Arquitectura de la JMF

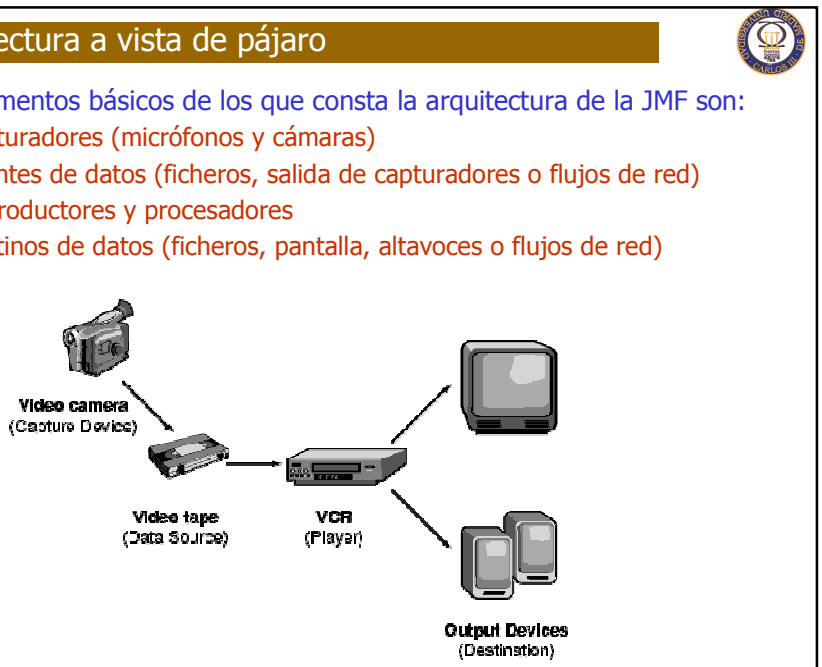
Servidores de Información Multimedia
2º Ingeniero Técnico de Telecomunicación – Imagen y Sonido

Departamento de Ingeniería Telemática
Universidad Carlos III de Madrid

2 Índice

- Arquitectura a vista de pájaro
- Modelo de tiempo
- Managers
- Modelo de eventos
- Fuentes de datos
- Tipos de datos
- Controles
- Componentes de la interfaz gráfica
- Extensibilidad de la JMF
- Procesadores y Players
- Sumideros de datos

3 Arquitectura a vista de pájaro



- Los elementos básicos de los que consta la arquitectura de la JMF son:
  - Capturadores (micrófonos y cámaras)
  - Fuentes de datos (ficheros, salida de capturadores o flujos de red)
  - Reproductores y procesadores
  - Destinos de datos (ficheros, pantalla, altavoces o flujos de red)

4 Modelo de tiempo

- Precisión de nanosegundos.
- Un instante de tiempo se representa por la clase Time.
- El diagrama de clases para el manejo del tiempo es el siguiente:

- Las clases de la JMF que manejan el tiempo implementan la interfaz *Clock*.
- Un objeto de clase *TimeBase* es un continuo tick de reloj.

5 Modelo de tiempo (cont)

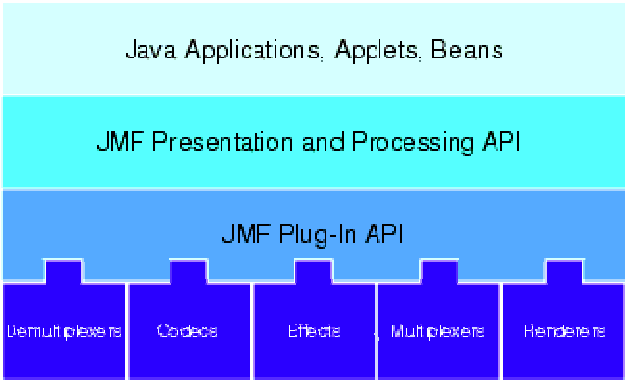
- Las clases que manejan el tiempo e implementan la interfaz *Clock* manejan dos tipos de tiempos:
  - El tiempo base o *Timebase* que representa el continuo paso del tiempo
  - El tiempo del medio o *MediaTime* que representa el instante de tiempo en el que se encuentra la reproducción del medio que va desde el instante 0 o inicio del medio hasta la duración del mismo.
- Las clases que implementan *Clock* tienen métodos para obtener ambos tiempos como objetos Time.
- Además se define la tasa de reproducción o "playback rate" que representa el número de unidades de tiempo de medio que transcurren cada unidad del tiempo base:
  - En una reproducción normal será 1.
  - En una reproducción hace adelante rápida tendrá un valor superior a 1.
  - En un rebobinado tendrá un valor negativo.
  - MediaTime = MediaStartTime + Rate(TimeBaseTime - TimeBaseStartTime)

6 Managers

- Los *managers* son clases que se utilizan para construir objetos. Por ejemplo, la construcción de un *Player* no la haremos directamente haciendo un *new* sino a través de un método del manager apropiado.
- Esto abre la posibilidad a contar con diferentes implementaciones de los objetos que tratan con los medios y será el manager apropiado quien instancie una clase u otra sin que el programador tenga que conocer los detalles de la implementación.
- Existen 4 tipos de managers básicos (existe un 5º pero lo dejamos de momento):
  - **Manager-** maneja la construcción de *Players*, *Processors*, *DataSources*, y *DataSinks*.
  - **PackageManager-** mantiene un registro de los paquetes Java que contienen clases de la JMF como *Players*, *Processors*, *DataSources*, y *DataSinks*.
  - **CaptureDeviceManager-**- mantiene un registro de los dispositivos capturadores.
  - **PlugInManager-** mantiene un registro de los plug-ins disponibles como *Multiplexers*, *Demultiplexers*, *Codecs*, *Effects*, y *Renderers*.

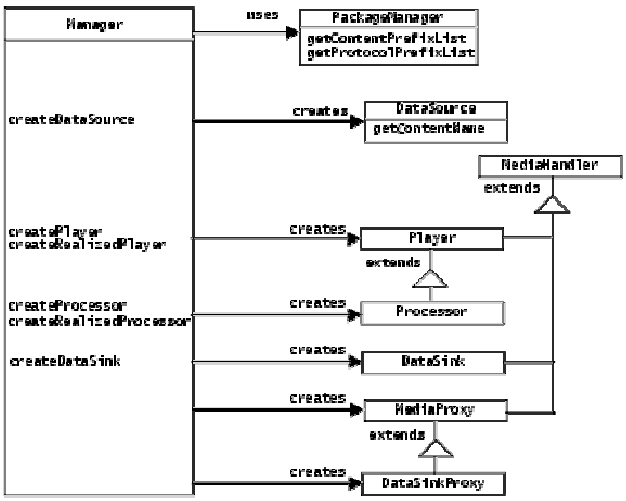
7 Managers (cont)

- Recordemos el modelo de extensión de la JMF:



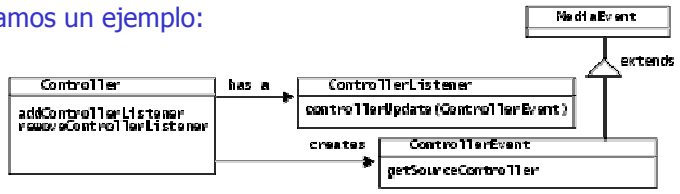
8 Managers (cont)

- La siguiente figura muestra más detalles de la clase Manager para la creación de los objetos de la JMF



9 Modelo de eventos

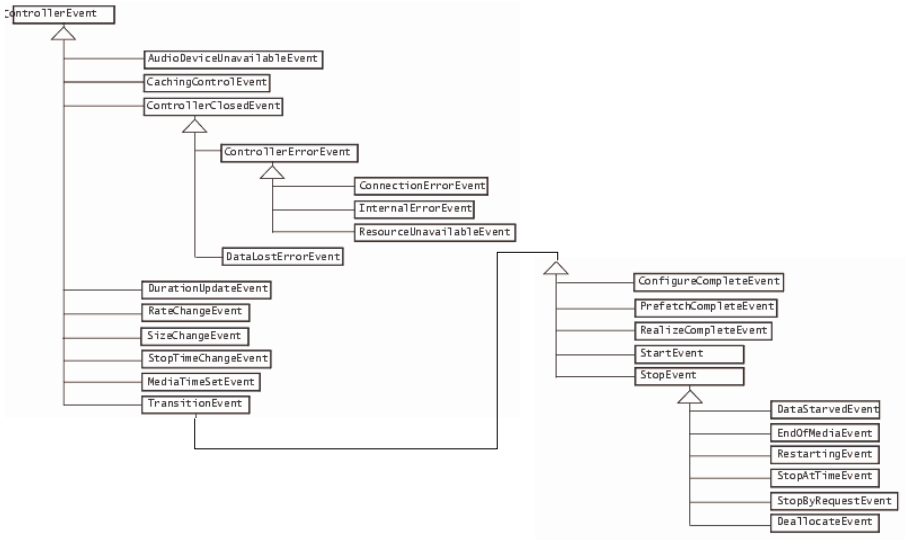
- Mecanismo basado en el mecanismo de Java basado en fuentes y escuchadores de eventos.
- Los objetos de la JMF lanzan diferentes tipos de eventos derivados de la clase *MediaEvent*.
- Nuestra aplicación puede escuchar esos eventos para responder a ellos suscribiendo un escuchador para el tipo de eventos deseado con la función add*Listener* apropiada en la fuente de eventos.
- Veamos un ejemplo:



- La clase *Controller* es la clase padre de los *Players* y *Processors*.
- Lanza *ControllerEvents* (por ejemplo para indicar cambios de estado en el controlador).
- Podemos implementar la interfaz *ControllerListener* en nuestra aplicación y registrarnos mediante *addControllerListener* en el controlador.

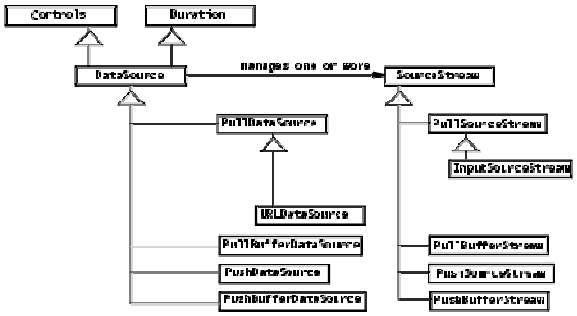
10 Ejemplo – los eventos de la clase *Controller*

- La clase *Controller* lanza los siguientes eventos



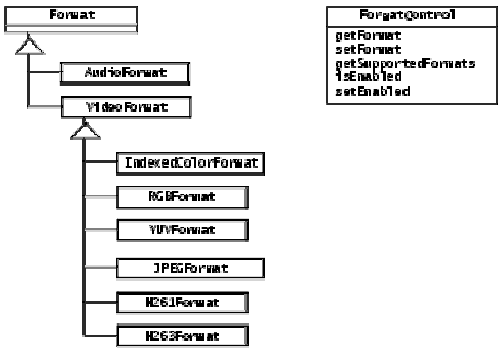
11 Fuentes de datos

- Los *DataSources* alimentan a los reproductores (Players) y procesadores (Processors).
- Dos tipos de fuentes de datos:
  - **Push**. Ejemplo broadcast por la red de una película
  - **Pull**. Película abierta del disco duro local
- Identificación-localización de una fuente de datos:
  - **MediaLocator** (puede construirse a partir de una URL). Puede usarse incluso sin que el protocolo al que referencia esté instalado en el sistema
  - **URL**
- Diagrama de clases:



12 Formatos de medios

- El formato de un tipo de medios se representa por la clase Format.
- La JMF extiende Format de la siguiente manera:

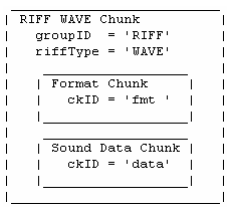


13 Formatos de tipos de ficheros

- A partir de la información en un fichero se crea un *DataSource* que lo lea.
- Un fichero puede contener varios medios (audio, vídeo).
- Por cada medio se construye un *Stream* con su formato asociado.
- Un *DataSource* tendrá tantos *Streams* como medios
- Un *DataSource* tendrá un método *getStreams()* para recuperar los Streams que contiene
- La JMF soporta varios tipos de formatos de ficheros:
  - AU, AVI, MIDI, MPEG, QuickTime, WAV…

14 Ejemplo - Formato de tipos de ficheros

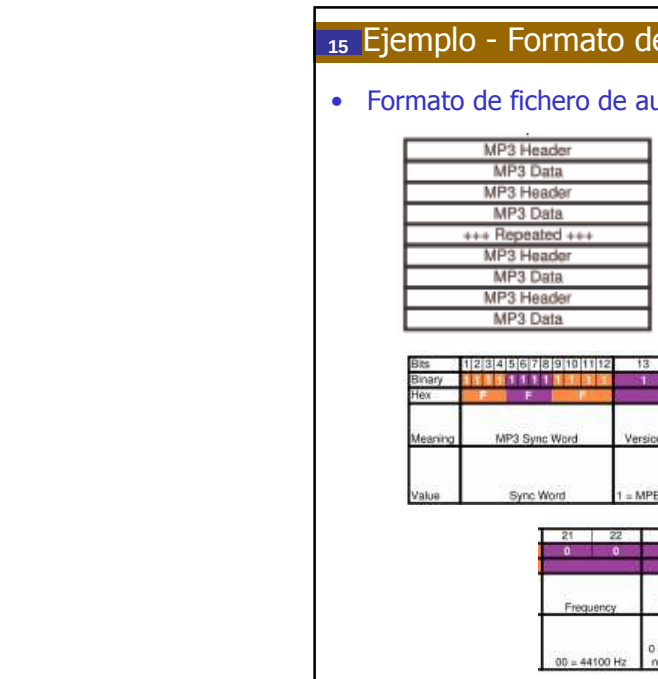
- Por ejemplo el formato WAV:
  - Acrónimo para Waveform
  - Formato para ficheros de audio por Microsoft e IBM
  - Variante del formato RIFF
  - Guarda los datos en fragmentos o "chunks"
  - Guarda normalmente audio sin comprimir pero soporta también formatos de audio con compresión
    - Formato por defecto LPCM (Pulse Code Modulation de Microsoft)
    - Algunos formatos de audio comprimidos soportados son: PCM, GSM, ADPCM, CELP, SBC, TrueSpeech y MPEG Layer-3



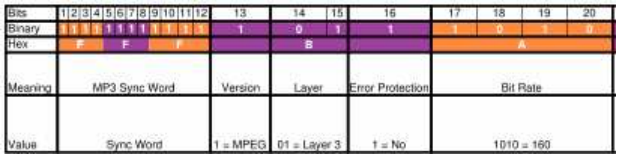

```
typedef struct {
  ID            chunkID;
  long          chunkSize;

  short         wFormatTag;
  unsigned short wChannels;
  unsigned long  dwSamplesPerSec;
  unsigned long  dwAvgBytesPerSec;
  unsigned short wBlockAlign;
  unsigned short wBitsPerSample;
} FormatChunk;
```

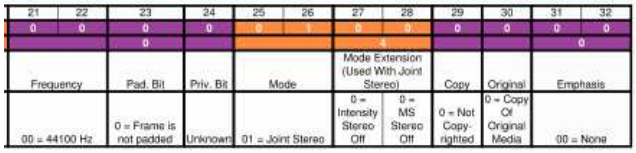
```
typedef struct {
  ID            chunkID;
  long          chunkSize;

  unsigned char  waveformData[];
} DataChunk;
```

15 Ejemplo - Formato de tipos de ficheros

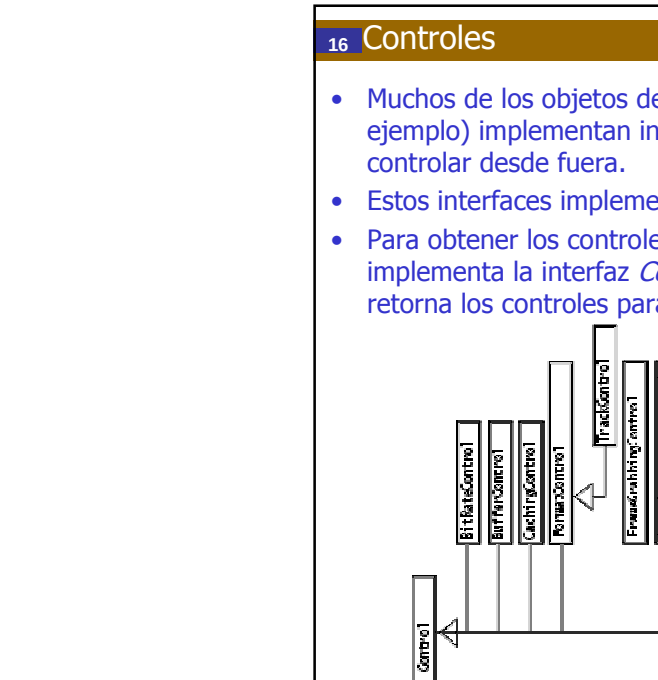


- Formato de fichero de audio mp3.

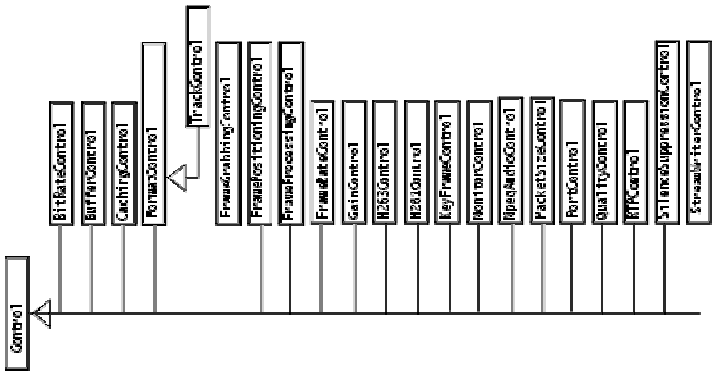




16 Controles

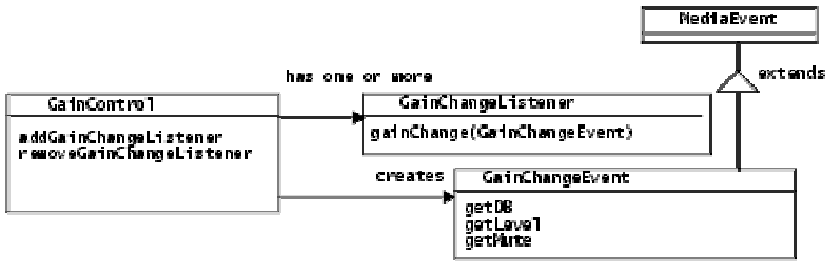


- Muchos de los objetos de la JMF (*DataSources*, *Controllers* y *DataSinks* por ejemplo) implementan interfaces a través de los cuales se les puede controlar desde fuera.
- Estos interfaces implementan los métodos de la interfaz *Controls*
- Para obtener los controles asociados a un determinado objeto, este implementa la interfaz *Controls* que tiene un método *getControls()* que nos retorna los controles para el objeto (que extenderá la interfaz *Control*).



17 Controles especiales

- **CachingControl** que permite mostrar el progreso en una descarga. Este control se obtiene directamente del evento de tipo CachingControlEvent lanzado por el Controller invocando el método *getCachingControl*().
- **GainControl** que permite el ajuste de volumen. Este control se obtiene del Player mediante el método *getGainControl*().



18 Componentes de la interfaz gráfica

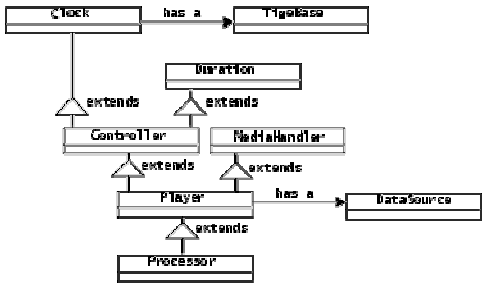
- Los controles pueden exponerse al usuario mediante un componente gráfico.
- Para obtener el componente de interfaz gráfico se llama al método *getControlComponent*() del interfaz Control o **CachingControl**.
- El componente devuelto se puede añadir como objeto awt a un contenedor.
- Los reproductores (Players) tienen un par de métodos para obtener los componentes visuales:
  - **getVisualComponent()** para obtener el componente donde se visualiza el vídeo reproducido.
  - **getControlPanelComponent()** para obtener el componente para el control de reproducción

19 Extensibilidad de la JMF

- Se tienen dos formas básicamente para extender la funcionalidad de la JMF:
  - Implementar clases a medida para el procesamiento de los medios (*plug-ins*) que pueden registrarse con el gestor de **plug-ins** y ser usados por los Processor de la JMF
  - **Implementando directamente** las interfaces Controller, Player, Processor, DataSource, o DataSink
- El primer método no requiere empezar de cero sino simplemente añadir la funcionalidad requerida (por ejemplo el codec).

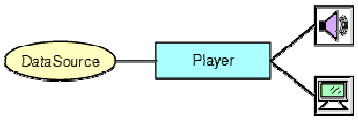
20 Presentación de medios en la JMF

- Para reproducir videos en pantalla y/o sonidos en los altavoces se usan los Players.
- Para procesar los medios y ofrecer los medios procesados como nuevas fuentes de datos, para guardar los medios procesados en disco o para retransmitirlos por la red se usan los Processors.
- La siguiente figura muestra el diagrama de clases para la presentación de medios:

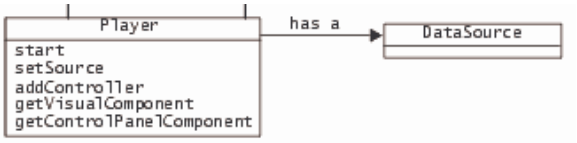


21 Players

- Lee una fuente de medios, los procesa y presenta (no los puede almacenar).
- El modelo se muestra en la siguiente figura:

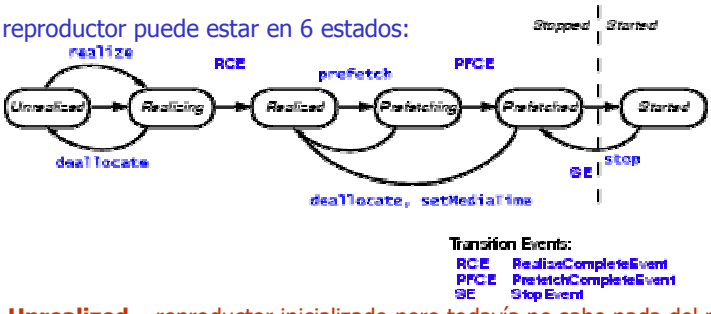


- Los reproductores (Players) no ofrecen la posibilidad de controlar externamente el procesamiento de los medios.
- Los reproductores si ofrecen interfaces para el control de la reproducción, el control del estado de la descarga del medio o el control de volumen.



22 Estados de un Player

- Un reproductor puede estar en 6 estados:



  - **Unrealized** – reproductor inicializado pero todavía no sabe nada del medio a reproducir.
  - **Realizing** – al llamar al metodo realize. Enproceso de determinar los requisitos de la fuente. Se adquieren los recursos del sistema (ejemplo acceso a tarjeta gráfica).Si el medio es en red se establece la conexión.
  - **Realized** – tras acabar el proceso de realización. A partir de aquí el Player puede devolver los componentes y controles asociados.
  - **Prefetching** – tras la llamada a prefetch. Se precarga el medio. Este estado puede ocurrir en un Player nuevo o tras haber cambiado el punto de reproducción.
  - **Prefetched** – al finalizar el prefetching. El Player está listo para reproducir.
  - **Started** – tras llamar a start. Los objetos de tiempo time-base y media time se mapean al reloj del sistema.

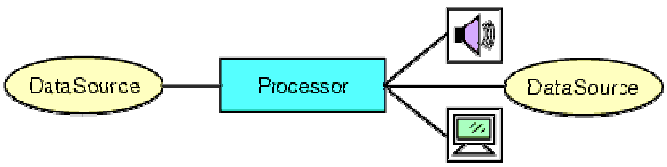
23 Métodos de un Player

- Dependiendo del estado en el que se encuentre un reproductor se pueden invocar sobre él una serie de métodos:

Method	Unrealized Player	Realized Player	Prefetched Player	Started Player
addController	NotRealizedError	legal	legal	ClockStartedError
deallocate	legal	legal	legal	ClockStartedError
getControlPanelComponent	NotRealizedError	legal	legal	legal
getGainControl	NotRealizedError	legal	legal	legal
getStartLatency	NotRealizedError	legal	legal	legal
getTimeBase	NotRealizedError	legal	legal	legal
getVisualComponent	NotRealizedError	legal	legal	legal
mapToTimeBase	ClockStoppedException	ClockStoppedException	ClockStoppedException	legal
removeController	NotRealizedError	legal	legal	ClockStartedError
setMediaTime	NotRealizedError	legal	legal	legal
setRate	NotRealizedError	legal	legal	legal
setStopTime	NotRealizedError	legal	legal	StopTimeSetError if previously set
setTimeBase	NotRealizedError	legal	legal	ClockStartedError
syncStart	NotPrefetchedError	NotPrefetchedError	legal	ClockStartedError

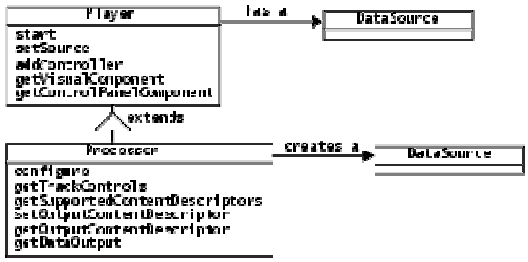
24 Procesadores (Processors)

- Los procesadores extienden a los reproductores.
- A diferencia de los reproductores que muestran el vídeo por pantalla y el audio por el sistema de sonido, los procesadores pueden ofrecer su salida como fuente de datos para otros procesadores o reproductores o pueden dirigir su salida a un "sumidero" de datos (*DataSink*).
- La siguiente figura muestra la cadena de medios en la que se insertan los procesadores



25 Las clase Player y Processor

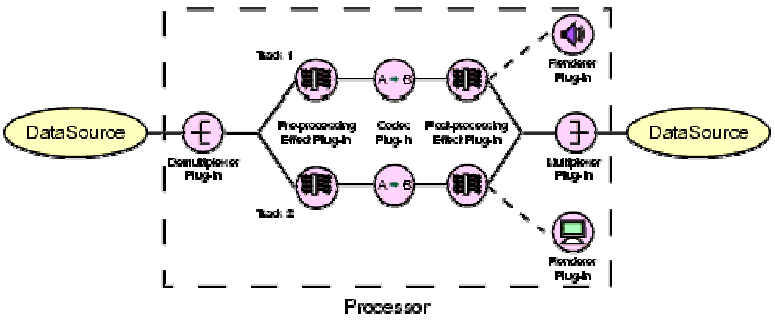
- Como hemos indiciado, la clase Processor extiende a la clase Player que a su vez extiende a la clase Controller.
- A modo simplificado veamos la siguiente figura:



- Tanto reproductores como procesadores reciben como entrada una fuente de medios o DataSource.
- Los procesadores pueden entregar la salida a un DataSink, ofrecerla como un nuevo DataSource o renderizarla en el sistema de audio-video

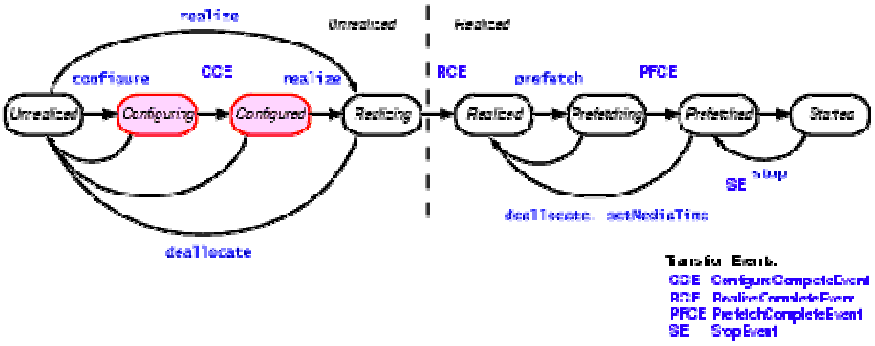
26 El esquema detallado de un procesador

- En la figura se muestra un esquema más detallado de un procesador por dentro.
- Para realizar el procesamiento de los medios, el procesador hace uso de plug-ins: *demultiplexor*, *efectos*, *codecs*, *mutiplexor* o *renderizador*.
- Mediante la implementación de pulg-ins a medida podemos extender la JMF.



27 Los estados de un procesador

- Añade un par de estados previos a los estados del Player:
  - **Configuring** – tras la llamada al método configure(). Se usa para abrir el DataSource y obtener información del mismo.
  - **Configured** – al finalizar la configuración. Este estado se puede usar para obtener los TrackControls (con el método **getTrackControls**()) que sirven para cambiar los plug-ins que se usarán para cada uno de los tracks.



28 Métodos de un Processor

Method	Unrealized Processor	Configuring Processor	Configured Processor	Realized Processor
addController	NotRealizedError	NotRealizedError	NotRealizedError	legal
deallocate	legal	legal	legal	legal
getControlPanelComponent	NotRealizedError	NotRealizedError	NotRealizedError	legal
getControls	legal	legal	legal	legal
getDataOutput	NotRealizedError	NotRealizedError	NotRealizedError	legal
getGainControl	NotRealizedError	NotRealizedError	NotRealizedError	legal
getOutputContentDescriptor	NotConfiguredError	NotConfiguredError	legal	legal
getStartLatency	NotRealizedError	NotRealizedError	NotRealizedError	legal
getSupportedContent-Descriptors	legal	legal	legal	legal
getTimeBase	NotRealizedError	NotRealizedError	NotRealizedError	legal
getTrackControls	NotConfiguredError	NotConfiguredError	legal	FormatChange-Exception
getVisualComponent	NotRealizedError	NotRealizedError	NotRealizedError	legal
mapToTimeBase	ClockStoppedException	ClockStoppedException	ClockStoppedException	ClockStopped-Exception
realize	legal	legal	legal	legal
removeController	NotRealizedError	NotRealizedError	NotRealizedError	legal
setOutputContentDescriptor	NotConfiguredError	NotConfiguredError	legal	FormatChange-Exception
setMediaTime	NotRealizedError	NotRealizedError	NotRealizedError	legal
setRate	NotRealizedError	NotRealizedError	NotRealizedError	legal
setStopTime	NotRealizedError	NotRealizedError	NotRealizedError	legal
setTimeBase	NotRealizedError	NotRealizedError	NotRealizedError	legal
syncStart	NotPrefetchedError	NotPrefetchedError	NotPrefetchedError	NotPrefetchedError

29 Construcción de Players y Processors

- Se llama al *Manager.createPlayer* o *Manager.createProcessor (*ambos reciben un *DataSource*).
- El Manager llama al método getContentName del DataSource (le llamamos <nombreContenido> en esta transparencia).
- El Manager pregunta al PackageManager por todos los paquetes registrados en la JMF (a cada prefijo de cada paquete le llamamos <content package-prefix>).
- Para crear un Player el Manager crea un objeto de la clase
  - <content package-prefix>. media.content.<nombreContenido>.Handler
- Para crear un Processor:
  - <content package-prefix>.media.processor.<nombreContenido>.Handler
- Para un DataSink:
  - <content package-prefix>.media.datasink.protocol.< nombreContenido >.Handler

Cuestiones de repaso

- Diga cuáles son los cuatro elementos básicos de los que consta la arquitectura JMF
- Diga cuál es la relación existente entre *MediaTime*, *TimeBase*, *Rate* y *TimeBaseStartTime* mediante la fórmula que los relaciona
- Diga cuál es la utilidad de un *Manager*
- Cuáles son los cuatro tipos de *Managers* definidos por JMF
- Diga cuál de los siguientes métodos no está definido en la clase **Manager**:
  - *createDataSource*, *createPlayer*, *createRealizedPlayer*, *CreatedProcessor*, *CreateRealizedProcessor, createDataSink, createRealizedDataSink*
- ¿Cómo sabe el **Manager** cuales son las clases que puede utilizar a la hora de crear un **Player** o un **DataSource**?
- Viendo el diagrama de eventos de la transparencia 10 diga qué evento generaría el multiplicar la velocidad de reprodución x2 en un **Player**. Compruébelo su hipótesis consultando el API de JMF

Cuestiones de repaso (cont)

- Diga cuáles son los dos tipos de fuentes de datos que identifica un *DataSource*
- ¿Cuál puede ser el motivo por el cual JMF modela diferentes tipos de formatos multimedia? ¿En qué tipo de aplicaciones se utilizan estos datos?
- ¿Cuántos *streams* tiene un *DataSource*?
- ¿Con qué método se obtienen los controles de un *DataSource*?
- ¿Cuál de los que siguen implementa la interfaz *Controls*?
  - GainControl, RTPControl, KeyFrameControl, MpegAudioControl, SilenceSuppressionControl
- ¿Qué funcionalidad especial nos permite controlar *CachingControl*? y ¿*GainControl*?
- ¿Cómo accedemos a los métodos que permiten obtener los componentes visuales de un *Player*?
- Describa brevemente los dos mecanismos que hay para extender JMF. ¿Cuál de ellos es mejor si no se quiere empezar desde cero?
- Describa la relación de herencia existente entre *MediaHandler*, *Player* y *Processor*.

Cuestiones de Repaso (cont)

- En principio, ¿puede un *Player* devolver el flujo que será almacenado en un fichero?
- Cite los seis estados por los que pasa un *Player*.
- Tomando la tabla de la transparencia 23 como punto de partida, ¿por qué resulta recomendable llamar al método *setRate*() después de *syncStart*()?
- ¿Puede el flujo de salida de un *Processor* almacenarse en un fichero?
- ¿Cómo se llaman los dos estados que introduce el *Processor* en el modelo de un *Player*? ¿Para que se usan?
- Vuelva a la trasparencia 28, ¿Puede llamarse al método *getTrackControls* cuando el *Processor* está en el estado de *Realized*? ¿Qué pasaría?
- ¿Hay forma de elegir la clase que utilizar *Processor* o el *Player* para instanciar un cierta clase en el *Manager*?¿Cómo?
- ¿Y para un *DataSink*? ¿Cómo?

Autoría

- Mario Muñoz Organero
- Pablo Basanta Val

  + Cuestiones de control