1 Conceptos Básicos.

1.1 Sistemas de supervisión.

El primer sistema de supervisión de este tipo fue el Circuito Cerrado de Televisión (CCTV), el cual consiste en un cámara analógica que capta las imágenes en el formato comercial de televisión, y las envía por cable a un monitor de televisión (de punto a punto), donde despliega las imágenes; la señal de video no es compartida con receptores ajenos a este sistema, de ahí su nombre de circuito cerrado.

Algunos sistemas permiten cambiar el canal (como un aparato de televisión comercial) para recibir las imágenes de otras cámaras. Otros sistemas tienen motores analógicos que mueven la cámara en un semicírculo, para poder supervisar una área mayor, pero no se puede controlar a voluntad la posición de la cámara.

Los sistemas de CCTV actualmente tienen los siguientes usos:

-Video vigilancia y prevención del crimen.
-Supervisión de procesos industriales.
-Monitoreo de tráfico vehicular.
-Fotografía digital de circuito cerrado, el cual utiliza una aplicación de cómputo especial para exportar una imagen a una computadora.

Actualmente existen diversos sistemas de supervisión comerciales, tanto de CCTV como de videocámaras digitales.

En México se tiene el servicio de "*Prodigy Cam*", que consiste en videocámaras digitales IP compatibles alámbricas o inalámbricas; las videocámaras se conectan a una computadora personal, recupera y almacena imágenes en el momento en que el usuario solicita una nueva imagen por medio de internet; algunas videocámaras cuentan con movimiento en una dimensión.

1.2 Comunicación de las computadoras personales por medio del puerto paralelo.

1.2.1 El puerto paralelo.

El puerto paralelo, conocido también como puerto de la impresora o puerto *centronics*, es una interfaz de conexión entre una computadora y diversos periféricos que fue presentado por la compañía IBM en 1981. Se caracteriza porque envía 8 bits (un byte) al mismo tiempo, en vez de enviar un bit de información a la vez, tal como lo hace el puerto serial; con esta característica se pretendía imprimir más rápidamente en impresoras de matriz de alto rendimiento, al enviar la información en menos tiempo.

1.2.2 Arquitectura del puerto paralelo.

El puerto paralelo consta de un conector con 17 líneas de señal y 8 líneas de tierra; las líneas de señal se dividen en los siguientes grupos:

-Control, de cuatro líneas, que coordina la comunicación entre la computadora y el periférico, y la señalización

de establecimiento de comunicación (*handshake*).

-Estado, de cinco líneas, el cual sirve para la señalización de establecimiento de comunicación y para indicar la situación actual del periférico, y señalar falta de papel, ocupado o errores del periférico.

-Datos, de ocho líneas, para el envío de información de la computadora al periférico, y que posteriormente permitió la comunicación bi-direccional.

1.2.3 El estándar *IEEE 1284*.

Conforme se fue popularizando el uso del puerto paralelo para conectar computadoras personales con impresoras y otros periféricos, también surgieron los siguientes problemas:

-La tasa de transferencia máxima era de 150 kilobytes por segundo, pero dicha tasa dependía de los programas utilizados.

-No había una interfaz eléctrica estándar, por lo que había problemas al tratar de conectar dos plataformas distintas.

-La transferencia en cables externos tenía el límite de distancia de hasta 6 pies (1.8288 metros).

Para resolver los anteriores problemas, la organización *Network Printer Alliance* presentó el estándar ***IEEE 1284*** en marzo de 1994, el cual define la comunicación bi-direccional entre una computadora personal y otros dispositivos periféricos, permitiendo una transferencia promedio de 2 megabits por segundo, y teóricamente hasta 4 megabits por segundo.

1.2.4 Definiciones del estándar *IEEE 1284*.

El estándar ***IEEE 1284*** resuelve los problemas mencionados anteriormente al proveer las siguientes definiciones:

-Cinco modos de transferencia de datos.

-Un método para que la máquina huésped (generalmente una computadora) y el periférico se comunique entre sí, para determinar los modos de transferencia de datos que soporta cada uno, y escoger uno de ellos.

-Interfaz física, tanto de cables como de conectores.

-Interfaz eléctrica, manejadores y receptores, terminación, e impedancia.

1.2.4.1 Modos de transferencia.

Los cinco modos de transferencia de datos proveen un método de enviar datos entre la PC y el periférico (sentido directo), entre el periférico y la PC (sentido inverso) o de forma bi-direccional (half duplex); dichos modos se explican a continuación:

-Modo de compatibilidad, estándar o "*Centronics*", el cual opera sólo en sentido directo.

-Modo *Nibble*, el cual es de sentido inverso, usando cuatro bits a la vez usando las líneas de estado para datos.

.

-Modo de Octeto (*Byte mode*), el cual usa 8 bits a la vez, usando las líneas de estado.

-Puerto Paralelo Extendido (*Enhanced Parallel Port EPP*); este modo es bi-direccional, lee o escribe un byte en un ciclo -incluyendo el saludo (*handshake*), en vez de los cuatro ciclos que necesita el modo de compatibilidad.

-Puerto de Capacidades Extendidas (*Extended Capability Port*), que también es bi-direccional; las implantaciones de esta arquitectura usan un bus o tren de datos *ISA*, usan memorias intermedias (*buffers*), soporte para transferencias *DMA* (*Direct Memory Access*, acceso directo a memoria) y soporte para compresión de datos.

1.2.4.2 Interfaz física del estándar *IEEE 1284*.

La interfaz física de este estándar sirve para asegurar una alta tasa de transferencia en un cable de 10 metros entre el equipo huésped y el periférico, definiendo los conectores y las configuraciones de las conexiones de los cables.

1.2.4.2.1 Cables.

No existe un cable paralelo estándar, aunque la configuración más popular de este estándar es el que tiene un conector tipo A (*DB25*) macho de un extremo, y un conector tipo B (*Centronics* de 36 pernos) hembra en el otro; tiene de 18 a 25 cables, ya que los últimos ocho cables son de tierra, y opera correctamente con una tasa de transferencia de 10K bytes/segundo con un cable de 1.8 metros (6 pies).

Algunos parámetros que define para las configuraciones de los cables son los siguientes:

-Todas las señales son de un par trenzado con retorno de señal y de tierra.

-Cada señal y tierra debe tener una impedancia no balanceada de 62 ± 2 [Ω] en la banda de frecuencia de 4 a 16 [Mhz].

-La interferencia de cable a cable no debe ser mayor del 10%.

-El cable deberá tener un trenzado visible mínimo de 85% sobre el recubrimiento.

-La cubierta del cable deberá unirse al armazón del conector usando un método concéntrico de 360º; no se aceptan nudos en la conexión del cable.

-El ensamblado del cable que cumpla con este estándar deberá portar la leyenda "***IEEE Std 1284-1994 Compilant***" ("Acorde con el Estándar *IEEE 1284-1994*").

Hay varias configuraciones de cables ya definidas, donde indican el tipo de conector en cada extremo, e indica, para cada cable, qué señal lleva y a que perno se conecta en cada conector. Las configuraciones más comunes se muestran en la siguiente tabla:

Configuración:	Conector 1:	Conector 2:
AMAM	Tipo A macho	Tipo A macho
AMAF	Tipo A macho	Tipo A hembra
AB	Tipo A macho	Tipo B
AC	Tipo A macho	Tipo B
BC	Tipo B	Tipo C
CC	Tipo C	Tipo C

Tabla 1: Configuraciones de conectores.

Las longitudes estándar son de 1.8, 3.5, 6.09 y 9.14 metros (6, 10, 20 y 30 pies, respectivamente).

1.2.4.2.2 Conectores.

Como parte de la definición del estándar de la interfaz física y asegurar la compatibilidad mecánica entre distintos equipos y periféricos, se definen los siguientes conectores:

- **DB25** (tipo A), el cual es una conector de 25 pernos el cual se conecta a la computadora generalmente; cabe mencionar que este estándar reemplaza y mantiene compatibilidad con el cable del puerto paralelo de la computadora personal definido por IBM. Dicho conector se muestra en las dos imágenes siguientes.

Figura 1: *Conector DB25 macho en la parte superior, y conector DB25 hembra en la parte inferior.*

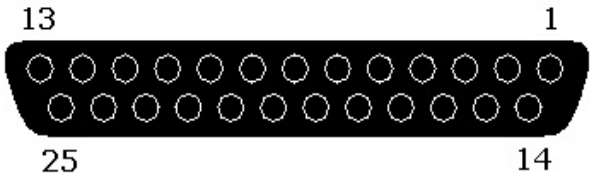

Figura 2: *Numeración de los pernos para el conector DB25.*

En la siguiente tabla se describen los pernos del conector DB25:

Número de perno	Nombre del perno	Descripción y/o función del perno.
1	nSTROBLE	Parpadeo; indica que existen datos válidos para transmitir.
2	D0	Bit de datos 0.
3	D1	Bit de datos 1.
4	D2	Bit de datos 2.
5	D3	Bit de datos 3.
6	D4	Bit de datos 4.
7	D5	Bit de datos 5.
8	D6	Bit de datos 6.
9	D7	Bit de datos 7.
10	nACK	Acknoledge (reconocimiento)
11	BUSY	Busy (Ocupada)
12	PE	Paper End (Fin de papel)
13	SEL	Select (Periférico seleccionado)
14	nAUTOFD	Autofeed (Autoalimentación de papel)
15	nERROR	Error
16	nINIT	Initialize (Inicializar)
17	nSELIN	Select In (periférico listo para entrada de datos).
18	GND	Tierra para la señal de *Stroble* (parpadeo).
19	GND	Tierra para los bits 1 y 2.
20	GND	Tierra para los bits 3 y 4.
21	GND	Tierra para los bits 5 y 6.
22	GND	Tierra para los bits 7 y 8.
23	GND	Tierra para las señales de Ocupada (*Busy*) y Error.
24	GND	Tierra para Fin del Papel (*Paper out*), Periférico Seleccionado (*Select*) y Reconocimiento (*Acknowledge*)
25	GND	Tierra para Autoalimentación (*AutoFeed*), Selección (*Select*) e Inicialización (*Initialize*).

Tabla 2: Funcionalidad de los pernos del conector DB25.

Los pernos cuyos nombres tienen el prefijo ‘n’ se refieren a que tienen un valor lógico inverso. Los pernos dos a nueve son de datos, donde D0 es el bit menos significativo, y D7 es el más significativo; dichos bit forman un byte.

- **Centronics** (tipo B), conector de 36 pernos de .085 pulgadas de longitud, de tipo *champ*, alineado respecto a un eje central, y con seguros de alambre. Reemplazó al cable ‘*centronics*’; generalmente se conecta al periférico, y cuya imagen aparece en la parte inferior.

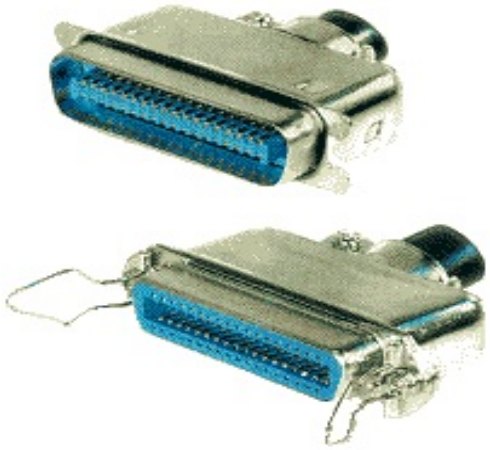

Figura 3: *Conector ‘Centronics’ hembra en la posición superior, y macho en la inferior.*

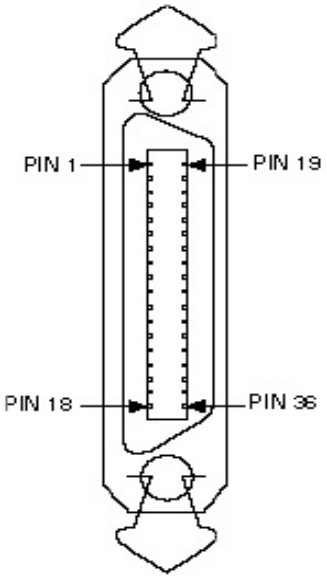

Figura 4: *Numeración de los pernos para el conector 1284-B.*

A continuación se muestra una tabla con la funcionalidad de los pernos del conector *1284-B*:

Número de perno	**Modo**				
	De compatibilidad o “centronics”	**Nibble**	**Modo de octeto (byte mode)**	**EPP**	**ECP**
1	nStrobe (parpadeo o estroboscopio)	HostClk (reloj del anfitrión, indica cuándo los datos son válidos).	HostClk (reloj del anfitrión, indica cuándo los datos son válidos).	nWrite (escritura)	HostClk (reloj del anfitrión, indica cuándo los datos son válidos).
2	D1	D1	D1	AD1	D1
3	D2	D2	D2	AD2	D2
4	D3	D3	D3	AD3	D3
5	D4	D4	D4	AD4	D4
6	D5	D5	D5	AD5	D5

7	D6	D6	D6	AD6	D6
8	D7	D7	D7	AD7	D7
9	D8	D8	D8	AD8	D8
10	nAcknoledge (reconocimiento)	PtrClk (reloj del periférico, indica cuándo los datos a enviar son válidos).	PtrClk (reloj del periférico, indica cuándo los datos a enviar son válidos).	Intr (interrupción)	PtrClk (reloj del periférico, indica cuándo los datos a enviar son válidos).
11	Busy (ocupada)	PtrBusy (puerto ocupado)	PtrBusy (impresora ocupada)	nWait (espera)	PtrBusy (impresora ocupada)
12	PError (error de la impresora)	AckDataReq (Reconocimiento de solicitud de datos)	AckDataReq (Reconocimiento de solicitud de datos)	User defined 1 (definido por el usuario 1)	AckDataReq (Reconocimiento de solicitud de datos)
13	Select (periférico seleccionado)	Xflag	Xflag	User defined 3 (definido por el usuario 3)	Xflag (bandera para indicar modo ECP).
14	nAutoFD (autoalimentación)	HostBusy (Anfitrión ocupado)	HostBusy (Anfitrión ocupado)	nDstrb	HostAck (reconocimiento del anfitrión)
15	Signal Ground (tierra para señales)	----------	----------	----------	----------
16	Logic Ground (tierra lógica)	----------	----------	----------	----------
17	Chassis Ground (tierra del chasís)	----------	----------	----------	----------
18	Peripheral Logic High (alto lógico del periférico)	----------	----------	----------	----------
19	Tierra para - *Strobe*	----------	----------	----------	----------
20	Tierra para D1	----------	----------	----------	----------
21	Tierra para D2	----------	----------	----------	----------
22	Tierra para D3	----------	----------	----------	----------
23	Tierra para D4	----------	----------	----------	----------

24	Tierra para D5	----------	----------	----------	----------
25	Tierra para D6	----------	----------	----------	----------
26	Tierra para D7	----------	----------	----------	----------
27	Tierra para D8	----------	----------	----------	----------
28	Tierra para - *ACK*, *PError* y *Select*	----------	----------	----------	----------
29	Tierra para *Busy* y *nFault*	----------	----------	----------	----------
30	Tierra para las señales *nAutoFd*, *nSelectIn* y *nInit*	----------	----------	----------	----------
31	nInit (inicializar)	nInit (inicialización)	nInit (inicialización)	nInit (inicialización)	nReverseRequest (petición inversa)
32	nFault (señal de fallo)	nDataAvail (datos disponibles)	nDataAvail (datos disponibles)	User Defined 2 (definido por el usuario 2).	nRequest (Petición del periférico).
33	Tierra	----------	----------	----------	----------
34	Tierra	----------	----------	----------	----------
35	Tierra	----------	----------	----------	----------
36	nSelectIn (selección de periférico).	1284 Activo (activo)	1284 Activo (activo)	nAStrb	1284 Activo (activo)

Tabla 3: Funcionalidad de los pernos de un conector tipo B.

Los nombres de los pernos con el prefijo ‘n’ indica que tiene un valor lógico inverso. Los pernos dos al nueve corresponden a los datos (D1 a D8), donde D1 es el bit menos significativo, y D8 es el más significativo; estos ocho bits conforman un byte.

- **Mini-Centronics** (tipo C), conector de 36 pernos de .050 pulgadas (*MDR36*), alineado respecto a un eje central y de seguros de presión; es de diseño más reciente pero que no fue muy aceptado.

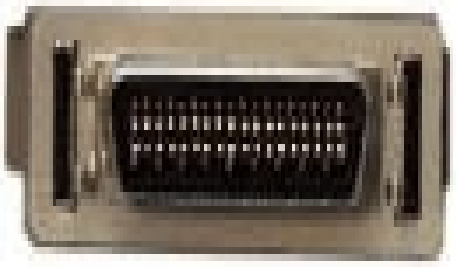

Figura 5: *Conector Mini - centronics macho.*

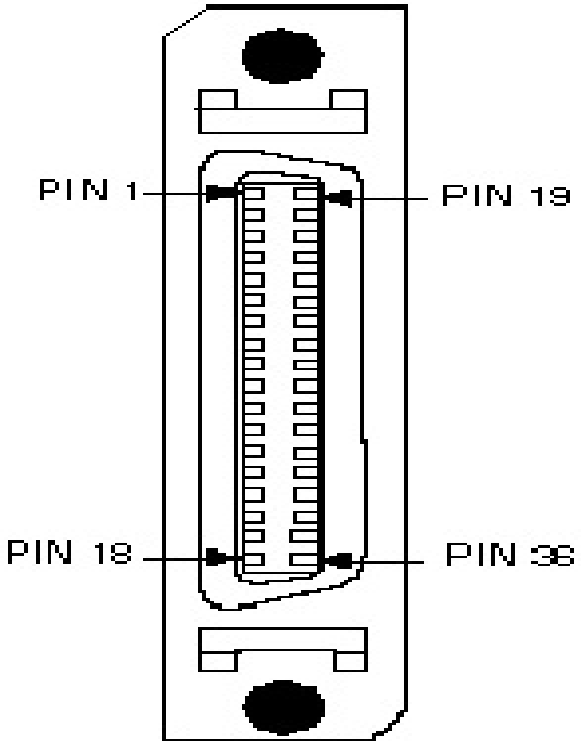

Figura 6: *Numeración de los pernos para el conector 1284-C.*

La funcionalidad de los pernos por cada modo de transferencia, se describe en la siguiente tabla:

Número de perno.	**Modo de transferencia.**				
	De compatibilidad	**Nibble**	**Byte (octeto)**	**EPP**	**ECP**
1	Busy (ocupado)	PrtBusy (Puerto ocupado)	PrtBusy (Puerto ocupado)	nWait (espera)	PerifAck (reconocimien-to)
2	Select (seleccionado)	Xflag	Xflag	Definido por el usuario 3.	Xflag
3	nAck (reconocimiento)	PtrClk (puerto del reloj).	PrtClk (puerto del reloj)	Int (interrupción)	PeriphClk (reloj del periférico)
4	nFault (falla)	nDataAvail (datos disponibles)	nDataAvail (datos disponibles)	Definido por el usuario 2.	nPeriphRequest (solitud del periférico)
5	PError (error de la impresora)	AckDataReq (reconocimiento de solicitud de datos).	AckDataReq (reconocimiento de solicitud de datos).	Definido por el usuario 1.	nAckReverse (solitud inversa de datos).
6	D1	D1	D1	AD1	D1

7	D2	D2	D2	AD2	D2
8	D3	D3	D3	AD3	D3
9	D4	D4	D4	AD4	D4
10	D5	D5	D5	AD5	D5
11	D6	D6	D6	AD6	D6
12	D7	D7	D7	AD7	D7
13	D8	D8	D8	AD8	D8
14	nInit (inicializar)	nInit (inicializar)	nInit (inicializar)	nInit (inicializar)	nReverseRequest ()
15	nStrobe (parpadeo)	HostClk (reloj del equipo anfitrión)	HostClk (reloj del equipo anfitrión)	nWrite (escribir)	HostClk (reloj del equipo anfitrión)
16	nSelectIn	1284 active (activo)	1284 active (activo)	nDStrb	1284 active (activo)
17	nAutoFd (autoalimentación)	HostBusy (equipo anfitrión ocupado)	HostBusy (equipo anfitrión ocupado)	nDStrb	HostAck (reconocimiento del equipo anfitrión)
18	Host Logic High (voltaje lógico alto del equipo anfitrión)	----------	----------	----------	----------
19	Tiera para la señal *Busy* (ocupado)	----------	----------	----------	----------
20	Tierra para la señal *Select* (periférico escogido).	----------	----------	----------	----------
21	Tierra para *nAck* (reconocimiento)	----------	----------	----------	----------
22	Tierra para *nFault* (fallo).	----------	----------	----------	----------
23	Tierra para PError (error del periférico).	----------	----------	----------	----------
24	Tierra para D1	----------	----------	----------	----------
25	Tierra para D2.	----------	----------	----------	----------
26	Tierra para D3	----------	----------	----------	----------
27	Tierra para D4	----------	----------	----------	----------

28	Tierra para D5	----------	----------	----------	----------
29	Tierra para D6	----------	----------	----------	----------
30	Tierra para D7	----------	----------	----------	----------
31	Tierra para D8	----------	----------	----------	----------
32	Tierra para *nInit* (inicializar)	----------	----------	----------	----------
33	Tierra para *nStrobe* (parpadeo)	----------	----------	----------	----------
34	Tierra para *nSelectIn* ().	----------	----------	----------	----------
35	Tierra para *nAutoFd* (autoalimentación).	----------	----------	----------	----------
36	Peripheral Logic High (alto lógico del periférico).	----------	----------	----------	----------

Tabla 4: Funcionalidad de los pernos del conector Mini-centronics (tipo C).

Los nombres de los pernos que comienzan con el prefijo 'n', indican que tienen un valor lógico inverso. Los bits de datos van del perno seis al trece (D1 a D8), del bit menos significativo al más significatio, para forma un byte de información.

1.2.4.3 Interfaz eléctrica del estándar *IEEE 1284*.

Antes de que se definiera este estándar, existían diversas configuraciones electrónicas que tenían diferentes parámetros electrónicos de voltajes, capacitancia e impedancia en las líneas de sus conexiones, por lo que no se aseguraba que al conectarse dos equipos, se comunicaran correctamente.

El estándar *IEEE 1284* define los niveles I y II para la interfaz eléctrica. El nivel I fue definido para equipos que no operan a altas tasas de transferencia, pero que necesita usar el modo inverso (recepción de datos); por el contrario, el nivel II se define para equipos que sí operan con altas tasas de transferencia, con cables largos y usan los modos más avanzados de comunicación, por lo que requiere de especificaciones más rigurosas.

Los parámetros eléctricos que debe cumplir el controlador, para el nivel II, son los siguientes:

1. El voltaje de salida en nivel alto, cuando el circuito esté abierto, no deberá exceder de +5.5 [V].
2. El voltaje de salida en nivel bajo, cuando el circuito esté abierto, no deberá ser menor de -0.5 [V].
3. El voltaje de salida en nivel alto, en circuito cerrado, deberá ser de al menos 2.4 [V] con una corriente de 14 [mA].
4. El voltaje de salida en nivel bajo, en circuito cerrado, no deberá exceder de 0.4 [V] con una corriente de 14 [mA].

5. La impedancia de salida del controlador (Ro), medida en el conector, deberá ser de 50 ± 5 [Ω] cuando el voltaje de salida sea de

$$\frac{V_{oh} - V_{ol}}{2}$$

Ecuación 1: *Voltaje de salida del controlador.*

6. La tasa de caída de voltaje deberá ser de 0.05-0.40 [V/nS].

Los requerimientos para el receptor del nivel II son los siguientes.

1. El receptor deberá soportar picos de voltaje transitorios de entre -2.0 [V] y 7.0 [V], sin que se dañe u opere incorrectamente.
2. El umbral del voltaje para pasar al nivel alto no deberá ser mayor de 2.0 [V].
3. El umbral del voltaje para el nivel bajo deberá ser de al menos 0.8 [V].
4. La curva de histéresis deberá ser de entre 0.2 [V] y 1.2 [V].
5. La caída de voltaje en el nivel alto no deberá ser mayor de 20 [μA] a los 2.0 [V].
6. La corriente de la fuente del receptor en el nivel bajo de voltaje no deberá ser mayor de 20 [μA] a los 0.8 [V].

7. La capacitancia parásita y del circuito no deberá ser mayor de 50 [pF].

A continuación se muestra la configuración electrónica recomendada para las conexiones entre el emisor y el receptor del estándar *IEEE 1284*.

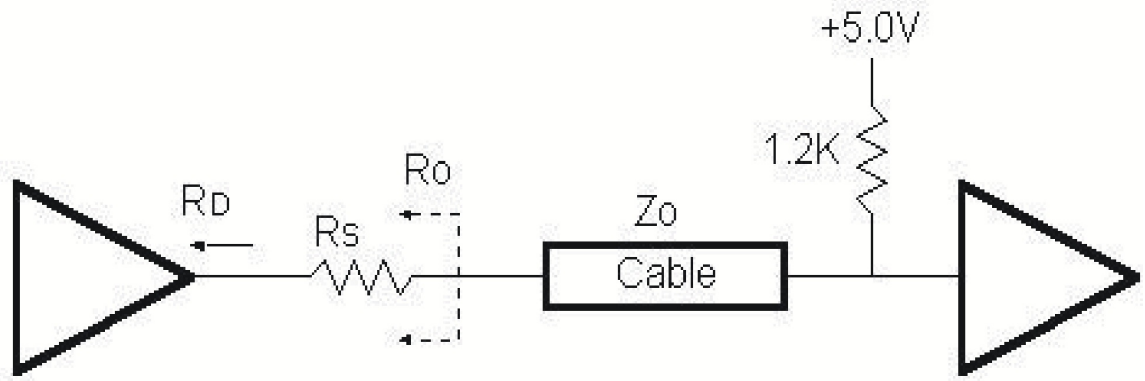

Figura 7: *Conexión emisor/receptor recomendada para evitar diferencia de impedancias.*

La figura 7 muestra la conexión recomendada para el par de conexión emisor/receptor del nivel II, donde R_o es la impedancia de salida del emisor; se recomienda que dicha impedancia sea igual que la del cable, para evitar el ruido causado por la diferencia de impedancias. Dependiendo del tipo de controlador usado, se puede usar una resistencia en serie para obtener la impedancia correcta.

Para un par emisor receptor del nivel II, tal como una línea de datos, se recomienda la configuración del circuito de la figura 8.

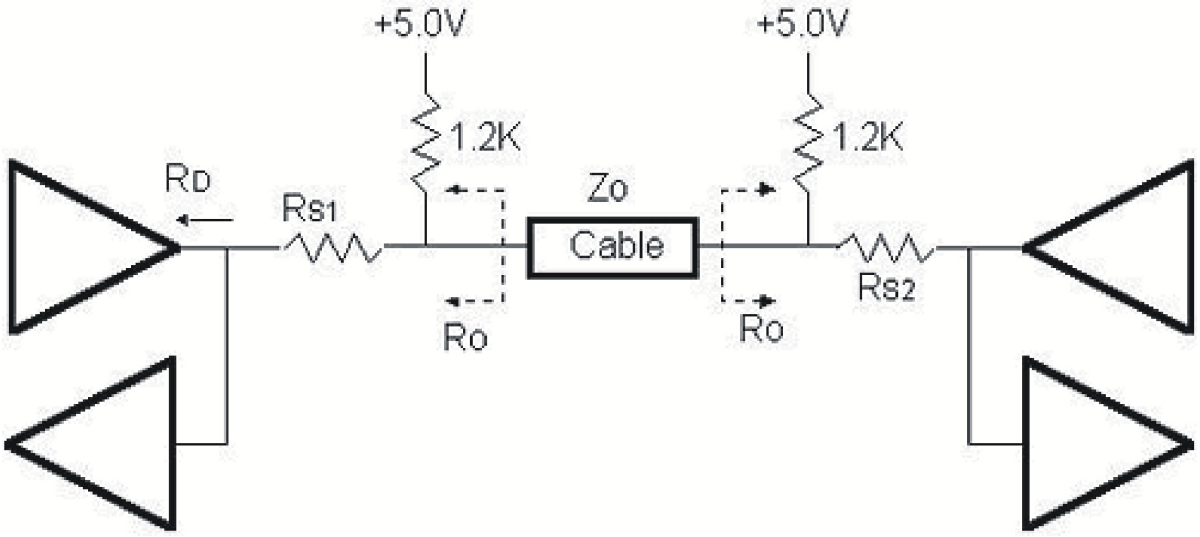

Figura 8: *Circuito recomendado para el par emisor/receptor del nivel II.*

De la especificación eléctrica anterior se concluye que se usa niveles de voltaje que corresponden a la lógica TTL; los fabricantes usan circuitos integrados 74LS374 para las líneas de datos, mientras para las de control usan los 7405.

1.2.5 La arquitectura del puerto paralelo en la computadora personal.

El puerto paralelo, como cualquier otro periférico, usa diversos recursos para comunicarse con el *CPU* (*Central Process Unit*); dentro de esos recursos se encuentra la asignación de un intervalo de direcciones de memoria, las cuales pueden variar. A muchos puertos se le asigna un nivel IRQ (*Interrrupt ReQuest*), y para el caso del modo *ECP*, se le asigna un canal *DMA* (*Direct Memory Access*, Acceso Directo a Memoria). Finalmente, se debe evitar conflictos en la asignación de recursos con otros puertos paralelos u otros periféricos.

El puerto paralelo usa tres direcciones de memoria contiguas, las cuales son la dirección base (también conocida como la dirección registradora de datos o solamente como dirección del puerto), la del registro del puerto y el registro del Control; generalmente se encuentran en alguna de los siguientes intervalos (en números hexadecimales):

3BCh, 3BDh, 3BEh
378h, 379h, 37Ah
278h, 279h, 27Ah

En la implantación de la arquitectura de los modos *EPP* y *ECP* se reservan direcciones adicionales de memoria para cada puerto. En el caso de un puerto *EPP* se añaden cinco registros más a partir de la dirección base más tres, para llegar a la dirección base más siete. Para el caso de un puerto tipo *ECP* añade tres direcciones más, las cuales van de la dirección base + 400h a la dirección base + 402h.

En las primeras computadoras personales, la dirección del puerto paralelo era la 3BCh, pero posteriormente cambió a la 378h en diseños más recientes. Dependiendo de los componentes físicos (*hardware*), dichas direcciones pueden cambiar.

Para facilitar la interacción con el usuario, los sistemas operativos *DOS* y *Windows* (diseñados para

computadoras personales), asignan los nombres **LPT1** para el primer puerto paralelo, **LPT2** para el segundo, y así sucesivamente; los nombres **LPT*n*** se refieren a *Line Printer*, ya que este puerto se conecta generalmente a impresoras. Cuando una computadora personal se enciende, el sistema operativo asigna cada puerto paralelo a una dirección base de memoria.

Dependiendo del equipo de cómputo, se puede cambiar el puerto paralelo a otra dirección, siempre y cuando el sistema operativo lo permita también.

1.3 La familia de Protocolos TCP/IP.

La familia de Protocolos de Control de Transmisión/Protocolo de Red (TCP/IP) tiene cinco niveles:

- Físico.
- Enlace de datos.
- Red.
- Transporte.
- Aplicación.

TCP/IP fue desarrollado antes de ISO OSI para crear una red de redes, por lo que los niveles de ambos modelos se corresponden exactamente entre sí. Los primeros cuatro niveles de TCP/IP proporcionan los estándares físicos, interfaz de red, conexión entre redes y funciones de transporte que corresponden con los cuatro primeros niveles del modelo ISO OSI, pero los tres últimos niveles del modelo OSI equivalen al nivel de aplicación de TCP/IP.

TCP/IP es un protocolo jerárquico compuesto por módulos interactivos; cada módulo tiene una funcionalidad específica, pero no son obligatoriamente inter dependientes entre sí. TCP/IP consiste en un conjunto de protocolos relativamente independientes, los cuales se pueden mezclar y hacer coincidir, dependiendo del sistema; la jerarquía de TCP/IP señala que cada protocolo del nivel superior está soportado por uno o más protocolos del nivel inferior.

Para el nivel de transporte, TCP/IP define los siguientes dos protocolos:

- Protocolo de Control de Transmisión (TCP).
- Protocolo de Datagramas de Usuario (UDP).

En el nivel de red, el principal protocolo es el Protocolo entre Redes (IP).

1.3.1 Protocolo IP.

IP es el mecanismo de transmisión de TCP/IP, con base en datagramas (explicados más adelante) sin conexión y no fiables, el cual ofrece el servicio de mejor entrega, el cual consiste en enviar los mensajes sin comprobar si llegaron a su destino, ni darles seguimiento. Cada datagrama viaja de forma independiente, por lo que algunos pueden perderse o llegar duplicados; al ser enviados sin conexión, no se crean circuitos virtuales por lo que no se asegura que lleguen a su destino, ni se avisa al destinatario de la llegada de los datos.

Los datagramas son paquete de datos del protocolo IP; cada paquete tiene una longitud variable, máxima de 65,536 bytes, formado por una cabecera y datos. La cabecera tiene de 20 a 60 bytes de información para encaminar y entregar los datos; la cabecera se divide en secciones de cuatro bytes. Cada campo de la cabecera se describe a

continuación:

- Versión de IP, la versión actual del protocolo es la cuatro (IPv4), que se indica en binario (0100).
- Longitud de cabecera, tiene cuatro bytes para indicar el tamaño del encabezado, por lo que el valor máximo que puede almacenar es 60.
- Tipo de servicio, con el que indica cómo debe manejar el datagrama; incluye bits para la prioridad, y bits para especificar el tipo de servicio que requiere el emisor, así como el nivel de prestaciones, fiabilidad y retardo.
- Longitud total, es un campo de dos bytes donde indica la longitud total del datagrama.
- Identificación, contiene un número secuencial, con el que se puede reconstruir el datagrama cuando se fragmenta para que se ajuste al tamaño de la trama de una red.
- Indicadores, que indican si el datagrama fue fragmentado o no, y si lo fue, indicar el número de fragmento por medio de un número secuencial.
- Desplazamiento del fragmento, el cual es un apuntador que indica el desplazamiento de los datos en el datagrama, si llegara a fragmentarse.
- Tiempo de vida, el cual es un número que indica cuántas veces salta en red, disminuyendo dicho valor en uno con cada salto; al llegar a cero se descarta el datagrama, para evitar que siga viajando en la red indefinidamente, o que regrese a su origen, evitando que hubiera un tráfico muy pesado en red.
- Protocolo, que indica cuál protocolo de nivel superior (TCP, UDP, ICMP, etcétera) se encuentra encapsulado en el datagrama.
- Suma de comprobación de cabecera, el cual es un campo de 16 bits que sirve para calcular la integridad de la cabecera del paquete.
- Dirección de origen, donde se almacena la dirección Internet de donde salió el paquete; utiliza 4 bytes para representarla.
- Dirección destino, que usa cuatro bytes para guardar la dirección Internet destino.
- Opciones, son campos para indicar funciones adicionales de IP, usados para controlar el encaminamiento, la temporalización, la gestión y el alineamiento.

El protocolo IP se forma, a su vez, por los siguientes protocolos:

- ARP
- RARP
- ICMP
- IGMP

1.3.2 UDP.

Es el protocolo más simple de los protocolos estándar del nivel de transporte; es de extremo a extremo, añadiendo direcciones de puertos, control de errores mediante sumas de comprobación y la longitud de datos del nivel superior. El datagrama de este protocolo se llama datagrama de usuario, cuyos campos son los siguientes:

- Dirección de puerto origen, la cual es la dirección del programa que envía este mensaje.
- Dirección de puerto destino, la cual es la dirección de la aplicación que recibirá este mensaje.
- Longitud total, donde indica la longitud total del datagrama del usuario en bytes.
- Suma de comprobación, el cual es un campo de 16 bits, usado para detectar errores.

1.3.3. TCP.

Este protocolo proporciona servicios completos de transporte a las aplicaciones, de puerto a puerto, por lo que brinda un flujo confiable. Es un protocolo orientado a la conexión, por lo que primero establece una conexión entre ambos extremos, creando un circuito virtual entre emisor y receptor, y posteriormente envía los datos; dicho circuito virtual permanece activo durante la transmisión. TCP establece la conexión, y luego indica que hay datagramas en camino, y para terminar la conexión indicando un fin de transmisión.

UDP e IP consideran a los datagramas como paquetes de datos individuales, pero TCP es un servicio orientado a conexión, con lo que se asegura de la entrega fiable de la información; dicha fiabilidad consiste en que detecte errores y retransmita tramas recibidas con errores. Todos los segmentos del datagrama deben ser recibidos y confirmados antes de que la transmisión se considere completa y se pueda cerrar el circuito virtual.

Si la transmisión es larga, TCP la divide y empaqueta (en el lado del emisor) en tramas de datos más pequeñas, llamadas segmentos. Los segmentos incluyen un número de secuencia, usado para reordenar y reconstruir el mensaje original en el receptor; también incluye un número identificador de confirmación, y un campo que indica el tamaño de la ventana deslizante usada en las confirmaciones. Cada segmento se transporta como un datagrama con el protocolo IP, y al ser recibido, TCP recibe y reordena los datagramas mediante su número de secuencia.

TCP es más lento pero más confiable que UPD; el primer protocolo requiere establecer la conexión y efectuar confirmaciones; para que tuviera la fiabilidad requerida, TCP requiere más campos en el segmento TCP, los cuales son descritos a continuación:

- Dirección de puerto origen, donde indica la dirección de la aplicación del emisor.
- Dirección de puerto destino, donde se anota la dirección de la aplicación de la computadora destino.
- Número de secuencia,donde registra un número secuencial cuando divide el mensaje original en segmentos.
- Número de confirmación, el cual tiene 32 bits para verificar si se recibió la transmisión. Funciona si el bit ACK (reconocimiento) está activo.
- Longitud de la cabecera (LC), el cual consiste en cuatro bits que indica el número de bytes del tamaño de la cabecera.
- Reservado, campo apartado para uso futuro.
- Control, son seis bits que funcionan de forma individual e independientemente; el bit de urgente activa el uso del campo de apuntador urgente. El bit ACK (reconocimiento) activa al campo de Número de Confirmación. El bit PSH indica que se requiere un mayor ancho de banda. El bit RST solicita reiniciar la conexión cuando hay confusión en los números de secuencia de los datagramas recibidos. El bit SYN indica que se sincronice los números de secuencia en tres tipos de segmentos (petición de conexión, confirmación de conexión, y recepción de confirmación). El bit FIN se usa para termina la conexión en los tres tipos de segmentos (petición de confirmación, confirmación de terminación, y confirmación de la confirmación de terminación).
- Tamaño de la ventana, el cual es un campo de 16 bits para indicar el tamaño de la ventana deslizante.
- Suma de comprobación, que es un campo de 16 bits usado para la detección de errores.
- Apuntador urgente, el cual es el último campo requerido dentro del encabezado del segmento, e indica que hay datos urgentes a transmitir en la porción de datos del segmento.
- Opciones y rellenos, que son campos opcionales para incluir información adicional o para alinear el segmento.

1.3.4 Protocolo de transferencia de hipertexto (HTTP).

Existen diversos protocolos dentro de Internet, los cuales sirven para enviar archivos, intercambiar mensajes, gestionar la red, etcétera; sin embargo, se explicará sólo el protocolo HTTP (que pertenece al nivel de aplicación), para mostrar cómo se solicitan y se reciben las páginas de Internet.

El protocolo de transferencia de hipertexto (*Hiper Text Transfer Protocol*, *HTTP*) sirve para enviar

información en forma de texto plano, hipertexto, sonido, video, etcétera, a través de Internet. Se denomina de hipertexto, porque en los documentos existen elementos llamados ligas, las cuales permiten el salto rápido de un documento a otro.

Este protocolo se parece a FTP y a SMPT; en cuanto a FTP, se parece porque también transfiere archivos y usa servicios de TCP, pero sólo usa una conexión para enviar datos, y no dos como FTP; carece de una conexión de control.

Respecto a SMTP, se le asemeja porque los mensajes entre servidor y cliente se parecen a mensajes SMTP, además de que los mensajes con controlados por las cabeceras de los mismos, pero difieren en cuanto a que existe una conexión directa entre cliente y servidor, los mensajes se envían de forma directa, y en que todos los mensajes son leídos e interpretados por el cliente (el navegador).

Las solicitudes del cliente al servidor son enviadas como si fueran de correo electrónico; el servidor da una respuesta, que se parece a una de correo electrónico. Los mensajes, junto con los datos, se intercambian en un formato similar al MIME de correo electrónico.

Las órdenes del cliente al servidor se insertan en un mensaje de petición, mientras que el contenido del archivo solicitado (o los datos solicitados) se envía en un mensaje de respuesta.

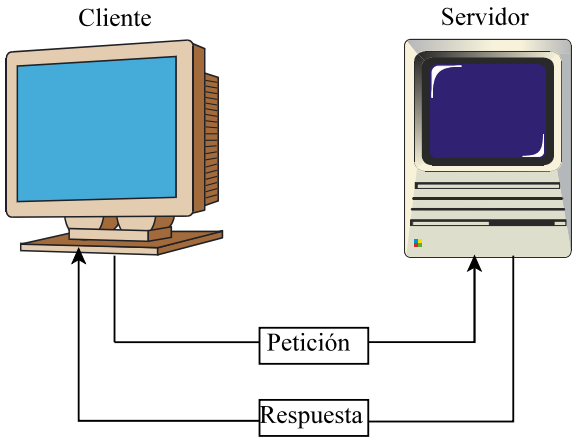

Figura 9: *Transacción HTTP.*

Una transacción HTTP consiste en una petición de parte del cliente y en una respuesta del servidor a la misma.

Existen sólo dos tipos de mensajes HTTP, el de petición y el de respuesta; ambos mensajes tienen el mismo formato.

La estructura del mensaje de petición consiste en una línea de petición, varias cabeceras, y a veces contiene un cuerpo.

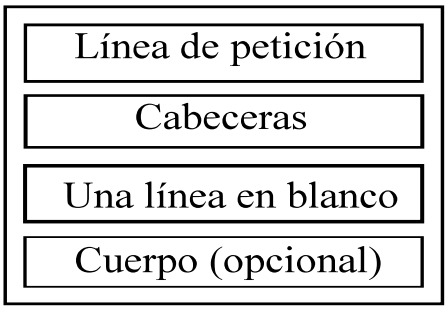

Figura 10: *Estructura del mensaje de petición HTTP.*

El mensaje de respuesta se conforma por medio de una línea de estado, varias cabeceras, y en algunas ocasiones contienen un cuerpo.

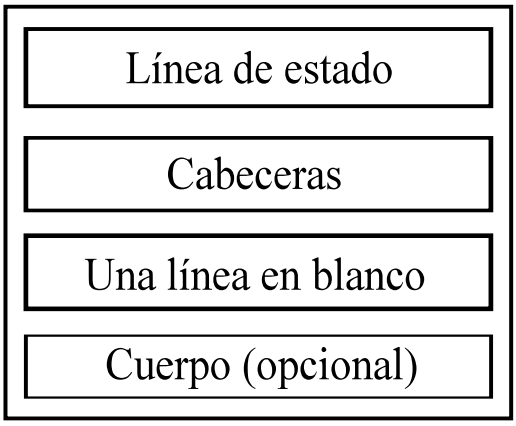

Figura 11: *Estructura del mensaje de respuesta HTTP.*

1.4 Adquisición de imágenes digitales.

1.4.1 La fotografía digital en la actualidad.

Las cámaras de video de estado sólido (cámaras digitales) se han vuelto los dispositivos de grabación de video más populares de los últimos años, desplazando a las cámaras analógicas (cámaras de tubo), tanto por su precio como por su resolución de imagen.

El uso de dichas cámaras digitales ha cobrado un gran auge, debido a que son más sencillas de manejar, están provistas de mayor funcionalidad que las cámaras analógicas, y porque las imágenes se pueden transmitir a otros dispositivos fácilmente o exportar a formatos digitales ampliamente difundidos.

Las cámaras tradicionales, con un rollo de película, capturan la imagen cuando la luz que refleja el objetivo entra por el obturador; dependiendo de la cantidad de luz y de los diversos espectros de la misma, se provoca una reacción química en los componentes fotosensibles de la película, la cual es un medio analógico donde queda registrada la imagen en negativo. El fotógrafo debe enfocar al objetivo con cuidado (para obtener una imagen adecuada), y evitar la exposición indebida del rollo a la luz. Finalmente, el rollo de película debe revelarse en un cuarto obscuro -el cual tiene una iluminación muy baja y con otro tipo de luz, para evitar otra reacción química en el rollo-, utilizando una solución química para obtener las imágenes en papel (en positivo).

Una cámara de video tradicional funciona de forma similar, tomando varias imágenes por segundo, y guardando las imágenes en la cinta de video (que es un medio analógico); no requiere un proceso químico posterior, pero la calidad de las imágenes es menor que las imágenes fotográficas. Posteriormente, al reproducirse la secuencia de imágenes, se da la sensación de continuidad. En ambos casos, si se requiere transmitir, procesar (para eliminar ruido, por ejemplo) o cambiar a otro formato, se necesitan conversores (digitales o analógicos), algunos de los cuales pueden ser costosos, por lo que no todos los usuarios tienen acceso a ellos.

Por otra parte, las cámaras fotográficas y de video más reciente capturan las imágenes de forma digital, y las guardan en formatos digitales, por lo que posteriormente se pueden transmitir, exportar a otros formatos, imprimirlas, y hacer procesamiento digital de imágenes sobre las mismas; incluso, durante el proceso de captura de imágenes, efectúan procesamiento digital de imágenes, para enfocar automáticamente al objetivo, eliminar ruido, ajustar brillo y contraste, etcétera, por lo que los usuarios requieren de menos habilidad para obtener una imagen de mayor calidad que usando una cámara analógica. Una vez tomada la imagen o el video, se puede apreciar los mismos sin necesidad de un proceso adicional.

La fotografía e impresión digitales han llegado a un punto tan avanzado que varias compañías dedicadas a la fotografía han abandonado la fotografía convencional (analógica) en favor de la digital, e incluso la NASA ha hecho lo mismo, ya que el uso de la fotografía digital en satélites y sondas espaciales es más confiable que la analógica.

1.4.2 Cámaras de acoplamiento de carga.

Las cámaras digitales son dispositivos electrónicos de estado sólido, entre las cuales se encuentran las cámaras de acoplamiento de carga *CCD* (*charged-coupled device*); estas últimas son las más populares y baratas de su tipo. Fueron inventadas por los Laboratorios Bell en 1969, siendo un medio muy confiable para adquirir imágenes, por lo que se han utilizado en el programa espacial de la NASA, y en los últimos años han reemplazado a las cámaras de rollo de película usadas por consumidores finales y profesionales.

Las cámaras de acoplamiento de carga se construyen al formar un arreglo de diodos con un proceso fotolitográfico, el cual tiene un patrón regular perfecto que evita distorsiones de la imagen. La luz que entra en cada sensor (diodo) provoca que en el semiconductor de mismo se desprendan electrones, los cuales entran en la banda de conducción; el número de electrones es proporcional a la intensidad de la luz. Como es imposible alambrar cada sensor para detectar su lectura de forma individual, lo que se hace es que en un ciclo de reloj, se recoge la lectura de varias columnas de sensores en un renglón de sensores; en el siguiente ciclo de reloj, los sensores del renglón pasan su lectura (electrones) a una "cubeta", la cual representa un *pixel*. La lectura acumulada se envía a un amplificador, y de ahí se cuantifica inmediatamente, para dar una salida numérica para una cámara digital, o se envía como señal analógica a una cámara de video.

La rotación de los electrones dentro del semiconductor se hace utilizando tres electrodos conectados al semiconductor, los cuales constituyen un *pixel*; a dos de ellos se aplica un voltaje para formar un campo que funciona como recipiente para los electrones; estos últimos quedan atrapados en la región central del semiconductor por el mayor de los campos de cada electrodo. Cambiando el voltaje a los electrodos en seis pasos se logra la rotación de los electrones. Posteriormente se repite el proceso, el cual se asemeja a una cinta de transporte que lleva "cubetas", de ahí su nombre.

Estos dispositivos son poco sensibles a la luz a la luz azul clara, y muy sensibles a la luz infrarroja, por lo que se utilizan filtros y antireflejantes, con fin de lograr la respuesta deseada.

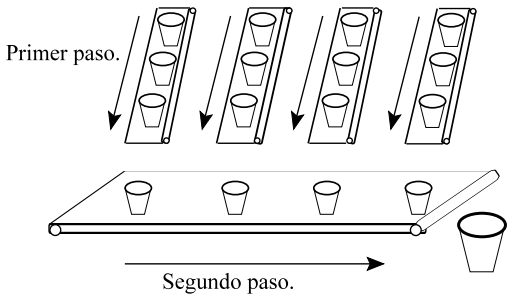

Figura 12: *Principio básico de la operación de un dispositivo CCD.*

1.4.3 Conceptos de imagen, video y color digitales.

1.4.3.1 Imagen.

Es una representación gráfica de un objeto o persona, en un momento de tiempo dado; contiene información que se aprecia a través del sentido de la vista.

La óptica define a una imagen como el resultado de la reflexión de la luz sobre un objeto; el objeto observado es iluminado por luz solar o luz blanca; parte de la radiación luminosa es retenida por el objeto, y lo que refracta es lo que el observador percibe con la vista.

El espectro visible se define como las distintas frecuencias que componen a la luz blanca. Al pasar por un prisma, la luz blanca se descompone en los colores violeta, índigo (añil), azul, verde, amarillo, magenta y rojo, los cuales son mencionados de menor a mayor longitud de onda. Dichos colores se clasifican en colores primarios y secundarios; los primeros, en cualquier combinación, crean a los demás colores. Los colores secundarios son creados con dos de los colores primarios. Los colores secundarios son creados con dos de los colores primarios.

El color del objeto se forma con la combinación de los espectros de luz visibles refractados; un objeto negro absorbe todas las longitudes de onda del espectro visible, mientras que un objeto blanco refracta toda la luz que recibe; un objeto azul refracta las ondas de luz cuyas longitudes de onda sean simulares a las del color azul. Los colores primarios, utilizando fuentes luminosas, son el azul, verde y rojo; tratándose de pigmentos (que refractan luz), los colores primarios son el amarillo, índigo (añil) y magenta.

La apreciación de las imágenes es un fenómeno psicofísico, ya que por una parte depende de las leyes de la física, y por otra parte depende de cómo trabaja el ojo humano y de cómo las interpreta el cerebro humano, debido a que el ojo humano sólo capta una parte del espectro de luz (espectro visible), y a que cada persona tiene mayor o menor agudeza visual, por lo que dicha apreciación puede ser subjetiva.

Para evitar interpretaciones subjetivas, las matemáticas definen a la imagen monocromática como una función bidimensional de la intensidad de luz $f(x, y)$, donde x y y son las coordenadas espaciales, y f, en un punto (x, y), es el valor proporcional al brillo (o nivel de gris) de la imagen en el punto dado.

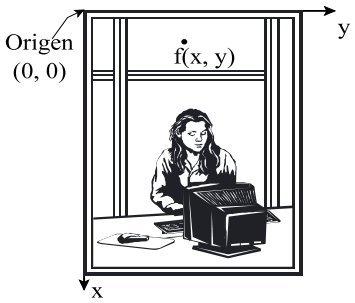

Figura 13: *Convención de los ejes para la representación digital de imágenes.*

De forma similar, una imagen policromática se puede definir como una función *f(x, y)*, donde *x* y *y* son las coordenadas espaciales, y *f* es el valor resultante de la combinación de los colores primarios; aquí se debe tomar en cuenta el espectro visible para el ojo humano y un modelo de color (de los que se hablará más adelante).

La importancia de las imágenes para los humanos es muy alta, debido a que nosotros percibimos muchos estímulos externos por el sentido de la vista; brinda una apreciación cuantitativa que puede ser interpretada fácil y rápidamente.

1.4.3.2 Video.

El video es una secuencia de imágenes, ordenadas en forma cronológica. Da la sensación de movimiento del objetivo de las imágenes, tal y como los humanos apreciamos las cosas con el sentido de la vista.

1.4.3.3 Imagen digital.

Imagen digital monocromática es una imagen *f(x,y)* discretizada tanto en las coordenadas espaciales como en el brillo; dicha imagen se representa como una matriz, donde los índices de renglones y columnas representan un punto *(x, y)*, y el valor registrado en la matriz en ese renglón y columna, indica el brillo en el punto señalado.

Cada elemento de la matriz antes mencionada se llama elemento de la imagen, o más comúnmente, *pixel* o *pels* (abreviación en inglés de *picture element*).

En esta tesis sólo se consideran imágenes en dos dimensiones; no se contempla a las imágenes de profundidad (3D, tres dimensiones). Para efectos prácticos, el alto y ancho de la imagen digital se determinan con potencias de dos.

1.4.3.4 Imagen digital policromática.

De forma similar a la definición anterior, la imagen digital policromática es una imagen f(x, y) discretizada, tanto en coordenadas espaciales como en el resultado de combinación de colores primarios. Hay que recordar que el concepto de color se basa en cómo el ojo humano percibe a las distintas frecuencias del espectro visible, y del modelo de color con que se obtienen todos los colores a partir de los colores primarios.

1.4.3.5 Color.

El color para los seres humanos es una parte muy importante del sentido de la vista, y tiene grandes implicaciones psicológicas. El color puede revelar si una fruta está madura, indicar peligro, o dar la sensación de

tranquilidad. En el análisis de diversos fenómenos se usa lo que llaman colores falsos, que es asignación de colores arbitrarios a ciertos grises, con el fin de distinguir a simple vista las partes de la imagen, y mejorar la comprensión de dicho fenómeno.

Como ya se había explicado anteriormente al principio de este capítulo, los humanos vemos sólo ciertos colores, de lo que se llama el espectro visible; dentro de ellos existen tres colores primarios, los cuales forman a los demás colores en diversas combinaciones.

En 1931, la CIE (*Commission Internationale de l'Eclairage*) definió a los colores primarios con las siguientes longitudes de onda:

- Azul=435.8 [nm]
- Verde=546.1 [nm]
- Rojo=700 [nm]

Cada color se distingue de los demás por las siguientes características:

- Brillo, que indica la intensidad del color.
- Matiz, que se relaciona con la longitud de onda predominante en la combinación que forma este color; la gente, al nombrar un color tal como el verde, rojo, naranja, etcétera, indica el matiz del mismo.
- Saturación, el cual indica la pureza relativa, o la cantidad del matiz mezclada con luz blanca; el grado de saturación es inversamente proporcional a la luz blanca con que se mezcló.

La cromaticidad es la unión de los conceptos de matiz y saturación, por lo que cada color se describe por su cromaticidad y el brillo. Las cantidades de rojo, verde y azul requeridas para crear un color se llaman colores triestímulo, denominados X, Y y Z, respectivamente. Cada color se especifica por sus coeficientes tricromáticos, los cuales son los siguientes:

$$x = \frac{X}{X+Y+Z}$$
$$y = \frac{Y}{X+Y+Z}$$
$$z = \frac{Z}{X+Y+Z}$$

donde

$$x + y + z = 1$$

Ecuación 2: *Fórmulas de los coeficientes tricromáticos.*

Los valores triestímulo de cualquier longitud de onda del espectro visible se pueden obtener a partir de tablas cuyos valores se obtuvieron a partir de valores experimentales; también se pueden calcular a través del diagrama de cromaticidad, el cual modela la composición de colores como una función *f(x, y)*, donde *x* es el rojo, *y* es el verde, mientras que *z* se calcula con la ecuación mostrada anteriormente.

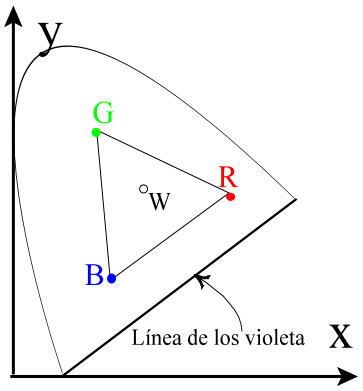

Figura 14: *Triángulo de los colores primarios.*

En la figura mostrada anteriormente, se puede apreciar el triángulo formado por los colores primarios; al ir avanzando adentro de dicho triángulo, disminuye la saturación de dichos colores, llegando a W, el cual representa la luz blanca. El área comprendida dentro del segmento de curva y la "línea de los violeta", contiene todos los colores del espectro visibles para el ser humano.

Los modelos de color sirven para representar de forma estándar a cada combinación de los mismos; el modelo a usar depende de la aplicación que requiera mostrar colores. A continuación se mencionan modelos de color y la aplicación que los usa:

- *RGB* (*red*, *green*, *blue*; rojo, verde y azul), el cual se usa para videocámaras y monitores a color.
- *CMY* (*cyan*, *magent*, *yellow*; cyan, magenta y amarillo), usado para las impresoras a color.
- *YIQ* (reflectancia o intensidad, infase y cuadratura cromáticas), empleado en para los televisores a color.
- *HSV* (matriz, saturación, valor).
- *HSI* (matriz, saturación e intensidad).

1.4.3.6 Formatos de imágenes y video.

Existen diversos formatos de imágenes fijas y de video, los cuales tienen diversas ventajas y desventajas tecnológicas; algunos fueron desarrollados por compañías (que tiene patente sobre sus formatos y cobran regalías por su uso comercial), otros fueron diseñados por comités, y algunos más, fueron definidos por partidarios de los programas de cómputo libres (en oposición a los formatos comerciales creados por compañías).

1.4.3.6.1 Formatos de imágenes.

Los formatos de imágenes aparecieron cuando surgió la necesidad de representar las imágenes de forma digital, para su almacenamiento, transmisión y/o aplicar algoritmos de procesamiento digital de imágenes.

Los primeros formatos que surgieron describen de forma extensiva la imagen, *pixel* por *pixel* (en un mapa de bits), ya sea en blanco y negro, en escala de grises o a color. Posteriormente surgieron algoritmos de compresión, los cuales se basan en que mucha información se repite (debido a que muchas veces un *pixel*, y sus vecinos son similares). Más adelante surgieron los formatos vectoriales, que describen cómo se forma la imagen, con base en la descripción de objetos geométricos que la conforman (líneas y puntos donde intersecan). Los formatos de imágenes más conocidos se describen a continuación.

- *Windows Bitmap* (*BMP*), se diseño para la plataforma *Windows*; tiene una cabecera que indica el ancho y el alto de la imagen. Cada *pixel* se representa por 24 bits, ocho por cada color (azul, rojo y verde), acomodados de arriba hacia abajo, y la información del color del *pixel* aparece en el orden Azul-Verde-Rojo También acepta un formato para escala de grises, usando 8 bits por *pixel*. Acepta compresión, pero generalmente no se usa.
- *Tag Interchange File Format* (*TIFF*), el cual se basa en etiquetas (*tags*); una cabecera describe la imagen, indicando número de líneas, columnas, número de bits por *pixel* y el tipo de compresión (*LWZ* sin pérdidas, o compresión con pérdidas *JPEG*. Cada etiqueta tiene un valor numérico asignado, y aquellas con valor igual o mayor a 32768 se denominan privadas, las cuales se usan para guardar información del usuario; algunas aplicaciones no pueden interpretar etiquetas privadas, y mal interpretan el formato, mientras que otras las detectan y avisan de su existencia, pero las ignoran. Un archivo de este tipo puede tener más de una imagen.
- *Graphic Interchange Format* (*GIF*), que fue creado originalmente por *Compuserve* para sus usuarios. Cada archivo *gif* cuenta con una paleta de 256 colores (ocho bits), y cada *pixel* se representa por tres valores *RGB*; en caso de que cambie la paleta de colores, cambian los colores de imagen. Puede utilizar compresión *LWZ*, y existe una versión en que permite ver una secuencia de imágenes, llamado *gif* animado. No es muy adecuado para representar imágenes fotográficas, porque el ojo humano es más sensible al contraste, por lo que se notaría la carencia de calidad de la imagen respecto a otros formatos que almacenan más colores; sin embargo, es un formato muy adecuado para las imágenes vectoriales, y el tamaño de los archivos de este tipo de imágenes es relativamente pequeño.
- *Portable Network File* (*PNG*), el cual tiene versiones de 8 y 24 bits, siendo diseñado para estar exento de regalías y reemplazar al formato *GIF*. *PNG-8* utiliza ocho bits para representar cada color de un *pixel*, usa compresión sin pérdidas *ZIP*, pero incorpora canales alfa (transparencia variable), corrección gama (para el ajuste de brillo) y entrelazado de dos dimensiones (para el despliegue progresivo); los archivos de *PNG-8* son relativamente chicos e ideales para ilustraciones e imágenes sencillas. *PNG-24* utiliza 24 bits para representar los colores de cada *pixel*, por lo que es ideal para imágenes fotográficas (en millones de colores) por lo que no hay pérdida de la calidad de la imagen, pero el tamaño de los archivos es relativamente grande. Es ideal para cualquier etapa de la edición profesional de imágenes, y la especificación de este formato permite que todas las aplicaciones interpreten unívocamente las imágenes de este formato.
- *Joint Picture Expert Group* (*JPEG*) que fue propuesto por el Grupo Articulado de Expertos en Fotografía (*Joint Photographers Expert Group*, *JPEG*), el cual se formó al juntar los esfuerzos de la *CCITT* (*International Telegraph and Telephone Consultative Committee*) con los de la ISO (*International Standards Organization*). Agrupa diversos métodos, varios de los cuales no fueron desarrollados explícitamente para almacenamiento de imágenes en computadoras digitales, y dentro de ellos se encuentra el principal algoritmo, el cual indica cómo tratar a un flujo de bytes tal y como se encuentran en la transmisión de una imagen. Este formato, como algunos otros, se basa en la idea de la compresión con pérdidas, en la que se busca reducir el tamaño de la imagen, tomando en cuenta cómo los seres humanos percibimos las imágenes; en vez de presentar toda la información de cada *pixel* de la imagen, se puede presentar valores aproximados del mismo, tomando en cuenta que percibimos sólo ciertas variaciones de matices de los colores, por lo que no se necesita tanta precisión para representar a cada color que compone al *pixel* (información). Por otra parte, en una imagen hay *pixeles* vecinos con valores muy parecidos entre sí, por lo que se podría almacenar la información de los *pixeles* más significativos -los representativos de un grupo de *pixeles* similares, o los que tengan cambios significativos respecto a sus vecinos.

La siguiente tabla indica las ventajas y desventajas de cada formato respecto a su uso sobre Internet.

Formato.	Ventajas.	Desventajas.
BMP		Son muy grandes para su transmisión en Internet.
TIFF		Los archivos sin compresión son muy grandes para su transmisión en Internet.
GIF	Ideal para guardar y/o transmitir imágenes vectoriales por su reducido tamaño, y para hacer animaciones sencillas.	Las imágenes fotográficas tienen un tamaño relativamente chico, pero pierden calidad.
PNG	*PNG-8* es ideal para imágenes sencillas y exento de pago de regalías; algunos sitios web generan y proporcionan una imagen con un "sello de agua" (agregando datos en un canal alfa) como comprobante. *PNG-24* no tiene pérdida de calidad para imágenes fotográficas.	No todos los navegadores le dan soporte a este formato. Los archivos de *PNG-24* son relativamente grandes para su transmisión en Internet.
JPEG	*JPEG* se volvió muy popular, ya que permite presentar imágenes con calidad fotográfica, con un tamaño relativamente pequeño, lo cual permite que se puedan transmitir por Internet. Todas las aplicaciones gráficas actuales que despliegan páginas web (navegadores o *browsers*) manejan ese estándar, así como las aplicaciones que despliegan y/o procesan imágenes digitales.	*JPEG* usa la transformada de cosenos discreta -*Discrete Cosine Transform*, *DCT*, que se parece a la transformada de Fourier-, la cual tiene pérdida de información (cuyo porcentaje puede ajustarse). Presenta distorsiones y defectos al perder información en bordes y esquinas, donde se requiere de mayores frecuencias para representar mejor esos detalles.

Tabla 5: Ventajas y desventajas de los formatos de imágenes digitales sobre Internet.

1.4.3.6.2 Formatos de video.

El video digital es una secuencia de imágenes digitales que brindan la sensación de continuidad, incluyendo sonido y/o texto (en otras pistas de estos formatos), y en general, la multimedia consiste en audio e imágenes cuyo contenido varía o depende del tiempo, tal como las secuencias de audio y/o video de los formatos anteriormente explicados. Cuando comenzaron a aparecer este tipo de formatos también comenzó el auge de Internet, por lo que también surgió la necesidad de transmitir video en Internet. En lo que se refiere a las imágenes, se dieron cuenta que hay más semejanzas que diferencias en una imagen y la que le sigue, por lo que aparecieron diversos formatos de compresión y/o predicción (de los cambios) de imágenes, del audio y de la información adicional, llamados *códecs* o compresores, los cuales también encapsulan la información en sus correspondientes pistas, para su almacenamiento y/o transmisión.

Cuando se quiere ver el video, lo que se hace es separar las pistas de audio, video e información adicional, para ser interpretadas y corregidas. A continuación se mencionarán los formatos de video más conocidos.

- *MPEG-2*, que codifica imágenes en movimiento y su correspondiente audio. Permite el flujo de video explorado(*scanned*) -progresivo o entrelazado-, y su unidad mínima es el campo o cuadro (dependiendo del tipo de flujo). El número de bits puede ser variable (*VBR*) o constante (*CBR*).
- *MPEG-4*, creado, a finales de 1998, para medios audiovisuales, videófonos y transmisiones televisivas.

Conserva muchas características del *MPEG-1* (mejor conocido como *MP3*) y del *MPEG-2*, pero incorpora nuevos estándares, como el *VRML* (para imágenes tridimensionales).

- *Windows Media Video* (*WMV*), el cual es un nombre genérico para el grupo de algoritmos de compresión de *Microsoft para el Windows Media* [20].
- *RealAudio (RM)*, el cual es un formato propietario de *Real Networks*, generalmente usado para transmisiones por Internet en tiempo real, por lo que se puede ver la transmisión sin que haya almacenado el video previamente. También se pueden ver videos por petición (*on demand*). Funciona al restringir el número de visitantes que tienen acceso a un archivo de video (en un servidor), al mismo tiempo, mediante su reproductor "*RealPlayer*"; el reproductor descarga el video en paquetes pequeños, mostrando una parte de ese video, y después aparece el aviso "*Buffering*", mientras se descarga otra parte del video. Para el dueño del video tiene la ventaja de que no se puede copiar.
- *Advanced Streaming Format* (*ASF*), este formato de Microsoft es contenedor, debido a que guarda audio y video de otros formatos, y fue diseñado para ver video en Internet sin tener que descargar todo el video.
- *QuickTime Movie* (*MOV*), el cual es otro formato contenedor, de *Apple*, el cual permite almacenar distintos formatos de audio, video e información, en sus correspondientes pistas, incluyendo el formato *H.264* (el cual ya forma parte del *MPEG-4*, parte 10).
- *3rd Generation Partnership* (*3GP*), Asociación de Tercera Generación, es un contenedor multimedia para celulares, similar al *QuickTime*.
- *Ogg* (cuyo nombre viene de un juego), fue creado por la Fundación Xip.org para comprimir audio y video; inicialmente era el *códec* para el formato de audio "*Squish*", pero después fue adaptado para otros *códecs* de audio y video de la misma organización.
- *Ogg Media* (*OGM*), que almacena audio, generalmente en formato *Vorbis*, y video en formato *DivX* o *Xvid*; surgió para resolver problemas de sincronización cuando se quería usar *AVI* usando *Ogg Vorbis* para el audio y *MPEG-4* para el video; es un *OGG* mejorado que tiene varios identificadores de formatos que permiten seleccionar el *códec* más fácilmente.
- *Matroska*, fue creado como contenedor universal mejorado, prácticamente compatible con cualquier *códec* de audio y video. Tiene las extensiones *mka* para audio, y *mkv* para video.
- *VP3*, creado por *On2 Technologies*, originalmente como *códec* propietario, con calidad similar a MPEG-4; en 2001 decidió donarlo como aplicación de código abierto.
- *Theora*, desarrollado por la Fundación Xiph.org a partir de un convenio con *On2 Technologies* hecho en 2002 con el que recibió el código de VP3, y liberado como aplicación de código libre. Este formato permite tener archivos *Ogg* en los cuales *Theora* es el *códec* de video, ayudando a usar audio y video sincronizados, sin tener que usar formatos cerrados o que exijan regalías.
- Video para *Windows* (*Video for Windows*, *VfW*) [20], consiste en un grupo de librerías de la plataforma *Windows* que permiten procesar datos de video; se introdujo por primera vez para las versiones de 16 bits. Aunque está arquitectura de multimedia se está reemplazando por *DirectShow*, también desarrollada por Microsoft, se sigue utilizando por muchos dispositivos ya existentes. Los componentes de este formato son funciones y macros de *AVIFile*, administrador de compresión de video, captura de video, archivo de especificaciones y controladores de flujo, y *DrawDib*; estos componentes son descritos en el anexo E.
- Audio y Video Intercalados (*Audio-Video Interleaved*, *AVI*) son usados por la arquitectura de multimedia *Video for Windows* (*VfW*); sus archivos tienen la extensión *AVI*. Generalmente tiene una pista de datos de video y una de audio, pero también puede tener ninguna o varias pistas de audio. Tiene una resolución de cuadros de 320 x 204 bits, a una velocidad de 30 cuadros por segundo, por lo que no son adecuados para mostrar video en pantalla completa, ni de animación de video completa. Guarda el audio y el video de forma simultánea, intercalándose los flujos de cada uno, para que, posteriormente, se reproduzcan simultáneamente. También es un formato contenedor, que guarda distintos formatos de audio y video. *AVI* se basa, a su vez, en el formato *RIFF*, el cual se forma con un encabezado de tipo *RIFF* seguido por cero o más listas y bloques de datos; *AVI* utiliza el código de Cuatro Caracteres (*four-character code*, FOURCCs) para identificar tipos de flujos, bloques de datos, entradas de índices y otro tipo de información. La estructura de este formato se explica en el anexo E.

- *DirectShow*, desarrollado por Microsoft para la plataforma *Windows*; es otro formato contenedor que puede almacenar y reproducir distintos formatos de audio y video (*ASF*, *MPEG*, *AVI*, etcétera). Le da soporte a al anterior formato de *Windows* (*Video for Windows*, *vfw*), y toma ventaja de la aceleración de hardware (si está disponible).

Como se había explicado anteriormente, la principal característica de la multimedia es que su procesamiento y entrega (emisión) dependen del tiempo; una vez que el flujo de datos inicia, los procesos de recepción y despliegue de los datos deben ser estrictamente sincronizados con el flujo. Cada flujo se identifica por su origen y por el protocolo usado para tener acceso a él (tal como *FILE* o *http)*. Los flujos multimedia se clasifican por el tipo de entrega de datos que efectúan, tal como se explica a continuación:

- Flujo por demanda, en el que el cliente inicia y controla la transmisión ("jala" los datos), como lo son los protocolos *http* y *file*.

- Flujo por emisión, donde el servidor inicia y controla la emisión ("empuja" los datos); *RTP* (*Real Transfer Procotol*) y *SGI Media Base* para video en demanda (*video-on-demand*, *VOD*) son ejemplos de protocolos de emisión.

A continuación se muestran los formatos multimedia (audio y video) más comunes, clasificados por sus principales características:

Formato	Tipo de contenido	Calidad	Requerimiento de CPU	Requerimiento de ancho de banda
Cinepak	AVI, QuickTime	Media	Bajo.	Alto
MPEG-1	MPEG	Alta	Alto	Alto
H.261	AVI, RTP	Baja	Medio	Medio
H.263	QuickTime, AVI, RTP	Media	Medio	Bajo
JPEG	QuickTime, AVI, RTP	Alta	Alto	Alto
Indeo	QuickTime, AVI	Media	Medio	Medio

Tabla 6: Formatos de Video.

Formato	Tipo de contenido	Calidad	Requerimiento de CPU	Requerimiento de ancho de banda
PCM	AVI, QuickTime, WAV	Alta	Bajo	Alto
Mu-Law	AVI, QuickTime, WAV, RTP	Baja	Bajo	Alto
ADPCM (DVI, IMA4)	AVI, QuickTime WAV, RTP	Media	Medio	Medio

MPEG-1	MPEG	Alta	Alto	Alto
MPEG Layer3	MPEG	Alta	Alto	Medio
GSM	WAV, RTP	Baja	Bajo	Bajo
G.723.1	WAV, RTP	Media	Medio	Bajo

Tabla 7: Formatos de audio.

1.5 Conceptos de programación básicos y avanzados del lenguaje de programación java.

1.5.1 Características básicas del lenguaje de programación java.

El lenguaje de programación Java fue concebido originalmente para la web, para crear programas interactivos llamados *applets*, que se ejecutan dentro las páginas web al desplegarse en los navegadores (*Netscape*, *Internet Explorer*, etcétera), pero posteriormente sus aplicaciones se extendieron, gracias a su versatilidad.

Conforme los usos de este lenguaje se extendieron, originaron un conjunto de herramientas llamado Tecnología Java, que consiste en lo siguiente:

- En un lenguaje de programación; en este capítulo se explicará este punto con mayor profundidad.
- Ambiente de desarrollo; cuenta con compilador, intérprete, depurador, etcétera, que se proporcionan en el *Java Development Kit* (*JDK*); además de las herramientas anteriores, se han construido Ambientes de Desarrollo Integrados (*Integrated Desktop Enviroment*, *IDE*).
- Ambiente de aplicaciones; las aplicaciones hechas en este lenguaje -programas de propósito general independientes de navegadores - son interpretadas en el *Java Runtime Enviroment* (*JRE*); ese ambiente cuenta con las clases básicas del lenguaje para poder llevara a cabo las instrucciones de los programas.
- Ambiente de distribución; existe el *JRE* proporcionado dentro del *JDK*, y el intérprete y ambiente de ejecución (*runtime*) proveídos en la mayoría de los navegadores, para el soporte de los *applets*.

Como se deduce de los primeros párrafos, la tecnología java sirve para crear o ejecutar (interpretar) programas hechos con este lenguaje de programación, por lo que el resto del punto 2.1 de este capítulo se enfocará en él, y se comenzará mencionando las características del mismo a continuación:

- Multi-plataforma.
- Orientado a objetos.
- Seguro y confiable.
- Distribuido.
- Multi-tarea

Las características antes mencionadas se explican con más detalle a continuación.

1.5.1.1 Multi-plataforma.

Multi-plataforma significa que el mismo código compilado se ejecuta en diversos sistemas operativos, por lo que se evita tener que modificar la programación por cada sistema operativo y /o re-compilar el código fuente, el cual contiene las instrucciones en lenguaje humano. El resultado de la compilación es el código de bytes (*bytecode*), que son instrucciones en lenguaje binario, independientes de la plataforma; dicho código se ejecuta sobre la máquina virtual de Java (*Java Virtual Machine JVM* o también conocida como intérprete de Java), la cual interpreta el código de bytes y lo convierte en instrucciones del sistema operativo.

La máquina virtual de java sí depende del sistema operativo, y se diseña específicamente para cada sistema operativo, pero al trabajar como una computadora virtual dentro de la misma computadora, garantiza la independencia del código compilado del sistema operativo en cual se ejecuta.

La desventaja de que este lenguaje sea multi-plataforma es que la ejecución de los códigos de bytes de un programa requiere más tiempo que un ejecutable (código nativo del sistema operativo) que a sido generado por un programa similar, ya que este último se comunica directamente con el sistema operativo, sin requerir el intérprete.

Aunque las nuevas versiones del lenguaje son más eficientes (versiones 1.2 en adelante) que las primeras (1.0 y 1.1), se puede compensar dicha "lentitud" optando por alguna de las siguientes opciones:

- Usando compiladores "justo a tiempo" (*Just In Time JIT*), los cuales convierte el código de bytes en código del sistema operativo en cual se ejecuta.

- Mediante el puente *JNI* (*Java Native Interface*), con el cual hace llamadas a código de máquina (código nativo del sistema operativo); esta opción se puede en procesos críticos -en el cual el código nativo se desempeña con mucha mayor eficiencia que un código de bytes de java-, o cuando se requiera inter actuar con dispositivos físicos (*hardware*) de la computadora donde se ejecuta.

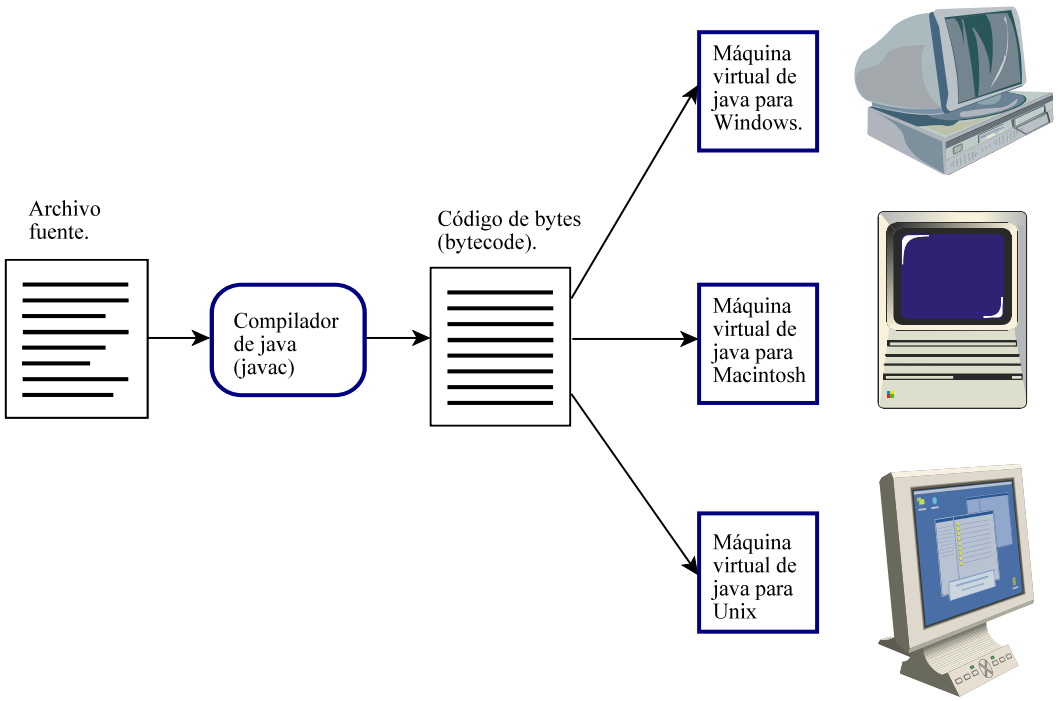

Figura 15: *Independencia del código de la plataforma.*

1.5.1.2 Conceptos de Orientación a Objetos del lenguaje java.

La programación orientada objetos (POO) consiste en simular el universo del problema a resolver, imitando las características y comportamiento de los elementos que intervienen, así como la interacción de dichos elementos entre sí. Esta característica también permite crear componentes que se pueden reutilizar.

La programación orientada a objetos permite crear código que se puede reutilizar, ya que una vez definida una clase, ésta se puede usar varias veces desde muchos lados, sin tener que modificar el código de la misma. Esta programación no se limita sólo a la clasificación de estereotipos, sino que cuenta con las siguientes principales características:

- Encapsulamiento.
- Herencia.
- Poliformismo.

Dichas características serán explicadas con más detalle a continuación.

1.5.1.2.1 Encapsulamiento.

El encapsulamiento indica el nivel de acceso de los atributos de un objeto para otros objetos, limitando u ocultando dichos atributos, y obligando al uso de una interfaz pública al objeto que desea conocer el valor del atributo. En el caso del lenguaje java, se usan los métodos accesorios para conocer o modificar el valor de un atributo, ya que de esta forma se controla el acceso a dicho atributo evitando incongruencias en otras partes del código.

1.5.1.2.2 Herencia.

La herencia significa que una clase recibe los atributos y métodos de la clase de la cual desciende, como si fueran propios. Es una característica transitiva, por lo que una clase (subclase) hereda todos los atributos y métodos de las clases (superclases) de las cuales desciende. La herencia establece una jerarquía entre las clases que heredan (superclases) y las clases que descienden de ellas (subclases).

También se pude ver a las superclases como clases muy genéricas, mientras que las subclases son especializaciones o clases restringidas de las primeras. Otro punto de vista es considerar a las subclases como extensiones de las superclases, ya que además de los métodos y atributos recibidos por herencia, agregan nuevos atributos y métodos.

En la programación orientada a objetos, si al diseñar una clase se encuentra que tiene muchos atributos y métodos en común con otra clase -y difiere en algunos atributos y métodos que son más específicos-, en vez de volver a escribir todos los métodos y atributos, se puede hace que la primera clase (más específica) herede de la segunda clase (más genérica), por lo que sólo quedaría escribir los atributos y métodos más específicos para la primera clase.

Al programar aplicaciones, generalmente se hace de forma ascendente, es decir, a partir de una clase dada, se concibe otra clase más genérica. Usando el concepto de herencia, se recomienda que la aplicación se construya de forma descendente, creando al principio clases más generales, y heredar (extender) de ellas para crear subclases más específicas, escribiendo los atributos y métodos que corresponden a ese nivel de abstracción.

Las ventajas de la herencia son que permite crear componentes re-utilizables, y ahorran tiempo al escribir y / o mantener el código, ya que un método que se va a heredar, se escribe una vez; en caso de modificaciones o corrección de errores, esta tarea sólo se hará una vez en un código, en contraposición de cuando no se hereda, ya que en este último caso se requiere cambiar *n* veces el código en las *n* veces donde se haya definido el método.

Las desventajas de usar herencia son: mayor tiempo de ejecución, requiere de más tiempo de ejecución para buscar y pasar mensajes a un objeto en memoria, hay más programas fuente y el diseño es más complejo. Sin embargo esas desventajas son mínimas, ya que la ventaja en tiempo de un código sin herencia es superada por la fiabilidad del código sí la use; las otras supuestas desventajas de la herencia quedan compensadas por la disminución del precio de las memorias, y por la mayor rapidez con que se desarrolla el código. Esto aplica siempre y cuando no se abuse del concepto de herencia al diseñar el sistema.

Existen dos tipos de herencia, que son:

- Herencia simple, que consiste en que una subclase hereda (o extiende) sólo otra clase.
- Herencia múltiple, en la cual una clase recibe herencia de una o más clases.

En el lenguaje de programación Java usa herencia simple, ya que la herencia múltiple presenta el problema en que, si dos superclases tienen un método con igual firma, no se sabe con claridad cuál método recibe la subclase que herede de ellas.

En caso de que una clase deba heredar de una superclase, pero pareciera conveniente que debiera recibir herencia de otra clase más, el lenguaje java emula la herencia múltiple mediante el uso de interfases, que consisten en imitar el comportamiento de otras clases.

1.5.1.2.3 Polimorfismo.

Polimorfismo viene de griego *poly* (muchas), y *morphos* (forma); literalmente, que tiene muchas formas. En el lenguaje java existen objeto y métodos polimórficos.

Un objeto polimórfico es aquel que pude contener valores de distintos tipos, por lo que puede tener muchas apariencias. En el lenguaje java, un objeto no sólo puede ser instancia de clase con que fue declarado, sino ser instancia de alguna subclase.

La sobrecarga de un método es cuando existen dos o más métodos con el mismo nombre; en este caso se dice que ese método es polimórfico, ya que se presenta de distintas formas en la misma clase.

Un método polimórfico en el lenguaje java, es aquel que tiene el mismo nombre que otro método definido previamente, regresa el mismo tipo de dato, pero varía el número y / o tipo de parámetros que recibe.

En la programación, el polimorfismo sirve para llevar a cabo una misma tarea, sin importar el número o tipo de parámetros; esto es como si un objeto adaptara su comportamiento a variaciones externas, con tal de cumplir el objetivo de dicho método.

1.5.1.3 Seguro y confiable.

Este lenguaje evita que existan códigos maliciosos y asegura que la información se maneje confidencialmente;

esto es, evita que se manipule la memoria directamente, por lo que no se permite el acceso a datos privados, ni los programas pueden alterar su código en tiempo de ejecución (como en el caso de los lenguajes 'C' o 'C++'). El modelo de seguridad tiene tres componentes:

- Cargador de clases, que coloca en memoria todas las clases necesarias para ejecutar un programa, cargando primero las clases obtenidas en forma local, y después las obtenidas a través de la red, para evitar "caballos de Troya". Al terminar de cargar las clases, determina el tamaño del ejecutable, y asigna direcciones específicas de memoria a la tabla de símbolos del compilador, evitando acceso no autorizado a dichas direcciones.
- Verificador de bytes, el cual revisa todo el código binario de las clases, comprobando su formato y evitando que no exista código que trate de crear apuntadores maliciosos, viole derechos de acceso a otros objetos, o trate de cambiar tipos de objetos (mutarlos).
- *SecurityManager*, el cual otorga o niega permisos a las aplicaciones a recursos específicos de la computadora, tales como permisos de escritura o lectura de archivos, establecer conexiones de red, o a la memoria. Los *applets*, por ejemplo, no tienen acceso a los recursos de la máquina donde se ejecute; estos permisos se pueden configurar, y junto con firmas digitales, se pueden otorgan más permisos de los que tienen por defecto las aplicaciones.

1.5.1.4 Distribuido.

Cuando se inició el mundo de la computación, tanto como la *CPU* (*Central Process Unit*), los recursos de cómputo y los programas se encontraban en un sólo equipo de cómputo, el servidor central (*mainframe*), al que los usuarios tenían acceso por medio de una terminal.

Conforme las computadoras se volvieron más pequeñas, rápidas y versátiles, surgieron las computadoras de escritorio *PCs* (*Personal Computer*, computadora personal), donde los usuarios ejecutaban una gran variedad de programas; el siguiente paso fue que se comunicaran las computadoras personales con los servidores centrales, para obtener información, y ejecutar programas descargados de esos servidores.

De ese enfoque nace el concepto cliente-servidor, en el que los clientes obtienen de un servidor común el programa y la información para ejecutar el mismo. La ventaja de este enfoque es que el cliente realiza parte del procesamiento, ahorrando trabajo al servidor. Posteriormente este modelo se mejoró usando servidores espejo y evitando que se descargue el programa cliente cada vez que requiera usando. Estos últimos programas se volvieron más complejos y grandes, y surgió el inconveniente, de que por cada sistema operativo del cliente (e incluso por versión del mismo), se tenía que desarrollar una versión del programa cliente. También surgió el problema, de que si cambia el protocolo del programa del servidor, se tenía que desarrollar una nueva versión del programa cliente, e instalar ese nuevo programa en todos los clientes.

La solución a los inconvenientes anteriores fue la computación distribuida, en que los programas cliente y servidor se ejecutan en ambientes heterogéneos, y los clientes solicitan recursos específicos de otros servidores o clientes -que se encuentran en otras direcciones de la red, local o remotamente-, usando protocolos de comunicación estándares. Los programas cliente invocan a otros programas, y a veces, atienden a otros clientes, convirtiéndose en servidores. Las ventajas de esta arquitectura son las siguientes:

- Disminuye el costo del desarrollo de los programas, ya que diversos programas cliente invocan al mismo programa servidor, evitando multiplicidad de esfuerzo.
- El uso de los recursos está más equilibrado, ya que el trabajo se distribuye con el paradigma cliente-servidor, por lo que los recursos de memoria, *CPU* y disco duro se utilizan más adecuadamente.
- Permite la independencia del código de la plataforma (*hardware*), mediante la comunicación de protocolos

estándares.

1.5.1.5 Multi-tarea.

En computadoras con dos o más *CPUs* se permite el multiproceso, en el cual puede haber dos o más procesos independientes entre sí, y que se ejecutan simultáneamente. En las computadoras con un sólo *CPU*, se simula lo anterior mediante la multi-tarea, que consiste en asignar a cada proceso un intervalo de tiempo de ciclo del reloj del *CPU*, el cual se divide entre todos los procesos; durante ese intervalo el *CPU* atiende un proceso, y al terminar guarda el estado del proceso; luego sigue con otro proceso, y así sucesivamente; al terminar el ciclo, vuelve a comenzar atendiendo al primer proceso. En ambos casos es el sistema operativo el encargado de controlar y / o asignar tiempo en el ciclo del reloj del *CPU* a los procesos.

El lenguaje de programación java aprovecha la característica arribe mencionada, y permite que se ejecuten varios proceso simultáneamente, por lo que es un lenguaje multi-tarea y permite la concurrencia. La concurrencia permite que varios procesos que se ejecutan simultáneamente compartan recursos y se comuniquen entre sí. La forma en que en que este lenguaje implanta los procesos multi-tarea, es mediante hilos (*threads*), los cuales son procesos más ligeros, y más rápidos de inicializar y utilizar que los procesos de los sistemas operativos.

Las primeras versiones de este lenguaje incluían versiones para *Unix* y otros sistemas operativos multi-tareas, mientras que para *Windows* se ofrecieron versiones para *Windows* 95, 98 y NT, pero no para *Windows* 3.1, ya que el mismo era un ambiente gráfico que no permitía multi-tareas.

1.5.2 La Programación Orientada a Objetos y la Ingeniería de *Software*.

La ingeniería del *software* consiste en aplicar principios sólidos de ingeniería a la construcción de sistemas de cómputo que funcionen fiable y eficientemente sobre máquinas reales.

Para cumplir con el objetivo anterior, se estableció el ciclo de vida de la Ingeniería del *software*, cuyos pasos son los siguientes:

- Análisis, se estudian los requisitos del sistema a resolver, y de los recursos disponibles.
- Diseño, que el desarrollo de una solución para el sistema a resolver.
- Implantación, que es la construcción del sistema de cómputo.
- Pruebas, verificación de que el sistema construido cumpla con las especificaciones indicadas.
- Mantenimiento, el cual se debe a algún cambio en los requerimientos originales, y que surge después de haber sido entregado o liberado el sistema de cómputo; en este caso se aplican los primeros cuatro puntos a un sistema de cómputo ya existente -algunos autores omiten este punto ya que consideran que este ciclo es continuo.

La programación orientada a objetos surge como un modelo para poder crear programas y sistemas de cómputo que se volvieron más completos durante la década de los 80's, los cuales ya no podían ser modelados o resueltos con metodologías anteriores. La metodología de la programación orientada a objetos permite llevar a cabo el paso de **Diseño**, del ciclo de vida de la Ingeniería de *Software*.

En este tipo de programación se especifican los elementos llamados **clases**, las cuales son plantillas que contienen **atributos** (características) y **métodos** (comportamiento), con los cuales se modelan los elementos que intervienen en el sistema de cómputo a desarrollar; los atributos y métodos de una clase son los miembros de la clase

en la cual fueron definidos.

1.5.3 Java y la computación distribuida.

El lenguaje java trató desde un principio aprovechar el concepto de computación distribuida, -tal y como se mencionó anteriormente en este capítulo, por lo que fue concebido como un lenguaje para ejecutar aplicaciones en y desde Internet, pretendiendo que controlara una red de dispositivos llamados *star7*.

Aunque no se popularizó la idea de que cafeteras y tostadoras se comunicaran en red con el refrigerador usando este lenguaje, lo que sí se popularizó fue el uso de los *applets*, los cuales son mini aplicaciones interactivas que se ejecutan dentro de la *Java Virtual Machine* (*JVM*) del navegador, cuando se descarga una página *HTML* que contenta una etiqueta *applet*, donde invoca a dicha aplicación.

Conforme el uso de los *applets* se fue extendiendo, también se puso en evidencia las desventajas de los mismos, las que mencionamos a continuación:

- Tamaño, los *applets* que son muy grandes, tardan mucho tiempo en descargase; una solución a este problema fue empaquetar el código en un archivo jar, y comprimir el mismo.
- Clases no estándares, hay que incluir dichas clases en el jar del *applet*, o solicitar que el usuario (por su propia cuenta) instale un "*plug-in*" en los navegadores, el cual contiene otras librerías con más clases y de versiones recientes, que las contenidas en la *JVM* que tienen por defecto los navegadores.
- Problemas de seguridad; los *applets*, por defecto, no tienen acceso a los recursos de la computadora donde se despliega; en caso de que requiera tener acceso a uno de esos recursos, tiene que ser firmado digitalmente, y cuando se descarga tiene que solicitar permisos al usuario para ejecutarse.

Como solución a los problemas anteriores, aparecieron otros componentes de tecnología Java, que se ejecutan en equipos servidores, a la vez que las aplicaciones java (ya existentes) se popularizaron ya que tienen pleno acceso a los recursos de los equipos donde se ejecuten, sin tantas restricciones de seguridad como los *applets*.

Tanto los *applets* -que inicialmente se usaban como animaciones gráficas-, las aplicaciones y otros componentes de tecnología Java, comenzaron a diversificar sus aplicaciones obteniendo recursos de Internet, inter actuando con base de datos, o incluso se construyen servidores que atienden a otras aplicaciones. Después, este lenguaje incorporó comunicaciones seguras (*https*) y *webservices*; más recientemente permitió que se descargue toda una aplicación mediante un *webservice*, y ejecutarla en el ámbito local mediante la aplicación de Java *WebStart*.

El principal paquete del lenguaje para establecer conexiones es *java.net*, donde se encuentran las clases *Socket*, *ServerSocket* y *URL*, entre otras; las dos primeras clases permiten la comunicación sobre ***TCP/IP***, estableciendo conexiones estándares de equipo a equipo, y la tercera permite la comunicación usando los protocolos más comunes de *URL* (como *ftp* y *http*).

Existen otros paquetes, como *javax.net* y *javax.net.ssl*, que permiten configuraciones más específicas para las conexiones y para manejar comunicaciones seguras, respectivamente. Si se desea implantar este tipo de arquitectura, se deben tomar en cuenta los siguientes dos factores:

- Latencia, que el tiempo de espera a una solicitud hecha; se debe analizar cuáles recursos deben ser locales y cuáles pueden tener acceso local. Los recursos a los cuales se tiene mediante un bus o tren de datos local responden más rápidamente que aquellos solicitados por la red; mientras los microprocesadores aumenten de número y de rapidez, la diferencia en el tiempo de respuesta entre un recurso local y uno remoto aumenta

considerablemente.

- Fallo parcial; si una aplicación local falla, por daño en el disco duro, la memoria se llena o desborda, o el *CPU* se comporta erráticamente, dicha aplicación fallará al tratar de ejecutarse; cuando se presentan dichos casos, una aplicación casi no puede recuperarse. En cambio, en una aplicación distribuida, un componente puede fallar (el servidor, la red o el programa cliente), por lo que los fallos parciales pueden ser detectados e iniciar procedimientos de recuperación (re intentar la solicitud, solicitar el servicio a otro servidor, esperar y re intentar, etcétera).

Algunas de las tecnologías java distribuidas son ***JDBC***, ***servlets*** y ***JSPs*** las cuales son explicadas más adelante.

1.5.3.1 *JDBC*.

La tecnología *Java Database Connectivity* (*JDBC*) es el estándar del lenguaje de programación Java para conectividad, independiente de base de datos, a distintos servidores de bases de datos. Las tres tareas principales que puede llevar a cabo son:

- Establecer una conexión a una base de datos o a una fuente de datos tabular (hoja de cálculo o archivo de texto).
- Enviar sentencias *SQL*.
- Procesar los resultados de las sentencias anteriores.

La arquitectura de la *API* de ***JDBC*** se divide en dos partes; una de ellas consiste en un conjunto de interfases que son usadas por los programadores de aplicaciones, y la otra parte son las interfases de bajo nivel usadas por los programadores de los controladores (los que efectúan la conexión a la base de datos). Las *APIs* de ***JDBC*** se clasifican en los siguientes cuatro tipos:

- Tipo 1: Puente *JDBC-ODBC*. *ODBC* (*Open Database Connectivity*) es una tecnología de Microsoft que permite el acceso a base de datos en la plataforma *Windows*. Este puente permite que aplicaciones del lenguaje java tengan acceso a controladores nativos (binarios de *Windows*) *ODBC* instalados para cierta base de datos. Aunque *ODBC* es un protocolo popular en Windows, no se recomienda para sistemas de producción, ya que presenta problemas de concurrencia al usar multi hilos.
- Tipo 2: Controlador parte Java con *API* nativo. Este controlador usa código nativo (del sistema operativo), el cual efectúa la conexión al administrador de base de datos, y maneja el protocolo con el que se comunica con dicho administrador. El código del lenguaje java encapsula al código nativo, y permite que las aplicaciones java tengan interacción con el mismo. Generalmente, este tipo de controladores tiene mejor rendimiento, pero la tiene la desventaja de que hay que instalar cierto código nativo en cada cliente.
- Tipo 3: Controlador de java puro, para "*middleware*" de base de datos (***JDBC****-Net*). Este tipo de controladores se comunica con un *middleware*, por medio de un protocolo de red independiente del administrador de base de datos (como *http*), por lo que es ideal para aplicaciones de Internet. Tiene una conexión indirecta a la base de datos, en la cual el "*middleware*", generalmente escrito en código nativo, administra las transacciones con la base de datos.
- Tipo 4: Controlador de java puro, de acceso directo a base de datos. Los controladores de este tipo se comunican con el administrador de base de datos mediante su protocolo de red propietario. Por lo general, es más pequeño y eficiente que otros controladores, ya que se comunica directamente con el administrador de base de datos, y sólo tiene clases del lenguaje java –no tiene que instalar código nativo. Es una excelente solución dentro de Intranets, pero puede haber problemas cuando tenga que resolver la dirección *URL* cuando

existen cortafuegos en la red.

A continuación se muestra una pequeña tabla para mostrar las propiedades de los tipos de controladores antes explicados.

Tipo de controlador:	¿E s java puro?	Protocolo de red:
Puente ***JDBC***-*ODBC*	No	Directo
Controlador Java con API nativa	No	Directo
Controlador Java ***JDBC***-*Net*	Sí	Conector
Controlador con protocolo nativo	Sí	Director

Tabla 8: Tipos de controladores *JDBC*.

Las principales características de ***JDBC*** son la siguientes:

-Permite total acceso a la metadata; además de dar soporte a *SQL*, tiene acceso a la metadata del administrador de base de datos, e incluso a distintos objetos de esa base.

-No requiere de instalación; sólo requiere de obtener las clases de controlador -al descargarlas por medio de un *applet* o al cargarlas desde un directorio conocido.

-Conexión a base de datos por medio de *URL*, lo que permite usar estándares de Internet para encontrar e identificar un recurso en la red; también permite encontrar fuentes de datos, para usar conexiones almacenadas y transacciones distribuidas, de forma transparente al usuario.

-Incluido dentro de la plataforma Java, por lo que está disponible en el lenguaje estándar, o se puede descargar de forma independiente. Los controladores para un administrador de base de datos específico, generalmente, se pueden obtener de forma gratuita del fabricante de base de datos; actualmente, la mayoría de los fabricantes de administradores de base de datos proveen de estos controladores, además de los controladores tradicionales que ya elaboraban (propietarios, *ODBC* o *SQL Links* de *Borland*).

JDBC tiene las siguientes ventajas:

-Tiene un mayor y mejor acceso a la información en el ámbito corporativo, al permitir que se siga utilizando su base de datos actual, pero la libera de seguir usando sólo arquitectura propietaria, al permitir que se construyan aplicaciones sobre diferentes plataformas, e incluso, usar distintos administradores de base de datos.

-Simplifica el desarrollo empresarial, al hacer el desarrollo de aplicaciones más barato y fácil, ya que ***JDBC*** efectúa las tareas complejas de conexión y manejo del protocolo por el programado; el *API* es fácil de programar, distribuir y de mantener.

-Sin necesidad de configurar para computadoras en red. A diferencia de otras tecnologías, como *ODBC* o *SQL Links*, que requieren configurar la conexión del cliente en un archivo del sistema, ***JDBC*** no requiere de configuración, ya que toda la información para la conexión se define en una URL de ***JDBC*** o en una fuente de datos (*DataSource*) registrada en un servicio *JNDI* (*Java Naming and Directory Interface*).

-Permite implantar sistemas de dos o más capas, al proveer a las aplicaciones cliente el código de conexión al administrador de base de datos- o a un *middleware*- distribuyendo las cargas de trabajo.

-Total soporte al estándar *ANSI SQL 92*. La gran mayoría de los fabricantes de base de datos se atienen a ese estándar, y aparte proveen otras funciones no estándares. Las *APIS* de *JDBC* puede llevar a cabo cualquier operación de dicho estándar, tal como selección, borrado, inserción y actualización. Además, dependiendo del fabricante, en el controlador ***JDBC*** del fabricante se encuentran clases del lenguaje para el manejo de elementos no estándares de su base de datos.

-Controladores ***JDBC*** disponibles. La mayoría de los fabricantes de datos ya proveen controladores ***JDBC*** para sus bases de datos, muchos de los cuales se proveen de forma gratuita, aparte de las aplicaciones cliente que ya ofrecían. Incluso, existen compañías que venden controladores ***JDBC*** para base de datos cuyos fabricantes no provean dicha aplicación.

La estructura de la *API* de ***JDBC*** se compone en su gran mayoría por grupos de interfaz, las cuales son presentadas por *Sun Microsystems* para indicar la funcionalidad que deben presentar; los fabricantes de datos son los encargados de implantarlas, siendo su responsabilidad definir cómo funcionan.

Las clases que conforman ***JDBC*** se dividen en dos niveles:

- Nivel del controlador, que se relaciona estrechamente con el administrador de base de datos, y maneja la forma en se conecta al administrador antes mencionado, y el protocolo con el que se comunica con el mismo. Las interfases más importantes que tiene son *DriverManager* y *Driver*.
- Nivel de la aplicación que se compone por tres interfaz principalmente, *Connection*, *Statement* y *ResultSet*, las cuales son implantadas por el fabricante de base de datos, pero que quedan a disposición de los programadores de aplicaciones (usuarios de la *API* ***JDBC***).

Las implantaciones de las interfases del nivel de aplicación son las usadas por los programadores para efectuar consultas y obtener datos; el sistema desarrollado tiene interacción con el administrador de clases *MySQL* por medio de estas clases.

1.5.3.2 *Servlets*.

Los ***servlets*** son componentes java creados para generar páginas web dinámicas - e incluso realizar acciones más completas- respondiendo a peticiones *HTTP* hechas a un servidor de aplicaciones. Este servidor trabaja junto con un compilador del lenguaje java, de tal forma que al recibir una petición a un recurso (en este caso, el ***servlet***), interpreta el código java del ***servlet*** efectuando diversas tareas, tales como ejecutar consultas a bases de datos, y generar una página HTML dinámica en la cual se incluyen los datos de la consulta.

Los ***servlets*** tienen a su disposición todos los recursos del equipo donde se ejecutan, a diferencia de los *applets*, los cuales tienen restricciones sobre los recursos del equipo cliente donde se descargan; incluso, ante tan amplios permiso con que cuentan los ***servlets***, es recomendable que restrinjan esos permisos en los contenedores de ***servlets*** o servidores de aplicaciones donde residan esos códigos, modificando el archivo *policy.proproperties*.

Los ***servlets*** se construyen con la *API* ***Servlets***, la cual es una extensión estándar del lenguaje de programación java; el API consta de dos paquetes, *javax.servlet* y *javax.servlet.http*, las cuales contienen clases e interfaz que definen el comportamiento de estos componentes. Al quedar esta *API* como una extensión estándar, permite que otras diversas compañías lleven a cabo sus propias implantaciones (con base en las especificaciones de *Sun Microsystems*) para sus propios servidores de aplicaciones.

Las principales ventajas de los ***servlets*** sobre otras tecnologías son las siguientes:

- Por cada vez que se invoca a un ***servlet***, se crea un nuevo proceso exclusivo para atender esa petición, en vez de que exista sólo un proceso que atienda a todas las peticiones, como en *CGI*.
- Multi-plataforma, como ya se había mencionado varias veces, los *bytecodes* de los ***servlets*** se ejecutan en distintos sistemas operativos, por lo que puede cambiar una aplicación web a otro servidor de aplicaciones de otro sistema operativo (que tenga mayor capacidad), o como estrategia de desarrollo, un equipo de trabajo puede hacer sus ***servlets*** en equipos *PCs*, y reunirlos todos en un equipo *Unix* de mayor capacidad.

La clase principal de este *API* es *GenericServlet*, la cual trabaja independientemente del protocolo, mediante su método service.

La clase más popular de este *API* es *HttpServlet*, la cual extiende *GenericServlet*, con el fin de atender peticiones con el protocolo *HTTP*.

Los ***servlets*** tienen el siguiente ciclo de vida:

1. Inicio, con el método *init*, el cual se ejecuta una vez, cuando se carga el ***servlet***; sirve para recuperar parámetros de configuración de la aplicación, y en general, obtener recursos que se utilizarán varias veces posteriormente.

2. Servicio, el cual lleva a cabo el método *service*; inicia un nuevo proceso para atender cada petición al ***servlet***; la excepción a la regla anterior es cuando se implanta el modelo *SingleThreadModel* (también llamado *Singleton*), el cual crea un sólo proceso para el ***servlet***. En la clase *HttpServlet* este método llama un método *doXXX*, el cual atiende en forma más específica la petición del cliente.

3. Destrucción, con el método *destroy*, el cual es llamado por el servidor de aplicaciones cuando se va a descargar el ***servlet***, al "tirarse" el servidor o detener la aplicación web donde resida; este método sirve para cerrar conexiones a bases de datos, cerrar archivos, y para liberar recursos que estaban asignados al ***servlet***.

Los ***servlets*** no sólo responden peticiones ***HTTP***, sino que también pueden manipular los datos de la petición, configurar la respuesta, y manejar la sesión del usuario en la aplicación web, mediante las interfases (o más bien la implantación de las mismas) *HttpServletRequest*, *HttpServletResponse* y *HttpSession*. Las interfases más importantes son descritas a continuación.

HttpServletRequest define cómo debe responder a las peticiones del cliente; sus implantaciones permiten recuperar los parámetros de la consulta, "galletas", el tipo de petición (*GET* o *POST*), la *IP* de la máquina cliente y otros datos de la petición. Se especializa en peticiones hechas por el protocolo *HTTP*, y hereda muchos métodos de *ServletRequest*, la cual maneja peticiones más genéricas.

HttpServletResponse es una interfaz, contraparte de *HttpServletRequest*, que indica cómo se responde a la petición del cliente; Las implantaciones de esta interfaz permite configurar la respuesta, especificando encabezados HTTP o agregando galletas. Por otra parte, hereda de ServletRequest varios métodos que indican el tipo de respuesta (*HTTP*, texto, algún archivo binario de un formato conocido, etcétera), la codificación (juego de caracteres), así como otros que abren flujos de entrada y salida, para enviar la respuesta (por ejemplo, la página web) y recibir mensajes del cliente (para controlar la transmisión).

HttpSession es una interfaz muy importante, ya que sus implantaciones permite almacenar información del usuario a través de varias páginas, durante la visita del mismo a la aplicación web. Cada vez que se detecta un nuevo cliente llamando a un ***servlet***, se crea un objeto sesión, el cual comienza a guardar datos del cliente.

ServletContext es una interfaz permite la comunicación entre los ***servlets*** y el contenedor de ***servlets***, re-dirigir peticiones, y escribir a la bitácora del contenedor; existe una instancia del objeto (que implanta esta interfaz) por cada aplicación web. Además cuenta con los métodos *setAttribute*, *getAttribute*, y *removeAttribute*, con los cuales guarda y recupera objetos en el ámbito de contexto, lo que permite que la información de dichos objetos se comparta a todas las sesiones de los usuarios de la aplicación web; a los objetos en este nivel también se les llaman atributos de aplicación, y según la implantación del contenedor de ***servlets***, estos objetos pueden ser conocidos por todas las aplicaciones web que en él residan.

El sistema desarrollado para esta tesis se basa principalmente en estas clases.

1.5.3.3 JSPs.

La tecnología *Java Server Pages* ***JSPs*** consiste crear páginas web dinámicas, al agregar código Java (para hacer el código dinámico) dentro del código *HTML* (el cual provee la presentación); una página de este tipo consiste en código HTML -la cual pudo ser hecha por un diseñador web con una aplicación de diseño gráfico- en un archivo con extensión "jsp", con fragmentos de código Java incrustados (llamados *scriplets*).

La ventaja de este enfoque es que estas páginas pueden ser creadas por un diseñador gráfico, quien tal vez no tenga conocimientos de lenguajes de programación, pero que se encarga de darle la presentación gráfica adecuada al archivo *HTML* (a renombrar a ***JSP***) o sobre un archivo ***JSP*** ya existente. Posteriormente, un programador (el cual puede carecer de habilidades de diseño gráfico), agrega los *scriplets*, para que la página tenga comportamiento dinámico. En cambio, los ***servlets*** o tecnologías similares, requieren ser compilados aunque el cambio sea sólo en la presentación y no en la funcionalidad del mismo (tarea que puede requerir varias pruebas, consumiendo mucho tiempo).

El archivo ***JSP*** resultante del diseño se coloca dentro del servidor de aplicaciones o contenedor de ***servlets*** (que también le da soporte a los ***JSPs***), dentro de la aplicación web en su ruta correspondiente (como si fuera una página *HTML*). Por cada petición a un ***JSP***, se busca en el contenedor el recurso solicitado, para revisar si tiene o no su ***servlet*** generado, comenzando con lo que se conoce como ciclo de vida del ***JSP***, cuyas fases son descritas a continuación.

- Pre-traducción: Es para un ***JSP*** que se solicita por primera vez, o que tenga cambios recientes.
- Traducción y compilación: En la fase de traducción se lee el archivo del ***JSP***, se interpreta su sintaxis, y se genera el código java correspondiente a un ***servlet***; el tiempo usado en esta fase se le conoce como tiempo de traducción. Luego sigue la compilación, en la cual se lee el código java generado anteriormente y se crea un ***servlet***, el cual se registra en la máquina de ***servlets***. De lo anterior se observa que el servidor siempre tardará más en responder cuando se solicita un ***JSP*** por primera vez, respecto a ocasiones subsecuentes.
- Inicialización, donde se asignan valores o se asignan recursos al ***JSP***, antes de recibir una petición.
- Servicio, en la cual atiende una petición.
- Fuera del servicio, la cual inicia el motor de ***JSPs*** (o de ***servlets***), detiene la aplicación web, por lo que el ***JSP*** es destruido y ya no puede recibir más peticiones.

Los componentes ***JSPs*** son instrucciones y código java incrustados dentro del código *HTML* del ***JSP***. Estos componentes se listan y explican a continuación:

- Elementos de guiones de comandos, que a su vez se dividen en:
  - Comentarios ***JSP*** y *HTML*: *<%–Comentario JSP.--%>, <!– - -–>*.
  - Declaración de variables: *<%! int nombreVariable=valor; %>*

- Expresiones java, las cuales sirven para mostrar el valor de una sentencia del lenguaje java, es decir, despliega el resultado de una operación, o el valor de un objeto -para esto, debe poder convertirse en cadena (*String*)-, o el valor de un tipo primitivo; dentro de su sintaxis no se incluye el caracter ";": *<%= expresionJava%>*
- *Scriplets*, que consisten en una o más sentencias válidas del lenguaje java, cuya sintaxis es la siguiente: *<% Sentencia(s) del lenguaje java %>*

- Directivas ***JSP*** cuya sintaxis es *<%@ directiva{atributo="valor"}* %>*, son instrucciones que la máquina de ***JSPs*** debe atender a la hora de interpretar y compilar el ***JSP***; no generan salida y deben estar al principio del ***JSP*** (con la excepción de la directiva *include*). A su vez, se dividen en los siguientes tipos:
  - *page*, con sintaxis *<%@ page listaAtributos %>*, donde *listaAtributos* es del tipo *nombreAtributo="valorAtributo"*.
  - *taglib*, la cual sirve para indicar que se van a utilizar etiquetas ***JSP*** creadas; su sintaxis es la siguiente: *<%@ taglib uri="URI_del_taglib" prefix="prefijo" %>*
  - *include*, la cual indica que se añade otro archivo dentro de la respuesta de este ***JSP***, lo cual se hace cuando el ***JSP*** se convierte a ***servlet***; su sintaxis es: *<%@ include file="URL" %>*

- Elementos de acción, que se dividen en las siguientes categorías:
  - Etiquetas de control, las cuales regulan el flujo de la navegación de la página, y de los *plug-ins*, tales como *<jsp:include>*, *<jsp:forward>*, *<jsp:plugin>* y *<jsp:param>*.
  - Etiquetas de *beans*, las cuales son sólo para los *JavaBeans*; permiten almacenar e intercambiar información entre el ***JSP*** y el contenedor donde residan.
  - Etiquetas hechas a la medida (*custom tags*), las cuales no forman parte de las librerías estándares, y que fueron programadas con un fin específico, como por ejemplo, crear una lista desplegable cuyo identificador sea numérico pero que se despliegue una descripción, y que los pares de datos se obtengan de una base de datos.

Como ya se había mencionado, los ***JSP*** se convierten a ***servlet***; a la hora de hacerlo se tienen convenciones de cómo se hace la conversión y qué objetos se crean. De estos últimos hay algunos que de antemano se conoce su nombre, pertenecen a las clases más usadas del paquete *httpservlet*, y a los cuales se tiene acceso. A continuación se nombran los objetos estándares conocidos dentro del ámbito de un ***JSP***:

Objeto implícito	**Clase a la que pertenece**	**Uso o función del objeto.**
application	javax.servlet.ServletContext	Este objeto tiene acceso al contexto de los ***servlets*** de la aplicación web (donde residen).
config	javax.servlet.ServletConfig	Lee y almacena la configuración inicial de la aplicación (contenida en web.xml).
exception	java.lang.Throwable	Este objeto sólo se puede usar dentro de un ***JSP*** de error; contiene la excepción que ocurrió en un ***JSP***.
out	javax.servlet.jsp.JspWriter	Tiene el control del flujo de salida, y mostrar alguna información en el ***HTML*** resultante.

page	java.lang.Object	Es la instancia del ***servlet*** que se generó a partir del ***JSP***; es equivalente a *this*, y es poco usado.
pageContext	javax.servlet.jsp.PageContext	Este objeto tiene acceso al contexto de los ***JSPs***.
request	javax.servlet.http.HttpServletRequest	Es el objeto que maneja la petición del cliente.
response	javax.servlet.http.HttpServletResponse	Este objeto maneja la respuesta al cliente.
session	javax.servlet.http.HttpSession	Maneja la sesión del cliente en la aplicación web.

Tabla 9: Objetos predefinidos en los *JSPs*.

1.5.4 La *API Java Communications.*

El paquete de comunicaciones *Communications API* permite la comunicación con los puertos serie y paralelo (***RS232*** e ***IEEE 1284***), en las plataformas *Linux*, *Solaris* y *Windows*. Es un paquete de extensión, ya que no se incluye en el *JDK*. Contiene las clases para utilizar los puertos serie y paralelo (independientemente de la plataforma), y código nativo -una *dll* para Windows, o librerías compartidas (*Shared Object*, de extensión *so*) para *Unix* y *Linux*. Incluye la siguiente funcionalidad:

1 Enumerar los puertos (serie y paralelo) disponibles en dicho equipo.
2 Abrir y señalar que está siendo ocupado el puerto de forma exclusiva (ante otras aplicaciones).
3 Resolver conflictos de uso de los puertos entre varias aplicaciones que intenten ocuparlo.
4 Realizar operaciones de entrada y salida de datos síncronas y asíncronas por los puertos.
5 Recibir y atender eventos de tipo *JavaBean* que indican cambios de estado en los puertos.

Este *API* está compuesto por tres niveles clases, que son:

1 Clases de alto nivel, que se encargan de tener acceso y registrar su uso, tal como *CommPortIdentifier* y *CommPort*.
2 Clases de bajo nivel, las cuales proveen una interfaz para las conexiones físicas a los puertos, como *SerialPort* y *ParallelPort*; dichas clases son las que se comunican con los puertos serial y paralelo, respectivamente.
3 Clases de nivel de controlador, que conforman una interfaz entre las clases de bajo nivel y el sistema operativo sobre el cual opera; no deben ser usadas por los programadores.

La forma en que se usa este *API* es la siguiente:

1 Detecta los puertos.
2 Se busca un tipo de puerto.
3 Se busca un puerto en específico, llamándolo por su nombre.
4 Abrirlo, en indicar que está en uso exclusivo.
5 Envío o recepción de datos a través del puerto.
6 Cerrar el puerto.

1.5.5 La *API Java Media Framework.*

Esta *API* permite manejar, almacenar, reproducir, capturar y convertir diversos formatos de audio y video, por lo que provee de herramientas multimedia multi-plataforma al lenguaje de programación java. Los objetivos de esta *API* son:

- Permite controlar, procesar y dar formato a multimedia.
- Controlar dispositivos y datos de entrada de multimedia.
- Provee acceso a datos multimedia sin formato, provenientes de dispositivos de captura.
- Permitir el desarrollo de demultiplexores, *códecs*, procesadores de efector, multiplexores y herramientas para dar formato y desplegar imágenes, los cuales puedan ser fácilmente obtenidos y ajustados por las aplicaciones clientes.
- Permitir a desarrolladores avanzados y proveedores de tecnología crear soluciones personalizadas, que tengan base en la *API* existente, pero a la vez que pueda integrar nuevas características.
- Incorpora la *API RTP*, la cual permite la transmisión y recepción de flujos de multimedia en tiempo real a través de la red, así como de aplicaciones que provean multimedia en demanda y servicios interactivos, tal como la telefonía en Internet.
- Ser fácil de programar.

Esta *API* tiene una arquitectura de alto nivel, por lo que posee un nivel de abstracción que modela a los elementos multimedia de forma similar a los que se maneja en cualquier otro sistema multimedia, digital o analógico; sus principales componentes son:

- Dispositivos de captura, recogen información a través de sensores, la cual guardan en una fuente de datos.
- Fuente de datos, almacena los datos multimedia en un formato conocido.
- Reproductor, lee los datos de la fuente de datos, utilizando controles para manipular su presentación en los dispositivos de salida.
- Dispositivos de salida, son el destino donde se presenta la multimedia.

También tiene una capa de bajo nivel, el "*Plug-in*" *JMF*, que permite integrarse con extensiones de esta *API* y con otros componentes, tales como multiplexores, demultiplexores, *códecs*, filtros de efecto y dispositivos de presentación.

La *API JMF* consiste básicamente de interfaz que definen el comportamiento e interacción de los objetos usados para capturar, procesar y presentar la multimedia, dentro de la arquitectura especificada por este API. *JFM* usa unos objetos intermedios llamados administradores para integrar articuladamente nuevas implantaciones de las interfases principales con nuevas clases; dichos administradores son los siguientes:

- *Manager* (administrador), el cual controla la construcción de reproductores (*Players*), procesadores (*Processors*), fuentes de datos (*DataSources*) y repositorios de datos (*DataSinks*). Este control indirecto permite que se construyan nuevos objetos (siguiendo los lineamentos indicados) que se integran de forma articulada con *JMF*.
- *PackageManager*, que tiene un registro de todos los paquetes que contienen clases de *JMF*.
- *CaptureDeviceManager*, el cual contiene una lista de los dispositivos de captura disponibles.
- *PlugInManager*, que registra los componentes de procesamiento de tipo *plug-in JMF*, tales como multiplexores, demultiplexores, *códecs*, filtros de efecto y medios de despliegue.

Si se quiere construir una aplicación que use *JMF*, se necesita invocar a los métodos de creación de *Manager* para poder crear los reproductores, procesadores, fuentes de datos y pilas de datos; de esta forma la aplicación los puede usar, a la vez que son reconocidos por la *JMF*.

De forma similar, si se requiere capturar datos de media de un dispositivo de entrada, se requiere que se use *CaptureDeviceManager* para obtener los dispositivos de entrada disponibles, así como la información necesaria para tener acceso a ellos.

Si se requiere saber que procesos manipulan los datos, se usa la *PlugInManager* para consultar los *plug-in* instalados; también se usa para registrar nuevos *plug-in*. Si se construyen nuevos reproductores, procesadores, fuentes de datos o repositorios de datos a la medida, se requiere que se registre el nuevo (y único) paquete que los contiene con *PackageManager*.

El modelo de datos de *JMF* consiste en que los reproductores de *JMF* generalmente usan objetos de tipo *DataSources* para controlar la transferencia de media; estos últimos encapsulan la localización de la media, así como el protocolo y la aplicación usados para entregar la media. Una vez obtenido, un *DataSource* no puede re-usarse para obtener otro tipo de media.

Un objeto *DataSource* se identifica por un *MediaLocator* de *JMF* o por una *URL* (localizador de recursos universal, *Universal Resource Locator*); estos dos últimos son similares, pero *MediaLocator* se distingue porque se puede construir sin que el controlador del protocolo esté instalado en el sistema operativo, mientras que un objeto de tipo *URL* sólo puede ser construido si su controlador correspondiente está instalado en el sistema.

Un *DataSource* administra un conjunto de objetos *SourceStream*; una fuente de datos estándar usa un arreglo de bytes como unidad de transferencia, mientras que una fuente de datos intermedia (*buffer*) usa un objeto de tipo *Buffer* como unidad de transferencia. *JMF* define varias clases de tipo *DataSource*.

Esta API es explicada con más detalle en el **Apéndice F**.

1.5.6 *Apache-Tomcat.*

Apache-Tomcat es un contenedor de ***servlets*** que también le da soporte a ***JSPs***; aunque no es un servidor de aplicaciones web, sirve como referencia a los mismos, indicando qué versión de ***servlets*** y ***JSPs*** acepta, y la forma en que los interpreta.

La estructura de directorios de ***Apache-Tomcat*** consiste en sub-directorios contenidos dentro del directorio de instalación de Tomcat, que agrupan distintos archivos por funcionalidad de los mismos. Esos sub-directorios, y los archivos más importantes se mencionan a continuación.

- ***bin***: Contiene los archivos de procesamiento por lotes, archivos con secuencias de comando del núcleo, y un precompilador de páginas ***JSP*** que sirve para mejorar el tiempo de respuesta de cuando se solicita por primera vez una página ***JSP***. En la instalación binaria se encuentran *tomcat6.exe* y *tomcat6w.exe*, los cuales son los archivos binarios con los que da servicio; en la versión sin instalador de *Windows* (la versión comprimida), se encuentran los archivos *startup.bat* y *shutdown.bat*, los cuales levantan y tiran el servidor.

- ***conf***: Contiene los archivos descriptores y de propiedades con que se configura *Tomcat*. Entre ellos se encuentran:
  - *server.xml*, el cual es el archivo de configuración principal, siendo leído cuando inicia la

aplicación. En las versiones 4.x y anteriores contenía la configuración de las páginas web, pero a partir de las versiones 5.x en adelante, ya no tiene información específica de las aplicaciones web.
- *Context.xml*, Contiene especificaciones que comparten todas las aplicaciones web.
- *web.xml*, el cual el archivo descriptor predeterminado para las aplicaciones web; si una aplicación web carece de descriptor, usa este archivo como descriptor.
- *tomcat-users.xml*, el cual tiene usuarios, contraseñas y funciones, con los cuales se crea un reino de seguridad para la autenticación y administración del *Tomcat*.
- *catalina.policy*, el cual contiene permisos con que el lenguaje java autoriza o rechaza el acceso de *Tomcat* a los recursos del equipo.
- *catalina.properties*, que almacena pares de nombres de propiedades y sus valores usados para el acceso a paquetes internos y de control de definición de *Tomcat*; se lee cuando se levanta la aplicación.

- ***lib***: Contiene archivos *JAR*, que funcionan como librerías, las cuales pueden ser usadas por *Tomcat*, o por cualquier aplicación web contenida en él.
- ***logs***: Contiene diversas bitácoras del *Apache-Tomcat*.
- ***temp***: Contiene archivos temporales.
- ***webapps***: Contiene las aplicaciones web; el sistema desarrollado reside en el sub-directorio ***videocamara***.
- ***works***: Contiene archivos temporales de las aplicaciones web, ***JSPs*** precompilados de las mismas, así como otros archivos que funcionan como memoria intermedia (***buffer***).

1.6 Base de datos relacionales.

Una base de datos es una colección organizada de información. La base de datos relacional fue propuesta en 1969 por el Dr. Edgar F. Codd, con el fin de presentar un modelo de datos que pudiera manejar grandes cantidades de datos y resolver problemas de consistencia e integridad de datos.

Las bases de datos relacionales son modelos de datos que consisten en un conjunto de tablas, y las interacciones entre las mismas.

Cada tabla es el modelo de una Entidad, la cual tiene existencia propia y tiene características de interés llamados atributos, los cuales requieren ser guardados; una instancia es un caso particular de una entidad, con valores propios que le permiten distinguirse de otras instancias de la misma entidad. Una entidad dependiente es aquella cuya existencia depende de otra; es decir, si se elimina la entidad independiente, las que dependan de ella dejan de existir o de tener sentido.

La tabla se forma como un arreglo de columnas -existe una columna por cada atributo, donde se indica el tipo de dato que corresponde al atributo- y renglones (llamados ***tuplas*** o registros), que almacenan los datos de una entidad en particular (instancias); dentro de cada renglón existe un valor único que permite identificarlo. Cada elemento formado por la intersección de un registro con una columna se le conoce como campo. Los valores almacenados por cada campo son escalales, es decir, sólo tienen un valor.

Las relaciones entre las tablas modelan las asociaciones de las distintas entidades entre sí.

En las tablas pueden existir algunas columnas con valores especiales que sirven como identificadores, las cuales pueden ser alguna de las siguientes:

- Llave primaria, tiene un valor único que permite identificar a una instancia de una entidad en particular.
- Llave primaria compuesta, la cual se forma con los valores de dos o más columnas, y con los cuales se identifica a una instancia de una entidad.
- Llave secundaria o llave alterna, es un atributo que también puede servir para identificar a las instancias de una entidad.
- Llave foránea, son valores que representan la llave primaria de otra tabla, y por medio de los cuales se puede relacionar ambas tablas.

1.6.1 Ventajas de las bases de datos relacionales.

Las bases de datos relacionales no sólo resuelven problemas que presentan otros modelos de bases de datos, si que además tienen las siguientes ventajas:

- Mantiene la integridad de los datos, manteniendo la validez de los datos en el ámbito de campo, y en el ámbito de tabla evita que se dupliquen registros o que falten llaves privadas. En el ámbito de relaciones se asegura que las relaciones entre las tablas sean válidas.
- Mantiene la independencia lógica y física de los datos respecto a las aplicaciones; los cambios en la aplicación cliente o del administrador de la base de datos no implican modificaciones a la estructura de la base de datos.
- Mantiene la consistencia y certeza de los datos.
- Permite el fácil acceso a los datos; el usuario puede recuperar datos de una tabla, o de varias tablas relacionadas, para que obtenga la información de muchas formas.

1.6.2 El administrador de base de datos relacionales.

El administrador de la base de datos (*Relational Database Management System*, *RDBMS*) es una aplicación (*software*) que sirve para crear, mantener, modificar y manipular bases de datos relacionales; efectúan operaciones de bajo nivel, manipulando archivos e inter actuando con el sistema operativo. Esas aplicaciones aparecieron a principios de los años 70s, fabricados por diversas compañías y para diversos sistemas operativos; desde entones, han aparecido todavía más administradores de base de datos, tanto para los sistemas operativos ya existentes como para los nuevos, en distintas versiones (ligeras, completas, empresariales) y en distintos esquemas de licenciamiento (pago por licencia, gratuitas, *open source*, etcétera).

Dentro de esas aplicaciones se destacan *Sysbase*, *DB2* e *Informix* de *IBM*, *Oracle*, *SQL Server* y *Access* de Microsoft, *Paradox*, *MySQL*, *DB* (de Sun Microsystems), entre tantos otros. Dichas aplicaciones tienen un mayor o menor grado de apego al modelo relacional, e incluso, algunas han ayudado a definir la implantación del modelo relacional.

Durante la primera parte de la década de los años 80, con la popularización de las computadoras personales, la administración de bases de datos pasó de los *mainframes* a aplicaciones desarrolladas para *PCs*. Posteriormente, contribuyeron al desarrollo de la arquitectura cliente/servidor, al proponer un servidor de base de datos que atendiera a varias aplicaciones cliente.

La importancia de las bases de datos en la actualizad es muy grande; contribuyeron al desarrollo de aplicaciones de escritorio que permiten a los usuarios finales manipular información fácil, rápida y fiablemente. Permiten almacenan grandes cantidades de información. Posteriormente, de la segunda parte de la década de los años 90 a la fecha, al difundirse el uso de Internet, surgieron aplicaciones *web*, las cuales usan como aplicación cliente a los navegadores, por lo que se ahorra la creación de aplicaciones cliente, y el esfuerzo de actualizaros; esto

permitió que aumentaran los clientes de tales aplicaciones, y ampliar enormemente el ámbito de acceso de las aplicaciones.

1.6.2.1 Recuperación de los datos.

Para poder guardar, recuperar o modificar los datos de una base de datos relacional, IBM propuso el Lenguaje de Consultas Estructurado (*Structured Query Language*, *SQL*) a principios de los años 70; más adelante, se formalizó con el estándar *ANSI SQL-92*. La mayoría de los administradores de bases de datos actuales utilizan este lenguaje, y varios de ellos tienen instrucciones adicionales, para brindarle mayor funcionalidad al usuario. *SQL* tiene los siguientes tipos de instrucciones:

- ***DDL*** (*Data Definition Language*), son instrucciones para crear bases de datos, tablas, otros objetos de la base, etcétera.
- ***DML*** (*Data Manipulation Language*), las cuales son instrucciones para seleccionar, insertar, modificar o borrar datos.

Otro lenguaje de consulta es ***QBE*** (*Query By Example*), el cual permite escoger tablas y columnas de forma gráfica; su ventaja es que es intuitivo, pero tiene la desventaja de que no todas las plataformas tienen un ambiente gráfico; en muchos casos, sólo los programas propietarios de un administrador de base de datos pueden interpretar este lenguaje, y para que se utilice en otras aplicaciones es necesario traducir su consulta a ***SQL***.

Estos lenguajes de consulta emplean álgebra relacional, la cual relaciona las entidades (vistas como conjuntos) mediante operaciones como unión, intersección, etcétera.

1.6.3 Normalización.

En párrafos anteriores ya se había mencionado que se construyen tablas, una por entidad. Para determinar exactamente cuántas tablas se construyen, e indicar por tabla cuántas columnas para guardar los atributos, se sigue un proceso llamado normalización, con el cual se organiza la estructura de la información y se controla la redundancia. Se inicia con la primera forma normal, y para el diseño de base de datos se sigue hasta la tercera forma normal, generalmente (existe hasta la sexta forma normal). Las tres primeras formas normales se explican en los párrafos subsecuentes.

1.6.3.1 Primera forma normal.

En esta forma se considera que los datos similares corresponden a una misma entidad, y para cada entidad se construye una tabla. Para cada entidad se agregan un atributo por cada tipo de dato de interés o relevante para el problema; cada atributo de la entidad corresponde a una columna de la tabla a construir. El objetivo en este paso es eliminar atributos repetidos, o derivados (que se obtienen a partir de otro atributo). Si existen grupos de atributos similares, se separan para crear otra entidad.

Para identificar a una entidad en particular, se escoge un atributo que sirva como identificador, y su columna correspondiente se denomina llave primaria; si para identificar a una entidad se requieren dos o más atributos, tales atributos forman una llave primaria compuesta. En caso de que no existan atributos que permita distinguir inequívocamente a cada entidad, se inventa uno de forma arbitraria (por ejemplo, un auto-incremento), para que sea la llave primaria.

1.6.3.2 Segunda forma normal.

Para pasar a esta forma, se revisan los atributos de cada entidad, para buscar atributos que dependan parcialmente de la llave primaria; en caso de que existan tales atributos, se quitan de la entidad y se agregan en una nueva entidad. Se agrega la relación entre la entidad ya existente con la nueva, para asociarlas.

1.6.3.3 Tercera forma normal.

En esta forma, cada entidad se revisa, para asegura que todos sus atributos dependan completamente de la llave primaria; en caso de que existan atributos que no sean de ese tipo, deberán ser extraídos de esa entidad para formar una nueva entidad, indicando la relación adecuada entre ambas entidades.

1.6.3.4 Forma normal *Boyce-Codd*.

Es un caso especial de la tercera forma normal, y muchas veces se confunde con la misma. Una entidad está en la forma normal *Boyce-Codd*, si cumple con la tercera forma normal, y todos los determinantes de la relación son candidatos a ser llaves primarias (no hay dependencia funcional entre las llaves candidato). Un determinante es uno o varios atributos con los que se puede encontrar los otros atributos de una misma relación [28].

1.6.3.5 Cuarta forma normal.

La cuarta forma normal es aquella en la que cumple con la forma normal ***Boyce-Codd***, y todas las dependencias multi-valuadas han sido resueltas de forma independiente. Una dependencia multi-valuada es aquella donde para cada valor de un atributo 'A', se relaciona con un conjunto finito de valores del atributo 'B', y también se relaciona con otro conjunto finito de valores de un atributo 'C'; los atributos 'B' y 'C' son independientes entre sí. Esta relación se descompone creando una entidad para 'B' y otra para 'C', indicando la asociación entre 'B' y 'A', y la de 'C' con 'A' [28].

1.6.3.6 Quinta forma normal (Unión-Proyección, *Union-Projection*).

La quinta forma normal indica que se deben resolver las dependencias agrupadas (*join dependencies*); este tipo de relación es extremadamente raro, e indica la restricción cíclica donde, por ejemplo, la entidad 'A' se relaciona con la entidad 'B', 'B' se relaciona con 'C', y a su vez, se relaciona con 'A', por lo que se obtienen ***tuplas*** donde necesariamente se encuentran las todas las entidades que participan en esa relación cíclica [28].

Se utilizan aplicaciones de análisis automático para determinar si un modelo ha llegado a esta forma normal.

1.6.3.7 Llave-dominio forma normal (*DK/NF*).

Es una forma normal más estricta que la quinta, en la que se garantiza que no existirán anomalías de actualización de datos. Una relación está en esta forma normal si cada una de sus restricciones es una consecuencia lógica de la definición de llaves y dominios. Un dominio es un conjunto de valores permitidos de un atributo, mientras que la llave (primaria) es la que permite identificar unívocamente a una ***tupla***. Las restricciones son aquellas reglas que son suficientemente precisas para determinar si es cierta o falsa la relación indicada [47].

1.6.3.8 Sexta forma normal.

Es una forma normal propuesta recientemente, en la cual se agregan objetos temporales a las llaves primarias, lo que permite hacer reportes históricos y análisis de datos respecto a marcos temporales [45].

1.6.4 Relaciones entre las Entidades.

Como ya se había mencionado, se debe indicar las relaciones entre las entidades, para mostrar cómo se asocian, así como la cardinalidad de las relaciones - el número de entidades de un mismo tipo con las que se puede asociar una entidad de otro tipo. Al momento de construir las tablas, las relaciones se convertirán en nuevas columnas donde se guardarán las llaves foráneas (las que son llaves primarias en otras tablas). También se indica si las relaciones son obligatorias u opcionales. Estas asociaciones pueden ser las siguientes:

- Uno a uno, donde sólo una entidad de tipo A se relaciona con una entidad de tipo B.
- Uno a cero o más, donde una entidad de tipo A puede no tener ninguna relación con entidades de tipo B, o estar asociada con una o más de ellas.
- Uno a n o más, donde una entidad de tipo A se relaciona con n, donde n es un número natural, o más entidades de tipo B.
- Muchos a muchos, donde una entidad de tipo A se relaciona con muchas entidades de tipo B, y viceversa. Este tipo de asociación se descompone posteriormente con una nueva tabla, cuyas columnas son las llaves públicas de ambas tablas.

1.6.5 Modelo Entidad-Relación de *Case*Method*.

El modelo Entidad - Relación de *Case*Method* sirve para diseñar una base de datos relacional con elementos gráficos, ya que de esa forma se puede mostrar de una forma fácil e intuitivamente las entidades y sus relaciones. A partir del modelo se definen las tablas a construir, con sus respectivas columnas indicando el tipo de dato. Finalmente se crea la base y las tablas, junto con otros elementos de la base de datos, mediante instrucciones ***DDL***.

Esta metodología fue diseñada por *Oracle*, y aunque se carece de herramientas ***CASE*** para diseñar y construir las tablas de forma automática, se decidió por utilizarla, ya que permite diseñar una base de datos relacional, resolviendo correctamente las asociaciones entre las entidades, sin importar el administrador de base de datos empleado. Esta metodología tiene la ventaja sobre el modelo de Chen, en que permite identificar y aclarar relaciones complejas entre las entidades.

Una vez terminado el modelo, se diseñaron las tablas indicando el tipo de dato por columna, y finalmente se hizo un guión para construir las tablas en *MySQL*, insertando los datos de las tablas que fungen como catálogos (aquellas cuyos datos rara vez cambian). A continuación se mostrarán los elementos gráficos de diseño, explicando sus convenciones.

1.6.5.1 Entidades.

Las entidades se representan con un rectángulo con puntas redondeadas, como se muestra a continuación:

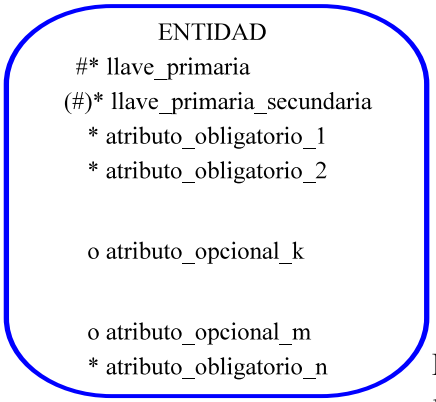

Figura 16: *Ejemplo de modelado de una Entidad.*

Para modelar la entidad y sus atributos, se siguen las siguientes reglas:

- El nombre de la entidad va en mayúsculas y en singular, en la parte superior.
- Si la entidad pudiera tener otro nombre distinto, se indica ese nombre entre paréntesis debajo del primer nombre de la entidad; existe otra convención que es utilizar una diagonal para separar este nombre del primero.
- Los atributos de la entidad van a continuación, en letras minúsculas, indicando el tipo de atributo del lado izquierdo -llave primaria, requerido u obligatorio.
- El primer atributo que aparece es la llave primaria, la cual se indica con el símbolo '#'.
- Si hubiera otro atributo que también pudiera ser identificador de la entidad, se indica con los caracteres '(#)'.
- Los atributos obligatorios, requeridos siempre para cada entidad, se representan con el símbolo '*'.
- Los atributos opcionales se representan con el símbolo 'o'.

En caso de que hubiera entidades del mismo tipo, se pueden modelar de la siguiente forma:

Figura 17: *Ejemplo de modelado de una superentidad, y en su interior, sus subentidades.*

Las entidades que van en el interior se llaman subentidades; la entidad externa se llama superentidad. Los atributos de la superentidad los comparte con las subentidades, las cuales pueden tener o no atributos propios; las relaciones pueden ir hacia la superentidad o hacia una subentidad en particular.

Si un atributo tiene tanta importancia, que de él surgen atributos y relaciones, debe convertirse en una entidad.

En el caso de una relación muchos a muchos (la cual se explicará más adelante), la relación se descompone por medio de una entidad de intersección, la cual tendrá como atributos las llaves primarias de las entidades de la relación original, y sus relaciones con cada una de ellas indicarán dependencia (lo cual se mostrará más adelante).

1.6.5.2 Relaciones.

Las relaciones indican cómo se asocian las entidades entre sí, indicando su dependencia, y cómo inter actúan entre sí.

Las relaciones se indican mediante líneas (recomendándose que éstas sean sólo horizontales o verticales), para unir sólo dos entidades (relación binaria), o una entidad así misma (relación recursiva). Las relaciones deberán tener, en cada extremo, lo siguiente:

- Un nombre que describa la relación en ese sentido.
- La cardinalidad o grado de la relación (a cuántos elementos se asocia).
- La opcionalidad, que indica si la relación es obligatoria u opcional.

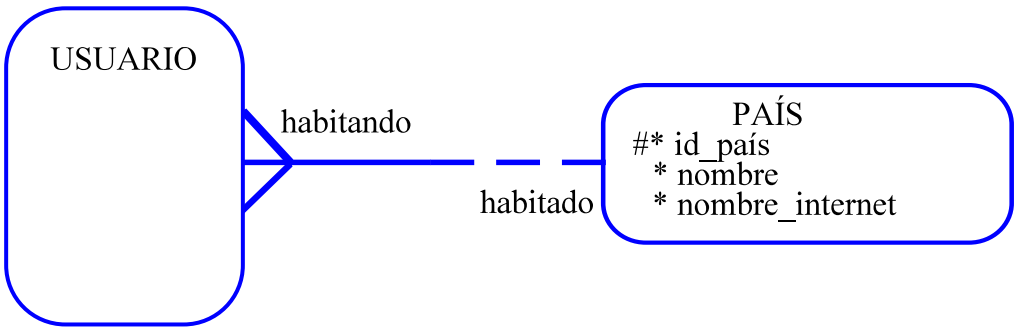

Figura 18: *Relación entre entidades.*

Una relación debe ser interpretada yendo de una entidad a la otra, tomando los tres factores anteriores en un mismo sentido (por ejemplo, de USUARIO a PAÍS). La contraparte de la relación anterior, es la que se interpreta en el otro sentido (de PAÍS a USUARIO).

1.6.5.2.1 Nombres de las relaciones.

Como ya se había mencionado, los nombres de las relaciones deben ser descriptivos; en este modelo se debe formar una oración de la siguiente forma:

Cada ***Entidad1*** [verbo ser/estar o poder estar] ***nombre_relación*** *cardinalidad* ***Entidad2***.

La oración formada debe ser lógica, e indica también la cardinalidad y la obligatoriedad de las relaciones. También se debe apreciar, tanto en el modelo como en las frases de la relación redactadas, que la relación sólo une dos entidades.

1.6.5.2.2 Cardinalidad.

Esta propiedad indica el número de entidades a las cuales se puede asociar una entidad de otro tipo, en una relación. Si la relación, yendo de la entidad *A* a la *B*, acaba en una línea simple, indica que una entidad de tipo *A* sólo se relaciona con una entidad de tipo *B*. Si de la línea de la relación se desprenden dos líneas al final de la misma (comúnmente llamadas 'patas de gallo'), indica que la entidad *A* se relaciona con una o más de la entidad *B* (relación múltiple), tal y como se muestra en el siguiente ejemplo:

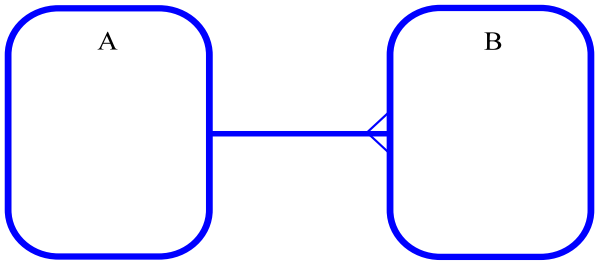

Figura 19: *Cardinalidad entre dos entidades.*

Como complemento a lo anterior, la relación de *B* a *A* indica que una entidad de tipo *B* sólo se relaciona con una entidad de tipo *A*.

Si se requiere, se pude indicar una restricción en la relación múltiple, proporcionando un número que indique la cota, y los signos de =, <, >, ≤ o ≥.

1.6.5.2.3 Opcionalidad.

Este atributo indica si la relación de una entidad a la otra es obligatoria u opcional. Si la relación se indica con una línea continua, dicha asociación es obligatoria; por el contrario, si se tiene una línea discontinua, la relación es opcional, tal y como se muestra en el siguiente ejemplo:

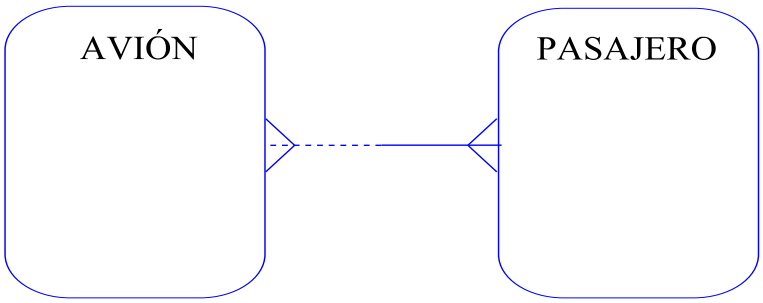

Figura 20: *Opcionalidad de la relación entre dos entidades.*

Como se había mencionado anteriormente, existen relaciones muchos a muchos, la cual se descompone -para aclarar la relación y hacer tanto la construcción como las consultas a tablas más fácilmente- mediante una entidad

de intersección.

Una entidad de intersección puede tener atributos propios. A la hora de construir la tabla, se colocan las columnas correspondientes a las llaves primarias de las entidades independientes, más los nuevos atributos que pudiera tener esta entidad.

1.6.5.2.3 Otras relaciones.

A continuación se mostraran algunas relaciones especiales:

- Relación recursiva: Esta relación es una asociación binaria de una entidad consigo misma; sirve para indicar que existe un papel o función distinta en la asociación de dos instancias de la misma entidad. Se representa por un semicírculo, en el cual también se indica la cardinalidad y la opcionalidad, revisando que la relación sea factible (para evitar relaciones infinitas), como se muestra en el siguiente ejemplo:

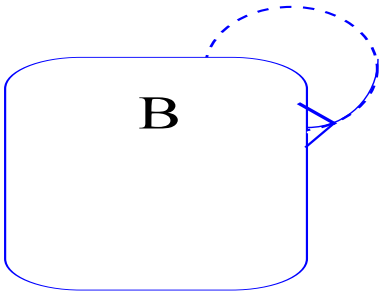

Figura 21: *Relación recursiva.*

- Relación de arco: Indica si una entidad A se asocia con una entidad B o con una C, pero no con ambas. Se indica como un arco que atraviesa las relaciones de A con las entidades con las que está opcionalmente relacionada (pueden ser dos o más); se puede eliminar la ambigüedad de cuáles entidades participan en esta relación agregando un punto donde interseca el arco con las relaciones opcionales.

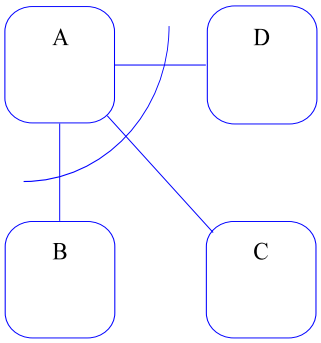

Figura 22: *Relación de arco: La entidad A se debe asociar con B, con C o con D, pero no se permite que A se asocie con una combinación de B, C y/o D.*

- Relaciones no transferibles: Una relación no transferible es una asociación, que una vez formada entre una instancia de *A* a una instancia de *B*, no puede volver a formarse, es decir, la instancia de *A* no puede desasociarse de la instancia original de *B* para asociarse con otra instancia de *B*. Se indica con un rombo en la relación del lado de la entidad que no puede transferir su asociación. Cuando se describe la relación,

se debe agregar a la frase la oración “y no se puede transferir nunca a otra NOMBRE_ENTIDAD_B”.

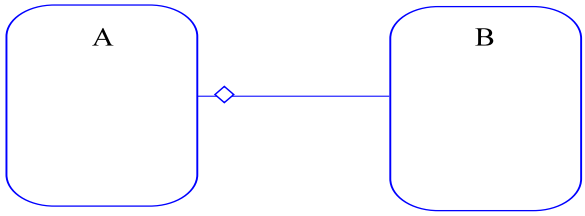

Figura 23: *Relación no transferible: la relación de una instancia de A a una de B no puede ser transferida.*

1.6.5.3 Matriz de relaciones.

Una vez con las entidades modeladas y asociadas, se hace una matriz donde el primer renglón y la primera columna tienen los nombres de todas las entidades; en las demás celdas se anota el nombre de la relación de las entidades que intersecan, o se indica que no hay ninguna, como se muestra a continuación:

	entidad1	**entidad2**	**entidad3**
entidad1	----------	es poseído por una	----------
entidad2	es dueño de una	----------	es generado por
entidad3	----------	puede estar procesando	es supervisado por

Tabla 10: Matriz de relaciones entre entidades.

Como último punto del sub-tema Base de Datos, se deja indicado que después de hacer la matriz de relaciones, sigue la construcción de las tablas, la cual se explicará con más detalle en el sub-tema de **Diseño del Sistema.**