# 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.

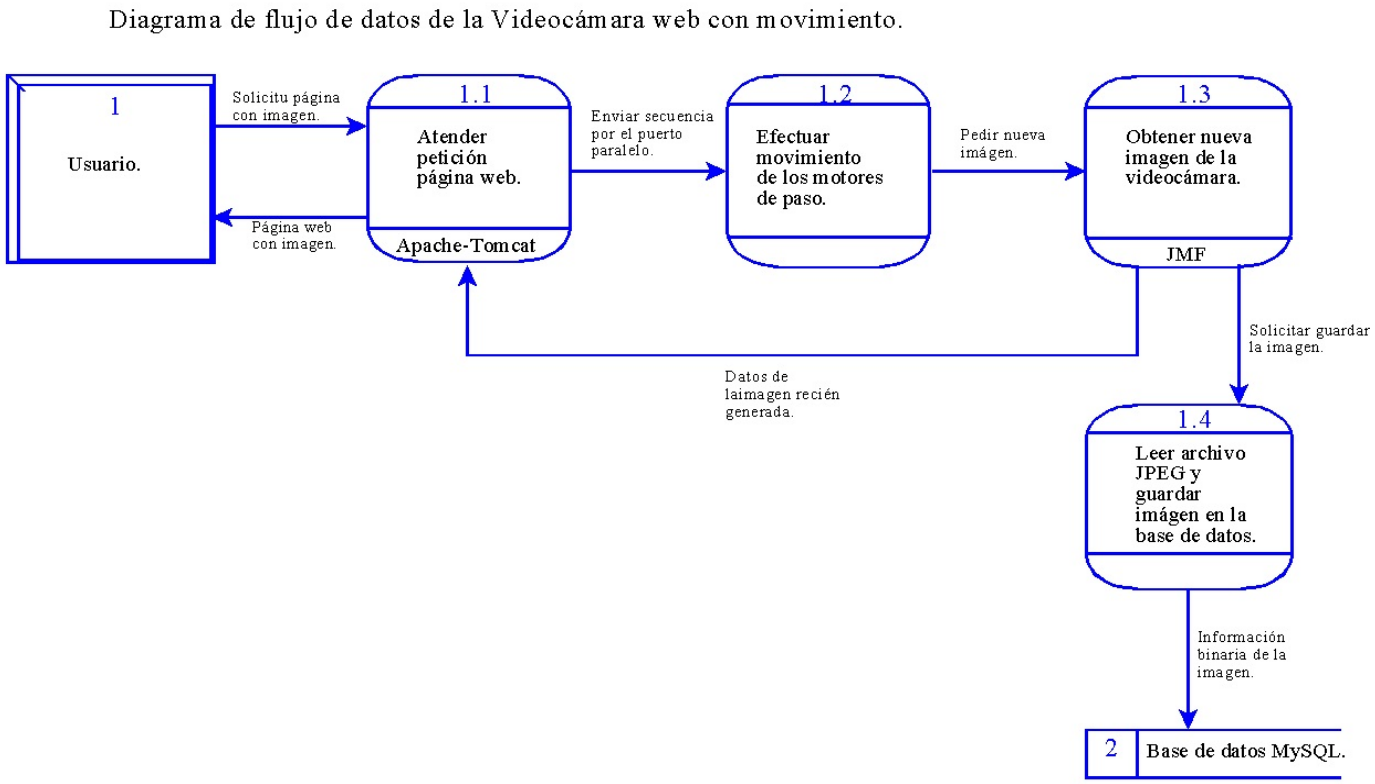



**Figura 27**: *Diagrama de Flujo de Datos del control de posición y obtención de imágenes de la videocámara.*

# Anexo B. Diagrama Entidad-Relación.

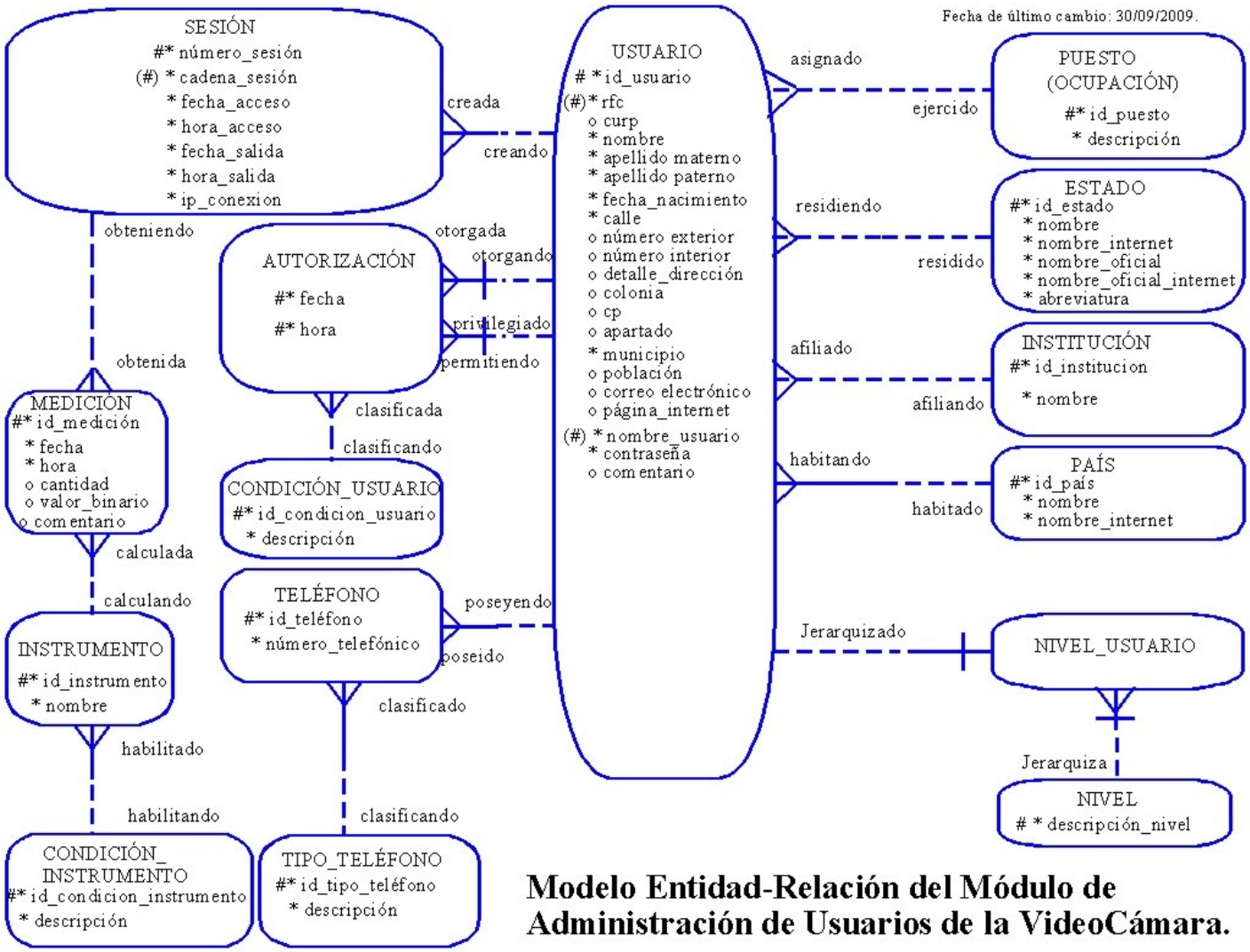



**Figura 28**: *Modelo Entidad Relación de la base de datos del sistema de la videocámara.*

# Anexo C. Diagramas *UML*.

## Anexo C.1 Casos de Uso.

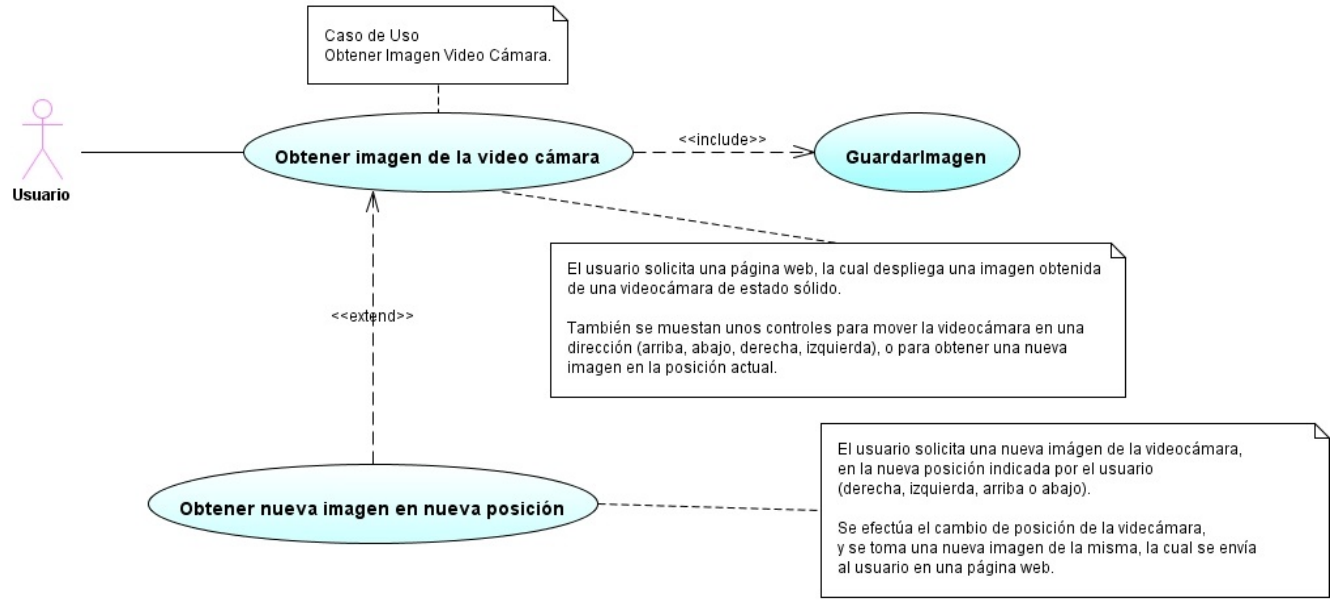



**Figura 29**: *Caso de Uso "Obtener Imagen Video Cámara".*

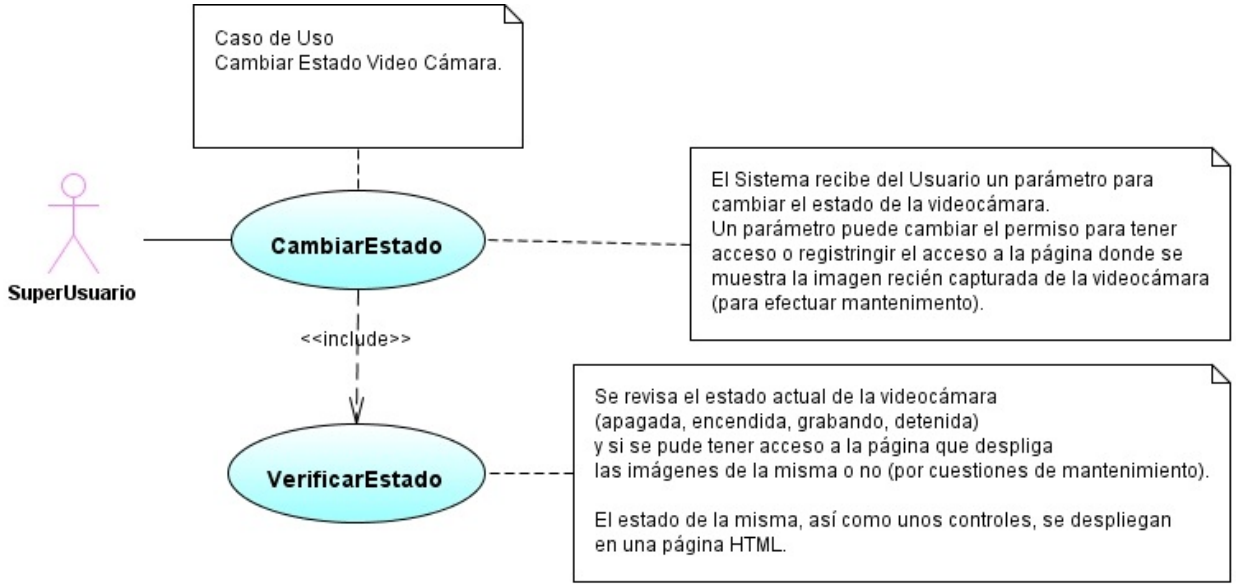



**Figura 30**: *Caso de Uso del cambio de estado de la videocámara.*

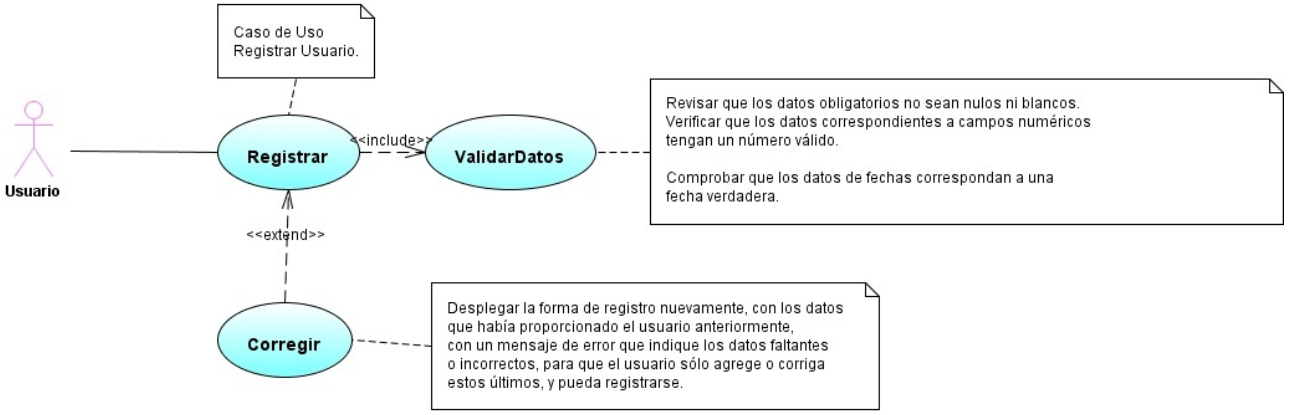



**Figura 31**: *Caso de Uso "Registrar Usuario".*

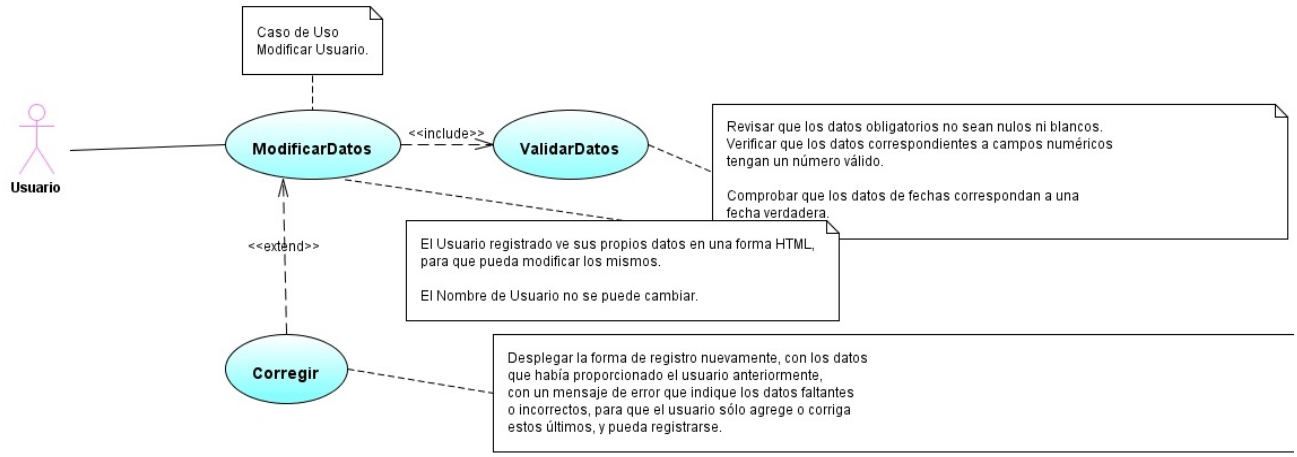



**Figura 32**: *Caso de Uso "Modificar Usuario".*

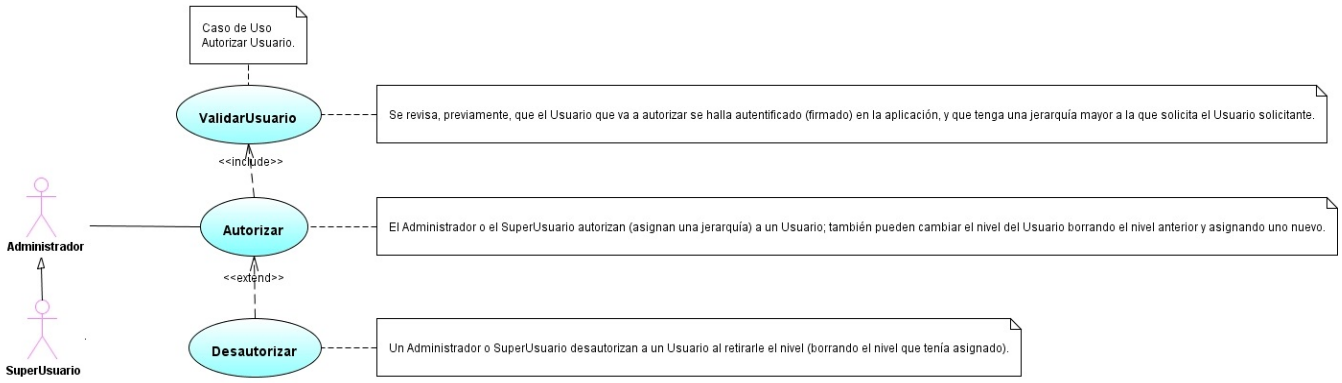



**Figura 33**: *Caso de Uso "Autorizar Usuario".*


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

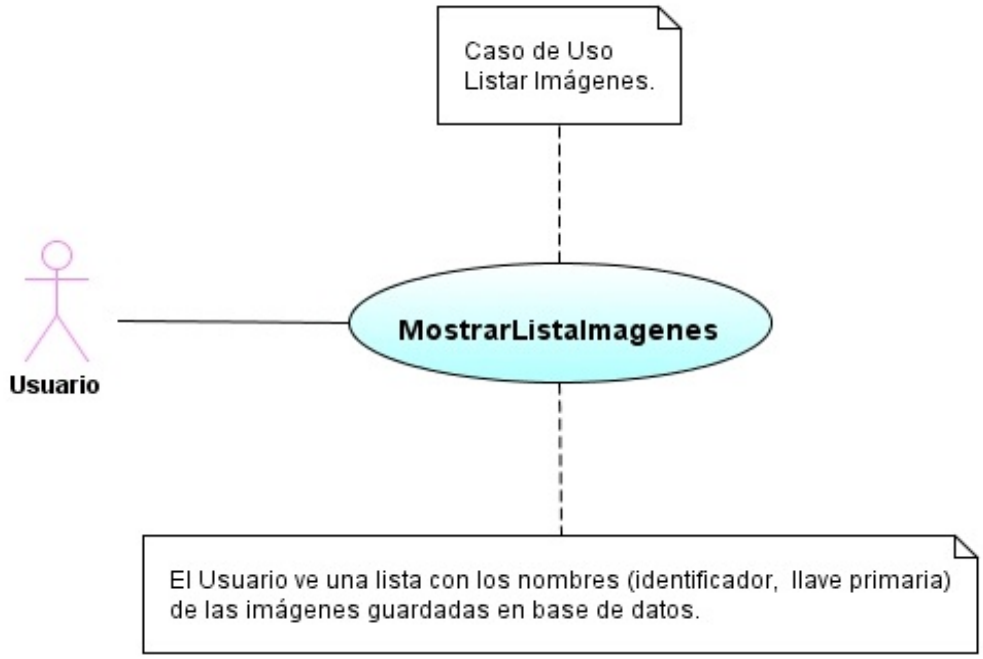



**Figura 35**: *Caso de Uso "Listar Imágenes".*

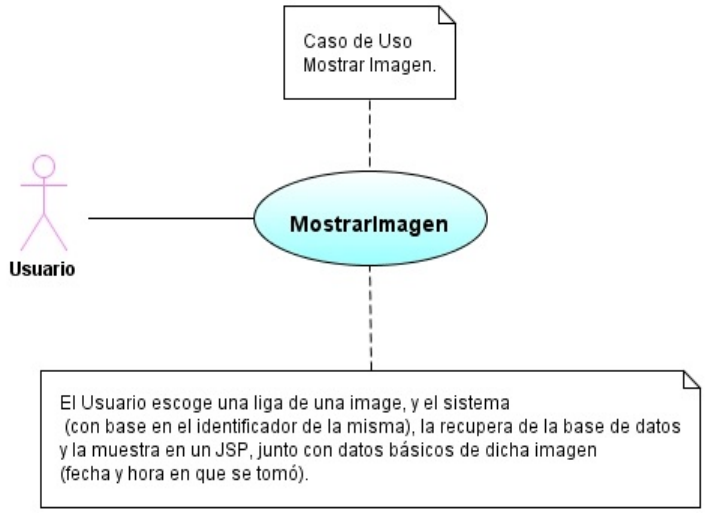



**Figura 36**: *Caso de Uso "Mostrar Imagen".*

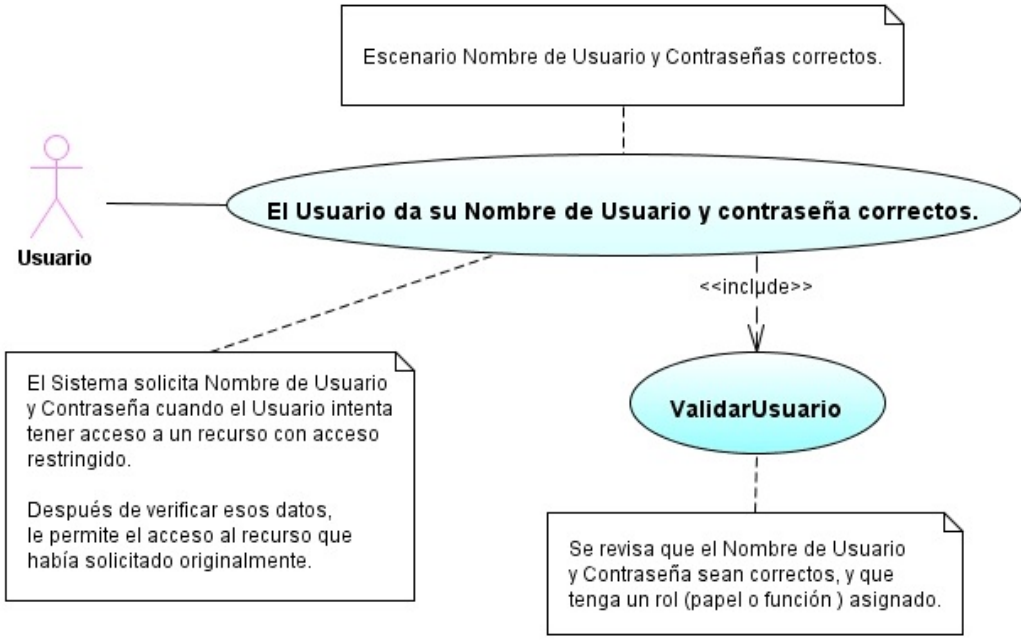



**Figura 37**: *Escenario "Nombre de Usuario y Contraseñas correctos".*

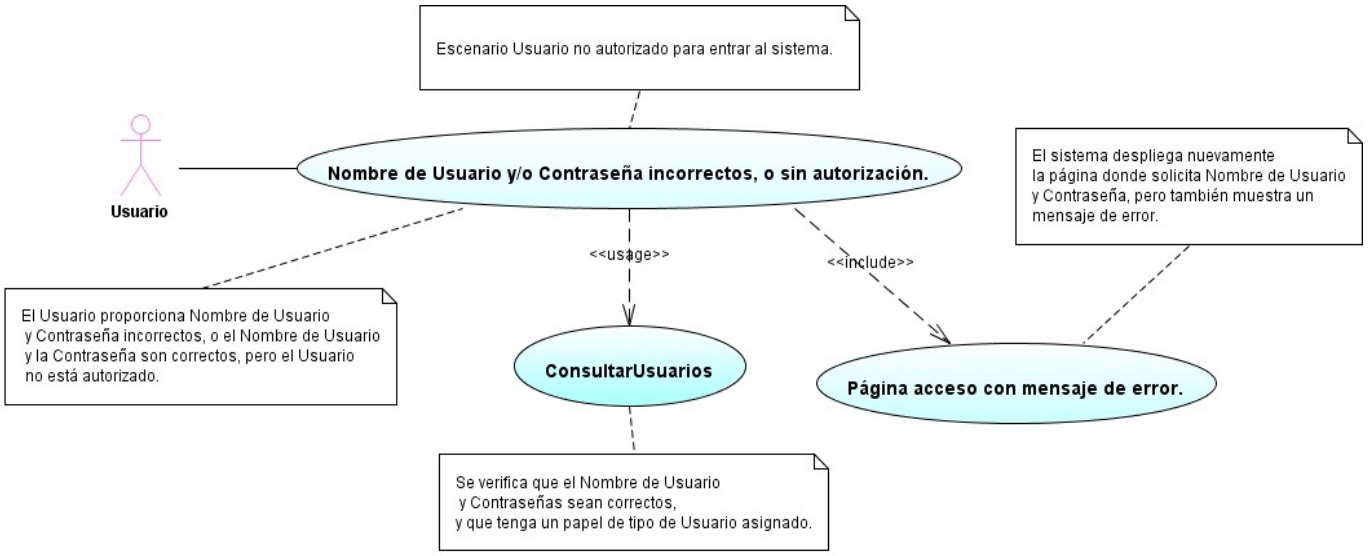



 **Figura 38**: *Escenario "Usuario no Autorizado".*

## Anexo C.2. Diagrama de Distribución.

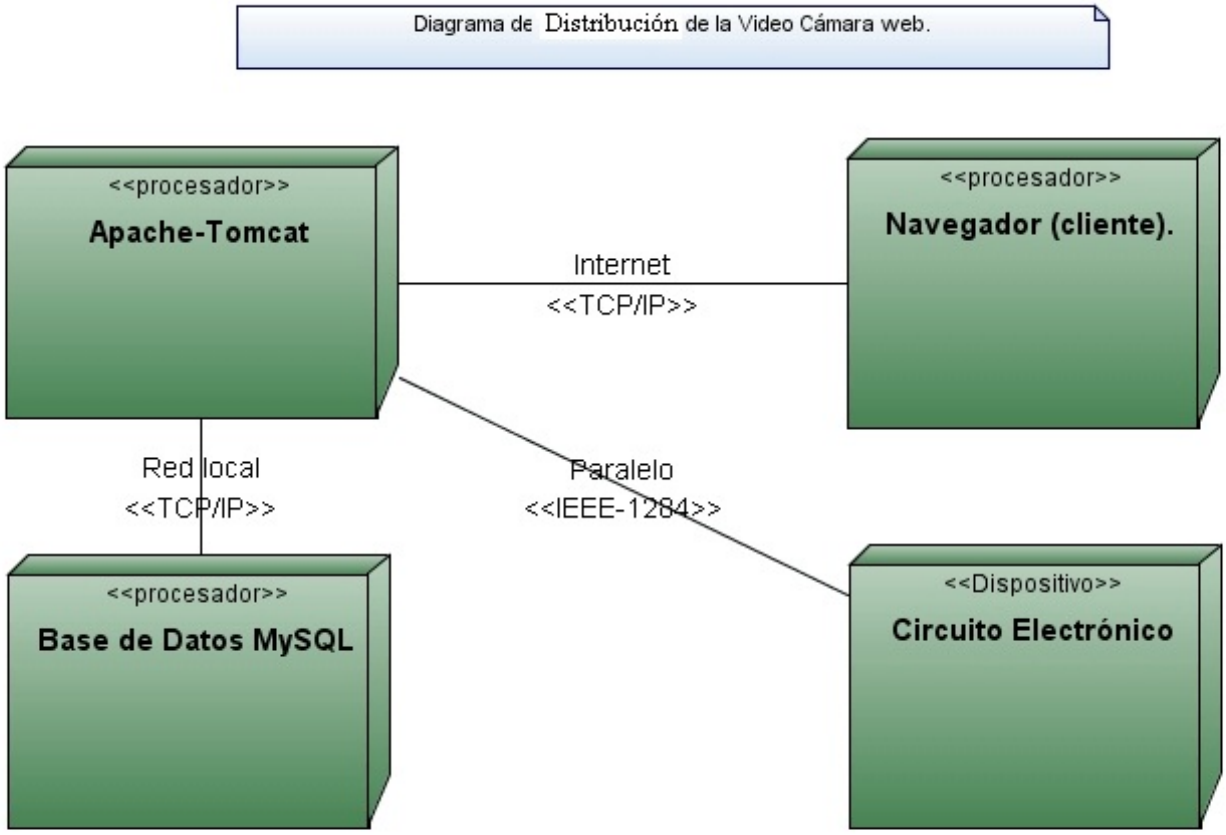



**Figura 39**: *Diagrama de Distribución de la Videocámara.*

# Anexo C.4. Diagrama de Clases.

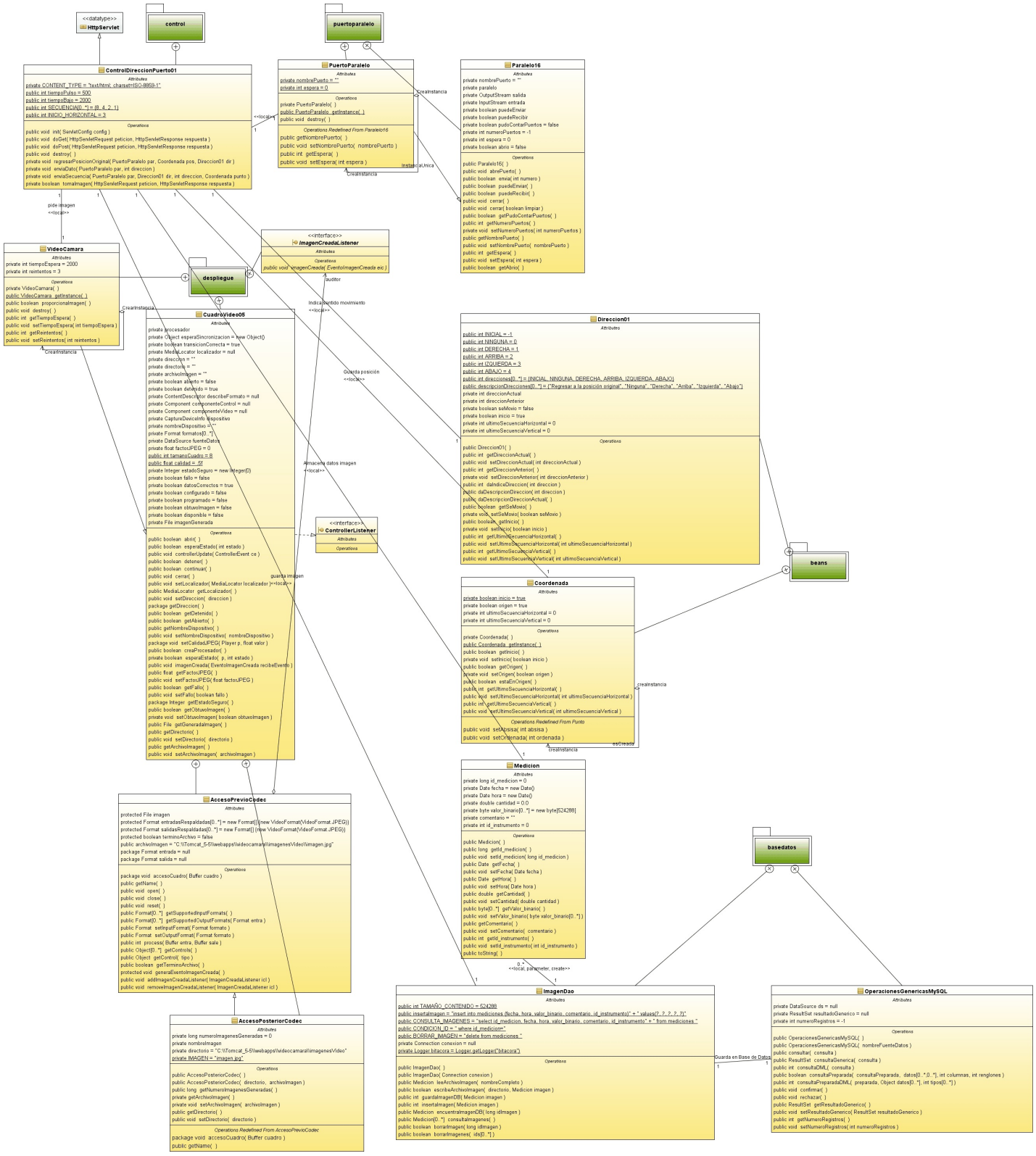



**Figura 40**: *Diagrama de Clases de la videocámara web.*

# Anexo C.5. Diagrama Actividades.

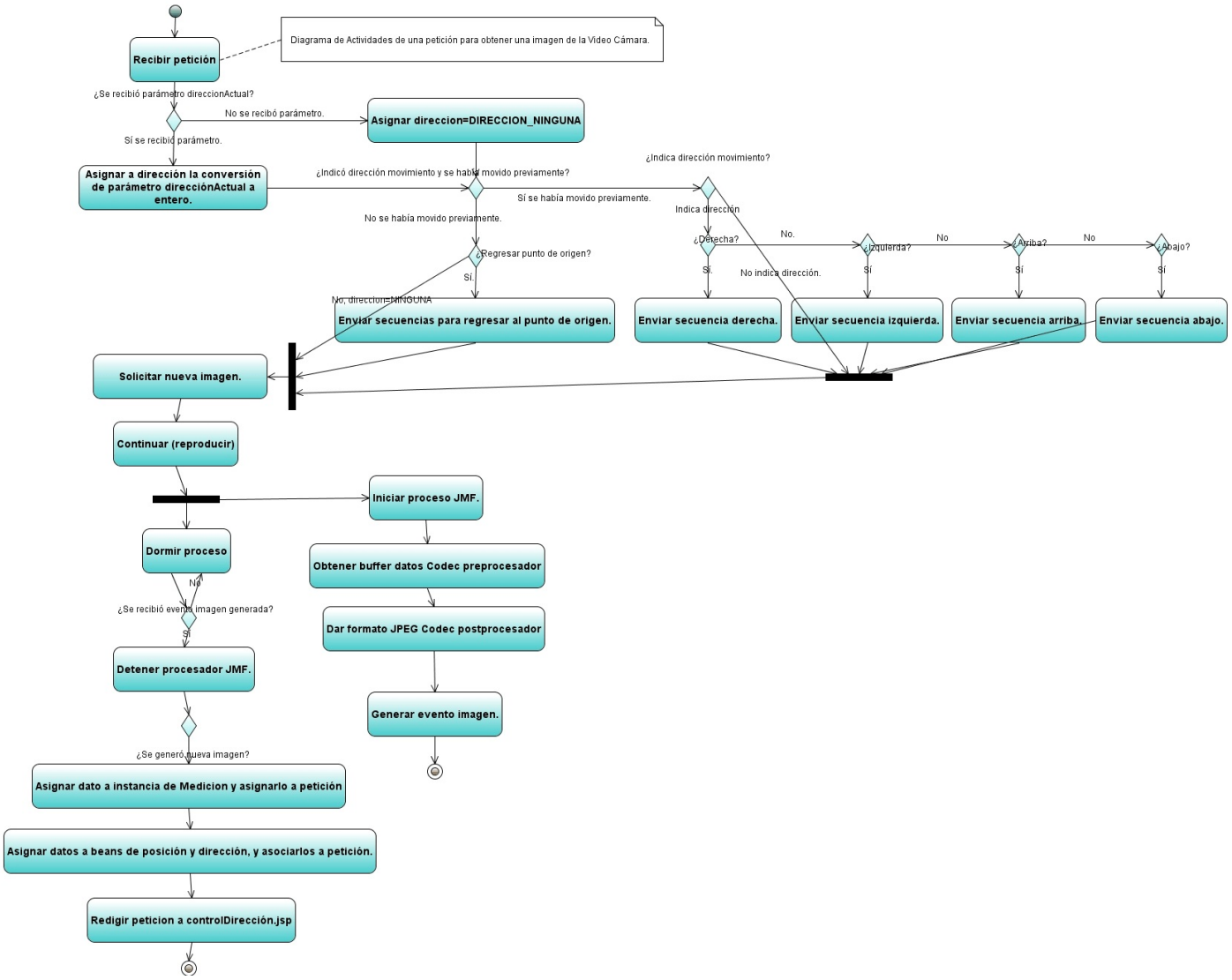



**Figura 41**: *Diagrama de Actividades de la Videocámara web.*

## Anexo C.6. Diagrama de Estados.

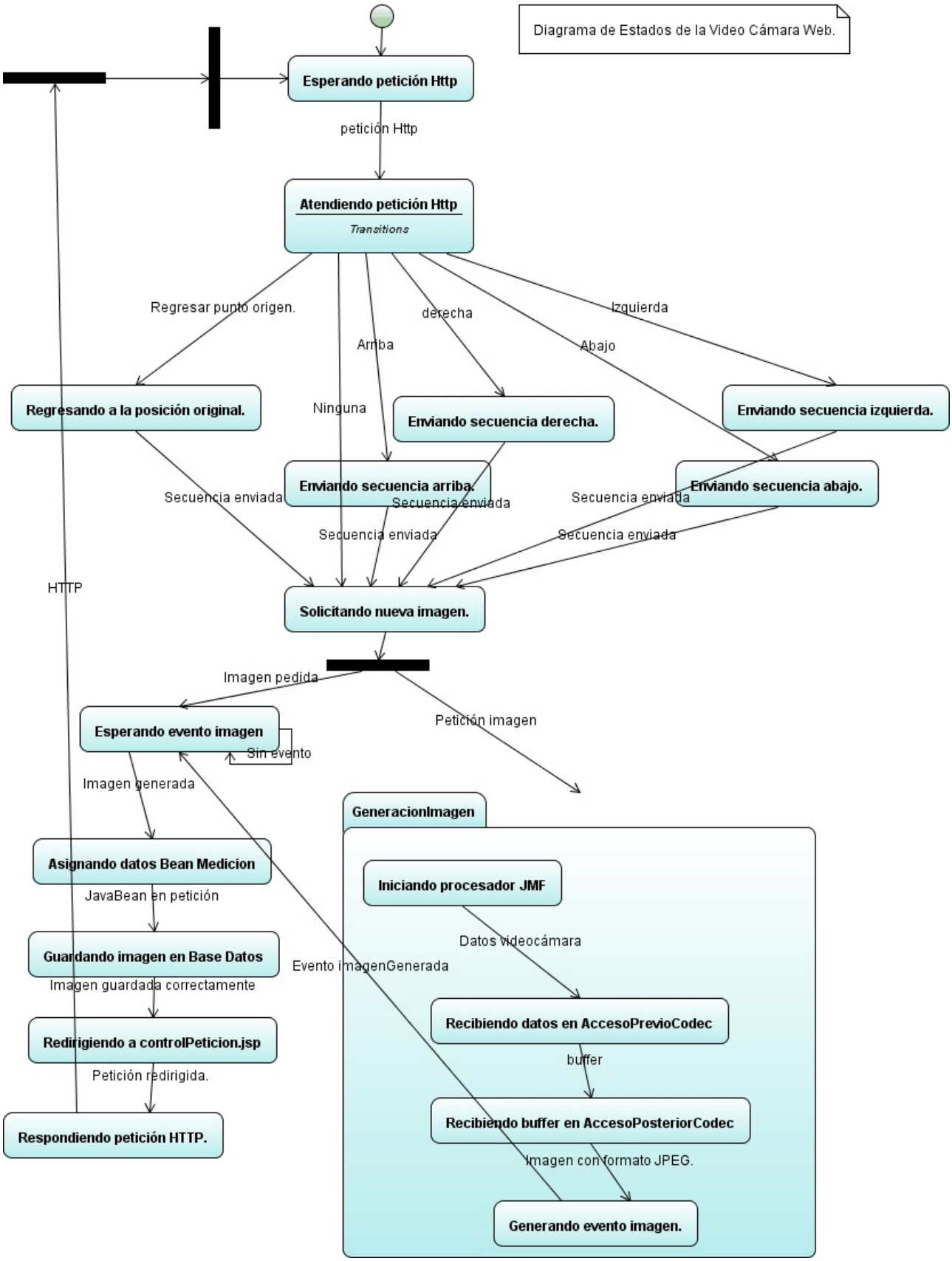



**Figura 42**: *Diagrama de Estados de la videocámara web.*

## Anexo C.7. Diagrama de Secuencias.

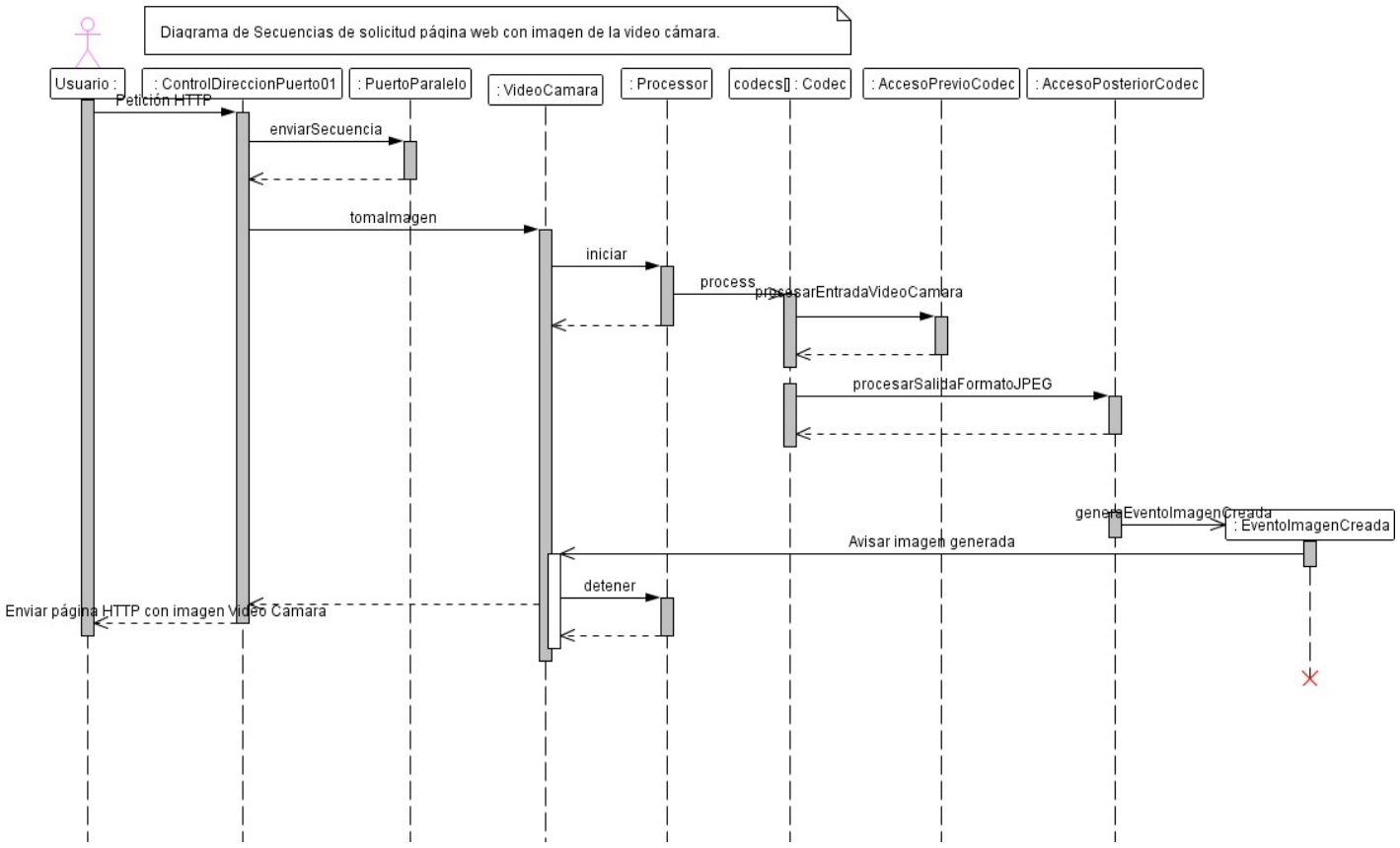



**Figura 43**: *Diagrama de Secuencias de la solicitud y obtención de imagen de la videocámara.*

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

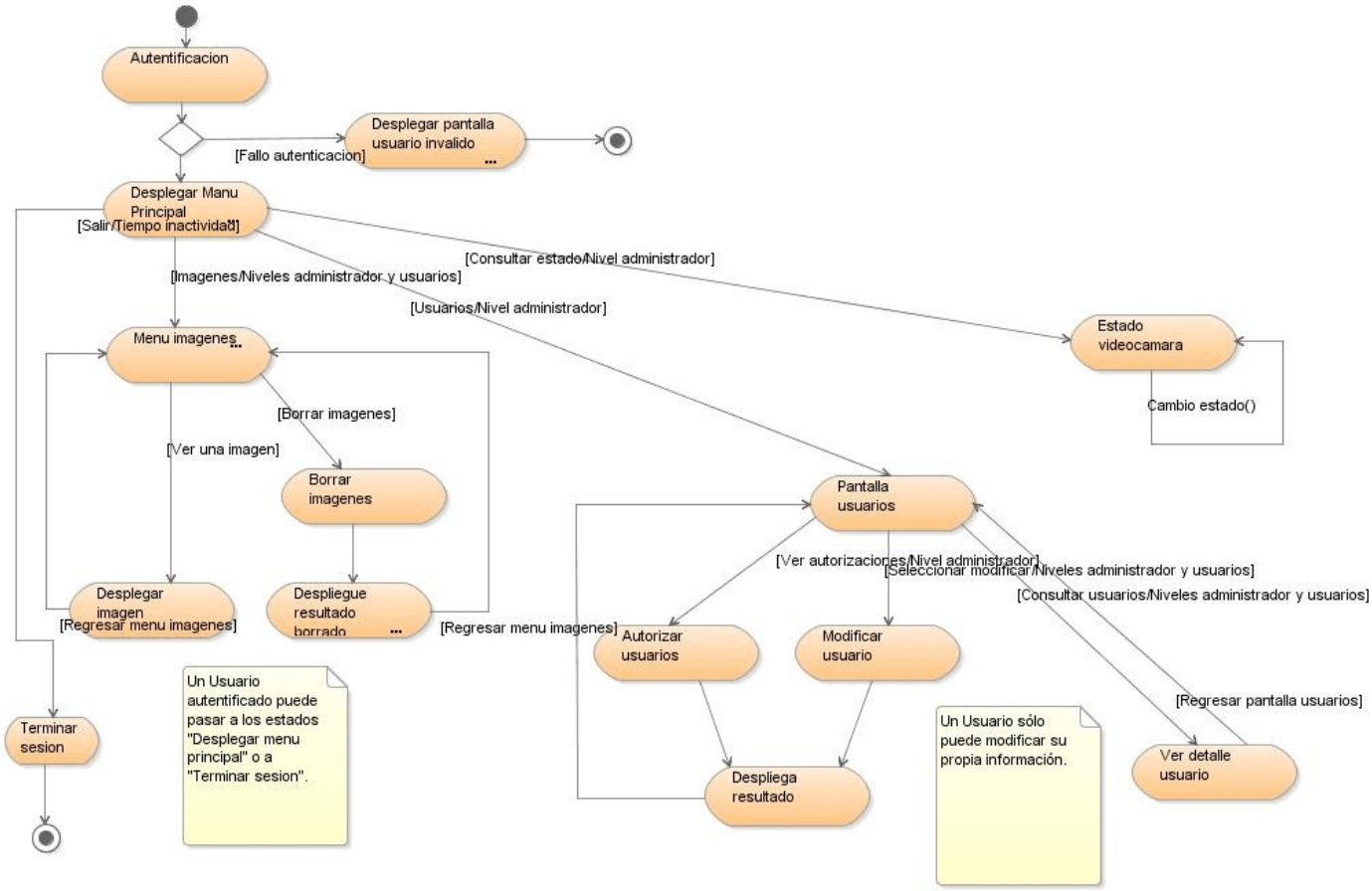



**Figura 44**: *Diagrama de estados de la navegación de la aplicación web videocámara.*

# 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.

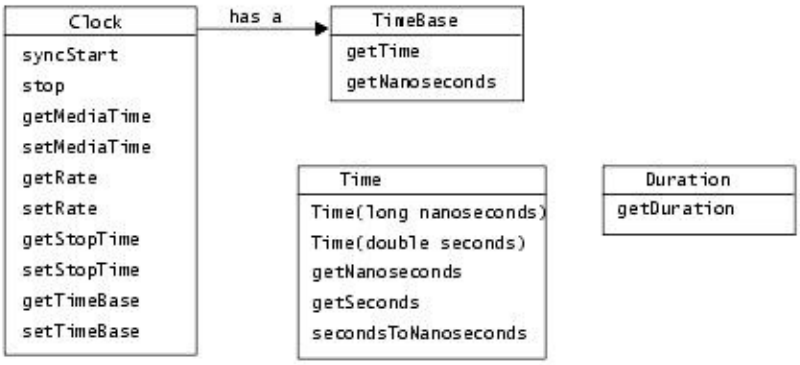



**Figura 46:** *Diagrama de clases del Modelo de Tiempo para JMF.*

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*.

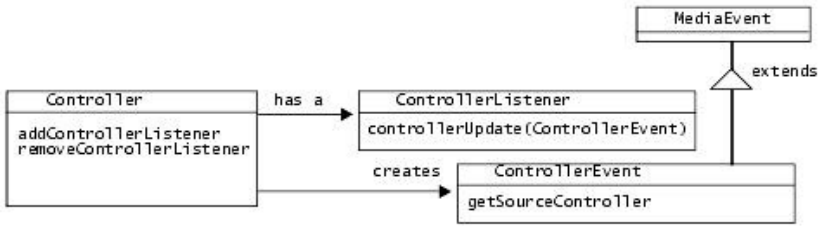



**Figura 47**: *Diagrama de clases del modelo de eventos de JMF.*

### 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*:

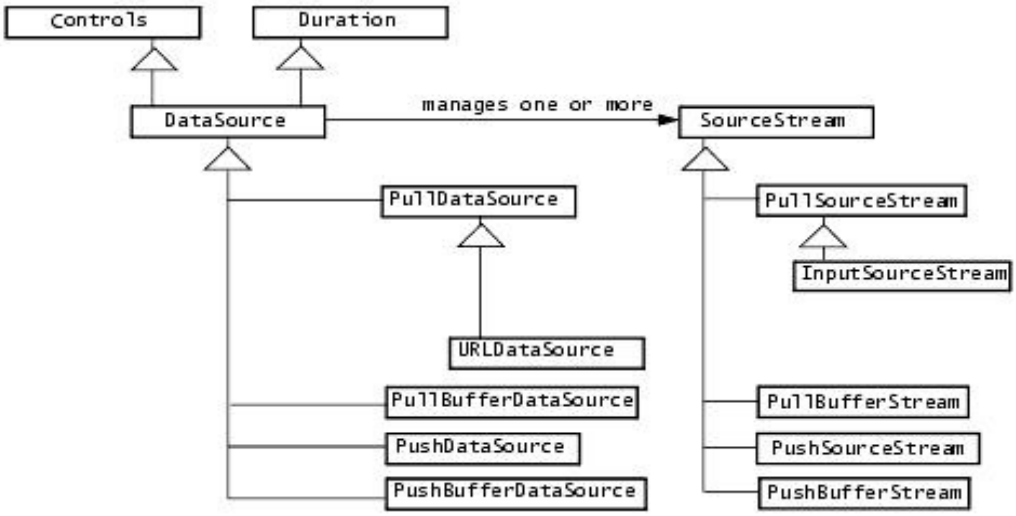



**Figura 48**: *Diagrama de clases de los tipos derivados de DataSource.*

*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*.

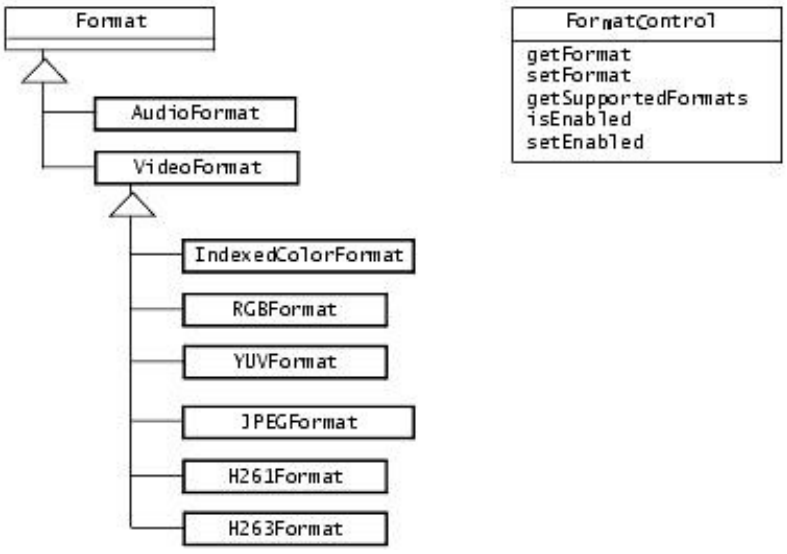



**Figura 49**: *Diagrama de clases de los formatos de video.*

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.

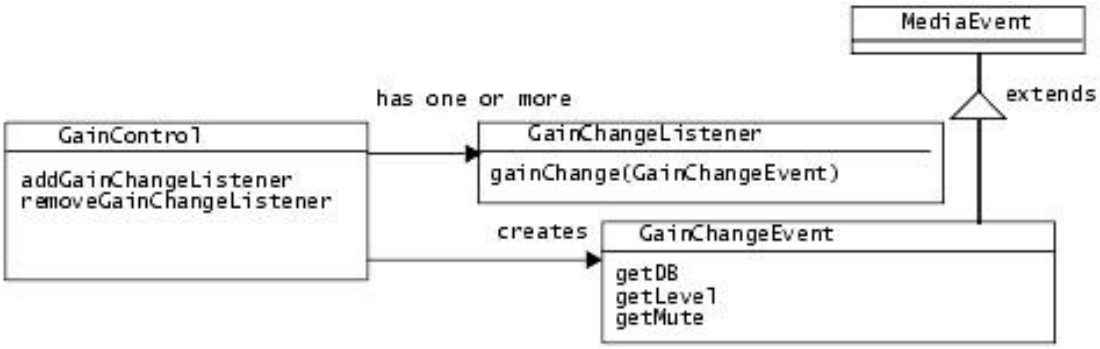



**Figura 50**: *Diagrama de clases de **Gain Control**.*

- *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.

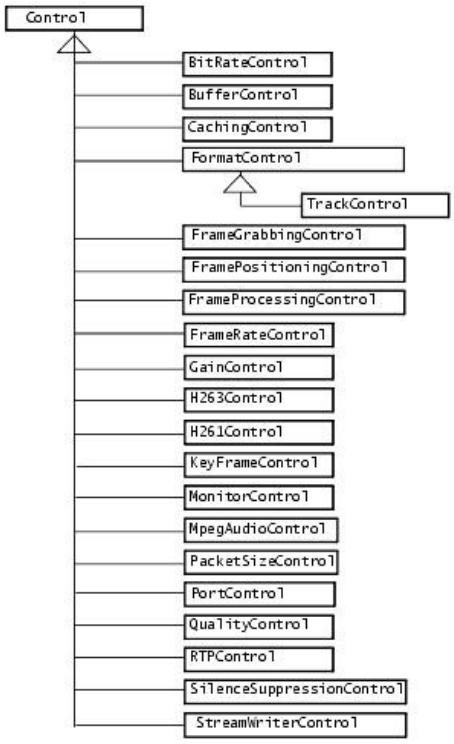



**Figura 51**: *Diagrama de clases de Control y de sus clases derivadas.*

*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.

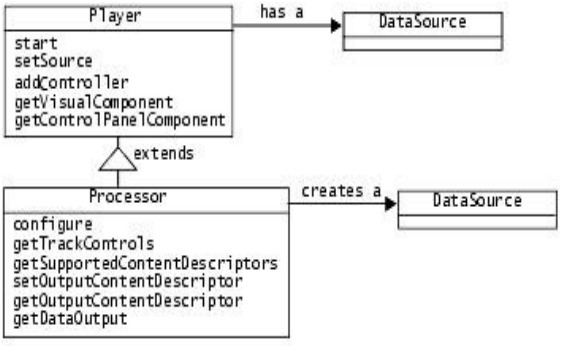



**Figura 52**: *Diagrama de clases de los controladores (reproductor y procesador).*

### 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*.

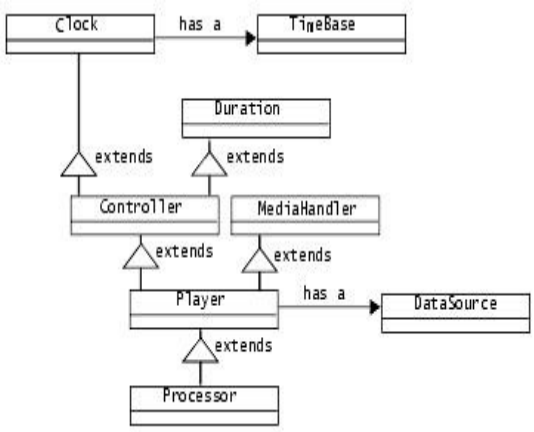



**Figura 53**: *Diagrama de clases del reproductor JMF.*

#### *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*).

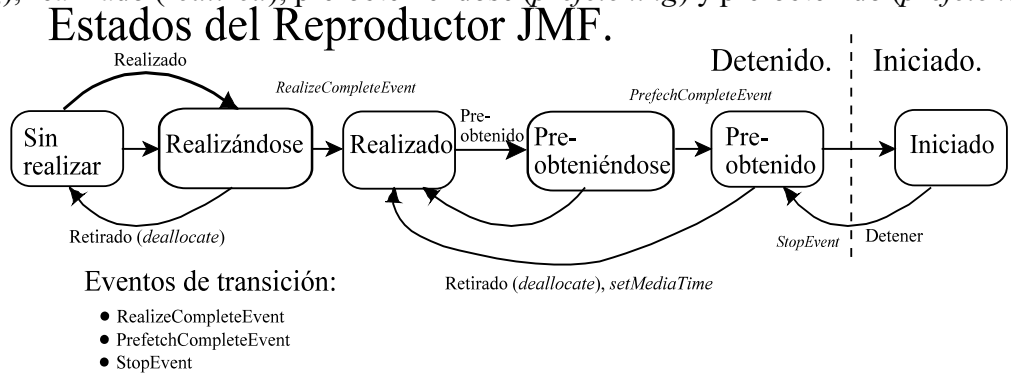



**Figura 54**: *Estados del Reproductor JMF.*

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.

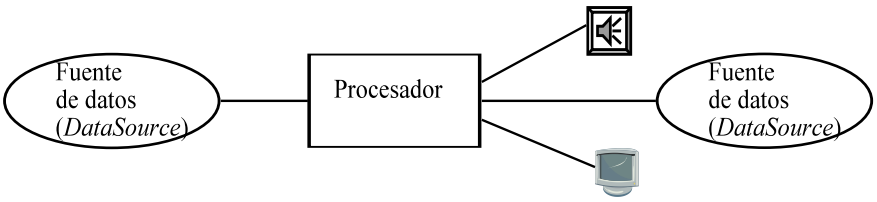



**Figura 55**: *Modelo del procesador JMF.*

#### *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.

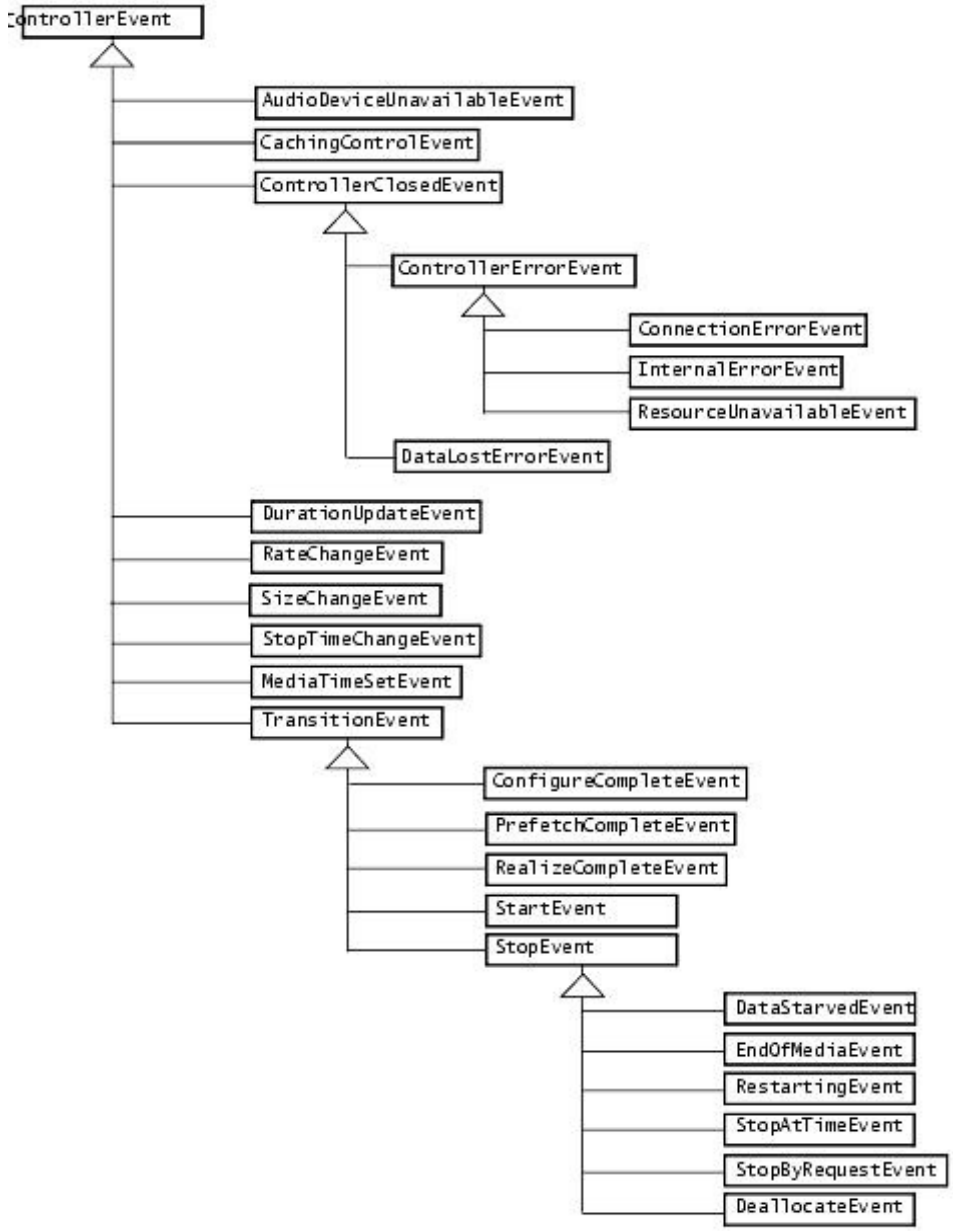



**Figura 56**: *Diagrama de clases de los eventos del controlador JMF.*

### 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.

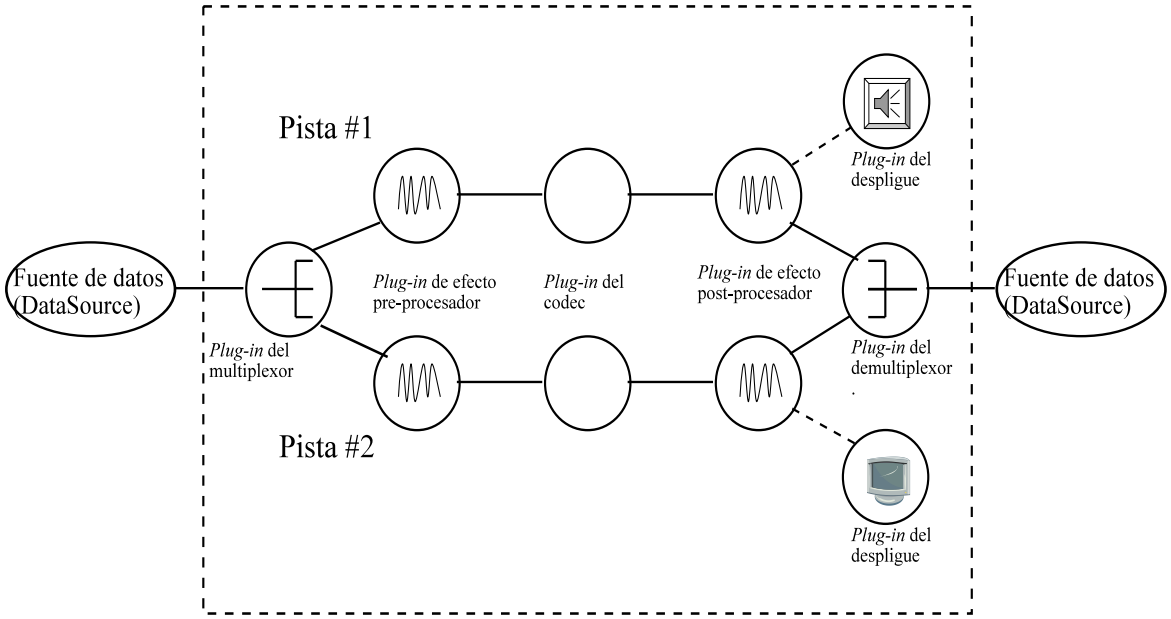



**Figura 57**: *Etapas del procesador JMF.*

- 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.*

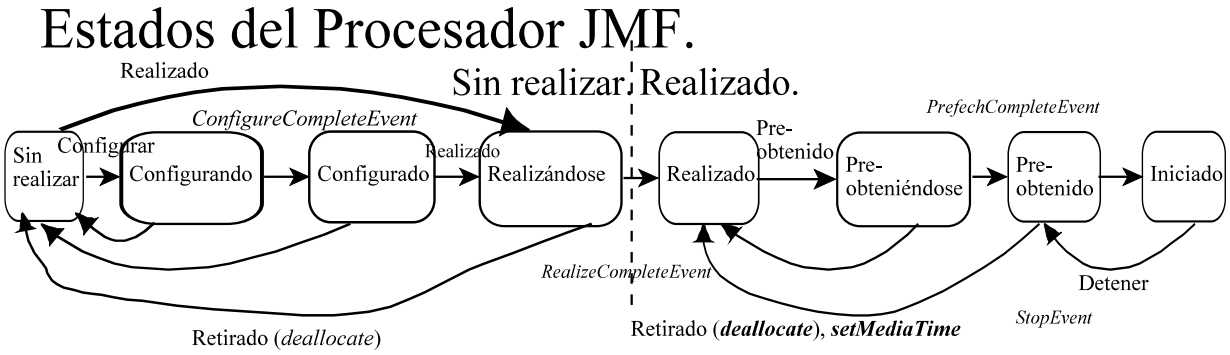
Un procesador tiene dos estados de espera adicionales, **configurando** y **configurado**, los cuales están antes del estado de realizando, y efectúan las siguientes tareas:

- 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*

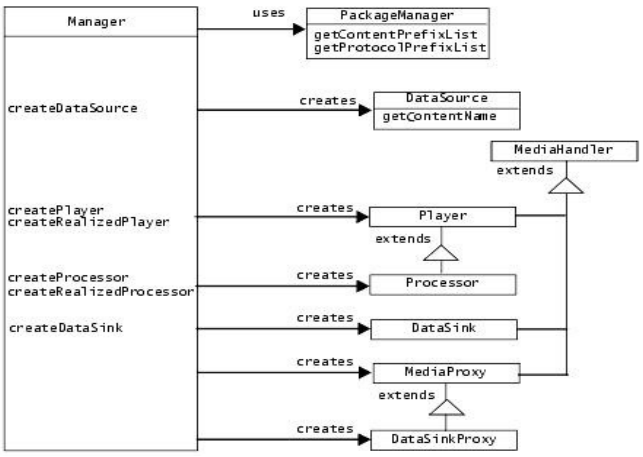



**Figura 59**: *Diagrama de clases del administrador de JMF y de los controladores de media.*

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.