archivo=abrir archivo(nombre,"rb")
modolectura='a'
```

Algoritmo para cerrar un archivo:

```
si (modolectura igual 'a') cerrar archivo(archivo)
si (modolectura igual 'p') CerrarPuerto()
modolectura=0
```

Algoritmo para leer un byte:

```
si ( falso modolectura) regresa(0)
si (modolectura igual 'a') regresa(obtener carácter(archivo))
si (modolectura igual 'p') si (puerto.HayDatos()) regresa(puerto.Leer())
regresa(0)
```

Algoritmo para verificar que existan datos en el registro de información:

```
si ( falso modolectura) regresa(0)
si (modolectura igual 'a') si (  falso fin de archivo(archivo) ) regresa(1)
si (modolectura igual 'p') si (HayDatos()) regresa(1)
regresa(0)
```

Algoritmo para guardar el carácter leído al buffer de memoria:

```
buffer[posicion del buffer]=c
posicion del buffer+ 1
si (posicion del buffer>=MaxBufInf-1) then
posicion del buffer=MaxBufInf-2
buffer[posicion del buffer]=0
```

Algoritmo para procesar el carácter leído:

```
si (HayDatos())
inicio
        c=LeerByte()

        si (estado igual 0  y  c igual 1)
        inicio
                posicion del buffer=0
                MeterByte(c)
                estado=1
        fin si no
        si (estado igual 1 y  c diferente 3)
        inicio
                MeterByte(c)
        fin si no
        si (estado igual 1 y  c igual 3)
        inicio
                MeterByte(c)
                estado=2
        fin si no
        si (estado igual 2)
        inicio
                estado=3
        fin si no
        si (estado igual 3)
        inicio
                si (crc diferente CRC  y  mensajes)
                inicio escribe("ERROR")  fin
                estado=0
                RegListo[posicion del buffer]=0
                lonreg=posicion del buffer
                posicion del buffer=0
                hayregistro=1
        fin
fin
fin
```

Algoritmo para obtener el registro del buffer:

```
si ( falso hayregistro)
inicio buffer[0]=0  regresa  fin
 (buffer_,RegListo,lonreg)
buffer[lonreg]=0
lonreg=0
hayregistro=0
```

Objeto Publicables

Algoritmo para agregar un registro que se tenga que publicar :

```
convierte(num,temp,10)
concatena(temp,"-")
concatena(temp,nombre)
lista+temp
```

Algoritmo para obtener el nombre del registro que se tenga que publicar :

```
salida[0]=0
repite (i=0 i<l i+ 1)
inicio
        si (reg[i] igual '-') inicio copia cadena(salida,reg+i+1) regresa
fin
```

Algoritmo para eliminar un registro que este publicado :

```
e=primero

mientras(e diferente NULO)
inicio
        si (apuntador a memoria de e diferente NULO)
        inicio
                SacarNombre(apuntador a memoria de e,temp)
                si ( falso compara cadena(temp,nombre))
                 e apunta a Borrar() regresa
        fin
        e=e apunta a sig
fin
```

Algoritmo que sirve para saber si la información se puede publicar :

```
e=primero
mientras(e diferente NULO)
inicio
        si (apuntador a memoria de e diferente NULO)
        inicio
                SacarNombre(e apunta a Mem,temp)
                si ( falso compara cadena(temp,nombre))
                regresa(convierte(e apunta a Mem))
        fin
        e=e apunta a sig
fin
regresa(0)
```

Objeto Puerto

Algoritmo para cerrar el puerto :

```
si (activo)
inicio
        cierra(com)
        activo=0
fin
```

Algoritmo para abrir el puerto :

```
si ((com= crear archivo )) igual NULO )
inicio
                si (mensajes) escribe("Error abriendo el puerto falso ")
                regresa(-1)
```

```
        fin
        Configuración()
        activo=1
        regresa(0)
```

Algoritmo para escribir:
EscribeArchivo(com,direccion de c,1,direccion de tot,NULO)

Algoritmo para leer el puerto:

```
        ReadFile(com, direccion de c,1,direccion de CuantosBytes,NULO)
        si (CuantosBytes igual 0) regresa(0)
        escribe(c)
        regresa(c)
```

Algoritmo que sirve para esperar caracteres del puerto:

```
mientras( falso HayDatos())
  regresa(Leer())
```

Objeto Página Virtual

Algoritmo para crear una página dependiendo del carácter de control :

```
        si (c igual AL)
        inicio
                posy+ 1
                posy=(posy>=tamy)
                si (traduccion) posx=0
                si (mostrar) escribe()
                regresa(1)
        fin
        si (c igual RC)
        inicio
                posx=0  regresa(1)
                si (traduccion)
                inicio
                        posy+ 1
                        posy=(posy>=tamy)
                fin
                si (mostrar)
        fin
        si (c igual TAB) inicio posx+=tab  regresa(1)  fin
        regresa(0)
```

Algoritmo para escribir un long :

```
        convierte(num,cad,10)
        Escribir(cad)
```

Algoritmo para escribir un carácter :

```
        si (ProcesoDeControl(c) regresa
        si ( posx >= tamx ) regresa
        si ( posy >= tamy ) regresa
        (buffer+pos)=c
        si (ncarateress>1)  (buffer+pos+1)=color
        si (posx<50  y  mostrar) escribe(c)
        posx+ 1
        si (posx>=tamx) inicio posx=0  posy+ 1  fin
```

```
        posy=(posy>=tamy)

Algoritmo para escribir un texto :
        for (i=0 i<l i+ 1) Escribir(texto[i])

Algoritmo para escribir alfanuméricos :
        for (i=0 i<l i+ 1) Escribir(texto[i])

Algoritmo para definir las coordenadas de la página :
        posx=x  posy=y
        posx=(posx>=tamx)
        posy=(posy>=tamy)

Algoritmo para limpiar la página virtual :
        for(x=0 x<tamx x+ 1)
        for(y=0 y<tamx y+ 1)
        inicio
                Escribir(carácter 32)
        fin
        posx=posy=0
```

Objeto Elemento Lista.

```
Algoritmo para agregar un nuevo elemento a la lista:
        si (Mem igual NULO)
        inicio
                Mem=nuevo  [tam]
                si (Mem igual NULO) regresa(0)
                tam=tamll=ll
                si (copia diferente NULO)  (Mem,copia,tam_)
                regresa(0)
        fin
        si no
        inicio
                si (sig igual NULO)
                inicio
                        sig=nuevo ElementoLista
                        si (sig igual NULO) regresa(NULO)
                        sig apunta a ant
                fin
                nuevo= sig apunta a Nuevo(tam,copia,ll)
                si (nuevo igual NULO) regresa(sig)
                regresa(nuevo)
        fin
        regresa(NULO)

Algoritmo para borrar un elemento de la lista:
        si (ant diferente NULO) ant apunta a sig =sig
        si (sig diferente NULO) sig apunta a ant =ant
        si (ll diferente NULO)
        inicio
                si ( ((ListaLigada) ll) apunta a  primero igual este) ((ListaLigada )ll) apunta a  primero=sig
                si ( ((ListaLigada ) ll) apunta a  ultimo igual este) ((ListaLigada )ll) apunta a  ultimo=ant
                ((ListaLigada  )ll) apunta a  total-1
                ((ListaLigada  )ll) apunta a  totalbytes-=tam
```

```
        fin
        sig=NULO
        delete este

Algoritmo para información de la lista:
        escribe(num,ant,este,sig)
        si (sig diferente NULO)  sig apunta a ImpInfo()
```

Objeto Lista Ligada

```
Algoritmo para iniciar la lista:
        ultimo=primero=NULO
        índice=0
        total=0
        totalbytes=0

Algoritmo para agregar un elemento a la lista:
        si (primero igual NULO)
        inicio
                primero=nuevo ElementoLista
                si (primero igual NULO) regresa(NULO)
                ultimo=primero
        fin
        ul=ultimo apunta a Nuevo(tam,copia este)
        si (ul igual NULO)
        inicio
                escribe("ListaLigada::No se puede crear el nuevo elemento.")
                regresa(NULO)
        fin
        ultimo=ul
        ultimo apunta a  num=índice
        índice+ 1  total+ 1  totalbytes=tam
        regresa( ultimo apunta a  Mem )

Algoritmo para listar elementos:
        e=primero
        escribe("Objetos guardados:")
        mientras(e diferente NULO)
        inicio
                si (apuntador a memoria de e diferente NULO)
                inicio
                        escribe(e apunta a num)
                        escribe(e apunta a Mem)
                fin si no
                escribe("NO ASIGNADO",e apunta a num)
                e=e apunta a sig
        fin
        escribe("Numero de elementos",total)
        escribe("Cada elemento ocupa",tamaño(ElementoLista))
        escribe("Total por elementos",tamaño(ElementoLista) total)
        escribe("Total ocupado por datos",totalbytes)
        escribe("En total se han ocupado",total tamaño(ElementoLista)+totalbytes)

Algoritmo para borrar elementos de la lista:
        si (e igual NULO) regresa
```

```
si (e igual ap)
inicio
        primero=e apunta a sig
        e apunta a sig=NULO
        e apunta a Borrar()
        regresa
fin
e=e apunta a sig
mientras(e diferente NULO)
inicio
        si ( e igual ap )
        inicio
                e apunta a Borrar()
                e=NULO
                regresa
        fin
        e=e apunta a sig
fin
```

Algoritmo para borrar apuntador a carácter de la lista:

```
ElementoLista e
e=primero
mientras(e diferente NULO)
inicio
        si ( apuntador a memoria de e igual ap )
        inicio
                e apunta a Borrar()
                e=NULO
                regresa
        fin
        e=e apunta a sig
fin
```

Algoritmo para borrar un long de la lista:

```
si (e igual NULO) regresa
mientras(e diferente NULO)
inicio
        si ( e apunta a num igual num )
        inicio
                e apunta a Borrar()
                e=NULO
                regresa
        fin
        e=e apunta a sig
fin
```

Algoritmo para el primero de la lista:

```
si (primero igual NULO) regresa
primero apunta a  Borrar()
```

Algoritmo para obtener el número del elemento de la lista:

```
e=primero
mientras(e diferente NULO)
inicio
        si ( e igual ap ) regresa(e apunta a num)
        e=e apunta a sig
```

```
fin
regresa(indice+1)
```

Algoritmo para *obtener el espacio* de memoria que necesita el elemento de la lista:

```
e=primero
mientras(e diferente NULO)
inicio
        si ( e igual ap ) inicio mem=apuntador a memoria de e  regresa(mem)  fin
        e=e apunta a sig
fin
regresa(NULO)
```

Algoritmo que sirve para obtener el apuntador al objeto de la lista ligada con la información del objeto elemento lista:

```
mientras(e diferente NULO)
inicio
        si ( e apunta a num igual num ) regresa(e)
        e=e apunta a sig
fin
regresa(NULO)
```

Algoritmo para limpiar la lista ligada:

```
mientras(primero diferente NULO)
inicio
        BorrarPrimero()
fin
```

Objeto Control Invision

Algoritmo que sirve para procesar una solicitud:

```
si (ObjInfo igual NULO) regresa
oi=(ObjetoInfosel  )ObjInfo
escribe("Solicitud: ")
 (temp1,sucio,8)
 (temp2,sucio,7)
temp1[8]=0  temp2[7]=0
si ( falso compara cadena(temp1,"NOTICIAS"))
        inicio
                ProcesarIndice(sucio)  oi apunta a  actindicenot=1
        fin
        si no
si ( falso compara cadena(temp2,"PÁGINAS"))
        inicio
                ProcesarIndice(sucio)  oi apunta a  actindicepag=1
        fin
        si no
si ( (sucio) igual 'N')
inicio
        escribe("Una noticia sera publicada.")
        PublicarNoticia(sucio)
fin
si no
si ( (sucio) igual '6')
inicio
        escribe("Un registro .")
```

```
            ProcesarRegistro(sucio)
        fin
        si no
        si ( (sucio) igual 'P')
        inicio
            escribe("Una página .")
            ProcesarPágina(sucio)
        fin
        si no
        si ( (sucio) igual 'p')
        inicio
            escribe("Una prueba.")
            Prueba(sucio)
        fin
        si no
        inicio
            Inv apunta a  DestruirObjeto("La solicitud fue mal escrita, por favor
                                    verifiquela.")
            regresa
        fin

Algoritmo que sirve para publicar una solicitud:
        si (oi igual NULO) regresa

        si (Inv igual NULO)
        inicio
            escribe("No se puede procesar solicitud")
            sescribe(temp,"Error interno en el servidor ",sucio)
            escribe(,temp)
            Inv apunta a  DestruirObjeto(sucio,0,temp)
            regresa
        fin
        copia cadena(temp,oi apunta a  dirnoti)
        concatena(temp,sucio)
        concatena(temp,".not")
        f=abrir archivo(temp,"r")
        si (f igual NULO)
        inicio
            escribe("<>",temp)
            escribe("No se puede procesar solicitud")
            sescribe(temp,"La noticia que ud. solicito no existe.",sucio)
            escribe(,temp)
            Inv apunta a  DestruirObjeto(sucio,0,temp)
            regresa
        fin
        mientras( falso fin de archivo(f))
        inicio
            si (pos<5000) pos+ 1  si no pos=4999
            buffer[pos]=0
        fin
        si (mem igual NULO)
        inicio
            escribe("No se puede procesar solicitud")
            sescribe(temp,"No hay memoria suficiente para procesar la solicitud.",sucio)
            escribe(,temp)
            Inv apunta a  DestruirObjeto(sucio,0,temp)
```

```
            regresa
        fin
        buffer=mem
        Geometria(ncmax,nl)
        limpia pantalla()
        tab=0
                numinv=Inv apunta a  CrearPágina(sucio,ncmax,nl)
                si (numinv igual 0)
                inicio
                escribe("No se puede procesar solicitud")
                sescribe(temp,"No se pudo crear el objeto con las API's.",sucio)
                escribe(,temp)
                Inv apunta a  DestruirObjeto(sucio,0,temp)
                libera memoria(mem)
                regresa
                fin

                copia cadena(temp,sucio)
                l=longitud de la cadena(sucio)
                si (sucio[l-1] igual 'A') temp[l-1]=0
                si no si (sucio[l-1]>'A') temp[l-1]=temp[l-1]-1
                si no temp[0]=0
                si (temp[0] diferente 0)  inicioletrerofin
                copia cadena(temp,sucio)
                l=longitud de la cadena(sucio)
                si (sucio[l-1]>='A') temp[l-1]=temp[l-1]+1
                si no concatena(temp,"A")
                copia cadena(temp2,oi apunta a  dirnoti)
                concatena(temp2,temp)
                concatena(temp2,".not")
                si (ExisteArchivo(temp2))
                        si (temp[0] diferente 0) inicioletrerofin
                si ( falso Inv apunta a  PublicarPágina(sucio,numinv,tam,mem))
                inicio
                        escribe("No se puede procesar solicitud")
                        escribe(temp,"No se pudo publicar el objeto con las API's.",sucio)
                        escribe(temp)
                        Inv apunta a  DestruirObjeto(sucio,0,temp)
                        libera memoria(mem)
                        regresa
                fin
        escribe("Si se pudo procesar solicitud")
        sescribe(temp,"Noticia publicada.",sucio)
        escribe(temp)
        libera memoria(mem)
        cerrar archivo(f)

Algoritmo que sirve para procesar un registro:
        si (dato igual NULO)
        inicio
                        escribe("No se puede procesar solicitud")
                        sescribe(temp2,"El objeto no se encuentra: ")
                        escribe(,temp2)
                        Inv apunta a  DestruirObjeto(nreg,0,temp2)
                        regresa
        fin
```

```
si ( falso numinv)
inicio
        escribe("No se puede procesar solicitud")
        sescribe(temp1,"No se pudo crear el objeto con las API's.")
        escribe(,temp1)
        Inv apunta a  DestruirObjeto(nreg,0,temp1)
        regresa
fin
res=Pubs apunta a  GuardarRegistro(temp2)
si (res)
inicio
    escribe("No se puede procesar solicitud")
    sescribe(temp2,"No se pudo registrar el objeto como publicable",res)
    escribe(,temp2)
    Inv apunta a  DestruirObjeto(nreg,0,temp2)
    regresa
fin
Pubs apunta a  ListarDatos()
Inv apunta a  PublicarPrimerRegistro(nreg,numinv,longitud de la cadena(temp1),(   )temp1)
Inv apunta a  PublicarPrimerRegistro(nreg,numinv,longitud de la cadena(invreg),(   )invreg)
```

Algoritmo que sirve para procesar los registros pendientes:

```
oi=(ObjetoInfosel  )ObjInfo
e=oi apunta a  PendPub.primero
mientras(e diferente NULO)
inicio
    si (apuntador a memoria de e diferente NULO)
    inicio
    si (e apunta a tipo igual 1)
    inicio
        escribe("Objeto publicable:  ",e apunta a Mem)
        Pubs apunta a  SacarDato(1, e apunta a Mem,nombre,100)
        Pubs apunta a  SacarDato(2,e apunta a Mem,numinvc,100)
        numinv=()convierte(numinvc)
        for (i=0 i<longitud de la cadena(e apunta a Mem) i+ 1)
        inicio
        si (e apunta a Mem[i] igual Pubs apunta a  separador)
            si (ncomas igual 3) termina
            i+ 1
            pub=(e apunta a Mem+i)
            escribe("PUBLICANDO PENDIENTES...")
            escribe("Publicando. ",numinv,nombre)
            escribe(,pub)
        fin
        si (e apunta a tipo igual 2)
        inicio
            escribe("Objeto publicable:  ",e apunta a Mem)
            numinv=()convierte(numinvc)
        fin
    fin
    e=e apunta a sig
fin
mientras(oi apunta a  PendPub.primero diferente NULO) oi apunta a  PendPub.BorrarPrimero()
```

Algoritmo que sirve para procesar páginas:

```
si (II igual NULO)
```

```
inicio
                escribe("No se puede procesar solicitud")
                sescribe(temp2,"No existe la página solicitada:sucio+1)
                escribe(,temp2)
                Inv apunta a  DestruirObjeto(sucio,0,temp2)
                regresa
fin
si (m diferente NULO) nreng=convierte(m)
si no
inicio
                escribe("No se puede procesar solicitud")
                sescribe(temp2,"Error en el formato de la página.",sucio+1)
                escribe(temp2)
                Inv apunta a  DestruirObjeto(sucio,0,temp2)
                regresa
fin
si (ll apunta a  ultimo diferente NULO) inicio si (ll apunta a  ultimo apunta a  Mem diferente NULO)
inicio
        si ( ll apunta a  ultimo apunta a  Mem igual '#'  y
           (ll apunta a  ultimo apunta a  Mem+1) igual 'M')
        inicio
                ncol=atoi(ll apunta a  ultimo apunta a  Mem+2)
        fin
fin
si no
inicio
        m=( ll)[3]
        si (m diferente NULO)
                ncol=convierte(m)
        si no
        inicio
                escribe("No se puede procesar solicitud")
                sescribe(temp2,"Error en el formato de la página.",sucio+1)
                escribe(,temp2)
                Inv apunta a  DestruirObjeto(sucio,0,temp2)
                regresa
        fin
fin
si ( falso ncol  or  falso nreng)
inicio
                escribe("No se puede procesar solicitud")
                sescribe(temp2,"La página no contiene informacion.",sucio+1)
                escribe(,temp2)
                Inv apunta a  DestruirObjeto(sucio,0,temp2)
                regresa
fin
si ( falso numinv)
inicio
                escribe("No se puede procesar solicitud")
                sescribe(temp2,"No se pudo crear el objeto con las API's.")
                escribe(,temp2)
                Inv apunta a  DestruirObjeto(sucio,0,temp2)
                regresa
fin
res=Pubs apunta a  GuardarRegistro(temp2)
si (res)
```

```
inicio
        escribe("No se puede procesar solicitud")
        sescribe(temp2,"No se pudo registrar el objeto como
        publicable",res)
        escribe(,temp2)
        Inv apunta a  DestruirObjeto(sucio,0,temp2)
        regresa
fin
```

Algoritmo que sirve para procesar un índice:

```
si ( falso numinv)
inicio
        escribe("No se puede procesar solicitud")
        sescribe(temp2,"No se pudo crear el objeto con las API's.")
        escribe(,temp2)
        Inv apunta a  DestruirObjeto(sucio,0,temp2)
        regresa
fin
res=Pubs apunta a  GuardarRegistro(temp2)
si (res)
inicio
        escribe("No se puede procesar solicitud")
        sescribe(temp2,"No se pudo registrar el objeto como
        publicable ",res)
        escribe(,temp2)
        Inv apunta a  DestruirObjeto(sucio,0,temp2)
        regresa
fin
si (i>=l)
inicio
    sescribe(temp2,"   Siguiente: [1] ",sucio)
    Inv apunta a  PublicarPágina(sucio,numinv,longitud de la cadena(temp2),0,21,(  )temp2)
fin
```

Objeto Invision

Algoritmo para arrancar el publisher:

```
si (res igual variable API)
inicio
    si (mensajes)
    inicio
        escribe("El servidor de Invision arranco bien.")
        escribe("Para el servicio:  ", nombre)
    fin
fin
si no
inicio
    si (mensajes) escribe("No se ha podido arrancar el Publicador: ",res)
    Evalua(res)
    inicio
    en caso de variable del API:
        si (mensajes) escribe("El Publisher ya esta arrancado")
        termina
    en caso de variable del API:
        si (mensajes) escribe("Error interno")
```

```
                termina
            en caso de variable del API:
                si (mensajes) escribe("No hay memoria")
                termina
            fin
            regresa
        fin

Algoritmo para detener el publisher:
        res=Detener Publicador()
        si (res igual variable del API)
        inicio
            si (mensajes) escribe("Publisher de Invision detenido")
        fin
        si no
        inicio
            si (mensajes) escribe("No se ha detener el Publicador: ",res)
            Evalua(res)
            inicio
            en caso de variable de API:
                si (mensajes) escribe("El Publisher ya esta arrancado")
                termina
            en caso de variable del API:
                si (mensajes) escribe("Error interno")
                termina
            fin
            regresa
        fin

Algoritmo para iniciar el servicio en línea:
        res=Servicio en Línea(1)
        si (res igual variable del API)
        inicio
            si (mensajes) escribe("En linea.")
        fin
        si no
        inicio
            si (mensajes) escribe("Posible error")
            Evalua(res)
            inicio
            en caso de variable del api:
                si (mensajes) escribe("El publicador no esta arrancado")
                termina
            en caso de variable del API:
                si (mensajes) escribe("Error interno")
                termina
            en caso de variable del API:
                si (mensajes) escribe("La identificación fuera de rango")
                termina
            fin
        fin

Algoritmo para indicar que el servicio esta fuera de línea:
        res = Servicio en Linea(1)
        si (res igual variable del API)
        inicio
```

```
                si (mensajes) escribe("Fuera de linea.")
        fin
        si no
        inicio
                si (mensajes) escribe("Posible error")
                Evalua(res)
                inicio
                en caso de variable del API:
                        si (mensajes) escribe("El publicador no esta arrancado")
                        termina
                en caso de variable del API:
                        si (mensajes) escribe("Error interno")
                        termina
                en caso de variable del API:
                        si (mensajes) escribe("La identificación fuera de rango")
                        termina
                fin
        fin

Algoritmo para crear página:
        res = crear objeto(direccion de número, direccion de definicion)

        si (res igual variable del API)
  inicio
          si (mensajes) escribe("  Página   "   creada OK",nombre)
  fin
  si no
  inicio
          Evalua(res)
          inicio
                  en caso de variable del API:
                  si (mensajes) escribe("  No hay espacios libres.")
                  termina

                  en caso de variable del API:
                  si (mensajes) escribe("  La página ya existe.")
                  termina
          fin
          regresa(0)
  fin
  regresa(número)

Algoritmo para publicar una página:
res = publicar objeto(direccion de ObjData)
  si (res igual variable del API)
  inicio
          si (mensajes) escribe("  Página   "   publicada OK",nombre)
  fin
  si no
  inicio
          Evalua(res)
          inicio
                  en caso de variable del API:
                  si (mensajes) escribe("  El publicador de Invision no esta activo.")
                  termina
```

```
                en caso de variable del API:
                si (mensajes) escribe(" Error interno.")
                termina
                en caso de variable del API:
                si (mensajes) escribe(" El nombre esta mal.")
                termina
        fin
        regresa(0)
  fin
  regresa(1)

Algoritmo para crear un registro:
        res = Crear Objeto(direccion de número, direccion de definicion)
        si (res igual variable del API)
          inicio
          si (mensajes) escribe(" Registro  "  creado OK",nombre)
        fin
        si no
        inicio
          Evalua(res)
          inicio
                en caso de variable del API:
                si (mensajes) escribe(" No hay espacios libres.")
                termina

                en caso de variable del API:
                si (mensajes) escribe(" El registro ya existe.")
                termina
          fin

Algoritmo para publicar el primer registro:
  res = Publicar objeto(direccion de ObjData)
  si (res igual variable del API)
    inicio
        si (mensajes) escribe(" Registro "  publicada OK",nombre)
  fin
  si no
  inicio
        Evalua(res)
        inicio
        en caso de variable del API
        si (mensajes) escribe(" El publicador de Invision no esta activo.")
                termina
                en caso de variable del API:
                si (mensajes) escribe(" Error interno.")
                termina
                en caso de variable del API:
                si (mensajes) escribe(" El nombre esta mal.")
                termina
        fin
  fin

Algoritmo para actualizar un registro:
res = Publicar objeto( direccion de ObjData)
  si (res igual variable del API)
  inicio
```

```
        si (mensajes) escribe("  Registro "   publicada OK",nombre)
fin
si no
inicio
        Evalua(res)
        inicio
                en caso de variable del API:
                si (mensajes) escribe("  El publicador de Invision no esta activo.")
                termina

                en caso de variable del API:
                si (mensajes) escribe("  Error interno.")
                termina
                en caso de variable del API:
                si (mensajes) escribe("  El nombre esta mal.")
                termina
        fin
fin
```

Objeto Infosel.

```
Algoritmo para  el objeto de informnación
inicio
        estado=0
        hayinfo=0
        mensajes=0
        noticias=bmv=1
        actindicenot=0
        actindicepag=0
        NuevaTabla("ec: t,nf,nt,ni,td")
        si (LeerConfExt("equiv.ini"))
        inicio
                escribe("No fue posible leer el archivo de equivalencias EQUIV.INI")
        fin
        rpnot=20
        concatena(dirnoti,"noticias\\")
fin

Algoritmo para  procesar la información.
inicio
        ProcesarByte
        si (hayregistro)
        inicio
                Esperar encabezado
                si (estado igual a 0)
                inicio
                        si (hayregistro)
                        inicio
                      procesar encabezados de páginas
                                si (RegListo[1] igual a '4')
                                inicio
                            escribe(RegListo)
                                cuerpo de la página
                                        si (RegListo[2] igual a 'B')
                                        inicio
```

```
                                    escribe("Páginas: Un jinete sin cabeza.")
                                    estado=0
                                    hayregistro=0
                                    retorna
                            fin
                            SacarRegistro(buftitulos)
                            si (mensajes) escribe("Página: ", buftítulos)
                            estado=1
                    fin
            procesar encabezados de noticias
                si (RegListo[1] igual a '5' y noticias)
                        inicio
                            si (RegListo[2] igual a 'B')
                        cuerpo de la noticia
                            inicio
                                escribe("Noticias: Un jinete sin cabeza.")
                                estado=0
                                hayregistro=0
                                retorna
                            fin
                            SacarRegistro(buftitulos)
                            si (mensajes) escribe("Noticia: ", buftitulos)
                            estado=1
                        fin
                    procesa informacion de la bolsa
                        si (RegListo[1] igual a '6' y bmv)
                        inicio
                                    SacarRegistro(buftitulos)
                                    si (mensajes) escribe(buftitulos)
                                    ProcesarReg(buftitulos)
                        fin
                fin
        fin
        si no
    procesar los cuerpos
    esperar encabezado
        si (estado igual a 1)
        inicio
                si (hayregistro)
                inicio
                        si (RegListo[1] igual a '5' y noticias)
                        inicio
                                escribe(RegListo)
                                si (RegListo[2] igual a 'A')
                                inicio
                                        escribe("Noticias: Una cabeza sin jínete.",RegListo)
                                        estado=0
                                        ProcesarInfo
                                        retorna
                                fin
                                hayregistro=0
                                estado=0
                                Verificamos si el número de nota es el mismo del encabezado
                                si (pos igual a NULO )
                                inicio
                                        escribe("No se encuentra el número de noticia 1")
```

```
                retorna
            fin
            SacarCadena(temp1,100,pos)

            pos=EncontrarCaracter((  apuntador )RegListo,1) pos+1
            si (pos igual a  NULO )
            inicio
                escribe("No se encuentra el número de noticia 2")
                retorna
            fin
            SacarCadena(temp2,100,pos)
            si ( compara cadena (temp1,temp2))
            inicio
                escribe("Los números de noticias no coinciden:
                    temp1,temp2)
                retorna
            fin
            Construir el nombre de archivo con el número de la nota
            nombre[0]=0
            concatena(nombre,"n")
            concatena(nombre,temp1)
            Vaciar en el archivo la nota.
            pos=EncontrarCaracter((  apuntador )buftitulos,"",1) pos+1
            SacarCadena(temp2,100,pos,3)
            l= longitud de la cadena (temp2) temp2[l-1]=0
            GuardarNoticia(nombre,RegListo)
            escribe(índice,"[N] ",temp1,temp2)
            si (mensajes) escribe(índice)
            INoticias + índice
            ProcesarIndice("NOTICIAS", dirección.INoticias)
fin
si no
Procesar cuerpos de página
si (RegListo[1] igual a '4')
inicio
            si (RegListo[2] igual a 'A')
            inicio
                escribe("Páginas: Una cabeza sin jinete.",RegListo)
                estado=0
                ProcesarInfo
                retorna
            fin
            hayregistro=0
            estado=0
            pos=EncontrarCaracter((  apuntador )buftitulos,2) pos+1
            si (pos igual a  NULO )
 inicio
    escribe("No se encuentra el número de página 1")
    retorna
 fin
            SacarCadena(temp1,100,pos')

            pos=EncontrarCaracter((  apuntador )RegListo,1) pos+1
            si (pos igual a  NULO )
  inicio
     escribe("No se encuentra el número de página 2")
```

```
                                    retorna
                                fin
                                    SacarCadena(temp2,100,pos)
                                    si ( compara cadena (temp1,temp2))
                                    inicio
                                        escribe("Los números de páginas no coinciden:
                                            diferente ",temp1,temp2)
                                        retorna
                                    fin
                                    ProcesarPágina(buftitulos,RegListo)
                            fin
                        si no
                            escribe("Falta cuerpo del Dato")
                        fin
        fin
fin

Algoritmo para buscar caracteres en una cadena.
inicio
        num2=0
        repite desde-hasta (i=0i< longitud de la cadena (buf)i+1)
                si (c igual a buf[i]) inicio num2+1 si (num2 igual a num) retorna (buf+i) fin
        retorna ( NULO )
fin

Algoritmo para obtener una cadena del buffer.
inicio
        i=0
        mientras (i<tambuf-1  y  buf[i] diferente fin)
        inicio
                salida[i]=buf[i]
                i+1
        fin
        salida[i]=0
fin

Algoritmo para recibir informacióndel archivo.
Regresa:
 1) No se puede abrir el archivo
inicio
        buf[1024],salida=0
        f= abrir archivo (nombre,"r")
        si (f igual a  NULO ) retorna (1)
        mientras ( falso salida)
        inicio
                LeerLinea(f,buf,1024)
                si ( apuntador buf igual a '#')
                inicio
                        si (tolower(buf[1]) igual a 'c')
                        inicio
                                si (mensajes) escribe("Campo: ",buf+2)
                                GuardarRegistro(buf+2)
                        fin
                        si (tolower(buf[1]) igual a 't')
                        inicio
                                si (mensajes) escribe("Nueva tabla: ",buf+2)
```

```
                        NuevaTabla(buf+2)
                    fin
                fin
                si ( fin de archivo (f)) salida=1
        fin
        cierra archivo (f)
        retorna (0)
fin

Algoritmo para obtener campos con formato de hora.
inicio
        hrs=buf[0]-'A',min=buf[1]-'A'
        destino[0]=0
        escribe(( apuntador )destino,( apuntador )hrs,( apuntador )min)
        retorna (2)
fin

Algoritmo para obtener campos con formato de fecha.
inicio
        dia=buf[1]-'A'+1,mes=buf[0]-'A'+1,a=buf[2]-'A'+1991
        destino[0]=0
        escribe(( apuntador )destino,"//",( apuntador )dia,( apuntador )mes,( apuntador )a)
        retorna (3)
fin

Algoritmo para obtener una cadena del buffer.
inicio
        p=0
      mientras ( apuntador pi diferente terminacon)
        inicio
                si(p+1 < max)
                inicio
                        apuntador (destino+p)= apuntador pi
                fin
                si no
                inicio
                        si (mensajes)
                                escribe("Error: La cadena en el buffer de entrada no cabe en el buffer
                                de salida.")
                        si (mensajes)
                                escribe("Tam. del buffer de salida:" max  "Posicion actual:"p)
                        retorna (p+1)
                fin
                pi+1 p+1
                apuntador (destino+p)=0
        fin
        retorna (p+1)
fin

Algoritmo para obtener campos de tipo numérico.
inicio
        i=0,j=0,m=1,tipo=0,l= longitud de la cadena ((const   apuntador )base)
        si (  falso  si es digito ( apuntador buf) )
        inicio
                base+1
                si (buf[0]>='a'  y  buf[0]<='g')
```

```
            inicio
              m=buf[0]-'a'
              tipo=1
            fin
               si (buf[0]>='A' y buf[0]<='G')
            inicio
              m=buf[0]-'A'
              tipo=2
            fin
        fin
        repite desde-hasta (i=0,j=0i<li+1)
        inicio
                si ( ( base[i]>='0' y base[i]<='9') ó base[i] igual a )
                inicio
                        buf2[j]=base[i] j+1
                        j=(j>99)? 99:j
                        buf2[j]=0
                fin si no termina
        fin
        si (j igual a 99) inicio retorna (1) fin
        val=(buf2)
        si (base diferente buf)
        inicio
                si (tipo igual a 1) val=val/div[m]
                si (tipo igual a 2) val=val apuntador div[m]
        fin
        si (val>1e10)
        inicio
                escribe("Valor muy grande.Tipo: ",tipo,div[m],buf-1)
                escribe("long:",buf2, longitud de la cadena (buf2))
                escribe(val)
        fin
        repite desde-hasta(i=0i<li+1)
         si ( falso si es digito (base[i]) y base[i] diferente )
            retorna ( (base+i) -buf +1)
   retorna (1)
fin

Algoritmo para procesar registros.
inicio
        copia cadena(bb,"ec: t=< >, nf=< >")
        bb[7]=buf[1]

        desp=SacarCadena((  apuntador )buf+3,(  apuntador )nvalor,49)
        pos=desp+3
        si (mensajes) escribe("[]:",nvalor)
        mientras ( falso salida)
        inicio
                Obtener la letra que identifica al campo de información
                c=buf[pos]
                si (c igual a 3) ve a Fin
                Buscar en la tabla de equivalecias de campos
                si (datoscampo igual a NULO )
                inicio
                        escribe("No se puede encontrar ninguna descripcion para el campo: ",c)
                        escribe("Se busco con la condicion: ",conds)
```

```
        escribe("Registro:",buf)
        retorna (1)
fin
datoscampo+= tamaño (long)
si (mensajes>1) escribe(datoscampo)
Obtener de la tabla el tipo de dato de este campo
SacarDato(5,datoscampo,td,50)
Repite (td)

Obtener de la tabla el nombre en las tablas del campo
SacarDato(3,datoscampo,temp2,100)
concatena(temp3,temp2)
concatena(temp3,"=<")
si (mensajes>1) escribetemp2,td)
si (td[0] igual a 'C')
inicio
        desp=SacarCadena((  apuntador )buf+pos+1,(  apuntador )temp1,100)
        concatena(temp3,temp1)
        si (mensajes>1) escribe("  a  ",temp1)
        pos+=desp+1
fin
si no
si (td[0] igual a 'N')
inicio
        desp=SacarNumero(  apuntador )buf+pos+1,número)
        escribe(temp1,número)
        QuitarCeros(  apuntador )temp1)
        concatena(temp3,temp1)
        si (mensajes>1) escribe("  a  ",temp1)
        pos+=desp
fin
si no
si (td[0] igual a 'H')
inicio
        desp=SacarHora((  apuntador )buf+pos+1,(  apuntador )temp1)
        concatena(temp3,temp1)
        si (mensajes>1)escribe("  a  ",temp1)
        pos+=desp+1
fin
si no
si (td[0] igual a 'F')
inicio
        desp=SacarFecha((  apuntador )buf+pos+1,(  apuntador )temp1)
        concatena(temp3,temp1)
        si (mensajes>1) escribe("  a  ",temp1)
        pos+=desp+1
fin
si no
si (td[0] igual a 'B')
inicio
        desp=(td+1)
        copia(temp1,buf+pos+1,desp)
        temp1[desp]=0
        concatena(temp3,temp1)
        si (mensajes>1) escribe("  a  ",temp1)
        pos+=desp+1
```

```
            fin
            si no    salida=1
            concatena(temp3,">,")
        fin
        escribe(" NO SE PUDIERON PROCESAR TODOS LOS CAMPOS")
        retorna (3)
Fin:
        ActualizarRegistro(nvalor,temp4)
        si (mensajes) escribe(temp3)
        desp=ReemplazarRegistro(cond,temp3,0)
        si (desp)
        inicio
            si (mensajes) escribe(nvalor)
            si (mensajes) escribe("No se puede actualizar el registro en la lista de datos,
                                    se esta crearlo ... ")
            desp=GuardarRegistro(temp3)
            si (desp)
            inicio
                escribe(nvalor)
                escribe("Error al crear registro, CÓDIGO",desp)
                escribe(buf)
                escribe(temp3)
            fin
            si (mensajes) escribe("OK")
        fin
        retorna (0)
fin

Algoritmo para transformar el formato de acuerdo a las equivalencias.
inicio
        si (mensajes>4) escribe(reg)
         apuntador invreg=0
        repite desde-hasta (i=1i<nci+1)
        inicio
            SacarDatoInser((  apuntador )reg,i,(  apuntador )nombre,50,(  apuntador )valor,500)
            copia cadena((  apuntador )temp1,"ec: t=< >, ")
            temp1[7]=reg[0]
            concatena((  apuntador )temp1,"nt=<")
            concatena((  apuntador )temp1,(  apuntador )nombre)
            concatena((  apuntador )temp1,">,")
            dcampo=BuscarUltimoRegistro((  apuntador )temp1)
            si (dcampo igual a  NULO )
            inicio
                escribe(" ERROR falso  falso  falso  : ", temp1)
                ve a fin
            fin
            dcampo+= tamaño (long)
            SacarDato(4,(  apuntador )dcampo,(  apuntador )temp2,50)
            si (mensajes>4) escribe("[] - = ",temp2,nombre,valor)
        fin
fin:
        si (mensajes>4) escribe(invreg)
fin

Algoritmo para crear registros vacios.
inicio
```

```
        ntab[0]=tipo ntab[1]=stipo ntab[2]=0 salida[0]=0
        destab=DesTabla(ntab)
        si (destab igual a  NULO )
        inicio
                escribe("No se encuentra la descripcion de la tabla:  ",destab)
                retorna
        fin
        destab+= tamaño (long)
        ncampos=ContarCamposDes(destab)
        repite desde-hasta (i=1i<=ncamposi+1)
        inicio
                SacarNombreCampoDes((  apuntador )destab,i,(  apuntador )temp1,100)
                escribe(temp2,"ec: t=<>, nt=<>,",tipo,temp1)
                descampo=BuscarUltimoRegistro(temp2)
                si (descampo igual a  NULO )
                inicio
                        escribe("Crear Registro VacioNo se encontro el nombre de campo "
                                del tipo  La condicion " no funciono",temp1,tipo,temp2)
                        retorna
                fin
                descampo+= tamaño (long)
                SacarDato(4,descampo,ninv,100)
                concatena(salida,temp3)
        fin
        concatena(salida,ntab)
fin

Algoritmo para actualizar registros.
inicio
        si (Pubs igual a  NULO ) inicio escribe(" ActReg: Oubs igual a  NULO  falso  ")retorna  fin
        escribe((  apuntador )temp1,"p: nom=<>",nompub)
        pub=BuscarUltimoRegistro(temp1)
        si (pub igual a  NULO ) retorna
        pub+= tamaño (long)
        SacarDato(4,pub,numinv,20)
        SacarDato(5,pub,nombre,100)
        c=separador
        escribe(c,nombre,c,numinv,c,reginv)
        PendPub+temp1
        ultimo a tipo=1
fin

Algoritmo para quitar ceros de los registros.
inicio
          posp=0,l= longitud de la cadena ((  apuntador )buf),i
        repite desde-hasta (i=0i<li+1) si (buf[i] igual a ) inicio posp=i i=l+1 fin
        repite desde-hasta (i=posp +1i<li+1)
        inicio
                si (buf[i] diferente '0') posp=i+1
        fin
        buf[posp]=0
fin

Algoritmo para procesar páginas.
inicio
        l= aparta memoria del tamaño de  ListaLigada
```

```
 si (l igual a  NULO ) retorna
Obtener el número de página
 pos=EncontrarCaracter(cab,2) pos+1
 SacarCadena(nombre,50,pos)
 npag=(nombre)
 si (mensajes) escribe("Procesando página:  ",npag)
  apuntador l+nombre
Obtener nombre
 pos=EncontrarCaracter(cab,3) pos+1
 SacarCadena((  apuntador )nombre,50,pos)
 si (mensajes) escribe(nombre)
  apuntador l+nombre
 copia cadena((  apuntador )nompag,(  apuntador )nombre)
Buscar número de renglones
 pos=EncontrarCaracter(cab,4) pos+1
 nreng=((  apuntador )pos)
 si (pos igual a  NULO   ó  nreng igual a 0)
 inicio
        escribe("No se encontro el número de renglones.")
        retorna
 fin
Guardar en la lista
 SacarCadena(nombre,50,pos)
 si (mensajes) escribe("Reng: ",nombre)
  apuntador l+nombre
Buscar el número de columnas
 pos=EncontrarCaracter(cab,5) pos+1
 ncol=((  apuntador )pos)
 si (pos igual a  NULO   ó  ncol igual a 0)
 inicio
        escribe("No se encontro el número de columnas.")
        retorna
 fin
Guardar en la lista
 SacarCadena(nombre,50,pos)
 si (mensajes) escribe("Cols: ",nombre)
  apuntador l+nombre
Buscar el inicio de la informacion
 pos=EncontrarCaracter(RegListo,4) pos+1
 si ( apuntador pos diferente "")
 inicio
        escribe("No se puede procesar página, no se encontro elcaracter .", apuntador pos)
        retorna
 fin
 pos+1 aporig=pos-RegListo apbuf=0 maxlet=maxlett=0

 f= abrir archivo ("1temp.txt","w")
 mientras (aporig<len)
 inicio
        c=RegListo[aporig]
        si (c igual a '')
        inicio
                buf[maxlet]=0
                escribe(f," a <-",buf)
                si (mensajes>1) escribe("[] ",buf,maxlet)
                maxlett= (maxlett<maxlet)? maxlet:maxlett
```

```
                    apbuf=0 maxlet=0
                     apuntador l+buf
                fin
                si (c>31)
                inicio
                        buf[apbuf]=c
                        apbuf=(apbuf<1022)?  apbuf+1:1022
                        si (c diferente 32) maxlet=apbuf
                fin
                aporig+1
        fin
        GuardarPágina(l)
        escribe(buf,"[P] ",npag,nompag)
        GuardarIndicePágina(buf)
        si (mensajes) escribe("Nombres de páginas guardadas:")
        si (mensajes) Listar
        si (mensajes) escribe("Fin")
        si (l diferente  NULO ) l a Limpiar
         cierra archivo (f)
fin

Algoritmo para guardar el índice.
inicio
        mientras (e diferente  NULO )
        inicio
                si ( falso  compara cadena ((  apuntador )e a memoria .tit)) retorna
                e=e a sig
        fin
        lPáginas+tit
        PublicarIndicePáginas
fin

Algoritmo para procesar índices.
inicio
         npags,pag=0,reng=0
        si (l igual a  NULO ) retorna
        npags=l a total/20+1
        e=l a ultimo
        copia cadena(nombre,nombrebase)
        escribe(temp,"p: nom=<>",(  apuntador )nombrebase)
        reg=BuscarUltimoRegistro(temp)
        si (reg diferente  NULO )
        inicio
                reg+= tamaño (long)
                SacarDato(4,reg,snuminv,50)
                numinv=(snuminv)
                escribe("  es publicable: ",nombrebase,reg)
        fin si no  escribe("No se encuentra <> en la lista de publicables.",nombrebase)
        c=separador
        mientras (pag<npags  y  e diferente  NULO )
        inicio
                si (e a memoria  diferente  NULO )
                inicio
                        escribe(e a memoria )
                        e=e a ant reng+1
                        si (reg diferente  NULO   y  e diferente  NULO )
```

```
                    inicio
                        memset(renglon,' ',80)
                        copia(renglon,( apuntador )e a memoria , longitud de la cadena
                        (( apuntador )e a memoria ))
                        renglon[79]=0
                        escribe(temp, c, nombre, c, numinv,c,c, reng-1, c, renglon)
                        PendPub+temp
                        ultimo a tipo=2
                    fin
                fin
                si (reng>=20)
                inicio
                    reng=0 pag+1
                    escribe(nombre,nombrebase,pag)
                    escribe(temp,p: nom=nombre)
                    reg=BuscarUltimoRegistro(temp)
                    si (reg diferente  NULO )
                    inicio
                        SacarDato(4,reg,snuminv,50)
                        numinv=(snuminv)
                        escribe("  es publicable: ",nombre,reg)
                    fin si no  escribe("No se encuentra <nombre> en la lista de publicables.",nombre)
                fin
        fin
fin
```

Objeto Tablas.

Algoritmo para listar la descripción de la tabla.

```
inicio
        ElementoLista  apuntador e=nombres.primero
        escribe("Tabla - Descripcion")
        mientras (e diferente  NULO )
        inicio
                si (e a memoria  diferente  NULO )
                inicio
                        escribe(  apuntador e a memoria ,  apuntador (e a memoria + tamaño (long)
                fin
                e=e a sig
        fin
fin
```

Algoritmo para listar los datos de la tabla.

```
inicio
        ElementoLista  apuntador e=primero
        escribe("Tabla - Datos")
        mientras (e diferente  NULO )
        inicio
                si (e a memoria  diferente  NULO )
                inicio
                        escribe(" apuntador e a memoria ,  apuntador (e a memoria + tamaño (long))
                fin
                e=e a sig
        fin
fin
```

```
Algoritmo para  obtener el nombre de la tabla de inserción ó campo de la tabla.
inicio
        i,j=0,act=0,l= longitud de la cadena ((  apuntador )buf),pos
        repite desde-hasta (i=0i< longitud de la cadena ((  apuntador )buf)i+1)
        inicio
                si (buf[i] igual a ':'  ó  buf[i] igual a ',')      act+1
                si (act igual a n)
                inicio
                        j=0
                        pos=(buf[i] igual a ',' ó buf[i] igual a ':')? i+1:i
                        mientras  (j<maxbuf2  y  buf[pos] diferente ':'  y
                        buf[pos] diferente ','  y  buf[pos] diferente 0)
                        inicio
                                buf2[j]=buf[pos]
                                j+1 pos+1
                        fin
                        buf2[j]=0
                        Repite ((  apuntador )buf2)
                        retorna
                fin
        fin
        buf2[0]=0
fin

Algoritmo para  contar el número de campos de la descripción.
inicio
        l= longitud de la cadena (buf),i
        num=0

        repite desde-hasta (i=0i<li+1) si (buf[i] igual a ','  ó  buf[i] igual a ':') num+1
        retorna (num)
fin

Algoritmo para  obtener la descripción de la tabla.
inicio
        ElementoLista  apuntador e=nombres.primero
        mientras (e diferente  NULO )
        inicio
                si (e a memoria  diferente  NULO ) si ( n igual a ( apuntador e  a memoria ) )
                        retorna ((  apuntador )e a memoria )
                e=e a sig
        fin
        retorna ( NULO )
fin

Algoritmo para  obtener la descripción de la tabla.
inicio
        ElementoLista  apuntador e=nombres.primero
          apuntador datos,buf[100]
        Repite (nombre)
        mientras (e diferente  NULO )
        inicio
                si (e a memoria  diferente  NULO )
                inicio
                        datos=e a memoria + tamaño (long)
```

```
                    SacarNombreCampoDes(datos,0,buf,100)
                    si ( falso  compara cadena (( apuntador )buf,nombre))
                        retorna (( apuntador )e a memoria )
            fin
            e=e a sig
        fin
        retorna ( NULO )
fin

Algoritmo para  verificar la inserción de un registro.
inicio
        apuntador destab,ntabla[100],ncampo[100], apuntador dato, apuntador registro
        nctab,ncdatos,i,j,maxdato= longitud de la cadena (datos),posreg=0, camposinsertados=0
        buf1[100],okcampo=1, sep[2]
        Repite (datos)
        ntabla = nombre de la tabla que se va  a insertar
        destab = descripcion de la tabla
        destab=DesTabla(ntabla)
        si (destab igual a  NULO ) retorna (2)
        destab=destab+ tamaño (long)
        registro=regtot+ tamaño (long)
        registro[0]=0
        nctab=ContarCamposDes(destab)
        si (nctab<1) retorna (4)
        ncdatos=ContarDatosInser(datos)
        si ( falso ncdatos) retorna (3)
        tamaño de memoria = maxdato
        si (dato igual a  NULO ) retorna (1)
        sep[0]=separador sep[1]=0
        repite desde-hasta (i=1i<=nctabi+1)
        inicio
            buf1 =  nombre del campo
            okcampo=0 posreg=0

            repite desde-hasta (j=1j<=ncdatosj+1)
            inicio
                Obtener dato a insertar
                si ( falso ncampo[0]) inicio  borrar  dato retorna (5) fin
                si ( falso  compara cadena (( apuntador )ncampo,( apuntador )buf1))
                inicio
                    si ( longitud de la cadena (registro)+ longitud de la cadena (dato)+
                    tamaño (long)+1 > maxregtot) inicio  borrar  dato retorna (6) fin
                    concatena(( apuntador )registro,( apuntador )sep)
                    concatena(( apuntador )registro,( apuntador )dato)
                    okcampo=1
                    camposinsertados+1
                    termina
                fin
            fin
            si ( falso okcampo)
            inicio
                concatena(( apuntador )registro,( apuntador )sep)
            fin
        fin
        apuntador ( apuntador )regtot= apuntador ( apuntador )(destab- tamaño (long))
        borrar  dato
```

```
        si (camposinsertados diferente ncdatos) retorna (8)
        retorna (0)
fin

Algoritmo para  obtener el nombre del campo a insertar
inicio
        l= longitud de la cadena ((  apuntador )buf),i,ncampo=0,pos=0, j
        salida=0
        nombre[0]=0 dato[0]=0
        repite desde-hasta (i=0i<li+1)
        inicio
                si (buf[i] igual a '='  y  buf[i+1] igual a '<')
                inicio
                        ncampo+1
                        si (ncampo igual a ndato)
                        inicio
                                ir hacia atrás hasta sacar el nombre
                                repite desde-hasta (j=i-1 j>-1 j--)
                                si ( buf[j] igual a ':'  ó  buf[j] igual a ','  ó  buf[j] igual a '>'  ó
                                   buf[j] igual a ' ') termina
                                nombre[0]=0
                                repite desde-hasta (j=j+1j<ij+1)
                                inicio
                                        nombre[pos]=buf[j]
                                        pos=(pos<maxbufn-1) modulo de pos+1:maxbufn-1 fin
                                nombre[pos]=0
                                pos=0 j=i+2
                                mientras  ( falso salida)
                                inicio
                                        si (buf[j] igual a '>'  y  ( buf[j+1] igual a ','  ó  buf[j+1] igual a 0
                                        ó buf[j+1] igual a ':' ) )
                                        inicio
                                                dato[pos]=0
                                                retorna
                                        fin si no
                                        inicio
                                                dato[pos]=buf[j]
                                                pos=(pos<maxbufd-1)? pos+1:maxbufd-1 j+1
                                                si (j>l) inicio nombre[0]=0 dato[0]=0 retorna  fin
                                        fin
                                fin
                        fin
                fin
        fin
fin

Algoritmo para  contar el número de datos insertados.
inicio
        num=0
        dosp=0
        repite desde-hasta( i=0i< longitud de la cadena (buf)i+1)
        inicio
                si (buf[i] igual a '='  y  buf[i+1] igual a '<') num+1
                si (buf[i] igual a ':') dosp=1
        fin
        si ( falso dosp) retorna (0) si no  retorna (num)
```

```
fin

Algoritmo para  contar el número de campos del registro.
inicio
        l= longitud de la cadena (reg),i,num=0
       repite desde-hasta (i=0i<li+1) si (reg[i] igual a separador) num+1
       retorna (num)
fin

Algoritmo para  verificar que dos campos sean idénticos.
inicio
        n1=ContarCamposReg(reg1),pos1=0,l1= longitud de la cadena (reg1)
        n2=ContarCamposReg(reg2),pos2=0,l2= longitud de la cadena (reg2)
      si (n1 diferente n2) retorna (7)
       buf1=( apuntador ) aparta memoria del tamaño de [l1]
       si (buf1 igual a  NULO ) retorna (1)
       buf2=( apuntador ) aparta memoria del tamaño de [l2]
       si (buf2 igual a  NULO ) inicio  borrar  buf1 retorna (1) fin
       repite desde-hasta (i=0i<n1i+1)
       inicio
              SacarCampoReg(reg1,buf1,l1, dirección pos1)
              SacarCampoReg(reg2,buf2,l2, dirección pos2)
              si ( compara cadena (buf1,buf2)  y  buf2[0])
              inicio
                     borrar  buf1
                     borrar  buf2
                    retorna (8)
              fin
       fin
        borrar  buf1
        borrar  buf2
       retorna (0)
fin

Algoritmo para  obtener un campo determinado del registro.
inicio
        l= longitud de la cadena (reg),i= apuntador pos1,pos=0
       buf[0]=0
       mientras (  apuntador (reg+i) diferente separador  y  i<l) i+1
       si (i>=l) retorna
       i+1
       mientras ( apuntador (reg+i) diferente separador  y  i<l)
       inicio
              buf[pos]=reg[i]
              i+1
              pos=(pos<maxbuf-1)? pos+1:maxbuf-1
       fin
       buf[pos]=0
        apuntador pos1=i
fin

Algoritmo para  realizar el formateode los registros.
inicio
        l= longitud de la cadena (reg),i,pos=0,modo=0,aguas=0
       repite desde-hasta (i=0i<li+1)
       inicio
```

```
                c=reg[i]
                si (modo igual a 0  y  c diferente ' ') modo=1 si no
                si (modo igual a 1  y  c igual a ' ') modo=0
                si (modo igual a 1  y  c diferente ' ')
                inicio
                        si (aguas igual a 1  y  c diferente '<') aguas=0
                        reg[pos]=toupper(reg[i])
                        pos+1
                fin
                si (modo igual a 2) inicio si (i-pos) reg[pos]=reg[i]  pos+1 fin
                si (modo igual a 2  y  c igual a '>'  y  (reg[i+1] igual a ','  ó
                    reg[i+1] igual a ':'  ó  reg[i+1] igual a 0) ) modo=0
                si (modo igual a 1  y  c igual a '='  y   falso aguas) aguas=1
                si (c igual a '<'  y  aguas igual a 1)  inicio modo=2 aguas=0 fin
        fin
        reg[pos]=0
fin

Algoritmo para  crear una nueva tabla.
inicio
         l= longitud de la cadena (descrip),tamtot=l+ tamaño ( long)+1
        Repite (descrip)
        buf=(   apuntador ) aparta memoria del tamaño de  [tamtot]
        si (buf igual a  NULO ) retorna (1)
         apuntador buf)=nt
        copia cadena((  apuntador )(buf+ tamaño (long)),descrip)
        res=nombres.Agregar(tamtot,buf)
         borrar  buf
        nt+1
        tamtabmax=(tamtabmax<l)? l:tamtabmax
        si (res igual a  NULO ) retorna (0)
        retorna (1)
fin

Algoritmo para  guardar el registro en el buffer.
inicio
         l= longitud de la cadena (reg)+ tamaño (long)+tamtabmax
        Alojar memoria para el registro modificado
        si (buf igual a  NULO ) retorna (1)
        res=ProducirRegistro(reg,buf,l)
        si (res) inicio  borrar  buf retorna (res) fin
        Agregar( longitud de la cadena (buf+ tamaño (long))+1+ tamaño (long),(   apuntador )buf)
         borrar  buf
        retorna (0)
fin

Algoritmo para  reemplazar registros.
inicio
        ElementoLista  apuntador e=primero
        cambios=0
         l1= longitud de la cadena (con)+ tamaño (long)+tamtabmax,
         l2= longitud de la cadena (donde)+ tamaño (long)+tamtabmax, res,l3
        Repite (donde)
        Repite (con)
        Verificar si la tabla existe
        SacarNombreCampoDes((   apuntador )donde,0,(   apuntador )ntab,100)
```

```
        destab=DesTabla(ntab)
        si (destab igual a  NULO ) retorna (2)
        Si la tabla existe se extrae la información
        numtab= apuntador (  apuntador )destab
        destab+= tamaño (long)
        Alojar memoria para el registro limpio
        si (registrolimpio igual a  NULO ) retorna (1)
        res=ProducirRegistro(con,registrolimpio,l1)
        Si hay algun error durante la limpieza
          inicio retorna (res)  borrar  registrolimpio fin
        l3= longitud de la cadena (registrolimpio+ tamaño (long))+ tamaño (long)+1
        Alojar memoria para el registro de condiciones
        si (condiciones igual a  NULO ) inicio  borrar  registrolimpio retorna (1) fin
        Limpiar condiciones
        res=ProducirRegistro(donde,condiciones,l2)
        si (res) inicio  borrar  registrolimpio  borrar  condiciones retorna (res) fin
        condiciones+= tamaño (long)
        mientras (e diferente  NULO )
        inicio
                si (e a memoria  diferente  NULO )
                inicio
                        numtabreg= apuntador (  apuntador )e a memoria
                        si (numtabreg igual a numtab)
                        inicio
                                datoslista=(  apuntador )(e a memoria + tamaño (long))
                                res=CompararCampos(datoslista,condiciones)
                                si ( falso res)
                                inicio
                                        si (e a memoria  diferente  NULO )  borrar  e a memoria
                                        e a memoria =(  apuntador )
                                        aparta memoria del tamaño de  [l3]
                                        si (e a memoria  diferente  NULO )
                                                copia(e a memoria ,registrolimpio,l3)
                                        cambios+1
                                fin
                        fin
                fin
                e=e a sig
        fin
        condiciones-= tamaño (long)
         borrar  condiciones
         borrar  registrolimpio
        si ( falso cambios) retorna (9)
        retorna (0)
fin

Algoritmo para  buscar el último registro.
inicio
        ElementoLista  apuntador e=ultimo
          l2= longitud de la cadena (donde)+ tamaño (long)+tamtabmax, res
        verificar si el registro existe
        destab=DesTabla(ntab)
        si (destab igual a  NULO ) retorna ( NULO )
       Si el registro existe
        numtab= apuntador (  apuntador )destab
        destab+= tamaño (long)
```

```
        Alojar memoria para el registro de condiciones
        si (condiciones igual a  NULO ) retorna ( NULO )
        Limpiar condiciones
        res=ProducirRegistro(donde,condiciones,l2)
        si (res) inicio  borrar  condiciones retorna ( NULO ) fin
        condiciones+= tamaño (long)
        e=ultimo
        mientras (e diferente  NULO )
        inicio
                moveralsig=1
                si (e a memoria  diferente  NULO )
                inicio
                        numtabreg= apuntador (  apuntador )e a memoria
                        si (numtabreg igual a numtab)
                        inicio
                                datoslista=(  apuntador )(e a memoria + tamaño (long))
                                res=CompararCampos(datoslista,condiciones)
                                si ( falso res)
                                inicio
                                        condiciones-= tamaño (long)
                                         borrar  condiciones
                                        ret=(  apuntador )e a memoria
                                        e= NULO
                                        retorna (ret)
                                fin
                        fin
                fin
                e=e a ant
        fin
        condiciones-= tamaño (long)
         borrar  condiciones
        retorna ( NULO )
fin

Algoritmo para  eliminar registros
inicio
        borrados=0
          l2= longitud de la cadena (donde)+ tamaño (long)+tamtabmax, res
        verificar si la tabla existe
        SacarNombreCampoDes((  apuntador )donde,0,(  apuntador )ntab,100)
        destab=DesTabla(ntab)
        si (destab igual a  NULO ) retorna (2)
        Si la tabla existe
        numtab= apuntador (  apuntador )destab
        destab+= tamaño (long)
        Alojar memoria para el registro de condiciones
        si (condiciones igual a  NULO ) retorna (1)
        Limpiar condiciones
        res=ProducirRegistro(donde,condiciones,l2)
        si (res) inicio  borrar  condiciones retorna (res) fin
        condiciones+= tamaño (long)
        mientras (e diferente  NULO )
        inicio
                moveralsig=1
                si (e a memoria  diferente  NULO )
                inicio
```

```
                    numtabreg= apuntador (  apuntador )e a memoria
                    si (numtabreg igual a numtab)
                    inicio
                        datoslista=(  apuntador )(e a memoria + tamaño (long))
                        res=CompararCampos(datoslista,condiciones)
                        si ( falso res)
                        inicio
                            es=e a sig
                            e a Borrar
                            moveralsig=0
                            e=es
                            borrados+1
                        fin
                    fin
                fin
                si (moveralsig) e=e a sig
        fin
        si ( falso borrados) retorna (9)
        retorna (0)
fin

Algoritmo para  obtener un campo del registro de entrada.
inicio
        i=0,pos=0
         nca=0
        c,salida=0
        buf[0]=0
        mientras (i< longitud de la cadena (reg))
        inicio
                c=reg[i]
                si (c igual a separador  y  nc igual a nca) retorna

                si (nc igual a nca  y  c diferente separador)
                inicio
                        buf[pos]=c
                        pos=(pos<maxbuf-1)? pos+1:maxbuf-1
                        buf[pos]=0
                fin
                si (c igual a separador  y  nc diferente nca)  nca+1
                i+1
        fin
fin
```

3.1.4. Modelo Funciónal.

La información se transforma a medida que fluye por un sistema. El sistema acepta entradas en gran variedad de formas, aplica elementos de hardware, software y humanos para transformar la entrada en salida, y produce salidas en una gran variedad de formas. La señal de entrada puede ser una señal de control transmitida por un controlador, una serie de números escritos por un enlace de red o un archivo voluminoso de datos recuperado de un CD-ROM. La transformación puede ser, desde una sencilla comparación lógica hasta un complejo algoritmo numérico o un mecanismo de reglas de inferencia de un sistema experto. La salida puede ser el encendido de un diodo emisor de luz (LED) o un informe de 200 páginas. Se puede crear un modelo de flujo para cualquier sistema, independientemente del tamaño y de la complejidad.

El análisis estructurado es una técnica del flujo de datos y del contenido de la información. Se representa el funcionamiento general del sistema como una única transformación de información, representada como una burbuja. En entidades externas, representadas por cuadros, se origina una o más entradas, que aparecen como flechas etiquetadas. La entrada conduce a la transformación que produce información de salida, dirigida hacia otras entidades externas. Se debe señalar que se puede aplicar el modelo al sistema completo o a un elemento de software. La clave está en presentar la información que entra y la que es producida por la transformación.

A medida que la información se mueve a través del software, es modificada por una serie de transformaciones. El diagrama de flujo de datos es una técnica que representa el flujo de la información y las transformaciones que se aplican a los datos al moverse desde la entrada hasta la salida. En la figura 3.1.21 se muestra la forma básica de un diagrama de flujo de datos.

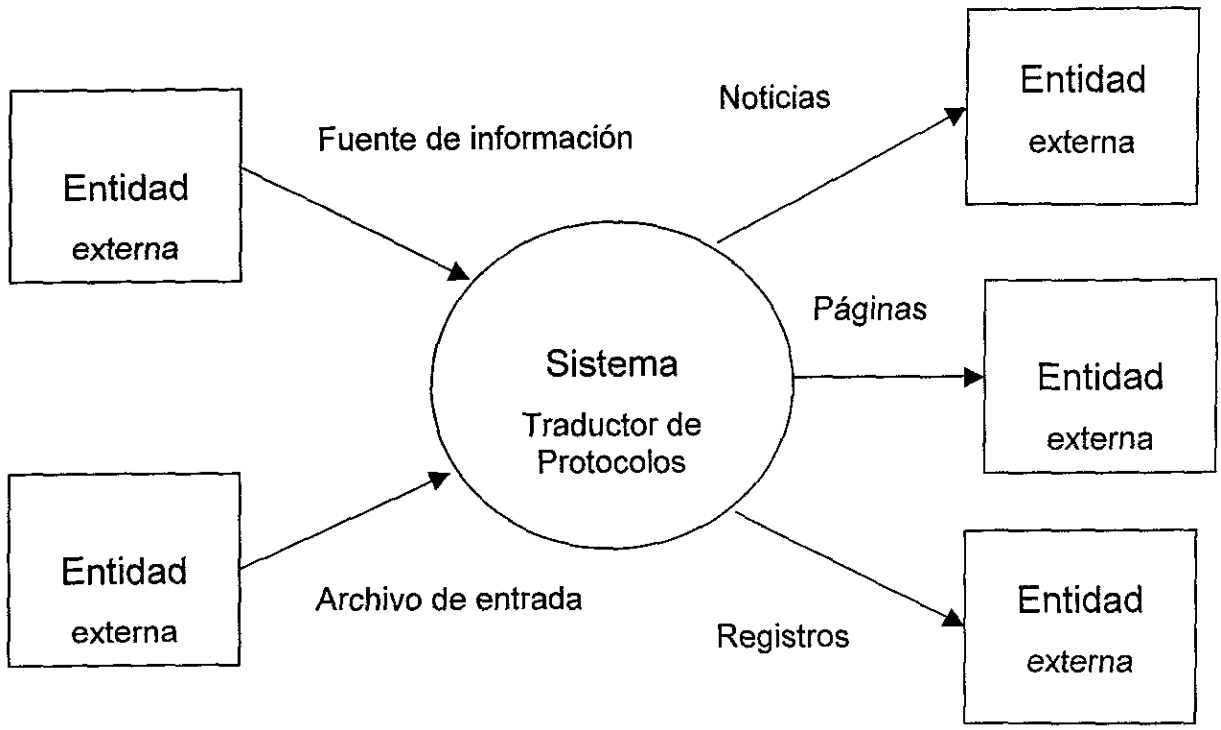


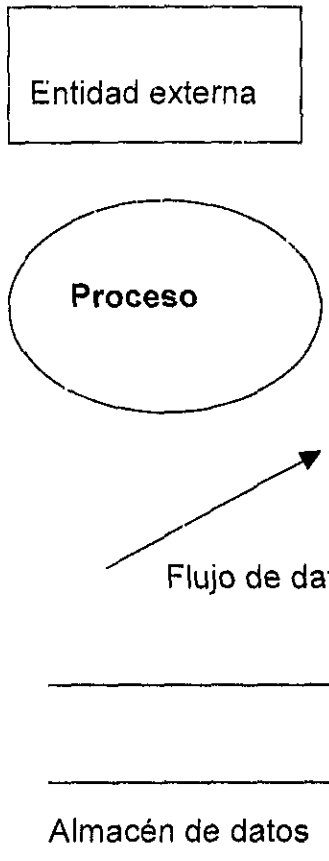
Figura 3.1.21. *Modelo del flujo de información* del Traductor de Protocolos nivel 0.

Se puede usar un diagrama de flujo de datos para representar un sistema o software a cualquier nivel de abstracción. De hecho los diagramas de flujo de datos pueden ser divididos en niveles que representan un mayor flujo de información y un mayor detalle funcional. Por consiguiente el diagrama de flujo de datos proporciona un mecanismo para el modelado funcional así como el modelado de flujo de información, con esto se genera el modelo funcional.

Un diagrama de flujo de datos de nivel 0 también es denominado modelo fundamental del sistema o modelo de contexto y representa al elemento completo de software como una sola burbuja con datos de entrada y salida representados por flechas de entrada y salida respectivamente. Al particionar el diagrama de flujo de datos de nivel 0 para mostrar más detalles, aparecen representados procesos y cambios de flujo adicionales. Por ejemplo, un diagrama de flujo de datos de nivel 1 puede contener 5 o 6 procesos

con flechas interconectadas. Cada uno de los procesos representados en el nivel 1 es una subfunción del sistema general en el modelo de contexto.

En la figura 3.1.22 se ilustra la notación básica que se utiliza para crear un diagrama de flujo de datos. El rectángulo se utiliza para representar una entidad externa, es decir un elemento del sistema, u otro sistema que produzca información a ser transformada por el software o función que reciba esta información. Un circulo u ovalo representan un proceso o transformación que se aplica a los datos y los cambia de alguna forma. Todas las flecha de un diagrama de flujo de datos deben estar etiquetadas. La línea doble representa un almacén de datos (información almacenada que es utilizada por el software). La sencillez de la notación de un diagrama de flujo de datos es una de las razones por las que las técnicas de análisis estructurado son ampliamente utilizadas.



Un productor o consumidor de información que reside fuera de los límites del sistema a ser modelado

Un transformador de información (una función) que reside dentro de los límites del sistema a ser modelado

Un flujo de datos; la cabeza de la flecha indica la dirección del flujo de datos.

Un depósito de datos que se va a almacenar para uso por uno o varios procesos; puede ser tan simple como una memoria intermedia o cola; o tan sofisticado como una base de datos relacional

Figura 3.1.22. Notación básica del diagrama de flujo de datos.

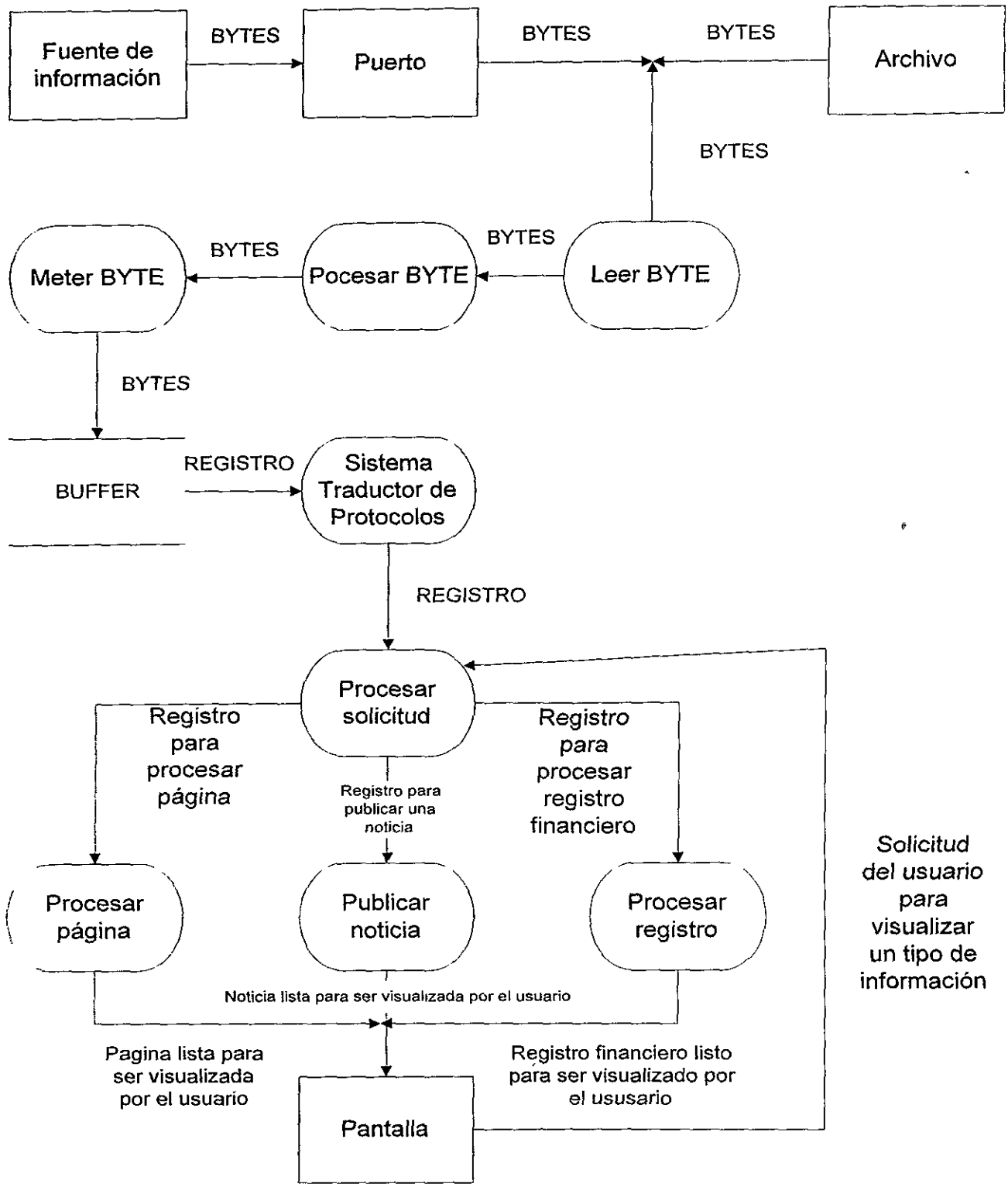


Figura 3.1.2. Sistema Traductor de Protocolos en un nivel 2 del diagrama de flujo de datos.

Es importante señalar que el diagrama no proporciona ninguna indicación explícita de la secuencia del procesamiento. El procedimiento o la secuencia puede estar implícitamente en el diagrama, pero la representación procedural explícita generalmente queda pendiente hasta el diseño del software.

Se puede refinar cada uno de los procesos en distintos niveles para mostrar un mayor detalle. La figura 3.1.23 muestra este concepto. El modelo fundamental del Sistema Traductor de Protocolos muestra dos entradas principales que pueden ser la fuente de información o un archivo para prueba y tres salidas que pueden ser una noticia, una página o un registro financiero. Refinando el modelo de flujo de nivel 0 para determinar que tipo de entrada es la que recibe el Sistema Traductor de Protocolos y que tipo de salida esta solicitando el usuario.

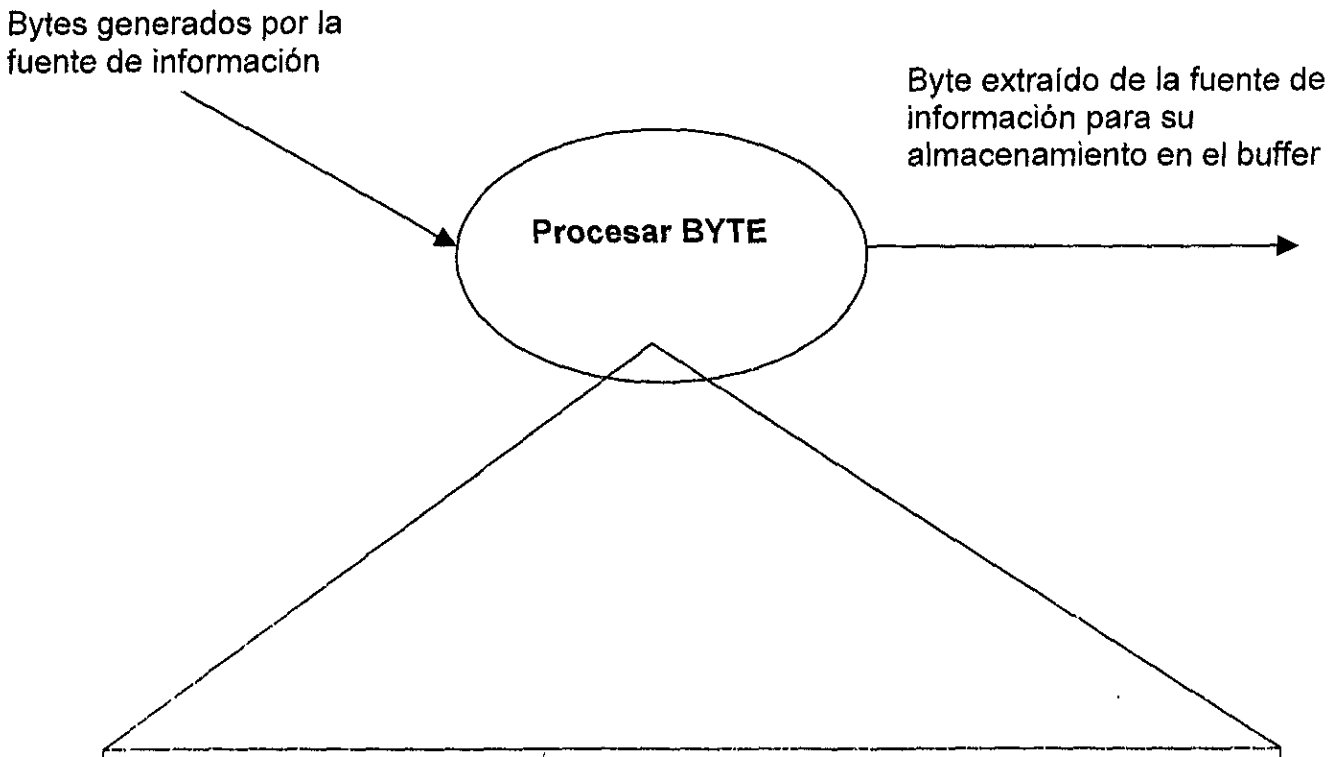
El diagrama de flujo de datos es una herramienta gráfica que puede ser muy valiosa durante el análisis de requisitos de software. Sin embargo el diagrama puede llevar a una confusión si se confunde su función con la del gráfico de flujo. Un diagrama de flujo de datos representa el flujo de la información sin representación explícita de la lógica de procesamiento (condiciones o bucles).

La notación básica que se utiliza para desarrollar un diagrama de flujo de datos no es en sí misma suficiente para describir los requisitos de software. Por ejemplo una flecha de un diagrama de flujo de datos representa un objeto de dato que entra o sale de un proceso. Un almacén de datos representa una colección organizada de datos. Pero ¿cuál es el contenido de los datos implicados en la flecha o en el almacén?, Si la flecha o el almacén representan una colección de objetos ¿cuáles son?. Para responder a estas preguntas, aplicamos otro componente de la notación básica del análisis estructurado, el diccionario de datos.

Finalmente, la notación gráfica representada en la figura 3.1.22 debe ser ampliada con texto descriptivo. Se puede usar una especificación del proceso para especificar los detalles que implica un proceso del diagrama de flujo de datos. La narrativa de

procesamiento describe la entrada al proceso el algoritmo que se aplica a esa entrada y la salida que se produce. Además la especificación del proceso indica las restricciones y limitaciones impuestas al proceso, las características de rendimiento que son relevantes al proceso y las restricciones de diseño que puedan tener influencia en la forma de implementar el proceso.

Se utiliza la especificación del proceso para describir todos los procesos del modelo de flujo que aparecen en el nivel final de refinamiento. El contenido de la especificación de procesamiento puede incluir una narrativa, una descripción en lenguaje de descripción de programa del algoritmo del proceso, ecuaciones matemáticas, tablas, diagramas o gráficos. Al proporcionar una especificación del proceso que acompañe cada proceso o función del modelo de flujo, se crea una mini-especificación que sirve como primer paso para la creación de la especificación de requisitos del software y constituye una guía para el diseño de la componente del programa que implementará el proceso, esto se representa en la figura 3.1.24.

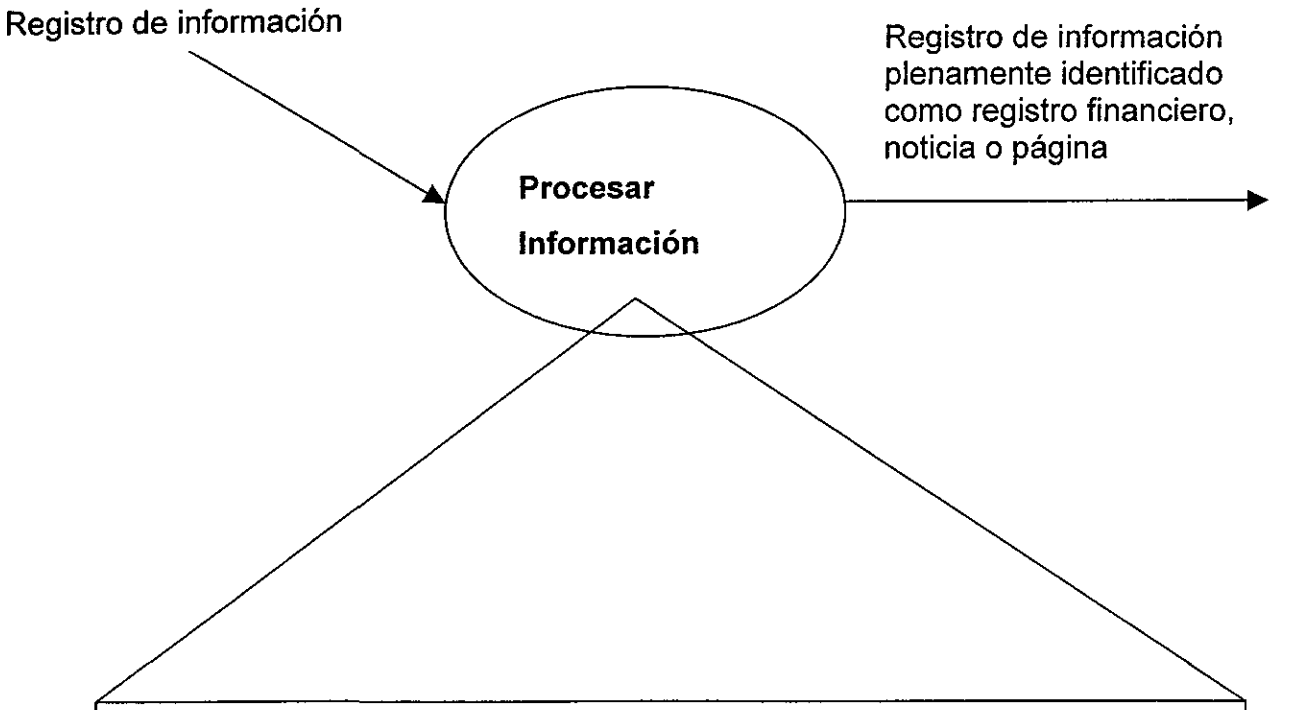


EP: Narrativa de procesamiento para la función Procesar BYTE

La función procesar BYTE va a leer cada byte de información ya sea del puerto de donde la información proviene directamente de una fuente de información o del archivo que se tiene en caso de querer realizar una prueba, esto lo define la función Leer BYTE. Realiza la lectura de cada byte previendo el fin de registro en el archivo o en caso de leer el puerto verifica la existencia de bytes en este. Estos bytes que se van leyendo se insertan en un buffer por medio de la función Meter BYTE hasta tener el registro completo y lista para pasar al traductor de protocolos.

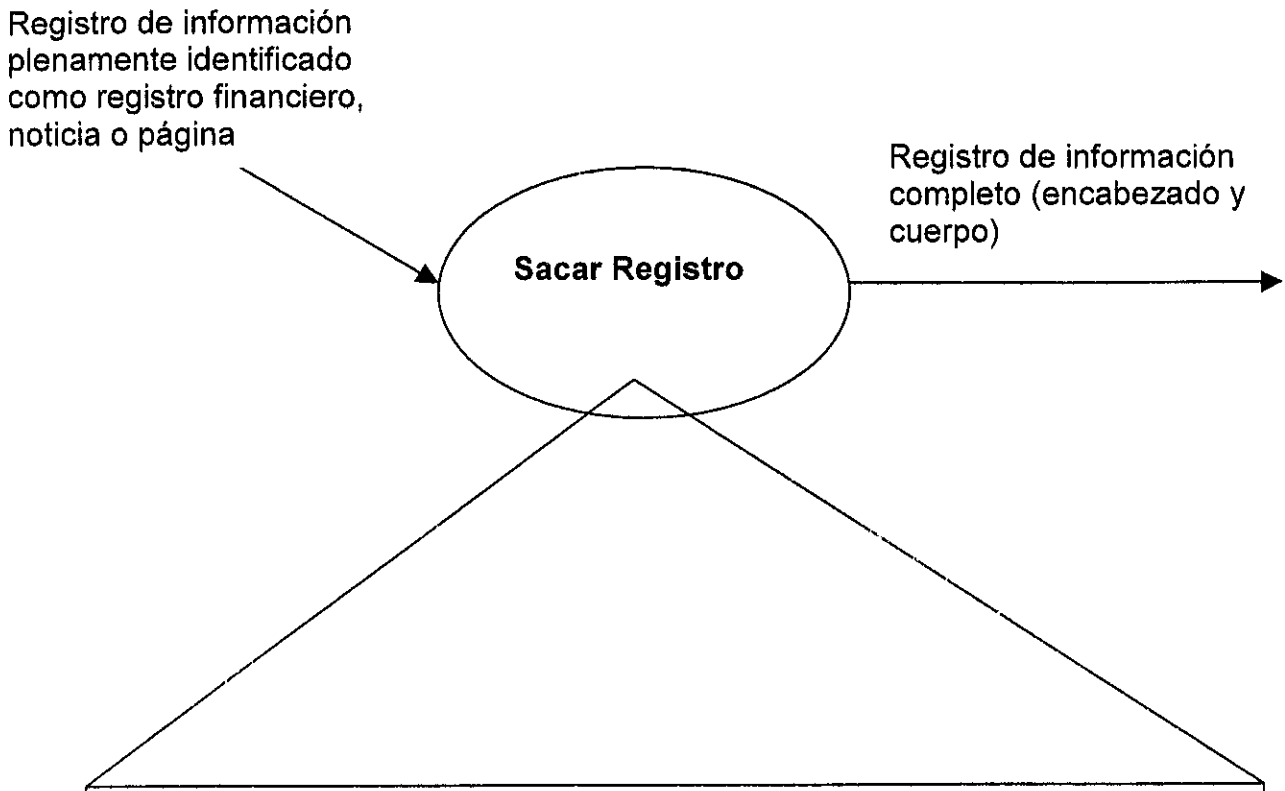
Figura 3.1.24. Especificación de procesamiento para la función procesar byte.

Especificaciones de Procedimientos.



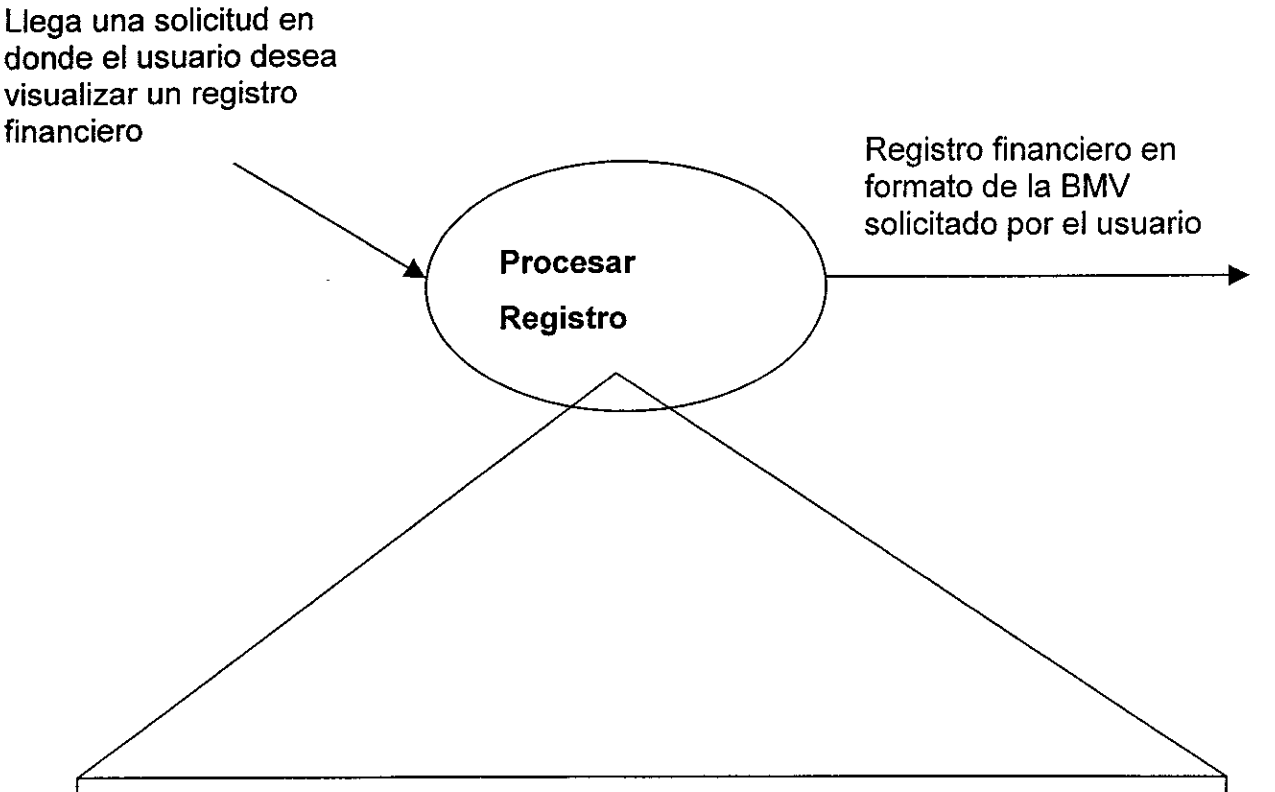
EP: Narrativa de procesamiento para la función Procesar Información
Este proceso toma el registro insertado en el buffer y checa el primer campo para determinar si se trata de una noticia (info.registrolisto='5'), una página (info.registrolisto='4') o un registro (info.registrolisto='6'), en el caso de noticias y páginas, posteriormente checa el segundo campo para determinar si se trata del encabezado (info.registrolisto='A') o del cuerpo (info.registrolisto='B').

Especificaciones de Procedimientos.



EP: Narrativa de procesamiento para la función Sacar Registro
Mediante un proceso llamado SACAR REGISTRO y al tener identificado ya el primer campo del registro, en el caso de noticias y páginas, se guarda el registro del encabezado en otro buffer, para poder así recibir el cuerpo en el buffer inicial y posteriormente realizar un match entre ellos y poder así asegurar mediante el # de página o noticia y nombre de las mismas que el cuerpo que se está recibiendo corresponde al encabezado y formar la noticia o página según sea el caso. En el caso de los registros de la BMV(Bolsa Mexicana de Valores), el proceso SACAR REGISTRO lo extrae del buffer inicial para dejarlo en un buffer temporal para posteriormente utilizarlo en el proceso PROCESA REGISTRO.

Especificaciones de Procedimientos.

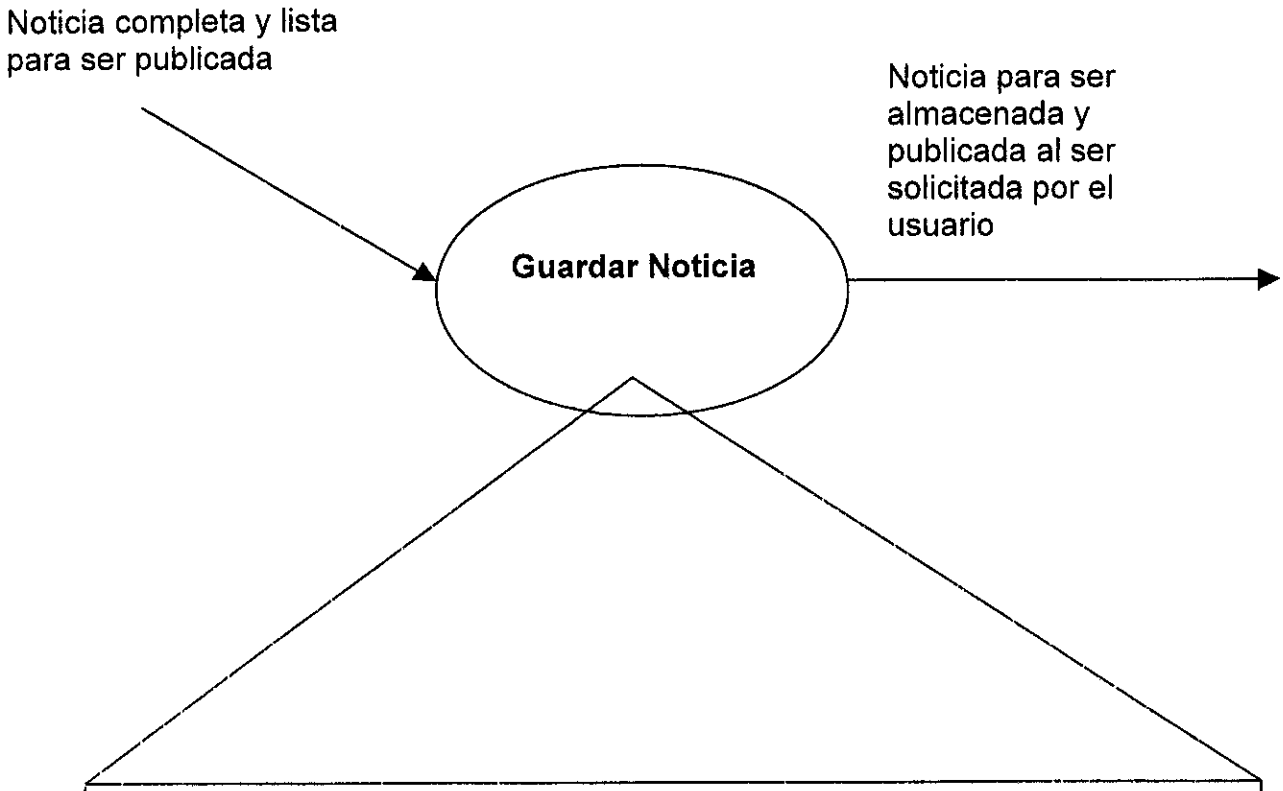


EP: Narrativa de procesamiento para la función Procesar Registro

Extrae el registro del buffer temporal que viene en formato de información de la BMV, es decir trae algunas claves en sus campos que a simple vista no son comprensibles para el usuario, además existen 5 tipos de registros de información de la BMV (posturas de warrants, transacciones de acciones de renta variable, posturas de acciones de renta variable, evolución de índices y transacciones de warrants), así que primeramente se identifica el tipo de registro, mediante una tabla de equivalencias que contiene la estructura de cada uno de estos tipos de registros y las equivalencias del registro de información BMV y el registro de la PD (Plataforma Digital). El registro se extrae del buffer temporal a un buffer BMV para ser comparado con los cinco tipos de registros y checar a que tipo de registro corresponde, ya identificado el tipo de registro se deja en el formato de este registro dentro de una tabla BMV (que es un buffer).

Por otro lado al tener ya el encabezado y el cuerpo en el caso de noticias y páginas se tienen los procesos GUARDAR NOTICIA y PROCESAR PÁGINA respectivamente.

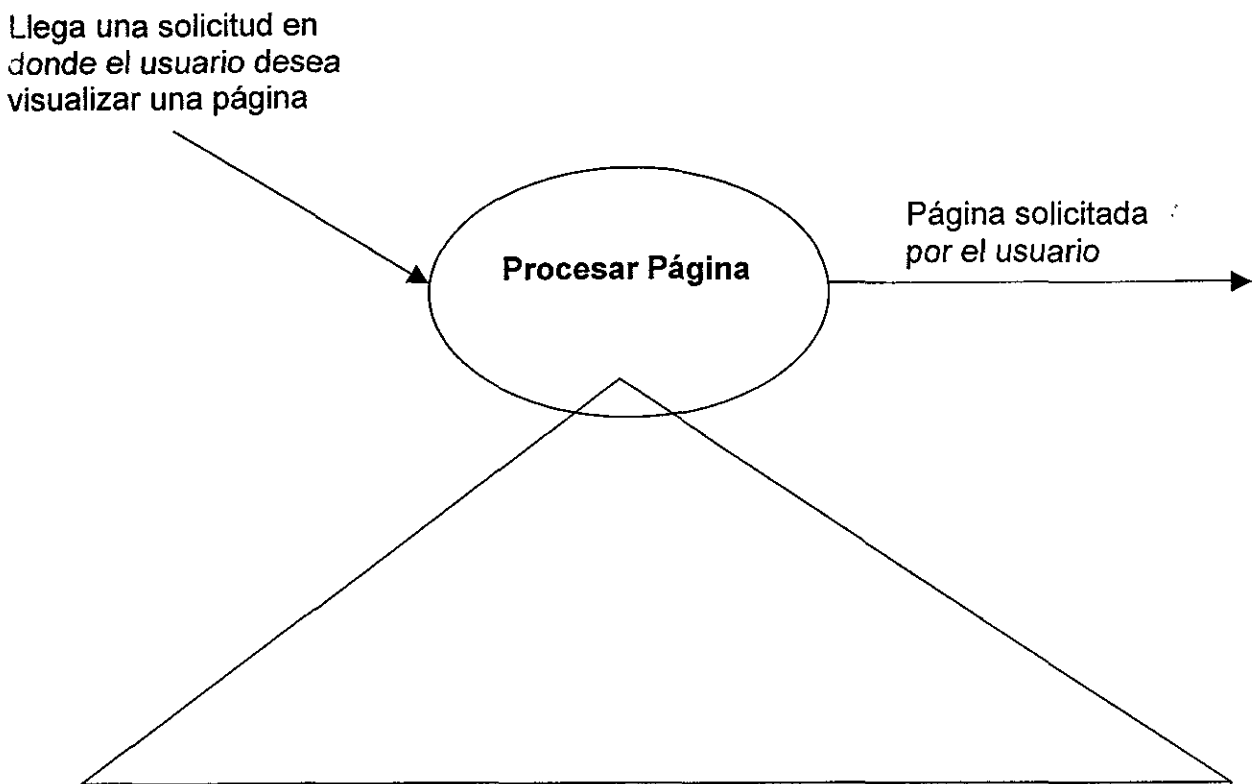
Especificaciones de Procedimientos.



EP: Narrativa de procesamiento para la función Guardar Noticia

Una vez realizada la comparación entre número y nombre del encabezado de la noticia con el cuerpo de la noticia, se genera un archivo con el número de la noticia para copiar lo que contiene el buffer temporal (encabezado de la noticia) y el buffer inicial (cuerpo de la noticia), listo para esperar solicitud para ser publicada.

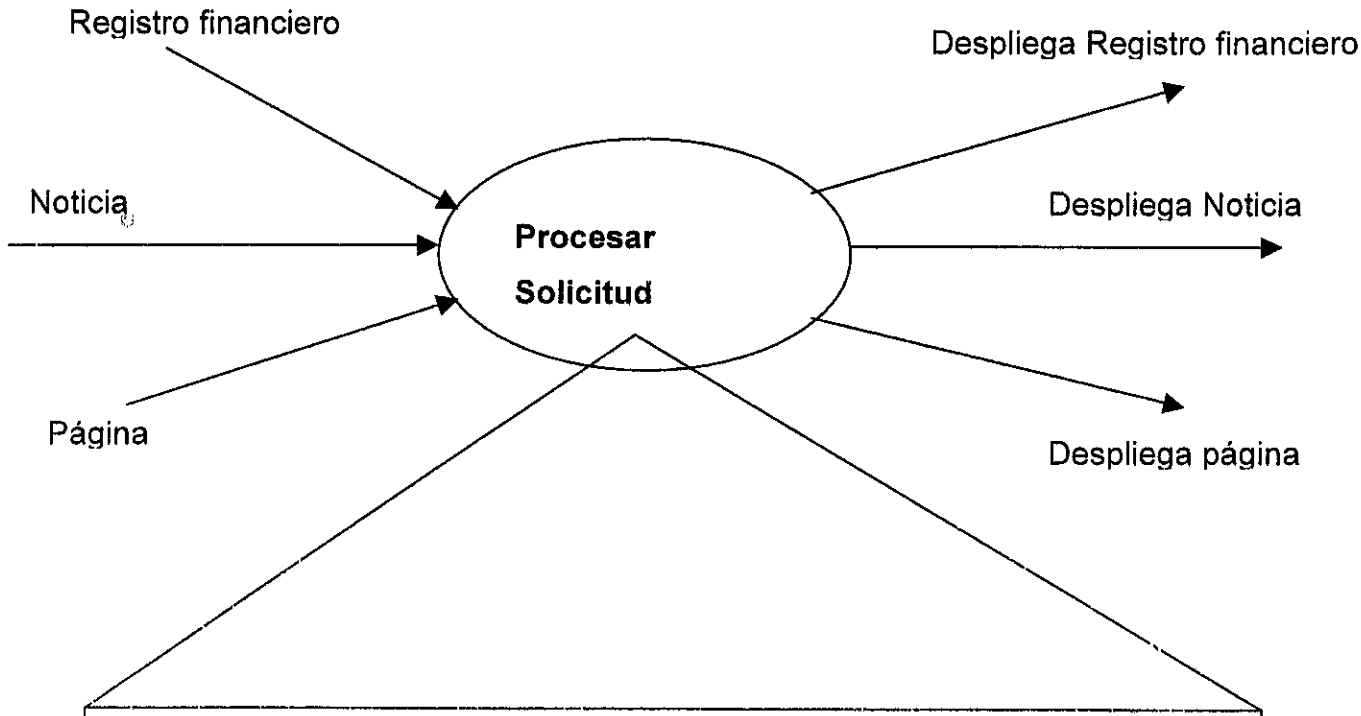
Especificaciones de Procedimientos.



EP: Narrativa de procesamiento para la función Procesar Página

Se extrae datos de la página como número de columnas, número de renglones, nombre y número de la página, para averiguar si existe esta página en la lista ligada de listas ligadas. Si la página no existe dentro de esta, se crea esta lista para vaciar la información contenida en el buffer inicial y en el buffer temporal, es decir el encabezado y el cuerpo de la página, si la lista ya existe, se compara cada campo de cada registro que componen a la página para encontrar las diferencias y agregarlas a la lista ligada. La razón de utilizar una lista ligada es para facilitar la ubicación de las diferentes páginas que se estén procesando.

Especificaciones de Procedimientos.



EP: Narrativa de procesamiento para la función Procesar Solicitud

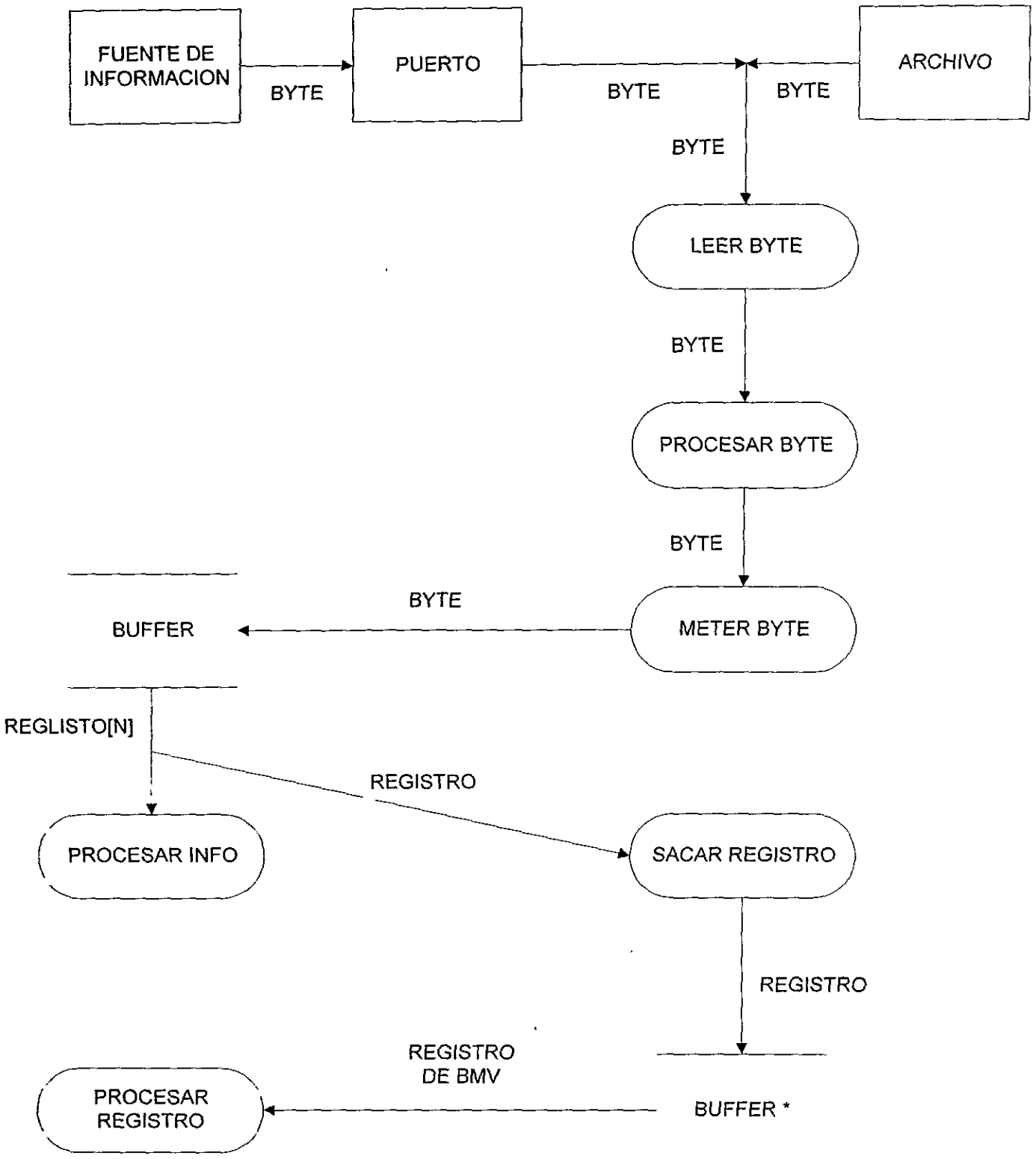
Este proceso identifica una solicitud emitida por el usuario la cual puede ser que requiera un registro, una página o una noticia.

En el caso de solicitar un registro, previamente identificado el tipo de registro por el proceso PROCESA REGISTRO, se extrae el registro de la tabla BMV y se realiza una comparación con la tabla de equivalencias para transformar este registro para poder ser desplegado en la Plataforma Digital, se almacena el registro en un buffer temporal, y mediante el proceso CREAR REGISTRO, se definen los atributos de el formato de despliegue y se depositan los campos del registro transformado en el formato de despliegue para ser apreciados por el usuario final.

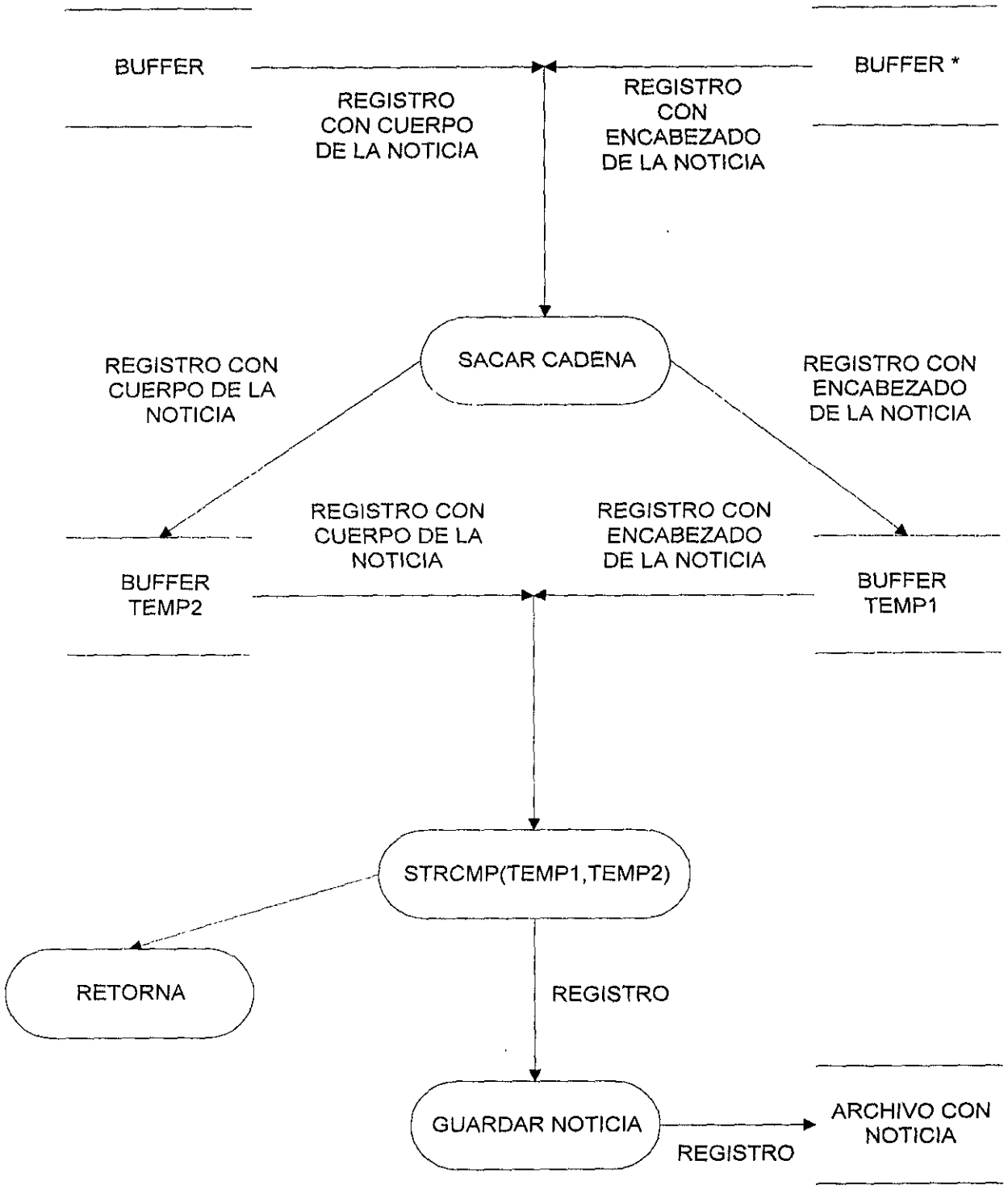
En el caso de solicitar una noticia, se copia la noticia solicitada del archivo a un buffer temporal, se definen los atributos para el formato de despliegue de la noticia y se vacía la información del encabezado y cuerpo de la noticia, que posteriormente es publicada para la apreciación del usuario.

En el caso de solicitar una página, esta es buscada en la lista ligada, y al ser localizada, es puesta en un buffer temporal, el cual se está actualizando cada que llega información que afecta a la página solicitada por el usuario (solo algunos campos), se presenta en su formato de despliegue para ser apreciado por el usuario.

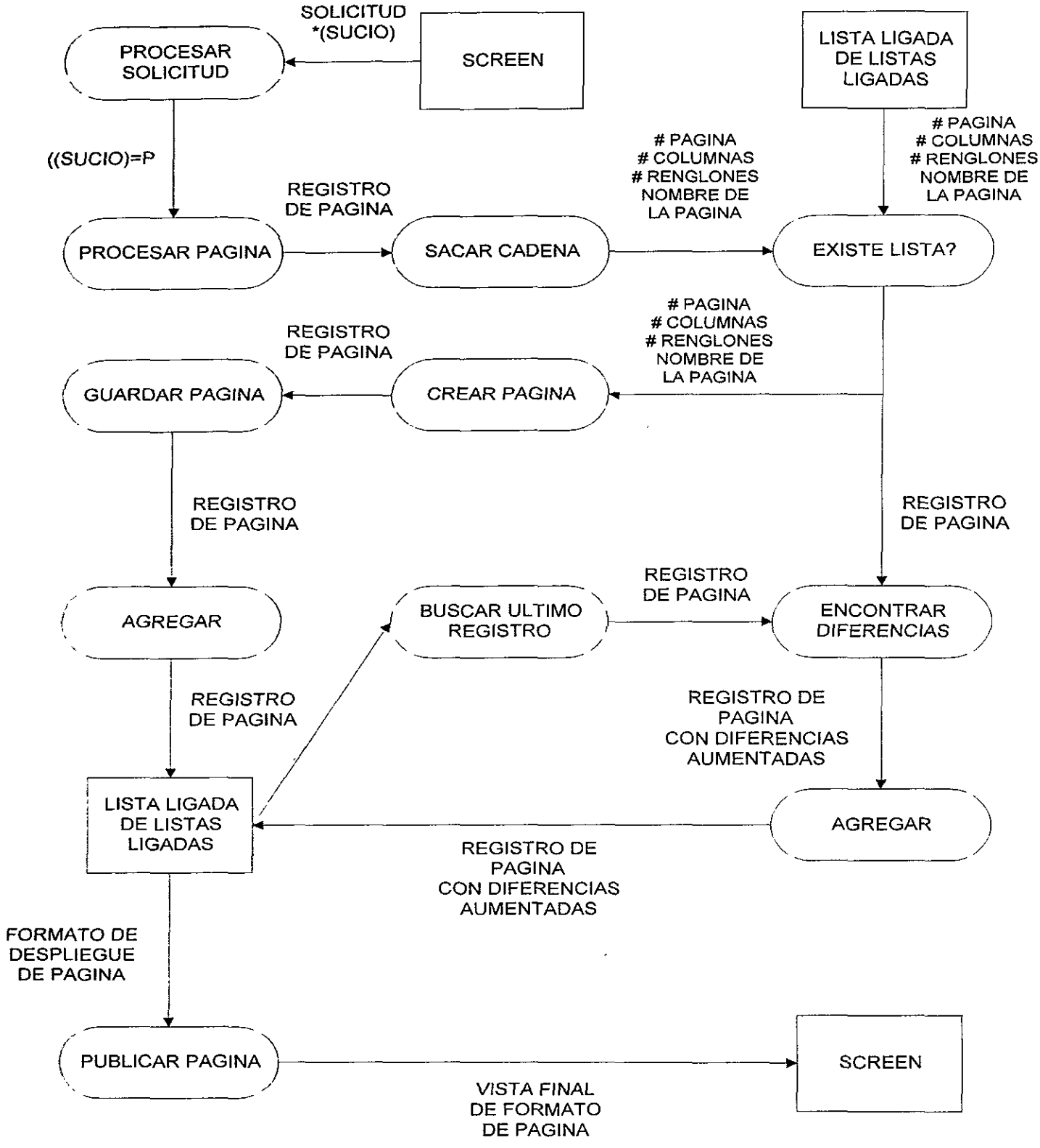
MODELO DE FLUJO DE DATOS DEL TRADUCTOR DE PROTOCOLOS



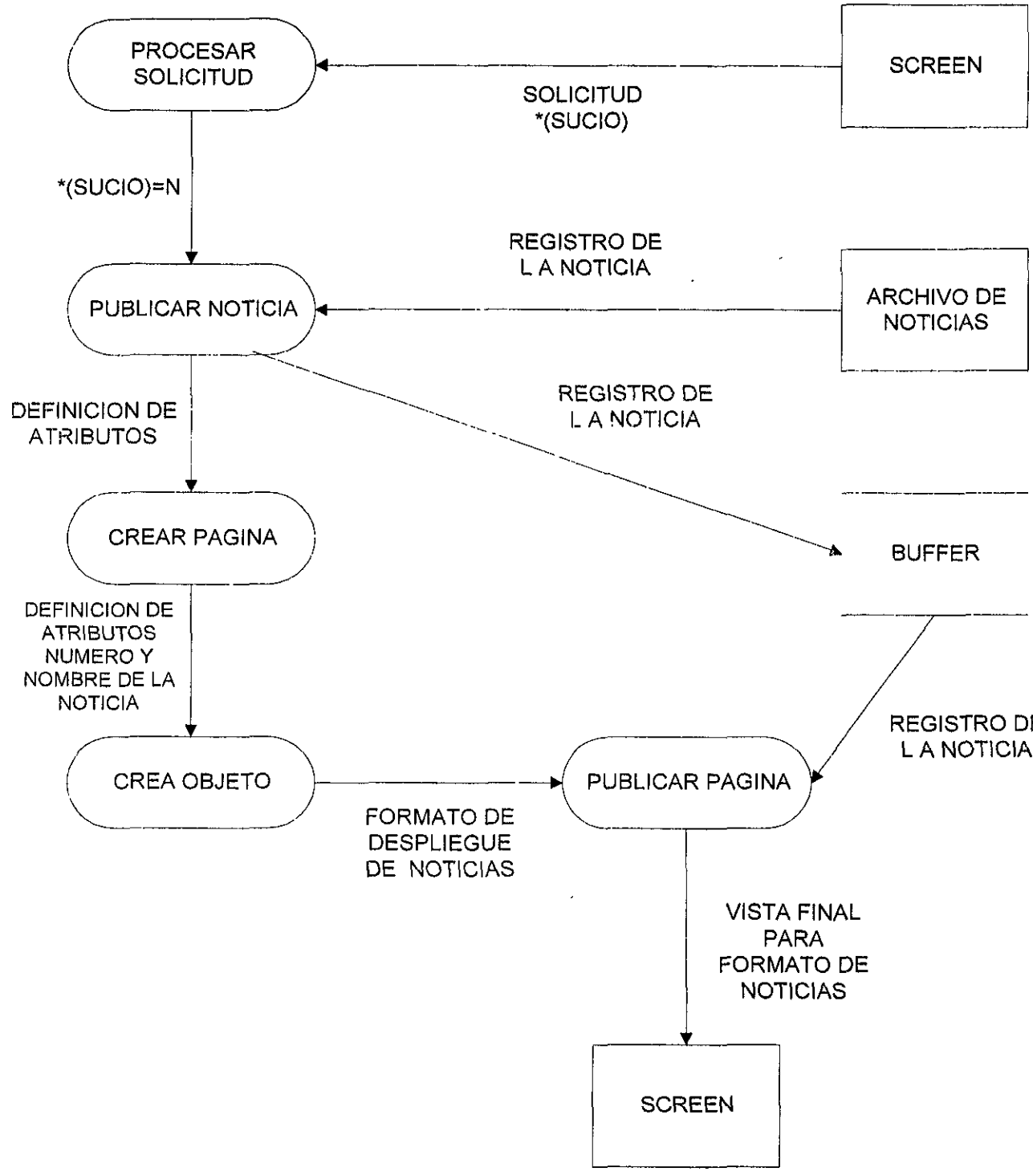
MODELO DE FLUJO DE DATOS DEL TRADUCTOR DE PROTOCOLOS



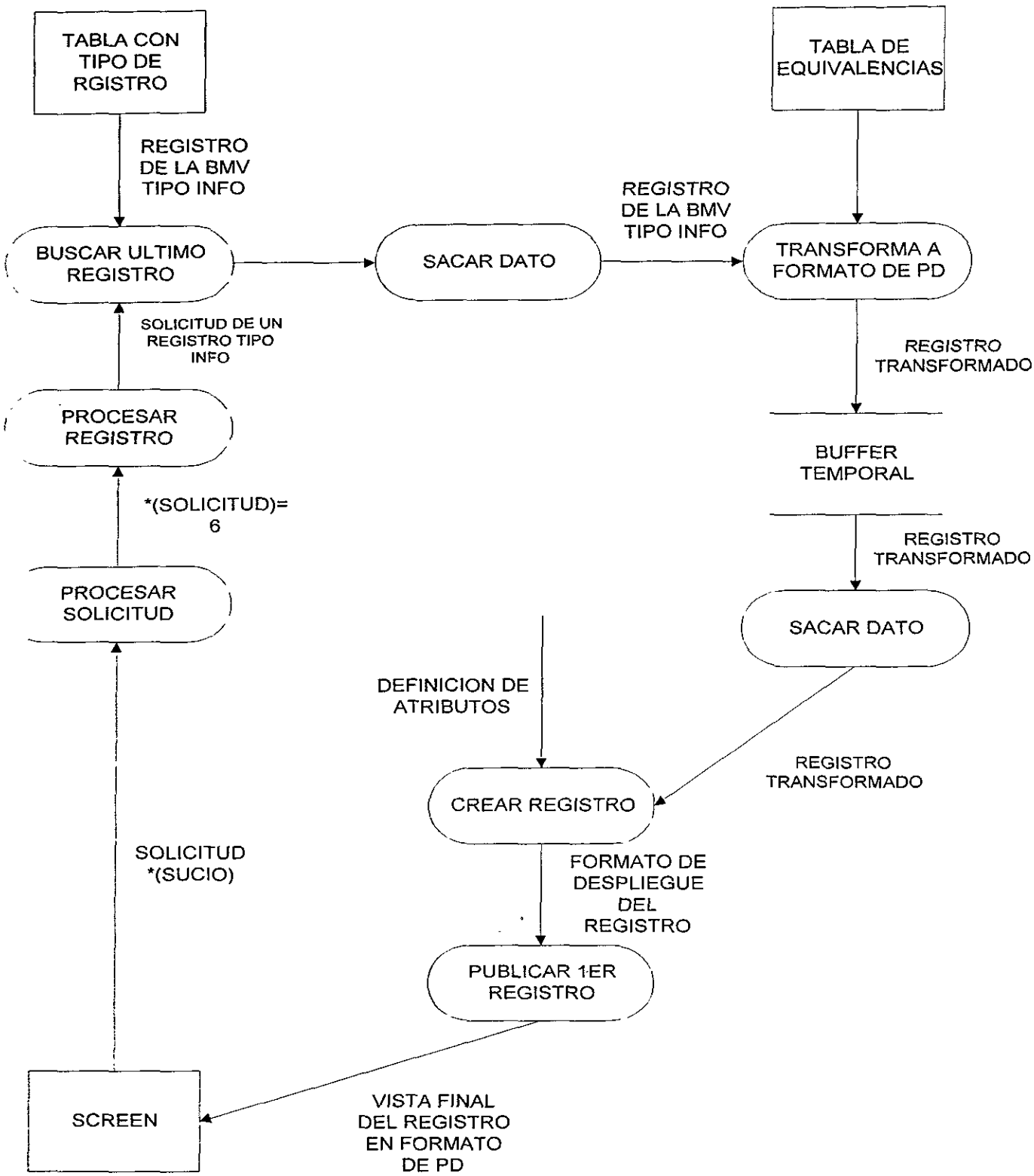
MODELO DE FLUJO DE DATOS DEL TRADUCTOR DE PROTOCOLOS



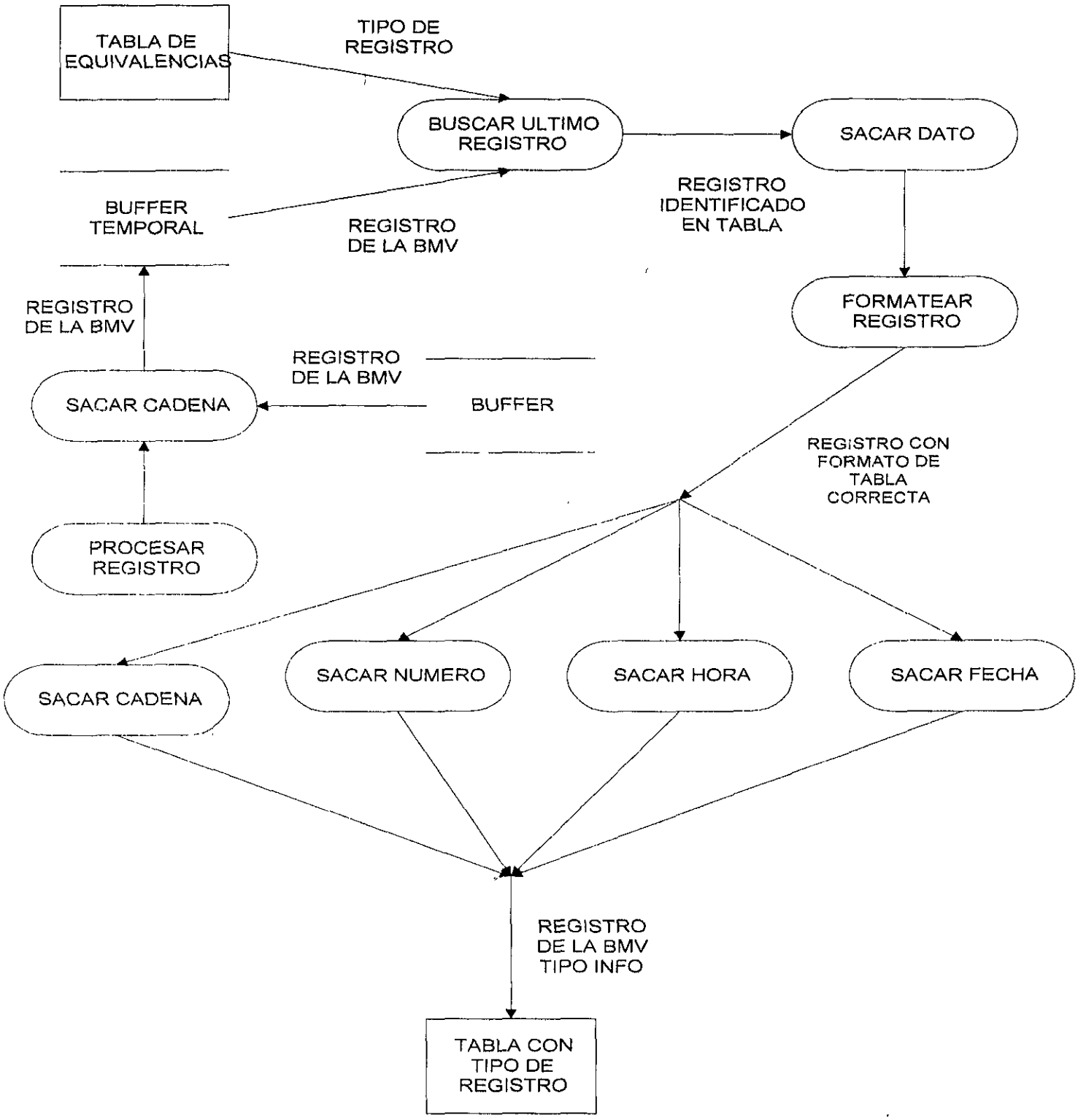
MODELO DE FLUJO DE DATOS DEL TRADUCTOR DE PROTOCOLOS



MODELO DE FLUJO DE DATOS DEL TRADUCTOR DE PROTOCOLOS



MODELO DE FLUJO DE DATOS DEL TRADUCTOR DE PROTOCOLOS



El proceso del diseño.

El diseño orientado al flujo de datos es un método de diseño arquitectónico que permite una cómoda transición desde el modelo de análisis a una descripción del diseño de la estructura del programa. La transición desde el flujo de información a una estructura se realiza en un proceso de cinco pasos: (1) se establece el tipo de flujo de información; (2) se indican los límites del flujo; (3) se convierte el diagrama de flujo de datos en la estructura del programa; (4) se define la jerarquía de control descomponiéndola mediante particionamiento; y (5) se refina la estructura resultante usando medidas y heurísticas de diseño.

Flujo de transformación.

En el modelo fundamental de sistema, la información debe introducirse y obtenerse del software en forma de datos escritos en un teclado, los tonos en una línea telefónica o las imágenes en el monitor. Tales datos externos deben convertirse a un formato interno para el procesamiento. La información entra en el sistema a lo largo de caminos que transforman los datos externos a un formato interno y se identifica como flujo de entrada. En el interior del software se produce una transición. La información entrante se pasa a través de un centro de transformación y empieza a moverse a lo largo de caminos que ahora conducen hacia la salida del software. Los datos que se mueven a lo largo de estos caminos se denominan flujo de salida. Cuando un segmento de un diagrama de flujo de datos presenta estas características, lo que tenemos presente es un flujo de transformación.

Flujo de transacción.

El flujo de información está caracterizado a menudo por un único elemento de datos, denominado transacción, que desencadena otros flujos de información a lo largo de uno de los muchos caminos posibles. El flujo de transacción se caracteriza por datos que se muevan a lo largo de un camino de entrada que convierte la información en una transacción. La transacción se evalúa y basándose en ese valor, se inicia el flujo a lo

largo de uno de muchos caminos de acción. El centro del flujo de información del que parten los caminos de acción se denominan centro de transacción.

Análisis de las transformaciones.

El análisis de las transformaciones es un conjunto de pasos de diseño que permite convertir un diagrama de flujo de datos, con características de flujo de transformación, en una plantilla predefinida para la estructura del programa.

Paso 1: Revisar el modelo fundamental del sistema. El modelo fundamental del sistema comprende el diagrama de flujo de datos de nivel 0 y la información que lo soporta. En realidad el paso de diseño empieza con una evaluación de la especificación del sistema y de la especificación de requisitos del software.

Paso 2: Revisar y refinar los diagramas de flujo de datos del software. La información obtenida de los modelos de análisis contenidos en la especificación de requisitos del software se refina para obtener mayor detalle.

Paso 3: Determinar si el diagrama de flujo de datos tiene características de flujo de transformación o de transacción. En general, el flujo de información dentro de un sistema puede representarse siempre como una transformación. Sin embargo, cuando se encuentra una característica obvia de transacción, se recomienda una estructura de diseño diferente.

Paso 4: Aislar el centro de transformación especificando los límites de los flujos de entrada y salida: Los límites del flujo de entrada y salida son interpretables. Es decir; los diseñadores pueden elegir puntos ligeramente como límites de flujo, se pueden obtener soluciones de diseño alternativas variando la posición de los límites de flujo.

Paso 5: Realizar una descomposición de primer nivel. La estructura del programa representa una distribución descendente del control. La descomposición en partes provoca una estructura de programa en la que los módulos del nivel superior realizan la

toma de decisiones y los módulos del nivel inferior realizan la mayoría del trabajo de entrada, cálculos y salida. Los módulos de nivel intermedio realizan algún control y cantidades moderadas de trabajo.

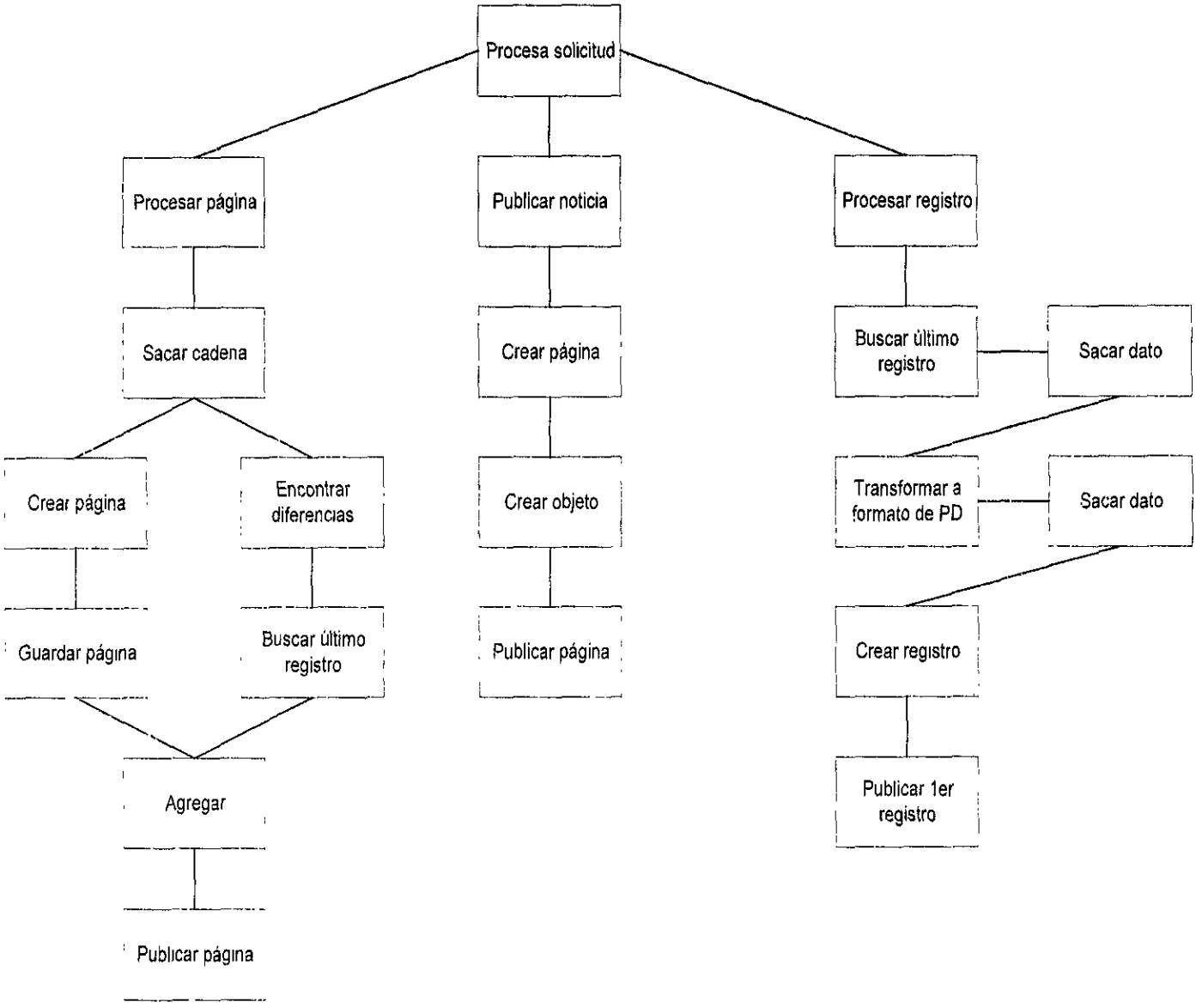
Paso 6: Realizar descomposición de segundo nivel. La descomposición de segundo nivel se realiza mediante la conversión de las transformaciones individuales de un diagrama de flujo de datos en los módulos correspondientes dentro de la estructura del programa. Empezando desde el límite del centro de transformación y moviéndonos hacia fuera a lo largo de los caminos de entrada, y luego de salida, las transformaciones se convierten en niveles subordinados de la estructura del software.

Paso 7: Refinar la estructura inicial del programa usando heurísticas para mejorar la calidad del software. Una primera estructura de programa siempre puede refinarse aplicando los conceptos de independencia de módulos. Los módulos se incrementan o reducen para producir una descomposición razonable, buena cohesión, acoplamiento mínimo y lo más importante, una estructura que se pueda implementar sin dificultad, probarse sin confusión y mantenerse sin problemas. Los refinamientos se rigen por consideraciones prácticas y por el sentido común.

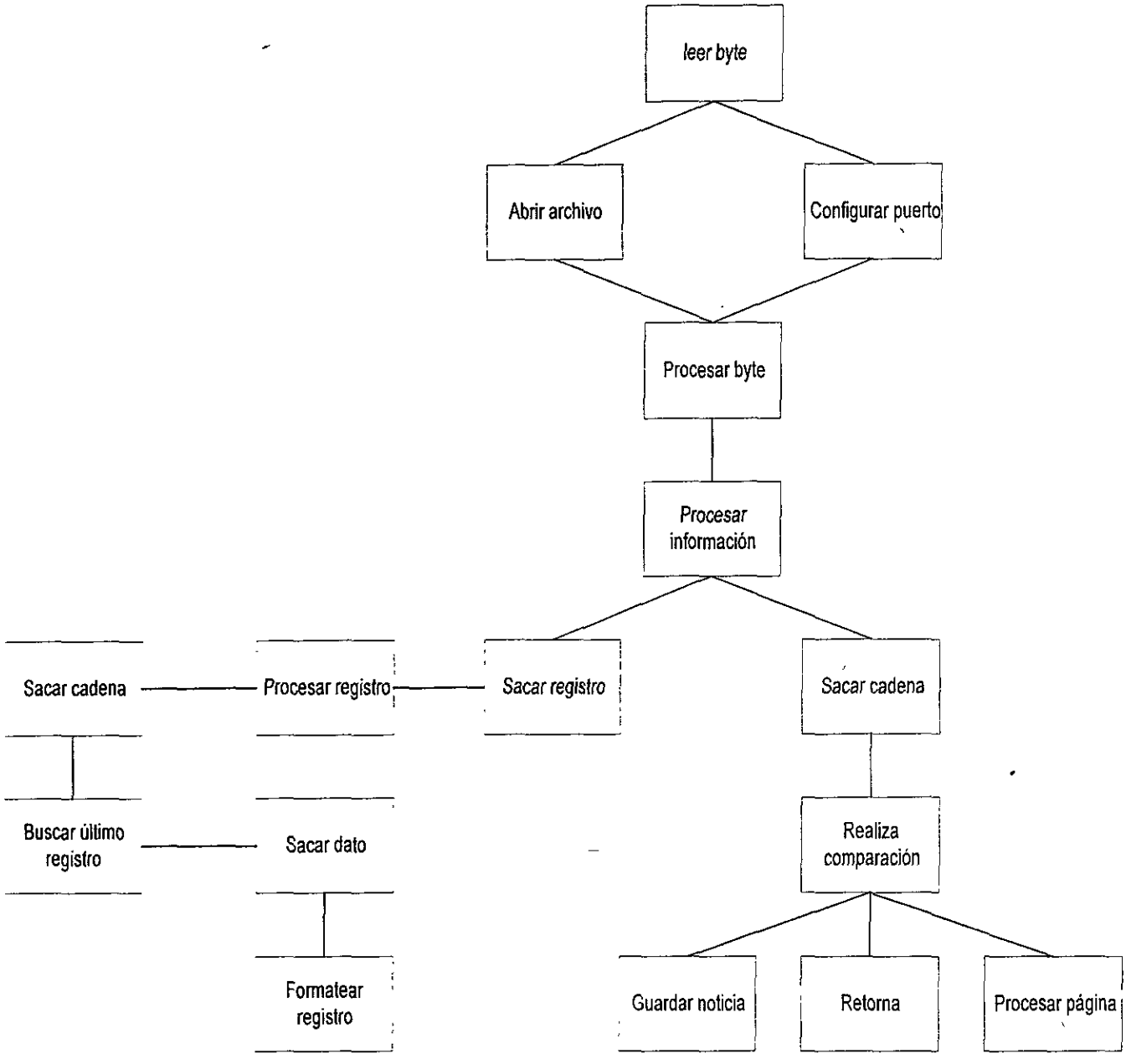
El objetivo de los siete puntos anteriores es desarrollar una representación general del software. Es decir, una vez que se ha definido la estructura, podemos evaluar y refinar la arquitectura del software viéndola en su conjunto.

El código del traductor de protocolo se encuentra listado en el apéndice I.

ESTRUCTURA REFINADA DEL TRADUCTOR DE PROTOCOLOS



ESTRUCTURA REFINADA DEL TRADUCTOR DE PROTOCOLOS



3.2. Diseño y Construcción de la Vista Final para el Usuario.

La vista final para el usuario será mostrada a través de la plataforma digital utilizada por el usuario, tomando esta como base para presentar la información que fue transformada por el traductor de protocolos.

El usuario esta familiarizado con esta plataforma y conoce las facilidades que le ofrece, la figura 3.2.1 muestra la plataforma digital con todas sus barras de herramientas, las cuales son de gran utilidad para el usuario para manipular la información que se le presenta en los registros financieros, y así poder presentarla de otras formas a través de gráficos o comparativos entre dos o más registros financieros.

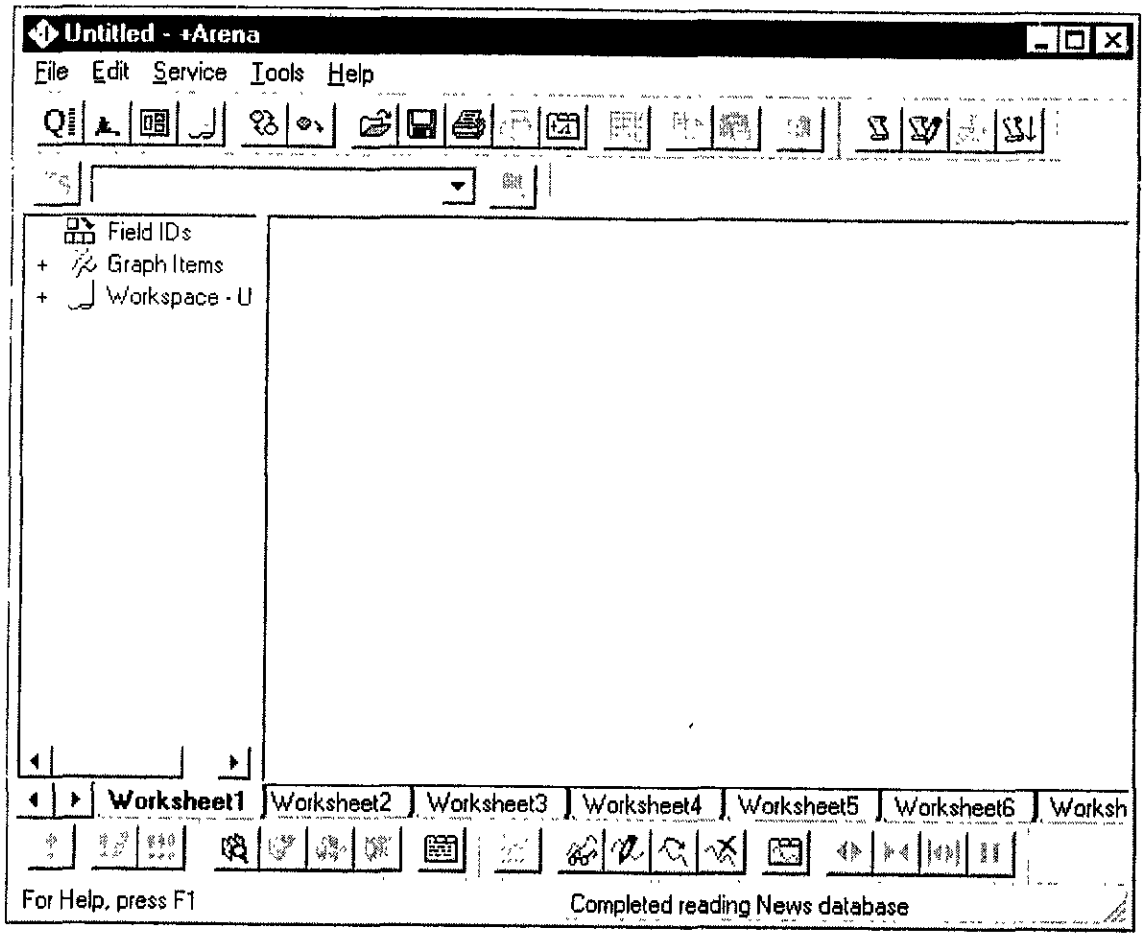


Figura 3.2.1. Plataforma digital manejada por el usuario para la visualización de la información financiera.

Al estar en ejecución el traductor de protocolos, recibiendo la información y transformando esta para la visualización del usuario, se activa una ventana que indica que el traductor esta trabajando, y el usuario puede proporcionar una solicitud para visualizar la información deseada, la figura 3.2.2 muestra esta situación.

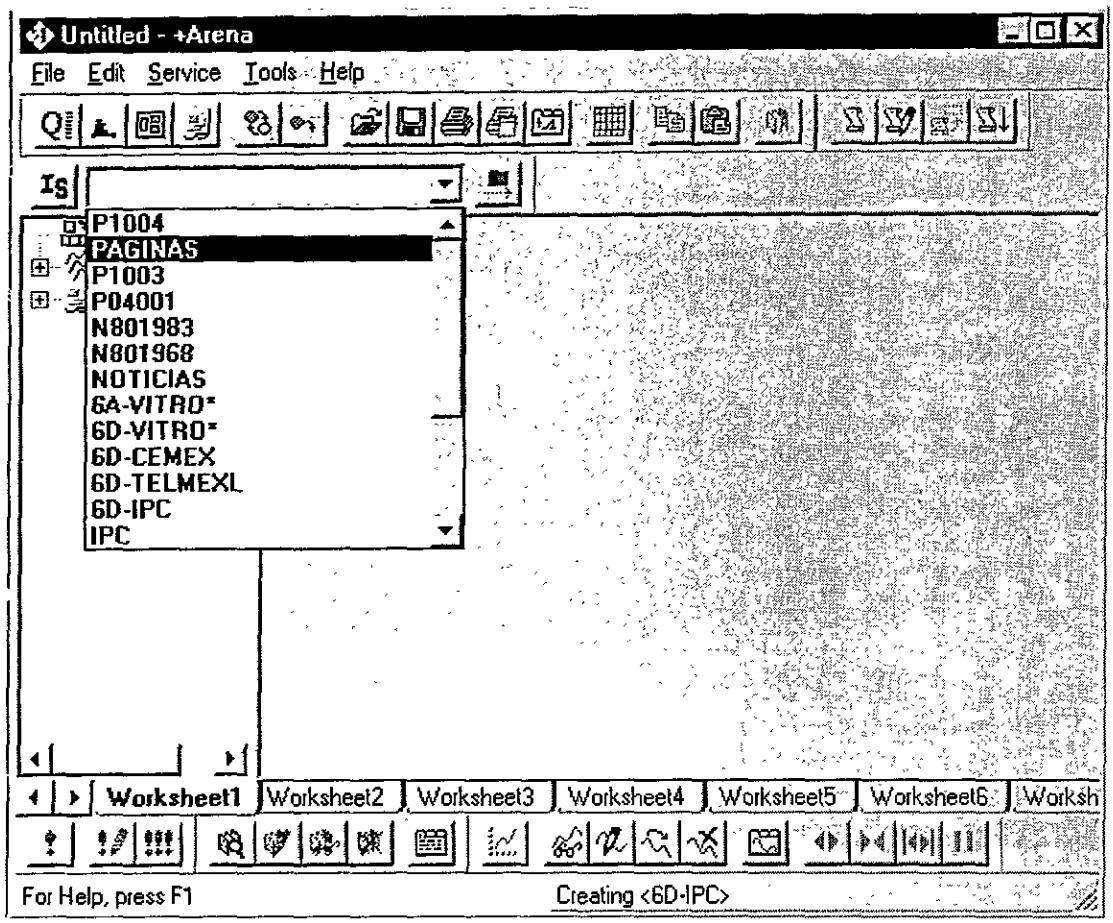


Figura 3.2.2. La plataforma digital indica al usuario cuando el protocolo esta activo para que pueda realizar solicitudes para visualizar información.

Las ultimas 20 solicitudes escritas por el usuario son guardadas en la plataforma digital para su rápida selección posteriormente.

El usuario para solicitar una noticia deberá escribir la palabra noticias, para que se despliegue una lista de noticias recibidas, la figura 2.2.3 muestra la pantalla que contiene la lista de noticias.

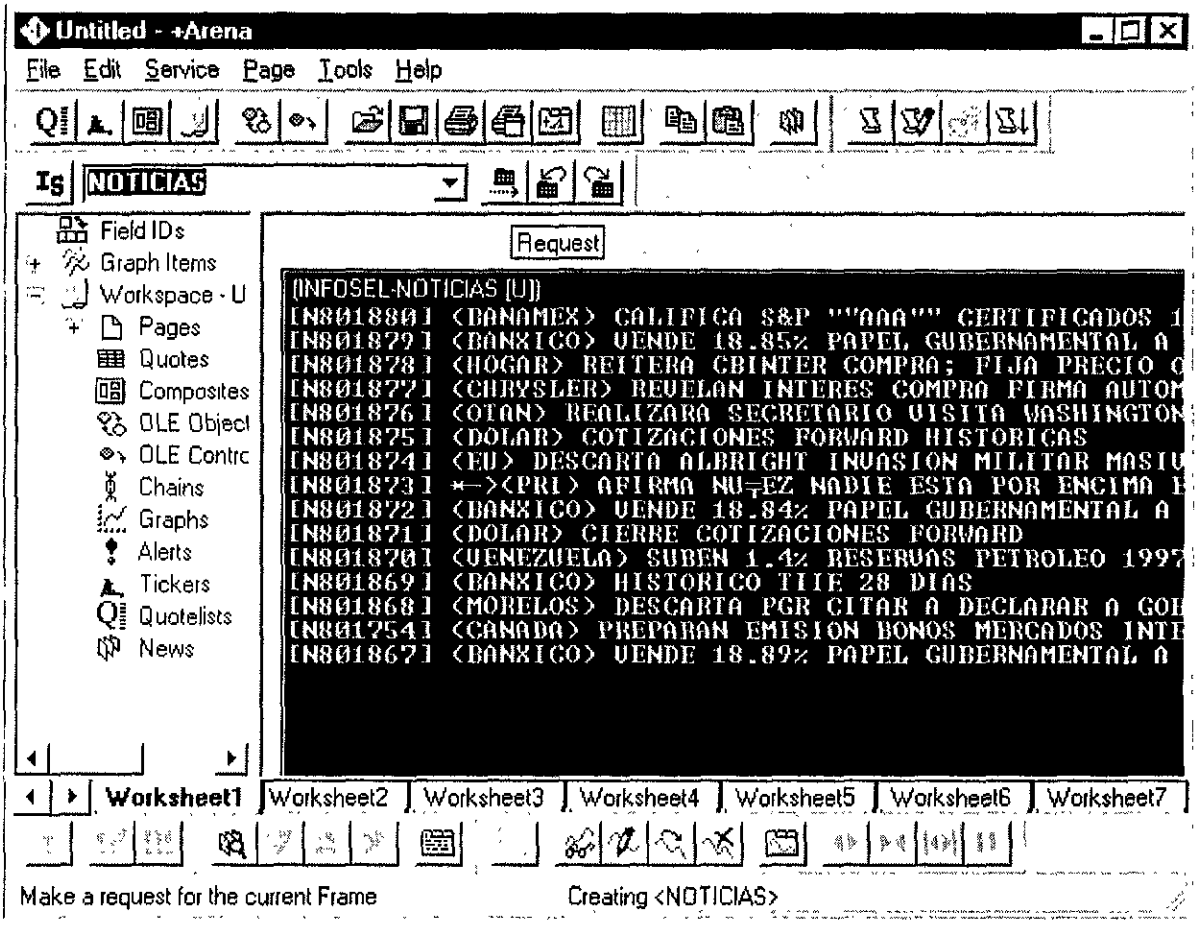


Figura 3.2.3. Solicitud de lista de noticias pedida por el usuario.

Una vez desplegadas las noticias el usuario deberá posicionarse en el número de la noticia deseada y posteriormente accionar dos veces el botón derecho del ratón. La figura 3.2.4 muestra esta situación.

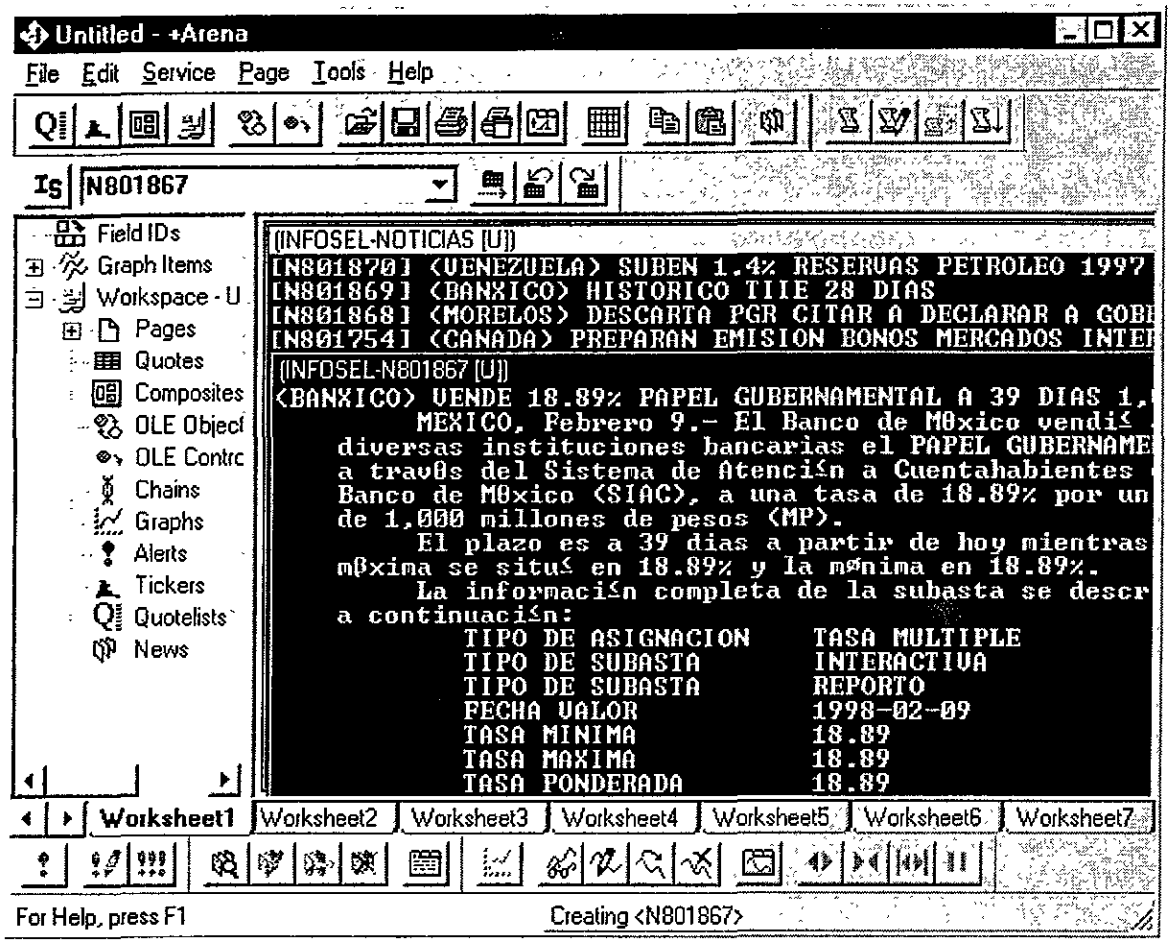


Figura 3.2.4. Seleccionando una noticia en particular.

De igual forma si el usuario desea ver páginas financieras, deberá escribir la palabra páginas, para visualizar la lista de páginas que arriban y que son transformadas por el traductor de protocolos.

La figura 3.2.5 nos muestra la solicitud escrita por el usuario para visualizar la lista de páginas.

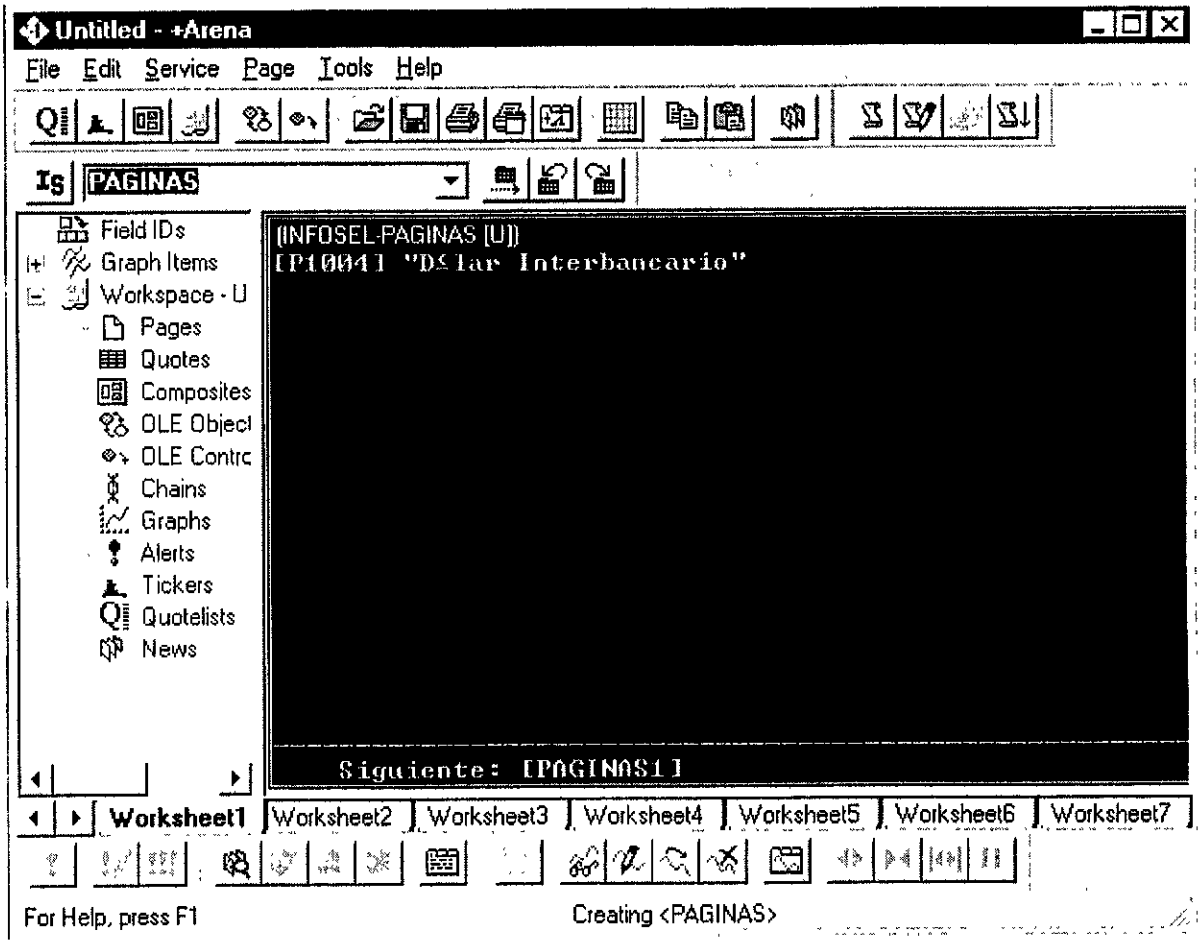


Figura 3.2.5. Solicitud para visualizar la lista de páginas financieras.

Si el usuario desea ver la página completa deberá posicionarse sobre el número de la página y accionar dos veces el botón derecho del ratón. La figura 3.2.6 muestra la selección de la página completa.

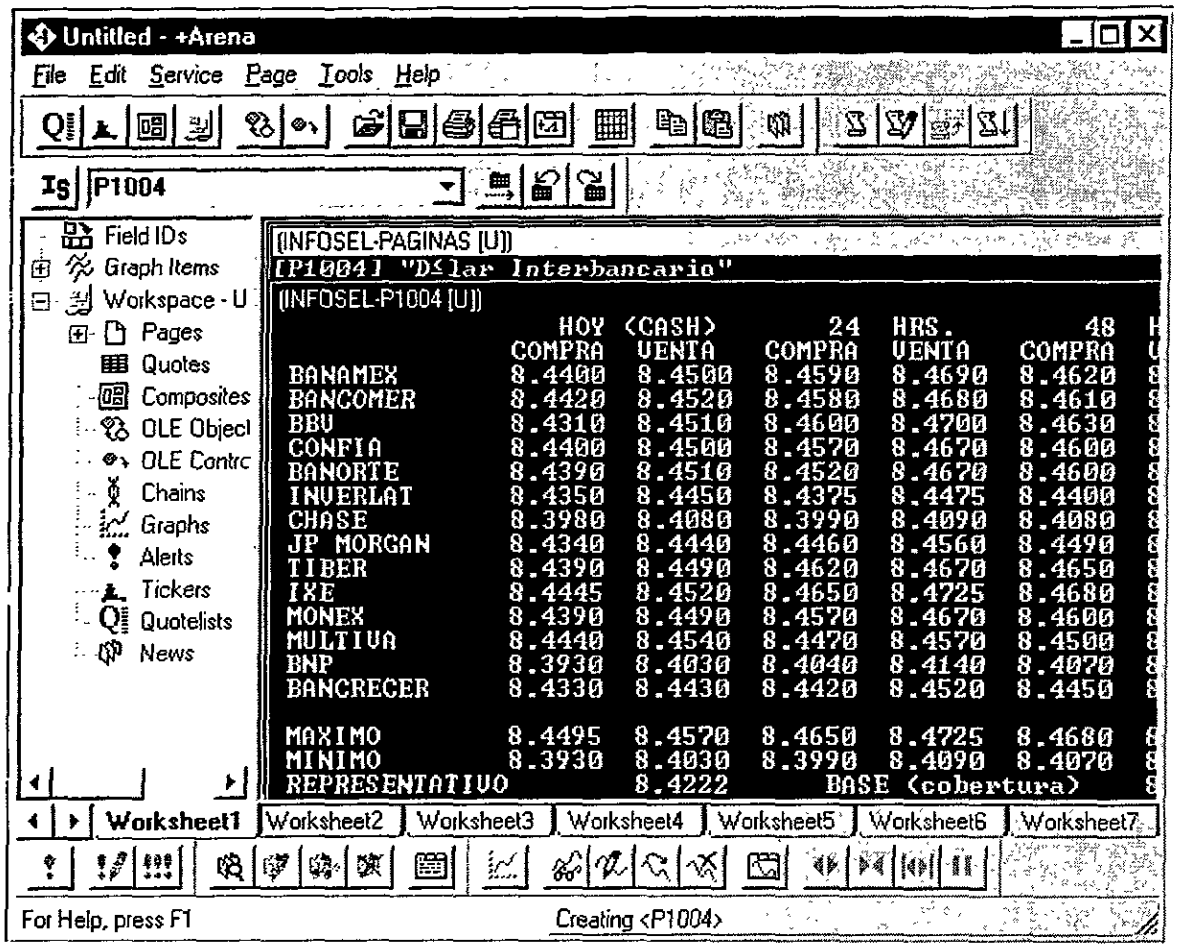


Figura 3.2.6. Selección de la página financiera.

El *usuario podrá visualizar* los registros financieros de la siguiente manera:

Deberá solicitar una Transacción de Acciones de Renta Variable escribiendo 6A-(nombre de la empresa). La figura 3.2.7 muestra un ejemplo de esto.

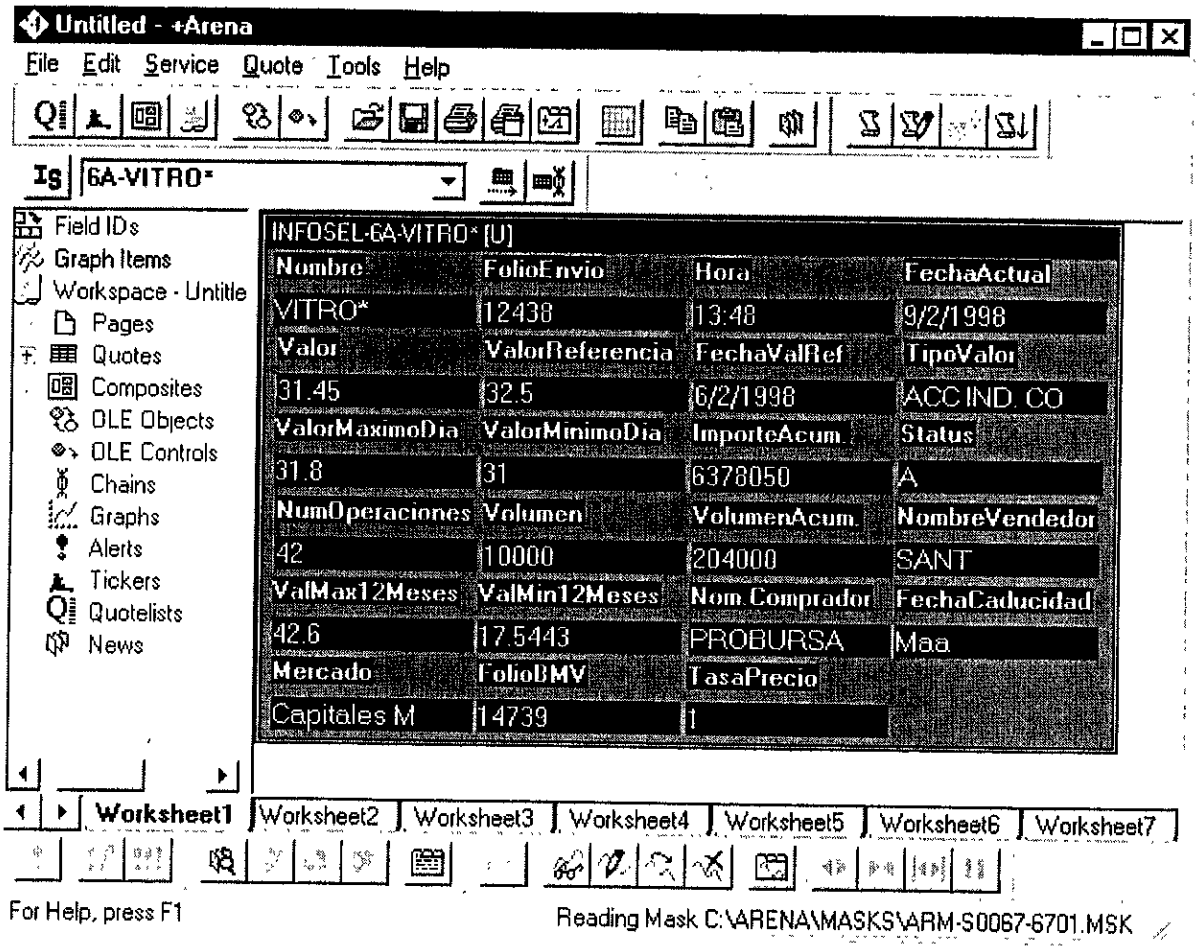


Figura 3.2.7. Pantalla con Transacción de Acciones de Renta Variable.

Deberá solicitar una Postura de Acciones de Renta Variable escribiendo 6B-(nombre de la empresa). La figura 3.2.8 muestra un ejemplo de esto.

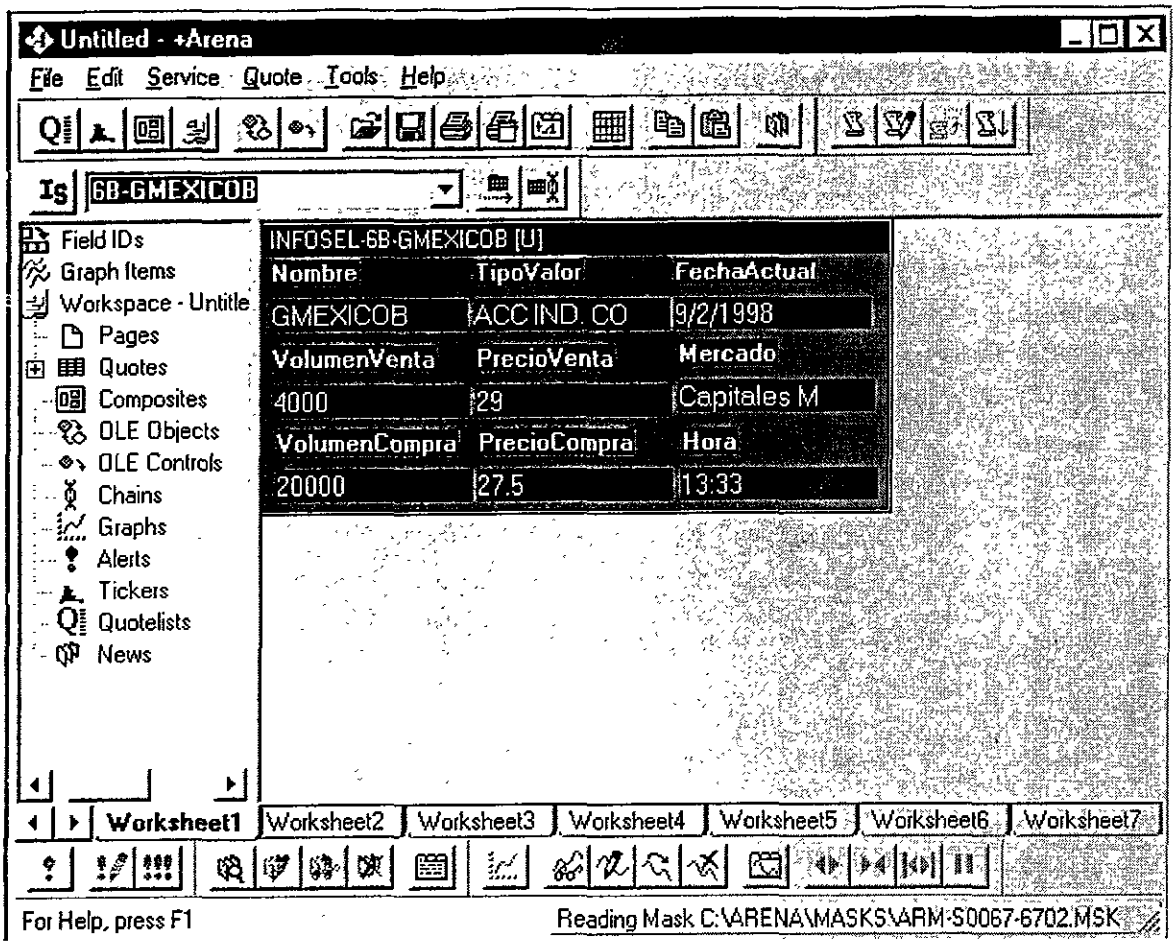


Figura 3.2.8. Pantalla con Postura de Acciones de Renta Variable.

Deberá solicitar una Evolución de Indices escribiendo 6D-(nombre de la empresa). La figura 3.2.9 muestra un ejemplo de esto.

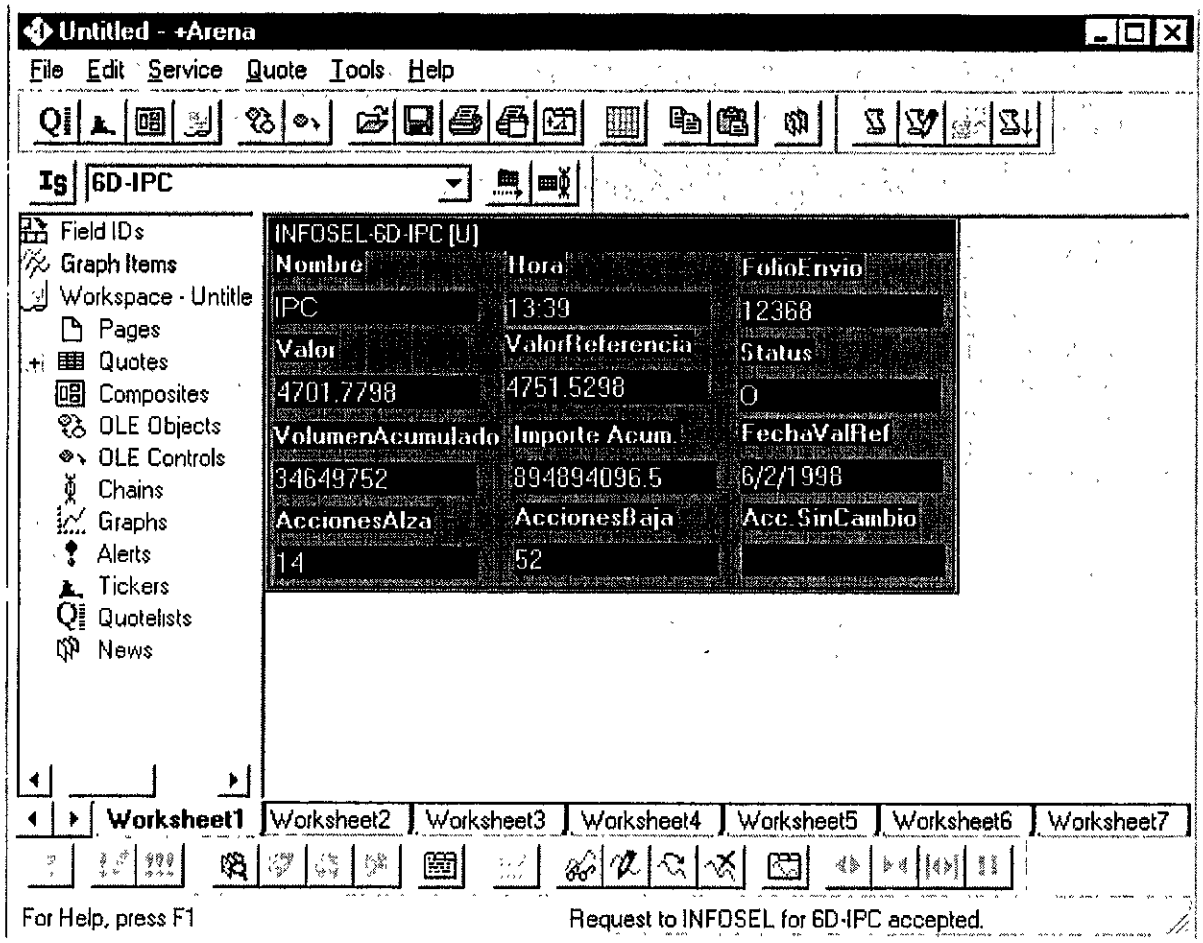


Figura 3.2.9. Pantalla con Evolución de Indices.

Deberá solicitar una Transacción de Warrants escribiendo 6F-(nombre de la empresa). La figura 3.2.10 muestra un ejemplo de esto.

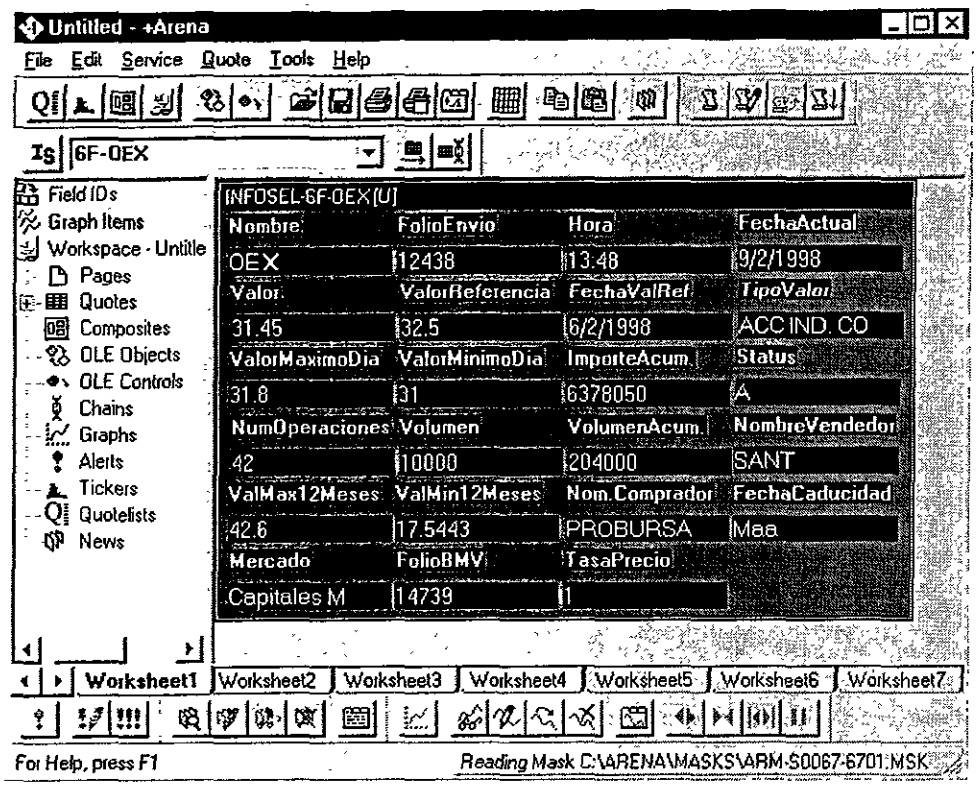


Figura 3.2.10. Pantalla de una Transacción de Warrants.

Por último deberá solicitar una Postura de Warrants escribiendo 6G-(nombre de la empresa). La figura 3.2.11 muestra un ejemplo de esto.

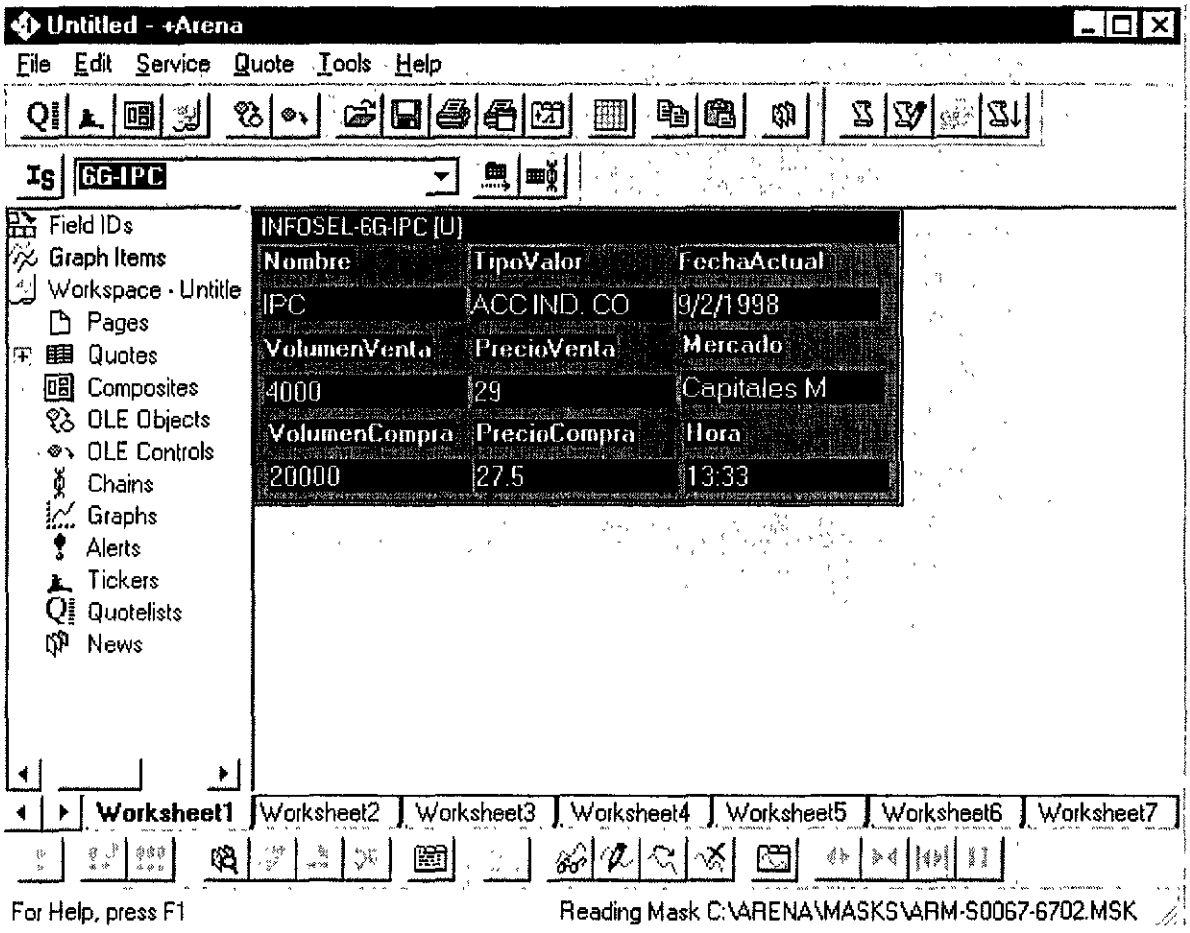


Figura 3.2.11. Pantalla de una Postura de Warrants.

3.3. Pruebas del Sistema.

La realización de pruebas de software, en el ciclo de vida del desarrollo de sistemas cobra gran relevancia, debido al costo en la inversión tanto de recursos humanos, económicos y de tiempo. Lo anterior no implica en forma principal, la detección de inconsistencias por mal funcionamiento en el sistema, sí no más bien radica, en el objetivo de entregar un producto de calidad, que ahorre costos de tiempo, esfuerzo y dinero, al efectuarse un menor mantenimiento del sistema, al momento de ser implantado.

La fase de pruebas se puede llevar mucho tiempo si es que no se hicieron eficientemente, el análisis, el diseño y la programación, por otro lado, si no se realizó un trabajo adecuado en estas fases puede volverse iterativo; las primeras pruebas muestran la presencia de errores, y las posteriores verifican si los programas corregidos funcionan correctamente.

En el momento en que se pone un sistema en operación, para que pueda ser utilizado por el cliente, se presentan diferentes dudas:

- ¿El sistema hace lo que se pidió?
- ¿Soportará la carga de trabajo planteada?
- ¿El sistema puede fallar?

Estas son las preguntas más comunes que se plantean al momento de instalar un sistema por primera vez. Es por lo anterior que se realiza el proceso de pruebas, las cuales disminuyen el riesgo de falla al operar el sistema.

La prueba de los programas es la técnica de confirmación del sistema, ésta se debe realizar antes de entregar el sistema al usuario. Las pruebas consisten en ejercitar el programa utilizando datos similares a los datos reales y observar los resultados e interpretar estos para detectar errores o insuficiencias en el sistema. Es importante comprender que las pruebas nunca demuestran que un programa esta correcto. Es probable que existan errores aún después de la prueba más completa. Un plan de

pruebas bien elaborado brindará como resultado la presencia de errores y de esta manera se podrá proceder a su depuración. El plan deberá indicar el nombre de la prueba, los datos de entrada, el objetivo de la prueba y los datos de salida, además de que es recomendable que se lleve a cabo con personal que no haya participado durante la programación, pero que conozca la operación del negocio que utilizará el sistema.

3.3.1. Flujo de Información para las Pruebas.

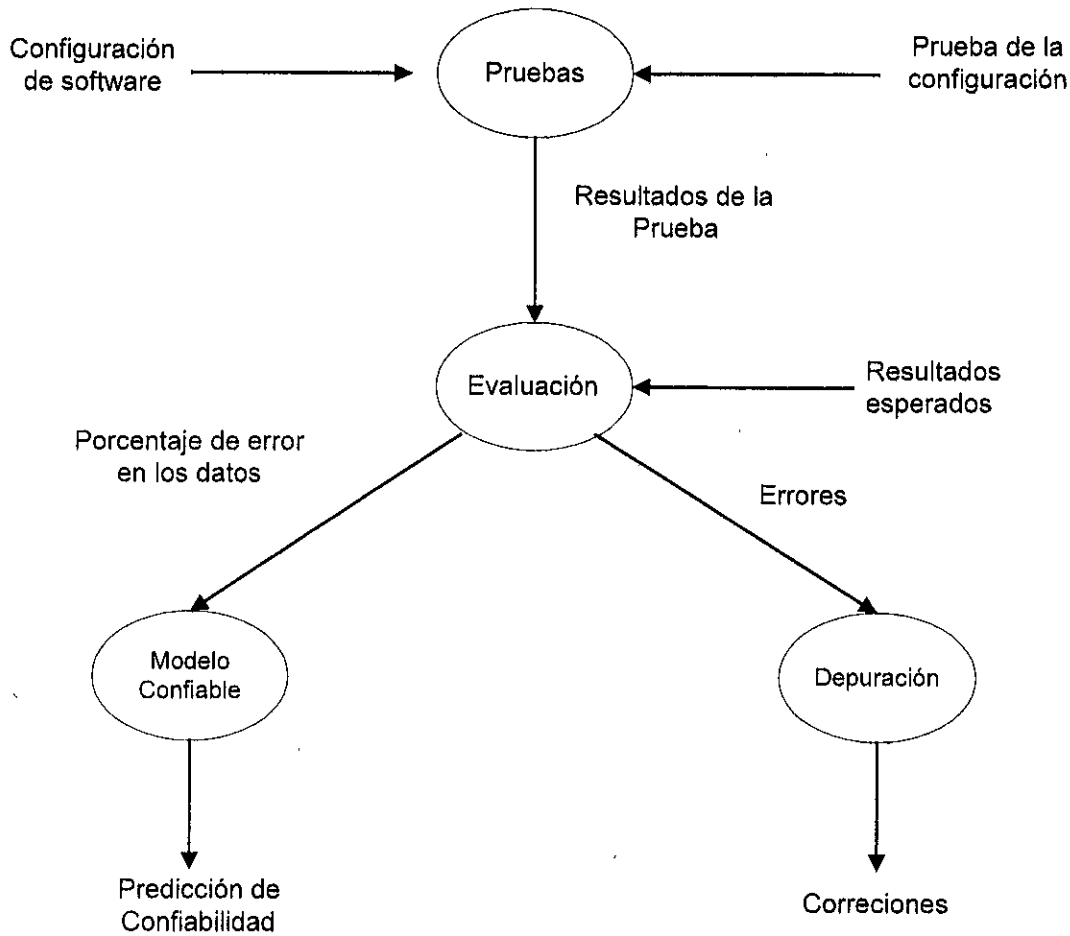


Figura 3.4.1. Flujo de información para las pruebas.

En la figura anterior se tienen dos entradas: La configuración del software que incluye la especificación de requisitos, la especificación del diseño y el código fuente; La

configuración de Prueba que incluye un plan y un procedimiento de prueba así como los casos de prueba a utilizar y los resultados esperados. Se realizan las pruebas y se evalúan los resultados esperados contra los generados. Cuando se descubren errores se inicia un proceso de depuración que tiene como finalidad realizar las correcciones pertinentes.

A medida que se recopilan y se evalúan los resultados de la prueba se puede determinar una medida cualitativa de calidad y fiabilidad del software. Si se encuentran frecuentemente errores que requieren modificaciones en el diseño, la calidad y fiabilidad del software quedan en entre dicho, del tal forma que es necesario seguir realizando pruebas. Si por el contrario, el funcionamiento del software parece ser el correcto y los errores que se encuentran son menores, se pudiese pensar que la calidad y fiabilidad del software son aceptables, o que las pruebas fueron inadecuadas, ya que no permitieron descubrir errores importantes. Así mismo, si durante la prueba no se descubren errores, quedará la sospecha de que las pruebas no fueron adecuadas y que el software puede estar defectuoso, para lo cual se considera que estos defectos pudiesen ser descubiertos por el usuario y deberán ser corregidos en la fase de mantenimiento o garantía.

3.3.2. Tipos de Prueba.

- Prueba Funciónal: Esta es la forma más común de prueba; su propósito es asegurar que el sistema realiza sus funciones normales de manera correcta. Así, los casos de prueba se desarrollan y se alimentan al sistema. Las salidas son examinadas para ver si son correctas.
- Prueba de Recuperación: El propósito de este tipo de prueba es asegurar que el sistema pueda recuperarse adecuadamente de diversos tipos de fallas. Esto es de particular importancia en los sistemas en línea de dimensiones grandes, así como sistemas que trabajen en tiempo real.
- Prueba de Desempeño: Su propósito es asegurar que el sistema pueda manejar el volumen de datos y transacciones de entrada especificados en el modelo de

implantación del usuario, además de asegurar que tenga el tiempo de respuesta requerido.

- Prueba Exhaustiva: Esta prueba consiste en generar casos de prueba que permitan cubrir cada entrada posible así como cada una de las combinaciones posibles de las situaciones que el sistema pudiera enfrentar, de tal forma que se puede asegurar un comportamiento adecuado

3.3.3. Técnicas de Prueba.

En cuanto a las técnicas para las pruebas del software existen dos enfoques denominados pruebas de caja negra y pruebas de caja blanca.

- Prueba de Caja Negra: Esta prueba pretende demostrar que la entrada se acepta en forma adecuada y que se produce una salida correcta, así como que la integridad en la información externa se mantiene. La prueba de caja negra examina algunos aspectos del modelo fundamental del sistema sin tomar mucho en cuenta la estructura interna lógica del sistema. Esta prueba pretende determinar:
  a) Funciónes incorrectas o ausentes.
  b) Errores de interface
  c) Errores en estructura de datos o en accesos a base de datos externas.
  d) Errores de rendimiento.
  e) Errores de inicialización y de terminación.

- Prueba de Caja Blanca: Esta prueba se basa en un minucioso examen de los detalles procedurales. Se comprueban los caminos lógicos del software, proponiendo casos de prueba que provoquen que se cumplan conjuntos especificos de condiciones examinando el comportamiento del programa en varios puntos para determinar si el resultado real coincide con el esperado. El inconveniente de la prueba es que para algunos casos es complicado probar todos los caminos lógicos. Algunas características que se deben cumplir son las siguientes:

a) Garantizar que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo.

b) Ejercitar todas la decisiones lógicas, verdaderas o falsas.

c) Ejecutar todos los ciclos en sus límites y con sus límites operacionales.

d) Ejercitar las estructuras internas de datos para asegurar su validez.

3.3.4. Estrategias para Pruebas.

1. Pruebas de Unidad.

   Las partes que conforman la prueba de unidad son las siguientes:

   - Interface. Se comprueba que la información fluye de forma adecuada hacia y desde la unidad o programa que está siendo probado. Se examinan las estructuras de los datos locales para asegurar que los datos que se mantienen temporalmente conservan su integridad durante todos los pasos de ejecución. Se prueban las condiciones límite para asegurar que el programa funciona correctamente en los límites establecidos como restricciones de procesamiento. De esta manera se prueban todos los caminos de la estructura de control con el fin de asegurar que todas las instrucciones del programa se ejecuten al menos una vez.
   - Manejo de errores. Entre los errores potenciales que se comprueban pueden ser los siguientes.
     - ◊ Descripción ininteligible del error.
     - ◊ El error señalado no corresponde con el error encontrado.
     - ◊ La condición de error hace que intervenga el sistema antes que el mecanismo de manejo de errores.
     - ◊ El procesamiento de la condición excepcional es incorrecto.
     - ◊ La condición de error no proporciona suficiente información para ayudar a la localización del error.

- La prueba de límites. Verifica los valores máximos y mínimos permitidos, los valores de los datos por debajo y por encima de los máximos y mínimos son apropiados para descubrir estos errores.

2. Pruebas de Integración.

Una vez que se han realizado las pruebas de unidad o de programas por separado, el paso siguiente es ponerlos ahora en interacción, es decir, realizar la prueba de integración. Para esta prueba se deben considerar los siguientes puntos:

- Un programa no debe tener efectos adversos o inadvertidos sobre otros.
- Las funciones cuando se combinan, pueden no producir la función principal deseada.
- Las estructuras de datos globales pueden presentar problemas.
- Los datos se pueden perder en alguna interface

Para este tipo de pruebas, existen dos formas de atacarlas: En forma ascendente y en forma descendente. El enfoque ascendente empieza por probar módulos individuales pequeños separadamente, luego los módulos individuales se combinan para formar unidades cada vez más grandes que se probarán en conjunto; esto se conoce como prueba de subsistemas. Finalmente, todos los componentes del sistema se combinan para probarse, esto se conoce prueba de sistema y suele estar seguido por las pruebas de aceptación, estas se realizan mediante el empleo de datos generados por la organización encargada de construir el sistema. La prueba de aceptación del sistema se efectúa con datos reales, que a menudo descubren errores en la definición de requerimientos del sistema.

El enfoque de prueba descendente empieza con un esqueleto del sistema, es decir, la estrategia de prueba supone que se han desarrollado los modelos ejecutivos de alto nivel del sistema, pero que los de bajo nivel existen sólo como módulos que no procesan nada. La selección de la estrategia de integración depende de las características del software y, a veces, del plan del proyecto. En términos generales se puede utilizar un planteamiento combinado.

3. Pruebas de Validación. Después de la prueba de integración el paso siguiente es la prueba de validación. La prueba de validación es la que se obtiene cuando el software desarrollado funciona de acuerdo a las expectativas razonables del cliente. La validación del software se obtiene aplicando una serie de pruebas que demuestran la conformidad con los requerimientos.
4. Pruebas del Sistema. Para esta parte se deberán considerar los siguientes puntos:
   - Prueba de recuperación. El propósito de este tipo de prueba es asegurar que el sistema pueda recuperarse adecuadamente de diversos tipos de fallas. Las pruebas de recuperación pueden requerir que el equipo que realiza el proyecto simule o provoque fallas de hardware, de corriente, en el sistema operativo, etc. Este tipo de prueba puede ser manual o automática. Si la recuperación es automática se deberá evaluar que el sistema se reinicie correctamente, si la recuperación requiere de la intervención humana, hay que evaluar los tiempos medios de recuperación para determinar si esta dentro de los límites aceptables.
   - Prueba de Seguridad. Consiste en verificar los mecanismos de protección incorporados al sistema. Estos mecanismos tienen la finalidad de proteger al sistema contra acciones impropias de gente que traten de perjudicar u obtener información en forma ilícita.
   - Prueba de resistencia. Ejecuta el sistema de forma que demande recursos en cantidades anormales, es decir, diseñar pruebas especiales que generen 10 interrupciones por segundo, pero sólo una o dos son normales, incrementar la frecuencia de datos de entrada en un orden de magnitud con el fin de comprobar como responden las funciones de entrada, ejecutar casos de prueba que puedan dar problemas con el esquema de gestión de memoria y diseñar casos de prueba que produzcan excesivas búsquedas de información.
   - Prueba de rendimiento. Está diseñada para probar el rendimiento del sistema en tiempo de ejecución dentro del contexto de un sistema integrado. Las pruebas de rendimiento a menudo se realizan de forma paralela con las pruebas de resistencia. La instrumentación consiste en monitorear los intervalos de ejecución, los sucesos ocurridos y muestras de los estados de la máquina en

funcionamiento normal. De esta forma se pueden descubrir situaciones que lleven a degradaciones y posibles fallos del sistema.

3.3.5. Pruebas y Evaluación de los Módulos del Sistema y Rutinas de Diagnóstico.

El objetivo general de las pruebas orientadas a objetos (encontrar el número de errores máximo con el mínimo de esfuerzo) es idéntico de las pruebas del software convencional. Pero la estrategia y táctica para las pruebas orientadas a objetos difieren significativamente. La visión de la prueba se amplia hasta incluir la revisión de los modelos de análisis y diseño.

Como los modelos de análisis y diseño y el código fuente resultante, están semánticamente enlazados, las pruebas (en forma de revisiones técnicas) comenzaron durante estas actividades de ingeniería. Por esta razón, la revisión de los modelos clase-responsabilidades-colaboraciones, objeto-relación y objeto-comportamiento puede verse como una primera etapa de las pruebas.

Una vez que fue realizada la programación orientada a objetos, pruebas de unidad fueron aplicadas a cada clase. En las pruebas (estas se diseñaron con el objetivo de que aplicaran funciones relevantes a la operación del sistema) se usaron dos métodos: pruebas aleatorias y de partición. Cada uno de estos métodos ejercito las operaciones encapsuladas por la clase.

Las pruebas de integración se realizaron usando estrategias basadas en relaciones o de uso. Las pruebas basadas en las relaciones integraron el conjunto de clase que colaboran para dar respuesta a una entrada o evento. Las pruebas basadas en uso evaluaron el sistema por capas, comenzando con aquellas clases que no hacen uso de clases servidores.

Se aplicaron además algunos métodos de prueba de caja negra lo cuales se centran en los requisitos funcionales del software, esto es, permiten el obtener el conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa..

Todos estos métodos fueron aplicados en forma aleatoria tanto a las clases como en sus relaciones y al comportamiento de las mismas (modelo de objetos, dinámico y funcional), tratando de usar el mayor número posible para otorgar una mayor diversidad de pruebas y como consecuencia obtener una mayor probabilidad de encontrar anomalías en la ejecución del software.

Hemos seguido el estándar como documentación general para el diseño orientado a objetos de casos de prueba:

1. Cada paso de prueba esta identificado unívocamente y asociado explícitamente con la clase a probar.
2. Se determina el Objetivo de la Prueba.
3. Esta establecido el propósito de la prueba. Se desarrolla una lista de pasos de prueba, la cual contiene:
   - La lista de estados especificados para el objeto a probar.
   - La lista de mensajes y operaciones a ejercitar como consecuencia de la prueba.
   - Una lista de excepciones que pueden ocurrir al probar el objeto.
   - Una lista de condiciones externas (por ejemplo, cambios en el entorno externo software que debe existir para conducir propiamente la ejecución de la prueba).
   - Información suplementaria que ayuda en la comprensión o implementación de la prueba.
4. Evaluación de la Prueba

Pruebas Aleatorias para Clases Orientadas a Objetos.

Este método consiste básicamente de ejercitar las operaciones (servicios) que conforman las clases mediante una secuencia mínima , esto implica, que pudiera

ocurrir una amplia variedad de comportamientos entre las secuencias planteadas. Estas se aplican en orden aleatorio para ejercitar las historias de vida de diferentes instancias de una clase. Ahora representaremos ejemplos de la Clases involucrada en el Sistema Traductor de Protocolo.

PRUEBA 1

1. **Identificación Única.**

   Prueba ElementoLista.Aleatoria.01

2. **Objetivo.**
   Comprobar que los elementos de una lista son realmente almacenados en memoria y destruidos cuando ya no se requieren.

3. **Pasos de la Prueba.**

   - **Estados a Probar.** Los estados que permiten la manipulación de los elementos de una lista son:

     La ListaLigada donde efectuaremos las operaciones de inserción y borrado de elementos.
     El identificador único del elemento dentro de la lista.,

     El tamaño de la memoria reservada para el elemento,

     El tipo de elemento que estamos almacenando,

     Los apuntadores a los elementos de la lista (anterior y siguiente) al elemento a que se va a insertar o eliminar y la información que lo compone.
   - **Secuencia de Operaciones.** A continuación presentamos la secuencia que fue aplicada a la clase ElementoLista:

     CrearElemento

     ImprimirElemento

     BorrarElemento

     ImprimirElemento
   - **Excepciones.** Debe existir la lista ligada sobre la cual se efectuarán las operaciones, además de estar definida la posición del elemento a insertar, borrar o imprimir y solo en el caso de insertar la información que lo conforma, esto de forma indirecta, pues la clase no tiene control de esto.

- **Condiciones externas.** Debe existir la memoria suficiente en el servidor para el elemento a insertar.
- **Información Suplementaria.** Estas pruebas deberán garantizar el correcto uso de la memoria por parte del sistema, además de soportar aquellos problemas de contexto que pudieran existir por la falta de inicialización de las condiciones iniciales de la instancia de la clase.

4. **Evaluación.** Pudimos corregir problemas como:
   - Mensajes por la imposibilidad de crear memoria para el elemento garantizando la operación del sistema y notificando al usuario para su corrección.
   - Garantizar la correcta inicialización de las condiciones iniciales de la instancia de la clase.
   - Destrucción del elemento y el correcto ligado de los elementos en la lista resultante.

PRUEBA 2

1. **Identificación Única.**

   Prueba Puerto.Aleatoria.01

2. **Objetivo.**

   Comprobar el correcto funcionamiento del puerto serial del servidor.

3. **Pasos de Prueba.**
   - Estados a Probar. La actividad del puerto mediante la lectura y escritura de mensajes al mismo.
   - Secuencia de Operaciones. A continuación presentamos la secuencia que fue aplicada a la clase ListaLigada:

     Secuencia 1

     Configuración

     AbrirPuerto

     HayByte

LeerByte

CerrarPuerto

Secuencia 2

Configuración

AbrirPuerto

EsperarByte

EscribirByte

CerrarPuerto

- Excepciones. La prueba no podrá realizarse si no existe flujo de mensajes por el puerto Serial
- Condiciones Externas. Es necesario que se tenga la computadora que contiene la fuente de información, así como el software que la filtra y la envía por el puerto serial hacia el servidor del traductor de protocolo.
- Información Suplementaria. Debemos revisar las especificaciones de los registros proporcionados por la fuente, para esto podemos ayudarnos del capitulo 2. en el cual se detallan.

4. **Evaluación.** Al final de las pruebas aseguramos la lectura de información por el puerto serial.

Pruebas de Partición al Nivel Clase

La prueba de particiones reducen el número de casos de prueba necesarios para ejecutar la clase. Las entradas y salida están agrupadas en categorías, y los casos de prueba están diseñados para ejercitar cada categoría. Existen dos tipos de partición en estas pruebas. Las particiones basadas en estados, las cuales agrupan las operaciones basándose en su habilidad para cambiar el estado de la clase. La partición basada en atributos agrupa las operaciones basándose en los atributos que se usan. Para los ejemplos que a continuación mostramos utilizamos el método de partición basa en atributos.

PRUEBA 3

1. **Identificación Única.**

   Prueba ListaLigada.Aleatoria.01

2. **Objetivo.**

   Comprobar el correcto uso de la información financiera alojado en la memoria de la computadora mediante el uso de listas circulares doblemente ligadas, para la cual la clase cuenta con servicios que permiten la inserción, borrado y consulta de los elementos que la componen.

3. **Pasos de Prueba.**

- Estados a Probar. Comprobar que los atributos que componen la lista brinden información correcta:

  Total número de elementos de la lista.

  Total de bytes en memoria utilizados por la lista.

  Apuntadores lógicos al primero y ultimo elemento de la lista..

- **Secuencia de Operaciones.** A continuación presentamos la secuencia que fue aplicada a la clase ListaLigada:

  Secuencia 1

  ListarListaLigada

  Queelemento

  AgregarElementoaLista

  ListarListaLigada

  Secuencia 2

  ListarListaLigada

  Queelemento

  BorrarElementoaLista

  ListarListaLigada

- **Lista de excepciones**. Solo serán insertados, eliminados y consultados los elementos que componen la lista ligada especificada.

- **Condiciones externas.**

Debe existir la memoria suficiente para la inserción de elementos a la lista.

Debe existir la lista en la cual efectuaremos las búsquedas y/o insertaremos elementos.

- **Información Suplementaria.** Para la localización de elementos dentro de la lista, la clase cuenta con servicios de búsqueda por diferentes métodos para lo cual es necesario tener definidos los atributos por los cuales podemos hacer dichas búsquedas (índice del elemento, contenido del elemento o el tamaño de memoria utilizada).

1. **Evaluación.**

Al final de las pruebas pudimos comprobar el correcto intercambio d mensajes entre las clases listaligada y elementolista, permitiendo una administración sana de la memora en el servidor. Sin embargo, se encontraron errores al perder la referencia del primero y ultimo registro de la lista ligada lo cual imposibilitaba l correcta búsqueda de información dentro de la lista, para lo cual fue necesario incluirlos como propiedades de la clase.

PRUEBA 4

1. **Identificación Única.**

Prueba informacion.Aleatoria.01

2. **Objetivo.**

Comprobar el correcto uso de la información financiera alojado en la memoria de la computadora mediante el uso de listas circulares doblemente ligadas, para la cual la clase cuenta con servicios que permiten la inserción, borrado y consulta de los elementos que la componen.

1. **Pasos de Prueba.**

- *Estados a Probar. Comprobar que los atributos* que componen la lista brinden información correcta:

  Total número de elementos de la lista.

  Total de bytes en memoria utilizados por la lista.

  Apuntadores lógicos al primero y ultimo elemento de la lista..

- **Secuencia de Operaciones.** A continuación presentamos la secuencia que fue *aplicada a la clase* ListaLigada:

  Secuencia 1

  ListarListaLigada

  Queelemento

  AgregarElementoaLista

  ListarListaLigada

  Secuencia 2

  ListarListaLigada

  Queelemento

  BorrarElementoaLista

  ListarListaLigada

- **Lista de excepciones**. Solo serán *insertados, eliminados* y *consultados* los elementos que componen la lista ligada especificada.

- **Condiciones externas.**

Debe existir la memoria suficiente para la inserción de elementos a la lista.

Debe existir la lista en la cual efectuaremos las búsquedas y/o insertaremos elementos.

- **Información Suplementaria.** Para la localización de elementos dentro de la lista, la clase cuenta con servicios de búsqueda por diferentes métodos para lo *cual es necesario tener definidos los atributos* por los cuales podemos hacer dichas búsquedas (índice del elemento, contenido del elemento o el tamaño de memoria utilizada).

1. **Evaluación.**

   Al final de las pruebas pudimos comprobar el correcto intercambio d mensajes entre las clases listaligada y elementolista, permitiendo una administración sana de la memora en el servidor. Sin embargo, se encontraron errores al perder la referencia del primero y ultimo registro de la lista ligada lo cual imposibilitaba l correcta búsqueda de información dentro de la lista, para lo cual fue necesario incluirlos como propiedades de la clase.

3.3.6. Pruebas con Respecto a la Estación de Trabajo.

Las pruebas con respecto a la estación de trabajo representan una revisión final de las especificaciones técnicas, del diseño y de asegurar que se cumplan los requerimientos definidos por los usuarios para su uso y desempeño. Se evaluaron los siguientes puntos para la realización de estas pruebas:

- Evaluación de los filtros de información para verificar que las validaciones establecidas en el sistema sean correctas.
- Evaluación con respecto al comportamiento del flujo de la información; es decir asegurarse que el sistema produzca las salidas apropiadas de acuerdo a diferentes entradas.
- Asegurarse que el sistema cumpla con los requerimientos definidos.
- Validación del flujo de información con respecto al flujo de las pantallas para que este sea el adecuado en el desempeño del sistema.

Se tomaron en cuenta las siguientes preguntas para la realización de las pruebas de interface :

Para ventanas:

¿El tamaño de las ventanas puede ser ajustado, moverse y desplegarse?

¿Se encuentra todo el contenido de la información dentro de la ventana accesible adecuadamente con el ratón, teclas de función?

¿Se regeneran las pantallas adecuadamente cuando se sobre escribe y se vuelve a abrir?

¿Operan todas las funciones relacionadas con la ventana?
¿Están disponibles y desplegados apropiadamente en la ventana todos los menús emergentes, barras de herramientas, barras deslizantes, cuadros de diálogo, botones, iconos y otros controles importantes?
¿Cuando se despliegan varias ventanas se representan adecuadamente el nombre de cada ventana?
¿Está resaltada adecuadamente la ventana activa?

Para menús:
¿Se muestra la barra de menú apropiada en el contexto apropiado?
¿Funciónan adecuadamente los menús de escape, paletas y barras de herramientas ?
¿Son todas las funciones del menú accesibles con el ratón?
¿Funciónan adecuadamente las operaciones de despliegue?
¿Es correcto el tipo, tamaño y formato del texto?
¿Son suficientemente claros los nombres de las funciones del menú?
¿Cambia el cursor indicando el procesamiento (por ejemplo un reloj) al invocar diferentes operaciones?

Entrada de datos:
¿Se repiten y son introducidos adecuadamente los datos alfanuméricos en el sistema?
¿Se reconocen los datos no válidos?
¿Son inteligibles los datos de entrada de datos?

En las pruebas de interface con el usuario respecto a la estación de trabajo se evaluaron los siguientes puntos:

- Funciónalidad de botones para verificar que realicen la operación correcta y presentar la pantalla correspondiente para lo cual fueron diseñados.
- La interface gráfica con el usuario debe de cumplir con los estándares.
- Revisar que no se encuentren errores ortográficos en las pantallas y en los mensajes de error.
- Verificar que el mensaje de error corresponda al suceso.
- Validar que los mensajes de error sean claros.
- Validación de campos de tipo fecha según el formato establecido en los requerimientos.
- Validación de los campos numéricos.

- Verificar que la información desplegada sea la correspondiente al evento, como por ejemplo el despliegue de una noticia, un ticker, un registro, etc.

3.3.6.1. Prueba de Sistemas de Tiempo Real.

La naturaleza asíncrona y dependiente del tiempo de muchas aplicaciones de tiempo real, añade un nuevo elemento a la complejidad de las pruebas: el tiempo ahora no sólo deben de considerarse las pruebas de la caja blanca y de caja negra, sino también el comportamiento de los sucesos y datos y los procesos que manejan datos en paralelo. En muchas situaciones los datos de prueba proporcionados al sistema de tiempo real cuando se encuentra en un determinado estado darán un resultado correcto, mientras que al proporcionárselos en otro estado conducirán a un error.

Existen dos tipos de pruebas que se deben de considerar como son las *pruebas entre tareas* y las *pruebas de volumen*. Para el caso de las primeras se prueban las tareas asíncronas que de antemano se sabe se comunican con otras, con diferentes datos y cargas de procesos para determinar si se producen errores de sincronización entre las tareas; así como también se prueban las tareas que se comunican mediante colas de mensaje o almacenes de datos, para detectar errores en el tamaño de almacenamiento de datos.

Para las pruebas de volumen se generan volúmenes reales de información requeridos para un crecimiento esperado, por lo cual el plan de pruebas debe de incluir la recuperación o respaldo de la información necesaria; generalmente este tipo de pruebas es inducido por cambios en el consumo de recursos y se ejecutan para conocer los efectos de un cambio.

3.3.7. Pruebas en Red.

El realizar pruebas en Red de una aplicación implica el analizar los paquetes que viajan a través de la misma propiciados por la aplicación "Traductor de Protocolo". El analizar la red en un ambiente Microsoft es dependiente de los protocolos que se usan en la misma y la aplicación que corre sobre el servidor y el cliente. En nuestro caso, la

aplicación trabaja estrictamente sobre el protocolo TCP/IP sin la cuál la aplicación no podría funcionar.

Para realizar el análisis de los paquetes que corren sobre la red hicimos uso del servicio de con que cuenta Microsoft para analizar el tráfico sobre la red (Network Monitor). Este permite conocer a detalle el contenido de los paquetes (paquetes cabeceras y mensajes). El análisis consistió en filtrar las diferentes direcciones IP (del Servidor –Traductor de Protocolo- y de diferentes clientes en momentos donde se estaban realizando peticiones de información financiera a la Plataforma Digital..

Cuando realizamos el análisis de los paquetes pudimos observar que éstos estaban conformados sólo por el protocolo TCP/IP, esto garantiza la configuración y buena ejecución de la aplicación, sin embargo, ocasionalmente ciertas peticiones no eran procesadas sobre todo cuando el tráfico era intenso sobre la red (con un alto índice de retransmisión de paquetes). Esto debido a que no previmos que el Heart Beat Time (este parámetro indica al servidor el tiempo que debe pasar para la transmisión a todos los clientes del siguiente pulso el cual les indica que el Servicio de Traducción y Resolución de Peticiones se encuentra levantado) trabajará con un índice alto de tráfico en la red.

Para resolver éste problema tuvimos que realizar pruebas forzando tráfico intenso en la red y ajustando en base a prueba y error el Heart Beat Time (quedando en 10 ms) con el apoyo del Network Monitor donde observamos los paquetes y sus tiempos de retransmisiones.

Otro problema encontrado y que no habíamos previsto es que cuando una computadora cliente es instalada esta requiere un nombre. El nombre seleccionado es el nombre NETBIOS utilizado para identificación, comunicación y propósitos de revisión sobre la red. Es vital que el nombre sea único sobre la red.

Típicamente una máquina cliente es notificada sobre la instalación de la misma si existe un nombre duplicado sobre la red. Si, por algún motivo, una máquina cliente es instalada fuera de la red y después conectada a la misma los nombres pueden existir.

Para resolver éste problema nos apoyamos nuevamente del Network Monitor y de herramientas del Sistema Operativo para identificar las computadoras que tenían el mismo nombre y así poder especificar uno único, quedando en la documentación de la instalación del sistemas el procedimiento para su identificación en el caso de volverse a presentar el problema.

3.4. Factibilidad Técnica y Operativa.

3.4.1. Factibilidad Técnica.

Como ya se ha mencionado antes, sistemas de este tipo se han construido en diversas partes del mundo para ajustar fuentes de información locales a la plataforma digital.

Para este efecto la plataforma digital cuenta con dos formas de transferir información entre los distintos componentes. La primera consiste en registros en los que cada campo esta identificado por un número de campo que, junto con su número de servicio y número de mascara, identifica a cada objeto de manera independiente y única dentro de la plataforma digital. La segunda consiste en crear páginas de información en modo texto. Estas páginas no son mas que ventanas sobre las que se escriben letras, números y otros símbolos de la tabla ASCII que son vistos en las terminales de los usuarios.

En el caso de nuestra fuente de información, se tienen varios tipos de datos encapsulados en forma de registros y que se leen de la fuente de datos. Los registros del tipo 6 tienen información de la BMV pertinente a movimientos de valores durante un día de operaciones. Los subtipos del tipo 6 determinan que tipo de información especifica se trata, el subtipo A trata de Hechos del Mercado de Capitales, el subtipo B sobre Posturas del Mercado de Capitales, el subtipo D sobre índices etc. En este caso resultó muy claro que todo el tipo 6 debía ser tratado con el formato de registros dentro de la plataforma digital.

Los registros de tipo 5 transportan la información de noticias. Una noticia es información fija que nunca cambia, y su longitud puede variar desde un renglón, hasta varios cientos de renglones. Existía la opción manejar los renglones como registros y enviarlos asociados a un número de mascara con una cantidad de renglones predefinidos, pero esto implicaría llevar mas control sobre la información procesada y

enviada haciendo mas lento el procesamiento. Se opto por tratar esta información como páginas en las que cada segmento de noticia esta ligado al anterior y al siguiente con un identificador alfa numérico.

Los registros del tipo 4 transportan información de páginas, una página es un objeto que no cambia tan frecuentemente como la información financiera del tipo 6, ni esta fija como las noticias del tipo 5. El tamaño de la información contenida en una página de la plataforma digital es fijo, por ejemplo, una página que transporta la información del dólar interbancario tendrá las mismas dimensiones durante todo el día de operaciones. En este caso tampoco hubo ninguna duda al respecto que estos registros deberían ser tratados como páginas de la plataforma digital.

Cada uno de los registros de la fuente de información es plenamente accesible desde el componente de software conocido como *fuente de datos*, y sus formatos de encriptación han sido proporcionados en forma de manuales técnicos al respecto. A demás de los tipos 4,5 y 6, la fuente de información maneja otros formatos de registros que no están disponibles a través de la fuente de datos porque contienen información de fuentes extranjeras, que no están permitidos a emitir vía esta fuente. Estos últimos no los hemos tomado en consideración por carecer de información sobre ellos.

El cliente ha proporcionará todos las facilidades necesarias durante todas las fases del desarrollo, incluyendo el software de sistema operativo, compilador y API´s para la programación en la plataforma digital.

De esta manera y considerando todos los puntos antes mencionados se llego a la conclusión de que el proyecto es factible técnicamente.

3.4.2. Factibilidad Operativa.

Los usuarios de este sistema son gente dedicada a realizar operaciones millonarias utilizando herramientas que cumplen con los estándares de plataformas digitales, mismos que cumple la plataforma digital que se utilizó.

De esta manera, aunque la interfaz que proporciona la fuente de información para leer esta información es muy distinta a la de la plataforma digital, el usuario esta acostumbrado a su uso. Además, parte de la idea de la integración a la plataforma digital es la homogeneización de procedimientos para el manejo de la información de varias plataformas.

Conclusiones.

Para la realización de este trabajo de tesis pudimos aplicar los conocimientos vistos en las materias de Ingeniería de sistemas, Programación estructurada, Redes de Computadoras, Temas especiales de computación, así como el conocimiento basado en la experiencia profesional de cada uno de nosotros.

El ingresar al PAT es un gran apoyo para las personas que ya tienen mucho tiempo de haber terminado la carrera; así como el contar con un asesor que nos apoye con sus conocimientos y nos exija terminar el trabajo en el período establecido por el programa.

En la actualidad la mayoría de las empresas operan sobre una plataforma como Windows la cual cuenta con librerías estándares que pueden utilizarse en diferentes lenguajes como Visual Basic, C++, etc.; para el desarrollo de esta tesis tuvimos que utilizar algunas de estas herramientas para facilitar el desarrollo y hacerlo de una manera más confiable y sencilla en su operación.

Las habilidades de poder analizar, evaluar, diseñar y poner en operación el sistema Traductor de Protocolos descrito en esta tesis se ven reflejadas en el resultado final, donde la definición adecuada de los requerimientos y el manejo de la arquitectura existente, el diseño e implementación dentro de la red de comunicaciones, la adecuación del software aplicativo y la integración de todos estos dieron como consecuencia la solución deseada. Las habilidades y conocimientos aplicados durante todo el proceso antes mencionado son parte de las capacidades, fundamentos, conocimientos y experiencias adquiridas durante la formación profesional que nos brindó la facultad de ingeniería, y que han quedado plasmados exitosamente en este trabajo de tesis.

El contar con un sistema confiable y eficiente para poder reunir información proveniente de diversas fuentes, donde la disposición y acceso inmediato de ellas son críticos para el cliente, era el objetivo principal de esta tesis. Se logró cubrir cabalmente el objetivo,

brindándole al usuario un sistema con múltiples herramientas que le permiten hacer labores de monitoreo de información, estadísticos, visualización personalizada de la información y consultas en tiempo real, de tal forma que le permiten obtener un mejor rendimiento, certidumbre y eficiencia en la toma decisiones aplicativas dentro del sector financiero.

La métodología empleada para el análisis y diseño, tanto en diagramas de contexto, como de objetos, dinámicos y funcionales, llegar a ser lo suficientemente claros para dar la idea general del sistema. No obstante al momento de derivar los diagramas a su último nivel, convierte la fase del análisis en un trabajo muy basto, que a la postre rinde frutos muy evidentes, ya que el tiempo invertido en esta etapa permite ganar exactitud en los resultados y las pruebas se vuelven tangiblemente mucho menos complicadas. Es evidente que un análisis completo de los requerimientos y un adecuado diseño permite ahorrarse tiempo y dolores de cabeza, ya que cualquier omisión en esta etapa se verá reflejada en el desarrollo del código, y de cualquier forma tendrá que repetirse el ciclo cronológico de la ingeniería de sistemas para subsanar el problema, ya que no es posible dentro del desarrollo corregir errores de análisis o diseño. Por lo anterior es necesario hacer resaltar que el análisis y el diseño por cuestiones obvias deben consumir la mayor parte del tiempo destinado para la conclusión del sistema, y que la documentación durante la vida de estos procesos es fundamental y no debe verse como un esfuerzo estéril.

El C++ es una extensión de el lenguaje C que incluye muchas de sus bondades, pero el uso inadecuado de algunas de estas extensiones puede entorpecer el desarrollo de un programa. Antes de usar alguna complicada o rebuscada característica de C++, se debe pensar si esa es la mejor solución, ya que fácilmente suele caerse en el desacierto de generar en una forma descontrolada objetos que en realidad no tienen una función que no pueda realizarse con código de el lenguaje C estándar. La programación orientada a objetos no es un sustituto para la inteligencia y el diseño: al contrario, cuando se hace desarrollo de aplicaciones con objetos, es necesario contar con un buen diseño antes de empezar a codificar.

La forma de manejar los registros financieros fue la más adecuada, ya que al mantenerlos en memoria mediante listas ligadas, cualquier petición o actualización de un registro financiero, es realizada al momento (en tiempo real), de esta forma el usuario puede realizar una toma de decisiones rápida y eficazmente con la seguridad de que fue tomada en base a datos muy actuales, ésto beneficia enormemente al usuario y se comprueba que el traductor de protocolos está realizando la función para la que fue creado.

La manera de presentar los registros financieros, registros de páginas, y registros de noticias, a través de la plataforma digital es la más eficaz y práctica para el usuario ya que una vez desplegadas las ventanas con la información el usuario puede personalizarlas cambiando atributos de estás, además de contar con la funcionalidad de DDE (Dynamic Data Exchange), y así poder exportar los datos con otras aplicaciones como Excel, Word y Visual Basic. De esta forma el usuario puede realizar gráficas y estadísticas a partir de su información real, la cual se actualizará en forma simultánea conforme a los datos del sistema se actualicen.

Bibliografía.

Ellis. M.A., y Stroustrup, B., "The Annotated C++ Reference Manual", Addison Wesley, 1a ED., 1990.

Stroustrup, b., "The C++ Programming Language", Addison Wsley, 1a ED. 1993.

Stroutrup, B., "Data Abstraction in C", AT&T Bell Laboratories, Technical Journal vol.63, No. 8, octubre 1984.

Kim. W., y Lochovsky, F.H., "object Oriented Concepts, Databases, an Aplications", AC; Press, 1a ED., 1989.

Lieberman, H., "Using prototipical Objects to Implement Shared Behavior in Object Oriented Systems", OOPSLA '86 Proceedings, ACM, 1986.

Eckel, B., "Using C++", Osborne McGraw-Hill, 1a. ED., 1989.

Schildt, H., "turbo C/C++ the Complete Reference", Borland-Osborne McGraw-Hill, 1a. ED., 1988.

Harrington James, "Measuring Fragmentation: Area malloc an free gragmenting your heap?", Dr. Dobb's Journal, No. 199, abril 1993.

Roger S. Pressman, "Ingeniería del software un enfoque práctico", Cuarta edición, Mc Graw Hill, España 1997.

Ladd Scott, " Applying C++", M&T Books, 1992.

James Rumbaugh, "Object, Oriented, Moudeling and Design".

Timothy Parker, "Aprendiendo TCP/IP en 14 días", Ed. Prentice Hall, Segunda Edición, 1997.

Dr. Sidnie Feit, "TCP/IP", Ed. Mc. Graw Hill, Primera Edición, 1998.

Salvatore Salamone, "Guía de gestión de conectividad remota", Ed. Mc. Graw Hill, Primera Edición, 1998.

Conceptos de Redes

http://www.itam.mx/info/extuni/sat/comp/redes/conceptos.html

APÉNDICE A (Código ASCII)

0	00	NUL	43	2B	+	86	56	V
1	01	SOH	44	2C	'	87	57	W
2	01	STX	45	2D	-	88	58	X
3	03	ETX	46	2E	.	89	59	Y
4	04	EOT	47	2F	/	90	5A	Z
5	05	ENQ	48	30	0	91	5B	[
6	06	ACK	49	31	1	92	5C	\
7	07	BEL	50	32	2	93	5D	]
8	08	BS	51	33	3	94	5E	^
9	09	HT	52	34	4	95	5F	_
10	0A	LF	53	35	5	96	60	`
11	0B	VT	54	36	6	97	61	a
12	0C	FF	55	37	7	98	62	b
13	0D	CR	56	38	8	99	63	c
14	0E	SO	57	39	9	100	64	d
15	0F	SI	58	3A	:	101	65	e
16	10	DEL	59	3B	;	102	66	f
17	11	DC1	60	3C	<	103	67	g
18	12	DC2	61	3D	=	104	68	h
19	13	DC3	62	3E	>	105	69	i
20	14	DC4	63	3F		106	6A	j
21	15	NAK	64	40	@	107	6B	k
22	16	SYN	65	41	A	108	6C	l
23	17	ETB	66	42	B	109	6D	m
24	18	AN	67	43	C	110	6E	n
25	19	EM	68	44	D	111	6F	o
26	1A	SUB	69	45	E	112	70	p
27	1B	ESC	70	46	F	113	71	q
28	1C	FS	71	47	G	114	72	r
29	1D	GS	72	48	H	115	73	s
30	1E	RS	73	49	I	116	74	t
31	1F	US	74	4A	J	117	75	u
32	20	SPC	75	4B	K	118	76	v
33	21	!	76	4C	L	119	77	w
34	22	"	77	4D	M	120	78	x
35	23	#	78	4E	N	121	79	y
36	24	$	79	4F	O	122	7A	z
37	25	%	80	50	P	123	7B	{
38	26	&	81	51	Q	124	7C	L
39	27	'	82	52	R	125	7D	}
40	28	(	83	53	S	126	7E	
41	29	)	84	54	T	127	7F	DEL
42	2A	*	85	55	U			

APÉNDICE B

ALGORITMO CRC

El algoritmo propuesto es el conocido como CRC de 16bits cuyo valor final es de 2 bytes (16 bits), se representaría en 4 caracteres ASCII en Hexadecimal.

```
Unsigned short CalculaCRC (unsigned char *paquete, unsigned short longitud)
{
unsigned short register crc;
unsigned short register i;
unsigned short register j;

crc = 0;
j = lognitud;
while(j--){
    i = (unsigned short)*paquete++;
    i = i << 8;
    crc = crc^i;
     for(i=0; i<8; ++i)
      if(crc & 0x8000){
        crc = (crc <<1);
        crc = crc ^0x1021;    }
     else
        crc=crc<<1;
}
return crc;
}
```

Básicamente es cuestión de pasar el parámetro del registro al que se le desea calcular el CRC y su longitud correspondiente (sin incluir los 2 bytes de CRC), y el resultado se concatena al registro mismo con un último campo.

En la recepción lo que se haría es obtener el CRC de los caracteres que se reciben (sin incluir el campo de CRC) y se compara con el CRC enviado, si son iguales se acepta el registro, si son diferentes se desecha y se comienza a buscar por el siguiente registro.

```
Ejemplo:
CharPrueba[ ]="Hola, esto es una prueba"
Unsigned short CRC;
CRC=CalculaCRC((unsigned char*)Prueba));
```

La variable CRC debe obtener el valor de 43007 que en string hexadecimal es A7FF que sería enviado como el último campo después de delimitarlo (EXT). En caso de que el valor resulte de menos de 4 caracteres ASCII, éste sería rellenado con ceros a la izquierda (ejemplo: 0AB3).

APÉNDICE C

TIPOS DE MENSAJES:

0	Control	Reservado
1	Reservado	NO debe procesarse
2	Reservado	NO debe procesarse
3	Reservado	NO debe procesarse
4	Páginas	Información diversa en formato de tabla
5	Noticias	Notas realizadas por las salas de redacción de la Fuente de Información Financiera
6	BMV	Bolsa Mexicana de Valores: incluye acciones, índices y warrants
7	USA	Información de USA, incluye acciones y opciones

APÉNDICE D

FORMATO DE DATOS

Se provee un estándar de formatos para representar diversos tipos de información que entre otros propósitos permite comprimir la información de tal manera que el ancho de banda del Sistema de Alimentación sea mejor aprovechado.

Fechas

[Mes][Día][Año]

Mes

El mes es representado con sólo un carácter, de la siguiente manera:

Enero	A	Julio	G
Febrero	B	Agosto	H
Marzo	C	Septiembre	I
Abril	D	Octubre	J
Mayo	E	Noviembre	K
Junio	F	Diciembre	L

Día

Al igual que los meses, los días se representan con sólo un carácter:

1	A	12	L	23	W
2	B	13	M	24	X
3	C	14	N	25	Y
4	D	15	O	26	Z
5	E	16	P	27	[
6	F	17	Q	28	\
7	G	18	R	29	]
8	H	19	S	30	^
9	I	20	T	31	_
10	J	21	U		
11	K	22	V		

Año

El año puede representarse de dos maneras:

* Con sólo un carácter si se trata de un año comprendido entre 1991 y 2020

1991	A	2001	K	2011	U
1992	B	2002	L	2012	V
1993	C	2003	M	2013	W
1994	D	2004	N	2014	X
1995	E	2005	O	2015	Y
1996	F	2006	P	2016	Z
1997	G	2007	Q	2017	[
1998	H	2008	R	2018	\
1999	I	2009	S	2019	]
2000	J	2010	T	2029	^

* Con cuatro dígitos si es un año no comprendido en los anteriores.

Ejemplos de fechas:

Dic 4, 1994	LDD
Feb 30, 1998	B^H
Feb 30, 1998	B^1998
Jul 28, 2010	GT
Abr 4, 1930	DD1930

Horas [Hora][Minuto]

Hora La hora es representada con sólo un caracter como sigue:

0	A	8	I	16	Q
1	B	9	J	17	R
2	C	10	K	18	S
3	D	11	L	19	T
4	E	12	M	20	U
5	F	13	N	21	V
6	G	14	O	22	W
7	H	15	P	23	X

Minuto Al igual que la hora los minutos se representan con sólo un carácter:

0	A	30	_
1	B	31	`
2	C	32	a
3	D	33	b
4	E	34	c
5	F	35	d
6	G	36	e
7	H	37	f
8	I	38	g
9	J	39	h
10	K	40	i
11	L	41	j
12	M	42	k
13	N	43	l
14	O	44	m
15	P	45	n
16	Q	46	o
17	R	47	p
19	S	48	q
19	T	49	r
20	U	50	s
21	V	51	t
22	W	52	u
23	X	53	v
24	Y	54	w
25	Z	55	x
26	[	56	y
27	\	57	z
28	]	58	{
29	^	59	|

Ejemplo de horas

10:35AM	Kd
17:22	RW
11:57PM	Xz

Numéricos.
Los numéricos de precios o volúmenes pueden expresarse sin comprimir o compresos mediante la adición de un prefijo que indique la posición del punto decimal ya sea multiplicando o dividiendo por factores de 10.

Tabla de prefijos de multiplicación.

A	10	(el número hay que multiplicarlo por 10)
B	100	(el número hay que multiplicarlo por 100)
C	1,000	
D	10,000	
E	100,000	
F	1000,000	
G	10,000,000	

Tabla de prefijos de división.

a	10	(el número hay que dividirlo por 10)
b	100	(el número hay que dividirlo por 100)
c	1,000	
d	10,000	
e	100,000	
f	1000,000	
g	10,000,000	

Ejemplos:
Algunos ejemplos de números son los siguientes:

1,456,700	B14567
10,000,000	G1A22
0.56	b56
0.45	d45

Numéricos con fracciones.

Los numéricos contienen fracciones de mercados internacionales que representan con un prefijo de 1 ó 2 caracteres. El primero de ellos indica el número de factores de 10 por los que se va a multiplicar la parte entera y el segundo carácter del prefijo indica la fracción.

Tabla de prefijos de multiplicación.

A	10	(el número hay que multiplicarlo por 10)
B	100	(el número hay que multiplicarlo por 100)
C	1,000	
D	10,000	
E	100,000	
F	1000,000	
G	10,000,000	

Nota:
Esto implica que el número decimal se corra a la derecha (se agreguen ceros a la derecha).

Tabla de prefijos para fracciones.

‘	1/8
@	¼,2/8
#	3/8
$	½,4/8
%	5/8
^	¾,6/8
&	7/8
*	Indica que los últimos números son sesenta y cuatroavos

Ejemplos de números fraccionados.

1,400 7/8	B&14
1,000 ¾	C^1
18 ½	$18
2,200 7/32	B*2214 (2,200 14/64)

APÉNDICE E

Fuentes de información de páginas y noticias.

PRUEBA Esta fuente indica que la información es de pruebas y NO debe ser para usuarios finales.

INFOFIN Fuente de Información Financiera.

Mercados que maneja el Sistema de Alimentación.

Capitales Mex
USA
Indices
Warrants

APÉNDICE F

Clasificación de Noticias.

Noticias México:		Noticias de Colombia:	
Capitales:		Cambios:	
101	Perspectivas	451	Noticias
102	Reportes Financieros	Capitales:	
103	Razones Financieras	401	Comentarios de las Bolsas
104	Información Oficial	406	Estados Financieros
105	Derechos	402	Información Oficial
183	Múltiplos	403	Noticias
182	B.M.V. Internacional	404	Perspectivas
180	Comentarios B.M.V.	405	Razones Financieras
184	Fondos de Inversión	407	Rumores
106	Noticias	Dinero:	
107	Rumor	411	Colocaciones Mercado Primario
Dinero:		413	Flujo de Liquidez
108	Subastas Banco de México	414	Información Oficial
109	Mesas de Dinero Fondeo	415	Noticias
110	Mesas de Dinero Plazo	416	Rumores
111	Tasas Bancarias	412	Subasta Banrepública y Tesorería
113	Información Oficial	417	Tasa a la vista
112	Colocaciones Primarias	Indicadores:	
114	Análisis	421	Comercio Exterior
181	Comentarios B.M.V.	422	Deuda Externa
115	Noticias	423	Empleo
116	Rumor	424	Fiscales
Cambios:		425	Lideres
117	Dólar Interbancario	426	Monetarios
118	Dólar Menudeo	427	Precios
120	Coberturas	428	Producción
119	Otras Divisas	429	Reservas internacionales
121	Noticias	430	Salarios
122	Metales Nacionales	Internacional:	
Noticias en General:		441	Bolsas del Mundo
124	Prensa Nacional	442	Commodities

125	Flash Noticioso	443	Divisas
126	México en el Mundo	444	Mercado de Dinero
128	Agenda Financiera	445	Metales
150	Prensa Internacional	Mercancías:	
151	América	462	Internacionales
152	Europa	461	Nacionales
127	Análisis Semanal	Noticias:	
153	Resto del Mundo	471	Agenda
Internacional:		472	Colombia en el Mundo
154	Bolsas del Mundo	473	Flash América del Norte
155	Mercado de Dinero	474	Flash América Latina
156	Mercado de Divisas	475	Flash Colombia
157	Metales	476	Flash Europa
159	Futuros	477	Flash Resto del Mundo
160	Petróleo	478	Prensa Internacional
Mercancías:		479	Prensa Nacional
123	Nacionales	480	Prensa Semanal
158	Internacionales	Acceso:	
		491	Acceso

APÉNDICE G

Descripción de archivos que componen el Sistema de Alimentación.

Archivos	Descripción
ddmmmaa.log	Archivo de registro (log) (donde ddmmmaa corresponde a la fecha en que se generó el archivo).
dfeed.ini	Programa que ejecuta la aplicación.
tblaccio.dat	Tabla con la información de los instrumentos manejados por la Fuente de Información Financiera.
tblbolsa.dat	Tabla con la información de las casas de bolsa que maneja la Fuente de Información Financiera.
tblvalor.dat	Tabla con la clasificación que asigna la BMV a los instrumentos que maneja la Fuente de Información Financiera.

Descripción de archivos que componen el Decodificador del Sistema de Alimentación.

Archivos	*Descripción*
isdfdeco.exe	Programa que ejecuta la aplicación
isdfdeco.ini	Archivo de configuración
ddmmmaa.log	Archivo de registro (log) (donde ddmmmaa corresponde a la fecha en que se generó el archivo)

APÉNDICE H

Bolsas donde cotizan las acciones de USA.

A	65	American Stock Exchange
B	66	Boston Stock Exchange
C	67	Cincinnati Stock Exchange
M	77	Chicago Stock Exchange
N	78	New York Stock Exchange
P	80	Pacific Stock Exchange
S	83	NASDAQ Small Caps
T	84	NASDAQ
W	87	Chicago Board Options Exchange
X	88	Philadelphia Stock Exchange

APÉNDICE I

Código Fuente

General.cpp

```
#include <stdio.h>
#include "general.h"

void LeerLinea(FILE *f,char *buf, unsigned long maxbuf)
{
        char c,salida=0;
        unsigned long pos=0;

        while(!salida)
        {
                        c=fgetc(f);
                        if (c=='\n' || c=='\r')
                        {
                                buf[pos]=0;
                                return;
                        }
                        buf[pos]=c;
                        pos=(pos>maxbuf-1)? maxbuf-1:pos+1;
                        if (feof(f)) salida=1;
        }

}
```

Informacion.h

```
#ifndef INFORMACIÓN
#define INFORMACIÓN

#include "p2.h"
#include <stdio.h>
#define _MaxBufInf_ 10240

class Informacion
{
public:
        Puerto puerto;
        FILE *archivo;
        char modolectura,mensajes;
        char buf[_MaxBufInf_],RegListo[_MaxBufInf_],estado,hayregistro;
        unsigned int posbuf,CRC,lonreg;

        Informacion();
        ~Informacion();
        void UsarPuerto(char *com);
        void UsarArchivo(char *nombre);
        void UsarNada();
        char HayDatos();
        int HayRegistro();
        char LeerByte();
        void LeerRegistro(char *buf);
        void ProcesarByte();
        void MeterByte(char c);
        void SacarRegistro(char *buf_);
        unsigned int QueCRC(unsigned char *buf);
};
```

```
#endif
```

Informacion.cpp

```
#include "informac.h"

Informacion::Informacion()
{
        archivo=NULL;
        modolectura=0;
        buf[0]=0;
        posbuf=0;
        estado=0;
        hayregistro=0;
        lonreg=0;
        mensajes=0;
}

Informacion::~Informacion()
{
        UsarNada();
}
void Informacion::UsarPuerto(char *com)
{
        if (modolectura) UsarNada();
        puerto.AbrirPuerto(com);
        modolectura='p';
}

void Informacion::UsarArchivo(char *nombre)
{
        archivo=fopen(nombre,"rb");
        modolectura='a';
}

void Informacion::UsarNada()
{
        if (modolectura=='a') fclose(archivo);
        if (modolectura=='p') puerto.CerrarPuerto();
        modolectura=0;
}

char Informacion::LeerByte()
{
        if (!modolectura) return(0);
        if (modolectura=='a') return(fgetc(archivo));
        if (modolectura=='p') if (puerto.HayDatos()) return(puerto.Leer());
        return(0);
}

char Informacion::HayDatos()
{
        if (!modolectura) return(0);
        if (modolectura=='a') if ( !feof(archivo) ) return(1);
        if (modolectura=='p') if (puerto.HayDatos()) return(1);
        return(0);
}

void Informacion::MeterByte(char c)
{
        buf[posbuf]=c;
```

```
        posbuf++;
        if (posbuf>=_MaxBufInf_-1) posbuf=_MaxBufInf_-2;
        buf[posbuf]=0;
        //printf("\n%s",buf);
}

void Informacion::ProcesarByte()
{
        unsigned char c,crc1;
        unsigned short crc;

        if (HayDatos())
        {
                c=LeerByte();

                if (estado==0 && c==1)
                {
                        posbuf=0;
                        MeterByte(c);
                        estado=1;
                } else
                if (estado==1&& c!=3)
                {
                        MeterByte(c);
                } else
                if (estado==1&& c==3)
                {
                        MeterByte(c);
                        estado=2;
                } else
                if (estado==2)
                {
                        CRC=c;
//                      printf("\n0x%X",CRC);
                        estado=3;
                } else
                if (estado==3)
                {
                        crc1=c;
//                      printf(":%X",crc1);
//                      printf("\n %X %X",CRC,crc1);
                        CRC=(crc1*256)|(CRC);

                        crc=QueCRC((unsigned char *)buf);
                        if (crc!=CRC && mensajes) { printf("\nERROR CRC (%x  %x)",crc,CRC); }

                        estado=0;
                        memcpy(RegListo,buf,posbuf);
                        RegListo[posbuf]=0;
                        lonreg=posbuf;
                        posbuf=0;
                        hayregistro=1;
                }
        }
}

void Informacion::SacarRegistro(char *buf_)
{
        if (!hayregistro) { buf[0]=0; return; }
        memcpy(buf_,RegListo,lonreg);
        buf_[lonreg]=0;
        lonreg=0;
```

```
        hayregistro=0;
}

unsigned int Informacion::QueCRC(unsigned char *buf)
{
        unsigned short i,j,crc;

        crc=0;
        j=strlen((char *)buf);

        while(j--)
        {
                i=(unsigned short) * buf ++;
                i=i<<8;
                crc=crc^i;
                for (i=0;i<8;++i)
                if (crc&0x8000)
                {
                        crc=(crc<<1);
                        crc=crc^0x1021;
                }
                else crc=crc<<1;
        }
        return(crc);
}
```

ObjInfo.h

```
#ifndef OBJETOINFOSEL
#define  OBJETOINFOSEL

#include "informac.h"
#include "tablas.h"
#include "lpags.h"
#include "general.h"
#include <conio.h>
#include <string.h>
#include <stdio.h>

#define _MaxBufTit_ 500
#define _MaxBufDatos_ 5000

class ObjetoInfosel
{
public:
        char hayinfo,estado,mensajes,noticias,bmv;
        char buftitulos[_MaxBufTit_],bufdatos[_MaxBufDatos_];
        Informacion info;
        Tabla ECampos,Datos,*Pubs;
        ListaLigada PendPub;
        ListaPáginas LP;


        ObjetoInfosel();

        void ProcesarInfo();
        int LeerObjeto(char *nombre);
        char *EncontrarCaracter(char *buf,char c,unsigned long num);
        void SacarCadena(char *salida,int tambuf,char *buf,char fin);
        void GuardarNoticia(char *nombre,char *RegListo);
        int ProcesarReg(char *buftitulos);
        void ProcesarPágina(char *cab,char *RegListo);
```

```
        void TransformarAInvision1(char *reg,char *invreg);
        void TransformarAInvision2(char tipo,char *reg,char *invreg);
        void CrearRegisroVacio(char tipo,char stipo,char *salida,unsigned int maxs);

        int LeerConfExt(char *nombre);
        int SacarCadena(unsigned char *buf,unsigned char *destino,unsigned int max,char terminacon);
        int SacarNumero(unsigned char *buf,double &val);
        int SacarHora(unsigned char *buf,unsigned char *destino);
        int SacarFecha(unsigned char *buf,unsigned char *destino);

        void ActualizarRegistro(char *nompub,char *reginv);
        void QuitarCeros(unsigned char *buf);
};

#endif
```

ObjInfo.cpp

```
#include "ObjInfo.h"
ObjetoInfosel::ObjetoInfosel()
{
        estado=0;
        hayinfo=0;
        mensajes=0;
        noticias=bmv=1;
        // Se crea la tabla de equivalencia de campos
        //      t       Tipo
        // nf   Nombre del campo en Infosel Financiero
        //      nt      Nombre en tabla de datos
        //      ni      Nombre de Invision
        //      td      Tipo de dato
        //              c = Cadena
        //              n = Numero
        //              f = Fecha
        //              h = Hora
        //              b = Byte, un número enfrente indica el número de bytes en el campo
        //              ? = Alguna de estas palabras, indica que es un campo que no tiene
        //                  longitud definida pero debe coincidir con cualquiera de las palabras
        //                  indicadas. Cada palabra debe estar separada por un '|'
        //                  Ej td=<?esto|o esto|o esto otro>, todos los espacios son tomados en
        //                          cuenta lo mismo que mayusculas o minusculas.
        ECampos.NuevaTabla("ec: t,nf,nt,ni,td");
        if (LeerConfExt("equiv.ini"))
        {
                printf("\nNo fue posible leer el archivo de equivalencias EQUIV.INI");
        }
        LP.PendPub=&PendPub;
}

void ObjetoInfosel::ProcesarInfo()
{
//      unsigned int i;
        char *pos,temp1[100],temp2[100],nombre[100];

        info.ProcesarByte();
        if (info.hayregistro)
        {
                if (estado==0) //Esperando un encabezado
                {
                        if (info.hayregistro)
                        {
```

```
// Aqui procesamos los encabezados de las páginas !!!!!!!!
                                    if (info.RegListo[1]=='4') //Es una página
                                    {
                                            //      printf("\n%s",info.RegListo);
                                            if (info.RegListo[2]=='B') //Es el cuerpo de la página
                                            {
                                                    //if (mensajes)
                                                    printf("\nPáginas: Un jinete sin cabeza.\n(%s)");
                                                    estado=0;
                                                    info.hayregistro=0;
                                                    return;
                                            }
                                            info.SacarRegistro(buftitulos);
                                            printf("\nPágina: %s", buftitulos);
                                            estado=1;
                                    }

// Aqui procesamos los encabezados de las noticias !!!!!!!!
                                    if (info.RegListo[1]=='5' && noticias) //Es una noticia
                                    {
                                            //      printf("\n%s",info.RegListo);
                                            if (info.RegListo[2]=='B') //Es el cuerpo de la noticia
                                            {
                                                    //if (mensajes)
                                                    printf("\nNoticias: Un jinete sin cabeza.\n(%s)");
                                                    estado=0;
                                                    info.hayregistro=0;
                                                    return;
                                            }
                                    }
// Aqui procesamos la informacion de la BMV !!!!!
                                    if (info.RegListo[1]=='6' && bmv) //Es un registro BMV
                                    {
                                            //      printf("\n%s",info.RegListo);
                                                    info.SacarRegistro(buftitulos);
                                                    if (mensajes) printf("\n%s",buftitulos);
                                                    ProcesarReg(buftitulos);
                                    }
                            }
                    }
// Aqui terminamos de procesar encabezados
                    else //if (estado==0)

//Empezamos a procesar los cuerpos
                    if (estado==1) //Esperando un encabezado
                    {
                            if (info.hayregistro)
                            {

//PROCESAMOS LOS CUERPOS DE NOTICIAS**********************************
                                    if (info.RegListo[1]=='5' && noticias) //Es una noticia
                                    {
                                            //printf("\n%s",info.RegListo);
                                            if (info.RegListo[2]=='A') //Es el encabezado de la noticia
                                            {
                                                    //if (mensajes)
                                                            printf("\nNoticias: Una cabeza sin
jinete.\n(%s)",info.RegListo);
                                                    estado=0;
                                                    ProcesarInfo();
                                                    return;
                                            }
```

```
                                        info.hayregistro=0;
                                        estado=0;
                                        SacarCadena(temp1,100,pos,',');

                                        pos=EncontrarCaracter((char *)info.RegListo,',',1); pos++;
                                        if (pos==NULL) { printf("\nNo se encuentra el número de noticia 2");
return; }
                                        SacarCadena(temp2,100,pos,',');

                                        if (strcmp(temp1,temp2))
                                        {
                                                printf("\nLos números de noticias no coinciden:
%s!=%s",temp1,temp2);
                                                return;
                                        }

                                        // Construimos un nombre de archivo con el número de la nota
                                        nombre[0]=0;
                                        strcat(nombre,"n");
                                        strcat(nombre,temp1);
                                        //strcat(nombre,".txt");

                                        //Vaciamos en el archivo la nota.
                                        GuardarNoticia(nombre,info.RegListo);
                                } // Fin  if (info.RegListo[1]=='5' && noticias) //Es una noticia


                                else

//PROCESAMOS LOS CUERPOS DE LAS PÁGINAS*********************************
                                if (info.RegListo[1]=='4') //Es una PÁGINA
                                {
                                        //printf("\n%s",info.RegListo);
                                        if (info.RegListo[2]=='A') //Es el encabezado de la PÁGINA
                                        {
                                                //if (mensajes)
                                                printf("\nPáginas: Una cabeza sin jinete.\n(%s)",info.RegListo);
                                                estado=0;
                                                ProcesarInfo();
                                                return;
                                        }
                                        info.hayregistro=0;
                                        estado=0;
                                        pos=EncontrarCaracter((char *)buftitulos,',',2); pos++;
                                        if (pos==NULL) { printf("\nNo se encuentra el número de página 1");
return; }
                                        SacarCadena(temp1,100,pos,',');

                                        pos=EncontrarCaracter((char *)info.RegListo,',',1); pos++;
                                        if (pos==NULL) { printf("\nNo se encuentra el número de página 2");
return; }
                                        SacarCadena(temp2,100,pos,',');

                                        if (strcmp(temp1,temp2))
                                        {
                                                printf("\nLos números de páginas no coinciden:
%s!=%s",temp1,temp2);
                                                return;
                                        }
                                        ProcesarPágina(buftitulos,info.RegListo);

                                } // Fin  if (info.RegListo[1]=='4') //Es una PÁGINA
```

```
//*****************************************************************************

                                    else printf("\nFalta cuerpo del Dato;");
                                }// if (info.hayregistro)

                        } // FIN if (estado==1) //Esperando un encabezado

                }
}

char *ObjetoInfosel::EncontrarCaracter(char *buf,char c,unsigned long num)
{
        unsigned long i,num2=0;
        for (i=0;i<strlen(buf);i++)
                if (c==buf[i]) { num2++; if (num2==num) return(buf+i); }
        return(NULL);
}

void ObjetoInfosel::SacarCadena(char *salida,int tambuf,char *buf,char fin)
{
        int i=0;

        while(i<tambuf-1 && buf[i]!=fin)
        {
                salida[i]=buf[i];
                i++;
        }
        salida[i]=0;
}

#include "gnoti.cpp"


int ObjetoInfosel::LeerConfExt(char *nombre)
// Regresa
// 1 No se puede abrir el archivo:
{
        FILE *f;
        char buf[1024],salida=0;

        f=fopen(nombre,"r");
        if (f==NULL) return(1);

        while(!salida)
        {
                LeerLinea(f,buf,1024);
                if (*buf=='#')
                {
                        if (tolower(buf[1])=='c')
                        {
                                if (mensajes) printf("\nCampo: %s",buf+2);
                                ECampos.GuardarRegistro(buf+2);
                        }
                        if (tolower(buf[1])=='t')
                        {
                                if (mensajes) printf("\nNueva tabla: %s",buf+2);
                                Datos.NuevaTabla(buf+2);
                        }
                }
                if (feof(f)) salida=1;
        }
```

```
	fclose(f);
	return(0);
}

int ObjetoInfosel::SacarHora(unsigned char *buf,unsigned char *destino)
{
	int hrs=buf[0]-'A',min=buf[1]-'A';
	destino[0]=0;
	sprintf((char*)destino,"%d:%d",(char*)hrs,(char*)min);
	return(2);
}

int ObjetoInfosel::SacarFecha
(unsigned char *buf,unsigned char *destino)
{
	int dia=buf[1]-'A'+1,mes=buf[0]-'A'+1,a=buf[2]-'A'+1991;
	destino[0]=0;
	sprintf((char*)destino,"%d/%d/%d",(char*)dia,(char*)mes,(char*)a);
	return(3);
}

int ObjetoInfosel::SacarCadena
(unsigned char *buf,unsigned char *destino,
 unsigned int max,char terminacon)
{
	unsigned int p=0;
	unsigned char *pi=buf;

	while(*pi!=terminacon)
	{
		if(p+1 < max)
		{
			*(destino+p)=*pi;
		}
		else
		{
			if (mensajes) printf("\nError: La cadena en el buffer de entrada no cabe en el buffer de
salida.");
			if (mensajes) printf("\nTam. del buffer de salida: %u  Posicion actual: %d ",max,p);
			//printf("\n(%c)<%s> \n  <%s> \n",*buf,buf,pi);

			return(p+1);
		}
		pi++; p++;
		*(destino+p)=0;
	}
	return(p+1);
}

int ObjetoInfosel::SacarNumero(unsigned char *buf,double &val)
{
	unsigned char *base=buf;
	double div[]={10.0,100.0,1000.0,10000.0,100000.0,1000000.0,10000000.0 };
	int i=0,j=0,m=1,tipo=0,l=strlen((const char *)base);
	char buf2[100];


	if ( !isdigit(*buf) )
	{
		base++;
		if (buf[0]>='a' && buf[0]<='g')		{ m=buf[0]-'a'; tipo=1; }
		if (buf[0]>='A' && buf[0]<='G')		{ m=buf[0]-'A'; tipo=2; }
```

```
        }

        for (i=0,j=0;i<l;i++)
        {
                if ( ( base[i]>='0' && base[i]<='9') || base[i]=='.' )
                {
                        buf2[j]=base[i]; j++;
                        j=(j>99)? 99:j;
                        buf2[j]=0;
                } else break;
        }
        if (j==99) { return(1); }

        val=atof(buf2);
        if (base!=buf)
        {
                if (tipo==1) val=val/div[m];
                if (tipo==2) val=val*div[m];
        }

        if (val>1e10)
        {
                printf("\n\nValor muy grande.Tipo: %d (%g) \n-> %s",tipo,div[m],buf-1);
                printf("\n    %s long:%d",buf2,strlen(buf2));
                printf("\n    %15.4f",val);
        }
        for(i=0;i<l;i++)     if ( !isdigit(base[i]) && base[i]!='.' ) return( (base+i) -buf +1);
    return(1);
}

int ObjetoInfosel::ProcesarReg(char *buf)
//Regresa
// 1 No se encontro descripcion para algun campo
// 2 No se pudo procesar todo el registro
// 3 No se pudo procesar algun campo
{
        char bb[32],c,salida=0;
        char cond[100], conds[100],*datoscampo,temp1[100],td[50],temp2[100];
        char temp3[1024],temp4[1024],
        char nvalor[50];
        unsigned long pos=2,desp;
        double número;

        strcpy(bb,"ec: t=< >, nf=< >");
        bb[7]=buf[1];

        desp=SacarCadena((unsigned char *)buf+3,(unsigned char *)nvalor,49,',');
        pos=desp+3;
        if (mensajes) printf("\n[%s]:",nvalor);

        //Preparamos el registro final en temp3
        temp3[0]=buf[1];
        temp3[1]=buf[2];
        temp3[2]=':';
        temp3[3]=0;
        strcat(temp3," !NOM=<");
        strcat(temp3,nvalor);
        strcat(temp3,">,");
        strcpy(cond,temp3);

        while(!salida)
        {
```

```
//Obtenemos la letra que identifica al campo en infosel
c=buf[pos];
if (c==3) goto Fin;

//Copiamos a conds la plantilla de bb
strcpy(conds,bb);

//Insertamos el caracter que define al campo en el lugar justo
conds[15]=c;

// Buscamos en la tabla de equivalecias de campos
datoscampo=ECampos.BuscarUltimoRegistro(conds);
if (datoscampo==NULL)
{
        printf("\nNo se puede encontrar ninguna descripcion para el campo: %c ",c);
        printf("\nSe busco con la condicion: %s",conds);
        printf("\nRegistro:\n%s\n",buf);
        return(1);
}
datoscampo+=sizeof(long);
if (mensajes>1) printf("\n%15s",datoscampo);

//Obtenemos de la tabla el tipo de dato de este campo
ECampos.SacarDato(5,datoscampo,td,50);
ECampos.FormatearRegistro(td);

//Obtenemos da la tabla el nombre en las tablas del campo
ECampos.SacarDato(3,datoscampo,temp2,100);

// Escribimos en temp3
strcat(temp3,temp2);
strcat(temp3,"=<");

if (mensajes>1) printf("-> %4s [%s] ",temp2,td);

if (td[0]=='C') // El campo es una cadena
{
        desp=SacarCadena((unsigned char *)buf+pos+1,(unsigned char *)temp1,100,',');
        strcat(temp3,temp1);
        if (mensajes>1) printf(" -> %s",temp1);
        pos+=desp+1;
}
else
if (td[0]=='N') // El campo es un Numero
{
        desp=SacarNumero((unsigned char *)buf+pos+1,número);
        sprintf(temp1,"%-12.4f",número);
        QuitarCeros((unsigned char *)temp1);
        strcat(temp3,temp1);
        if (mensajes>1) printf(" -> %s",temp1);
        pos+=desp;
}
else
if (td[0]=='H') // El campo es una Hora
{
        desp=SacarHora((unsigned char *)buf+pos+1,(unsigned char *)temp1);
        strcat(temp3,temp1);
        if (mensajes>1)printf(" -> %s",temp1);
        pos+=desp+1;
}
else
if (td[0]=='F') // El campo es una Fecha
```

```
                {
                        desp=SacarFecha((unsigned char *)buf+pos+1,(unsigned char *)temp1);
                        strcat(temp3,temp1);
                        if (mensajes>1) printf(" -> %s",temp1);
                        pos+=desp+1;
                }
                else
                if (td[0]=='B') // El campo es un conjunto de bytes
                {
                        desp=atol(td+1);
                        memcpy(temp1,buf+pos+1,desp);
                        temp1[desp]=0;
                        strcat(temp3,temp1);
                        if (mensajes>1) printf(" -> %s",temp1);
                        pos+=desp+1;
                }
                else    salida=1;
                strcat(temp3,">,");
        }
        printf("\n NO SE PUDIERON PROCESAR TODOS LOS CAMPOS");
        return(3);
Fin:
        TransformarAInvision1(temp3,temp4);
        ActualizarRegistro(nvalor,temp4);

        if (mensajes) printf("\n%s",temp3);
        desp=Datos.ReemplazarRegistro(cond,temp3,0);
        if (desp)
        {
                if (mensajes) printf("\n%s:",nvalor);
                if (mensajes) printf("\nNo se puede actualizar el registro, se intenta crearlo ... ");
                desp=Datos.GuardarRegistro(temp3);
                if (desp)
                {
                        printf("\n%s:",nvalor);
                        printf("Error al crear registro, CÓDIGO%d",desp),
                        printf("\n\n%s",buf);
                        printf("\n\n%s",temp3);
                }

                if (mensajes) printf("OK");
        }
        return(0);
}

void ObjetoInfosel::TransformarAInvision1
(char *reg, char *invreg)
{
        unsigned int nc=Datos.ContarDatosInser(reg),i;
        unsigned char nombre[50],valor[500],temp1[100],temp2[50];
        char *dcampo;

        if (mensajes>4) printf("\n\n%s\n",reg);
        *invreg=0;
        for (i=1;i<nc;i++)
        {
                strcpy((char *)temp1,"ec: t=< >, ");
                temp1[7]=reg[0];
                strcat((char *)temp1,"nt=<");
                strcat((char *)temp1,(char *)nombre);
                strcat((char *)temp1,">,");
```

```
                //printf("\nBuscando %s :",temp1);
                dcampo=ECampos.BuscarUltimoRegistro((char *)temp1);
                if (dcampo==NULL) { printf(" ERROR!!! : %s", temp1); goto fin; }

                dcampo+=sizeof(long);
                Datos.SacarDato(4,(char *)dcampo,(char *)temp2,50);

                if (mensajes>4) printf("\n[%s] - %s=%s ",temp2,nombre,valor);

                // MUCHISIMO CUIDADO, AQUI NADA PREVIENE QUE SE
                // ESCRIBA EN MEMORIA AJENA AL PROCESO
                sprintf((char *)temp1,"%c%s%c%s",0x1e,(char *)temp2,0x1f,(char *)valor);
                strcat((char *)invreg,(char *)temp1);
        }

fin:
        if (mensajes>4) printf("\n----------");
        if (mensajes>4) printf("\n%s",invreg);
        if (mensajes>4) printf("\n----------");
}

void ObjetoInfosel::CrearRegisroVacio(char tipo,char stipo,char *salida,unsigned int maxs)
{
        char ntab[3],*destab,temp1[100],temp2[100],*descampo,ninv[100],temp3[500];
        int ncampos,i;

        ntab[0]=tipo; ntab[1]=stipo; ntab[2]=0; salida[0]=0;

        destab=Datos.DesTabla(ntab);
        if (destab==NULL)
        {
                printf("\nNo se encuentra la descripcion de la tabla: %s ",destab);
                return;
        }
        destab+=sizeof(long);
        ncampos=Datos.ContarCamposDes(destab);
        //printf("\n%d campos: %s\n ",ncampos,destab);

        for (i=1;i<=ncampos;i++)
        {
                Datos.SacarNombreCampoDes((unsigned char *)destab,i,(unsigned char *)temp1,100);
                sprintf(temp2,"ec: t=<%c>, nt=<%s>,",tipo,temp1);
                descampo=ECampos.BuscarUltimoRegistro(temp2);
                if (descampo==NULL)
                {
                        printf("Crear Registro Vacio\nNo se encontro el nombre de campo '%s' del tipo %c \nLa
condicion '%s' no funciono",temp1,tipo,temp2);
                        return;
                }
                descampo+=sizeof(long);
                Datos.SacarDato(4,descampo,ninv,100);
                sprintf((char *)temp3,"%c%s%c%s",0x1e,(char *)ninv,0x1f,"0");
                strcat(salida,temp3);
        }
        sprintf(ntab,"%c",0x1c);
        strcat(salida,ntab);

}

void ObjetoInfosel::TransformarAInvision2
(char tipo,char *reg, char *invreg)
{
```

```
        unsigned int nc=Datos.ContarCamposDes(reg),i;
        unsigned char ncampo[50],valorc[500],temp1[1024],ninvc[50];
        char *vals,*tabla,fininv[2],*dcampo;
        unsigned long numtab;

        numtab=*(unsigned long *)reg;
        vals=reg+sizeof(long);

        tabla=Datos.DesTabla(numtab);
        if (tabla==NULL)
        {
                invreg[0]=0;
                return;
        }
        tabla+=sizeof(long);
        nc=ECampos.ContarCamposDes(tabla);

        if (mensajes>4) printf("\n\n%s\n",reg);

        *invreg=0;
        for (i=1;i<=nc;i++)
        {
                if (dcampo==NULL) { printf(" ERROR!!! : %s", temp1); *invreg=0; return; }

                // Obtenemos el nombre en Invision de el campo solicitado
                dcampo+=sizeof(long);
                Datos.SacarDato(4,(char *)dcampo,(char *)ninvc,50);

                //Obtenemos el valor del campo del registro que estamos transformando
                Datos.SacarDato(i,reg+sizeof(long),(char *)valorc,500);

                //Lo escribimos todo al un registro temporal

                sprintf((char *)temp1,"%c%s%c%s",0x1e,ninvc,0x1f,valorc);
                strcat(invreg,(char *)temp1);
        }

        fininv[0]=0x1c; fininv[1]=0;
        strcat(invreg,fininv);
        if (mensajes>4) printf("\n----------");
        if (mensajes>4) printf("\n%s",invreg);
        if (mensajes>4) printf("\n----------");
}

void ObjetoInfosel::ActualizarRegistro
(char *nompub,char *reginv)
{
        char temp1[1024],numinv[20],nombre[100],*pub,c;

        if (Pubs==NULL) { printf(" ActReg: Oubs==NULL! ");return; }
        sprintf((char *)temp1,"p: nom=<%s>",nompub);

        pub=Pubs->BuscarUltimoRegistro(temp1);
        if (pub==NULL) return;

        pub+=sizeof(long);
        Pubs->SacarDato(4,pub,numinv,20);
        Pubs->SacarDato(5,pub,nombre,100);

        c=Pubs->separador;
        sprintf((char *)temp1,"%c%s%c%s%c%s",c,nombre,c,numinv,c,reginv);
        PendPub+temp1;
```

```
        PendPub.ultimo->tipo=1;
        //printf("\n----------");
        //PendPub.Listar();
}

void ObjetoInfosel::QuitarCeros(unsigned char *buf)
{
        unsigned int posp=0,l=strlen((char *)buf),i;

        for (i=0;i<l;i++) if (buf[i]=='.') { posp=i; i=l+1; }
        for (i=posp +1;i<l;i++)
        {
                if (buf[i]!='0') posp=i+1;
        }
        buf[posp]=0;
}

void ObjetoInfosel::ProcesarPágina(char *cab,char *RegListo)
{
        ListaLigada *l;
        char *pos,buf[1024],salida=0,c,nombre[50];
        unsigned long nreng,ncol,maxlet,maxlett,aporig=0,len=strlen(RegListo),maxcol=0,apbuf;

        l=new ListaLigada;
        if (l==NULL) return;

//Sacamos el número de página
        pos=EncontrarCaracter(cab,',',2); pos++;
        SacarCadena(nombre,50,pos,',');
        if (mensajes) printf("\nProcesando página: %s ",nombre);
        *l+nombre;

//Sacamos su nombre
        pos=EncontrarCaracter(cab,',',3); pos++;
        SacarCadena(nombre,50,pos,',');
        if (mensajes) printf("%s",nombre);
        *l+nombre;

//Buscamos el número de renglones
        pos=EncontrarCaracter(cab,',',4); pos++;
        nreng=atol((char *)pos);
        if (pos==NULL || nreng==0)
        {
                printf("\nNo se encontro el número de renglones.");
                return;
        }
//Y lo guardamos en la lista
        SacarCadena(nombre,50,pos,',');
        if (mensajes) printf("Reng: %s",nombre);
        *l+nombre;

//Buscamos el número de columnas
        pos=EncontrarCaracter(cab,',',5); pos++;
        ncol=atol((char *)pos);
        if (pos==NULL || ncol==0)
        {
                printf("\nNo se encontro el número de columnas.");
                return;
        }
//Y lo guardamos en la lista
        SacarCadena(nombre,50,pos,',');
        if (mensajes) printf("Cols: %s",nombre);
```

```
        *l+nombre;


//Buscamos el inicio de la informacion
        pos=EncontrarCaracter(RegListo,',',4); pos++;
        if (*pos!="")
        {
                printf("\nNo se puede procesar página, no se encontro el\ncaracter %c.",*pos);
                return;
        }

        memset(buf,' ',1023);
        buf[1022]=0;
        pos++; aporig=pos-RegListo; apbuf=0; maxlet=maxlett=0;

        FILE *f;
        f=fopen("1temp.txt","w");
        fprintf(f,"%s",pos);
        fprintf(f,"\n\n////////////////////////////////PROCESO////////////////////////////////\n\n",pos);

        while(aporig<len)
        {
                c=RegListo[aporig];
                if (c=='\n')
                {
                        buf[maxlet]=0;
                        fprintf(f,"->%s<-\n",buf);
                        fflush(f);
                        if (mensajes>1) printf("\n[%s] %d",buf,maxlet);
                        maxlett= (maxlett<maxlet)? maxlet:maxlett;
                        apbuf=0; maxlet=0;
                        *l+buf;
                }
                if (c>31)
                {
                        buf[apbuf]=c;
                        apbuf=(apbuf<1022)?  apbuf+1:1022;
                        if (c!=32) maxlet=apbuf;
                }
                aporig++;
        }

        sprintf(buf,"#M %d",maxlett);
        *l+buf;

        LP.GuardarPágina(l);
        if (mensajes) printf("\nNombres de páginas guardadas:");
        if (mensajes) LP.Listar();
        if (mensajes) printf("\nFin");
        if (l!=NULL) l->Limpiar();

        fclose(f);
}
```

CtrlInv.h

```
#include "Listalig.h"
#include "invision.h"
#include "tablas.h"

class ControlInvision
{
public:
        Tabla *Pubs;
        Invision *Inv;
        char *ObjInfo;

        ControlInvision();
        void ProcesarSolicitud(char *sucio);
        void PublicarNoticia(char *nombre);
        void ProcesarRegistro(char *nreg);
        void ProcesarPágina(char *nreg);

        void ProcesarPendientes();
        void Prueba(char *nreg);
};
```

CtrlInv.cpp

```
#include "ctrlinv.h"
#include "pagvir.h"
#include "objinfo.h"
#include "lpags.h"
#include <stdio.h>
#include <string.h>

ControlInvision::ControlInvision()
{
        Pubs=NULL;
        Inv=NULL;
        ObjInfo=NULL;
}

void ControlInvision::ProcesarSolicitud(char *sucio)
{
        printf("\nSolicitud: ");
        if (*(sucio)=='N')
        {
                printf("\nUna noticia sera publicada.");
                PublicarNoticia(sucio);
        }
        else
        if (*(sucio)=='p')
        {
                printf("\nUna página .");
                ProcesarPágina(sucio);
        }
        else
        if (*(sucio)=='p')
        {
                printf("\nUna prueba.");
                Prueba(sucio);
        }

        else
```

```
        {
                Inv->DestruirObjeto(sucio,0,"La solicitud fue mal escrita, por favor verifiquela.");
                return;
        }
}

void ControlInvision::PublicarNoticia(char *sucio)
{
        char temp[100],c;
        char buf[5000];
        char *mem;
        unsigned long pos=0,nl=0,nc=0,ncmax=0,tam=0;
        FILE *f;
        PáginaVirtual pag;

        if (Inv==NULL)
        {
                printf("\nNo se puede procesar solicitud");
                sprintf(temp,"Error interno en el servidor.",sucio);
                printf("\n%s",temp);
                Inv->DestruirObjeto(sucio,0,temp);
                return;
        }

        strcpy(temp,sucio);
        strcat(temp,".txt");

        f=fopen(temp,"r");
        if (f==NULL)
        {
                printf("\nNo se puede procesar solicitud");
                sprintf(temp,"La noticia que ud. solicito no existe.",sucio);
                printf("\n%s",temp);
                Inv->DestruirObjeto(sucio,0,temp);
                return;
        }

        while(!feof(f))
        {
                c=getc(f);
                buf[pos]=c;
                if (pos<5000) pos++; else pos=4999;
                buf[pos]=0;
                nc++;
                if (c=='\n')
                {
                        nl++;
                        ncmax=(nc>ncmax)? nc:ncmax;
                        nc=0;
                }
        }
        ncmax++,
        tam=ncmax*nl;
        mem=(char *)malloc(tam);
        if (mem==NULL)
        {
                printf("\nNo se puede procesar solicitud");
                sprintf(temp,"No hay memoria suficiente para procesar la solicitud.",sucio);
                printf("\n%s",temp);
                Inv->DestruirObjeto(sucio,0,temp);
                return;
        }
```

```
	pag.buf=mem;
	pag.Geometria(ncmax,nl);
	pag.Cls();
	pag.tab=0;
	///////// Inicia el proceso de publicacion
		unsigned long numinv=Inv->CrearPágina(sucio,ncmax,nl);
		if (numinv==0)
		{
			printf("\nNo se puede procesar solicitud");
			sprintf(temp,"No se pudo crear el objeto con las API's.",sucio);
			printf("\n%s",temp);
			Inv->DestruirObjeto(sucio,0,temp);
			free(mem);
			return;
		}
		pag << buf;
		if (!Inv->PublicarPágina(sucio,numinv,tam,(unsigned char *)mem))
		{
			printf("\nNo se puede procesar solicitud");
			sprintf(temp,"No se pudo publicar el objeto con las API's.",sucio);
			printf("\n%s",temp);
			Inv->DestruirObjeto(sucio,0,temp);
			free(mem);
			return;
		}
		//Pubs->AgregarPublicable(numinv,sucio);
	///////// Termina el proceso de publicacion

	printf("\nSi se pudo procesar solicitud");
	sprintf(temp,"Noticia publicada.",sucio);
	printf("\n%s",temp);
	free(mem);
	fclose(f);
}

void ControlInvision::ProcesarRegistro(char *nreg)
{
	char temp1[1000],temp2[200],nobj[100],*dato,invreg[1024],nvalor[100];
	ObjetoInfosel *oi=(ObjetoInfosel *)ObjInfo;
	unsigned long numinv,res;
	unsigned short nmasc;

	strcpy(nobj,nreg+3);
	sprintf(temp2,"6%c: !nom=<%s>",nreg[1],nobj);
	dato=oi->Datos.BuscarUltimoRegistro(temp2);
	//printf("\Buscando en Datos con condicion: '%s'",temp2);
	if (dato==NULL)
	{
			printf("\nNo se puede procesar solicitud");
			sprintf(temp2,"El objeto no se encuentra: '%s'",nobj);
			printf("\n%s",temp2);
			Inv->DestruirObjeto(nreg,0,temp2);
			return;
	}

	oi->TransformarAInvision2(*nreg,dato,invreg);
	oi->Datos.SacarDato(1,nreg+sizeof(long),nvalor,100);


	nmasc=6700+(nreg[1]-'A'+1);
	numinv=Inv->CrearRegistro(nreg,nmasc);
```

```
	if (!numinv)
	{
			printf("\nNo se puede procesar solicitud");
			sprintf(temp1,"No se pudo crear el objeto con las API's.");
			printf("\n%s",temp1);
			Inv->DestruirObjeto(nreg,0,temp1);
			return;
	}

	sprintf(temp2,"p: t=<%c>, st=<%c>, nom=<%s>, numinv=<%d>,
nompub=<%s>",*nreg,nreg[1],nobj,numinv,nreg);
	//printf("\n%s",temp2);
	res=Pubs->GuardarRegistro(temp2);
	if (res)
	{
			printf("\nNo se puede procesar solicitud");
			sprintf(temp2,"No se pudo registrar el objeto como publicable (%d).",res);
			printf("\n%s",temp2);
			Inv->DestruirObjeto(nreg,0,temp2);
			return;
	}
	Pubs->ListarDatos();

	//printf("\nPublicando:\n%s",temp1);

	Inv->PublicarPrimerRegistro(nreg,numinv,strlen(temp1),(unsigned char *)temp1);
	//printf("\nRegistro Invision:\n%s",invreg);
	Inv->PublicarPrimerRegistro(nreg,numinv,strlen(invreg),(unsigned char *)invreg);
	//printf("...OK");
}

void ControlInvision::Prueba(char *nreg)
{
	char temp[100];
	unsigned long numinv;

	sprintf(temp,"%c1%cPrueba%c",0x1e,0x1f,0x1c);
	numinv=Inv->CrearRegistro(nreg,6799);
	Inv->PublicarPrimerRegistro(nreg,numinv,strlen(temp),(unsigned char *)temp);
	printf("\nRegistro de prueba:%s",temp);
}

void ControlInvision::ProcesarPendientes()
{
	ObjetoInfosel *oi;
	ElementoLista *e;
	char nombre[100],numinvc[100],*pub,act[1024];
	unsigned long numinv,i;
	int ncomas=0,x,y;

	oi=(ObjetoInfosel *)ObjInfo;
	e=oi->PendPub.primero;

	while(e!=NULL)
	{
		if (e->Mem!=NULL)
		{
			if (e->tipo==1)
			{
				printf("\nObjeto publicable: %s ",e->Mem);
				Pubs->SacarDato(1,(char *)e->Mem,(char *)nombre,100);
```

```
                                        Pubs->SacarDato(2,(char *)e->Mem,(char *)numinvc,100);
                                        numinv=(unsigned long)atol(numinvc);
                                        for (i=0;i<strlen((char *)e->Mem);i++)
                                        {
                                                if (e->Mem[i]==Pubs->separador) ncomas++;
                                                if (ncomas==3) break;
                                        }
                                        i++;
                                        pub=(char *)(e->Mem+i);
                                        printf("\n\nPUBLICANDO PENDIENTES...");
                                        printf("\nPublicando: <%d> - [%s]\n",numinv,nombre);
                                        printf("\n%s",pub);
                                        Inv->ActualizarRegistro(nombre,numinv,strlen((char *)e-
>Mem+ncomas+1),(unsigned char *)pub);
                                }
                        }
                        e=e->sig;
                }
                while(oi->PendPub.primero!=NULL) oi->PendPub.BorrarPrimero();
                //printf("\n----------");
}

void ControlInvision::ProcesarPágina(char *sucio)
{
        ListaLigada *ll;
        ObjetoInfosel *oi;
        char temp2[100],*m;
        unsigned int nreng,ncol,i,res;
        unsigned long numinv;

        oi=(ObjetoInfosel *)ObjInfo;

        ll=oi->LP.Existe(sucio+1);

        if (ll==NULL)
        {
                        printf("\nNo se puede procesar solicitud");
                        sprintf(temp2,"No existe la página solicitada: '%s'.",sucio+1);
                        printf("\n%s",temp2);
                        Inv->DestruirObjeto(sucio,0,temp2);
                        return;
        }

        // Obtenemos el número de renglones de la página
        m=(*ll)[2];
        if (m!=NULL) nreng=atol(m);
        else
        {
                        printf("\nNo se puede procesar solicitud");
                        sprintf(temp2,"Error en el formato de la página.",sucio+1);
                        printf("\n%s",temp2);
                        Inv->DestruirObjeto(sucio,0,temp2);
                        return;
        }

        // Obtenemos el número de columnas de la página
        if (ll->ultimo!=NULL) { if (ll->ultimo->Mem!=NULL)
        {
                        if (*ll->ultimo->Mem=='#' && *(ll->ultimo->Mem+1)=='M')
                        {
                                ncol=atoi((char *)ll->ultimo->Mem+2);
                        }
```

```
}}
else
{
        m=(*ll)[3];
        if (m!=NULL)
                ncol=atol((char *)m);
        else
        {
                printf("\nNo se puede procesar solicitud");
                sprintf(temp2,"Error en el formato de la página.",sucio+1);
                printf("\n%s",temp2);
                Inv->DestruirObjeto(sucio,0,temp2);
                return;
        }
}

// Si ncol o nreng son cero, vamonos.
if (!ncol || !nreng)
{
                printf("\nNo se puede procesar solicitud");
                sprintf(temp2,"La página no contiene informacion.",sucio+1);
                printf("\n%s",temp2);
                Inv->DestruirObjeto(sucio,0,temp2);
                return;
}

// Creamos la página en Invision
numinv=Inv->CrearPágina(sucio,ncol,nreng);
if (!numinv)
{
                printf("\nNo se puede procesar solicitud");
                sprintf(temp2,"No se pudo crear el objeto con las API's.");
                printf("\n%s",temp2);
                Inv->DestruirObjeto(sucio,0,temp2);
                return;
}


//Registramos el objeto en la lista de publicables
sprintf(temp2,"p: t=<4>, nom=<%s>, numinv=<%d>, nompub=<%s>",
        sucio+1,numinv,sucio);
//printf("\n%s",temp2);
res=Pubs->GuardarRegistro(temp2);
if (res)
{
                printf("\nNo se puede procesar solicitud");
                sprintf(temp2,"No se pudo registrar el objeto como publicable (%d).",res);
                printf("\n%s",temp2);
                Inv->DestruirObjeto(sucio,0,temp2);
                return;
}

//Publicamos renglones
for (i=0;i<nreng;i++)
{
        m=(*ll)[i+4];
        if (m!=NULL)
        {
                printf("\n%s",(char *)m);
                Inv->PublicarPágina(sucio,numinv,strlen(m),0,i,(unsigned char *)m);
        }
```

```
        }
}
```

Lpags.h

```
#ifndef LISTAPÁGINAS
#define LISTAPÁGINAS

#include "tablas.h"

class ListaPáginas
{
public:
        char *oi,mensajes;
        ListaLigada *PendPub,datos;
        Tabla *Pubs;

        ListaPáginas();
        int GuardarPágina(ListaLigada *l);
        ListaLigada *ExisteLista(char *nombre);
        void EncontrarDiferencias(ListaLigada *lorig,ListaLigada *l,ListaLigada *difs);
        void BuscarDifCadenas(char *mem1,char *mem2, unsigned int nreng, ListaLigada *Dif);
        ListaLigada *Existe(char *nombre) { return(ExisteLista(nombre)); }

        void Listar();
};

#endif
```

Lpags.cpp

```
#include "lpags.h"
#include <stdio.h>
#include <string.h>

ListaPáginas::ListaPáginas()
{
        oi=NULL;
        Pubs=NULL;
        mensajes=0;
}

int ListaPáginas::GuardarPágina(ListaLigada *lorig)
//1     La lista l_ no esta alojada
//2 No se pudo agregar otro elemento de LL
{
        ListaLigada *l=NULL,difs;
        ElementoLista *e;
        char temp[100],*pub,npub[50],ninv[50],c,*dif;


        if (lorig==NULL) return(1);
        if (lorig->primero->Mem!=NULL)
                l=ExisteLista((char *)lorig->primero->Mem);
        else return(1);

        if (l==NULL)
        {
                l=(ListaLigada *)datos.Agregar(sizeof(ListaLigada),NULL);
                if (l==NULL) return(2);
```

```
                l->Inic();
                e=lorig->primero;
                while(e!=NULL)
                {
                        *l+(char *)e->Mem;
                        e=e->sig;
                }
        }
        else
        {
                if (Pubs!=NULL)
                {
                        sprintf(temp,"p: nom=<%s>",(char *)lorig->primero->Mem);
                        pub=Pubs->BuscarUltimoRegistro(temp);
                        if (pub!=NULL)
                        {
                                EncontrarDiferencias(lorig,l,&difs);
                                if (difs.total>0)
                                {
                                        pub+=sizeof(long);
                                        Pubs->SacarDato(5,pub,npub,50);
                                        Pubs->SacarDato(4,pub,ninv,50);
                                        c=Pubs->separador;
                                        e=difs.primero,
                                        while(e!=NULL)
                                        {
                                                dif=(char *)e->Mem;
                                                sprintf(temp,"%c%s%c%s%c%d%c%d%c%s%c",
                                                c,npub,c,ninv,c,
                                                *(unsigned int*)(dif+sizeof(int)), c,
                                                *((unsigned int*)dif)-4, c,
                                                (dif+sizeof(int)*2),c );////fin sprintf

                                                (*PendPub)+temp;        PendPub->ultimo->tipo=2;
                                                e=e->sig;
                                        } //WHile

                                }
                                //SustituirInfo(lorig,l);
                        }
                }

        }

        l=NULL;
        return(0);
}

ListaLigada *ListaPáginas::ExisteLista(char *nombre)
{
        ElementoLista *e;
        char *lret;

        e=datos.primero;
        ListaLigada *l;
        while(e!=NULL)
        {
                if (e->Mem!=NULL)
                {
                        l=(ListaLigada *)e->Mem;
                        if (l->primero) if (l->primero->Mem)
```

```
                {
                        if (!strcmp((char *)l->primero->Mem,nombre))
                        {
                                lret=(char *)l;
                                l=NULL; e=NULL;
                                return((ListaLigada *)lret);
                        }
                }
        }
        e=e->sig;
    } //while(e!=NULL)
    l=NULL;
    return(NULL);
}

void ListaPáginas::Listar()
{
    ElementoLista *e=datos.primero;
    ListaLigada *l;

    while(e!=NULL)
    {
        if (e->Mem!=NULL)
        {
            l=(ListaLigada *)e->Mem;
            if (l->primero) if (l->primero->Mem)
            {
                    if ((*l)[0]!=NULL) printf("\n%s: ",(*l)[0]);
                    if ((*l)[1]!=NULL) printf("%s ",(*l)[1]);
            }//if (l->primero) if (l->primero->Mem)
        }//if (e->Mem!=NULL)
        e=e->sig;
    } //while(e!=NULL)
    l=NULL;
}

void ListaPáginas::EncontrarDiferencias
(ListaLigada *lorig,ListaLigada *l,ListaLigada *difs)
{
    unsigned int i=0;
    char *mem1,*mem2,salida=0;

    difs->Limpiar();
    while(!salida)
    {
        mem1=(*lorig)[i]; mem2=(*l)[i];
        if (mem1!=NULL && mem2!=NULL)
        {
            if (strcmp(mem1,mem2))
            {
                if (mensajes) printf("\n DIF [%s]\n  [%s]",mem1,mem2);
                BuscarDifCadenas(mem2,mem1,i,difs);
            }
        } else salida=1;
        i++;
    }
}

void ListaPáginas::BuscarDifCadenas
(char *mem1,char *mem2,
 unsigned int nreng, ListaLigada *Dif)
{
```

```
        unsigned int l1=strlen(mem1), l2=strlen(mem2),i,tope,post=0;
        char modo=0,temp[1024],*tdat;

        tope=(l1>l2)? l1:l2;
        tdat=temp+sizeof(int)*2;
        *(unsigned int *)temp=nreng;

        for (i=0;i<tope;i++)
        {
                if (modo==0 && mem1[i]!=mem2[i])
                {
                        modo=1;post=0;
                        *(unsigned int *)(temp+sizeof(int))=i;
                }
                if (modo==1 && mem1[i]!=mem2[i])
                {
                        tdat[post]=mem2[i];
                        post=(post<1000)? post+1:1000;
                        tdat[post]=0;
                }
                if (modo==1 && mem1[i]==mem2[i])
                {
                        Dif->Agregar(sizeof(int)*2+strlen(tdat)+1,(unsigned char *)temp);
                        //printf("\nXDIF: %d, %d : '%s'",*(unsigned int *)temp,*(unsigned int
*)(temp+sizeof(int)),tdat),
                        modo=0;
                }

        }
}
```

Main.cpp

```
#include "p2.h"
#include "objinfo.h"
#include "invision.h"
#include "ctrlinv.h"
#include <stdio h>
#include <conio.h>

ListaLigada Pubs;

void main()
{
        ObjetoInfosel oi;
        char salida=0,temp[100];
        char c;
        int res;
        Invision Inv;
        ControlInvision ci;
        Tabla Pubs;

        //Asignamos Relaciones entre objetos ...
        ci.Inv=&Inv;
        ci.Pubs=&Pubs;
        ci.ObjInfo=(char *)&oi;
        oi.Pubs=&Pubs;
        oi.LP.Pubs=&Pubs;

        //No quiero noticias ahora
        oi.noticias=1;
```

```
oi.mensajes=0;

Pubs.NuevaTabla("p: t,st,nom,numinv,nompub");


Inv.ArrancarPublisher("INFOSEL");
Inv.ServicioEnLinea();


oi.info.UsarArchivo("infosel.dt");
printf("\nCRC=%d",oi.info.QueCRC((unsigned char *)"Hola, esto es una prueba"));

printf("\nLlamando al sistema...");
system("del n*.txt");

printf("\nSe inicia el proceso de informacion ...\n");
while(!salida)
{
        oi.ProcesarInfo();
        ci.ProcesarPendientes();
        if (oi.mensajes)
        {
                printf("\nCambio oi.mensajes");
                getch();
        }
        if (kbhit())
        {
                c=getch();
                if (c==27) salida=1;
                if (c=='1') Pubs.ListarDatos();
                if (c=='2') oi.ECampos.ListarDatos();
                if (c=='3') oi.Datos.ListarDatos();
                if (c=='4') Pubs.ListarDatos();
                if (c=='5') oi.PendPub.Listar();
                if (c==' ') getch();
                printf("\n");
        }
        Inv.EsperarMensaje();
        if (Inv.mensaje==1)
        {
                printf("\nProcesando: %s",Inv.obj->szName);
                ci.ProcesarSolicitud(Inv.obj->szName);
                printf("\n");

        }
        if (Inv.mensaje==3)
        {
                sprintf(temp,"p: nompub=<%s>,",Inv.obj->szName);
                printf("\nQuitando: %s [%s]",Inv.obj->szName,temp);
                res=Pubs.BorrarRegistro(temp);
                if (res) printf("\nError %d",res);
        }
}
printf("\nProceso terminado\nSalida: %d ",salida);
getch();
```

APÉNDICE J

Formatos de petición de información.

Los siguientes formatos describen como deben los usuarios solicitar la información de la plataforma digital.

Formatos de petición de índices.

Tipo de índice	Commando
Noticias	**NOTICIAS**
Páginas	**PÁGINAS**

Formato de petición de noticias.

'N' + *Numero de noticia* + [*Identificador de segmento*]

Cada petición de noticias debe empezar con la letra 'N', seguido del número de la noticia y el segmento. Cada noticia se divide en segmentos de 20 renglones hasta llegar al final. Los segmentos se identifican con las letras mayúsculas del alfabeto empezando por la A, el primer segmento no lleva ningún identificador.

Ejemplo.

Una noticia con el número 1000 y 50 renglones en total se dividirá en tres segmentos y el usuario deberá solicitarlos desde su terminal usando estos nombres:

N1000, N1000A y **N1000B**

Los números de las noticias deben ser consultados en el índice de noticias. Si la noticia solicitada no existe, se regresará un mensaje de error en la barra de información en la terminal.

Formato de petición de páginas.

'P' + *Numero de página*

Cada petición de páginas debe empezar con la letra 'P', seguido del número de la página. Las páginas se presentan en una sola ventana y solo existe la necesidad de llamarlas una sola vez.

Ejemplo.

Una página con el número 1003 debe ser solicitada con el nombre:

P1003

Los números de páginas deben ser consultados en el índice de páginas. Si la noticia solicitada no existe, se regresará un mensaje de error en la barra de información en la terminal.

Formato de petición de hechos de capitales.

'6A-' + *Nombre del valor* + *Serie del valor*

Cada petición de páginas debe empezar con la cadena '6A-', seguido del nombre del valor y su serie. Los nombres y series de valores son emitidos por la Bolsa Mexicana de Valores y se deben consultar en las fuentes de la misma.

Ejemplo.

Si se desea consultar un hecho de la acción TELMEX serie L, se debe usar la cadena:

6A-TELMEXL

No hay manera de consultar los nombres de valores desde el sistema.

Formato de petición de posturas de capitales.

'6B-' + *Nombre del valor* + *Serie del valor*

Cada petición de páginas debe empezar con la cadena '6B-', seguido del nombre del valor y su serie. Los nombres y series de valores son emitidos por la Bolsa Mexicana de Valores y se deben consultar en las fuentes de la misma.

Ejemplo.

Si se desea consultar una postura de la acción CEMEX serie B, se debe usar la cadena:

6B-CEMEXB

No hay manera de consultar los nombres de valores desde el sistema.

Formato de petición de hechos de warrants.

'6F-' + *Nombre del valor*

Cada petición de warrants debe empezar con la cadena '6F-', seguido del nombre del valor. Los nombres de valores son emitidos por la Bolsa Mexicana de Valores y se deben consultar en las fuentes de la misma.

Ejemplo.

Si se desea consultar un hecho del warrant TG167829A , se debe usar la cadena:

6F-TG167829A

No hay manera de consultar los nombres de valores desde el sistema.

Formato de petición de posturas de warrants.

'6G-' + *Nombre del valor*

Cada petición de warrants debe empezar con la cadena '6G-', seguido del nombre del valor. Los nombres de valores son emitidos por la Bolsa Mexicana de Valores y se deben consultar en las fuentes de la misma.

Ejemplo.

Si se desea consultar una postura del warrant CON605A, se debe usar la cadena:

6G- CON605A

No hay manera de consultar los nombres de valores desde el sistema.

Formato de petición de índices.

'6D-' + *Nombre del índice*

Cada petición de un índice debe empezar con la cadena '6D-', seguida del nombre del índice. Los nombres de los indices son emitidos por la Bolsa Mexicana de Valores y se deben consultar en las fuentes de la misma.

Ejemplo.

Si se desea consultar una postura índice IPC, se debe usar la cadena:

6D-IPC