Universidad Nacional Autónoma de México.

Facultad de Ingeniería.

“Programación en Internet con Aplicaciones en Instrumentación Remota.”

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

PRESENTA:

HUGO RUÍZ ZEPEDA

DIRECTOR DE TESIS:

M. I. BENJAMÍN VALERA OROZCO

CIUDAD UNIVERSITARIA. FEBRERO 2010.

Tesis.

Programación en Internet con Aplicaciones en Instrumentación Remota.

Índice.

Introducción.

Los sistemas actuales de video vigilancia son sistemas analógicos no interactivos; algunos de ellos son digitales y que además sólo trabajan con ciertos modelos de videocámaras específicas, con un cliente propietario y mediante el pago para operar dicho sistema. El sistema propuesto permite un sistema interactivo, utilizando un sistema cliente-servidor, donde el cliente es cualquier navegador (*Netscape* o *Internet Explorer*) y el servidor -sobre el cual se instarán los códigos desarrollados- puede ser uno de libre distribución o bajo licencia; también se busca que este sistema controle un sistema mecánico que se pueda adaptar a diversas videocámaras que actualmente carecen de movimiento, para ahorrar a los usuarios el costo de una videocámara que sea móvil.

Por otra parte, se quiere mostrar la forma para manipular otro tipo de dispositivos a través de Internet, proporcionando soluciones para construir sistemas similares que proporcionen mediciones de instrumentos que se encuentren en servidores ubicados en lugares distantes de la computadora donde un operador recaba y/o manipula dicho dispositivo.

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.*

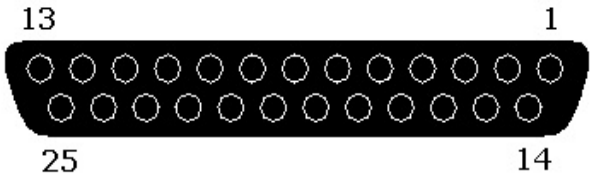


Figura 2: *Numeración de los pernos para el conector DB25.*

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.

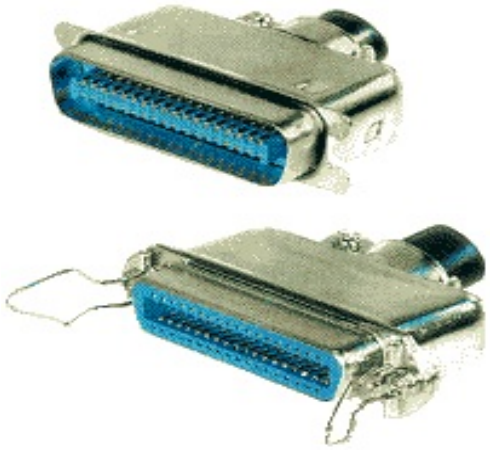


Figura 3: *Conector ‘Centronics’ hembra en la posición superior, y macho en la inferior.*

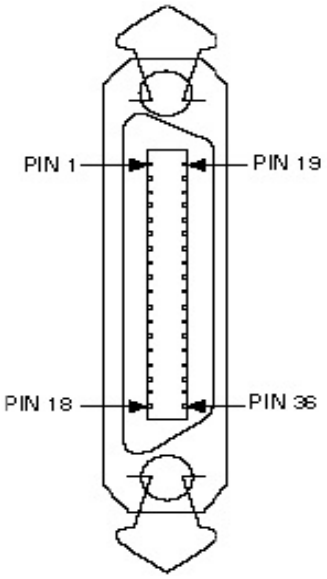


Figura 4: *Numeración de los pernos para el conector 1284-B.*

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.

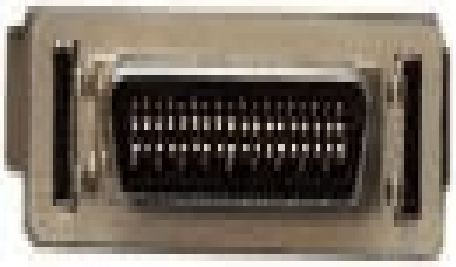


Figura 5: *Conector Mini - centronics macho.*

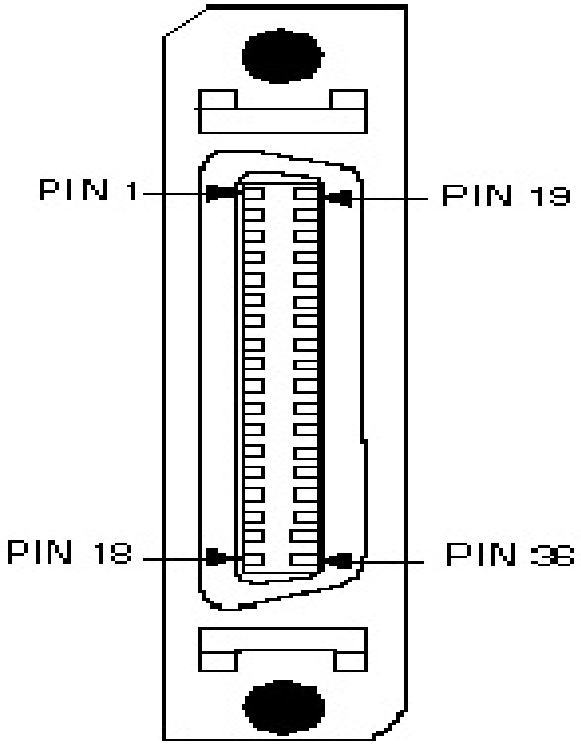


Figura 6: *Numeración de los pernos para el conector 1284-C.*

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*.

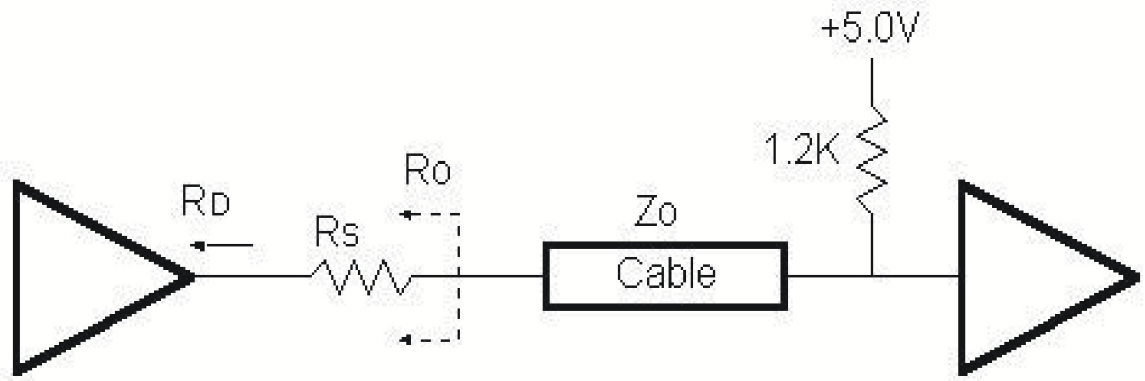


Figura 7: *Conexión emisor/receptor recomendada para evitar diferencia de impedancias.*

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.

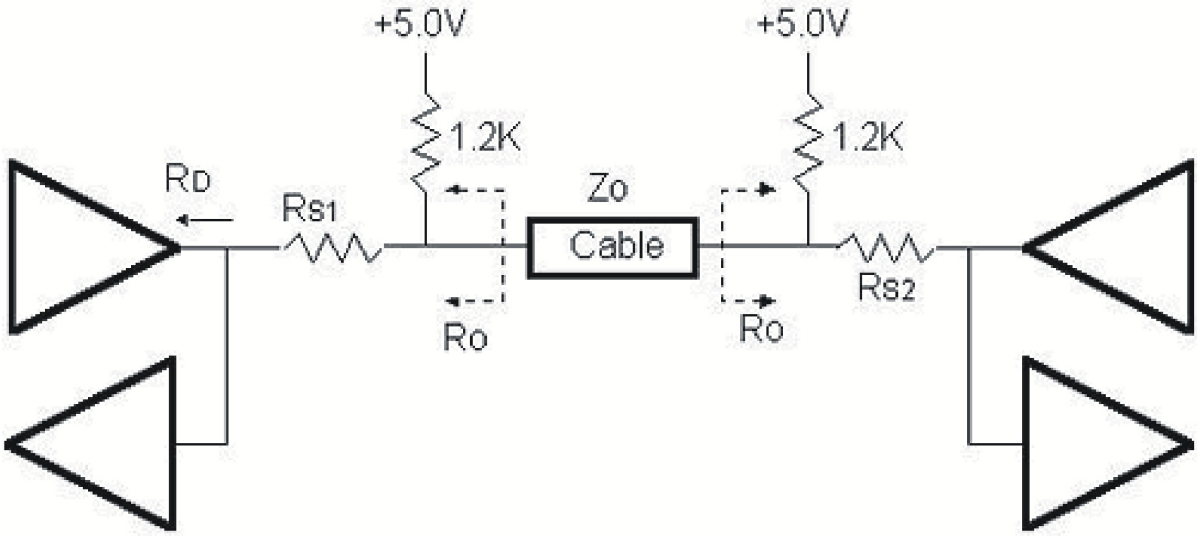


Figura 8: *Circuito recomendado para el par emisor/receptor del nivel II.*

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.

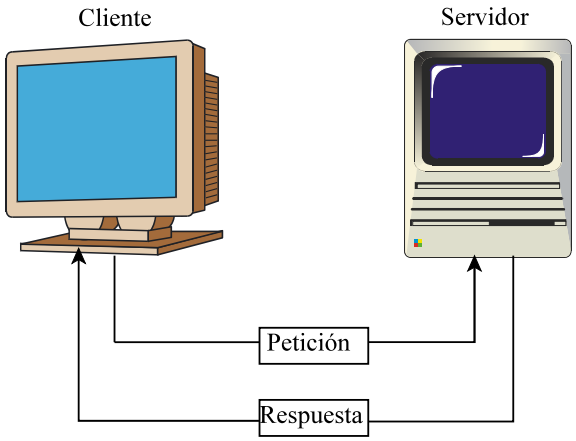


Figura 9: *Transacción HTTP.*

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.

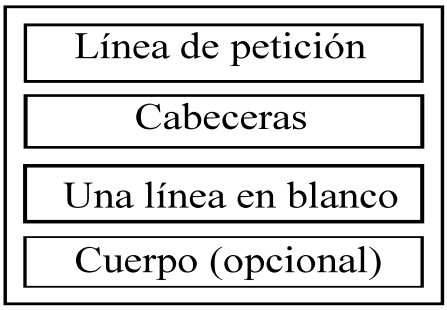


Figura 10: *Estructura del mensaje de petición HTTP.*

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

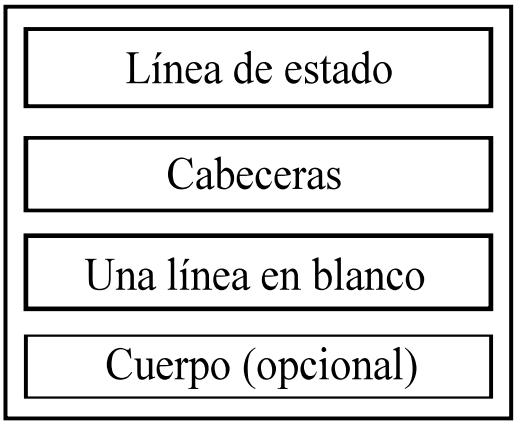


Figura 11: *Estructura del mensaje de respuesta HTTP.*

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.

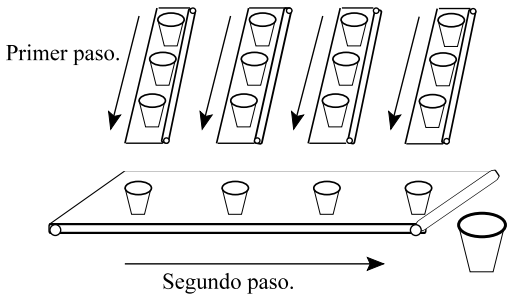


Figura 12: *Principio básico de la operación de un dispositivo CCD.*

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.

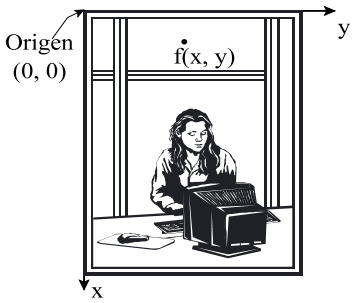


Figura 13: *Convención de los ejes para la representación digital de imágenes.*

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.

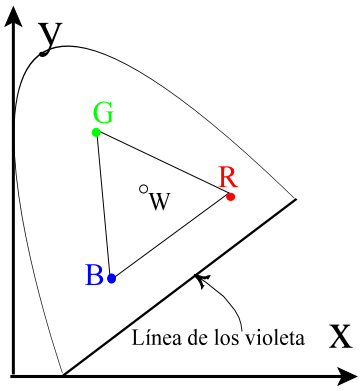


Figura 14: *Triángulo de los colores primarios.*

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.

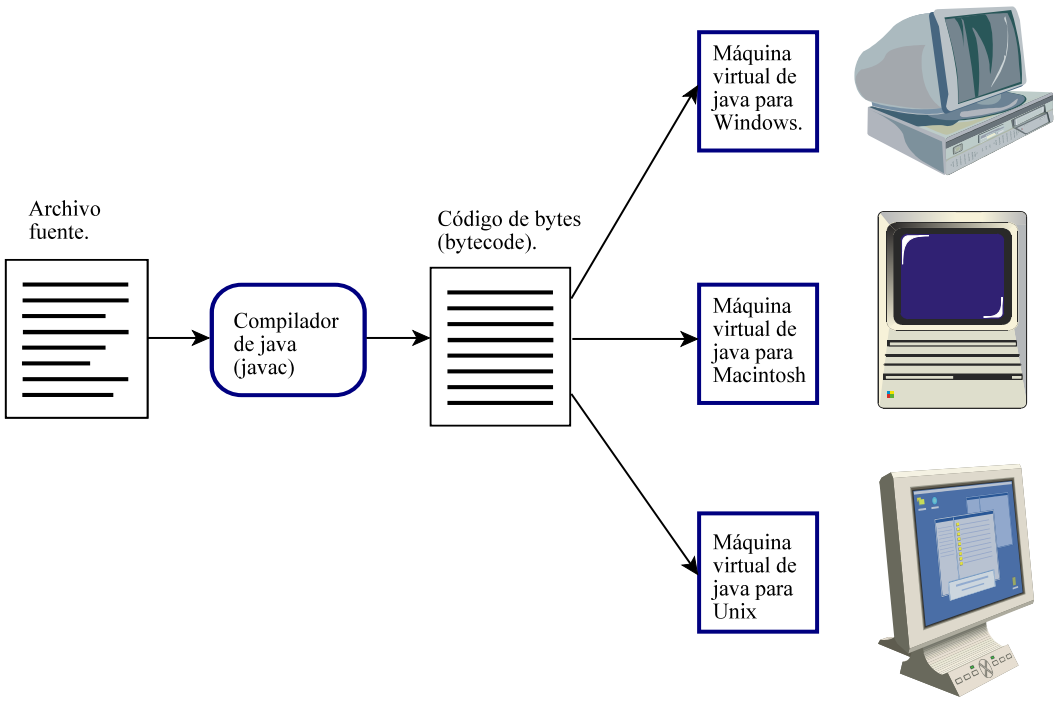


Figura 15: *Independencia del código de la plataforma.*

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:

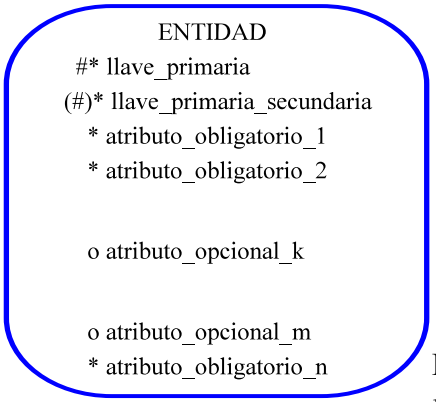


Figura 16: *Ejemplo de modelado de una Entidad.*

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.

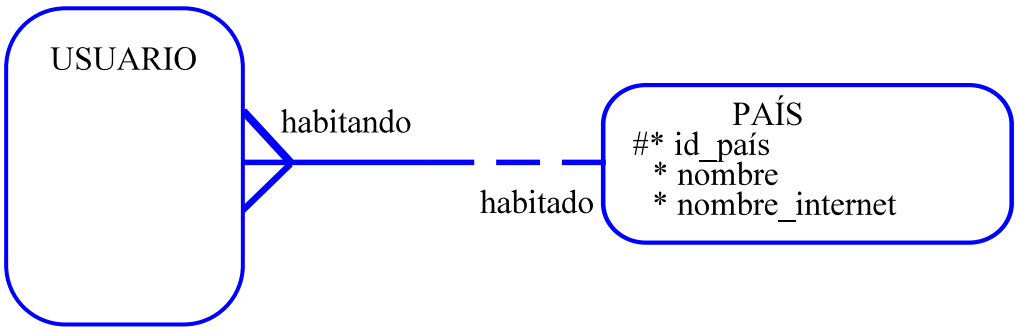


Figura 18: *Relación entre entidades.*

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:

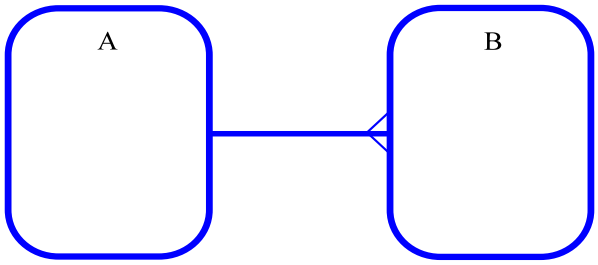


Figura 19: *Cardinalidad entre dos entidades.*

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:

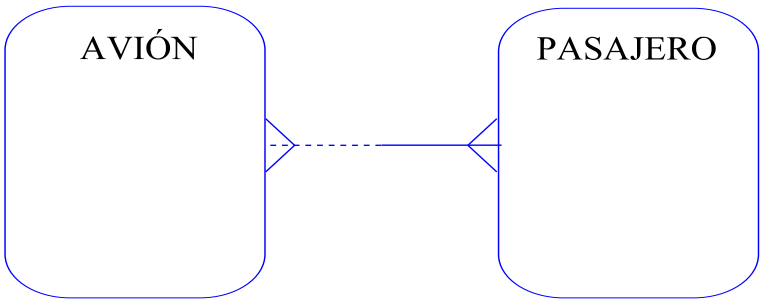


Figura 20: *Opcionalidad de la relación entre dos entidades.*

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:

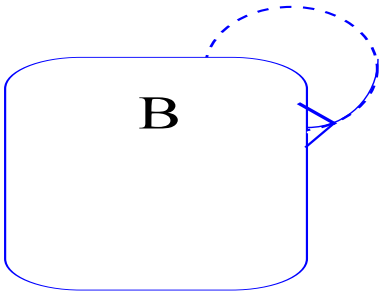


Figura 21: *Relación recursiva.*

- 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.

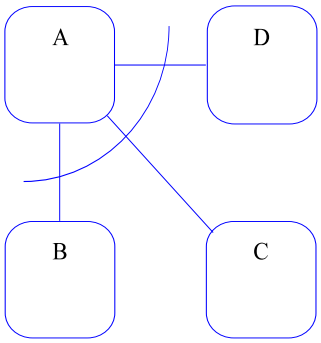


Figura 22: *Relación de arco: La entidad A se debe asociar con B, con C o con D, pero no se permite que A se asocie con una combinación de B, C y/o D.*

- 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”.

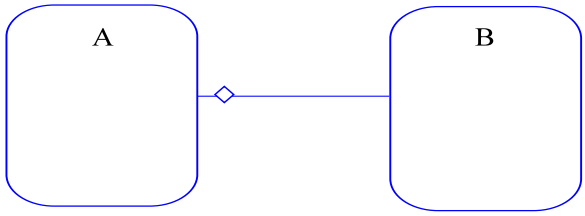


Figura 23: *Relación no transferible: la relación de una instancia de A a una de B no puede ser transferida.*

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.**

2 Análisis y Diseño del Sistema.

De islas cuadradas flotantes y de cámaras movidas en Internet;
del empirismo al razonamiento, y del sueño a la práctica.

2.1 Análisis y Diseño de la Aplicación.

El análisis, diseño, y posteriormente, la construcción, comenzó a partir de recolectar los requerimientos del problema (manipular la videocámara en un ambiente Internet), y comenzar a buscar un lenguaje de programación, así como componentes físicos (para el circuito de control) con los que se pudiera construir una aplicación que pudiera obtener una imagen a voluntad de dicho dispositivo, cambiar la posición física de la videocámara, y obtener una nueva imagen en la nueva ubicación de la videocámara; también se requería guardar información, para consultarla posteriormente. También se buscaron herramientas ***CASE*** y de desarrollo de aplicaciones, a la vez que se buscó algún estándar de análisis, diseño y desarrollo de aplicaciones al cual apegarse.

Dentro de las metodologías para el desarrollo de sistemas de cómputo se encuentran el de desarrollo en cascada, y *RUP* (*Rational Unified Process*). El primero es para crear sistemas a partir de sistemas ya existentes o para automatizar procesos manuales; en ambos casos ya existen reglas claras de los procesos existentes, contando con documentación escrita acerca de los mismos. La otra metodología, *RUP*, fue desarrollada por *Rational Software*, para crear sistemas donde no existan tales reglas, y se tenga que actualizar el proceso de desarrollo y adaptarse a cambios; esta metodología no está desarrollada para un lenguaje de cómputo específico, pero recomienda el uso herramientas para automatizar el desarrollo de sistemas (sobre todo los productos de *Rational Software*). Para las fases de creación de este sistema de cómputo no se adquirieron aplicaciones para seguir esta metodología, sino que se siguen lo que llaman las mejores prácticas, que son reglas que surgieron al estudiar cómo se construyeron exitosamente sistemas de cómputo. Dichas reglas son:

1 Desarrollar los sistemas de cómputo (*software*) iterativamente.
2 Administrar requerimientos.
3 Utilizar arquitecturas que tengan base en componentes.
4 Desarrollar la aplicación con base en modelo visual.
5 Verificar la calidad del sistema.
6 Tener un control de cambios de los códigos fuente.

Para las fases de análisis, diseño, desarrollo y mantenimiento del sistema, se procuró seguir las reglas de *RUP*.

2.1.1 Análisis de la aplicación.

El análisis comenzó por buscar un lenguaje o lenguajes de programación con el que se pudiera programar y construir la aplicación, teniendo en cuenta que dicha aplicación o sistema tendría que unir elementos muy heterogéneos entre sí, tales como los puertos de comunicación de una computadora personal y la construcción de páginas de web; también se buscó que dicho lenguaje fuera orientado a objetos, para aprender desarrollar una aplicación con ese paradigma de programación (que ha reemplazado en gran medida a la programación estructurada), y aprovechar la experiencia adquirida en otros proyectos.

Otro factor que influyó en la selección del lenguaje fue el económico, descartándose aquellos lenguajes de programación que cobraran licencia por su uso, o por el servidor de aplicaciones web. Se analizaron algunos

lenguajes de programación, los cuales se comparan en la siguiente tabla:

Lenguaje	Ventajas:	Desventajas:
'*C*'.	Existen varios compiladores gratuitos; puede tener interacción de forma directa con la memoria reservada para periféricos, y permite hacer llamadas a rutinas en lenguaje ensamblador. Se le considera un lenguaje rápido y eficiente. Existen compiladores para diversos sistemas operativos, tanto gratuitos como de licencia.	Se pueden hacer páginas de web que llamen a código binario mediante *CGI*, pero por cada llamada a estas páginas se genera un proceso, relativamente "pesado", porque por cada llamada a librerías, incorpora el código binario de las mismas al ejecutable que se genere. No es un lenguaje orientado a objetos, por lo que no se podría re-usar código fuente. Los apuntadores mal empleados pueden causar graves problemas. Se evaluó que las librerías para video y para bases de datos se tendían que comprar y/o conseguir de terceros. El código fuente no es totalmente portable entre sistemas operativos; tiene más diferencias entre funciones y librerías mientras el código fuente se relacione con componentes físicos y/o llamadas al sistema operativo.
C++	Es la versión orientada a objetos de '*C*', lo que lo hace más poderoso. Puede manejar la memoria reservada a periféricos, lo que le permite tener interacción con los componentes físicos de la computadora, por lo que diversas compañías que construyen periféricos hacen aplicaciones en ese lenguaje, y algunas crean librerías de aplicaciones (generalmente para *Windows*). Existen algunos compiladores gratuitos, pero los ambientes gráficos de programación son de licencia.	Permite la herencia múltiple, lo cual puede hacer el código confuso. El uso descuidado o malintencionado de los apuntadores pueden hacer un sistema inestable, o afectarlo negativamente. También, por sólo incluir el nombre de librerías (aunque no se usen) hace que se incorpore su código binario al ejecutable que se genere. Se puede construir páginas web dinámicas mediante *CGI*, para que se invoque un ejecutable; por cada vez que se llame, se genera un proceso, el cual reserva cierta cantidad de recursos de la computadora para su uso exclusivo, pero múltiples llamadas pueden agotar los recursos de cómputo del servidor.
PHP	Es un lenguaje de ejecución de comandos de guiones, diseñado para crear páginas web dinámicas, el cual puede tener interacción con muchas bases de datos, y tiene soporte en varios sistemas operativos, por lo que el código desarrollado no necesita ser adaptado al cambiarse a otro sistema operativo. Es de código abierto.	No es un lenguaje que tenga todas las características de Orientación a Objetos. Permite el manejo de *CGI* para ejecutar código binario, pero por sí solo no tiene la capacidad para ejecutar código para el control de los periféricos.

ASP.	Es un lenguaje de ejecución de guiones (del lado del servidor) para crear páginas web dinámicas, creado por Microsoft para *Windows*. Usa los lenguajes de programación *Visual Basic*, *J#* y *C#*. *C#* es la versión de *C++* de Microsoft; *J#* es la versión Microsoft de Java - la cual no cumple con las especificaciones de *Sun Microsystems*, los creadores del lenguaje java. *Visual Basic* es un lenguaje para crear aplicaciones de escritorio en *Windows*. Antes de la plataforma *.NET*, cobraban licencia por su uso. Usa *ODBC* para usar bases de datos, pero varias librerías *ODBC* cobran licencia.	Está limitado a la plataforma *Windows*, pero se había contemplado que en un futuro este sistema, o algunas partes del mismo fueran migradas a *Linux* o *Unix*; esta tecnología se usa en el *IIS* (*Internet Information Server*), el cual cobraba licencia, por lo que se descartó para programar el sistema (actualmente es gratuito en los servidores *Windows* y en las versiones *Windows 2000* y *XP Professional*). Las aplicaciones en estos lenguajes frecuentemente tienen problemas de estabilidad, y no tienen el mismo rendimiento las aplicaciones programadas en lenguajes similares.
Dephi	Es un entorno gráfico de programación cuyo lenguaje era Pascal orientado a objetos, y actualmente tiene su propio lenguaje *Delphi Win32*; aunque fue ideado para crear aplicaciones de escritorio, Pascal también puede tener acceso a la memoria de los periféricos, y hacer rutinas de bajo nivel para tener interacción con ellos.	Permite crear páginas web dinámicas, usado *CGI*, que llama a librerías compartidas (*dlls*) para ejecutar el código, pero el tiempo de respuesta es muy grande. Cobra licencia por su uso, y se limita a la plataforma *Windows*.
Ruby	Es un lenguaje orientado a objetos gratuito y de código abierto. Permite incorporar extensiones programadas en '*C*' fácilmente. Las librerías que tiene son creadas y mantenidas por grupos de programadores.	Fue liberado en 1995, pero inicialmente no tuvo respaldo de grandes compañías de Tecnología de Información, y la primera documentación de este lenguaje estaba en japonés. Es a partir de 2006 cuando tiene mayor difusión.
Java	Es un lenguaje propiedad de *Sun Microsystems*, de uso gratuito. Es orientado a objetos y multi-plataforma, que ha sido respaldado por muchas empresas. Permite crear fácilmente páginas web dinámicas por medio de sus tecnologías de ***servlets*** y ***JSPs***. Puede usar muchas bases de datos en diversos sistemas operativos. Posee la tecnología ***Java Native Interface*** (***JNI***), con la cual puede llamar a librerías dinámicas. Tiene las ***APIs Java Communications***, ***Media Framework*** y ***JDBC***, para uso de puertos locales, multimedia y base de datos, respectivamente.	Es necesario que esté instalada la *Java Virtual Machine* en la computadora donde se ejecute alguna aplicación de este lenguaje, para que pueda interpretar los *bytecodes*, y ejecutar las instrucciones en el lenguaje máquina del sistema operativo; la interpretación de los *bytecodes* hace más lenta la ejecución del programa, en comparación con otros lenguajes que generan binarios (los cuales son sólo para un sistema operativo).

Tabla 11: Comparación de ventajas y desventajas de lenguajes de cómputo para programar una aplicación web.

Después de comparar ventajas y desventajas de diversos lenguajes de computación -excluyendo a *Ruby*, del cual no se tenía noticia-, se escogió el lenguaje java, ya que parecía que con un sólo lenguaje se podía construir el

sistema y tuviera interacción de forma armoniosa; incluso, *JNI* puede llamar a librerías dinámicas recuperando datos complejos (como resultado de las llamadas a las funciones nativas), si se tuviera que programar alguna parte del sistema en otro lenguaje de cómputo. Otros lenguajes permiten crear páginas web fácilmente, pero no pueden tener interacción con componentes físicos por sí solos -tales como la videocámara de estado sólido y los puertos de comunicación de la computadora; se tendría que usar otro lenguaje de cómputo que genere binarios con el lenguaje máquina, para poder manipular los dispositivos físicos, además de que la comunicación entre los lenguajes sería más limitada.

También influyó en la selección de este lenguaje la capacidad de la interfaz ***Java Media Framework*** (***JMF***); no se iba a usar software de desarrollo *Logitech*, porque limitaría al sistema a que funcionara con videocámaras de esta marca, mientras que ***JMF*** permite llamar a dispositivos multimedia, de distintas marcas, de una forma genérica.

Otro requerimiento del sistema es que se guarde información, para consultarla posteriormente; se decidió que la mejor manera de cumplir con ese objetivo es a través de un administrador de base de datos, ya que permite almacenar información (incluyendo datos binarios), consultarla y/o manipularla. Aunque el tipo principal de información a guardar son datos binarios de las imágenes obtenidas, también se planeó que se pudiera guardar información de otros tipos, incluyendo un registro de usuarios, para hacer un módulo de administración del sistema.

El paso siguiente fue la elección del sistema operativo; en este caso se carecía de acceso a un "*mainframe*", a la vez que tenía más acceso a computadoras con sistema operativo *Windows* instalado (por ser más populares que *Unix* y otros sistemas operativos). Aunque ya había aparecido el sistema operativo *Linux*, se carecía del conocimiento técnico para montar y administrar *Linux* en una computadora personal, por lo que se decidió crear el sistema en una plataforma *Windows*, sin perder de vista que dicho sistema de cómputo pudiera ser migrado (total o parcialmente) a otro sistema operativo, y o aprovechar las experiencias adquiridas en desarrollo del mismo para efectuar dicha migración. El equipo que se tenía para programar el sistema tenía *Windows Millenium* instalado, aunque algunos componentes java ya construidos se habían hecho en *Windows 98*.

Se procedió a escoger el administrador de base de datos. Ya se había escogido el lenguaje con el cual programar, java (que es multi-plataforma), y se tenía la restricción de que se construiría sobre el sistema operativo *Windows*, surgiendo los siguientes requerimientos para el administrador de base de datos:

- Que se pudiera instalar en *Windows*, pero que de preferencia tuviera versiones para otros sistemas operativos.
- Que tuviera librerías ***JDBC*** gratuitas (para poderla usarla desde java), y de preferencia que fueran del tipo 4, que son 100% java, y no requieren de instalación de otro software.
- Debería poder almacenar datos numéricos, cadenas, fechas e información binaria (de preferencia, campos *blob*).
- El administrador debería ser gratuito (por restricciones económicas).
- La base de datos debería correr en una computadora persona, dejando recursos disponibles para ejecutar algunas otras aplicaciones al mismo tiempo (tales como el sistema a desarrollar, procesador de palabras, etcétera).

Después de revisar los requerimientos que debía cumplir la base de datos, se efectuó una comparación entre los tipos de base de datos. El tipo más popular de base de datos son las relacionales -las cuales fueron el último punto de ***Conceptos Básicos***- en las que los datos se guardan de forma separada en columnas (del mismo tipo de dato). El otro tipo de base de datos, más reciente, es la base de datos orientada a objetos, en la cual se guardan colecciones de objetos (los cuales no se dividen); esto ahorra mucho esfuerzo de programación, al evitar que se separe la información y guardar dato por dato en columnas, lo que permite que se integre fácilmente con aplicaciones orientadas a objetos.

Se había encontrado que varios fabricantes de administradores de bases de datos orientadas a objetos cobran licencia por su uso, o permiten su uso (para evaluación) por un periodo de tiempo. Aunque ya existían algunos administradores gratuitos -tal como *db4o*, la versión gratuita de *Versant*-, también influyo el factor humano, ya que no todos conocen las bases de este tipo, por lo que no habría alguna persona que pudiera mantener dicha base si se legara la aplicación usando bases orientadas a objetos; por otra parte, ya existían proyectos utilizando bases de datos relacionales, con las cuales se tenía más familiaridad. Por esto último se decidió usar una base de datos relacional, procediendo tanto a escoger el administrador de base de datos relacional como a diseñar el modelo Entidad-Relación -el diseño se hizo de antemano, ya que se debía verificar que dicho administrador pudiera manejar los tipos de datos requeridos en el diseño.

A continuación se muestra una tabla donde se comparan algunos administradores de base de datos -no se compararon todos los administradores de bases de datos más populares, por la gran cantidad de los mismos-:

Administrador de base de datos relacional.	**Ventajas:**	**Desventajas:**
Oracle.	Posee diversas herramientas y funciones que lo han convertido en referencia respecto a otras bases de datos. Tiene versiones para diversos sistemas operativos.	Requiere licencia y muchos recursos de cómputo; la versión gratuita, *Oracle Lite*, tiene menos herramientas pero aún así necesita muchos recursos de cómputo.
Access	Es relativamente popular. No se apega al *ANSI SQL*.	Es de licencia; no es eficiente para manejar concurrencia.
SQL Server	Maneja concurrencia adecuadamente.	Es de licencia; no proporciona librerías ***JDBC***, por lo que la comunicación con el lenguaje java es por medio del puente ***ODBC-JDBC***.
Paradox	Es una base de datos para aplicaciones de escritorio. Utiliza *QBE* (*Query by Example*) para facilitar las consultas a los usuarios, y posee conectores *SQL Links* nativos para comunicarse a otras bases de datos.	Es de licencia, y sólo para plataforma Windows; no provee librería ***JDBC***, por lo que una aplicación java tendría que usar el puente ***ODBC-JDBC***, a la vez que se crea un *SQL Link* a *ODBC*, o usar un ***JDBC*** de otro proveedor (pagando licencia).
DB2	Es una administrador confiable	Una computadora personal no le puede dar los recursos necesarios para que corra adecuadamente; brinda una librería ***JDBC***, pero varias de sus herramientas requieren el *Java Development Kit* de *IBM*. Es de licencia.
SQL Anywhere	Es confiable y tiene varias herramientas de administración y manipulación de datos.	Es de licencia; la versión de evaluación es por tiempo limitado.

Java DB	Es una base de datos relacional construida en el lenguaje java; usa *ANSI SQL*. Tiene versiones para varios sistemas operativos, y al ser una aplicación relativamente chica, consume pocos recursos de cómputo.	Es de reciente aparición; no existía al momento de iniciar el análisis, pero se menciona por las características que ofrece.
MySQL Community Edition	Es un administrador de bases de datos relacionales gratuito bajo la licencia *GNU* -la que permite modificar el código fuente, si los cambios hechos son publicados de forma gratuita. Existen muchas versiones para diversos sistemas operativos. Tiene la funcionalidad de cualquier administrador de base de datos comercial, pero no requiere tantos recursos de cómputo como las versiones *lite* o personales de productos similares de otros vendedores. Posee utilerías y librería ***JDBC*** gratuitas.	

Tabla 12: Comparación entre administradores de bases de datos relacionales.

Después de comparar diversos administradores de base de datos, se aprecian las ventajas de *MySQL Community Edition* (excluyendo a *Java DB*, que es reciente), sobre todo porque se puede usar gratuitamente, no utiliza muchos recursos a la vez que brinda la funcionalidad de administradores de bases de datos relacionales. Por otra parte, en el **Centro de Ciencias Aplicadas y Desarrollo Tecnológico** (**CCADET**, antiguo **Centro de Instrumentos**) ya se habían hecho o iniciado proyectos con dicho administrador, por lo que ya existe personal que conozca y administre.

La última aplicación esencial para poder crear el sistema es el servidor de aplicaciones web, o en su defecto, un contenedor de ***servlets*** y ***JSPs***. Los requisitos que aparecieron fueron los siguientes:

- Que fuera gratuito para su uso.
- Utilizara el lenguaje Java, y pudiera ejecutar el código de ***servlets*** y ***JSPs***.
- Que pudiera utilizar la librería ***JDBC*** de ***MySQL*** (de tipo 4, la cual no requiere instalaciones adicionales).
- Que pudiera llamar a interfases que invoquen a código nativo y/o usar la interfaz *Java Native Interface* (*JNI*), tanto para el control de los puertos locales como el video de la videocámara de estado sólido.

De forma simular, se compararon diversos servidores de aplicaciones, comparándolos en la siguiente tabla:

Servidor de aplicaciones web.	Ventajas.	Desventajas.
Apache-Tomcat.	Es una aplicación de código abierto que implanta los estándares de ***servlets*** y ***JSPs***, desarrollada y distribuida bajo la licencia de *Apache Software*. Algunos servidores de aplicaciones web lo siguen como referencia, o iniciaron con base en alguna de las versiones de *Tomcat*.	No es un servidor de aplicaciones web, sino un contenedor de ***servlets*** y ***JSPs***. No tenía respaldo de entornos de desarrollo integrados gratuitos hasta apenas unos cuantos años.
Resin.	Tiene varias versiones para diversos sistemas operativos, proveyendo de respaldo para ***servlets***, ***JSPs***, *EJBs*, *PHP* y otros estándares recientes para aplicaciones web. Ofrece versiones de código abierto y otras comerciales (de mayor rendimiento, pero son de licencia).	Cuando inició el proyecto no se tuvo acceso a la versión de código abierto, y las demás cobraban licencia.
Sun Glassfish Application Server.	Ofrece el respaldo para todas las tecnologías web de java que ofrece *Sun Microsystems*.	Actualmente es gratuito, pero las primeras versiones (que tenían otros nombres) eran de licencia. Requiere de más recursos de cómputo que otros servidores, debido a la cantidad de servicios que levanta para atender las diversas tecnologías que implanta.
JBoss	Es un servidor de aplicaciones empresariales web, que tiene base en ***Apache-Tomcat*** y en el *Apache Web Server*, por lo que también tiene contenedor de EJBs, e integra otras tecnologías.	Es de licencia; su consola de administración es confusa.
WebLogic	Es un servidor de aplicaciones web poderoso, el cual puede contener ***servlets***, ***JSPs***, *EJBs*, *webservices*, contando también con su propio entorno de desarrollo integrado.	La versión personal sólo permite un dominio en un equipo.
WebSphere	Es un servidor de aplicaciones web robusto, de *IBM*. La versión gratuita, *WebSphere Community Edition*, es una versión personal construida con base en productos de *Apache Software*.	Son de licencia. Las primeras versiones del mismo estaban más enfocadas para equipos y productos *IBM*; las versiones más recientes ya colaboran con otros productos java de terceros. Aunque es tecnología 100% java, este servidor utiliza configuraciones y librerías exclusivas de *IBM*; una de las últimas versiones tenía errores que impedían generar *webservices*.
Oracle Application Server (OAS).	Fue reemplazado por *WebLogic*, cuando *Oracle* adquirió *Bea Systems*.	Estaba más inclinado a integrarse con otras herramientas de *Oracle*.

Tabla 13: Comparación entre servidores de aplicaciones *web*.

Después de comparar sus características, se descartaron los servidores que requerían de pago de licencia;

después se descartaron los que requerían otras aplicaciones para funcionar y los de licencias personales (los que requieren un registro por cada instalación).

Se prefirió el ***Apache-Tomcat*** porque es gratuito, porque se puede redistribuir respetando su licencia (de *Apache Software*), porque sirve de referencia para construir otros servidores de aplicaciones (a los cuales se podría migrar la aplicación posteriormente), y porque no requiere de muchos recursos de cómputo. Aunque no es un servidor de aplicaciones web (porque no ofrece el respaldo de todas las tecnologías web) ni posee la capacidad administrativa de los servidores de aplicaciones, sí tiene la capacidad para ejecutar el código de los ***servlets*** y ***JSPs***, integrar otras librerías (como el ***JDBC*** de *MySQL* y las de ***Java Media Framework***), y de llamar a librerías dinámicas (nativas del sistema operativo) por medio de *JNI*.

Algunos servidores de aplicaciones ofrecían mejor rendimiento, pero se prefirió desarrollar la aplicación en un estándar como el *Apache-Tomcat*, y si fuera necesario cambiar a un servidor de aplicaciones, sería más fácil mover migrar la aplicación web, y después efectuar la configuración del nuevo servidor para la aplicación.

Las últimas herramientas escogidas (no esenciales) fueron un entorno de desarrollo integrado (*IDE*), alguna herramienta para diagramar circuitos electrónicos, y alguna de *CASE*.

Los entornos de desarrollo integrados no son considerados como absolutamente indispensables (pero más adelante fueron casi vitales), pero sí fueron de mucha ayuda para el proyecto, para poder vincular el editor con el compilador, ahorrando tiempo para detectar errores antes y al compilar la aplicación. Los códigos más simples (y varios de prueba) fueron hechos en el editor de notas y compilados en una ventana de consola. Más adelante se decidió aprovechas las bondades de los entornos gráficos, utilizando la versión personal del *JBuilder*, cambiando posteriormente al *JDeveloper* (el cual revisa la sintaxis de los ***JSPs***), y más recientemente, con ***NetBeans***, que permite la depuración de las aplicaciones web.

2.1.2 Diseño de la aplicación.

El diseño de la aplicación se dividió en dos partes, las cuales son independientes en esta fase de desarrollo del sistema:

- El diseño de la base de datos relacional.
- El diseño de la programación de la aplicación (*software*).

2.1.2.1 El diseño de la base de datos relacional.

El diseño de la base de datos relacional se hizo con la metodología ***CASE*Method*** (explicada en el punto ***1.6 de Conceptos Básicos***), prefiriéndose ésta sobre otras (como la de Chen), debido a que es más fácil modelar relaciones complejas, como la de muchos a muchos, y descomponerla con ayuda de una nueva entidad dependiente. A pesar de no contar con herramientas *CASE*, se siguió la metodología diagramando con una aplicación para graficar y hacer presentaciones.

El diseño debía permitir guardar datos del sistema, y de usuarios del mismo; debía poder guardar información binaria, y datos flotantes -para poder guardar otro tipo de mediciones, en la misma base de datos; también debía ser independiente de los lenguajes de programación, y de los administradores de bases de datos relacionales (el cual todavía se seguía escogiendo todavía). El diseño siguió los siguientes pasos:

1 Identificar las entidades y sus atributos.

2 Establecer las relaciones entre las entidades.
3 Pasar el diseño a la primera forma normal.
4 Llevar el diseño a segunda forma normal.
5 Concluir el diseño pasando a la tercera forma normal.

El modelo Entidad-Relación obtenido se encuentra en el anexo B. También se construyó la tabla de relaciones, que muestra las asociaciones de las entidades del sistema, la cual se muestra a continuación:

		USUARIO	PUESTO	INSTITUCIÓN	PAÍS	ESTADO	NIVEL_USUARIO	NIVEL	SESIÓN	MEDICIÓN	INSTRUMENTO	CONDICIÓN_INSTRUMENTO	AUTORIZACIÓN	CONDICIÓN_USUARIO	TELÉFONO	TIPO_TELÉFONO
USUARIO		asignado	afiliado	habitando	residiendo	jerarquizado		creando				otorgando		posey endo		
PUESTO	ejercido															
INSTITUCIÓN	afiliando															
PAÍS	habitado															
ESTADO	residido															
NIVEL_USUARIO																
NIVEL	jerarquizando															
SESIÓN	creada															
MEDICIÓN	obtenida								obteniendo							
INSTRUMENTO									calculando		habilitado					
CONDICIÓN_INSTRUMENTO										habilitando						
AUTORIZACIÓN	otorgada												calificando			
CONDICIÓN_USUARIO												calificando				
TELÉFONO	poseído															clasificado
TIPO_TELÉFONO														clasificando		

Tabla 14: Matriz de relaciones del modelo Entidad-Relación para la base de datos del Administrador de la Videocámara.

Para el caso de la entidad Usuario, los atributos Colonia, Municipio, Código Postal, Apartado y Población,

tendrían a su vez atributos, por lo que se convertirían en Entidades (para pasar a la primera forma normal ***NF1***) si el modelo se apegara fielmente a la organización de países en estados (o departamentos), los cuales contienen municipios (o división política análoga), que se dividen a su vez en colonias o barrios, los cuales tienen asignado uno o más códigos postales. Esto implica que también se tendría que modelar las relaciones y las dependencias de las nuevas entidades, lo cual aumentaría la complejidad del modelo.

También se tendría que capturar todos los estados, municipios, poblaciones, etcétera, de todos los países, lo cual es posible, pero consumiría mucho tiempo; esto se podría hacer para un sistema de mensajería, donde sí importa la asociación de la dirección con la división política. Para la forma de registro de Usuarios de este sistema, se considerará que los atributos mencionados anteriormente son atómicos (conteniendo sólo cadenas de caracteres o números), y que sólo dependen de la llave primaria de Usuario.

Para el sistema desarrollado para esta tesis no tiene tanta relevancia el estado o país donde residan los usuarios del sistema; lo más importante es comprobar que se puede manipular un dispositivo periférico a través de Internet, y obtener una respuesta del mismo. La base de datos se construyó para guardar información de la respuesta de dicho periférico, y para registrar algunos datos de usuarios; los cuales tienen diferentes permisos para borrar datos o bloquear el acceso al periférico.

El modelo de la base de datos fue modificado hasta llegar a la tercera forma normal (***3NF***), la cual es recomendada para la mayoría de los modelos de bases de datos relacionales. Existen otras formas normales más estrictas, pero no se revisó si el modelo Entidad-Relación cumple con esas formas normales, ya que se considera que dicho análisis está fuera del alcance de esta tesis.

El paso siguiente fue el diseño de las tablas, en donde cada entidad se convierte en una tabla; el nombre de la tabla es el plural del nombre de la entidad, y cada atributo de la entidad se convierte en una columna de la nueva tabla. La entidad en la cual la relación con otras entidades es obligatoria o tenga cardinalidad múltiple, su tabla correspondiente se construye agregando una columna por cada relación que tenga con otras entidades; este tipo de columnas se conocen como llaves foráneas.

Por cada tabla se forma una matriz, en la que se indica por cada columna, si es obligatoria (No Nula, *NN*) u opcional (si debe tener dato o puede ser nulo), si es única (U), y además se indica si la columna es un tipo de llave -primaria (*PK*), secundaria o foránea (*FK*)-. Después de agregar las columnas, se pude agregar ejemplos de tipos de datos por columna, que corresponden de uno a tres registros que podría tener la tabla; estos datos sirven como referencia para la construcción física de las tablas. El diseño físico de las tablas se muestra a continuación:

Columna:	id_puesto	descripcion
Tipo llave:	PK	
No nulo/Único:	NN, U	NN
Ejemplos:	1	Investigador
	2	Profesor
	3	Alumno

Tabla 15.1: Diseño físico de la tabla *puestos* (*ocupaciones*).

Columna:	id_estado	nombre	nombre_internet	nombre_oficial	nombre_oficial_internet	abreviatura
Tipo llave:	PK					
No nulos Únicos:	NN, U					
Ejemplos:	9	Distrito Federal	Distrito Federal	Distrito Federal	Distrito Federal	D. F.
	16	Michoacán	Michoacá	Michoacán de Ocampo	Michoacá de Ocampo	Mich.
	17	Morelos	Morelos	Morelos	Morelos	Mor.

Tabla 15.2: Diseño físico de la tabla *estados*.

Columna:	id_institucion	nombre
Tipo llave:	PK	
No nulos Únicos:	NN, U	NN
Ejemplos:	1	U. N. A. M.
	2	Centro de Instrumentos
	3	Facultad de Ingeniería

Tabla 15.3: Diseño de la tabla *instituciones*.

Columna:	id_pais	nombre	nombre_internet
Tipo llave:	PK		
No nulos Únicos:	NN, U	NN	NN
Ejemplos:	2	Albania	Albania
	115	México	México
	143	República Checa	República Checa

Tabla 15.4: Diseño de la tabla *paises*.

Columna:	descripcion
Tipo llave:	PK
No nulos **Únicos:**	NN, U
Ejemplos:	Administrador
	Usuario

Tabla 15.5: Diseño de la tabla *niveles*.

Columna:	descripcion_nivel	nombre_usuario
Tipo llave:	FK	FK
No nulos **Únicos:**	NN	NN
Ejemplos:	SU	hrz
	Administrador	Benjas
	Usuario	Juan

Tabla 15.6: Diseño de la tabla *niveles_usuarios*.

Columna:	id_telefono	numero_telefonico	id_usuario	id_tipo_telefono
Tipo llave:	PK		FK	FK
No nulos **Únicos:**	NN, U	NN	NN	NN
Ejemplos:		56 22 86 02 Ext. 122	1	2
		56 76 02 54	2	1
		445551234567	1	3

Tabla 15.7: Diseño de la tabla *telefonos*.

Columna:	id	descripcion
Tipo llave:	PK	
No nulos **Únicos:**	NN, U	
Ejemplos:	1	Casa
	2	Oficina
	3	Celular

Tabla 15.8: Diseño de la tabla *tipos_telefonos*.

Columna:	id_condicion_usuario	descripcion
Tipo llave:	PK	
No nulos **Únicos:**	NN, U	NN
Ejemplos:	1	Activo
	2	Inactivo

Tabla 15.9: Diseño de la tabla *condiciones_usuarios*.

Columna:	fecha	hora	autorizado	autoriza	id_condicion_usuario
Tipo llave:	PK	PK	PK, FK	PK, FK	FK
No nulos **Únicos:**	NN	NN	NN	NN	NN
Ejemplos:	37248	0.64596	7	1	1
	37257	0.74333	14	1	2

Tabla 15.10: Diseño de la tabla *autorizaciones*.

Columna:	numero_sesion	cadena_sesion	fecha_acceso	hora_acceso	fecha_salida	hora_salida	ip_conexion	id_usuario
Tipo llave:	PK1	PK2						FK
No nulos Únicos:	NN, U	NN, U	NN	NN	NN	NN	NN	NN
Ejemplos:	3e+07	ACD15FB	37278	0.5	37279	0.785	18721411	1
	5e+07	BDSDF1A	37255	0.604	36891	0.09	127.0.0.1:8090	2

Tabla 15.11: Diseño de la tabla *sesiones*.

Columna	id_medicion	fecha	hora	cantidad	valor_binario	id_instrumento
Tipo llave:	PK					FK
No nulos Únicos:	NN, U	NN	NN			NN
Ejemplos:	1	37294	0.41763	112		1
	2	37294	0.41807		ABD$#%$#	1

Tabla 15.12: Diseño de la tabla *mediciones*.

Columna:	id_condición_instrumento	descripción
Tipo llave:	PK	
No nulos Únicos:	NN, U	NN
Ejemplos:	1	Habilitado
	2	Inhabilitado
	3	Descompuesto

Tabla 15.13: Diseño de la tabla *condiciones_instrumentos*.

Columna:	id_instrumento	nombre	id_condición_instrumento
Tipo llave:	PK		FK
No nulos Únicos:	NN, U	NN	NN
Ejemplos:	1	Videocámara	1
	2	eesdf	2

Tabla 15.14: Diseño de la tabla *instrumentos*.

Columna:	id_usuario curp nombre apellido_paternoapellido_maternofecha_nacimientonumero_exteriornumero_interiorcalle cp dirección apartado municipio población correo_electrónico pagina_internet nombre_usuario id_puesto id_estado id_institución id_país
Tipo llave:	PK1 PK2 FK FK FK FK
No Nulos / Únicos:	NN,U NN,U NN NN NN NN NN NN NN NN NN
Ejemplos:	1 RUZH730706 Hugo Ruiz Zepeda 2 Portal 91 Cerca Jardines Delfín 56050 Xochimilco México hruiz@internet.com.mx http://www.hugo.com.mx Ruiz 00ABC 3 9 2 115
	41 PEHJ700214 Juan Pérez Hernández 25612 Carretera México-Puebla Km 50 Addsedd Adjikjh juan jjjiui 3 21 1 115

Tabla 15.15: Diseño de la tabla *usuarios*.

Las tablas se construyen con base en sentencias *SQL* de tipo *DDL* (*Data Definition Language*), iniciando con la sentencia de creación de la base de datos, siguiendo con cada tabla, indicando el nombre de la tabla, el nombre y tipo de columna, y si puede ir nulo o no (dato no obligatorio u obligatorio, respectivamente). Al último se construyen otros elementos de la base de datos, tales como índices. La mayoría de los administradores de bases de datos relacionales aceptan las sentencias *DDL* del *ANSI 92*, aunque la construcción varía dependiendo de dicho administrador, según las mejorías que quiera implantar dicho administrador.

En este caso, ya se había escogido *MySQL* como administrador de base de datos, por lo que la base de datos y sus tablas se construyeron mediante sentencias *DDL* propias de *MySQL*, mapeando cada matriz, identificando el tipo de dato que corresponde a cada columna, indicando la opcionalidad, y si fuera el caso, señalando si es llave primaria. Todas estas instrucciones *DDL* se escribieron en un archivo *SQL* para ejecutarlo como un guión.

Finalmente, se hizo otro guión con instrucciones ***SQL*** de tipo *DML* (*Data Manipulation Language*), para insertar datos en la base de datos recién creada, creando un usuario por defecto, el super administrador (*SU*), y agregando información en las tablas que se conocen popularmente como catálogos (debido a que rara vez cambian y sirven como referencia).

2.1.2.2 Diseño del sistema.

El diseño del sistema gira en torno del lenguaje de programación java, ya que este último provee todos los elementos de presentación, control, adquisición de imágenes, de base de datos, etcétera, y además permite integrar dichos elementos de forma armoniosa.

Siguiendo la metodología de *RUP*, que aconseja el desarrollo iterativo, y apegarse a arquitecturas para poder administrar las clases de java a generar, se decidió dividir el diseño y desarrollo del sistema por módulos, probar los códigos de forma individual, y una vez resuelto los problemas técnicos más cruciales de un módulo, integrarlo en el ambiente de web, y probar y corregir de nuevo. Se evitó concentrar toda la programación en un archivo fuente, que una misma clase efectuara tareas de distinta naturaleza, prefiriendo que la programación se dividiera en clases, y a su vez, estas últimas se agruparían por el tipo de tareas que llevan a cabo. Siguiendo esta pauta, el diseño del sistema fue subdividido por el tipo de función que lleva a cabo:

- Circuito electrónico de control.
- Módulo de control del puerto paralelo
- Módulo de base de datos.
- Módulo de video (control de la videocámara *CCD* y obtención de imágenes fijas).
- Módulo de web (páginas *HTML*, ***servlets*** y ***JSPs***), el cual contiene elementos de presentación y otros que controlan e integran a los módulos anteriores.

Las primeras clases del lenguaje java programadas para esta aplicación tenían una arquitectura muy simple, siguiendo las convenciones del lenguaje java, dividiendo la funcionalidad de la clase en varios métodos. Se dejó la división de clases por módulo; más adelante, en algunas clases se comenzó a aplicar el concepto de herencia, para evitar volver a programar métodos ya hechos, y para evitar tener muchas versiones de clases similares.

La complejidad del problema a resolver impedía diseñar todo el sistema a detalle antes de programarlo; *RUP* recomienda el desarrollo iterativo, en la que se programa, se prueba, y se hacen los cambios o ajustes necesarios; para los módulos de control del puerto paralelo, de base de datos, y el de video, no se especificó una arquitectura más específica (aún), pero se hizo un ***Diagrama de Flujo de Datos*** sencillo del proceso donde el usuario solicita el cambio de posición de la videocámara, y se le responde con una página *HTML* con la imagen recién obtenida, y se decidió que era mejor que se pasara a la fase de desarrollo.

Sin embargo, antes de comenzar a programar ***servlets*** y otros componentes web, sí se compararon arquitecturas web, tanto para organizar el desarrollo de los componentes web como para facilitar la integración de los otros módulos (diseñados y construidos de forma independiente). Primero se compararon las arquitecturas enfocadas a la página o foliocentristas[49], en las cuales (después de recibir la petición) un ***JSP*** de presentación llama a otro *JSP* que tiene código de procesamiento y de presentación; esta arquitectura permite el desarrollo rápido de páginas web, pero si el código de procesamiento es complejo o cambiante, causaría que se tenga que modificar mucho el JSP que concentre el procesamiento. Debido a lo anterior, se descartó este tipo de arquitectura.

El siguiente tipo de arquitectura revisada fue el modelo de despachador o remisión, la cual se basa en el modelo ***MVC*** (*Model-View-Controller*, Modelo-Vista-Control)[49], cuyas partes se explican a continuación:

- Modelo, el cual es la plantilla de los datos; en esta aplicación son los *JavaBeans*.

- Vista, la cual se expone al usuario mostrando los datos del modelo. Los *JSPs* son los objetos vista en la aplicación web.
- Control, el cual recibe las peticiones del usuario (incluyendo datos de entrada), procesa información, llaman

a otras clases, invocan procesos, y en general, ejecutan diversas tareas; crea instancias de modelo, agregándoles los datos de salida (los que se van a mostrar al usuario) y se los envía a la Vista, a la cual le entregan el flujo del proceso. Los ***servlets*** son los objetos control del sistema desarrollado.

El modelo ***MVC*** es relativamente sencillo, pero a la vez poderoso, porque en un sistema cambiante, se modifican los métodos de las clases de control (y las auxiliares que use); las clases de modelo (*JavaBeans*) cambian poco, mientras que la vista (***JSPs***) no se altera o tiene cambios mínimos. También permite (a diferencia de la arquitectura foliocéntrica) que dos o más personas trabajen a la vez en un módulo, ya que un diseñador puede estar modificando los ***JSPs***, mientras que un programador cambia las reglas de negocio o el proceso que efectúa un servlet. Otra ventaja adicional es que la navegación por la aplicación web no se afecta mucho mientras se hacen los cambios.

El modelo anterior fue expandido en las arquitecturas web de despachador, y en específico, se implantó la arquitectura ***Servicios al Trabajador***[50], la cual hace una separación más clara de la presentación y del procesamiento; esta arquitectura se recomienda cuando se tiene que hacer mucho procesamiento de la petición, por la cantidad de información de una forma *HTML*, o en este caso, para recibir y procesar los parámetros para controlar la videocámara.

Con base en el modelo antes descrito, se desarrollaron las pantallas (***JSPs***), los cuales son vistos por el usuario, despliegan información y tienen interacción con el mismo. Las peticiones del usuario son enviadas a los ***servlets***, los cuales procesan las peticiones (junto con los parámetros), llaman a otras clases, tienen interacción con la base de datos, crean y alimentan de datos a objetos ***JavaBeans***, los cuales son enviados como atributos (de petición, sesión, o de aplicación), y re-direcciona la aplicación a otro ***JSP***, el cual recupera los ***JavaBeans***, y se despliega al usuario, mostrando los datos contenidos en los *JavaBeans*.

Cuando se comenzaron a integrar los módulos de control del puerto paralelo y de la videocámara con los de web, apareció otro problema distinto. Dichos módulos se habían desarrollado y probado como aplicaciones de consola (un sólo usuario), pero dentro del ambiente de web pasaban a un ambiente web multi-usuario, en la que peticiones simultáneas de usuarios podrían causar conflicto por el control de dichos periféricos. En este caso, ya se había desarrollado parte del sistema, y se regresó de nuevo a la fase de diseño, lo cual es permitido por *RUP*. Para resolver el problema, se implantó el modelo ***Singleton*** (un único elemento)[39] tanto para el control del puerto paralelo como de la videocámara en nuevas clases sincronizadas; al recibir peticiones simultáneas que involucren el puerto paralelo o la videocámara, se van atendiendo las peticiones una por una (comenzado por la primera que llegó), y hasta que se acabe de atender una petición, se atiende la siguiente.

Al comenzar a integrar el módulo de base de datos con la aplicación web, también se regresó de la implantación (construcción) al diseño. Aunque al principio de la construcción del sistema web ya se tenía código que tenía interacción con la base de datos, se le dio mayor preferencia al desarrollo e integración de los módulos del puerto paralelo y de la videocámara, para concluir el módulo de base de datos al último; *RUP* también recomienda dar mayor prioridad a las partes más complejas del proyecto. Para concluir e integrar el módulo de base de datos con el resto de la aplicación web, se amplió la arquitectura ya existente.

Para poder almacenar información en la base de datos e integrar los demás módulos, se implantó el modelo ***DAO*** (*Data Access Model*, Modelo de Acceso a Datos)[48], el cual consta de los siguientes componentes.

1. ***DAO***, que son clases sencillas que contienen la lógica de manipulación de datos. El intercambio de información se hace con un objeto *Data Source* (fuente de datos); cada ***DAO*** contiene los métodos de consulta, inserción, modificación y/o borrado relacionados con una entidad (tabla de la base de datos).
2. ***Business Objects*** (objetos de negocio), los cuales crean e invocan los métodos de los objetos ***DAO***.

En este sistema, los ***servlets*** se convierten en ***Business Objects***, sin tener que reemplazarlos; los ***servlets*** ya no hacen consultas directas a la base de datos, removiendo las sentencias *SQL* de los ***servlets*** y pasándolas a un objeto ***DAO***. Los ***servlets*** siguen controlando el procesamiento de peticiones y flujo de la aplicación.

3. ***Data Source*** (fuente de datos), es una clase que encapsula el origen de los datos; le da un nombre y forma de acceso único al resto de la aplicación. Se creó un ***Data Source*** en ***Apache-Tomcat***, aprovechando las nuevas características de ***Tomcat*** versión 5 en adelante, reemplazando a la clase *PozoConexiones.java*, que creaba y administraba varias conexiones; los nuevos objetos ***DAO*** buscan la fuente de datos, y le solicitan una conexión a la base de datos.
4. ***Transient Objects*** (objetos de transferencia), los cuales llevan datos de los ***Business Objects*** a los ***DAO***, y viceversa. En este sistema, los *JavaBeans* son usados como ***Transient Objects***; no se cambia la estructura de los *JavaBeans*, sino la forma de utilizarlos. El mismo tipo de *JavaBean* (modelo) con que se pasan datos de un ***servlet*** (controlador) a un ***JSP*** (vista) en la arquitectura ***MVC***, ahora sirve como objeto de transición, reusando código y ampliando el papel de estos componentes.

Estos elementos se crearon para modelar las tablas de la base de datos relacional; cada tabla se representa por una clase de tipo objeto de transferencia, y cada columna se convierte en un atributo de la clase. No se usó ninguna herramienta automatizada para crear estos objetos a partir de la base de datos.

La última parte del diseño fue la de seguridad; en esta aplicación web ya se había construido un módulo de administración, pero no todos los usuarios lo deberían usar, sólo los operadores registrados podrían detener la videocámara o cambiar permisos a otros usuarios. Por otra parte, tampoco se quería restringir el acceso a toda la aplicación, ni rehacer el código. Se modificó la base de datos, cambiando la tabla ***niveles***, para indicar los papeles o funciones de los usuarios del sistema; se crea otra llamada ***niveles_usuario***, la cual relaciona a un usuario con un nivel (función o papel)[43].

En la configuración del ***Apache-Tomcat*** en *web.xml* se indican los mismos papeles o funciones contenidos en la tabla niveles, y se delega a ***Tomcat*** la autenticación de usuarios.

2.2 Análisis y diseño de los Componentes Físicos.

2.2.1 Análisis del circuito de control de movimiento de la videocámara.

Originalmente el circuito empleado para las pruebas de desarrollo era un circuito formado por *buffers* de salidas de tres estados SN74LS241N (para aumentar la señal del puerto paralelo), y circuitos de tipo *latch* SN74LS373N (para retener el valor del bit proveniente del *buffer*), cuya salida se desplegaba en *leds* -con unas resistencias de por medio, para reducir la corriente.

Más adelante, cuando se hicieron varias pruebas satisfactorias de enviar valores y secuencias por el puerto paralelo desde una página web (encendiendo y apagando leds del circuito mencionado anteriormente), se procedió a diseñar un circuito electrónico de control.

Por una parte, los datos de salida del puerto paralelo debían conservarse temporalmente, antes de que se recibiera el siguiente byte por el mismo puerto, dando tiempo a que el circuito electrónico produjera una respuesta ante tal valor de entrada, pero se descartó el uso de circuitos electrónicos sofisticados, como los procesadores digitales de señales (*DSPs*, *Digital Signal Processors*).

Por otra parte, se requería que la camara se moviera con precisión y que se desplazara uniformemente, por lo que se optó por usar motores de paso, los cuales siempre rotan la misma cantidad de radianes en cada paso. También se seleccionó que dieran un paso completo para transferir el movimiento directamente a la videocámara, y no de medio paso -que regresa a su posición original, transfiriendo el movimiento a través de un sistema de engranes.

La configuración del alambrado de control le permite al motor de pasos dar un paso completo en un sentido, al enviar una secuencia de cuatro señales por los cuatro bits de datos (provenientes del puerto paralelo), alterando el orden en que se alimentan las bobinas del motor, y por efecto del juego de atracción y repulsión electromagnética se lleva a cabo el movimiento de motor. El orden en que se activan y desactivan las bobinas se muestra a continuación:

Paso	Bobina 1	Bobina 2	Bobina 3	Bobina 4
1	Alto	Bajo	Bajo	Bajo
2	Bajo	Alto	Bajo	Bajo
3	Bajo	Bajo	Alto	Bajo
4	Bajo	Bajo	Bajo	Alto

Tabla 17: Secuencias de pasos para el motor bipolar.

La lógica de programación del Módulo de Control del puerto paralelo indica la secuencia indicada para girar en uno u otro sentido; ya que se usa dicho puerto para el control de dos motores (movimientos horizontal y vertical) simultáneamente, la lógica de programación también indica que se envíen los mismos datos de los cuatro bits correspondientes al motor que no se mueve (evitando que se altere la polaridad de las bobinas), tal y como se indica en ***Análisis y Diseño de la Aplicación***.

2.2.2 Diseño del circuito de control de movimiento.

El circuito de control se diseñó como un circuito de control abierto, el cual efectúa el cambio de posición (horizontal o vertical) a voluntad del usuario, sin una retroalimentación del cambio de posición. Por lo anterior, se decidió usar piezas relativamente simples y de bajo costo.

Para poder mover la videocámara, se escogió un motor de pasos (uno para el movimiento horizontal y otro para el vertical), ya que se puede controlar con precisión los movimientos de cada motor, independientemente de la velocidad del microprocesador del equipo servidor.

Cabe mencionar que se pudiera construir un circuito con motores de corriente directa que se movieran más rápido que uno de pasos, pero el problema es que para que se muevan uniformente los motores, el lapso entre que se inicia y termina el movimiento tendría que ser constante; esto podría resolverse mediante programación, al tener un contador que inicie, se verifique a intervalos si el lapso de permitir movimiento no ha sido excedido, y si es así, detener el movimiento. Sin embargo, si el equipo donde se ejecuta ese programa tiene mucha carga de trabajo, pudiera no mandar la orden de detener movimiento a tiempo.

También se podrían hacer circuitos con contadores y motores de corriente directa, pero este tipo de circuitos tendría que ser calibrado uno por uno (si se hacen varios de ellos) o usar componentes eléctricos de precisión. En ambos caso aumentaría el costo de los circuitos, en comparación con el circuito con motores de paso.

Por lo anterior se prefirió el circuito con motores de paso, y para crearlo se alambró un circuito con una etapa de potencia que alimenta a un motor de pasos PM55L-048-HPG9; el circuito integrado empleado (por motor) es un ULN2003AN al cual le llegan cuatro bits de datos del puerto paralelo; estos circuitos consisten en arreglos de diodos *darlington*, los cuales permiten acoplar circuitos de lógica *TTL* (de 5 [V]) con la fuente de 12 [V], 300 [mA], que alimenta a los motores.

El motor de pasos PM55L-048 (HPG9 es su serie) es un motor bipolar, lo que significa que puede girar en ambos sentidos al alternar la alimentación de las bobinas. Tiene 5.5 [cm] de diámetro, y requiere de 48 pasos para una revolución completa (gira 7.5º por cada paso).

El circuito electrónico empleado es el siguiente:

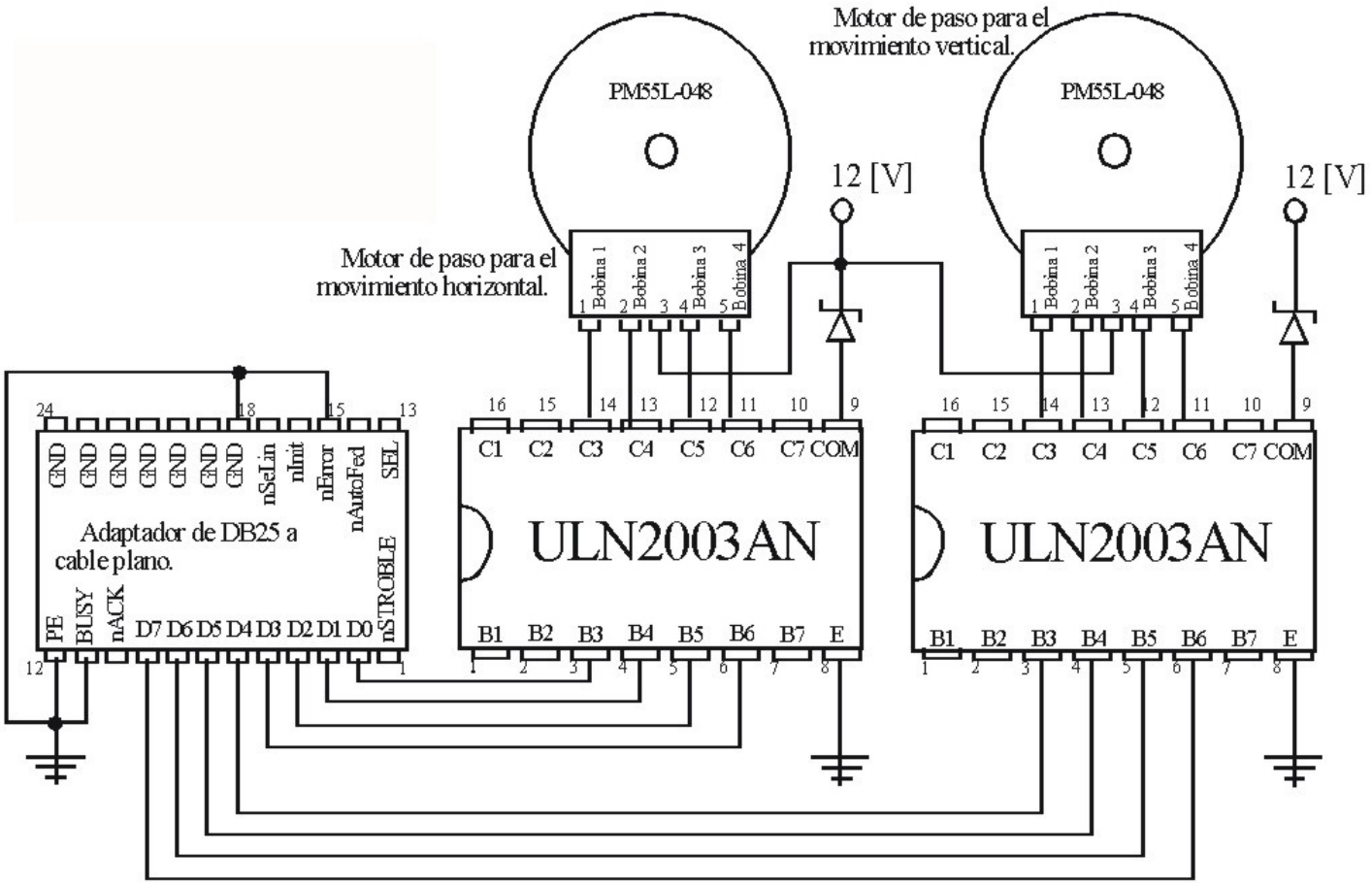


Figura 24*: Diagrama del circuito electrónico de control horizontal y vertical de la videocámara web.*

En el circuito mostrado, se utilizaron las entradas B3 a la B6 (y sus correspondientes salidas C3 a C7) en el *ULN2003AN* sólo por cuestión de espacio en la tableta *protoboard*, ya que originalmente se iba a alambrar otro circuito. Se utiliza un diodo *zener* para conectar de común a la fuente de 12 volts, ya que esto se recomienda para evitar cualquier corriente inversa que pudiera venir de la fuente.

2.3 Implantación y construcción de la aplicación.

2.3.1 Implantación de la aplicación.

La implantación (construcción) de la aplicación comenzó por la instalación de las aplicaciones con las que se construiría el sistema:

- *Java Development Kit* (*JDK*), el cual provee el compilador del lenguaje java, y otras herramientas de programación.
- *Java Runtime Enviroment* (*JRE*), el cual le da soporte para la ejecución de aplicaciones java; al principio no se usaba para desarrollo, pero más adelante las últimas versiones de ***Apache-Tomcat*** pueden ejecutarse con el *JRE* y ya no exclusivamente con el *JDK*.
- Administrador de base de datos *MySQL*.
- Contenedor de ***servlets Apache-Tomcat***.
- Interfaz *Java Media Framework 2.1.1e* con *Performance Pack* para *Windows*.
- Interfaz *Java Communications 2.0* para *Windows* (con librería dinámica de 32 bits).
- *JDBC Connector/J* para *MySQL*.
- Entorno de desarrollo integrado (*IDE*, *Integrated Development Enviroment*).

Durante el desarrollo del sistema se cambiaron las versiones de las aplicaciones mencionadas anteriormente; se cambió el entorno de desarrollo integrado para poder utilizar la depuración de código en línea y de otras ventajas que ofrecían, mas no el código fuente ya desarrollado previamente. La construcción del sistema inició sobre *Windows Millenium*, debido a que era más accesible una computadora personal con dicho sistema operativo.

Se decidió hacer, al principio, clases de prueba por cada módulo, para verificar que se podía hacer uso del puerto paralelo, que se podía conectar y manipular información de la base de datos, hacer ***servlets*** y que respondieran generando *HTML*, y más adelante, se comenzaron a hacer las clases para manipular la multimedia de la videocámara. Cada módulo se comenzó a desarrollar individualmente como si fueran aplicaciones de consola (excepto la de web) creando clases del lenguaje java con la funcionalidad más básica de su módulo, y siguiendo las convenciones de nombres de clase y de variable que recomienda *Sun Microsystems*.

Después de hacer clases que efectuaran las acciones más básicas de su módulo, resolviendo los problemas técnicos del mismo, se hicieron otras versiones en las que se incluía la funcionalidad ya probada, agregando métodos nuevos, y eliminado el uso de rutas fijas y otras configuraciones del código, buscando que estas versiones tuvieran la función más general, mientras que para tareas más específicas se crearon otras clases, que importan las clases más genéricas (composición); en algunas se comenzó a aplicar el concepto de herencia, para no re-escribir todo el código, usar los métodos heredados como propios a la vez de agregar sus métodos y atributos propios. Estas clases fueron probadas como aplicaciones de consola, por módulo, e independientes entre sí. De forma más específica, el desarrollo por módulo fue el siguiente:

- Módulo del puerto paralelo: Fue el primer módulo en diseñarse y construirse; se crearon clases que identificaban los puertos locales de la computadora, se abría el puerto paralelo construyendo flujos para enviar un dato numérico, y más adelante, se enviaban secuencias, verificando en un circuito *protoboard* con memorias *buffers* -las cuales retienen el valor recibido del puerto paralelo hasta que reciban otro dato- y *leds* (diodos emisores de luz), que estos últimos se prendieran y apagaran para mostrar en binario el número enviado. Una vez que se consiguió dicha funcionalidad, se creó la clase **Paralelo16.java**, en la cual se colocan los métodos relacionados con el uso del puerto paralelo, tales como identificar y abrir el puerto paralelo, abrir un flujo de datos hacia el mismo, enviar de un dato numérico, y cerrar el puerto. Se delega

a otras clases el envío de datos específicos y/o secuencias, las cuales tienen que crear una instancia de **Paralelo16.java**, y hacer usos de sus métodos. Cabe mencionar que estos códigos se desarrollaron y probaron en *Windows 9x* y *Millenium*, funcionando correctamente sobre estas plataformas.

- Módulo de base de datos: Se comenzó más adelante, con clases de prueba que cargaban el controlador ***JDBC***, se abría la conexión a la base de datos con una *URL* fija en el código, y se hacían consultas simples, revisando que se pudiera recuperar información de las consultas; después se hicieron otras clases de prueba para comprobar que pudiera efectuar inserciones, borrados y ejecutar *stored procedures* hacia *MySQL*. Al resolver los problemas de configuración de la *URL* de conexión, se comenzó a crear otras clases. Para conectarse a *MySQL* se había creado la clase **ConexionGeneralBD.java**, la cual carga un archivo de propiedades, construye la *URL* de conexión, y se encarga de conectarse y desconectarse a la base de datos. Otra de ellas era **PozoConexiones.java**, la cual cargaba el controlador ***JDBC***, leía un archivo de propiedades, creaba y almacenaba varias conexiones a la base de datos, para que posteriormente proporcionara una conexión cuando se le solicitara, con el fin de administrar las conexiones activas a la base de datos; más adelante fue reemplazada al configurarse una fuente de datos en ***Apache-Tomcat*** que tenía la misma funcionalidad, pero que permite ser llamada desde cualquier parte de la aplicación web, ahorrando el esfuerzo de crear o llamara a instancias de **PozoConexiones.java** para obtener una conexión a *MySQL*. Como este módulo no era tan crítico, se pospuso esta parte del desarrollo, para darle mayor prioridad al módulo de video y a su integración en web, tal y como aconseja *RUP*. Se hizo la clase **OperacionesGenericasMySQL.java**, la cual contiene métodos genéricos que reciben cadenas y efectúan operaciones ***DML*** (*Data Manipulation Language*) en la base de datos *MySQL*, regresando valores *booleanos*, enteros u objetos *ResultSet*.
- Módulo de video: Este es el módulo más complejo, y que requirió más tiempo y esfuerzo, a pesar que la interfaz ***Java Media Framework*** prometía ser fácil de usar. Comenzó por buscar el medio con el que se controla el video de la videocámara de estado sólido, utilizando la interfaz ***JMF***. Al igual que los otros módulos, se comenzó por desarrollar clases que se utilizaran como aplicaciones de consola, las cuales primeramente, buscaban conectarse a la videocámara para activarla y apagarla.
- Módulo de web: Se comenzó por construir ***servlets*** y ***JSPs*** de prueba que generaran una respuesta ***HTML*** sencilla, los cuales eran llamados desde *Internet Explorer* y *Netscape*. Después se inició con la construcción de otras páginas con base en la arquitectura ***MVC*** (*Model-View-Controller*, Modelo-Vista-Controlador), creando ***servlets***, ***JavaBeans*** y ***JSPs*** para la presentación. También se había iniciado la construcción de una pantalla de autenticación de usuarios con *Base64* y un archivo de usuarios y contraseñas, sobre ***Apache-Tomcat 3x***. En las versiones de ***Tomcat 3x*** y ***4x*** se configuraba *server.xml* para indicar cuál era la aplicación web (**videocamara**, en un directorio del mismo nombre), mientras que los archivos *class* de los ***servlets*** se colocaban manualmente en un sub-directorio de *WEB-INF*; los ***JSPs*** se colocaban también manualmente en un sub-directorio de **videocamara**. Este módulo fue concluido hasta el último, ya que los módulos de control del puerto paralelo y de video tenían mayor prioridad.

El código de cada módulo se comenzó a organizar en paquetes. *RUP* permite el desarrollo iterativo, lo que significa que se puede regresar a fases anteriores, para terminar de detallar o redefinir partes de los módulos; también señala que hay que darle mayor prioridad a partes más críticas del proyecto, dejando al último las tareas más sencillas o que consumen más tiempo. Algunas clases de este sistema se crearon con editor de texto, y se compilaban y probaban en una ventana de consola, pero para poder acelerar el desarrollo y depuración de código se comenzaron a usar ambientes de desarrollo integrado, el primero era *JBuilder*, y más adelante se cambió por *JDeveloper*.

El desarrollo del módulo del puerto paralelo, utilizándolo como aplicación de consola en esta etapa, no tuvo sobresaltos, ya que se enviaban datos numéricos por dicho puerto de forma exitosa, requiriendo pausar el proceso entre cada envío de dato, para garantizar que el circuito electrónico respondiera adecuadamente ante cada dato recibido; se utilizó el método *Thread.sleep(int tiempo)* para detener el proceso a intervalos regulares, y no contadores

con decremento, debido a que el lapso de pausa varía de acuerdo con la velocidad del microprocesador. La forma en que trabaja la clase **Paralalelo16.java** para el control y comunicación con el puerto paralelo de la computadora es la siguiente:

1. Se crea una instancia de **Paralelo16.java**.

2. Se llama a *public synchronized void abrePuerto()*.

   2.1 Obtiene los puertos locales disponibles.

   2.2. Busca dentro de los puertos encontrados los que sean de tipo paralelo, y cuyo nombre sea "**LPT1**".

   2.3 Abre el puerto paralelo "**LPT1**", registrando a "**Paralelo16**" como la aplicación que usa el puerto.

   2.4 Abre un flujo de salida (*OutputStream*) del puerto.

3. Llama al método enviar (tantas veces como se desee), el cual transmite un entero por el flujo de salida.

4. Finalmente, se cierran los flujos de puerto, y el puerto mismo.

Como se había indicado, *RUP* aconseja darle mayor prioridad a las partes más críticas. El módulo de video fue el que requirió mayor esfuerzo y diversas pruebas para controlar el dispositivo, y luego obtener una imagen fija. Al instalar la interfaz ***Java Media Framework***, también se instaló una herramienta llamada *Java Media Player*, la cual se utilizó como herramienta de diagnóstico para verificar que la videocámara y la interfaz JMF estuvieran instaladas correctamente, pero no se usó como parte del sistema, ni se aplicó ingeniería inversa para revisar el código de la misma. Se revisaron ejemplos del uso de ***JMF*** pero ningún ejemplo resolvía el problema de obtener una imagen fija, por lo que se revisó la documentación y tutoriales con los cuales se entendió cómo trabaja la interfaz.

Después de hacer varios códigos de prueba para activar la videocámara, obtener video, detenerla y cerrar el medio, quedaba el problema de obtener una imagen (correspondiente a un cuadro de video) en un formato conocido que pudiera desplegarse en una página ***HTML***. En las pruebas iniciales se había encontrado que el sistema operativo (*Windows Millenium*) usaba *Video for Windows*, y la videocámara utiliza el formato ***AVI***. Se hicieron unas clases para transmisión y recepción de video en formato ***RTP*** (***Real Transfer Protocol***), y de ahí obtener una imagen fija en un formato conocido; este formato tiene menor resolución, y se encontró el mismo problema de que no se podía obtener una sola imagen con los métodos ya existentes.

El problema es que se podía iniciar y detener el proceso de toma de video por parte de la videocámara, y una vez iniciado el proceso de grabación (previamente configurado), no se tiene el control directo para manipular el video obtenido; considerando que el video es una secuencia de cuadros de imágenes, se buscó la forma de obtener uno de ellos del video de forma programática. La videocámara y la tarjeta de video usan Video para *Windows* (***VfW***, ***Video for Windows***) para obtener imágenes; la interfaz ***JMF*** puede usar el formato ***API***, por lo que se buscó la manera de que buscara una fuente de video de tipo Video para Windows, y que el video obtenido lo convirtiera a formato ***AVI*** (que es un formato contenedor), y de ese formato extraer la pista de video.

Una vez extraído el video, se implantan ***códecs*** propios que reciben un bloque de datos binarios sin formato (crudos) del flujo de video, que corresponden a un cuadro de video, se les da el formato ***JPEG***, y se guardan en un archivo en disco duro; debido a que no se podía calcular cuándo se tenía una imagen capturada y guardada, ni un

control preciso sobre el proceso de captura de imágenes una vez iniciado, se implantó la generación de un evento y su auditor correspondiente, para avisar que se tiene una nueva imagen capturada y guardada, y detener el proceso de grabación del video, para evitar sobre-escribir la imagen ya obtenida.

El código del módulo de video que implanta la funcionalidad descrita en el párrafo anterior, la efectúa la clase **CuadroVideo05.java**, la cual lleva a cabo los pasos básicos para utilizar un dispositivo multimedia con la interfaz ***Java Media Framework***. Dichos pasos son los siguientes:

1. Busca un dispositivo de multimedia que corresponda con el indicado en el archivo de propiedades - uno que maneje *Video for Windows*.
2. Se obtienen los formatos que maneja el dispositivo multimedia.
3. Obtiene una instancia de *MediaLocator*, el cual es equivalente a una *URL*, y sirve como referencia para tener acceso a la fuente multimedia.
4. Llama al método ***creaProcesador***, el cual utiliza a la instancia de ***MediaLocator*** para invocar al administrador(***Manager***) y crear una fuente de datos (***DataSource***); el administrador es el componente que se encarga de llamar a las librerías nativas para comunicarse con el sistema operativo, y tomar el control del hardware de un dispositivo multimedia. El objetivo de este método es crear un procesador y no un reproductor porque es más difícil desplegar el reproductor (que tiene interfaz gráfica propia) en una página web, y que tenga interacción de forma remota con el dispositivo multimedia; por otra parte, un procesador tiene los controles para manipular al dispositivo multimedia (invocados desde el ***servlet***), y si bien carece de interfaz gráfica, el ***JSP*** (creado posteriormente) le proporciona una.
5. A partir de la fuente de datos, siguiendo dentro de ***creaProcesador***, se crea un procesador, el cual se encarga de obtener los datos multimedia del dispositivo, y darles formato.
6. El procesador pasa del estado **Sin Realizar** (estado inicial) al de **Configurado**, pasando por el de **Configurando**.
7. Se obtienen las pistas multimedia del procesador; para la videocámara se obtienen dos pistas, una de audio y otra de video.
8. De las pistas obtenidas, se revisa cuál de ellas corresponde a Video, y se procede a darle formato, el cual tiene base en un formato de ***JPEG*** al cual se le indica la velocidad de encuadre (cuántos cuadros de video toma por segundo), y el alto y ancho de la imagen a obtener (tomando en cuenta las limitaciones del formato ***AVI***).
9. Se crea una instancia de **AccesoPosteriorCodec**, la cual es una clase interna que extiende **AccesoPrevioCodec**, la cual implanta la interfaz *códec*. El objetivo de **AccesoPosteriorCodec** es recuperar el flujo de datos que corresponde a un cuadro de video recién tomado de la videocámara, descomprimir los datos dándoles el formato ***JPEG***, guardar esos datos formateados en un archivo con extensión ***JPEG***, y generar un evento de tipo **EventoImagenGenerada** para avisar que se obtuvo un cuadro de video de la videocámara y que se guardó en disco duro; ese evento y la implantación de su correspondiente auditor (**Listener**) fueron diseñados específicamente para este caso, ya que la videocámara, una vez que inicia, sigue un proceso independiente que toma imágenes continuamente.
10. Se especifican los *códecs* de entrada y salida de datos (con base en **AccesoPosteriorCodec**, creado en el paso anterior), y se encadena a con los otros *códecs* que maneja la pista de video.
11. Se le da la descripción del tipo de media al procesador -en este caso, ***CONTENT_UNKNOWN***, porque no se puede saber con certeza que formatos de video maneje la pista de video.
12. Llama al método ***esperaEstado***, y se espera hasta el que procesador pase al estado de **Realizado**.
13. Especifica un factor de calidad ***JPEG*** con el que se generarán las imágenes.
14. Obtiene una instancia de **DataSource** del procesador ya realizado.
15. Sale del método ***creaProcesador*** regresando verdadero.
16. Llama al método continuar, el cual revisa que el procesador haya sido construido correctamente, y lo arranca para que comience a tomar imágenes.

Por otra parte, la clase **CuadroVideo05.java** cuenta con varios atributos booleanos que indican el estado del procesador, de la fuente de datos y si ha conseguido una nueva imagen; cuenta con un atributo de tipo *File*, el cual indica ruta y nombre del archivo generado, en el cual se guardó una nueva imagen. Así mismo, cuenta con varios métodos con los cuales se manipula a la videocámara, los cuales se explican a continuación:

- *public void controllerUpdate(ControllerEvent ce)*, este método realiza la interfaz ***ControllerUpdateListener***, sincronizando el código de la aplicación con los procesos de las librerías nativas que controlan directamente los dispositivos multimedia, asegurando que el procesador pase de un estado a otro correctamente.

- *public boolean esperaEstado(int estado)*, similar al anterior, pero la diferencia consiste en que el método anterior responde a eventos, arrojados por el mismo procesador, mientras que este responde a una solicitud de cambio de estado (identificado por un entero).

- *public synchronized boolean detener()*, el cual termina la obtención de imágenes del procesador.

- *public synchronized boolean continuar()*, el cual reinicia con la obtención de imágenes de la videocámara.

- *void setCalidadJPEG(Player p, float valor)*, el cual asigna un factor de calidad que esté en el intervalo (0, 1), tomando en cuenta cuáles formatos de salida son del tipo ***JPEG***.

- *private synchronized boolean esperaEstado(Processor p, int estado)*, similar a los mencionados anteriormente, pero recibe como parámetros la instancia de procesador, y el estado al cual pasa; ayuda a la transición de estados, sincronizando las tareas del procesador con el código de esta clase.

- *public void imagenCreada(EventoImagenCreada recibeEvento)*, el cual implanta la interfaz **ImagenCreadaListener**, con la cual se auditan (o esperan) los eventos de tipo **EventoImagenCreada** que arroja la clase **AccesoPosteriorCodec** (que se ejecuta en un proceso independiente al de **CuadroVideo05**).

- *public synchronized void cerrar()*, el cual detiene a la videocámara, cerrando los flujos del mismo, y liberando los recursos multimedia del dispositivo.

Los módulos de control del puerto paralelo y de video se desarrollaron (en una fase intermedia) y probaron como si fueran aplicaciones de consola (mono-usuario), pero para integrarse en el ambiente de web (multi-usuario) se encontró el problema de concurrencia hacia recursos únicos, en este caso el puerto paralelo y la videocámara de estado sólido. Se regresó a la etapa de diseño, para resolver este problema; una posible solución era sincronizar los métodos que utilizaran el puerto paralelo o la videocámara. La otra solución era el modelo o patrón **Singleton**[39, 49], el cual garantiza que existe una sola instancia de dicha clase (mediante un constructor privado) en la máquina virtual de java, lo que permite representar con mayor exactitud al puerto paralelo y a la videocámara (ambos son objetos físicos y únicos en la computadora personal); al crear una sola instancia permite que inicie sólo una vez, sin tener que configurar nuevamente la clase, y también permite que se pueda invocar desde cualquier parte de la aplicación web.

Para no re-escribir código también se aplica el concepto de herencia, para crear las clases **PuertoParalelo.java** y **VideoCamara.java**, las cuales extienden a las clases **Paralelo16.java** y **CuadroVideo05.java**, y los métodos nuevos o los sobre-escritos están sincronizados para resolver el problema de concurrencia. Estas clases se probaron nuevamente como aplicaciones de consola, trabajando correctamente, por

lo que se procedió a construir los elementos de web del modelo ***MVC*** para invocar a ***PuertoParalelo*** y a ***VideoCamara***.

Se creó el ***servlet*** **ControlPuertoParalelo.java**, su correspondiente ***JSP*** y unos *JavaBeans* para guardar datos de la posición relativa de la videocámara, sobre *Apache-Tomcat 4.x*, en la cual sólo se indicaba el nombre de la aplicación web, mientras que los archivos compilador (*class*) se colocaban manualmente en una ruta de la aplicación web (sub-directorio de *WEB-INF*), mientras que el *JSP* se colocaba en otro sub-directorio de la misma aplicación. Se comenzó a tener problemas entre ***Apache-Tomcat*** y la interfaz ***Java Communications***, hasta que las librerías de este última y su archivo de propiedades se colocaron dentro de las librerías externas del ***JRE*** (***Java Runtime Enviroment***), probando que se controlara el envío de datos por el puerto paralelo desde un ambiente de web.

La incorporación del módulo a la aplicación web presentó problemas adicionales, ya que si bien la clase **VideoCamara.java** trabajaba correctamente como aplicación de consola, dentro del ambiente de web no funcionaba correctamente al inicio. Las librerías de la interfaz ***JMF*** se habían colocado en el sub-directorio *lib* de ***Tomcat***, en *JAVA_HOME/lib*, *JRE/lib* (directorios de instalación del compilador y del ambiente de java, respectivamente) y en el directorio *WEB-INF/lib* de la aplicación web **videocamara**, se activaba y desactivaba la cámara, pero no se obtenía ninguna imagen; se agregaron los archivos de propiedades y de configuración de ***JMF*** al directorio *WEB-INF\lib* de la aplicación web, y se comenzó a obtener imágenes apareciendo otros problemas de sincronización de la clase *VideoCamara.java* con el ***servlet*** **controlvideocamara.java**, debido a que este último esperaba indefinidamente, o respondía inmediatamente enviando un *JavaBean* sin datos al ***JSP*** **controlImagen.jsp**.

Mediante prueba y error se terminó de modificar la clase **VideoCamara.java**, para intentar hasta tres veces la captura de la imagen, y en caso de no obtenerla, el ***servlet*** **controlvideocamara.java** presentaría otro ***JSP*** con un mensaje de error. Se decidió juntar en un sólo ***JSP*** la presentación de la imagen capturada, datos de la misma y los controles de cambio de posición de la videocámara, para que los usuarios puedan manipular ese dispositivo, recibiendo la misma página pero con la nueva imagen en la nueva posición y con datos actualizados. La funcionalidad del control se concentró en **ControlDireccionPuerto01.java**, ya que en cada cambio de posición se debe capturar y presentar una nueva imagen; se siguieron utilizando los *JavaBeans* **Punto.java**, **Coordenada.java**, **Direccion01.java** y **DatosImagen.java**, ya que representan los cambios de posición, la posición actual y los datos de la imagen recién capturada.

Estos ***servlets*** y ***JSPs*** se probaron y ejecutaron en ***Apache-Tomcat 4x***, colocando los archivos compilados (***servlets***, *JavaBeans* y archivos auxiliares) en sub-directorios de *WEB-INF*, mientras que los ***JSPs*** se colocaban en otros sub-directorios de la aplicación web. En la arquitectura de esta versión de ***Tomcat*** los ***servlets*** se actualizan de forma automática al cambiar el archivo ***class***, mientras que las clases estáticas se cambian al reiniciar el contenedor de ***servlets***. Mientras que la aplicación se desarrollaba y crecía, apareció el problema de tener actualizadas las clases presentes en ***Tomcat***, por lo que se creó un archivo *WAR* (*Web ARchive*) para copiar todas las clases actualizadas, ***JSPs***, archivos de propiedades, *HTMLs* e imágenes fijas, del ambiente de desarrollo a ***Tomcat***. También permitió copiar las librerías y archivos de propiedades de las interfases externas utilizadas (***Java Communications***, ***Java Media Framework*** y *J/Connector*) al sub-directorio *WEB-INF\lib*, para que en el caso de tener que actualizar alguna de estas librerías, se volviera a generar el archivo *WAR* y no tener que cambiar la configuración de ***Tomcat***. Esta tarea se efectuó de forma automática con el entorno de desarrollo integrado (*JDeveloper* y/o *NetBeans*), el cual crea un archivo con el mismo tipo de compresión que los archivos *ZIP*, con la estructura de la aplicación web.

El código desarrollado y probado hasta ese entonces se había hecho en ***Apache-Tomcat 4x***, sobre el sistema operativo *Windows Millenium*, pero por diverso problemas con dicho sistema operativo se tuvo que cambiar a *Windows XP*, el cual se presentó como una plataforma más robusta y segura que las versiones anteriores de *Windows*.

Cuando se migró la aplicación de *Windows Millenium* a *Windows XP* con *Service Pack 1*, se cambió el nombre con que se identifica al Video para *Windows*, por lo que cambió el nombre y se re-compiló la aplicación web, funcionando, hasta ese momento, como lo hacía anteriormente.

También se cambió la versión de ***Apache-Tomcat*** a la 5 (y más adelante a la 6), para aprovechar las mejoras de seguridad, por lo que se cambió la forma de conexión a base de datos creando una fuente de datos reemplazando a las clases **ConexionGeneralBD.java** y **PozoConexiones.java**; esta fuente de datos se utilizaría para la aplicación web como para implantar un reino de seguridad.

Para que el sistema tuviera interacción con la base de datos se regresó a la etapa de diseño, buscando cómo guardar y recuperar datos más fácilmente; ya se había probado que el sistema web podía efectuar operaciones ***DML*** con *MySQL* a través de ***JDBC***, pero se tendría que programar mucho código relacionado con base de datos en ***servlets***, y pasarle datos a los ***JSPs***. Se encontró que la arquitectura ***DAO*** (***Data Access Object***)[48] podría ayudar.

Se crearon objetos ***DAO***, los cuales son clases simples del lenguaje java, pero contienen las sentencias ***DML*** (consultas, borrados, inserciones, modificaciones) para *MySQL*; se movió el código de algunos métodos de los ***servlets*** a los objetos ***DAO***, con el objetivo de que ningún ***servlet*** efectuara consultas a la base de datos directamente, sino que enviaran o recibieran objetos de transferencia a o desde de un objeto ***DAO***, y este último se encargue de efectuar la consulta. Los ***JSPs*** tampoco harían consultas a la base de datos, sino que recibirían *JavaBeans* o vectores con *JavaBeans*, para desplegar información. Los *JavaBeans* ya existentes y los nuevos se utilizaron como objetos de transferencia. Se modificó la clase **OperacionesGenericasMySQL.java**, para encontrar la fuente de datos y obtener de ella una conexión hacia *MySQL*; los objetos ***DAO*** prepararan las sentencias *SQL*, incluyendo sentencias almacenadas (*Stored Procedures*), e invocan a los métodos de **OperacionesGenericasMySQL.java**, los cuales efectúan las consultas, borrados, modificaciones e inserciones, incluyendo los procedimientos almacenados (para sentencias *SQL* repetitivas).

Los objetos de transferencia, que son *JavaBeans*, se construyeron manualmente creando uno por cada tabla de la base de datos; cada uno tiene un atributo por cada columna de la tabla; existen herramientas que efectúan esta tarea de forma automática. Algunas de estas clases se emplean como *JavaBeans* en el modelo ***MVC***[49] implantado (o extendiendo el modelo hacia ***Servicios al Trabajador***[50]).

Para guardar o recuperar imágenes de la base de datos *MySQL* se utiliza un procedimiento almacenado que maneja un arreglo de bytes para almacenar o extraer información de un campo de tipos *blob*; se encontró que otras sentencias *SQL*, que no estuvieran almacenadas, no guardan ni recuperan información binaria correctamente del campo *blob* de *MySQL*.

La clase **OperacionesGenericasMySQL.java** tiene métodos invocados por los objetos de tipo ***DAO***, recibiendo objetos de transferencia (con los datos), los tipos de datos (en un arreglo) y una cadena con el procedimiento almacenado a efectuar; se crea el procedimiento almacenado, los datos se extraen de los objetos de transferencia y junto con el tipo de dato indicado, se efectúa la sustitución de parámetros. Después de revisar todos los objetos de transferencia se efectúa el procedimiento almacenado.

Conceptualmente, la parte más difícil del módulo de base de datos, fue crear métodos genéricos para que pudiera manipular cualquier conjunto de objetos de transferencia recibidos. La lógica implantada en los objetos ***DAO*** es relativamente sencilla, pero consumió mucho tiempo, debido a que se tiene que concatenar cadenas para preparar las sentencias ***DML***, o simplemente para asignar o recuperar datos de los atributos de los objetos de transferencia. Este módulo se hubiera podido hacer más rápida y fácilmente si se hubiera hecho con una base orientada a objetos, ya que en este última se guardan directamente los objetos, sin tener que concatenar cadenas y/o manipular los atributos de los objetos.

La última parte construida fue la implantación de la seguridad y de un filtro de peticiones -para permitir o desviar solicitudes a la página de control de la videocámara. Se regresó a la fase de diseño, para cambiar el modelo Entidad-Relación, modificando en *MySQL* las tablas **Usuarios** y **Niveles**, y creando otra tabla dependiente de las anteriores llamada **Usuarios-Niveles**, con el fin de implantar un reino de seguridad con autenticación con base en ***JDBC***; se modificaron los guiones de creación de tablas de la base de datos, y en el guión de inserción de datos se creó un usuario por defecto, el "Super-Usuario" ***SU***, con el cual entrar al sistema por primera vez, y autorizar a los nuevos usuarios.

La tabla **Usuarios** conserva los campos *nombre_usuario* y *contraseña*, pero se le quita el campo *id_nivel*, para que en la tabla *usuarios-niveles* se aprecie cuáles usuarios tienen un nivel, y cuáles no. La tabla **niveles** se cambia, para tener una sola columna con la descripción de los niveles existentes para los usuarios, "SU" de super-usuario, "Administrador" y "Usuario", con permisos más reducidos.

Una vez modificada la base de datos, se configuró ***Apache-Tomcat*** para crear el reino de seguridad con base en ***JDBC***, delegando a ***Tomcat*** el control de acceso a las partes privadas de la aplicación; el contenedor de ***servlets*** también permite indicar cuáles páginas pueden ser mostradas sin pedir usuario y contraseña, y cuales sí requieren que se autentifique al usuario. A diferencia del resto del código de la aplicación web -que pude funcionar en otros servidores de aplicaciones que le den respaldo a la tecnología java- estas configuraciones son específicas para el contendedor ***Apache-Tomcat***, por lo que si se requiere tener control de acceso en un servidor de aplicaciones, se tendría que estudiar la configuración propia de dicho servidor, la cual puede ser igual (para los servidores con base en ***Tomcat***) o similar.

La configuración del reino de seguridad [43, 49] en ***Tomcat*** requiere lo siguiente:

1 El archivo *JAR* de *Connector/J* en *<directorio_tomcat>/lib*, para que el contenedor (y no sólo la aplicación web) lo pueda encontrar.
2 Indicar en *web.xml* los nombres de las tablas de usuarios y papeles.
3 Se creó ***acceso.jsp***, el cual tiene una forma *HTML* con nombre especial...
4 Se creó la página de error ***error.jsp***, la cual es mostrada cuando falla la autenticación del usuario.
5 Se configuró *web.xml* para indicar cuáles recursos web requieren de control de acceso.

Para autorizar a un nuevo usuario o administrador, se verifica las credenciales del usuario en sesión, y si su nivel es mayor que el solicitado para el nuevo usuario, continúa con las validaciones de los demás datos.

Se creó un filtro de peticiones hacia la página de la videocámara, para desviar las peticiones, hacia dicha página cuando se requiera mantenimiento o calibración sin tener que detener toda la aplicación. Se modificó en *web.xml* lo siguiente:

```
<filter>
   <filter-name>RedirigirVideoCamara</filter-name>
   <filter-class>filtro.AuthorizationFilter</filter-class>
</filter>
<filter-mapping>
     <filter-name>RedirigirVideoCamara</filter-name>
     <url-pattern>/controlDireccionPuerto/*</url-pattern>
</filter-mapping>
<filter-mapping>
     <filter-name>RedirigirVideoCamara</filter-name>
     <url-pattern>/jsp/controlDireccion.jsp</url-pattern>
```

</filter-mapping>

La etiqueta *filter* indica se define un filtro de peticiones hacia las *URLs* definidas dentro de *filter-mapping* en las etiquetas *url-pattern*. La clase **AuthorizationFilter** (siempre debe tener dicho nombre) revisa las peticiones, para permitir que se muestre la pantalla del control de dirección, o re-dirigir la petición *HTTP*.

Algunos ***servlets*** y ***JSPs*** utilizan la interfaz *Log4j* para reportar a bitácora; esta interfaz permite reportar distintas acciones en un archivo de forma uniforme, y según la jerarquía de la acción, evento o excepción a reportar, desde nivel de depuración hasta errores severos. En las aplicaciones web es recomendable tener una bitácora, para guardar la información de los hechos relevantes ocurridos (transacciones, errores, inicio y fin de sesiones); sin embargo, para esta aplicación web fue más útil la depuración en tiempo real con *NetBeans*.

Para facilitar la colocación de la aplicación web en ***Tomcat***, se crea un archivo *WAR* (*Web ARchive*), generado en *JDeveloper* o en *NetBeans*, con lo que se evita pasar elemento por elemento (clases, ***JSPs***, archivos *JAR* o *HTMLs*). El archivo *WAR* se genera de forma automática, y aunque existen otras herramientas para generarlo (como *ANT*), no se explicará cómo se crea; basta mencionar que es un archivo comprimido, creado con el mismo algoritmo de compresión que los archivos *ZIP*. Este archivo se copia en el sub-directorio *webapps* de ***Apache-Tomcat***, el cual se encarga de descomprimirlo en el directorio de la aplicación web.

Durante el desarrollo y al hacer pruebas, se definieron y/ modificaron las siguientes variables de ambiente, de las cuales también se muestra su valor correspondiente:

- **JAVA_HOME:** *C:\Java\jdk1-6-0_17*
- **PATH:** *%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;%JAVA_HOME%\bin;C:\MySQL\MySQL_Server_5-1\bin*
- **CLASSPATH:** *.;C:\Java\JavaAPI\commapi\comm.jar*

La variable de ambiente **JAVA_HOME** señala dónde está instado el ***JDK*** del lenguaje java. La variable **PATH** indica las rutas a los ejecutables de las aplicaciones, en este caso, del ***JDK*** de java, y del administrador de *MySQL*.

La configuración mínima para ejecutar la aplicación web es la siguiente:

- *Windows 95*, *98*, *Millenium*, o *Windows XP*.
- Computadora con puerto paralelo ***IEEE 1284***.
- *Java 1.5* o superior; los módulos de control de puerto paralelo y de video trabajan con *Java 1.2,* pero el código del módulo de administración y de base de datos incorporan el concepto de *genéricos* [15].
- ***Apache-Tomcat 5.x*** en adelante, en conjunto con un ***Java Runtime Enviroment*** versión 1.5
- Administrador de base de datos *MySQL 3* en adelante.
- Interfaz ***Java Communications 2.0*** para *Windows*, instalada en el ***JRE*** o en el ***JDK*** que use ***Apache-Tomcat***.
- Interfaz ***Java Media Framework 2.1.1.e*** con *Performance Pack* para *Windows*.
- Interfaz *J/Connector 4* en adelante.
- Interfaz *Log4j 1.2* en adelante.

2.3.2 Construcción del circuito de control.

La construcción del circuito de control comenzó una vez que se tenía el diseño del circuito electrónico y se habían conseguido las piezas del mismo; debido a que se alambran componentes físicos, se tuvo que seguir el diagrama del circuito.

Originalmente el circuito empleado para las pruebas de desarrollo era un circuito formado por *buffers* de salidas de tres estados SN74LS241N (para aumentar la señal del puerto paralelo), y circuitos de tipo *latch* SN74LS373N (para retener el valor del bit proveniente del *buffer*), cuya salida se despliega en diodos emisores de luz *(leds*) -con unas resistencias de por medio, para reducir la corriente. Este circuito usaba lógica *TTL* de 5 [V], alimentado por una pila de 6 [V].

Una vez que se probó exitosamente que se podían enviar datos por el puerto paralelo a través de la interfaz web, se comenzó a diseñar un circuito electrónico que pudiera mover motores de paso, para mover la videocámara. Se probó primero un circuito de configuración de medio paso, encontrándose que no era adecuado para el movimiento deseado, por lo que se descartó esta configuración.

Sin embargo, con base en la experiencia anterior, se diseñó otro circuito de paso completo utilizando los mismos componentes, el cual se muestra en la figura 24; al terminar de diseñar el nuevo circuito con la configuración de paso completo, se alambraron los componentes, siguiendo el nuevo diseño. Este circuito permite acoplar la lógica *TTL* (de 5[V]) del puerto paralelo de la computadora personal con la parte de potencia que mueve los motores de paso (de 12 [V] y de mayor corriente). La etapa de potencia de este circuito se alimenta con un eliminador de 12 [V] y 1000 [mA].

2.4 Pruebas de la Aplicación.

Las pruebas de la aplicación se hicieron, primero por módulo, y a lo largo de la fase de construcción; se evitó programar grandes bloques de código y luego probarlos, sino construir elementos más pequeños, y luego probarlos uno por uno. Luego, estos elementos (clases) se probaron junto con el resto del módulo, y más adelante se probó la aplicación al integrar los módulos.

Los módulos del puerto paralelo y los de video se desarrollaron y probaron de forma independiente, como si fueran aplicaciones de consola, enviando datos por el puerto paralelo y obteniendo una imagen de la videocámara, respectivamente, probándolos en el sistema operativo *Windows Millenium*.

Al integrar ambos módulos en web, se requirió de regresar a la fase de diseño, a desarrollar nuevos códigos, y por lo tanto, de efectuar nuevas pruebas en el ambiente de web. Como resultado de las mismas se ajustaron algunos valores usados como constantes, así como otros de configuración, y se hicieron algunos ajustes menores a la aplicación. Estas pruebas se efectuaron usando ***Apache-Tomcat***, también en ***Windows Millenium***.

Debido a problemas con el sistema operativo *Windows Millenium*, se cambió el sistema operativo por *Windows XP* con *Service Pack 1*; después de instalar las aplicaciones y rehacer el ambiente de desarrollo de esta aplicación, se efectuaron pruebas nuevamente, encontrando que el nombre del video variaba ligeramente de *Windows Millenium* a *Windows XP*, por lo se cambió el nombre de la fuente de video a utilizar. Después de dicho cambio se efectuaron pruebas para verificar a través de web que siguiera obteniendo imágenes de la videocámara y siguiera enviando datos por el puerto paralelo, lo cual lo hacía exitosamente hasta ese momento.

El resto de la aplicación, el módulo de administración, el cual integra base de datos y web, se terminó de desarrollar sobre *Windows XP*; conforme se desarrollaba este módulo, también se aplicaban "parches" de seguridad

a *Windows XP*, y más adelante se instalaron el "*Service Pack 2*" y el "*Service Pack 3*" en *Windows XP*, debido a que prometían mejoras de seguridad de utilidad del sistema operativo, pero no mencionaban explícitamente que cambian la forma en que *Windows XP* administra los periféricos.

Estos cambios no se apreciaron de inmediato, debido a que se estaba construyendo código relacionado con base de datos, y además no se tenía todo el tiempo el circuito de control conectado y energizado. Cuando se cambió la versión de ***Tomcat*** 4 a la 5, se probó nuevamente el código de la aplicación, encontrando que variaba el comportamiento del puerto paralelo. Se había creído que dicho comportamiento era causado por datos que se habían quedado en la memoria reservada para el puerto paralelo después de haber hecho uso de la impresora; más adelante se apreció que se debía a la instalación de uno o varios parches para *Windows XP*, y no por datos que se hayan quedado en memoria, ni por el modo del puerto paralelo elegido en la configuración de *BIOS*. Se verificó que en el *BIOS* estuviera seleccionado el modo *ECP* (*Extended Capability Port*) para el puerto paralelo, el mismo modo con que *Windows XP* trabaja al puerto paralelo (aunque lo llama puerto de impresión).

Durante este periodo de pruebas también se estaba diseñando y probando el circuito de control de movimiento, por lo que se volvieron a ejecutar programas de java que se había hecho para el control del puerto paralelo; se revisó que todos los elementos de la interfaz *Java Communications* estuvieran en las rutas correctas y la variable de ambiente **CLASSPATH** incluyera la ruta hacia *comm.jar* . Se encontró que para funcionara correctamente esta interfaz (como cuando se usaba en *Windows Millenium* o *Windows XP* con *Service Pack 1*), se tenía que efectuar lo siguiente:

1 Inhabilitar el puerto paralelo en el Administrador de Dispositivos del Sistema de *Windows XP*.
2 Conectar y energizar el circuito de control.
3 Reiniciar *Windows XP*.
4 Habilitar el puerto paralelo.

Mediante los pasos anteriores se comprobó que se podía utilizar la interfaz *Java Communications* para enviar datos por el puerto paralelo tal y como se había hecho en *Windows 95*, *98* y *Millenium*. También se trató de obtener datos del puerto paralelo, utilizando el circuito de pruebas que usa circuitos tipo *latch*, pero se encontró que esta interfaz lanza la excepción *Unsupported operation. Output only mode* al tratar de cambiar el modo del puerto paralelo de *SPP* a *ECP* o a *EPP*, por lo que tampoco se pudo crear el flujo de entrada, ni recibir datos, a pesar de que la documentación de esta interfaz indica que sí se puede hacer.

Se efectuaron otras pruebas para verificar si la interfaz *Java Communications* respondía mejor en *Windows XP*, tal como habilitar el "*Plug and Play*" heredado en el controlador del puerto paralelo, pero al tratar de usar el puerto la excepción "*PortOwnershipException*" era arrojada, aunque no hubiera otra aplicación usando el puerto paralelo, por lo que no se podía hacer uso del puerto.

Se habilitó la opción "Usar cualquier interrupción asignada al puerto" en "Método de filtrado de recursos" de la configuración de puerto; al cambiar la configuración del puerto paralelo, la interfaz *Java Communications* vuelve a enviar datos correctamente. Si se conserva la configuración del puerto paralelo, y se reinicia el equipo, se tiene que inhabilitar y habilitar el puerto paralelo en el "Administrador de dispositivos", para que *Java Communications* envíe datos correctamente por dicho puerto.

También se trató de registrar la librería dinámica *win32com.dll* en *Windows XP* ejecutando el comando *regsvr32*, ejecutándolo de la siguiente forma:

regsvr32 /n /i: win32com.dll

Sin embargo, el sistema operativo envió el mensaje de advertencia "Se descargó *win32comm.dll*, pero no se encontró el punto de entrada *DllInstall*. No se puede registrar este archivo". Esto se debe a que *Windows XP* no lo reconoce como una librería dinámica válida, o porque *win32com.dll* requiere de otra librería dinámica (que no indica en ninguna parte de la documentación).

Se efectuaron pruebas secuenciales y recurrentes al ***servlet*** que controla la posición y proporciona la página web con la imagen solicitada de la videocámara; estas pruebas se efectuaron solicitando dicho ***servlet***, tal y como lo haría *Internet Explorer* o *Firefox*. Las primeras pruebas fueron las secuenciales, de las cuales se tomó el tiempo de inicio de la petición, el tiempo de fin, calculándose la duración de la misma; los resultados de la misma se muestran sus resultados a continuación:

Pruebas de tiempo de respuesta a solicitudes de URL.	**Datos de las pruebas.**
Línea de comando con los parámetros de ejecución.	C:\Java\jdk1-6-0_14\bin\javaw.exe -client -classpath C:\Proyectos\Java\PruebasAccesosSimultaneos\PeticionesURL\PeticionesSimultaneas\classes peticionessimultaneas.PruebaSolicitudesDistintas
	URL: http://192.168.0.10/videocamara/controlDireccionPuerto?direccionActual=1 Tiempo inicio solicitud: Tue Sep 29 22:45:49 CDT 2009 Tiempo inicio solicitud: Tue Sep 29 22:46:15 CDT 2009 Lapso solicitud: 25782 [ms]
	URL: http://192.168.0.10/videocamara/controlDireccionPuerto?direccionActual=1 Tiempo inicio solicitud: Tue Sep 29 22:46:15 CDT 2009 Tiempo inicio solicitud: Tue Sep 29 22:46:21 CDT 2009 Lapso solicitud: 5516 [ms]
	URL: http://192.168.0.10/videocamara/controlDireccionPuerto?direccionActual=2 Tiempo inicio solicitud: Tue Sep 29 22:46:21 CDT 2009 Tiempo inicio solicitud: Tue Sep 29 22:46:26 CDT 2009 Lapso solicitud: 5734 [ms]
	URL: http://192.168.0.10/videocamara/controlDireccionPuerto?direccionActual=3 Tiempo inicio solicitud: Tue Sep 29 22:46:26 CDT 2009 Tiempo inicio solicitud: Tue Sep 29 22:46:33 CDT 2009 Lapso solicitud: 6547 [ms]
	URL: http://192.168.0.10/videocamara/controlDireccionPuerto?direccionActual=4 Tiempo inicio solicitud: Tue Sep 29 22:46:33 CDT 2009 Tiempo inicio solicitud: Tue Sep 29 22:46:39 CDT 2009 Lapso solicitud: 6359 [ms]

	URL: http://192.168.0.10/videocamara/controlDireccionPuerto?direccionActual=2 Tiempo inicio solicitud: Tue Sep 29 22:46:39 CDT 2009 Tiempo inicio solicitud: Tue Sep 29 22:46:45 CDT 2009 Lapso solicitud: 5625 [ms]
	URL: http://192.168.0.10/videocamara/controlDireccionPuerto?direccionActual=3 Tiempo inicio solicitud: Tue Sep 29 22:46:45 CDT 2009 Tiempo inicio solicitud: Tue Sep 29 22:46:51 CDT 2009 Lapso solicitud: 6250 [ms]
	URL: http://192.168.0.10/videocamara/controlDireccionPuerto?direccionActual=1 Tiempo inicio solicitud: Tue Sep 29 22:46:51 CDT 2009 Tiempo inicio solicitud: Tue Sep 29 22:46:57 CDT 2009 Lapso solicitud: 5625 [ms]
	URL: http://192.168.0.10/videocamara/controlDireccionPuerto?direccionActual=4 Tiempo inicio solicitud: Tue Sep 29 22:46:57 CDT 2009 Tiempo inicio solicitud: Tue Sep 29 22:47:03 CDT 2009 Lapso solicitud: 5844 [ms]
	URL: http://192.168.0.10/videocamara/controlDireccionPuerto?direccionActual=1 Tiempo inicio solicitud: Tue Sep 29 22:47:03 CDT 2009 Tiempo inicio solicitud: Tue Sep 29 22:47:08 CDT 2009 Lapso solicitud: 5328 [ms]
Tiempos inicial y final de las pruebas.	Terminó las pruebas. Tiempo de inicio de las pruebas: Tue Sep 29 22:45:49 CDT 2009 Tiempo de fin de las pruebas: Tue Sep 29 22:47:08 CDT 2009 Tiempo total de pruebas: 78640[ms] Process exited with exit code 0.

Tabla 18: Pruebas secuenciales de la página de la videocámara.

De los datos de las pruebas anteriores se puede apreciar que sólo se inició la prueba subsecuente hasta que hubiera terminado la prueba anterior; en cada prueba incluye el inicio de la petición *HTTP*, el lapso en el cual efectúa el movimiento de los motores de paso, la obtención de la nueva imagen, la generación y recepción de la respuesta *HTTP*. También se puede observar que el tiempo de la primera prueba es aproximadamente 4.5 veces mayor que el tiempo de las demás respuestas; esto se debe a que en la primera prueba se activa la videocámara, se abre el puerto paralelo y se pre-compila por primera vez (desde que arrancó ***Apache-Tomcat***) el ***JSP*** con el cual proporciona la respuesta ***HTTP***. Para las pruebas subsecuentes, la videocámara ya ha sido activada, y sólo se reinicia la toma de video y se detiene; el puerto paralelo ya ha sido abierto y disponible para la aplicación, y respecto al *JSP*, ya se había iniciado su ciclo de vida al pre-compilarse, por lo se vuelve a compilar con mínimos cambios respondiendo mucho más rápido.

Se efectuaron otras pruebas, de forma secuencial, con el fin de verificar que atienda las solicitudes de forma sincronizada; la primera solicitud que llegue deberá ser totalmente satisfecha (cambiar la posición de la videocámara

y obtener una imagen), antes de atender solicitudes subsecuentes, para evitar cambios de posición correspondientes a solicitudes más recientes, cuando apenas está obteniendo la imagen correspondiente a la posición solicitada en la primera solicitud. Se hizo una pequeña aplicación de prueba (también con el lenguaje de programación java), la cual crea varios hilos, y cada uno de ellos efectúa una petición distinta de los otros -cambiando los parámetros que corresponden a la nueva posición solicitada-, los cuales inician prácticamente al mismo tiempo; la prueba termina cuando ya no hay hilos activos (cada hilo cumplió con su ciclo de vida). Los resultados de las pruebas se muestran en la siguiente tabla:

Pruebas de tiempo de respuesta a solicitudes simultáneas de URL.	**Datos de las pruebas.**
Línea de comando con los parámetros de ejecución.	C:\Java\jdk1_6_0-16\bin\javaw.exe -client -classpath D:\Proyectos\Java\PruebasAccesosSimultaneos\PeticionesURL\PeticionesSimultaneas\classes -Dhttp.proxyHost=http://99.90.56.56/ -Dhttp.proxyPort=80 -Dhttp.nonProxyHosts= -Dhttps.proxyHost=http://99.90.56.56/ -Dhttps.proxyPort=80 -Dhttps.nonProxyHosts= peticionessimultaneas.PruebaSolicitudesDistintas
	URL: http://192.168.0.12/videocamara/controlDireccionPuerto?direccionActual=1 Tiempo de inicio prueba número 1: Wed Oct 28 22:05:06 CST 2009 Tiempo de fin prueba número 1: Wed Oct 28 22:05:57 CST 2009 Tiempo duración prueba número 1: 50953 [ms].
	URL: http://192.168.0.12/videocamara/controlDireccionPuerto?direccionActual=1 Tiempo de inicio prueba número 2: Wed Oct 28 22:05:06 CST 2009 Tiempo de fin prueba número 2: Wed Oct 28 22:06:00 CST 2009 Tiempo duración prueba número 2: 53922 [ms].
	URL: http://192.168.0.12/videocamara/controlDireccionPuerto?direccionActual=2 Tiempo de inicio prueba número 3: Wed Oct 28 22:05:06 CST 2009 Tiempo de fin prueba número 3: Wed Oct 28 22:06:03 CST 2009 Tiempo duración prueba número 3: 56578 [ms].
	URL: http://192.168.0.12/videocamara/controlDireccionPuerto?direccionActual=3 Tiempo de inicio prueba número 4: Wed Oct 28 22:05:06 CST 2009 Tiempo de fin prueba número 4: Wed Oct 28 22:06:11 CST 2009 Tiempo duración prueba número 4: 64781 [ms].
	URL: http://192.168.0.12/videocamara/controlDireccionPuerto?direccionActual=4 Tiempo de inicio prueba número 5: Wed Oct 28 22:05:06 CST 2009 Tiempo de fin prueba número 5: Wed Oct 28 22:05:46 CST 2009 Tiempo duración prueba número 5: 40218 [ms].

	URL: http://192.168.0.12/videocamara/controlDireccionPuerto?direccionActual=2 Tiempo de inicio prueba número 6: Wed Oct 28 22:05:06 CST 2009 Tiempo de fin prueba número 6: Wed Oct 28 22:06:08 CST 2009 Tiempo duración prueba número 6: 62218 [ms].
	URL: http://192.168.0.12/videocamara/controlDireccionPuerto?direccionActual=3 Tiempo de inicio prueba número 7: Wed Oct 28 22:05:06 CST 2009 Tiempo de fin prueba número 7: Wed Oct 28 22:06:05 CST 2009 Tiempo duración prueba número 7: 59250 [ms].
	URL: http://192.168.0.12/videocamara/controlDireccionPuerto?direccionActual=1 Tiempo de inicio prueba número 8: Wed Oct 28 22:05:06 CST 2009 Tiempo de fin prueba número 8: Wed Oct 28 22:05:51 CST 2009 Tiempo duración prueba número 8: 45312 [ms].
	URL: http://192.168.0.12/videocamara/controlDireccionPuerto?direccionActual=4 Tiempo de inicio prueba número 9: Wed Oct 28 22:05:06 CST 2009 Tiempo de fin prueba número 9: Wed Oct 28 22:05:54 CST 2009 Tiempo duración prueba número 9: 48390 [ms].
	URL: http://192.168.0.12/videocamara/controlDireccionPuerto?direccionActual=1 Tiempo de inicio prueba número 10: Wed Oct 28 22:05:06 CST 2009 Tiempo de fin prueba número 10: Wed Oct 28 22:05:49 CST 2009 Tiempo duración prueba número 10: 42656 [ms].
Tiempos inicial y final de las pruebas.	Terminó las pruebas. Tiempo de inicio de las pruebas: Wed Oct 28 22:05:06 CST 2009 Tiempo de fin de las pruebas: Wed Oct 28 22:06:11 CST 2009 Tiempo total de pruebas: 65000 [ms] Process exited with exit code 0.

Tabla 19: Resultados de las pruebas de solicitudes simultáneas a la página de la videocámara.

De los resultados de la tabla anterior, se puede observar que el inicio de todas las peticiones fue prácticamente el mismo, pero dichas peticiones no llegaron al servidor en el mismo orden en que iniciaron; al llegar la primera petición, el ***servlet ControlDireccionPuerto.java*** abrió el puerto paralelo e inició la videocámara para obtener una imagen; las demás peticiones tuvieron que esperar a que concluyera y se enviara la respuesta a la petición que llegó primero, para que atendiera la segunda, y así sucesivamente. El tiempo final de cada prueba varía, y la duración de cada prueba no corresponde con el orden en que iniciaron las solicitudes; dichos tiempos son relativamente grandes para el tiempo de respuesta de ***servlets*** o ***JSPs***, pero esto se debe a que la primera petición a **ControlDireccionPuerto.java** consume más tiempo en lo que se inicia el ***servlet*** y abre los periféricos; las otras peticiones incluyen el tiempo de la primera solicitud, más el tiempo de las solicitudes que lo anteceden.

También se puede apreciar que la sincronización del ***servlet*** evita conflictos entre peticiones, a costa de la rapidez de respuesta a las solicitudes hechas.

2.5 Mantenimiento.

El primer mantenimiento del sistema fue cuando se cambió el sistema operativo de *Windows Millenium* a *Windows XP*; el sistema no estaba concluido y faltaba de programar el módulo de base de datos, pero se le debe considerar un mantenimiento ya que se revisó la funcionalidad del sistema, haciendo correcciones menores para que siguiera funcionando el sistema y no programarlo todo de nuevo. En este cambio de sistema operativo, se cambió el nombre con que se identifica a la fuente de video.

Posteriormente a esta migración de sistema operativo, hubo otra fase de desarrollo, relacionada con base de datos. Más adelante al cambiar la versión de ***Tomcat*** 4 a la 5, para aprovechar las ventajas de seguridad y aquellas relacionadas con base de datos de la última, se encontró que ***Apache-Tomcat*** requiere que se registren todos los ***servlets*** de la aplicación web en *web.xml*, de lo contrario ignora las peticiones a los ***servlets*** y no ejecuta el código de los mismos; en este caso se tuvo que registrar el nombre calificado de las clases de los ***servlets*** y registrarlos en *web.xml*, pero no se cambió el código de ningún ***servlet***.

Se cambió el administrador de la base de datos *MySQL* de la versión 4.1 a la 5.1, por lo que se hicieron ajustes menores a los guiones de creación de las tablas de base de datos, eliminando el tipo de tabla a crear, pero se mantuvieron iguales los nombres de tablas, columnas y tipos de datos; la versión 5 de *MySQL* permite configurarlo para indicar el tipo de aplicación que va a usar la base de datos, así como el número de conexiones concurrentes estimadas.

Se ha cambiado varias veces la versión del ***Java Development Kit*** y del ***Java Runtime Enviroment***, por lo que se ha tendido que cambiar los valores de las variables de ambiente, y volver a colocar los elementos de la interfaz ***Java Communications*** en las rutas de las nuevas versiones de java.

El último mantenimiento que se le dio a esta aplicación fue cambiar algunos datos usados como constantes y rutas de archivos, dentro del código de la aplicación, por parámetros de inicio de la aplicación web.

3 Resultados.

¡Pero se mueve!
Galileo Galilei.

Al construirse el sistema se comprobó que sí se pudo manipular componentes físicos (*hardware*) a través de Internet, mediante una página web donde se muestra la respuesta de la videocámara (la obtención de la imagen), y donde también se despliegan los controles de posición de la misma (para manipular el puerto paralelo). En este caso, estos periféricos son controlados por ***servlets***, los cuales no tienen restricciones para tener acceso a todos los recursos computacionales del equipo donde se ejecutan -a diferencia de los *applet*s, que sí tienen restricciones para acceder a los recursos de la máquina donde se ejecutan.

Los ***servlets***, en particular la clase **ControlDireccionPuerto01.java**, invocan a las interfases ***Java Media Framework*** y ***Java Communications***, que implantan, a su vez, la interfaz *Java Native Interface*, la cual permite la comunicación entre el lenguaje java y librerías dinámicas (nativas del sistema operativo) escritas en otros lenguajes de computación, que controlan directamente los periféricos.

La combinación de ***servlets***, y el uso (directo o indirecto) de la interfaz *Java Native Interface* (*JNI*) y librerías dinámicas, permite, por un lado, invocar a métodos nativos (que están en el lenguaje binario del sistema operativo) y obtener distintos valores (enteros, flotantes, cadenas, etcétera) como respuestas, las cuales son más descriptivas que evaluar el resultado de invocar a un proceso del sistema operativo; *JNI*, también permite obtener respuestas de forma síncrona de los métodos de las librerías dinámicas, lo que brinda mayor control y certeza que esperar respuestas asíncronas de procesos independientes que se ejecutan periódicamente. Por otro lado, los ***servlets*** permiten construir aplicaciones web complejas y escalables, de forma sencilla, usando patrones tal como el ***MVC*** (***Model View Controller***, Modelo Vista Control) o alguno otro derivado de éste.

También al construir y migrar el sistema se encontraron ventajas y desventajas de la ***API Java Communications*** para *Windows* de 32 bits, en su uso para la instrumentación y supervisión a distancia.

Cuando se comenzó a evaluar y a usar esta *API*, la documentación (muy escueta) indicaba que estaba diseñada para el control de los puertos serie y paralelo, en un lenguaje orientado a objetos (Java), por lo que las ventajas parecían obvias, ya que se evitaba el problema de implantar rutinas de bajo nivel para el control de los puertos. Durante el desarrollo del sistema, se observó que sí se podía usar para abrir el puerto paralelo, y enviar datos; posteriormente, al integrarlo en un ambiente web (en ***Apache-Tomcat***, usando ***Windows Millenium*** como sistema operativo), funcionó correctamente.

Más adelante, en algunas pruebas, se modificó el alambrado para tratar de leer un dato por el puerto paralelo, pero la ***API Java Communications*** arrojó una excepción (*Unsupported operation. Output only mode*) al tratar de cambiar el modo *SPP* a *ECP* o a *EPP*, indicando que sólo permitía datos de salida. En una referencia bibliográfica (***Java I/O™***, de Elliotte Rusty Harold [33]), se encontró que la ***API*** sólo permite el modo ***SPP***, es decir, sólo puede enviar datos por el puerto paralelo, a pesar que en la documentación de la ***API*** ofrece abrir el puerto paralelo en distintos modos.

Se probaron varias combinaciones de las opciones de configuración del puerto paralelo en *Windows XP* con la interfaz ***Java Communications***, pero esta no opera correctamente en *Windows XP*, puesto que no envía datos, a menos que se inhabilite el puerto, se reinicie el equipo, y se habilite el puerto; incluso, a pesar de que esté trabajando correctamente, si se reinicia la computadora, la interfaz ***Java Communications*** vuelve a presentar problemas, por lo que se tiene que repetir los pasos de inhabilitar el puerto, reiniciar el equipo y habilitar el puerto.

También se intentó registrar la librería dinámica en el registro de *Windows XP*, pero el sistema operativo envía mensajes de advertencia al tratar de hacerlo, no importando cualquier combinación de opciones del comando

regsvr32, ya que *Windows XP* no reconoce a *win32com.dll* como una librería dinámica válida.

Cuando se cambió el sistema operativo a *Windows XP Profesional 2002* (con los *Service Pack* 2 ó 3), se encontró otra desventaja adicional; como el circuito electrónico no es un dispositivo apegado a los estándares *Plug and Play* -que requiere que el periférico envíe un identificador del dispositivo en una cadena en modo *nibble*-, el sistema operativo no lo reconoce, y la ***API Java Communications*** no opera correctamente. Se tiene que inhabilitar el puerto paralelo, apagar la computadora, conectar el circuito (alimentado correctamente), encender la computadora, habilitar el puerto paralelo, y finalmente, levantar ***Tomcat***. Otra forma de hacer que funcione la interfaz ***Java Communications*** en *Windows XP*, es tener seleccionada la opción "Usar cualquier interrupción asignada al puerto, en las propiedades de Puerto de impresora *ECP*", del Administrador de dispositivos; al cambiar la configuración esta interfaz puede enviar datos correctamente, y si ya se tenía dicha configuración, se tiene que inhabilitar y habilitar el puerto paralelo, sin necesidad de reiniciar la computadora.

Sin embargo, la librería dinámica *wincom32.dll* si presentó muchos conflictos al cambiar el sistema operativo, a pesar de que Windows ofrece compatibilidad con aplicaciones hechas para versiones anteriores. Por una parte, dicha librería dinámica no está completa y no se puede registrar en *Windows XP* porque depende de otras librerías dinámicas -las cuales son desconocidas porque no se mencionan en ninguna parte-; *Windows XP* sí puede registrar librerías dinámicas hechas para otras versiones, tales como *NT* y *Windows 2000*. Por otra parte, *Windows XP* con *Service Pack 2* ó *Service Pack 3* cambia la forma en que se tiene acceso a los periféricos, algo que no menciona explícitamente; la compañía *Microsoft Corporation* recomienda la instalación del *Service Pack 2* ó del *Service Pack 3*, presentándolas como mejoras a su sistema operativo, las cuales corrigen varios defectos y aumentan la seguridad del mismo, pero no mencionan de antemano que efectúan cambios radicales respecto a la comunicación con los dispositivos periféricos.

Por otra parte, este sistema de cómputo demuestra que sí se puede operar un periférico complejo, tal como la videocámara, por medio de Internet; la videocámara requiere de una configuración compleja para activala, y de varios pasos para obtener imágenes de ella, tan sólo como aplicación de consola; en ambiente de web se resolvieron los problemas de sincronización y de concurrencia. En este caso, al solicitar una imagen (por medio del ***servlet*** **ControlDireccionPuerto01.java**), la aplicación, por medio de las clases **VideoCamara.java**, de su clase padre, **CuadroVideo05.java** y la interfaz ***Java Media Framework***, sí se activa (se prende) la videocámara de estado sólido al ser llamada por primera vez, comienza a obtener video, se consigue una imagen del flujo de video y se detiene la toma de video de la videocámara. En llamadas posteriores, las clases **VideoCamara.java** y **CuadroVideo05.java** detectan que ya se había encendido la videocámara, por lo que no es necesario encenderla nuevamente, por lo que reinician la toma de video hasta obtener una nueva imagen, deteniendo la toma de video hasta que llegue otra solicitud.

En el caso de la interfaz ***Java Media Framework***, ésta usa a su vez la interfaz ***Java Native Interface*** para comunicarse con las librerías dinámicas que controlan los componentes físicos de audio y video (*hardware*). Esta colaboración entre el lenguaje java (independiente de la plataforma) y las librerías dinámicas (con código del sistema operativo en que se crearon) es más eficiente y obtiene más datos del resultado del llamado al método o función nativos, que si se invocara a un proceso del sistema operativo (el cual regresa menos datos), o si se tuviera que esperar el resultado de un proceso totalmente independiente (asíncrono), ya que en este último caso no se garantiza que se reciba respuesta después de hacerle una petición.

Respecto a la parte mecánica del movimiento de la videocámara, se encontró que el movimiento horizontal mediante un motor de pasos es relativamente sencillo; el movimiento vertical por medio de otro motor de pasos sí es factible, pero se requiere de la colaboración de otras ramas de la ingeniería para hacer una más eficiente. Se encontró que el peso de la videocámara y de un motor de pasos es mucho para que un segundo motor de pasos los mueva completamente, pero se hizo un mecanismo controlado por el segundo motor de pasos, el cual efectúa un movimiento de inclinación (vertical) del primer motor de pasos (y de la videocámara a la cual está unida); el diseño

mecánico para efectuar ambos movimientos de forma eficiente está fuera del alcance de esta tesis, ya que corresponde a la ingeniería mecánica.

Se pudo construir un módulo de administración relativamente sencillo, el cual tiene interacción con la base de datos *MySQL* para insertar, modificar, borrar o consultar información, mediante *J/Connector*, el cual implanta las interfases de ***JDBC***. También se pudo implantar la autenticación de usuarios, al construir un reino de seguridad con base en ***JDBC*** y papeles o funciones de usuarios, para permitir o negar el acceso a los recursos de la aplicación web, ya sean dinámicos (como los ***servlets*** y ***JSPs***), o estáticos (archivos ***HTML*** o imágenes). Se construyó un filtro de peticiones, para permitir o desviar solicitudes a un recurso web específico.

La construcción del módulo de administración comprobó que se pude tener interacción fácil y eficientemente con la base de datos relacional *MySQL*, la cual permite guardar datos numéricos, cadenas y fechas, así como información binaria, la cual puede corresponder a imágenes (como en el caso de este sistema) o que podrían ser datos de mediciones de otros sistemas. Esta base de datos puede ser utilizada por otro sistema de cómputo, para compartir información. Se pudo apreciar que las bases de datos relacionales son muy populares y fáciles de usar, pero se requiere de mucho trabajo hacer el código que ejecute las sentencias *SQL*.

Las interfases para el usuario (la respuesta ***HTML*** de los ***JSPs***), tanto del módulo de administración como del control de posición de la videocámara, son más sencillos y menos poderosos que los que se podrían tener mediante la construcción de una aplicación cliente hecha en otro lenguaje (que genere ejecutables), pero son eficaces, por lo que la aplicación web ahorra la construcción de la aplicación cliente (usando en cambio el navegador), mientras que se concentran los esfuerzos en la construcción del servidor; también permite que se tenga acceso a esta aplicación desde diversos sistemas operativos, siempre y cuando tengan acceso a Internet. También se ahorra la instalación y mantenimiento de aplicaciones cliente, de los sistemas Cliente-Servidor tradicionales, ya que al cambiar alguna regla, se tenía que reemplazar el cliente en todos los equipos donde estuviera instalado.

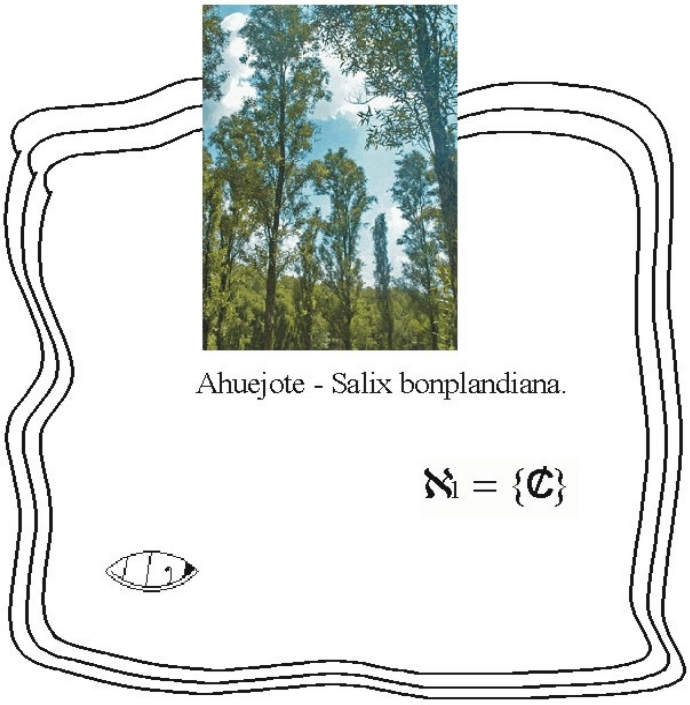


Figura 25: *La construcción de un sueño.*

4 Conclusiones.

Un número irracional para entender el universo,
y uno imaginario para inventar nuevas soluciones.

4.1 Conclusiones sobre el sistema de cómputo.

Después de construir el sistema y ponerlo en línea, se encontró que sí se puede controlar de forma remota un dispositivo, pero con varias condiciones. En el caso de la videocámara, la respuesta del dispositivo (la toma de una imagen), se efectúa de forma indirecta; en este caso, la computadora (servidor) se comunica con la videocámara a través de un cable *USB*, mientra que la aplicación web en ***Tomcat*** llama a las clases de la librería ***JMF***, las cuales invocan a métodos nativos (ejecutables del sistema operativo). El control de posición de la videocámara se logra con datos de salida únicamente.

La idea principal de este sistema es demostrar que es posible hacer uso de un dispositivo periférico a través de Internet, y con base en las técnicas aplicadas y en las experiencias obtenidas se quiere mostrar cómo se podrían utilizar otros periféricos, tales como instrumentos de medición.

En primer lugar, el lenguaje de programación Java demostró que es un lenguaje de programación bastante poderoso y flexible, con el cual se pude hacer distintos tipos de sistemas; las características del mismo, tales como la orientación a objetos y la herencia sencilla, permitieron resolver problemas lógicos tales como el modelado de características y comportamiento de los periféricos (el puerto paralelo y la videocámara de estado sólido) a través del uso de atributos y métodos.

El desarrollo del módulo de administración mostró que se puede hacer una aplicación compleja sobre web, usando ***servlets***, ***JSPs*** y *JavaBeans* con el modelo ***MVC*** (Modelo, Vista, Control), junto con la interfaz ***JDBC*** *J/Connector*, la cual es de tipo 4 (que no requiere de instalación o código nativo); también al implantar un reino de seguridad con base en ***JDBC*** en ***Apache-Tomcat***, para permitir o denegar el acceso a las páginas web mediante nombres de usuarios, contraseñas y papeles o funciones de usuarios, así como el filtrado de peticiones, muestra la versatilidad del contenedor de ***servlets Apache-Tomcat*** y del lenguaje java.

Por otra parte, los ***servlets*** y ***JSPs*** tienen acceso ilimitado a los recursos de la computadora sobre la cual se ejecutan, por lo cual pueden llamar a procesos o comunicarse con los periféricos de ese equipo, probándose de forma efectiva en ***Apache-Tomcat*** (el cual es gratuito); en servidores de aplicaciones comerciales cambiaría la configuración, pero dichos servidores son más eficientes y tienen mayor capacidad.

El lenguaje java también puede hacer uso de otros componentes de programación hechos en otros lenguajes de programación; a través de su interfaz *Java Native Interface* pude invocar a funciones de librerías dinámicas. No todos los sistemas de cómputo pueden construirse con un sólo lenguaje de programación, por lo que es importante identificar los alcances y limitaciones de cada lenguaje, sobre todo en este tipo de sistemas, donde se manipulan periféricos. Los lenguajes '*C*', '*C++*', o ensamblador, son más adecuados para controlar periféricos, pero llevaría mucho tiempo construir un sistema de cómputo con sólo lenguaje ensamblador (y el mantenimiento del mismo también sería costoso), mientras que los lenguajes '*C*' y '*C++*' presentan muchas desventajas para hacer páginas de Internet, ya que los *CGI* de dichos lenguajes crean procesos que demandan muchos recursos computacionales. El lenguaje java, al pretender ser multi-plataforma, no puede tener acceso directo a los periféricos de una computadora, pero en cambio, puede construir elementos de programación que tienen interacción con base de datos o que construyen páginas web rápida y eficientemente.

Respecto a otros lenguajes de programación, tiene más ventajas económicas y técnicas que otros lenguajes, a excepción del lenguaje *Ruby*. Algunos lenguajes de programación tal como el *Delphi*, puede crear aplicaciones que sí tengan acceso a los periféricos, e incluso crear aplicaciones web -mediante *CGI*, con las ventajas y desventajas

mencionadas en el capítulo de *Análisis*-, pero requiere el pago de licencia, y si se quisiera crear una aplicación similar en otro sistema operativo (puesto que existen instrumentos con controladores para otros sistemas operativos), se tendría que repetir todo el ciclo de desarrollo, ya que *Delphi* genera aplicaciones sólo para la plataforma *Windows*.

Algunos lenguajes de guiones crean páginas web rápidas y eficientes, e incluso se acoplan con distintas bases de datos, pero para la interacción con periféricos, tienen la misma desventaja que el lenguaje java, por lo que se tendrían que elaborar módulos en lenguaje '*C*' o '*C++*' para manipular los puertos locales y la videocámara; la ventaja del lenguaje java en este aspecto, es que cuenta con la interfaz *Java Native Interface*, la cual permite llamar a librerías dinámicas.

Respecto al lenguaje *Ruby*, parecería quedar en desventaja. El lenguaje Ruby parece ser un lenguaje de programación ideal, pero ha tenido menos respaldo y difusión que el lenguaje java; también genera código que necesita ser interpretado, y carece de la capacidad para tener interacción directa con los periféricos de una computadora, por lo que también necesitaría que se construyeran módulos en los lenguajes '*C*' o '*C++*' para controlar los puertos locales y la videocámara. *Ruby* cuenta con librerías para el uso de dispositivos multimedia, pero la interfaz ***Java Media Framework*** es más poderosa, debido a que fue creada por el acuerdo entre varias empresas de sistemas y de fabricantes de periféricos multimedia, lo que permite que dicha interfaz pueda manipular distintos dispositivos multimedia mediante métodos genéricos.

4.2 Acerca de los formatos de imágenes.

Las partes más complejas del sistema se hicieron en *Windows Millenium*, y al cambiar el sistema a *Windows XP*, el módulo de video presentó muy pocos problemas. Se pudo activar, obtener video, una imagen fija del video en formato ***JPEG***, y apagar la videocámara, a través de Internet; el formato ***JPEG*** es un formato muy ampliamente difundido, y adecuado para presentar imágenes en páginas de la red Internet, por lo que la gran mayoría de los navegadores o exploradores de Internet pueden interpretarla y desplegarla correctamente. Por otra parte, el formato ***JPEG*** tiene compresión con pérdida, por lo que no se recomienda para mediciones científicas, siendo más adecuado el formato ***PNG***, cuyos archivos son de mayor tamaño que los de ***JPEG***, requieren de más tiempo de transmisión en Internet, y no todos los navegadores o exploradores pueden desplegarlo correctamente.

El formato ***JPEG*** es adecuado para un sistema de video-vigilancia o supervisión remota; para un sistema el cual requiera capturar imágenes como medición científica, se requeriría cambiar la videocámara por otra de mayor resolución y cuyos controladores manejen el formato ***PNG***; en este caso, se podría usar la misma técnica de tratamiento de video, especificando que el formato de los *códecs* sea ***PNG***.

4.3 Conclusiones sobre las bases de datos.

Cuando se comenzó con el análisis y desarrollo del sistema, y se revisó el requerimiento de guardar la información, se consideró adecuado utilizar una base de datos relacional para guardar información de los usuarios del sistema, de las imágenes del sistema, e incluso para que otras aplicaciones pudieran guardar información de otros tipos de mediciones. El administrador de base de datos *MySQL* sí ha permitido que la aplicación guarde y recupere información, aunque el desarrollo de los objetos de acceso a datos (***DAOs***) y de otras clases que tienen interacción con la base de datos consumieron mucho tiempo. Sin embargo, el modelo Entidad-Relación diseñado, encausó la construcción de la aplicación, junto con el uso del modelo ***MVC*** (Modelo-Vista-Control).

De haber usado una base de datos orientada a objetos, se hubiera ahorrado mucho tiempo, evitando la construcción de los objetos ***DAOs*** y otras clases del lenguaje java relacionadas con base de datos, ya que ese tipo

de datos permite guardar las instancias de objetos java directamente; en esta caso se hubiera tenido que hacer más énfasis en el diseño de un diagrama de clases para guiar correctamente la construcción de la aplicación. Las bases orientadas a objetos ofrecen muchas ventajas, pero no han tenido mucha difusión, más que nada porque las bases de datos relaciones siguen siendo muy populares, tanto para aplicaciones propietarias de los fabricantes de bases de datos, de aplicaciones orientadas a objetos, y de lenguajes estructurados o con base en guiones.

Es por la última razón por la cual se debe seguir conociendo las bases de datos relacionales; las bases de datos orientadas a objetos ofrecen nuevas perspectivas, pero sólo los lenguajes orientados a objetos las pueden usar; también hay que considerar que sólo algunos fabricantes de administradores de bases de datos ofrecen administradores de bases de datos orientados a objetos libres de licencia y/o código abierto.

4.4 Conclusiones sobre el uso de puertos locales y periféricos con el lenguaje de programación java.

Respecto al uso de los puertos locales por parte de una aplicación de red Internet, aparecen ventajas y desventajas, influyendo el cambio de sistema operativo en el cual se ejecuta la aplicación (de *Windows Millenium* a *Windows XP*). El módulo de control del puerto paralelo comenzó a presentar problemas conforme se aplicaron "parches" de seguridad a *Windows XP*, ya que cambian la arquitectura con que la cual el sistema operativo permite el acceso a los periféricos. Los sistemas operativos y los periféricos de una computadora están íntimamente relacionados, ya que el sistema operativo coordina el envío y recepción de información por los canales de datos que unen los periféricos con el microprocesador, por lo que un cambio de versión de sistema operativo sí afecta el comportamiento de una aplicación que haga uso de periféricos -a pesar de que *Windows XP* garantizaba compatibilidad con aplicaciones hecha en versiones anteriores de *Windows*.

Para otro tipo de dispositivos, ¿cómo se podrían controlar y obtener una respuesta de los mismos, en Internet? A la anterior respuesta se ofrecen las siguientes soluciones:

- Para dispositivos que tengan controladores y protocolos de comunicación entre el periférico y la computadora bien definidos, se puede utilizar *JNI* (*Java Native Interface*) para implantar métodos que llamen a las funciones de código nativo, o utilizar la *API Jacob* (http://danadler.com/jacob/), la cual tiene base en *JNI*; finalmente, los ***servlets*** de su contenedor o servidor de aplicaciones, pueden llamara al código que utilice *JNI*.
- Utilizar la comunicación vía puerto serial y la ***API Java Communications***.
- Probar la ***API Java Communications*** en otros sistemas operativos.
- Reemplazar la librería dinámica ***win32.dll***, por otra que tenga el mismo nombre, pero tenga mayor funcionalidad, implantando las rutinas de bajo nivel (en lenguaje '*C*' o ensamblador).
- Desarrollar una ***API*** de comunicaciones (tanto como las clases como las rutinas de bajo nivel).
- Reemplazar la ***API Java Communications*** por la ***API RXTX*** (http://www.rxtx.org/).

En sistemas operativos anteriores, tal como ***MS-DOS***, las aplicaciones que deseaban usar los puertos paralelo o serie, solicitaban una referencia de memoria reservada a dichos puertos, y luego escribían datos en la memoria asignada al puerto correspondiente (envío), o leían datos de dichas direcciones de memoria (recepción).

Al aparecer *Windows 95*, *98* y *Millenium*, con su arquitectura de 32 bits, cambió la forma de tener acceso a dichos puertos. La arquitectura de 32 bits de *Windows* cambió la arquitectura con la que permitía la comunicación con los periféricos de una computadora, ya que en un ambiente multi-tareas se deseaba evitar conflictos entre aplicaciones por el uso de los periféricos, por lo que debían solicitarle permiso al sistema operativo para tener acceso

a los puertos locales, y registrar que están haciendo uso de los mismos (*ownership*). Microsoft permitía el desarrollo de controladores de periféricos por medio del *Device Development Kit* (*DDK*).

Más adelante, en Windows XP, y en específico, con los *Service Pack 2* y *Service Pack 3*, Microsoft cambia la arquitectura con la permite hacer el uso de los periféricos de la computadora; dichos *Service Pack* fueron introducidos como paquetes de mejoras a su sistema operativo, pero no mencionaban explícitamente que se cambiaba la arquitectura del uso de periféricos, reemplazando las herramientas del *Device Development Kit* por el *Windows Driver Kit (WDK)*, ocasionando problemas de compatibilidad para controladores de diversos periféricos, hechos con el *DDK*, ya que dicha compañía considera que dichas librerías dinámicas ocasionan problemas de seguridad.

Microsoft seguía permitiendo el acceso a los periféricos (puertos locales y otros) para crear nuevos controladores, mediante el pago de una licencia para el *WDK*, o adquiriendo alguna de sus herramientas de programación. Por otra parte, Microsoft, en los últimos sistemas operativos que ha liberado, ha reducido las capacidades del puerto paralelo, cambiándolo de un puerto de comunicaciones a uno exclusivamente de impresión.

También ha influido que los puertos serie, y en especial el paralelo, se reemplacen por puertos *USB*; los puertos serie y paralelo quedaron definidos por estándares físicos y eléctricos, pero cada sistema operativo define cómo envía y recibe datos de ellos. Por otra parte, el puerto *USB* (*Universal Serial Bus*) aparece como un acuerdo entre fabricantes de periféricos, de sistemas operativos y de fabricantes de computadoras, con el fin de presentar un puerto de comunicaciones mejorado. Este acuerdo permite que puedan usar distintos tipos de periféricos en distintos tipos de computadoras, que pueden tener distintos sistemas operativos.

Al crear el sistema de la presente tesis se demuestra que sí es posible controlar y hacer uso de dispositivos a través de páginas de la red Internet, pero debido a cambios de los sistemas operativos más recientes surge una disyuntiva.

Por un lado, para la instrumentación remota, se podría hacer uso de un servidor dedicado a un instrumento en particular; dicho servidor no necesitaría ser la computadora más reciente, ni tener el sistema operativo más actual; se podría re-usar un equipo de cómputo, y seguir utilizando el puerto paralelo o se podría usar el puerto serie; si bien la interfaz ***Java Communications*** presenta varios defectos y *Sun Microsystems* ha dejado de darle mantenimiento y respaldo para la plataforma *Windows*, no significa que no se pueda hacer, sino que se podría reemplazar por otra interfaz de programación -*Sun Microsystems* ha señalado que ***Java Communications*** se reemplaza por el proyecto ***RXTX***, el cual es un proyecto de *GNU*-, o se podría crear otra interfaz, creando una librería dinámica para las rutinas de bajo nivel, y hacer uso de la interfaz ***Java Native Interface*** (*JNI*), para que aplicaciones del lenguaje de programación java puedan hacer uso de dicha librería dinámica. Incluso, se podría utilizar otro sistema operativo, tal como *Linux*, que requiera menos recursos computacionales, con el fin de ahorrar dinero en compra de equipo de cómputo.

Por otra parte, para poder difundir una aplicación de videocámara con movimiento (en cualquier tipo de licencia), sería muy recomendable reemplazar el uso del puerto paralelo por uno *USB*; para poder hacer uso de instrumentos a través de puertos locales en las versiones más recientes de *Windows*, se tendría que desarrollar nuevos controladores de dispositivos, mediante el pago de una licencia a *Microsoft*, en especial si se crea un nuevo instrumento de medición, ya sea para uso local o remoto.

4.5 Trabajo a futuro.

En la aplicación web creada, se controla una videocámara, y las clases desarrolladas para ese fin se pueden aplicar en otros proyectos.

Para el caso concreto del control de la videocámara y de su posición, este sistema podría funcionar en un servidor dedicado a este proceso, sobre todo para evitar el problema de tener que inhabilitar/habilitar el puerto paralelo y reiniciar el equipo; también se podría instalar en *Windows 95*, *98* o *Millenium*, donde la ***API Java Communications*** tiene menos conflicto.

Respecto a la parte mecánica, se encontró que el movimiento horizontal es fácil de conseguir, pero para el movimiento vertical se encuentran problemas con el peso combinado de la videocámara y el motor de pasos del movimiento horizontal; para la presente tesis se contempló cómo resolver problemas lógicos de programación, principalmente. Para llevar a cabo el movimiento vertical de forma efectiva, se requiere de un nuevo diseño mecánico (que está fuera del alcance esta tesis), pero se puede esbozar uno, en el cual la videocámara y el motor de pasos del movimiento horizontal se encontrarían en una esfera -este motor movería la cámara, no la esfera-, con dientes de engrane en la parte posterior; la esfera descansaría en una semiesfera con un engrane (movido por el motor de pasos del movimiento vertical) que movería la esfera, pero el diseño de la parte mecánica ya es otro problema distinto que pertenece al área de mecatrónica.

Las videocámaras de estado sólido se han vuelto más populares, además de que manejan otros formatos de video de mejor definición, a la vez de que su tamaño se ha reducido, por lo que se pueden utilizar motores de paso más pequeños que consuman menos electricidad; para el módulo de video, se pueden cambiar los formatos de video y fotografía (siempre y cuando tengan soporte en la ***API Java Media Framework***).

Debido al los cambios con el que *Windows XP*, y otros sistemas operativos más recientes de Microsoft, administran el acceso a los periféricos, a la vez que *Sun Microsystems* descontinúa su interfaz de comunicación por puertos locales, se podría desarrollar nuevas librerías dinámicas para el uso de los puertos locales para las nuevas plataformas.

Por otra parte, el uso del puerto paralelo ha disminuido, siendo reemplazado por los puertos *USB* y *Firewire*, por lo que se podría desarrollar librerías de comunicaciones para las mismas, creando las rutinas de bajo nivel en lenguaje ensamblador o en '*C*', cuyas funciones serían invocadas desde el lenguaje java mediante *JNI*, y a su vez, estas clases se pueden llamar desde ***servlets*** o ***JSPs*** en ***Apache-Tomcat*** (o desde un servidor de aplicaciones).

Recientemente, Microsoft permitió la distribución gratuita del *WDK*[51] (desde finales de diciembre del 2009), como incentivo para los programadores que deseen desarrollar controladores y aplicaciones de periféricos para los sistemas operativos que ha liberado recientemente (en específico, *Windows 7*). Esto brinda más oportunidades para el desarrollo de aplicaciones que requieran tener acceso a puertos locales -sin importar si es a través de web o de una aplicación que se ejecute localmente, ya que están proporcionando las únicas librerías que permiten la comunicación entre periféricos y aplicaciones dentro de los sistemas operativos más recientes de Microsoft.

El uso remoto de dispositivos de medición y/o control permitirá seguir haciendo uso de instrumentos o dispositivos cuando existan problemas de causa mayor que impidan o limiten el tránsito de personas (tal como la contingencia sanitaria por la epidemia de la influenza *AH1N1*), por lo que se deberá desarrollar nuevos controladores para los sistemas operativos más recientes, así como crear librerías compartidas para invocar a dichos controladores (tanto por aplicaciones locales, como aplicaciones de Internet).

Por último, faltaría probar la comunicación de los puertos serie y paralelo en otros sistemas operativos (tal como *Linux* o *Solaris)*, donde se podrían controlar otro tipo de dispositivos no comerciales; también se podría probar las capacidades de los puertos *USB* y *firewire* en los sistemas operativos antes mencionados, para ampliar la variedad de plataformas en las cuales se podrían usar dispositivos (comerciales o no), ya sea por medio de aplicaciones locales, o usando *JNI* y ***servlets*** para crear aplicaciones de Internet.

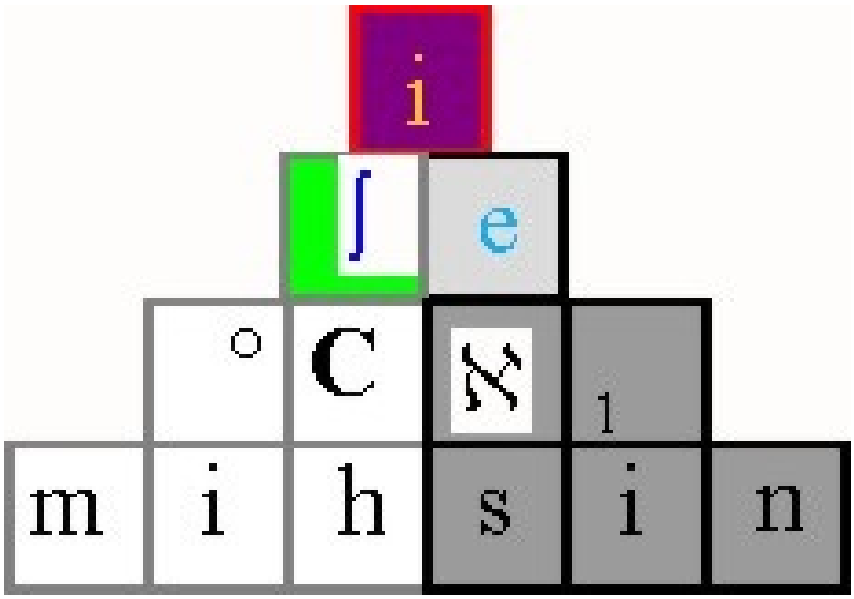


Figura 26: *Los conocimientos reunidos.*

Glosario.

Bytecodes (Códigos de bits), son instrucciones en binario almacenadas en un archivo *class*, resultado de la compilación de un código fuente escrito en el lenguaje java.

CASE (*Computer-aided Software Engineering*, Ingeniería de la Programación Asistido por Computadora): Conjunto de aplicaciones que ayudan a automatizar el desarrollo de programas de cómputo.

JDK (*Java Development Kit*, Conjunto de Herramientas del lenguaje Java): son varias aplicaciones, que incluyen el compilador, el intérprete de códigos de bits, el depurador, entre otras.

JRE (*Java Runtime Enviroment*, Ambiente de Ejecución en Tiempo Real del lenguaje Java): Es una aplicación que interpreta los códigos de bits del lenguaje Java, traduciéndolos a código máquina para ejecutar las instrucciones de los programas java.

Página web: Documento HTML de la red Internet con hipertexto que lo vinculan con otras páginas o recursos de Internet.

Pixel o ***pel*** (*Picture Element*):Unidad mínima de una imagen digital; cada ***pixel*** es un elemento de la matriz de la imagen, y contiene información de la ubicación espacial, del brillo y/o color.

Web (red): Conjunto de redes que conforman Internet.

Bibliografía.

[1] Intelligent Distributed Video Surveillance Systems (Professional Applications of Computing), primera edición.
Velastin, Sergio; Remagnino, Paolo (editores).
Editorial: Institution of Engineering and Technology, febrero de 2006, Estados Unidos de América.
Idioma: Inglés.
ISBN-10: 0863415040
ISBN-13: 978-0863415043

[2] PC Hardware in a Nutshell, segunda edición.
Bruce Thompson, Robert; Fritchman Thompson, Barbara.
Editorial: O'Reilly, junio del 2002, Estados Unidos de América.
Idioma: Inglés.
ISBN: 0-596-00353-6

[3] Parallel Port Complete: Programming, Interfacing & Using the PC'S Parallel Printer Port.
Axelson, Jan.
Editorial: Lakeview Research, febrero de 1997, Estados Unidos de América.
Idioma: Inglés.
ISBN-10: 0965081915
ISBN-13: 978-0965081917

[4] IEEE Standard 1284-1994.
Varios autores.
Editorial: IBM PS/2 Technical Reference, IBM ECP Specification, Microsoft 1284 Daisy Chain Specification, DISCTEC, 1994, Estados Unidos de América.
Idioma: Inglés.

[5] Upgrading and Repairing PCs, decimoséptima edición.
Mueller, Scott.
Editorial: Que, marzo del 2006, Estados Unidos de América.
Idioma: Inglés.
ISBN-10: 0-7897-3404-4
ISBN-13: 978-0-7897-3404-4

[6] Ingeniería de *Software*. Un Enfoque Práctico. Tercera Edición.
Pressman, Roger S.
Editorial: McGraw-Hill/Interamericana de España, S. A., 1993, México.
Idioma: Español.
ISBN: 84-481-0026-3

[7] Métodos Orientados a Objetos. Consideraciones prácticas. Primera edición.
Martin, James; Odell, James J.
Editorial: Prentice-Hall Hispanoamericana, S. A., 1997, México.
Idioma: Español.
ISBN: 970-17-0057-0

[8] Introducción a la Programación Orientada a Objetos.
Budd, Timothy.
Editorial: Addison-Wesley Iberoamericana, S. A., 1994, Estados Unidos de América.
Idioma: Español.

ISBN: 0-201-60103-6

[9] Aprendiendo Java 2 en 21 Días.
Lemay, Laura; Cadenhead, Roger.
Editorial: Prentice Hall Hispanoamericana, S. A., 1999, México.
Idioma: Español.
ISBN: 970-17-0229-8

[10] Java Programming Language. SL-275. Studen Guide. Revisión D.
Varios autores.
Editorial: Sun Microsystems, Incorporated, abril del 2000, Estados Unidos de América.
Idioma: Inglés.

[11] Descubre Java 1.2 .
Morgan, Mike.
Editorial: Prentice Hall Iberia S. R. L., 1999, España.
ISBN: 84-8322-142-X.

[12] Distributed Programming With Java Technology. Sl-301. Student Guide.
Editorial: Sun Microsystems, septiembre de 1999, Estados Unidos de América.
Idioma: Inglés.

[13] JavaBeans Component Development. Student Guide. Revisión B.2.
Editorial: Sun Microsystems Incorporated, noviembre de 1999, Estados Unidos de América.
Idioma: Inglés.

[14] Programación en Java 2.
Zukowski, John.
Editorial: Ediciones Anaya Multimedia, S. A., 1999, España.
Idioma: Español.
ISBN: 84-415-0948-4

[15] JDK™ 6 Documentation.
Varios autores.
Editorial: Sun Microsystems Incorporated.
URL: http://java.sun.com/javase/6/docs/
Idioma: Inglés.

[16] JavaTM Platform Enterprise Edition, v 5.0 API Specifications.
Varios autores.
Editorial: Sun Microsystems.
URL: http://java.sun.com/javaee/5/docs/api/
Idioma: Inglés.

[17] Redes Para Todos. Segunda edición.
Gibbs, Mark; Brown, Todd.
Editorial Prentice Hall Hispanoamericana, S. A., 1995, México.
Idioma: Español.
ISBN: 968-880-583-1

[18] Transmisión de datos y redes de comunicaciones. Segunda Edición.
Forouzan, Behrouz A.
Editorial: McGraw-Hill / Interamericana de España, S. A. U., 2002, España.
Idioma: Español.
ISBN: 84-481-3390-0

[19] Física. Materia, átomos, energías.
Varios autores.
Editorial: Plaza y Janés Editores, S. A. de C. V., 1985, España.
Idioma: Español.
ISBN: 84-01-31154-3

[20] Windows Multimedia. Video for Windows.
Varios autores.
Editorial: Microsoft Developer Network Library.
URL: http://msdn.microsoft.com/en-us/library/ms713492(VS.85).aspx
Idioma: Inglés.

[21] Microsoft® Windows® XP Professional Resource Kit, by The Microsoft Windows Team, segunda edición.
Varios autores.
Editorial: Microsoft Press, Junio del 2003.
Idioma: Inglés.
ISBN-10: 0-7356-1974-3
ISBN-13: 978-0-7356-1974-6

[22] The Image Processing Handbook.
Russ, John C.
Editorial:. CRC Press, cuarta edición, 2002.
Idioma: Inglés.
ISBN: 0-8493-1142-X

[23] Tratamiento digital de imágenes.
González, Rafael C.; Woods, Richard E.
Editorial: Addison-Wesley Iberoamericana S. A. de C. V. / Diaz de Santos, S. A., 1996, Estados Unidos de América.
Idioma: Español.
ISBN: 0-201-62576-8

[24] Imágenes Digitales. Procesamiento práctico con Java.
Pajares, Gonzalo; de la Cruz, Jesús M.; Molina, José M; Cuadrado, Juan; López, Alejandro.
Editorial: Alfaomega Grupo Editor S. A. De C. V., 2007, México, D. F.
Idioma: Español.
ISBN: 970-15-0951-X.

[25] Fotografía y video en Internet. Primera edición.
Orozco, Óscar.
Editorial: Alfaomega Grupo Editor, S.A. de C.V., 2009, México. D. F.
Idioma: Español.
ISBN: 978-970-15-1404-7

[26] Why Quicktime?
Varios autores.
Apple Incorporated, 2009, Estados Unidos de América.
URL: http://www.apple.com/quicktime/whyqt/
Idioma: Inglés.

[27] Java™ Media Framework API Guide.
Varios autores.
Editorial: Sun Microsystems Incorporated, noviembre de 1999, Estados Unidos de América.
URL: http://java.sun.com/javase/technologies/desktop/media/jmf/2.1.1/guide/index.html
Idioma: Inglés.

[28] DataBase Design for Mere Mortals: A Hands-On Guide to Relational Database Design. Segunda edición.
Hernández, Michael J.
Editorial: Addison Wesley Professional, tres de marzo del 2003, Estados Unidos de América.
Idioma: Inglés.
ISBN-10: 0-201-75284-0
ISBN-13: 978-0-201-75284-7

[29] Designing Effective Database Systems.
Riordan, Rebecca M.
Editorial: Addison Wesley Proffesional, diez de enero del 2005, Estados Unidos de América.
Idioma: Inglés.
ISBN-10: 0-321-29093-3
ISBN-13: 978-0-321-29093-9

[30] El modelo entidad-relación Case*Method™.
Barker, Richard.
Editorial: Addison-Wesley Iberoamericana, S. A. / Ediciones Díaz de Santos, S. A., 1994, Estados Unidos de América.
Idioma: Español.
ISBN: 0-201-60111-7.

[31] Physical Computing: Sensing and Controlling the Physical World with Computers.
O'Sullivan, Dan; Igoe, Tom.
Editorial: Course Technology, 2004.
ISBN: 1-59200-346-X
Idioma: Inglés.

[32] Tomcat Kick Start
Bond, Martin; Law, Debbie.
Editorial: Sams, Noviembre del 2002, Estados Unidos de América.
ISBN-10: 0-672-32439-3
ISBN-13: 978-0-672-32439-0

[33] Java™ I/O, segunda edición.
Rusty Harold, Elliote.
Editorial: O'Reilly, mayo del 2006, Estados Unidos de América..
ISBN-10: 0-596-52750-0.
ISBN-13: 978-0-596-52750-1.

Idioma: Inglés.

[34] 1.0 Programmers Guide.(Java Media Framework)
Varios autores.
Editorial: Sun Microsystems Incorporated, mayo de 1998, Estados Unidos de América.
URL: http://java.sun.com/javase/technologies/desktop/media/jmf/1.0/guide/index.html
Idioma: Inglés.

[35] Java™ Communications API Users Guide.
Varios autores.
Editorial: Sun Microsystems Incorporated, 1998.
Idioma: Inglés.

[36] Java Communications API Specifications.
Varios autores.
Editorial: Sun Microsystems Incorporated.
URL: http://java.sun.com/products/javacomm/reference/api/index.html
Idioma: Inglés.

[37] Java Communications API Read Me.
Varios autores.
Editorial: Sun Microsystems Incorporated, 1997.
Idioma: Inglés.

[38] Aprendiendo HTML 4 para web en una semana, tercera edición.
Lemay, Laura.
Editorial: Prentice Hall Hispanoamericana, S. A., 1998, México.
Idioma: Español.
ISBN: 970-17-0174-7.

[39] Servlets y JavaServer Pages. Guía Práctica.
Hall, Marty.
Editorial: Pearson Educación de México, S, A. de C. V., 2001, México.
Idioma: Español.
ISBN: 970-26-0118-5.

[40] Edición Especial MySQL.
Dubois, Paul.
Editorial: Pearsons Educación, S. A., 2001, España.
Idioma. Español.
ISBN: 84-205-3299-1.

[41] MySQL Reference Manual.
Varios autores.
Editorial: Sun Microsystems Incorporated, noviembre del 2008..
URL: http://dev.mysql.com/doc/refman/4.1/en/index.html
Idioma: Inglés.

[42] MySQL Connector/J.
Varios autores.

Editorial: Sun Microsystems Incorporated.
URL: http://dev.mysql.com/doc/refman/5.0/es/index.html
Idioma: Inglés.

[43] Apache Tomcat 5 Profesional.
Chopra, Vivek; Bakore, Amit; Eaves, Jon; otros autores.
Editorial: Ediciones Anaya Multimedia (Grupo Anaya , S. A.), 2005, España..
Idioma: Español.
ISBN: 84-415-1780-0.

[44] Apache Tomcat 6.0 Documentation Index.
Varios autores.
Editorial: Apache Software Foundation, 2008.
URL: http://tomcat.apache.org/tomcat-6.0-doc/index.html
Idioma: Inglés.

[45] The DAMA Guide to the Data Management Body of Knowledge (DAMA-DMBOK).
DAMA International, primera edición.
Editorial: Technics Publications, LLC; marzo del 2009.
Idioma: Inglés.
ISBN-10: 0977140083
ISBN-13: 978-0977140084

[46] Database in Depth.
C.J. Date
Editorial: O'Reilly Media, Inc., mayo del 2005, Estados Unidos de América.
Idioma: Inglés.
ISBN-13: 978-0-596-10012-4

[47] SQL For Dummies, sexta edición.
Taylor, Allen G.
Editorial: For Dummies; agosto del 2006.
Idioma: Inglés.
ISBN-10: 047004652X
ISBN-13: 978-0470046524

[48] Patrones de Diseño Aplicados a Java.
Stelting, Stephen; Maassen, Olav.
Editorial: Pearson Educación, S. A., 2003. España.
Idioma: Español.
ISBN: 84-205-3839-6

[49] Professional JSP, segunda edición.
Brown, Simon; Burdick, Robert; Falkner, Jayson, y otros.
Editorial: Wrox Press Ltd., 2001 Estados Unidos de América.
Idioma: Inglés.
ISBN: 1-861004-95-8

[50] Core J2EE Patterns: Best Practices and Design Strategies.
Alur, Deepak; Crupi, John; Malks, Dan

Editorial: Prentice Hall / Sun Microsystems Press, 2001.
ISBN: 0130648841
URL: http://java.sun.com/blueprints/corej2eepatterns/Patterns/ServiceToWorker.html

[51] Windows Driver Kit.
Varios autores.
Editorial: Microsoft Corporation, noviembre 2009.
Idioma: Inglés.
URL: http://msdn.microsoft.com/en-us/library/aa972908.aspx

Anexos.

1. **Anexo A: Diagrama de Flujo de Datos.**
2. **Anexo B: Diagrama Entidad-Relación.**
3. **Anexo C: Diagramas UML.**
4. **Anexo D: Otros Conceptos de Programación Orientada a Objetos.**
5. **Anexo E: Referencia del Formato *AVI.***
6. **Anexo F: Referencia de la Interfaz *Java Media Framework (JMF).***
7. **Anexo G: Código Fuente de la Aplicación.**

Anexo A. Diagrama de Flujo de Datos.

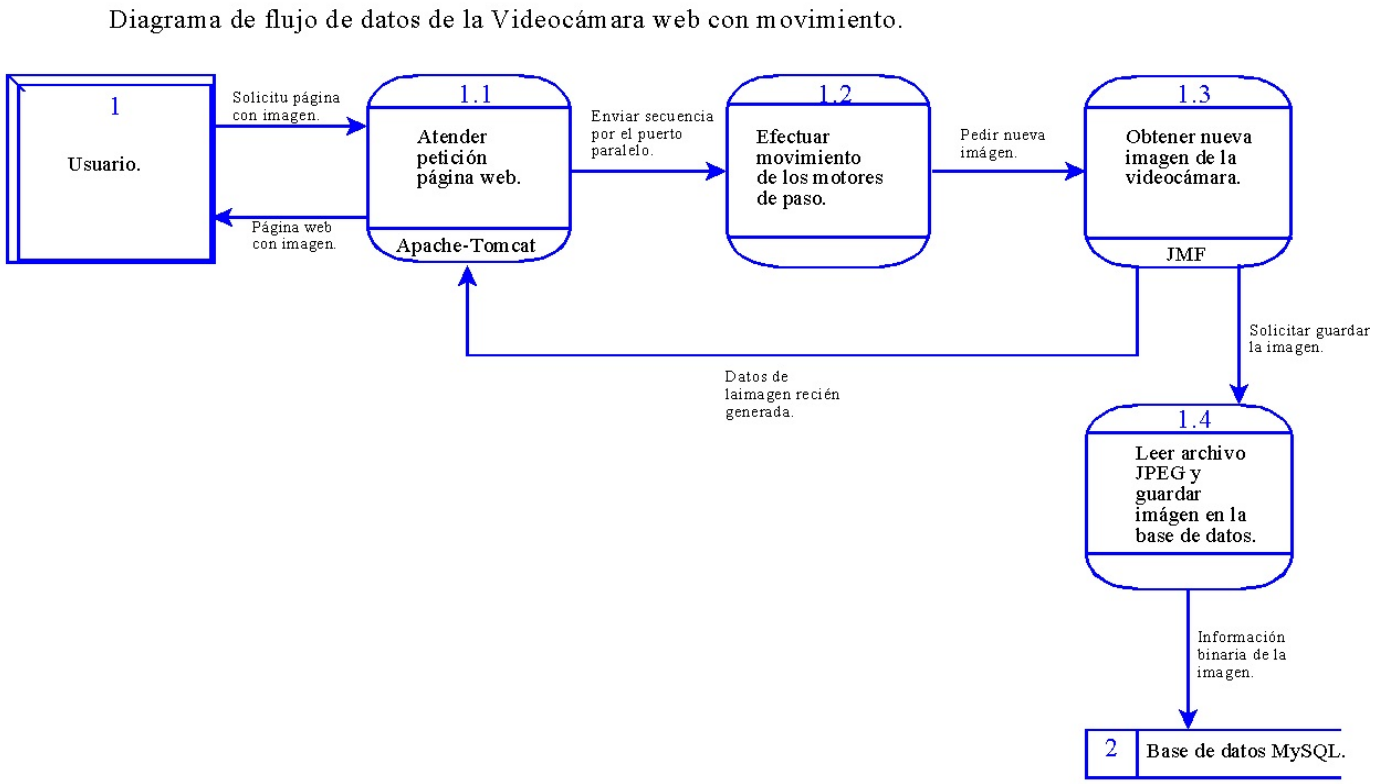


Figura 27: *Diagrama de Flujo de Datos del control de posición y obtención de imágenes de la videocámara.*

Anexo B. Diagrama Entidad-Relación.

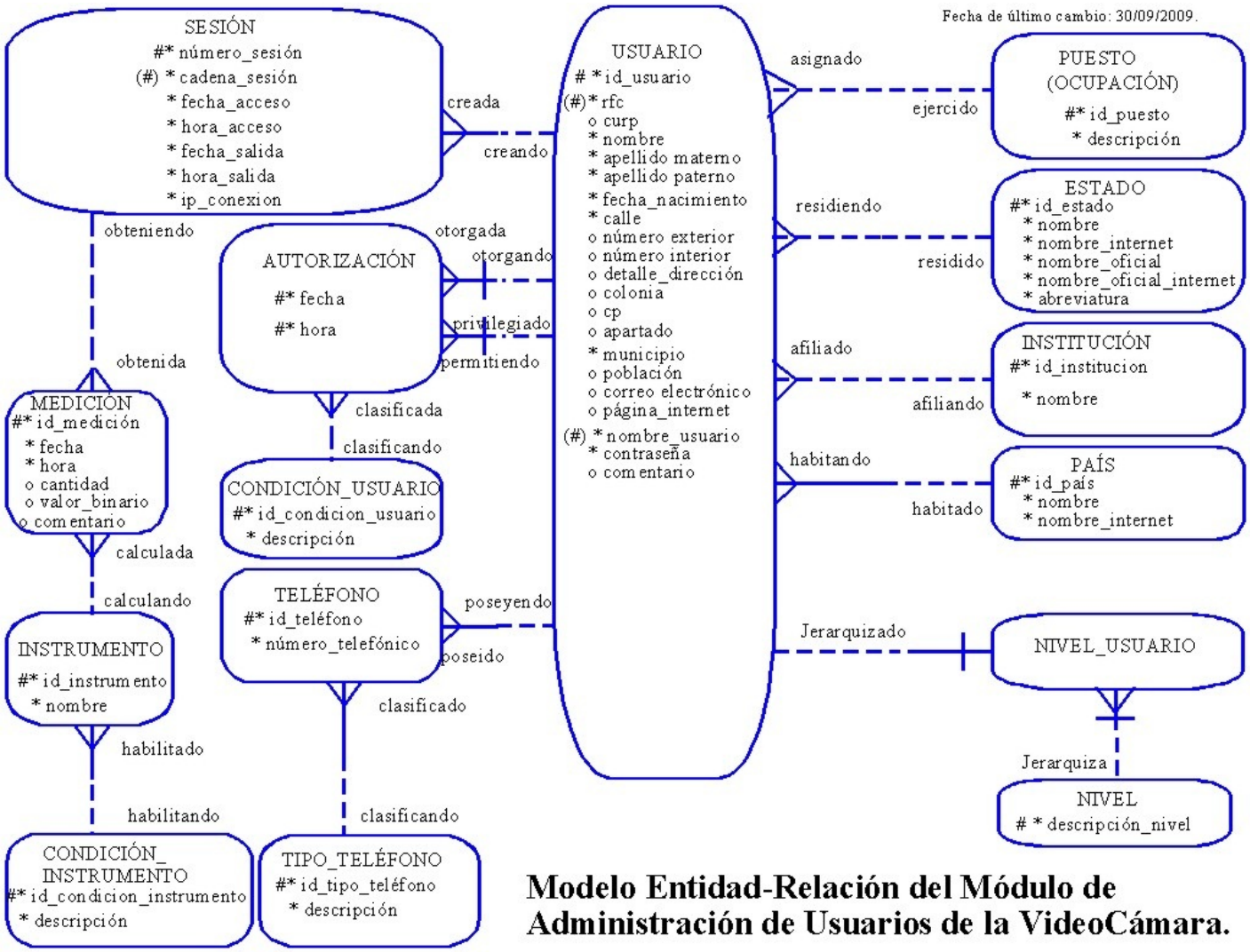


Figura 28: *Modelo Entidad Relación de la base de datos del sistema de la videocámara.*

Anexo C. Diagramas *UML*.

Anexo C.1 Casos de Uso.

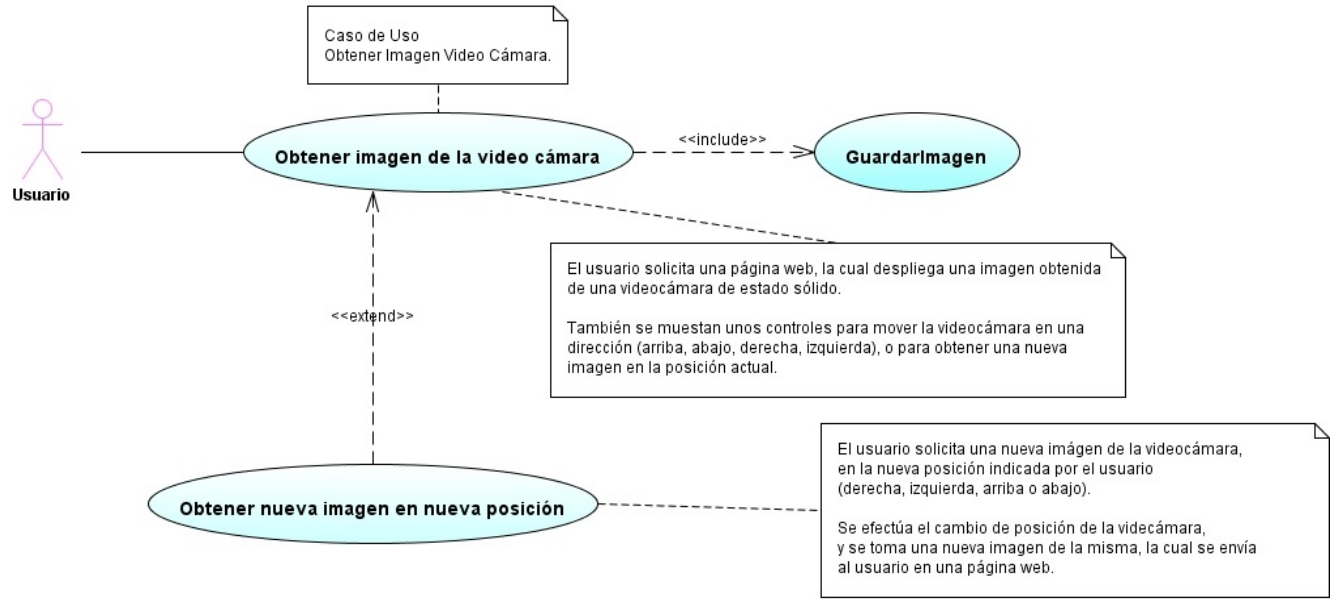


Figura 29: *Caso de Uso "Obtener Imagen Video Cámara".*

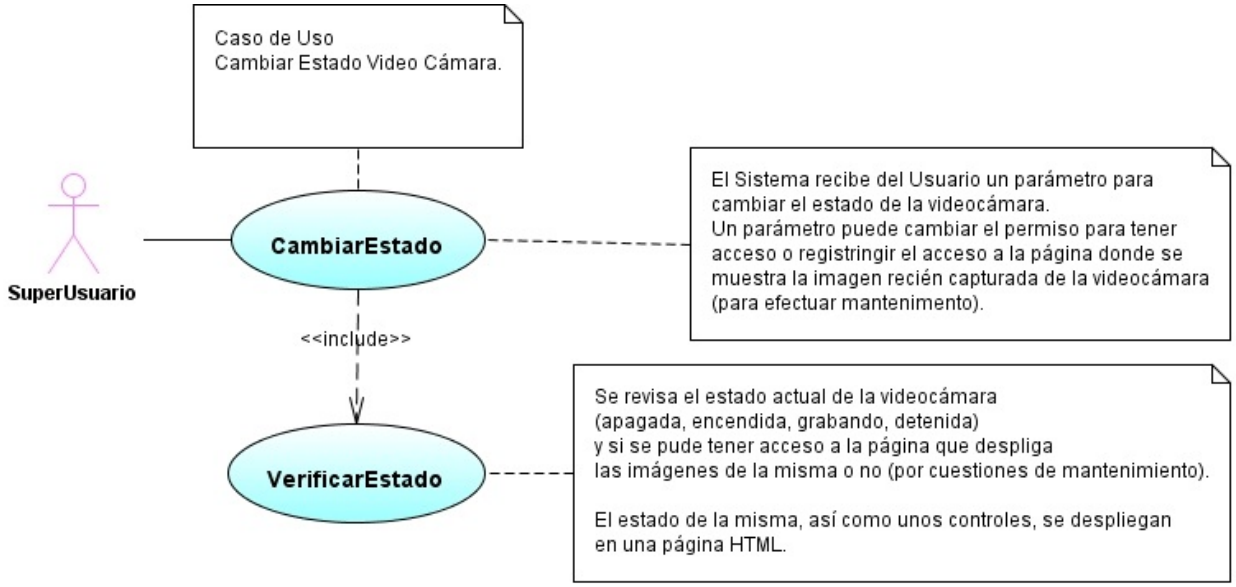


Figura 30: *Caso de Uso del cambio de estado de la videocámara.*

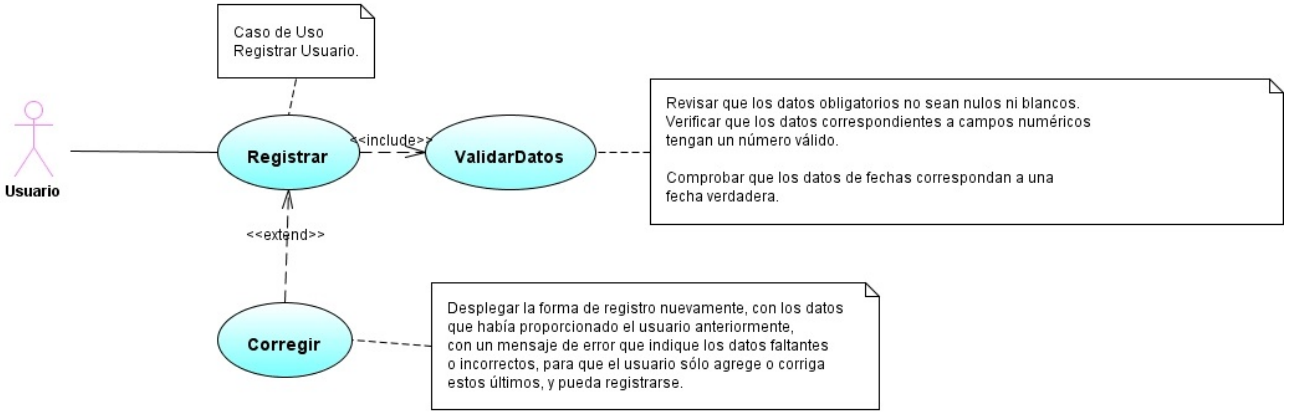


Figura 31: *Caso de Uso "Registrar Usuario".*

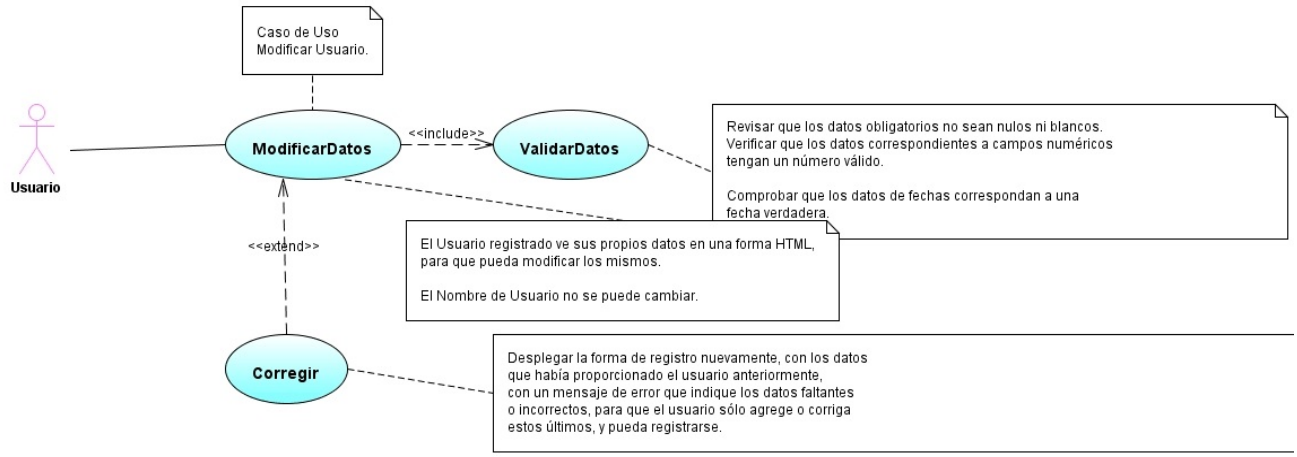


Figura 32: *Caso de Uso "Modificar Usuario".*

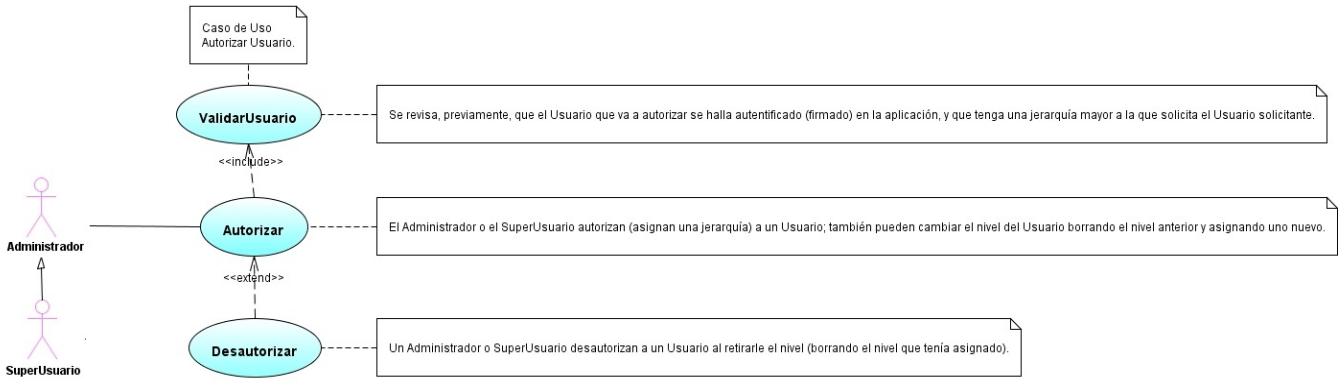


Figura 33: *Caso de Uso "Autorizar Usuario".*

Figura 34: *Caso de Uso "Consultar Usuarios Registrados.*

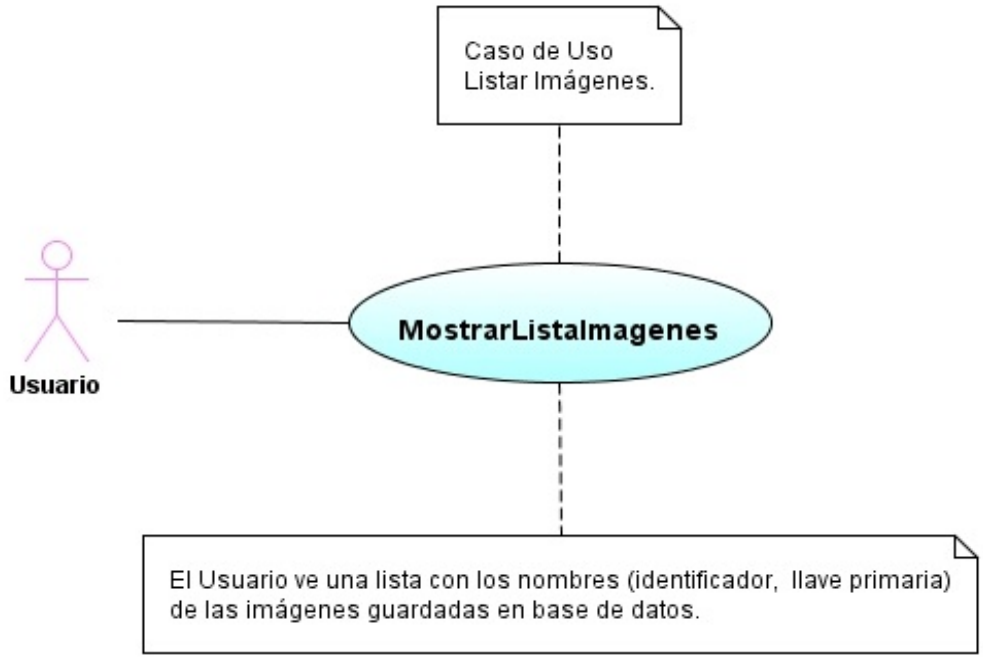


Figura 35: *Caso de Uso "Listar Imágenes".*

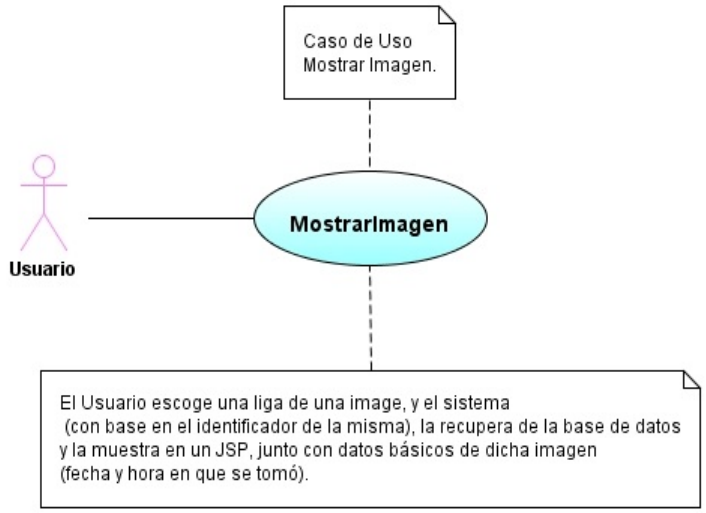


Figura 36: *Caso de Uso "Mostrar Imagen".*

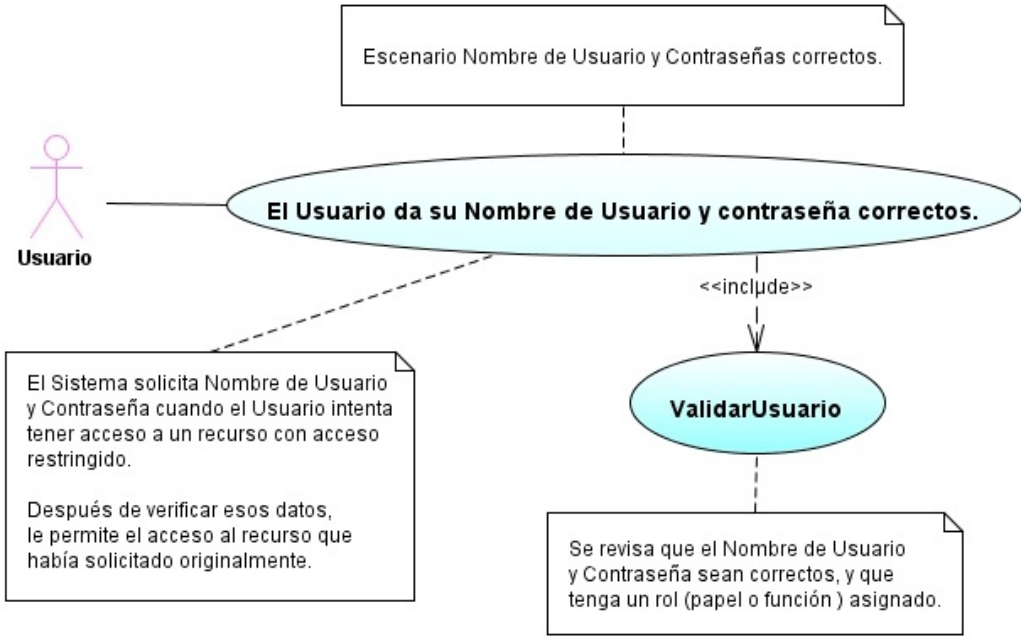


Figura 37: *Escenario "Nombre de Usuario y Contraseñas correctos".*

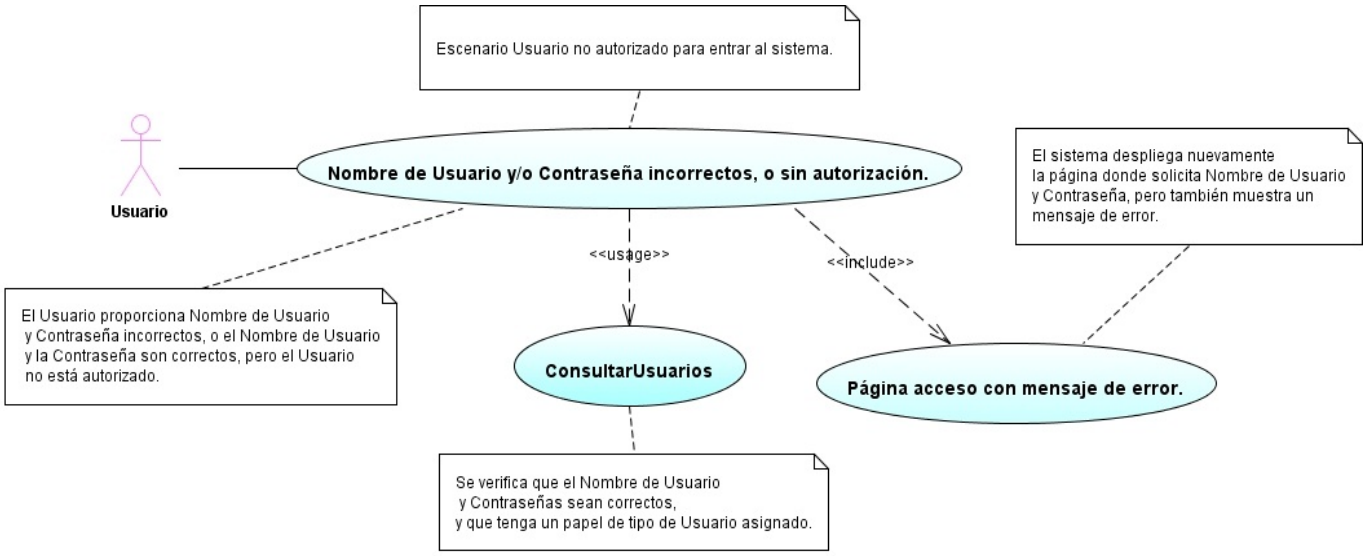


 Figura 38: *Escenario "Usuario no Autorizado".*

Anexo C.2. Diagrama de Distribución.

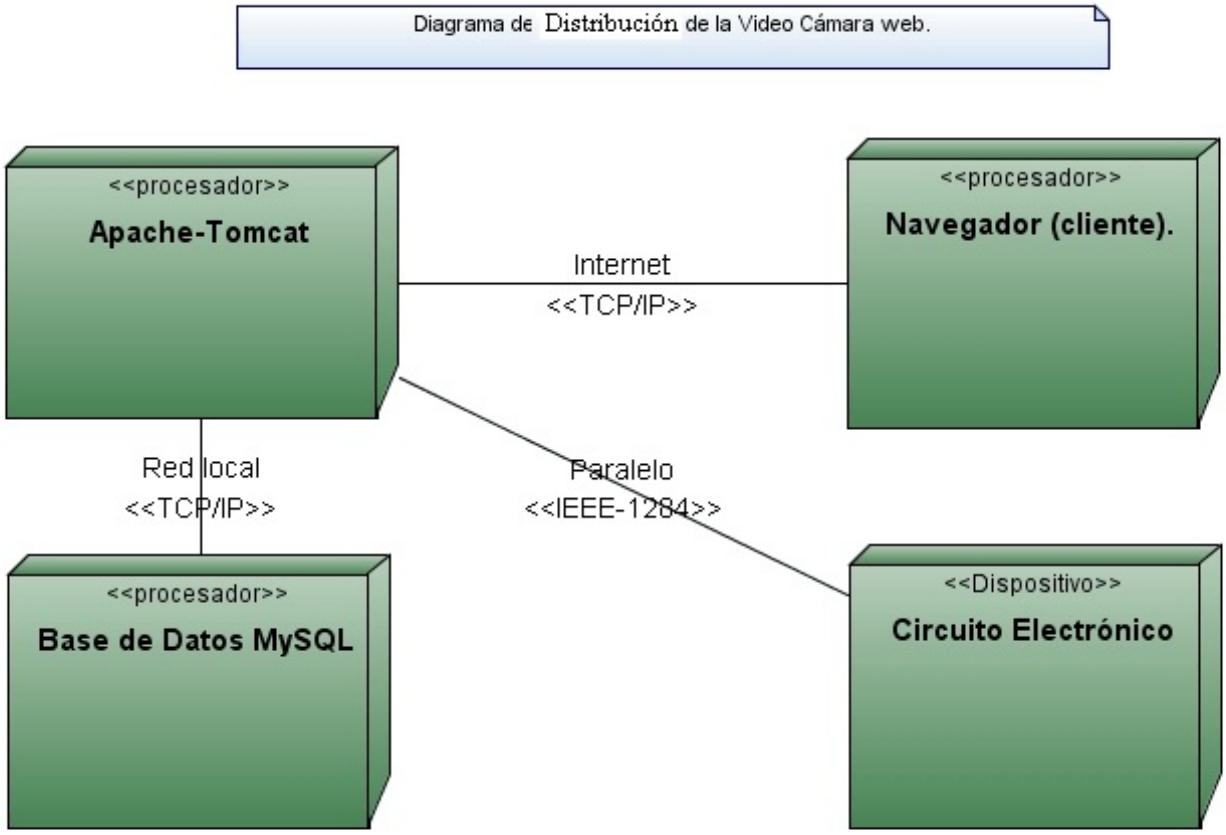


Figura 39: *Diagrama de Distribución de la Videocámara.*

Anexo C.4. Diagrama de Clases.

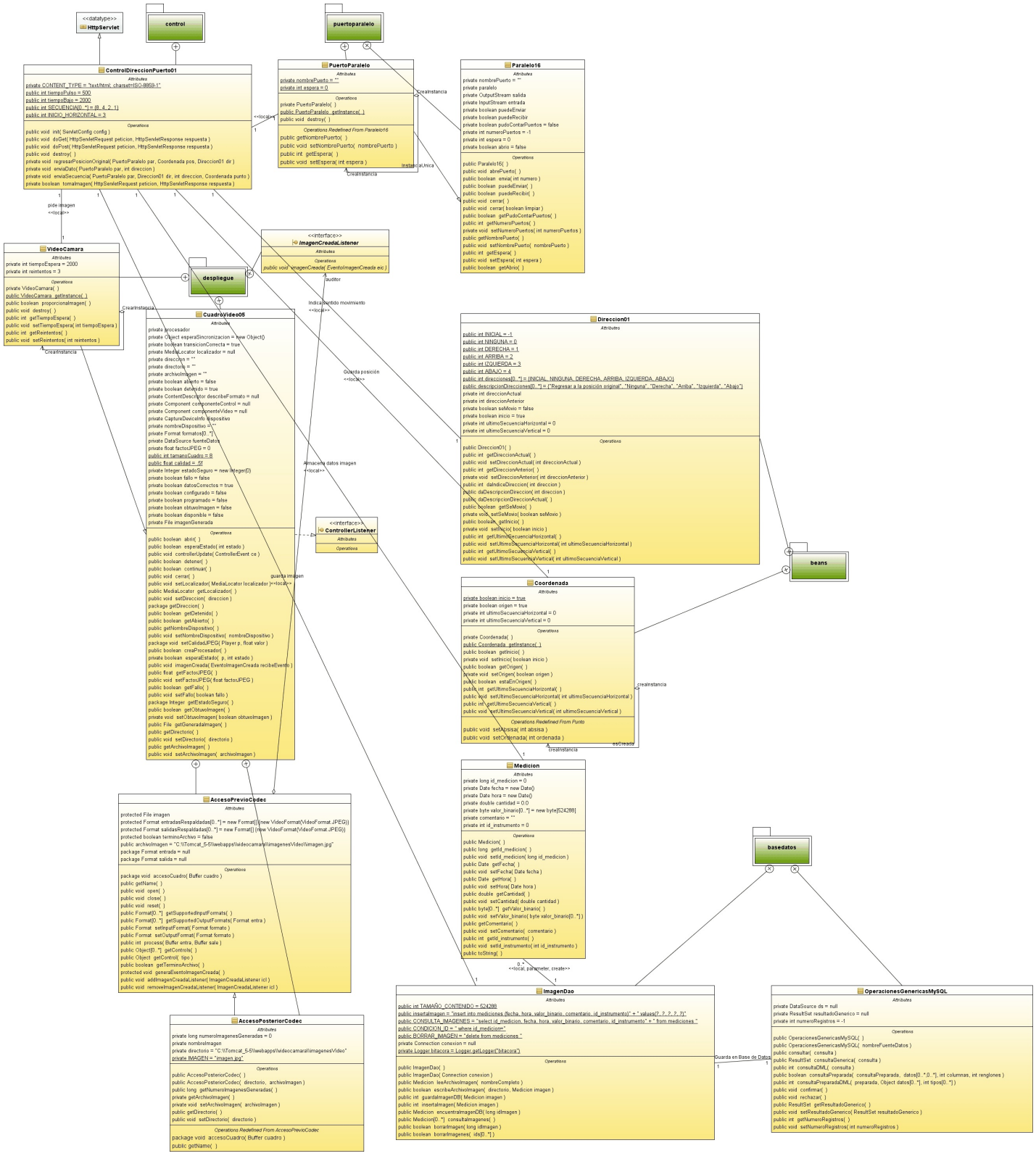


Figura 40: *Diagrama de Clases de la videocámara web.*

Anexo C.5. Diagrama Actividades.

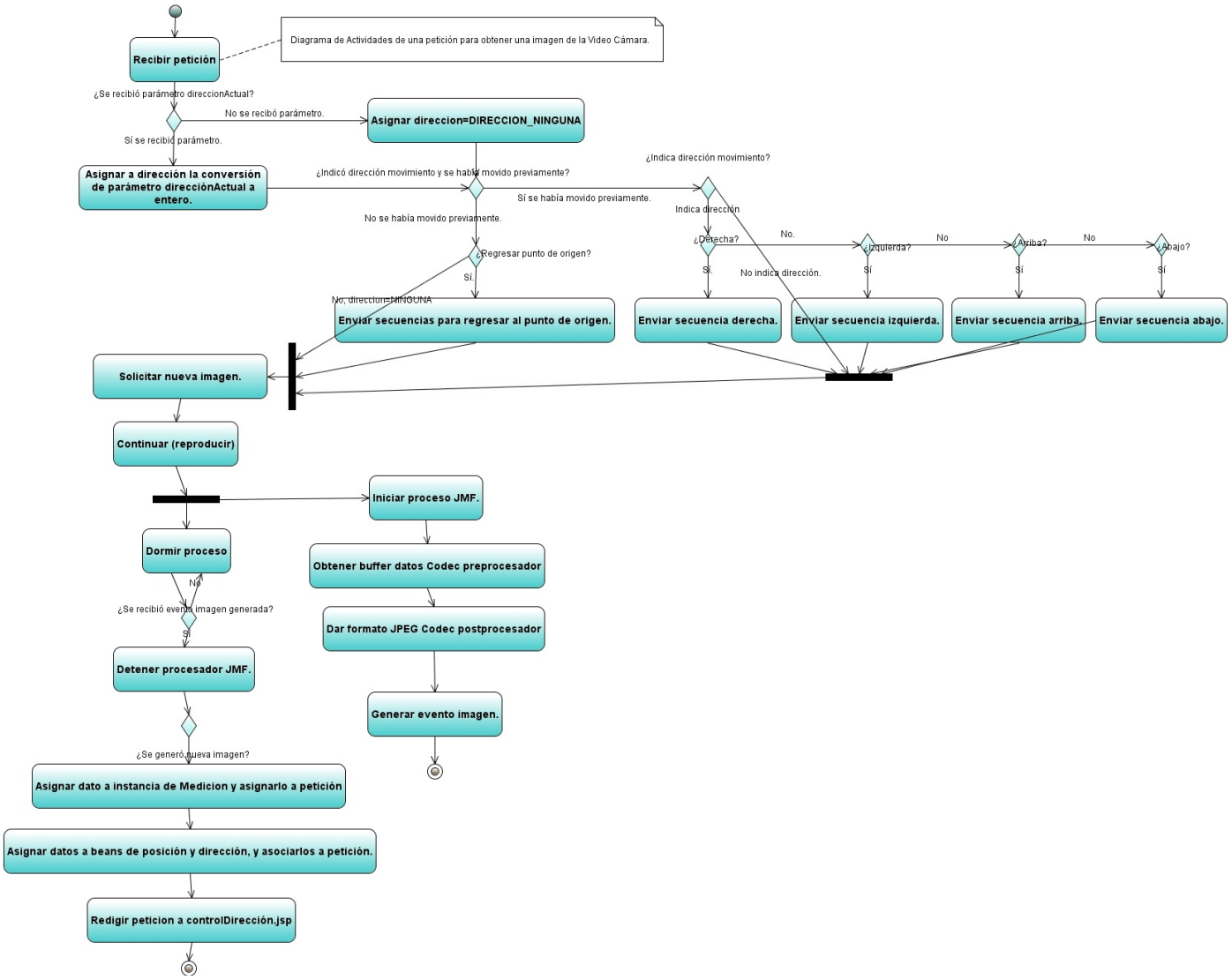


Figura 41: *Diagrama de Actividades de la Videocámara web.*

Anexo C.6. Diagrama de Estados.

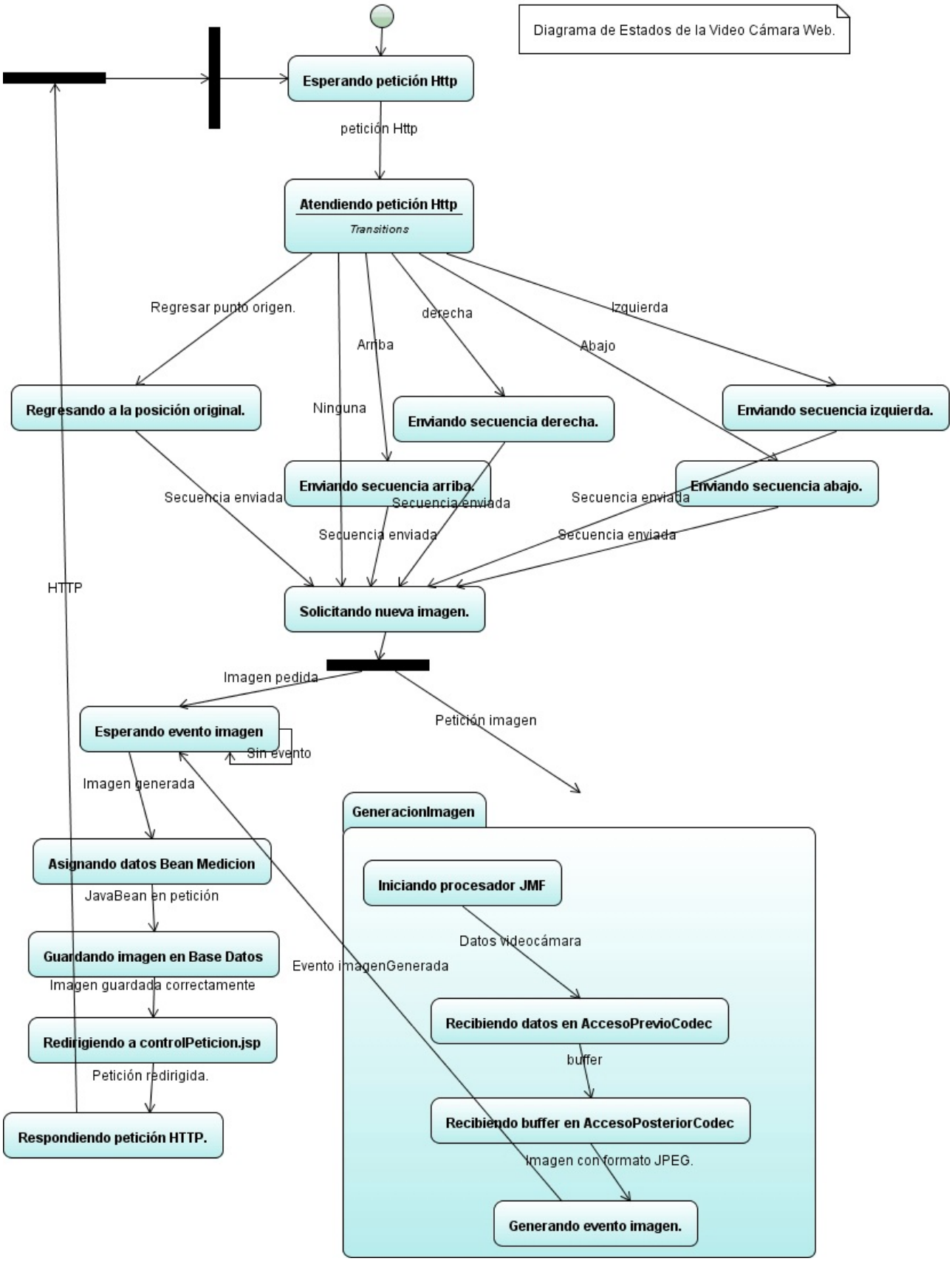


Figura 42: *Diagrama de Estados de la videocámara web.*

Anexo C.7. Diagrama de Secuencias.

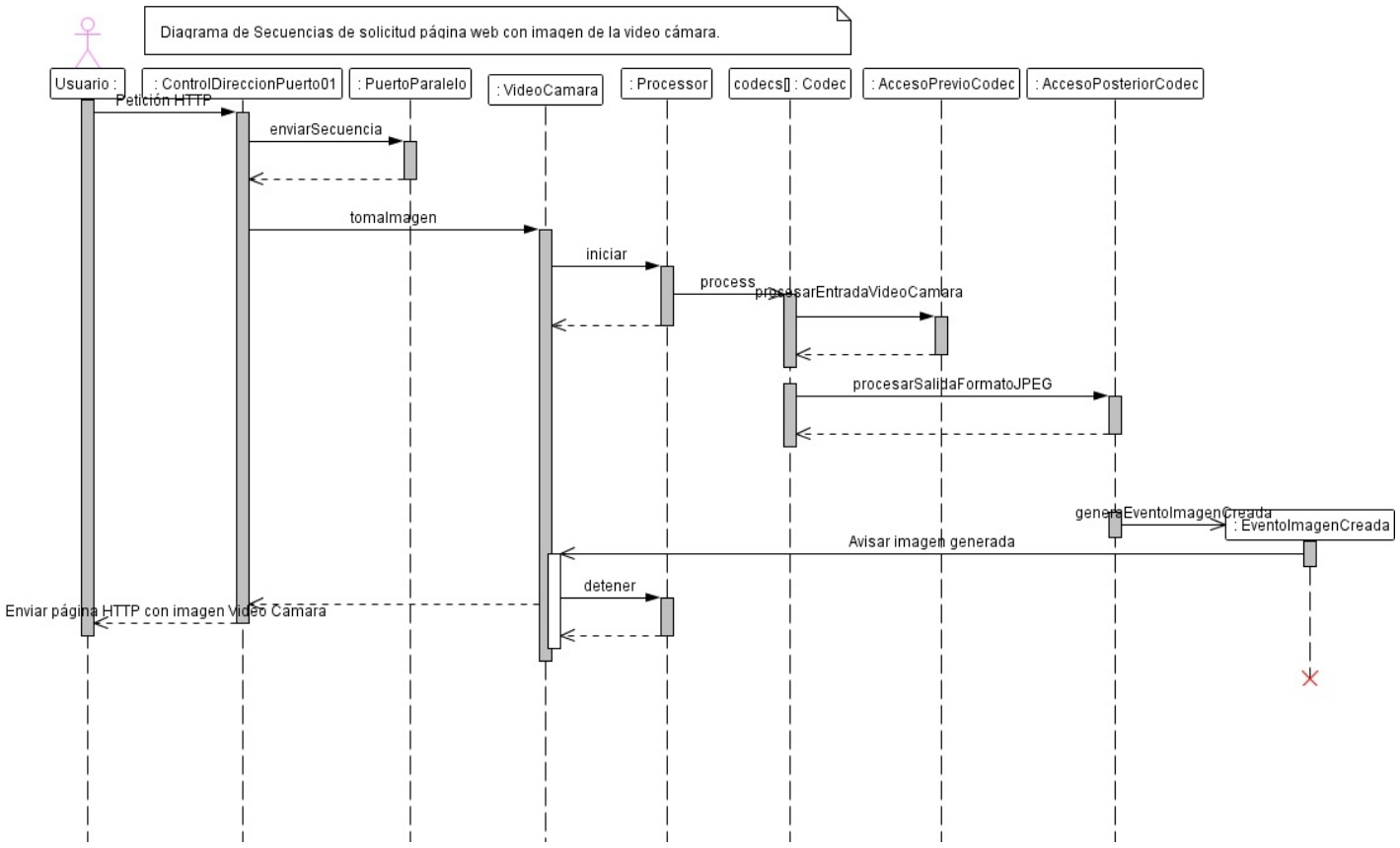


Figura 43: *Diagrama de Secuencias de la solicitud y obtención de imagen de la videocámara.*

Anexo C.8. Diagrama Estados de la navegación del sistema.

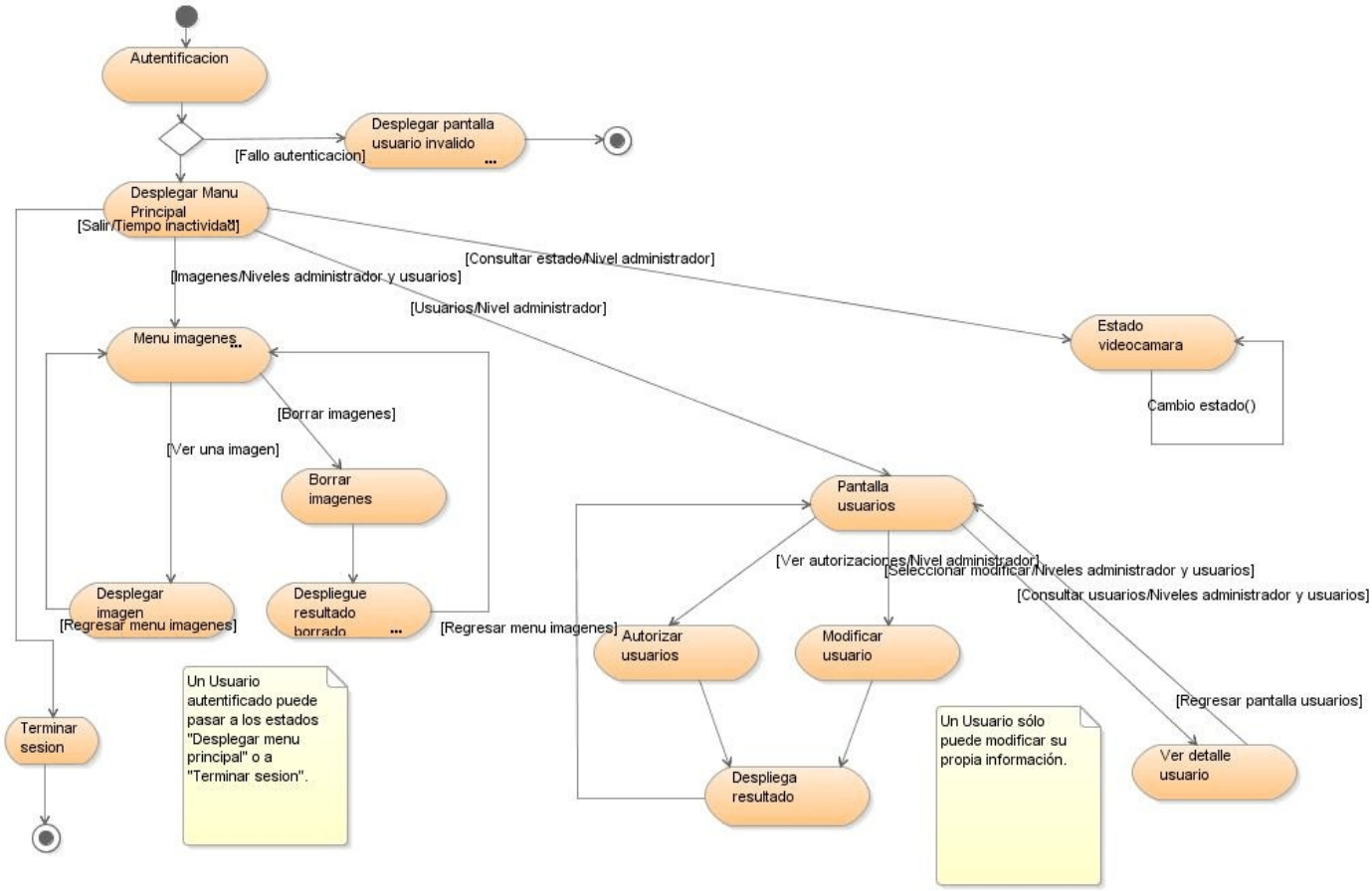


Figura 44: *Diagrama de estados de la navegación de la aplicación web videocámara.*

Anexo D. Otros Conceptos de Programación Orientada a Objetos.

Además de las principales características de la Programación Orientada a Objetos, hay otros conceptos, menos importantes pero no despreciables, por que se explican a continuación.

Sobreescritura.

La **Sobreescritura** (*override*) de un método consiste en que una clase hija vuelve a definir un método con la misma firma de una clase antecesora; la nueva definición del método prevalece, cuando en las instancias de la clase hija se invoca al método sobrescrito.

Eventos.

Consiste en mensajes asíncronos, enviados a otros objetos, sin esperar respuesta o confirmación de recepción de los mismos. En el lenguaje java las clases interesadas en recibir dichos mensajes usan (implantan) la interfase (*listener*) correspondiente al evento que se desea auditar.

Paquetes.

Este lenguaje permite la organización jerárquica de clases en paquetes; especifica la estructura en que se organizan las clases, agrupando clases simulares entre sí, e indica cómo pueden inter actuar entre sí. La sintaxis de la declaración de paquetes es la siguiente:

package <paquete_superior><sub_paquete_n>;*

Una clase del lenguaje java puede o no estar dentro de un paquete; si la clase no tiene indicado algún paquete, se considera que está en el paquete por defecto -el directorio donde reside dicha clase. En caso de que la clase sí indique un paquete, dicho paquete deberá corresponder con una estructura de directorios, donde cada directorio es un paquete.

Recolección de basura.

El lenguaje de programación Java tiene un **Recolector de Basura**, que consiste en un proceso que se ejecuta periódicamente en la *Java Virtual Machine*, para busca objetos no usados en la memoria, con el fin de eliminarlos y liberar recursos de la computadora.

Finalizadores.

Los finalizadores son métodos que se encargar de liberar recursos y / o terminar procesos dentro de una clase, con el fin de eliminar la instancia de esa clase de la memoria; se llama a dicho método cuando se quiere desechar el objeto que ya no se va a usar y que ya no tiene referencias en otros objetos, para que el **Recolector de Basura** los elimine en el siguiente ciclo.

Anexo E. Referencia del formato *AVI*.

E.1 Video para Windows (*Video for Windows*, *VfW*).

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 *Video para Windows* son los siguientes:

8. Funciones y *macros* de *AVIFile*.
9. Administrador de compresión de video.
10. Captura de video.
11. Archivo de configuración a la medida y controladores de flujo.
12. *DrawDib*.

E.1.1 Funciones y *macros* de *AVIFile*.

Permiten el acceso a archivos que tienen base en el tiempo y que usan el formato de archivos de recursos de información (*Resource Information File Format*, *RIFF*), tal como los archivos de audio con ondas, o los de video *AVI* (*audio-video interleaved*).

Estas funciones y *macros* ahorran el trabajo de manipular el contenido *RIFF* del archivo; por otra parte permiten obtener datos de los archivos como flujos de datos, en vez de bloques de datos, por lo que se pueden recuperar y procesar varias pistas multimedia de un sólo archivo.

E.1.2 Administrador de compresión de video (*VCM*).

Permite a compresores instalables de las aplicaciones tener acceso a la interfase que permite manejar datos que cambian respecto al tiempo; las funciones que provee la interfase son las siguientes:

1. Comprimir y descomprimir datos de video.
2. Enviar datos comprimidos de video a un dispositivo de despliegue, y asegurar que los muestre correctamente.
3. Comprimir, descomprimir o dibujar datos con un dispositivo de despliegue creado desde una aplicación.
4. Usar dispositivos de despliegue para manipular textos y datos personalizados.

E.1.3 Captura de video.

La adquisición de video la efectúa la clase de *Windows AVICAP*; dicha clase provee a las aplicaciones con una interfase sencilla que tiene base en intercambio de mensajes, la cual permite tener acceso a los dispositivos físicos (*hardware*) de video y de ondas de audio, y controlar el proceso de dirigir los flujos de multimedia mencionados a disco (para su almacenamiento).

La clase *AVICAP* permite captura de video y de un sólo cuadro de video en tiempo real. También permite controlar de fuentes de video que sean dispositivos de tipo *Media Control Interface* (*MCI*), para que los programadores puedan crear aplicaciones que brinden un control para iniciar y detener el video en cualquier parte

del mismo, y permitir la captura de cuadro por cuadro. Aunque en el sistema desarrollado no se llama directamente a esta clase, se hace uso de ella por medio de la *API Java Media Framework*, la cual recibe el nombre del protocolo de video y el nombre de dispositivo, para que las librerías con código nativo -código binario para *Windows*- puedan manipular la información multimedia. A continuación se mencionan las principales tareas que puede llevar a cabo:

1. Capturar flujos de audio y video a un archivo de tipo de audio y video intercalados (*AVI*).
2. Conectar y desconectarse de dispositivos de audio y video dinámicamente.
3. Ver una señal de video entrante en vivo usando métodos de previsión o de sobre-posición.
4. Especificar el nombre de un archivo para almacenar, y para copiar el contenido de un archivo de captura a otro archivo.
5. Especificar la tasa de captura.
6. Desplegar ventanas de diálogo que permitan controlar la fuente de video y su formato.
7. Crear, guardar y recuperar paletas de formatos de colores.
8. Copiar imágenes y paletas de colores al pisapapeles.
9. Capturar y guardar una sola imagen como un mapa de bits independiente de la plataforma (*DIB*).

E.1.4 Controladores de flujos y de archivos hechos a la medida.

Los controladores de archivos y de flujo hechos a la medida proveen interfases consistentes a las aplicaciones que controlan datos multimedia. En el caso del sistema operativo *Windows*, los controladores de archivos y flujos usan video y audio con forma de onda almacenados en el formato de audio y video intercalados (*audio-video interleaved AVI*), y en el formato de audio en forma de onda.

Estos controladores son creados por programadores para obtener multimedia de otras fuentes (como de un formato propietario), para leer un archivo de tipo *AVI* que contenga pistas de datos adicionales, o para hacer un controlador que genere sus propios datos multimedia.

E.1.5 *DrawDib*.

Las funciones de *DrawDib* brindan las capacidades para dibujo de imágenes de alto rendimiento para los mapas de bits independientes de dispositivos. Esas funciones escriben directamente a la memoria de video, y no dependen de las funciones de la interfase del dispositivo de gráficos (*Graphic Device Interface*, *GDI*). Las funciones de *DrawDib*, pueden manipular imágenes de ocho, dieciséis, veinticuatro, y de treinta y dos bits.

No se describen estas funciones con mayor profundidad, porque no son usadas en el sistema desarrollado, y quedan fuera del alcance de esta tesis.

E.2 El formato de video *AVI*.

El formato de audio y video intercalados (*audio-video interleaved*, *AVI*) de video es usado 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.

E.2.1 El código de cuatro caracteres (*four-character code*, *FOURCCs*).

El formato *AVI* utiliza códigos de cuatro caracteres (*FOURCCs*) para identificar tipos de flujos, bloques de datos, entradas de índices y otro tipo de información. Este código de 32 bits sin signo se forma al concatenar en forma ascendente el código *ASCII* de cuatro caracteres; por ejemplo, el código *FOURCC* de la cadena 'abc ' es 0x20636261.

E.2.2 El formato de archivos de tipo RIFF.

El formato *AVI* tiene base en el formato *RIFF*, el cual se forma con un encabezado de tipo *RIFF* seguido por cero o más listas y bloques de datos. El encabezado *RIFF* tiene la siguiente estructura:

'*RIFF*' *tamañoArchivo tipoArchivo* (datos)

1. '*RIFF*' representa el código *FOURCC* de la cadena '*RIFF*'.
2. *tamañoArchivo* es valor de cuatro bytes que indica el tamaño de los archivos en el archivo; este valor considera el tamaño del código de *tipoArchivo* y de datos, pero no incluye los cuatro bytes de este valor, ni el código anterior (de la cadena '*RIFF*').
3. *tipoArchivo* es un código *FOURCC* que identifica el tipo de archivo.
4. Los datos son listas y bloques de datos, los cuales aparecen en cualquier orden.

Los trozos de datos se forman de la siguiente manera:

bloqueID bloqueTamaño bloqueDatos

1. *bloqueId* es un código *FOURCC* que identifica este bloque.
2. *tamañoBloque* es un valor de cuatro bytes que indica el tamaño de los datos del bloque de datos. Este valor no incluye el código de *bloqueId*, el de este dato, ni del relleno de bloque datos; indica sólo el tamaño de datos válidos de *bloqueDatos*, tal y como se menciona abajo.
3. *bloqueDatos* son cero o más bytes de datos; este bloque se rellena, para que el tamaño del bloque sea un múltiplo de la longitud de una palabra de datos.

Las listas tienen la siguiente sintaxis:

'*LIST*' *tamañoLista tipoLista datosLista*

1. '*LIST*' es código *FOURCC* de dicha cadena.
2. *tamañoLista* es un valor de cuatro bytes que indica el tamaño de la lista; este valor incluye los valores subsecuentes, pero excluye el anterior código *FOURCC* y los cuatro bytes de este código.
3. *tipoLista* es un código *FOURCC*.
4. *datosLista* consiste en bloques de datos o listas, presentados en cualquier orden.

Para terminar de explicar el formato *AVI*, se simplifica la notación de los bloques de datos de la siguiente manera:

bloqueId (*bloqueDatos*), donde se considera que el tamaño del bloque es implícito.

De forma similar, las listas se representan así:

'*LIST*' (*tipoLista* (*listaDatos*))

Los elementos opcionales estarán dentro de corchetes [].

E.2.3 Formato *RIFF* para *AVI*.

Los archivos de tipo *AVI* contienen el código *FOURCC* '*AVI*' en encabezado *RIFF*. Todos los *AVI* tienen dos listas, las cuales definen el formato de los flujos y el flujo de datos, respectivamente; también puede tener bloque que funge como índice, la cual indica la posición de los bloques de datos dentro de este archivo. La estructura antes descrita, donde la secuencia de los componentes no debe ser alterada, se muestra a continuación:

```
RIFF ('AVI '
        LIST ('hdrl' ... )
        LIST ('movi' ... )
         ['idx1' (<AVI Index>) ]
        )
```

1. La lista '*hdrl*' define el formato de los datos y es el primer bloque de tipo lista requerido.
2. La lista '*movi*' contiene los datos para la secuencia *AVI*, y es el segundo bloque de tipo lista requerido.
3. La lista '*idx1*' contiene el índice.

Las listas '*hdrl*' y '*movi*', a su vez, se componen de bloques de datos; estas estructuras se muestran con mayor detalle a continuación:

```
RIFF ('AVI '
    LIST ('hdrl'
        'avih'(<Encabezado principal AVI>)
        LIST ('strl'
            'strh'(<Encabezado de flujo>)
            'strf'(<Formato de flujo>)
            [ 'strd'(<Encabezado de datos adicionales>) ]
            [ 'strn'(<Nombre del flujo>) ]
            ...
           )
         ...
        )
    LIST ('movi'
        {SubBloque | LIST ('rec '
                    SubBloque1
                    SubBloque2
                    ...
                   )
          ...
        }
```

```
        ...
        )
    ['idx1' (<Índice AVI>) ]
    )
```

E.2.3.1 Encabezado principal AVI.

La lista '*hdrl*' comienza con este encabezado, el cual se encuentra en un bloque de tipo '*avih*'. Este encabezado contiene información global usada para todo el archivo *AVI*, tal como el número de flujos dentro del archivo, y el ancho y largo de la secuencia *AVI*, y su estructura '*AVIMAINHEADER*' es la siguiente:

```
typedef struct _avimainheader {
    FOURCC fcc;
    DWORD  cb;
    DWORD  dwMicroSecPerFrame;
    DWORD  dwMaxBytesPerSec;
    DWORD  dwPaddingGranularity;
    DWORD  dwFlags;
    DWORD  dwTotalFrames;
    DWORD  dwInitialFrames;
    DWORD  dwStreams;
    DWORD  dwSuggestedBufferSize;
    DWORD  dwWidth;
    DWORD  dwHeight;
    DWORD  dwReserved[4];
} AVIMAINHEADER;
```

Los miembros y su significado se mencionan a continuación:

1. *fcc*, especifica un código *FOURCC*, que debe tener el valor '*avih*'.
2. *cb*, indica el tamaño de la estructura, excluyendo los primeros ocho bytes iniciales.
3. *dwMicroSecPerFrame*, señala el número de microsegundos entre cuadros, acompasando a todo el archivo.
4. *dwMaxBytesPerSec*, especifica el tamaño máximo de la tasa de datos del archivo que el sistema debe manejar para presentar la secuencia *AVI* tal y como la especifican otros parámetros contenidos en este encabezado y en los encabezados de los bloques de flujo.
5. *dwPaddingGranularity*, es un valor en bytes que alinea los datos; los datos se rellenan con múltiplos de este valor.
6. *dwFlags*, contiene una combinación de cero o más de los siguientes valores:

Valor:	Descripción:
AVIF_COPYRIGHTED	Indica si el archivo AVI contiene datos y/o *software* con registro de autor; si la bandera está presente, el software deberá impedir la copia de datos.
AVIF_HASINDEX	Señala si el archivo contiene un índice.

AVIF_ISINTERLEAVED	Indica si el archivo es intercalado.
AVIF_MUSTUSEINDEX	Indica que la aplicación debe usar el índice, en vez que el orden físico de los bloques dentro del archivo, para determinar el orden de presentación de los datos.
AVIF_WASCAPTUREFILE	Indica que el archivo está colocado para captura de video en tiempo real. Las aplicaciones deben advertir al usuario antes de escribir sobre este archivo, cuando la bandera está presente, porque podría defragmentar este archivo.

Tabla 20: Combinación de valores de *dwFlags*.

7. *dwTotalFrames*, especifica el número total de cuadros en el archivo.
8. *dwInitialFrames*, indica el cuadro inicial para archivos intercalados; los archivos intercalados deben indicar el valor cero; cuando se crear archivos, se deberá indicar el número de cuadros en el archivo antes del cuadro inicial.
9. *dwStreams*, especifica el número de flujos en el archivo; por ejemplo, un archivo con audio y video tiene dos flujos.
10. *dwSuggestedBufferSize*, especifica el tamaño de la memoria intermedia para leer el archivo; generalmente es mayor al mayor bloque en el archivo. Si se indica que sea cero o un valor pequeño, el software deberá que reasignar suficiente memoria durante la reproducción, lo cual reduce el rendimiento. Para un archivo intercalado, el tamaño de la memoria intermedia deberá de ser un tamaño suficientemente grande para contener todo un registro al momento de la lectura.
11. *dwWidth*, indica el ancho de los cuadros del archivo, en *pixeles*.
12. *dwHeight*, señala el alto de los cuadros del archivo, en *pixeles*.

El encabezado debe estar en el archivo *Aviriff.h* .

E.2.3.1.1. Encabezados de flujos AVI.

Después del encabezado principal pueden ir uno o más listas '*strl*'; cada una de ellas es requerida para cada flujo de datos, debe contener un bloque del encabezado del flujo y un bloque del formato del flujo. También puede contener un bloque de datos del encabezado del flujo de tipo '*strd*' y un bloque del nombre del flujo '*strn*'. El encabezado '*strh*' tiene una estructura de tipo *AVISTREAMHEADER*, la cual tiene la siguiente sintaxis:

```
typedef struct _avistreamheader {
    FOURCC fcc;
    DWORD  cb;
    FOURCC fccType;
    FOURCC fccHandler;
    DWORD  dwFlags;
    WORD   wPriority;
    WORD   wLanguage;
    DWORD  dwInitialFrames;
    DWORD  dwScale;
    DWORD  dwRate;
    DWORD  dwStart;
```

```
    DWORD  dwLength;
    DWORD  dwSuggestedBufferSize;
    DWORD  dwQuality;
    DWORD  dwSampleSize;
    struct {
       short int left;
       short int top;
       short int right;
       short int bottom;
    }  rcFrame;
} AVISTREAMHEADER;
```

Los miembros del encabezado del flujo se explican a continuación:

1. *ffc*, especifica un código *FOURCC* cuyo valor es '*strh*'.
2. *cb*, especifica el tamaño de la estructura, excluyendo los ocho bytes iniciales.
3. *ffcType*, contiene un código *FOURCC* que especifica el tipo de dato contenido en el flujo; el valor de dicho código puede ser alguno de los siguientes estándares *AVI* para audio o video:

FOURCC	**Descripción:**
'*auds*'	Flujo de audio.
'*mids*'	Flujo de tipo MIDI.
'*txts*'	Flujo de texto.
'*vids*'	Flujo de video.

Tabla 21: Descripción de los códigos *FOURCC* para *ffcType*.

4. *fccHander*, es un atributo opcional que contiene un controlador específico de datos, al cual se le da preferencia para controlar el flujo. Para flujos de audio y video especifica el códec para decodificar el flujo.
5. *dwFlag*, contiene diversas banderas para el flujo de datos; los bits más significativos de la palabra son específicos para el tipo de datos contenidos en el flujo. Se han definido las siguientes banderas estándares:

Valor	**Descripción**
AVISF_DISABLED	Indica que el flujo no debe estar habilitado por defecto.
AVISF_VIDEO_PALCHANGES	Indica que el flujo de video tiene cambios de paleta, para que el programa reproductor cambie la paleta.

Tabla 22: Descripción de las banderas para *dwFlag*.

6. *wPriority*, especifica la prioridad de un tipo de flujo; si un archivo tiene varios flujos, el de mayor prioridad será el flujo por defecto.
7. *wLanguage*, etiqueta del idioma.

8. *dwInitialFrames*, indica que tanto difieren los datos de audio respecto de los de video en archivos intercalados; generalmente es de .75 segundos.
9. *dwScale*, es un valor usado en conjunto con *dwRate*, para especificar la escala de tiempo a usar en este flujo. Al dividir *dwRate* entre *dwScale* se obtiene el número de muestras por segundo. Para los flujos de video ese valor se conoce como la tasa de transferencia; en el caso de flujos de audio, dicho valor es equivalente al tiempo necesario para reproducir *nBlockAlign* bytes de audio; en PCM audio este valor es la tasa de muestreo.
10. *dwRate*, se explica junto con *dwScale*.
11. *dwStart*, indica cuándo inicia el flujo; sus unidades se especifican por *dwRate* y *dwScale*, Generalmente tiene un valor de cero, pero puede ser mayor para indicar un retraso para este flujo que no inicia al mismo tiempo que el archivo.
12. *dwLength*, indica el tamaño de este flujo, y sus unidades se especifican con *dwRate* y *dwScale*.
13. *dwSuggestedBufferedSize*, indica el tamaño recomendado para la memoria intermedia que lea este flujo; debe ser igual al tamaño del bloque de datos más grande. Si el valor está correctamente indicado, la reproducción del flujo será más eficiente; si se desconoce el valor más adecuado, se le debe asignar cero.
14. *dwQuality*, indica un factor de calidad de los datos del flujo; tiene un valor entre cero y 10,000. Para datos comprimidos indica el valor del parámetro pasado al software que comprime; si tiene valor -1, los controladores usan el valor por defecto.
15. *dwSampleSize*, indica el tamaño de una muestra de datos. Debe ser cero si las muestras pueden variar de tamaño, y cada muestra debe estar en un bloque separado de las demás; si no es cero, varias muestras de datos pueden ser agrupadas en un sólo bloque dentro del archivo. Para los flujos de video, su valor es cero, generalmente; puede ser distinto de cero si todos los cuadros de video tienen el mismo valor. Para flujos de audio, este valor debe ser igual que el de *nBlockAlign*, de la estructura *WAVEFORMATEX*, la cual describe el audio.
16. *rcFrame*, especifica el rectángulo destino para el texto o flujo de video dentro del rectángulo de la película, cuyas dimensiones son señaladas por los atributos *dwWidth* y *dwHeigth* del encabezado principal. Generalmente se usa cuando hay varios flujos de video. Este rectángulo se fija a las coordenadas correspondientes del rectángulo de la película con el fin de actualizar todo este último. Sus unidades se indican en pixeles, y la esquina superior izquierda es relativa a la esquina superior izquierda del rectángulo de la película.

Algunos de los atributos mencionados también se encuentran en la estructura *AVIMAINHEADER*, pero la diferencia es que los valores indicados en *AVISTREAMHEADER* aplican sólo a un flujo. Este encabezado se indica en Aviriff.h .

Después del bloque del encabezado del flujo, debe seguir un bloque del formato del flujo '*strf*'; este encabezado describe el formato de los datos en el flujo. Los datos contenidos en este bloque dependen del tipo del flujo; para flujos de video, la información es una estructura de tipo *BITMAPINFO*, que puede incluir información de la paleta de colores. Si son flujos de audio, la información viene en una estructura *WAVEFORMATEX*, la cual se describe a continuación.

E.2.3.1.1. Formato de audio WaveFormatex.

Es una estructura que define la estructura de datos de audio con forma de onda, y sólo se incluye información común a todos los formatos de ondas de audio. Para formatos que requieran información adicional, su estructura es incluida como primer miembro en otra estructura, junto con la información adicional. Se declara en Mmreg.h, y su sintaxis es la siguiente:

```
typedef struct {
WORD wFormatTag;
WORD nChannels;
DWORD nSamplesPerSec;
DWORD nAvgBytesPerSec;
WORD nBlockAlign;
WORD wBitsPerSample;
WORD cbSize;
} WAVEFORMATEX;
```

Los atributos de esa estructura son:

1. *wFormatTag*, indica el tipo del formato de audio con forma de onda; se registran con *Microsoft Corporation* para su uso en muchos algoritmos de compresión.

2. *nChannels*, señala el número de canales para el audio con forma de onda. Los datos de tipo monoaural tiene un canal, y los datos en estéreo usan dos canales.
3. *n SamplesPerSec*, es la frecuencia de muestreo, su medida son los hertz. Si el valor de *wFormatTag* es de tipo *WAVE_FORMAT_PCM*, los valores más comunes que puede tener son 8, 11.025, 22.05 y 44.1 [kHZ]; para formatos que no sean *PCM*, su valor se calcula de acuerdo con la especificación de *wFormatTag*.
4. *nAvgBytesPerSec*, proporciona la tasa promedio de transferencia de datos en bytes por segundo a la etiqueta de formato. Si el valor de *wFormatTag* tiene el valor de *WAVE_FORMAT_PCM*, este atributo debe ser igual que el producto de *nSamplesPerSec* y de *nBlockAlign*. Para formatos que no sean *PCM*, su valor se calcula de acuerdo con las especificaciones del fabricante.
5. *nBlockAlign*, indica la alineación de los bloques en bytes, la cual es la unidad atómica mínima usada para los valores de la etiqueta *wFormatTag*; si esta última tiene valores de *WAVE_FORMAT_PCM* o *WAVE_FORMAT_EXTENSIBLE*, *nBlockAlign* debe ser igual al producto de *nChannels* y de *wBitesPerSample*, dividido entre ocho (para obtener el resultado en bytes). Si el formato no es *PCM*, su valor se calcula de acuerdo a las especificaciones del fabricante. Los reproductores deben procesar una cantidad de bytes de bytes, que es múltiplo de este valor, cada vez, y es ilegal leer a la mitad de una muestra (fuera de los límites del bloque alineado).
6. *wBitPerSample*, da los bits por muestra para el valor de la etiqueta *wFormatTag*. Si el valor de la última es *WAVE_FORMAT_PCM*, este atributo debe ser igual a 8 ó a 16; en caso de que no sea formato PCM, su valor deberá ser el indicado por el fabricante. En caso de que *wFormatTag* sea *WAVE_FORMAT_EXTENSIBLE*, su valor puede ser múltiplo de ocho. Algunos algoritmos de compresión no pueden definir un valor para este atributo, por lo que le asigna cero.
7. *cbSize*, indica el tamaño en bytes de la información extra añadida al final de esta estructura. Los formatos que no sean *PCM* pueden usar este atributo para guardar información extra de *wFormatTag*; si no hay información extra, deberá tener valor cero. Los formatos *WAVE_FORMAT_PCM* ignoran el valor aquí indicado.

Esta estructura se guarda en *Dshow.h* . Existe otra estructura, *WAVEFORMATEXTENSIBLE*, la cual describe los formatos con más de dos canales o resoluciones más altas de muestreo que las que puede contener *WaveFormatex*; ya no se muestra aquí, ya que se pretende describir el formato *AVI*, mas que todos los formatos existentes.

Si el bloque del encabezado de flujo de datos '*strd*' existe, le sigue un bloque del formato de flujo, cuyo formato y contenidos son definidos por los controladores del '*codec*'; estos controladores usan esta información para configurar, la cual viene en una cadena de texto que no tiene terminador nulo, y que describe el flujo. Las aplicaciones que leen y escriben los archivos *AVI* no interpretan esta información, y sólo la transfieren del y hacia el controlador como bloques de memoria.

Los encabezados de flujo en la lista '*hdrl*' se relacionan con los flujos de datos de la lista '*movi*', de acuerdo con el orden de los bloques '*strl*'; el primer bloque '*strl*' se asocia con el flujo cero, el segundo bloque con el flujo uno, y así sucesivamente.

E.2.3.2. El flujo de datos (lista '*movi*').

Después del bloque del encabezado, se encuentra una lista '*movi*', la cual contiene los datos de los flujos (los cuadros de video y las muestras de sonido). Los bloques de datos pueden encontrarse directamente en esta lista, o estar agrupados en listas '*rec*'; al hacer esto último, los bloques de datos pueden leerse del disco al mismo tiempo, tal como la multimedia que se lee desde CD-ROM.

Los códigos *FOURCC* que identifican a cada bloque de datos se componen por dos dígitos para el número de flujo, y después viene un código de dos caracteres, que indican el tipo de información de este bloque de datos; estos últimos códigos se muestran a continuación:

Código de dos caracteres.	**Descripción.**
db	Cuadros de video sin comprimir.
dc	Cuadros de video comprimidos.
pc	Cambio de paleta.
wb	Datos de audio.

Tabla 23: Descripción de los códigos *FOURCC* para los bloques de la lista '*movi*'.

Los bloques de datos pueden definir otra paleta durante la reproducción de un video *AVI*; los datos del bloque de la nueva paleta ('*xxpc*') están en una estructura *AVIPALCHANGE*, y además cambia la bandera *AVISF_VIDEO_PALCHANGES* -que viene en el atributo *dwFlags* de la estructura *AVISTREAMHEADER* de ese flujo. Los flujos de texto pueden usar arbitrariamente códigos de dos caracteres. La estructura *AVIPALCHANGE* es la siguiente:

```
typedef struct
{
   BYTE        bFirstEntry;
   BYTE        bNumEntries;
   WORD        wFlags;
   PALETTEENTRY  peNew[];
} AVIPALCHANGE;
```

Se guarda en *Aviriff.h*, y sus miembros (atributos) son los siguientes:

1. *bFirstEntry*, señala el índice de la primera entrada a cambiar.
2. *bNumEntries*, indica el número de entradas a cambiar, o puede ser cero con el fin de cambiar las 256 entradas de la paleta.
3. *wFlag*, atributo reservado.
4. *peNew*, contiene un arreglo de estructuras *PALETTEENTRY*, del tamaño de *bNumEntries*.

E.2.3.3. Entradas de índice *AVI*.

El bloque opcional del índice '*idx1*' debe estar presente después de la lista '*movi*'. Ese índice contiene una lista de los bloques de datos y de su ubicación en el archivo, la cual se organiza en una estructura *AVIOLDINDEX*, que contiene una entrada por bloque de datos, incluyendo los de tipo '*rec*'. Si el archivo contiene un índice, se usa la bandera *AVI_HASINDEX*, del atributo *dwFlags* de la estructura *AVIMAINHEADER*. La sintaxis de *AVIOLDINDEX* es:

```
typedef struct _avioldindex {
  FOURCC  fcc;
  DWORD   cb;
  struct _avioldindex_entry {
    DWORD   dwChunkId;
    DWORD   dwFlags;
    DWORD   dwOffset;
    DWORD   dwSize;
  } aIndex[];
} AVIOLDINDEX;
```

Describe el índice de un archivo *AVI 1.0* (con formato '*idx1*'); los nuevos archivo de tipo *AVI (2.0)* deben usar el formato '*indx*'; sus atributos son los siguientes:

1. *fcc*, indica un código *FOURCC*, cuyo valor debe ser '*idx1*'.
2. *cb*, indica el tamaño de la estructura, excluyendo los primeros ocho bytes de *fcc*.
3. *dwChungkId*, indica un código *FOURCC* que identifica un flujo dentro del archivo *AVI*; debe tener la forma 'xxyy' donde *xx* es el número del flujo, y *yy* es un código de dos caracteres que identifica el contenido del flujo, y sus posibles valores son:

Código de dos caracteres.	**Descripción.**
db	Cuadros de video sin comprimir.
dc	Cuadros de video comprimidos.
pc	Cambio de paleta.
wb	Datos de audio.

Tabla 24: Descripción de los códigos de dwChungkId.

4. *dwFlags*, indica una combinación de cero o más de las siguientes banderas:

Valor.	**Descripción.**
AVII_KEYFRAME	El bloque de datos es marco principal.
AVIIF_LIST	El bloque de datos es una lista 'rec'.
AVIIF_NO_TIME	El bloque de datos no afecta a la sincronización del flujo.

Tabla 25: Descripción de las banderas de dwFlags.

5. *dwOffset*, indica la posición del bloque de datos en el archivo, señalando cuántos bytes debe ignorar desde le inicio del encabezado '*movi*', para llegar al bloque. Algunos archivos *AVI* calcular este valor contando a partir del principio del archivo.
6. *dwSize*, señala el tamaño del bloque de datos en bytes.

Esta estructura comienza con un bloque inicial *RIFF* (para *ffc* y *cb*) seguida de una entrada para cada bloque de datos en la lista '*movi*'. Se encuentra en el encabezado *Aviriff.h* .

E.2.3.4. Otros bloques de datos.

Los datos pueden ser alineados dentro de archivo ***AVI***, insertando tantos bloques '*JUNK*' (basura) como sean necesarios; sirven de relleno, y las aplicaciones que reproduzcan este tipo de video deberán ignorar estos bloques.

Anexo F. Referencia de la interfaz *Java Media Framework* (*JMF*).

En **Conceptos Básicos** se han descrito los elementos y formatos usados para obtener una imagen, por lo que ahora se explicará este API -la cual presenta clases, código nativo (binarios de un Sistema Operativo específico) y otros recursos- que permite manejar a los elementos mencionados anteriormente. También se explicará una serie de definiciones, para entender cómo algunas clases del sistema funcionan o colaboran con este API.

Este 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 este 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 el API existente, pero a la vez que pueda integrar nuevas características.
- Incorpora el 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.

Anexo F.1. *JMF* y la multimedia.

La multimedia consiste en audio e imágenes cuyo contenido varía o depende del tiempo, tal como las secuencias de audio y/o video. El origen de la multimedia puede ser archivos locales o en red, cámaras, micrófonos o transmisiones en vivo. El modelo fundamental del procesamiento de datos multimedia es el siguiente:

Figura 45: *Modelo del procesamiento multimedia de JMF.*

Anexo F.1.1. Flujos de datos multimedia.

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. Las películas y las pistas de sonido son ejemplos de lo anterior, ya que no deben ser reproducidas rápidamente (porque no se podrían apreciar), ni demasiado lento o con pausas anormales -ya que perdería la sensación de continuidad de la película o de la música.

Anexo F.1.1.1. Tipo de contenido.

El tipo de contenido es el formato en que se almacena la multimedia, y se relaciona con el tipo de archivo

(la extensión del archivo).

Anexo F.1.1.2. Flujos de multimedia.

Los flujos de multimedia pueden tener distintos orígenes, tales como archivos locales, obtenidos por la red, u obtenidos por medio de un dispositivo de captura; cada flujo se identifica por su origen y por el protocolo usado para tener acceso a él, tal como *FILE* o *http*. En caso de que no se pueda usar una *URL*, el flujo se puede obtener por medio de un localizador de media. 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.

Anexo F.1.1.2.1. Medios de presentación.

Los medios de presentación, o destinos, son los dispositivos en que se presentan los datos de multimedia, tales como bocinas, monitores, o archivos.

Un flujo de multimedia, sin importar su origen, contiene varios canales de datos llamados pistas; una película MPEG o de Quicktime, contiene una pista de video y otra de audio. Los flujos con múltiples pistas se conocen como flujos multiplexados o completos; el proceso de extraer una pista en particular de un flujo completo se conoce como demultiplexar.

Cada pista tiene un tipo de dato que identifica el tipo de datos multimedia que contiene; el formato de la pista indica cómo están estructurados de los datos.

Anexo F.1.1.2.2. Controles de presentación.

Al presentar un flujo de datos, la API provee la funcionalidad para manejar la reproducción del mismo, similares a las que tienen los aparatos actuales, como los controles de video caseteras y reproductores de DVD; dichas funciones son para iniciar, detener, avance rápido y re- enbobinado del flujo.

Anexo F.1.1.2.3. Latencia.

El lapso que transcurre entre el momento en que se solicita el inicio de reproducción del medio, y en el que inicia el despliegue del mismo, se llama latencia de inicio; esto es más frecuente cuando el medio se solicita a través de una red. La latencia es un factor importante cuando un medio tiene más de una pista de datos, como una película con sonido, ya que se deben sincronizar los flujos de dichas pistas.

Anexo F.1.1.2.4. Calidad de la presentación.

La calidad de la presentación del medio, depende de los siguientes factores:

- Compresión usada.
- La capacidad de procesamiento del reproductor.
- Ancho de banda disponible.

Generalmente, mientras mayor sea la calidad, el archivo es mayor y requiere de mayor poder de procesamiento y ancho de banda – también conocido como tasa de transferencia de bits, ya que es el número de bits transferidos en un periodo de tiempo. Para obtener la mejor calidad de video, se recomienda que el número de cuadros de video desplegados en un periodo de tiempo sea el mayor posible; generalmente dicho valor es de 30 cuadros de video por segundo, para que su calidad sea como la de transmisiones de televisión o reproducción de cintas de video.

Anexo F.1.1.2.5. Procesamiento de video.

Generalmente, los datos del flujo multimedia son manipulados antes de ser desplegados al usuario; las operaciones más comunes de procesamiento son las siguientes:

1. Si el flujo multimedia está multiplexado, cada pista individual es extraída.
2. Si las pistas individuales están comprimidas, se decodifican.
3. Si es requerido, las pistas de datos son convertidas a otro formato.
4. Se pueden aplicar filtros a las pistas decodificadas, con el fin de aplicar un efecto.

Después de aplicar alguno de los procesos anteriores, las pistas son entregadas a sus correspondientes dispositivos de salida, para que sean presentadas. En caso de que se desee los flujos de multimedia, los pasos varían; por ejemplo, para guardar en un archivo audio y video capturados por una videocámara, los pasos a seguir son los siguientes:

1. Se capturan las pistas de audio y de video.
2. Se pueden aplicar filtros a las pistas “crudas”, con el fin de lograr un efecto.
3. Cada pista se codifica individualmente.
4. Las pistas comprimidas (codificadas) se multiplexan en un solo flujo multimedia.
5. El flujo multimedia multiplexado se guarda en un archivo.

Anexo F.1.1.2.6. Multiplexores y demultiplexores.

Un demultiplexor extrae un una pista multimedia de un flujo multimedia multiplexado. Un multiplexor lleva a cabo la función contraria, fusionando varias pistas de datos multimedia en un solo flujo multimedia.

Anexo F.1.1.2.7. Códecs.

Un códec (abreviación de *enc**o**der-**dec**oder*) efectúa la compresión y descompresión de datos multimedia; al codificar una pista de datos, se le da un formato en el cual se comprime con el fin de poderla guardarla o

transmitirla. Al decodificar la pista, se descomprime con el fin de obtener los datos crudos y poderlos presentarlos. Cada códec puede manejar ciertos formatos entrada y otros de salida, y si es necesario se pueden utilizar varios códecs.

Anexo F.1.1.2.8. Filtros de efecto.

Estos filtros modifican la pista de datos con el fin de obtener un efecto especial, como hacer una imagen borrosa, quitar ruido de una imagen, etcétera; se aplican mediante operaciones matriciales. Los filtros de efecto se dividen en filtros pre-proceso y en post-proceso, dependiendo si se aplican antes o después de procesar la pista con un códec; generalmente se aplican a los datos sin comprimir ("crudos").

Anexo F.1.1.2.9. Dispositivos de despliegue (renderers).

Estos dispositivos son abstracciones de los dispositivos de presentación -como la tarjeta de sonido para audio, o el monitor para video.

Anexo F.1.1.2.10. Composición.

La composición es el proceso multimedia de combinar múltiples pistas de datos en un sólo medio de presentación, como sobreponer texto en video. Este proceso puede llevarse a cabo en *hardware* o en *software*, los cuales fueron diseñados para esta funcionalidad; estos dispositivos especializados se abstraen como un dispositivo de despliegue que puede recibir varias pistas de datos.

Anexo F.1.2. Captura de multimedia.

La captura de multimedia ocurre cuando se registran los datos de entrada de una fuente en vivo, para su procesamiento y reproducción; al momento de la recepción de los datos, éstos se pueden capturar y manipular en pistas multimedia distintas, o dejarlos todos juntos en una sola pista multiplexada, tal y como es el caso de la captura de audio y video desde una videocámara. La captura de los datos puede ser tan difícil como es la fase de entrada del modelo de procesamiento multimedia estándar.

Anexo F.1.2.1. Dispositivos de captura.

Son elementos de hardware especializados en recoger datos multimedia, tal como el micrófono y la tarjeta de sonido para el audio, o la tarjeta de sintonización de televisión para capturar transmisiones televisivas. Los sistemas operativos tienen forma de reconocer cuáles dispositivos de este tipo tienen a su disposición. Estos elementos se clasifican de la siguiente forma:

- Fuentes de emisión, las cuales envían continuamente información.
- Fuentes de por demanda, en las cuales el usuario (mediante controles) solicita los datos.

El formato en que se guarda la multimedia capturada depende del procesamiento que lleve a cabo el dispositivo de captura; algunos dispositivos pueden dejar los datos sin comprimir ni procesar ("crudos"), mientras que otros pueden dejar los datos en un formato comprimido.

Anexo F.1.2.2. Controles de captura.

Son elementos que permiten al usuario manipular el proceso de captura, tales como detener o iniciar la captura; generalmente se le provee al usuario de estos controles.

Anexo F.2. Arquitectura de la Java Media Framework.

A continuación se explicará como esta estructurada este API, así como el comportamiento de la misma.

Anexo F.2.1. Arquitectura de alto nivel.

Este API tiene 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.

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

Anexo F.2.1.1 Modelo de tiempo.

La manipulación de multimedia está intrínsecamente relacionada con el tiempo; JMF utiliza a la clase *Time* para representar un punto en particular del tiempo, pero JMF puede tener precisión de milisegundos.

Las clases que dan soporte al modelo de tiempo de JMF implantan la clase *Clock*, para poder seguir la secuencia de un flujo multimedia durante su reproducción; dicha interfaz define las operaciones básicas de sincronización y calendarización requeridos para controlar la presentación multimedia.

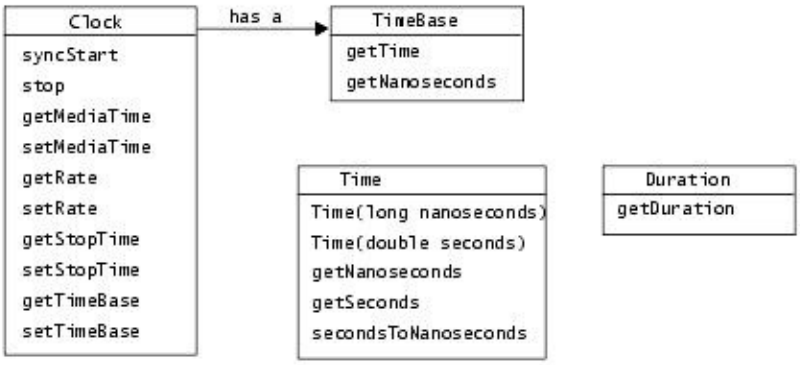


Figura 46: *Diagrama de clases del Modelo de Tiempo para JMF.*

La clase *Clock* se sirve de la clase *TimeBase* para medir cómo transcurre el tiempo mientras se presenta la multimedia; la segunda clase marca intervalos regulares de tiempo, brinda la hora actual (conocida también como el tiempo base); no puede detenerse ni reiniciarse, y generalmente se obtiene de la hora del sistema operativo.

El atributo tiempo media de un objeto de tipo *Clock* indica la posición actual dentro de un flujo multimedia; el inicio de la multimedia tiene un tiempo media cero, mientras que el fin del flujo es el máximo del tiempo media. La duración del flujo multimedia es el tiempo transcurrido desde el inicio hasta el fin de la presentación del medio. Para poder obtener valores actualizados del tiempo media, la clase *Clock* emplea los siguientes valores:

- El tiempo de inicio del tiempo base, es decir, cuando inicia la reproducción.
- El tiempo de inicio del medio, el cual señala en qué parte del flujo multimedia inició la reproducción.
- La tasa de reproducción, lo que indica qué tan rápido se ejecuta la clase *Clock* en relación con su objeto de tipo *TimeBase*; esta tasa es un factor de escala aplicado a *TimeBase*, para indicar que tan rápido se presenta el flujo multimedia, e incluso puede representar el sentido del flujo (valores negativos indica que el flujo se reproduce al revés).

Cuando la representación inicia, se convierte el tiempo del medio al tiempo base, y el avance de este último se usa para medir el transcurrir del tiempo. Durante la reproducción, el tiempo del medio se calcula usando la siguiente fórmula:

TiempoMedia=TiempoMediaInicial+TasaTransferencia(TiempoBaseActual - TiempoBaseInicial)

Cuando la presentación se detiene, el tiempo del medio es detenido también, pero el tiempo base continúa en su avance. Si se reinicia la presentación, el tiempo del medio se vuelve a calcular con el valor actual del tiempo base.

Anexo F.2.1.2. Administradores.

La API JMF consiste básicamente de interfases 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. JMF 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*.

Anexo F.2.1.3. Modelo de eventos.

JMF tiene un mecanismo estructurado de eventos para informar a las aplicaciones que hacen uso de este API para avisar el estado actual del sistema de medios, y también hace posible que dichas aplicaciones puedan manejar situaciones tales como errores provocados por la media (sin datos disponibles o fuente de datos fuera de alcance). En estos casos se envía un evento de tipo *MediaEvent*, el cual sigue todo el modelo JavaBean de eventos; la clase *MediaEvent* se extiende para poder reportar más eventos particulares.

Así mismo, por cada tipo de objeto que propaga eventos de tipo *MediaEvent*, existe su correspondiente interfaz auditora (*listener*). Este mecanismo es igual al manejo de eventos del resto del lenguaje Java, un objeto propaga los eventos, mientras que otra clase implanta el auditor correspondiente, la cual se registra en la clase que va a enviar los eventos invocando el método *addListener*.

Objetos de tipo *Controller* (como los reproductores y procesadores), *RTPSessionManager*, y algunos de tipo *Control* (tal como *GainControl*) envían eventos de tipo *MediaEvent*.

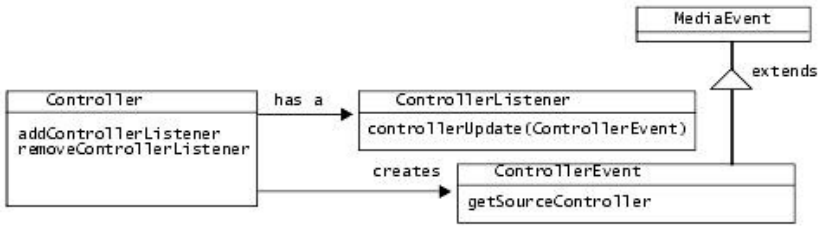


Figura 47: *Diagrama de clases del modelo de eventos de JMF.*

Anexo F.2.1.4. Modelo de datos.

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 el software usados para entregar la media. Una vez obtenido, un *DataSource* no puede reutilizar 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 los siguientes objetos de tipo *DataSource*:

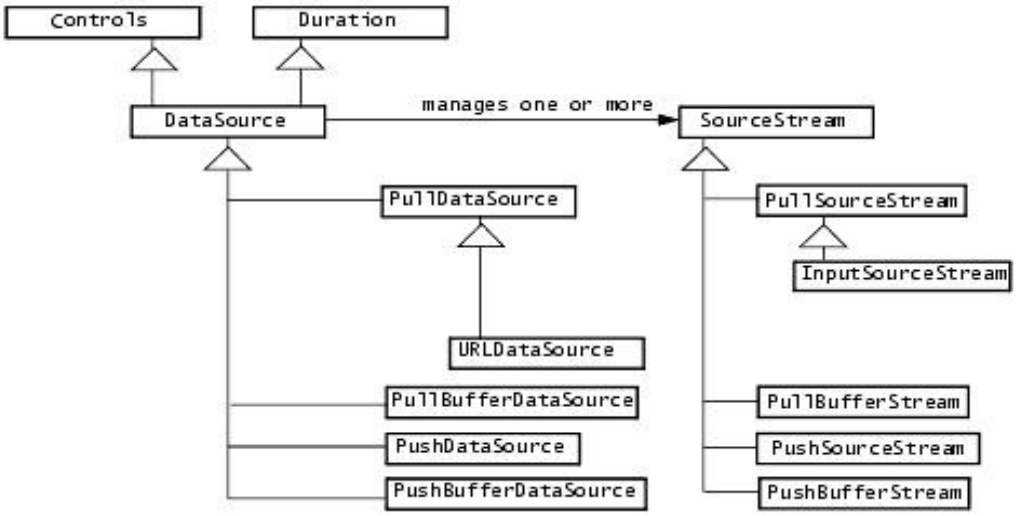


Figura 48: *Diagrama de clases de los tipos derivados de DataSource.*

Anexo F.2.1.4.1. Fuentes de datos por demanda y de emisión.

Los datos multimedia pueden tener distintos orígenes, tales como archivos locales, archivos obtenidos por la red, o transmisiones en vivo; estas fuentes de datos se clasifican por el modo en que inicia la transferencia de datos:

- Fuente de datos por demanda (*pull Data-Source*), en la cual el cliente inicia la transmisión y controla el flujo de datos desde la fuente, tales como los protocolos *HTTP* y *FILE*; JMF define *PullDataSource* y *PullBufferDataSource* como fuentes de este tipo, las cuales usan un objeto *Buffer* como unidad de transferencia.
- Fuente de datos de emisión (*push Data-Source*), en la cual el servidor inicia la transferencia de datos y controla el flujo de datos, como las transmisiones multimedia, la multi-emisión de media, y el video en demanda (*video-on-demand*, VOD); para el primer ejemplo, se usa el protocolo de transporte en tiempo real (*Real-time Transport Protocol*, RTP), el cual desarrolla el *Internet Engineering Task Force* (IETF). VOD emplea el protocolo *MediaBase*, desarrollado por SGI. JMF define dos fuentes de tipo, *PushDataSource* y *PushBufferDataSource*, los cuales usan un objeto *Buffer* como unidad de transferencia.

De la clasificación anterior se determina el tipo de control que se tiene sobre el medio; para el primer tipo se pueden usar controles para iniciar nuevamente la reproducción o buscar una nueva posición, mientras que para el segundo tipo no se puede buscar una nueva posición dentro de su flujo -la excepción es VOD, que tiene controles limitados con los cuales buscar una nueva posición, pero no avanzar ni retroceder.

Anexo F.2.1.4.2. Fuentes de datos especiales.

La API JMF define dos fuentes de datos especiales, las cuales son fuentes de datos clonables y fuentes de datos de fusión.

Una fuente de datos clonable permite crear copias de sí misma, tanto de objetos *DataSource* por demanda como de emisión. Se crea al invocar el método *createCloneableDataSource* de *Manager*, pasando el objeto *DataSource* a clonar como parámetro. Una vez que se clone, el original no debe ser usado, y sólo se deben utilizar

el *DataSource* clonable y sus clones. Estos objetos clones implantan la interfaz *SourceClonable*, que sólo define el método *createClone*; con este último método se pueden crear cualquier cantidad de clones del objeto *DataSource* clonable.

Los clones pueden controlar por medio del objeto *DataSource* clonable, ya que al invocar a sus métodos *connect*, *disconnect*, *start* o *stop*, sus llamadas se propagan a los clones. Los clones no necesariamente tienen las mismas propiedades del *DataSource* con que construyó, o del original; un *DataSource* clonable creado de un dispositivo de captura sólo funcionará como maestro si captura datos, de lo contrario los clones no proveerán datos. Si se enlaza una fuente de datos clonable con uno o más clones, estos últimos producirán datos con la misma tasa que su fuente maestra.

La otra fuente de datos especial es *MerginDataSource*, la cual se crea mediante el administrador (*Manager*), invocando su método *createMerginDataSource* y pasando como parámetro un arreglo de las fuentes de datos a fusionar; se pueden fusionar si las fuentes son del mismo tipo. La duración de la fuente fusionada es la mayor de las duraciones de cada fuente original, y su tipo de contenido es *application/mixed-media*.

Anexo F.2.1.4.3. Formatos de datos.

El formato de media se especifica por medio del objeto *Format*, el cual no indica parámetros específicos de codificación, ni de ajustes de tiempo globales, sino que indica el nombre de la codificación del medio, así como el tipo de datos que requiere el formato.

Un formato de audio (clase *AudioFormat*) contiene atributos específicos de un formato de audio, tales como tasa de muestreo, bits por muestra y número de canales. Un formato de video (clase *VideoFormat*) guarda información relacionada con los datos de video; esta clase se extiende para poder definir otros formatos de video, mediante las siguientes clases:

- *IndexedColorFormat*.
- *RGBFormat*.
- *YUVFormat*.
- *JPEGFormat*.
- *H261Format*.
- *H263Format*.

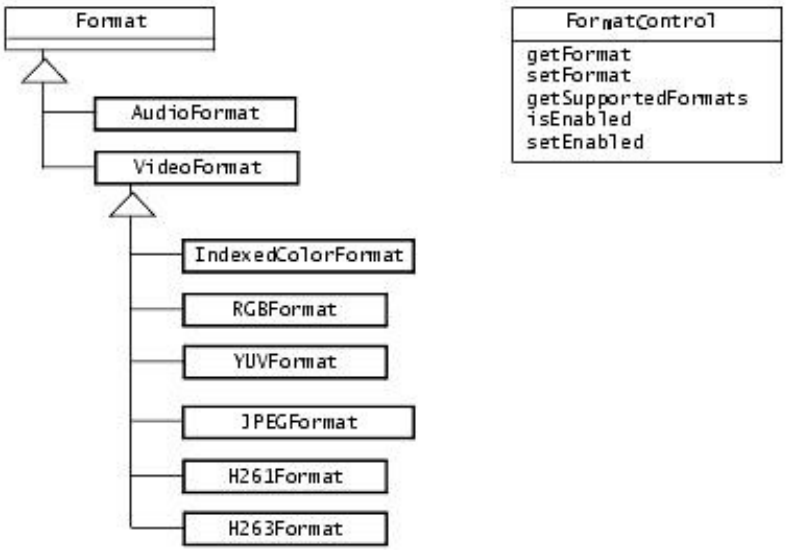


Figura 49: *Diagrama de clases de los formatos de video.*

En el caso de que se requiera recibir notificación de cambios de formato desde un controlador (*Controller*), se puede implantar la clase *ControllerListener*, para escuchar eventos de tipo *FormatChangeEvents*.

En el sistema desarrollado, se construye un objeto *MediaLocator*, al cual se le indica que se emplea Video para Windows (*VfW*), con el fin de encontrar el dispositivo de captura (videocámara). Después, se construye un *DataSource* con base en el *MediaLocator*, que recibe los datos de video, y a partir de él se obtiene un objeto de tipo *JPEGFormat*, el cual maneja el formato de video ***AVI***, pero a partir de él se puede extraer cuadros de video en formato JPEG, el cual es muy conocido, como ya se había explicado anteriormente.

Anexo F.2.1.5. Controles.

Los controles son clases que permiten asignar un obtener atributos de objetos relacionados con JMF, con el fin de que un usuario pueda manipularlos; las clases que generalmente tiene un control al cual acceder son *Controller*, *DataSource*, *DataSink* y *plug-ins* de JMF.

Anexo F.2.1.5.1. Controles estándares.

JMF define los siguientes controles estándares:

- *CachingControl*, el cual permite monitorear y mostrar el avance de la obtención de un recurso de la red; un reproductor o un procesador pueden implantar este control, para reportar el avance mediante una barra de progreso.

- *GainControl*, el cual permite cambiar el volumen, o suprimirlo; contiene un auditor para avisar de cambios en el volumen.

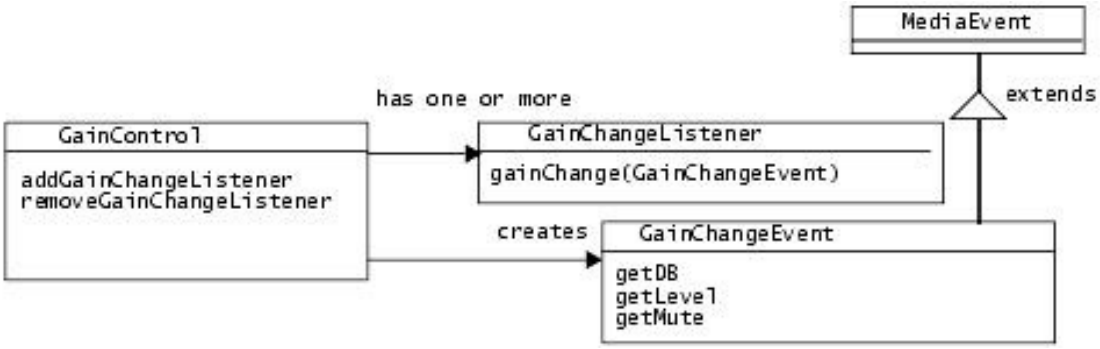


Figura 50: *Diagrama de clases de **Gain Control**.*

- *StreamWriterControl* permite limitar el tamaño del flujo creado, para que objetos como *DataSink* o *Multiplexer* puedan leer datos de media de un *DataSource* y escribirlos en un archivo.

- Los controles *FramePositicionControl* y *FrameGrabbingControl* permiten manejar cuadros de video a los reproductores y a los procesadores; el primero permite colocarse en la posición de un cuadro específico, mientras que el segundo permite obtener una imagen fija (cuadro de video) a partir del flujo de video, e incluso puede ser implantado por los dispositivos de despliegue (*renderers*).

- *FormatControl* es una interfaz que permite fijar o consultar el formato de los objetos que tengan un atributo de tipo Formato.
- *TrackControl* es un tipo de control que permite controlar los procesos llevados en un objeto de tipo

procesador (*Processor*) sobre una pista de media, tales como conversiones de formatos, efectos, códecs o *plug-ins* de despliegue que son aplicados.

- *PortControl* y *MonitorControl* permiten manejar los procesos de captura; el primero permite el control sobre la salida de un dispositivo de captura, mientras que el segundo permite tener una vista previa sobre la media recién capturada o codificada.

- *BufferControl* permite manejar la memoria intermedia de un objeto en particular.

JMF define varios otros controles de códecs, para permitir la manipulación de codificados y decodificadores (de *software* o de *hardware*), los cuales son:

- *BitRateControl*, que permite conocer la información acerca de la tasa de transferencia de bits [bits/segundo] de un flujo entrante, así como tener control sobre dicha tasa.

- *FrameProcessingControl*, permite especificar los parámetros de procesamiento de los cuadros de video para efectuar el mínimo procesamiento en un códec, cuando éste está fallando al procesar datos de entrada.
- *FrameRateControl*, permite cambiar la tasa de cuadros de video.

- *H261Control*, con el que se maneja el modo de transferencia de imágenes fijas del códec de video H.261 .

- *H263Control*, que permite manipular los parámetros del códec de video H.263, incluyendo apoyo para los de vector irrestricto, codificación aritmética, predicción avanzada, cuadros PB, y extensiones de compensación de errores.
- *KeyFrameControl*, el cual especifica el intervalo deseado entre cuadros principales (el codificador es el que finalmente indica ese valor).

- *MpegAudioControl*, el cual exporta las capacidades de un códec de audio MPEG, y permite asignar el valor de ciertos parámetros de codificación MPEG.

- *QualityControl*, el cual permite asignar los valores preferidos en la negociación entre la calidad y el uso de CPU durante el proceso llevado a cabo por un códec.

- *SilenceSuppressionControl*, con el que asignan los parámetros para la supresión de silencio para los códecs de audio; si el modo supresión de silencio está encendido, un codificador de audio no envía ninguna salida si detecta silencio como su entrada.

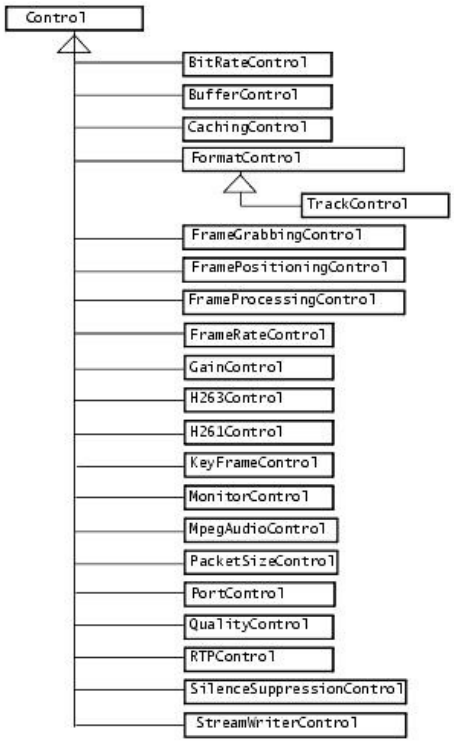


Figura 51: *Diagrama de clases de Control y de sus clases derivadas.*

Componentes de la interfaz de usuario.

Un control puede brindar una interfaz de tipo *Component*, para mostrar la funcionalidad de los mismos a un usuario final. La interfaz por defecto de un control se obtiene por medio de su método *getControlComponent*, el cual regresa un objeto de tipo *AWT Component*, la cual se puede agregar directamente en un applet o en una aplicación gráfica.

Un objeto Controller también puede proporcionar acceso a componentes de la interfaz de usuario, por lo que un reproductor (*Player*) puede brindar acceso tanto a un componente visual como a un componente de panel de control, mediante los métodos *getVisualComponent* y *getControlPanelComponent*, respectivamente.

En el caso de que no se quiera los componentes de un control por defecto, se puede construir un control a la medida, empleando el mecanismo de los auditores de eventos para ejecutar las acciones correspondientes y/o actualizar la media. Por ejemplo, un reproductor puede tener componentes visuales que implanten *ControllerListeners*, para manipular la reproducción de la media, e incluso actualizar la interfaz gráfica de acuerdo a los cambios del reproductor.

Anexo F.2.2. Extensibilidad.

Los programadores avanzados pueden extender la funcionalidad JMF de la siguiente forma:

- Implantando componentes de procesamiento a la medida (*plug-ins*) que pueden ser intercambiados con los componentes de procesamiento estándares usados por un procesador JMF (clase *Processor*).

- Implantando las interfases de controladores, reproductores, procesadores, fuentes de datos o pilas de datos.

Implantar un *plug-in* de JMF permite extender o configurar a la medida las capacidades de un procesador sin tener que escribir el código desde cero. Una vez que el *plug-in* se registra con JMF, se puede seleccionar como una opción de procesamiento desde cualquier procesador que acepte los *plug-ins* de JMF; estos *plug-ins* pueden efectuar las siguientes tareas;

- Extender o reemplazar la capacidad de procesamiento de un procesador, al seleccionar cierto *plug-in*.
- Tener acceso a los datos de la media en cierto punto específico del flujo de datos, como en el pre-procesamiento o en el post-procesamiento de los datos en un procesador de media.
- Procesar datos de media fuera de un reproductor o de un procesador; un *plug-in* demultiplexor, por ejemplo, se puede usar para recuperar una pista de datos específica de un flujo de media multiplexado.

Si se requiere una mayor flexibilidad al usar componentes JMF, se puede crear interfases de controladores, reproductores, procesadores, fuentes de datos o repositorios de datos a la medida, los cuales se pueden usar articuladamente con componentes ya existentes. Por ejemplo, un decodificador de hardware se puede incorporar con un reproductor a la medida que efectúe la lectura de la media, la decodificación y el despliegue, todo en un sólo paso. Los reproductores y procesadores hechos a la medida pueden ser diseñados para integrarse con motores de media, tales como *Media Player* de Microsoft, o *RealPlayer* de RealNetworks.

Por último, hay que aclarar que no todos los reproductores y procesadores pueden utilizar *plug-ins*; los reproductores de la versión 1.0 de JMF, y algunos procesadores de la misma versión, no puede utilizarlos.

Anexo F.2.3. Presentación.

Dentro del API JMF los procesos de presentación son modelados por la interfaz del controlador (*Controller*), la cual define los estados básicos y los mecanismos de control de un objeto que controle, presente o capture media que tenga base en el tiempo. También define las fases por las que atraviesa un controlador, y brinda un mecanismo para controlar las transiciones entre dichas fases. Antes de presentar la media, ocurren varias operaciones, las cuales pueden consumir mucho tiempo, por lo la API JMF permite el control mediante la programación cuando ocurre ese caso.

Un controlador publica diversos eventos de tipo *MediaEvent*, relacionados con un controlador específico, para notificar cambios de estado. Para recibir eventos de un controlador, como un reproductor, se necesita implantar la interfaz *ControllerListener*. JMF define los siguientes tipos controladores:

- Reproductores (*Players*).
- Procesadores (*Processors*).

Ambos tipos se construyen a partir de una fuente de datos (*DataSource*) en particular, y que generalmente no se reutilizan para presentar otro tipo de datos de media.

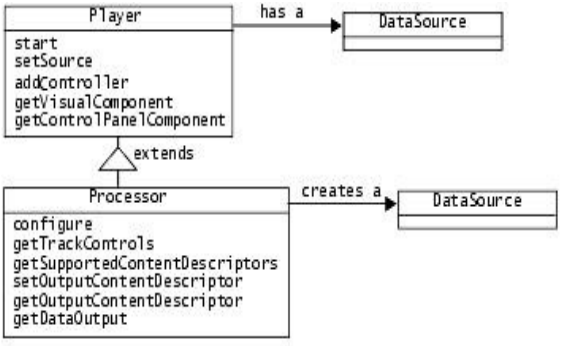


Figura 52: *Diagrama de clases de los controladores (reproductor y procesador).*

Anexo F.2.3.1. Reproductores.

Los reproductores (*players*) procesan un flujo de datos de media de entrada, y lo despliega en un momento específico de tiempo. Se usa un objeto de fuente de datos (*DataSource*) para entregar el flujo de media de entrada al reproductor. El reproductor no provee ningún control sobre el procesamiento que es llevado a cabo, ni de cómo se despliegan los datos de media. El destino de despliegue depende del tipo de media presentado.

Los reproductores pueden emplear controles estandarizados, y flexibilizan algunas de las restricciones operacionales impuestas por *Clock* y *Controller*.

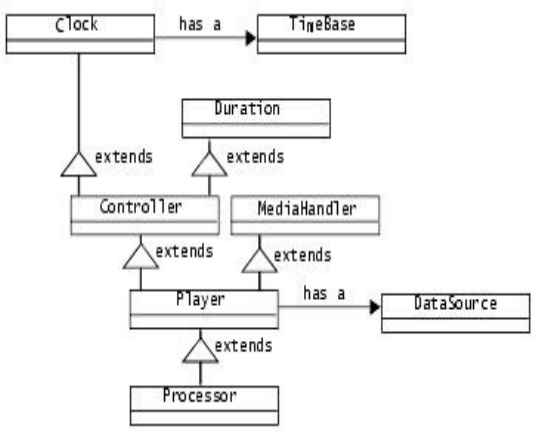


Figura 53: *Diagrama de clases del reproductor JMF.*

Anexo F.2.3.1.1. Estados del reproductor.

Un reproductor tiene dos estados primarios, detenido e iniciado; el primero se divide en cinco subestados, para facilitar la administración de recursos del mismo, los cuales son sin realizar (*unrealized*), realizándose (*realizing*), realizado (*realized*), pre-obteniéndose (*prefetching*) y pre-obtenido (*prefetched*).

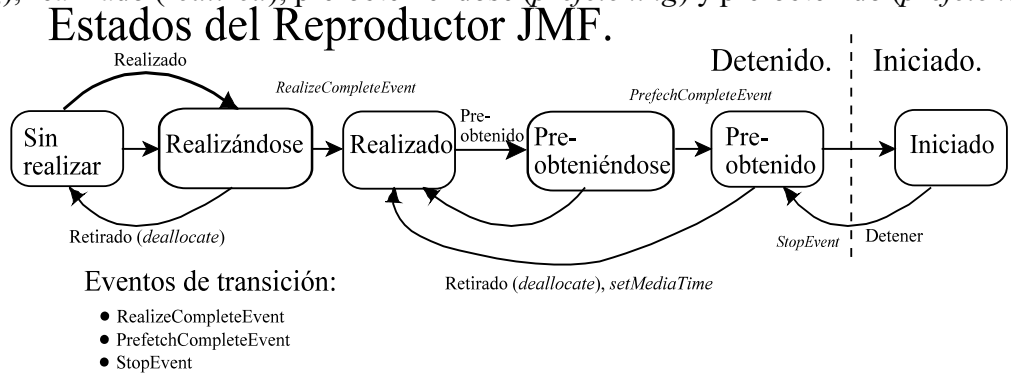


Figura 54: *Estados del Reproductor JMF.*

Cuando un reproductor trabaja de forma normal, recorre cada estado hasta llegar al estado de ***iniciado*** de la siguiente forma:

- Un reproductor en el estado sin realizar es aquel que se obtuvo como una instancia, pero todavía no conoce nada de la media; siempre que se crea un reproductor queda en este estado.
- Cuando se invoca el método *realize*, se cambia el estado de ***Sin Realizar*** a ***Realizando***, en el cual determina los recursos necesarios y adquiere aquellos que sólo va a necesitar una sola vez.

- Una vez que termina con las actividades del estado ***Realizando***, pasa al estado ***Realizado***, donde ya conoce los recursos necesarios y la información del tipo de media a presentar; provee componentes visuales y controles para presentar la media, ya que cuenta con información acerca de la misma. Aunque se relaciona con otros elementos del sistema, no tiene acceso exclusivo a ellos, por lo que otro reproductor puede iniciar.
- Al invocar el método pre-búsqueda, el estado cambia de ***Realizado*** a ***Pre-buscando***, donde se prepara a presentar la media al pre-cargar los datos de media, obtiene recursos de forma exclusiva, y ejecuta cualquier otra tarea necesaria para poder presentar la media. Si la presentación de la media se vuelve a re-posicionar, o si se cambia la tasa de presentación y se requiere de algún proceso alterno o de mayor memoria intermedia, se regresa al estado pre-obteniendo.
- Una vez que termina con los procesos de pre-obteniendo, pasa a pre-obtenido, donde el reproductor está listo para presentar la media.
- Al llamar al método iniciar (*start*), el reproductor pasa al estado iniciado, donde el tiempo del objeto tiempo base y el de tiempo media se mapean, y el reloj de la media está corriendo.

Un reproductor envía eventos de tipo *TransitionEvents* cuando cambia de estado, por lo que se podría implantar la interfaz *ControllerListener* para tener conocimiento del estado en que se encuentra, con el fin de administrar la latencia en los estados ***Realizando*** y ***Pre-obteniendo***, y para saber cuándo llamar a los métodos del reproductor.

Anexo F.2.3.1.2. Métodos disponibles para cada estado del reproductor.

Para evitar conflictos entre procesos para asignar valores a ciertos recursos, no todos los métodos de un reproductor están disponibles en cualquier estado, y en caso de que se quiera usar un método no permitido en un estado, se arroja una excepción. A continuación se muestra una tabla donde indica si la llamada a un método está permitida, o el tipo que excepción que lanza, por cada estado del reproductor.

Método	**Estado del reproductor.**			
	Sin realizar.	**Realizado.**	**Pre-búsqueda.**	**Iniciado.**
addController	NotRealizedError	Permitido	Permitido	ClockStartedError
deallocate	Permitido	Permitido	Permitido	ClockStartedError
getControlPanelComponent	NotRealizedError	Permitido	Permitido	Permitido
getGainControl	NotRealizedError	Permitido	Permitido	Permitido
getStartLatency	NotRealizedError	Permitido	Permitido	Permitido
getTimeBase	NotRealizedError	Permitido	Permitido	Permitido
getVisualComponet	NotRealizedError	Permitido	Permitido	Permitido
mapToTimeBase	ClockStoppedException	ClockStoppedException	ClockStoppedException	Permitido
removeController	NotRealizedError	Permitido	Permitido	ClockStartedError
setMediaTime	NotRealizedError	Permitido	Permitido	Permitido
setRate	NotRealizedError	Permitido	Permitido	Permitido

setStopTime	NotRealizedError	Permitido	Permitido	StopTimeSetError (si fue previamente asignado)
setTimeBase	NotRealizedError	Permitido	Permitido	ClockStartedError
syncStart	NotPrefetchedError	NotPrefetchedError	Permitido	ClockStartedError

Tabla 26: Métodos permitidos por estado del reproductor.

Anexo F.2.3.2. Procesadores.

Los procesadores son un tipo especial de reproductores, los cuales permiten control sobre el proceso del flujo de media entrante, y manejan los mismos controladores de presentación que posee un reproductor. Un procesador también puede usarse para presentar la media, o para le dé salida a la media mediante un objeto DataSource, para que se reciba en otro procesador, en un reproductor, o sea entregado a otro destino (como un archivo).

Modelo del procesador JMF.

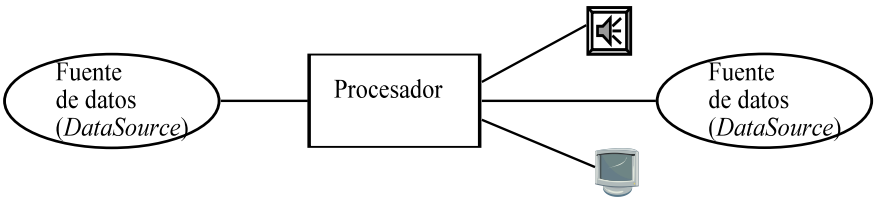


Figura 55: *Modelo del procesador JMF.*

Anexo F.2.3.2.1. Controles de presentación.

Son mecanismos adicionales de los reproductores y procesadores, a los controles ya definidos por Controller, para ajustar el volumen de la reproducción. Estos controles específicos se obtienen al llamar al método *getControls*.

*Anexo F.2.3.2.*2. Componentes de la interfaz estándar del usuario.

Los reproductores y los procesadores generalmente tienen dos componentes estándares de la interfaz de usuario, los cuales son un componente visual y otro de tipo de panel de control, los cuales son obtenidos al llamar a los métodos *getVisualComponent* y *getControlPanelComponent*. También se puede implantar componentes de la interfaz de usuario hechos a la medida, y agregar auditores de eventos para determinar cuándo necesitan ser actualizados.

Anexo F.2.3.2.3. Eventos del controlador (Controller).

Estos eventos (*ControllerEvents*) del controlador, ya sean reproductores o procesadores, se dividen en estas categorías:

- Eventos de notificación de cambios, tales como *RateChangeEvent*, *DurationUpdateEvent* o *FormatChangeEvent*; se envían para avisar de un cambio en un atributo del controlador, el cual, por lo general, es causado por la invocación a un método.
- Eventos de transición (*TransitionEvents*), que indican cambios de un estado del controlador a otro.

- Eventos de cierre (*ControllerClosedEvents*), los cuales son enviados cuando un controlador se apaga (cerrar); una vez enviado un evento de este tipo, ya no se puede utilizar el controlador. Un evento de tipo *ControllerErrorEvent* es un caso especial de *ControllerClosedEvent*, el cual se utiliza para manejar el caso en que el controlador no funcione adecuadamente.

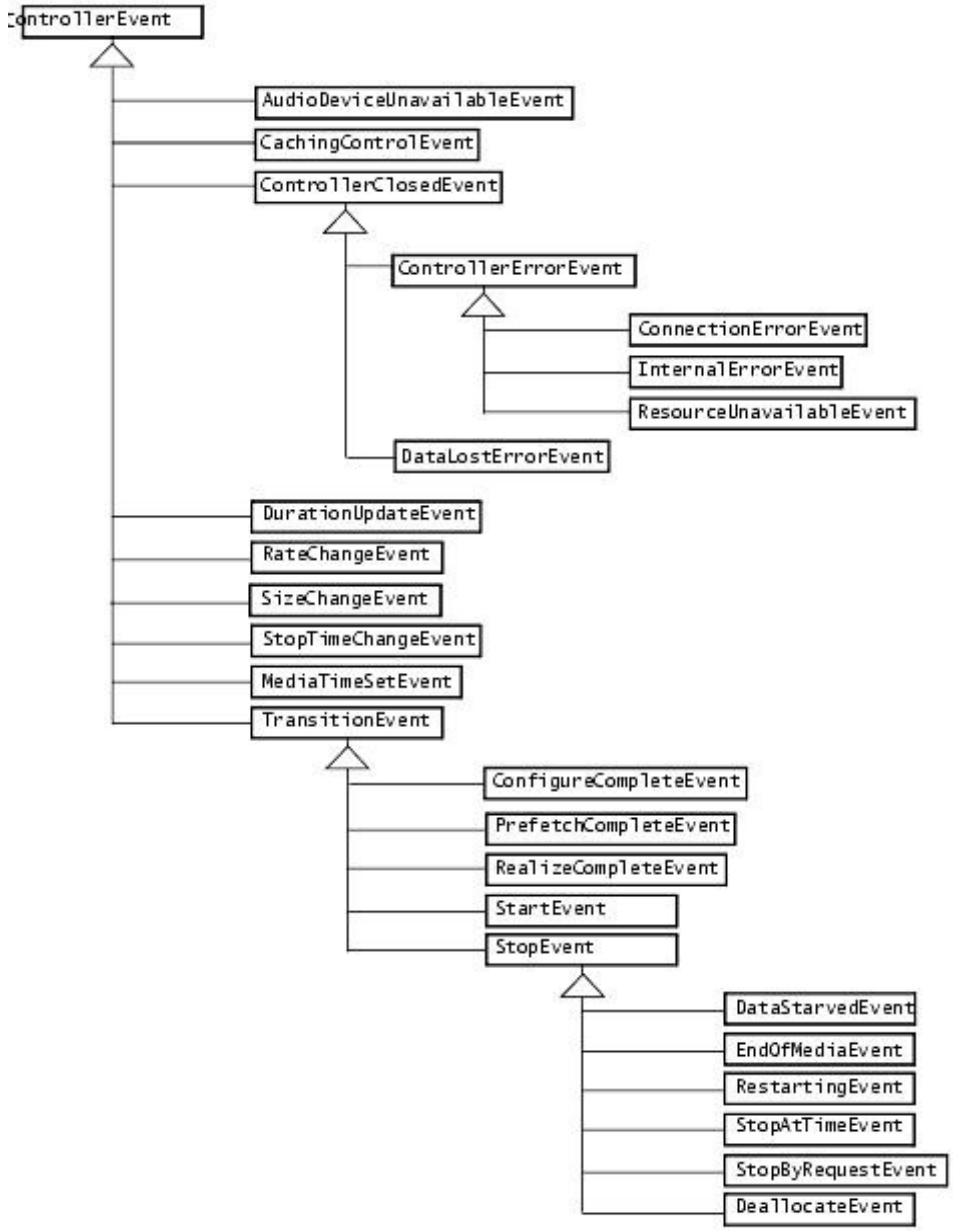


Figura 56: *Diagrama de clases de los eventos del controlador JMF.*

Anexo F.2.3.3. Procesamiento.

Como ya se había explicado, un procesador es un tipo especial de reproductor, el cual puede procesar flujos de media, con el fin de aplicar efectos, mezclar o componer en tiempo real. A continuación se muestra un diagrama que ilustra cada etapa, explicadas después del gráfico:

Etapas del procesador.

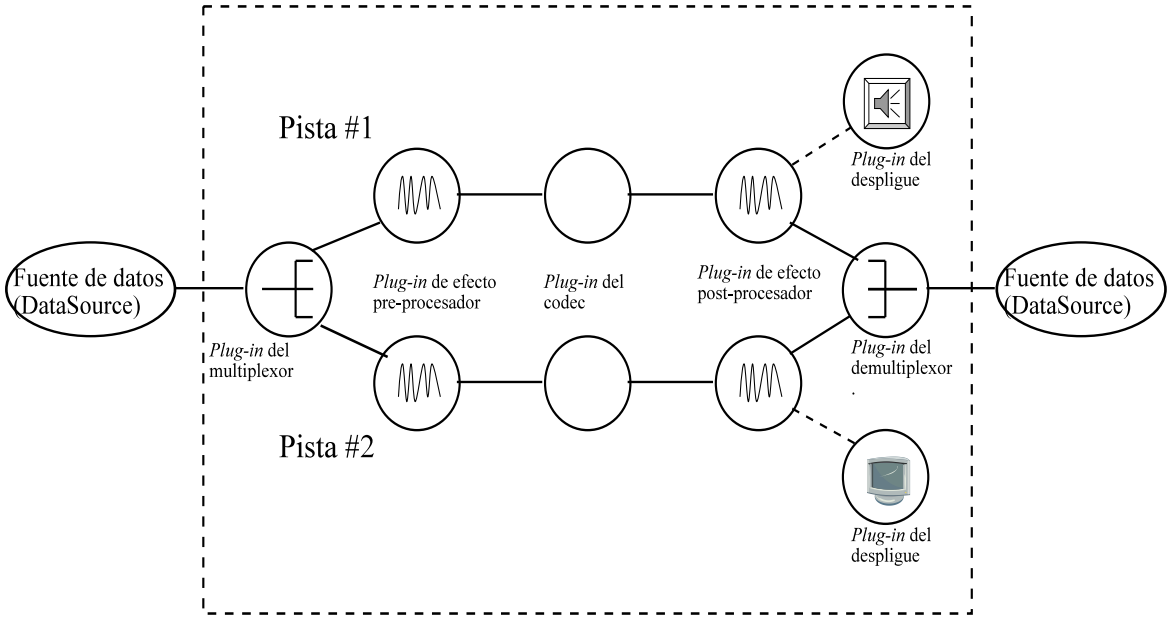


Figura 57: *Etapas del procesador JMF.*

- Demultiplexar es el proceso de revisar el flujo de entrada para separar las pistas individuales.
- Pre-procesamiento es la acción de aplicar algoritmos con el fin de lograr un efecto sobre las pistas extraídas anteriormente.
- Transcodificar es el proceso de convertir el formato de entrada de cada pista de media a otro formato; el caso especial en el cual se convierte un formato comprimido a otro sin comprimir se le llama decodificar, y el proceso inverso es la codificación.
- Post-procesamiento es cuando se aplican algoritmos de efecto a pistas decodificadas.
- Multiplexar es intercalar pistas de media transcodificadas en un sólo flujo de salida, como en el caso de mezclar una pista de audio y otra de video para obtener un flujo de salida AVI o MPEG-1; el método *setOutputContentDescriptor* del procesador se usa para especificar el tipo de datos del flujo de salida.
- Desplegar (*rendering*) es la acción de presentar la media al usuario.

El procesamiento en cada etapa es llevado a cabo por componentes de procesamiento independientes, llamados *plug-ins*. Si un procesador le da soporte a *TrackControls*, se puede indicar qué *plug-ins* se quieren utilizar para procesar una pista en particular. Estos componentes pueden ser de los siguientes cinco tipos:

- Demultiplexor, el cual lee y revisa la sintaxis de un flujo de media (*parse*), y separa el flujo en pistas particulares.
- De efecto, el cual realiza un procesamiento de efecto especial a una pista de media.
- Códec, el cual efectúa la codificación y decodificación de datos.
- Multiplexor, el cual combina varias pistas de datos de entrada en un solo flujo de salida intercalado, el cual es entregado como una fuente de salida de datos (*DataSource*).
- De despliegue, el cual procesa la media de una pista y lo entrega a su destino (como una pantalla o bocinas).

Anexo F.2.3.3.1. Estados del procesador.

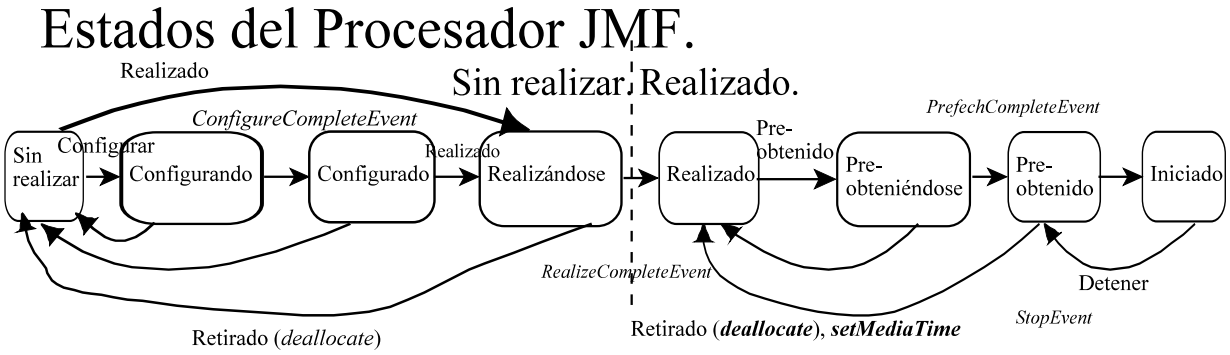
Un procesador tiene dos estados de espera adicionales, **configurando** y **configurado**, los cuales están antes del estado de realizando, y efectúan las siguientes tareas:

- El procesador entra a ***Configurando*** cuando se llama al método *configure*; es aquí donde se conecta a la fuente de datos, demultiplexa el flujo de entrada, y obtiene información acerca del formato de los datos de entrada.

- Una vez que concluye las tareas anteriores, el procesador pasa al estado ***Configurado***, enviando un evento de tipo *ConfigureCompleteEvent*; es en este estado donde puede llamar al método *getTrackControls*, para obtener los controles de las pistas individuales, con el fin de indicar cuáles operaciones de procesamiento efectuará el procesador.

- Cuando se llama al método *Realize*, el procesador cambia de estado a ***Realizado***, en el cual se encuentra completamente construido. Si se intenta configurar al procesador en este estado, generalmente fallará, ya que sólo unas pocas implantaciones brindan soporte a esta funcionalidad.

Si se llama al método *realize* en un procesador que esté en ***sin realizar***, provoca que pase automáticamente a los estados ***Configurando***, ***Configurado*** y ***Realizado***, sin tener la opción de configurar dicho procesador, ya que se usan las opciones por defecto.



Eventos de transición:

- PrefetchCompleteEvent
- StopEvent
- RealizeCompleteEvent
- ConfigureCompleteEvent

Figura 58: *Estados del Procesador JMF.*

Anexo F.2.3.3.2. Métodos disponibles para cada estado del procesador.

De forma similar a lo que ya se explicó para los reproductores, y recordando que los procesadores son tipos especiales de reproductores, existen restricciones para llamar a ciertos métodos en determinados estados; en caso de hacerlo, se enviará un error o una excepción. A continuación se muestra una tabla que muestra si es permisible llamar a un método en cierto estado, o la excepción o error arrojados:

Método.	Procesador sin realizar.	Procesador configurando.	Procesador configurado.	Procesador realizado.
addController	NotRealizedError	NotRealizedError	NotRealizedError	Permitido
deallocate	Permitido	Permitido	Permitido	Permitido
getControlPanelComponent	NotRealizedError	NotRealizedError	NotRealizedError	Permitido
getControls	Permitido	Permitido	Permitido	Permitido
getDataOutput	NotRealizedError	NotRealizedError	NotRealizedError	Permitido
getGainControl	NotRealizedError	NotRealizedError	NotRealizedError	Permitido
getOutput-ContentDescri-ptor	NotConfiguredError	NotConfiguredError	Permitido	Permitido
getStartLatency	NotRealizedError	NotRealizedError	NotRealizedError	Permitido
getSupported-Content-Descriptors	Permitido	Permitido	Permitido	Permitido
getTimeBase	NotRealizedError	NotRealizedError	NotRealizedError	Permitido
getTrackControls	NotConfiguredError	NotConfiguredError	Permitido	FormatChange-Exception
getVisualComponet	NotRealizedError	NotRealizedError	NotRealizedError	Permitido
mapToTimeBase	ClockStoppedException	ClockStoppedException	ClockStoppedException	ClockStoppedException
realize	Permitido	Permitido	Permitido	Permitido
removeController	NotRealizedError	NotRealizedError	NotRealizedError	Permitido
setOutputContentDescriptor	NotConfiguredError	NotConfiguredError	Permitido	FormatChangeException
setMediaTime	NotRealizedError	NotRealizedError	NotRealizedError	Permitido
setRate	NotRealizedError	NotRealizedError	NotRealizedError	Permitido
setStropTime	NotRealizedError	NotRealizedError	NotRealizedError	Permitido
setTimeBase	NotRealizedError	NotRealizedError	NotRealizedError	Permitido
syncStart	NotPrefetchedError	NotPrefetchedError	NotPrefetchedError	NotPrefetchedError

Tabla 27: Métodos permitidos por estado del procesador JMF.

Anexo F.2.3.3.3. Controles del procesamiento del procesador.

Como ya se había mencionado, el objeto *TrackControl* permite indicar qué operaciones de procesamiento se pueden aplicar a una pista de datos; este objeto se obtiene por medio de *getTrackControls*, el cual obtiene todas las instancias de ese tipo para todas las pistas del flujo de media.

Con *TrackControl* se eligen explícitamente los efectos, códecs, y "*plug-ins*" de despliegue a aplicar a la pista; también se puede usar *PlugInManager* para averiguar qué "*plug-ins*" están instalados.

El método *getControls* de *TrackControl* brinda los controles asociados con una pista, para poder averiguar qué proceso de transcodificación se lleva a cabo en dicha pista por un códec en específico; algunos de los controles regresados son *BitRateControl* y *QualityControl*.

Para especificar el formato de los datos de salida se usa el método *setFormat*; el procesador escoge el códec y el dispositivo de despliegue adecuados para el formato indicado. También se puede indicar el formato de salida al construir el procesador con un modelo *ProcessorModel*, el cual define los requerimientos de entrada y salida para su procesador; al proporcionar esta instancia al método de creación apropiado del administrador (*manager*), éste se esfuerza en crear un procesador con los requerimientos indicados.

Anexo F.2.3.3.4. Datos de salida.

Los datos de salida de un procesador se recuperan como un *DataSource* invocando al método *getDataOutput*; ese objeto puede servir como entrada para otros objetos, tales como reproductores, procesadores o repositorios de datos (*data sinks*). La instancia regresada puede ser de cualquier tipo (*PushDataSource*, *PushBufferDataSource*, *PullDataSource* o *PullBufferDataSource*).

Algunos procesadores no proporcionan datos de salida porque despliegan directamente los datos procesados, por lo que más bien son reproductores configurables.

Anexo F.2.3.3.5. Captura de media.

Un dispositivo de captura de multimedia, como un micrófono o una videocámara, puede funcionar como una fuente de entrega de multimedia. JMF abstrae tales dispositivos como DataSources, que pueden ser de cualquier tipo. Algunos dispositivos entregan flujos de datos múltiples, con una fuente de datos (DataSource) que contiene múltiples fuentes de flujos (SourceStreams) que mapean a los flujos de datos que provee el dispositivo.

Anexo F.2.3.3.6. Transmisión y almacenamiento de datos de media.

Un repositorio de datos *(DataSink*) se usa para leer datos de media de una fuente de datos, y desplegarlos en algún destino, el cual, por lo general, no es un dispositivo de presentación; en otras palabras, almacena datos de forma temporal antes de enviarlos a otro lado. Estos objetos pueden escribir datos en un archivo, escribirlos a través de la red, o funcionar como transmisores RTP.

Los repositorios de datos se construyen por medio de un administrador al cual se le proporciona una fuente de datos. Para manipular el proceso de escritura de datos a archivos se puede usar *StreamWriterControl*.

Anexo F.2.3.3.7. Controles de almacenamiento.

Un repositorio de datos envía eventos de tipo *DataSinkEvent* para reportar sus estados e indicar algún suceso al guardar los datos; estos eventos se pueden enviar con algún código, o enviarse alguno de los siguientes subtipos.

- *DataSinkErrorEvent*, el cual indica problemas que ocurrieron al escribir datos.
- *EndOfStreamEvent*, el cual reporta que el flujo completo fue escrito exitosamente.

Para escuchar y responder a los eventos antes descritos, se implanta la interfaz *DataSinkListener*.

Anexo F.3. Extensibilidad de JMF.

Para aumentar las capacidades de este API, se pueden construir diversos elementos a la medida, tales como "*plug-ins*", controladores de media, o fuentes de datos.

Anexo F.3.1. Implantación de "plug-ins".

Al implantar alguna interfaz de JMF se obtienen las capacidades para tener acceso directo y manipular los datos de media de un procesador. A continuación se mencionan las nuevas capacidades por cada interfaz implantada:

- Implantando la interfaz *Demultiplexer* permite controlar la forma en que las pistas individuales son extraídas de un flujo de media multiplexado.
- Implantando *codec*, permite llevar a cabo el procesamiento para decodificar datos de media, convertir formatos de media, y codificar (comprimir) datos de media "crudos".
- Implantando *Effect* permite efectuar procesamiento a la medida en los datos de media.
- Implantando *Multiplexer* permite cómo se mezclan las pistas individuales para crear un nuevo flujo intercalado de salida, el cual se le proporciona a un procesador.
- Implantando *Renderer*, se obtiene el control de cómo se procesan y despliegan los datos.

Antes de implantar un "*plug-in*", hay que recordar que sólo los procesadores y reproductores del modelo JMF 2.0 les dan soporte pleno; los reproductores del modelo JMF 1.0 no les dan soporte, y sólo algunos procesadores de este modelo pueden usarlos.

Los procesadores pueden disponer de los "*plug-ins*" de códecs, efectos y dispositivos de despliegue hechos a la medida, por medio de la interfaz *TrackControl*. En el caso del procesador por defecto, o aquellos hechos con base con un modelo (*ProcessorModel*), pueden disponer de un *plug-in*, si se registra previamente con el *PlugInManager*; posteriormente se obtiene el *plug-in* por medio del método *getPlugInList*, y el administrador (*Manager*) puede obtenerlos cuando se construye alguna instancia de procesador.

Anexo F.3.2. Implantando controladores de media y fuentes de datos.

La API JMF también permite crear controladores de media hechos a la medida, tales como reproductores, procesadores, fuentes de datos y repositorios de datos, los cuales se integran de forma articulada al API. Estas clases hechas a la medida pueden se registran con un prefijo de paquete único con *PackageManager*, el cual coloca al nuevo paquete en la jerarquía de paquetes predefinida, para que posteriormente el administrador (*Manager*) encuentre a la clase y construya el objeto solicitado.

Anexo F.3.2.1 Construcción del controlador de media.

Los controladores de media, reproductores, procesadores y repositorios de datos, siempre se construyen a partir de una fuente de datos (*DataSource*), ya que todos ellos leen datos de ella. Cuando se invoca al método *createMediaHandler* para crear este controlador, el administrador recupera el nombre del tipo de contenido (*content-*

type) de la fuente de datos con el fin de crear un controlador de media(*MediaHandler*) apropiado.

JMF también le da soporte a otro tipo de controlador de media, *MediaProxy*, el cual procesa el contenido de una fuente de datos para crear otra; generalmente lee un archivo de configuración que contiene los datos para conectarse a un servidor y obtener datos de media del mismo. Para crear un reproductor a partir de este objeto, se efectúan los siguientes pasos:

1. Construir una fuente de datos para el protocolo indicado por el objeto *MediaLocator*.

2. Usar el tipo de contenido de la fuente de datos para crear un *MediaProxy*, el cual leerá el archivo de configuración.

3. Obtener una nueva fuente de datos a partir del objeto *MediaProxy*.

4. Usar el tipo de contenido (*content-type*) de la nueva fuente de datos para construir el reproductor.

El mecanismo que el administrador emplea para localizar y construir un controlador de media para una fuente de datos en particular consiste en los siguientes pasos:

- El administrador genera una lista de las clases de tipo *MediaHandler*, a partir de una lista de prefijos de paquete de contenidos recuperados con *PackageManager*.
- Cada clase es revisada por el administrador, hasta que se encuentra alguna que se llame *Handler*, la cual pueda ser construida y a la cual se le pueda agregar una fuente de datos.

Al construir reproductores y procesadores, el administrador genera una lista de clases de controladores disponibles a partir de la lista del contexto de prefijo de paquetes extraída y del nombre de tipo de contenido de la fuente de datos. Para encontrar reproductores el administrador busca clases que tengan la siguiente forma:

<prefijo de paquete de contenido>.media.contenido.<tipo de contenido>.Controlador

Para los procesadores, el administrador busca las clases con la forma:

<prefijo de paquete del contenido>.media.procesador.<tipo de contenido>.Controlador

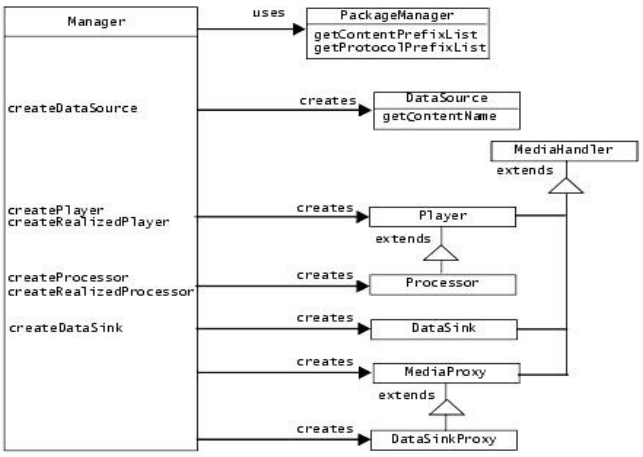


Figura 59: *Diagrama de clases del administrador de JMF y de los controladores de media.*

Si la clase localizada es de tipo *MediaProxy*, el administrador extrae de ella una nueva fuente de datos y repite la búsqueda. Si no se encuentra ningún controlador adecuado, se cambia el nombre del tipo de contenido por *unknow* (desconocido) y se repite la búsqueda, con el fin de obtener reproductores genéricos, los cuales manejan muchos tipos de media.

Al momento de crear un repositorio de datos, hay que tomar en cuenta tanto como su origen como su destino, ya que estos objetos despliegan la media de forma inmediata. El administrador recupera una lista de prefijos de paquete y el protocolo del localizador de media que identifica su destino. El administrador busca clases con el siguiente nombre:

<prefijo de paquete del contenido>.media.repositorioDatos.protocolo.Controlador

Si el controlador de media encontrado es de tipo repositorio de datos (*DataSink*), el administrador crea una instancia de él, asigna su fuente de datos y su localizador de media (*MediaLocator*), regresando este nuevo objeto. Si el controlador es de tipo *DataSinkProxy*, el administrador recupera el nombre del tipo de media y genera una lista con las clases de tipo *DataSink* que den soporte al protocolo del destino de *MediaLocator*, y el tipo de contenido que se recupera con el *proxy*; el nombre de las clases tiene el formato:

<prefijo paquete contenido>.media.repositorioDatos.protocolo.<tipo contenido>.Controlador

Ese proceso continúa hasta que se encuentra una clase *DataSink* adecuada, o hasta terminar de revisar todo el contenido de los prefijos de paquete.

Anexo F.3.2.2. Construcción de fuentes de datos.

De forma similar, el administrador construye fuentes de datos (*DataSources*); la diferencia consiste en que la lista con nombres de clases que genera es a partir de los nombres de los prefijos de paquetes de protocolos instalados. Los nombres de clases que agregan tienen la siguiente forma:

<prefijo de paquete del protocolo>media.protocolo.<protocolo>. FuenteDatos

El administrador itera por cada clase de la lista hasta que encuentra una fuente de datos (*DataSource*) de la que pueda crear una instancia y agregarle un localizador de media (*MediaLocator*).

Anexo F.4. El protocolo RTP.

El Protocolo de Transporte en Tiempo Real (*Real-Time Transport Protocol, RTP*) sirve para transmitir flujos de media en tiempo real; las principales aplicaciones que tiene son las videoconferencias y las transmisiones de media (emisiones de radio y de televisión por Internet), en las cuales no se requiera una calidad excepcional, y acepten cierta pérdida de la fidelidad de la media con el fin de transmitir la media, a la vez de que no consume tanto ancho de banda.

Varias aplicaciones que usan Internet, disponen del protocolo TCP, el cual es un protocolo de capa de transporte diseñado para la comunicación de datos en redes con un ancho de banda bajo y con altas tasas de error; cuando se detecta que un paquete de datos se perdió o está corrupto, se retransmite. Este mecanismo de fiabilidad

de datos consume tiempo, por lo que la transmisión de media llevaría mucho medio; por lo anterior se escogió al protocolo UDP, el cual no es un protocolo confiable, el cual no garantiza que lleguen los paquetes de información, ni que lleguen duplicados, corruptos o en desorden. RTP acepta y administra los defectos anteriores, compensándolos con el fin de presentar la mayoría de la media.

Este protocolo se define en el documento técnico IEFT RFC 1889. Ya no se describirá más este protocolo, ya que está fuera del alcance de esta tesis, y sólo se mostró a grandes rasgos, con el fin de mostrar sus capacidades y principales aplicaciones.

Anexo F.5. Utilerías de Java Media Framework.

Este API contiene algunas herramientas que se mencionan, pero no describen profundidad.

Anexo F.5.1. *JMFRegistry*.

Es una aplicación de Java que permite registrar nuevas fuentes de datos, controladores de media, plug-ins y dispositivos de captura.

Anexo F.5.2. *MediaPlayer Bean*.

Un JavaBean es un componente de software que se puede reutilizar; este es un componente para media que contiene la funcionalidad de JMF, pero que se puede utilizan en gran variedad de aplicaciones.

Anexo F.5.3. *JMFStudio*.

Es una aplicación Java la cual es posible encontrar dispositivos de captura de media, activarlos, capturar, guardar, presentar la media, etcétera.

Anexo F.6. Otras capacidades de Java Media Framework.

Además de que permite utilizar componentes por defecto, o crear eso componentes a la medida, este API permite extraer información de dichos componentes, para tener control programático sobre los mismos, integrarlos con otras aplicaciones mostrando sus componentes, o convertir entre formatos.

Anexo G. Código Fuente de la Aplicación.

Los códigos fuente se encuentran en el *CD-ROM* anexo, organizados de la siguiente forma:

- Códigos fuente del lenguaje java, junto con los archivos de creación de la aplicación web.
- Archivos de *NeatBeans* de los elementos *UML* del sistema y de los diagramas *UML*.
- Guiones para la creación de la base de datos ***control***, de sus tablas, y de carga de datos.
- Diagramas en formato *WordPerfect Graphics* de *Corel Presentations*.
- Archivos binarios de instalación del *JDK*, *JRE*, de *Apache-Tomcat*, de *MySQL*, y de las interfases usadas por la aplicación.