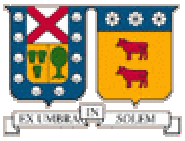
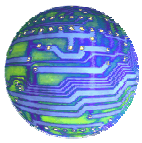


UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
Programación de Sistemas



PROYECTO SEMESTRAL

"JAVA MEDIA FRAMEWORK"

Integrantes	Christian Nievas G.
	Sergio Catalán O.
Fecha	**29 Octubre 2003**

Resumen.

Este Informe tiene como fin presentar de una manera más entendible el concepto que hay debajo de JMF (Java Media Framework), que gracias a las ventajas que otorga la plataforma Java, JMF promete que sus aplicaciones se "Escriben una vez, corren en cualquier parte". Los temas que se tocaran en el presente informe son los siguientes:

→ Una presentación de los "Medios Basados en el Tiempo" (Time-Based Media).

→ El funcionamiento de JMF para trabajar con estos tipos de medios.

→ Presentando medios basados en el tiempo con JMF.

→ Procesando medios basados en el tiempo con JMF.

→ Capturando datos de medios con JMF.

Por ultimo se presentaran ejemplos de usos de JMF y detalles de esta herramienta en el anexo adjunto, y se ha habilitado una pagina en donde están este informe , el anexo y las referencias mas importantes para que el lector pueda interiorizarse de mejor manera con lo que se expondrá.

1. Introducción.

Java™ Media Framework (JMF) es un API (Application Programming Interface), para la incorporación de medios basados en el tiempo (time-based medias) en aplicaciones Java y Applets. Los medios basados en el tiempo son medios tales como el audio, video, MIDI y animaciones que cambian con respecto al tiempo.

Inicialmente, el JMF 1.0 API, habilitaba a los programadores para desarrollar software de tipo Java que presentaban estos tipos de medios. Actualmente, el JMF 2.0 API, extiende el área de trabajo, para proveer soporte para captura y almacenaje de medios (basados en el tiempo), controlando el tipo de procesamiento que es efectuado durante la reproducción y para la personalización del procesamiento sobre flujos de medios.

Los objetivos principales del diseño de JMF 2.0 API son:

- Ser más fácil para programar.
- Soporte para la captura de medios.
- Habilita el desarrollo de aplicaciones para flujos de medios (media streaming) y conferencias en Java.
- Habilitación de tecnología y desarrollo avanzado que permita implementar soluciones personalizadas basadas en APIs ya existentes y nuevas características fácilmente integrables.
- Proveer el acceso a datos *Raw Media.*
- Habilitar el desarrollo de demultiplexores, codecs, procesadores de efectos, multiplexores, y renderers personalizables y descargables (JMF plug-ins).
- Mantener la compatibilidad con JMF 1.0 API.

2. Datos Basados En El Tiempo (Time-Based Media).

Toda información que tenga cambios significativos con respecto al tiempo puede ser catalogada como un medio basado en el tiempo, como lo son los clips de audio, secuencias MIDI, clips de video, etc. Estos medios pueden ser obtenidos de diversas fuentes, como archivos locales o remotos, cámaras, micrófonos y difusiones en vivo.

A continuación se presenta un modelo que describe las características de estos medios y la manipulación que a estos se les debe aplicar.

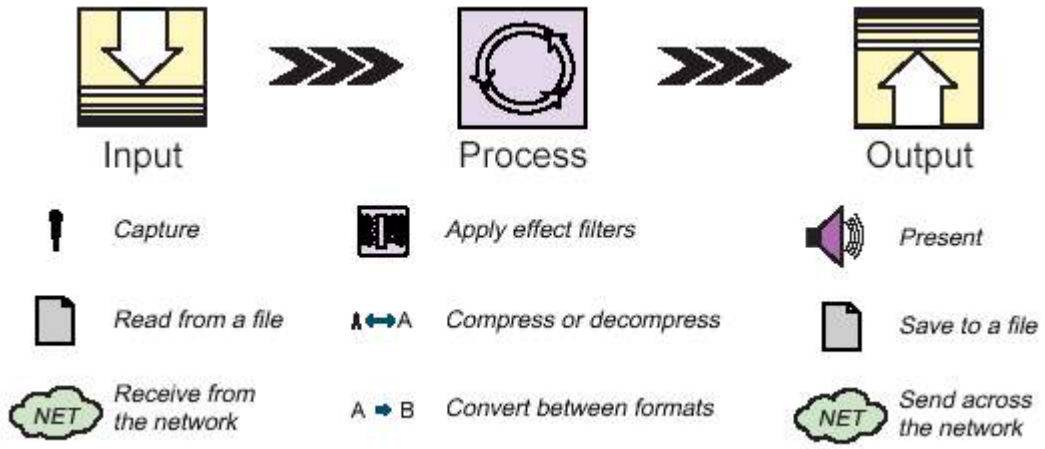


Figura 1.
Modelo de Procesamiento de Medios.

Las características claves de los medios basados en el tiempo son las siguientes:

- ***Flujo de Medios (Streaming Media).***

  La característica principal de este medio es que requiere de un tiempo de entrega y de procesamiento, y por esto se debe controlar, ya que una vez iniciado el flujo de datos, se deben satisfacer ciertos limites de tiempo.

- ***Presentación de Medios (Output).***

  La mayoría de estos medios pueden ser presentados a través de dispositivos de salida tales como parlantes y monitores, u otras destinaciones (Ej.: a la red). Comúnmente a estos destinos de medios se le llaman D*ataSinks.*

- ***Procesamiento de Medios (Process).***

  En muchas instancias, la información contenida en un medio es manipulada antes de ser presentado al usuario, ya sea multiplexándola, filtrándola, comprimiéndola, o convirtiéndola en otro tipo de medio.

- ***Captura de Medios (Input).***

  Estos pueden ser capturados desde una fuente en vivo para procesarla y reproducirla o puede ser adquirida de un archivo de forma remota.

3. Entendiendo a JMF.

JMF provee una arquitectura y un protocolo de mensajes unificado, para administrar la adquisición, procesamiento y entrega de medios basados en el tiempo. Para llevar a cabo su cometido, JMF cuenta con las siguientes características:

- ***Arquitectura de alto nivel.***

  Una parte integral de JMF son las fuentes de datos y los reproductores para la administración de la captura, presentación y procesamiento de los medios, presentando la estructura de la figura 2.

  Algunos de los principales elementos (clases e interfaces) de JMF se presentan a continuación[1]:

  ♦ Time model. ♦ Managers. ♦ Event Model.

  ♦ Data Model. ♦ Controls.

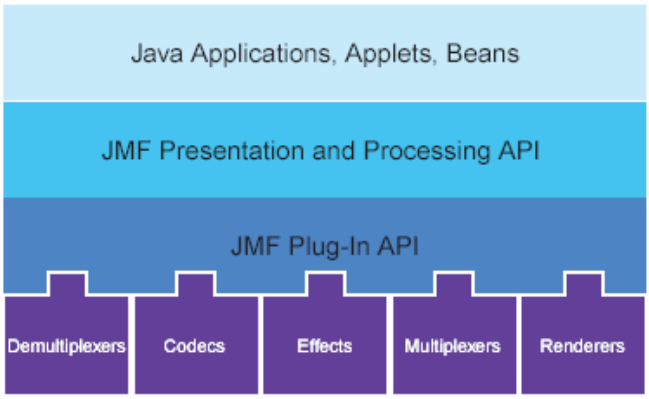


Figura 2.
Arquitectura de Alto Nivel.

- ***Presentación.***

El proceso de presentación es modelado por la interfaz controladora, que define el estado básico y el mecanismo de control para el objeto presentado o capturado. En este caso se definen 2 tipos de controladores: Reproductores y Procesadores.

- ***Procesamiento.***

El procesador es un reproductor que toma una fuente de datos, realiza algunos procesos definidos por el usuario, y luego entrega la información de medio procesada. Las etapas por las que el medio atraviesa son la de demultiplexación, Transcoding, Multiplexación, Renderización.

- ***Captura.***

Un dispositivo de captura multimedia puede actuar como una fuente de entrega de medios basados en el tiempo. Algunos dispositivos entregan múltiples flujos de datos que pueden ser separados mediante el procesamiento.

- ***Almacenamiento y transmisión de medios.***

Un *DataSink* es usado para leer medios desde una fuente y renderizarlo a algún destino. Particularmente el *DataSink* puede escribir datos a un archivo, escribir datos a través de la red o funcionar como un RTP broadcaster.

4. Presentando medios basados en el tiempo con JMF

Para presentar medios con JMF se debe usar un reproductor, la reproducción puede ser programada o se puede controlar desde el panel. Un *Processor* es un tipo especial de reproductor que provee control de cómo se procesan los datos antes de ser presentados. Un *Processor* puede ser creado usando un `ProcessorModel`, llamando al metodo `Manager.createRealizedProcessor`. El `ProcessorModel` define los requerimientos de entrada y salida para el *Processor*.

El *Media Player bean* es un Java Bean que encapsula un JMF Player para proveer una vía fácil para presentar flujos de medios desde un applet o aplicación. El *Media Player bean* automáticamente construye un nuevo reproductor cuando un flujo de medios diferente es seleccionado, lo cual hace mas fácil reproducir una serie de clips o permite que el usuario seleccione cual clip quiere reproducir.

Se puede crear indirectamente un reproductor usando el media *Manager*. Para mostrar el reproductor se deben agregar los componentes de objetos en su espacio de presentación de la applet o en la ventana de aplicación. Cuando se necesita crear un nuevo reproductor, este se pide al *Manager* llamando a `createplayer` o `createprocessor`.

Un reproductor generalmente tiene dos tipos de componentes en la interfase, un componente visual y un panel de control, en JMF esto se puede setear a elección, cada componente debe ser agregado a la applet para que pueda ser usado, ya sea el control de volume, de progreso, etc. También se puede setear la posición inicial, el contador de tiempo y otros componentes típicos de los reproductores.

Un reproductor puede proveer información sobre sus parámetros actuales, incluyendo velocidad de reproducción, tiempo transcurrido y duración. Para obtener la velocidad se llama a `getRate` quien retorna la velocidad como un número flotante, para obtener el tiempo se llama a `getMediaTime` la cual retorna un objeto `Time`, para obtener la duración del clip se llama a `getDuration`.

Para responder a eventos durante la reproducción se debe implementar una interface llamada `ControllerListener`, la cual debe ser implementada en una clase y luego registrar

esta clase llamando `addControllerListener` sobre el controlador del cual se quiere recibir los eventos.

Para sincronizar la reproducción de múltiples flujos estos se asocian con una misma base de tiempo, para esto se usa `getTimeBase` y `setTimeBase`, por ejemplo se puede setear *player1* usando la base de tiempo de *player2*[2]*:*

```
player1.setTimeBase(player2.getTimeBase());
```

5. Procesando medios basados en el tiempo con JMF

Un *Processor* puede ser usado como un reproductor programable que sirva para decodificar procesos, pero también puede ser usado para codificar y multiplexar los datos de medios capturados. Se puede controlar lo capturado por el *Processor* de diferentes maneras:

- Usar un `ProcesorModel` para construir un *Processor* que tenga ciertas características de entrada y salida.
- Usar el método `TrackControl setFormat` para especificar que formato de conversiones se realizan para cada pista.
- Usar el método `Processor setOutputContentDescriptor` para especificar el formato de múltiplexión de los datos de salida.
- Usar el método `TrackControl setCodecChain` para seleccionar el efecto o codec plug-ins que es usado por el *Processor*.
- Usar el método `TrackControl setRenderer` para seleccionar el revender plug-ins usado por el *Processor*.

Configurando el *Processor*

Seleccionando las opciones de procesamiento de las pistas:

Para seleccionar que plug-ins es usado para cada pista en el flujo de medios se debe hacer lo siguiente:

1. Llamar el método `PlugInManager.getPlugInList` para determinar que plug-ins esta disponible. El `PlugInManager` retorna una lista de plug-ins que tienen la especificación de los formatos de entrada y salida y el tipo de plug-ins.

2. Llamar `getTrackControls` sobre el *Processor* para hacer un `TrackControl` para cada pista del flujo de medios.

3. Llamar los metodos `TrackControl setCodecChain` o `setRenderer` para especificar el plug-ins que se desea usar para cada pista.

Cuando se usa `setCodecChain` para especificar el codec y efecto plug-ins para un *Processor*, el orden en que el plug-ins aparece realmente en la cadena de procesamiento está determinado por los formatos de entrada y salida de cada plug-ins cargado.

Convirtiendo datos de medios de un formato a otro

Se puede seleccionar el formato para una pista en particular a través del `TrackControl` para cada pista:

1. Llamar `getTrackControls` sobre el *Processor* para hacer un `TrackControl` para cada pista en el flujo de medios. El *Processor* debe estar en el estado *Configurado* antes de llamar `getTrackControls.`
2. Usar el método `TrackControl setFormat` para especificar el formato a que se quiere convertir la pista seleccionada.

Especificando el formato de salida

Se puede usar el método `Processor setContentDescriptor` para especificar el formato de salida de los datos para el *Processor*. Se puede tener una lista de los formatos soportados llamando `getSupportedContentDescriptors`.

Se puede también seleccionar el formato de salida usando un `Processor-Model` para crear el *Processor*.

Especificando el destino

Se puede especificar un destino para el flujo de medios seleccionando un particular `Renderer` para una pista a través de `TrackControl`, o usando la salida del *Processor* como entrada a una particular `DataSink`. También se puede usar la salida del *Processor* como entrada a otro reproductor o a un *Processor* con distinto destino.

Seleccionando un Redender

Para seleccionar el *Revender* que se desea usar se debe hacer lo siguiente:

1. Llamar `getTrackControls` sobre el *Processor* para hacer un `TrackControl` para cada pista del flujo de medios. El *Processor* debe estar en estado *Configurado* antes de llamar `getTrackControls.`
2. Llamar el método `TrackControl setRenderer` para especificar el *Renderer* plug-ins.

Escribiendo datos de medios en un archivo

Se puede usar un `DataSink` para leer datos desde el objeto de salida del *Processor* `DataSource` y enviar los datos a un archivo.

1. Reciba la salida `DataSource` desde el *Processor* llamando `getDataOutput`.
2. Construya un archivo escritor `DataSink` llamando `Manager.createDataSink`. Pongalo en la salida `DataSource` y un `MediaLocator` que especifican la ubicación del archivo al que usted quiere escribir.
3. Llame `open` sobre el `DataSink` para abrir el archivo.
4. Llame `start` sobre el `DataSink` para comenzar a escribir los datos.

El formato del dato escrito para el archivo especificado es controlado a través del *Processor*.

6. Capturando datos de medios con JMF

Se puede usar JMF para capturar datos de medios desde un dispositivo de captura como un micrófono o una cámara de video. Los archivos capturados pueden ser procesados y almacenados para un uso futuro[2].

Para capturar datos de medios se debe hacer lo siguiente:

1. Localizar el dispositivo de captura que se quiera usar consultando el `CaptureDeviceManager`.
2. Hacer un objeto `CaptureDeviceInfo` para el dispositivo.
3. Hacer un `MediaLocator` desde el objeto `CaptureDeviceInfo` y usar este para crear un `DataSource`
4. Crear un reproductor o *Processor* usando el `DataSource`.
5. Iniciar el reproductor o *Processor* para empezar el proceso de captura.

Accediendo a dispositivos de captura

Para acceder a dispositivos de captura se debe hacer a través del `CaptureDeviceManager`. El `CaptureDeviceManager` es un registro central para todos los dispositivos de captura disponibles para JMF. Se puede hacer una lista con los dispositivos disponibles llamando al método `CaptureDeviceManager.getDeviceList`.

Almacenando datos de medios

Para almacenar los datos capturados en un archivo, se necesita usar un *Processor* en lugar de un reproductor. Se usa un `DataSink` para leer los datos desde la salida del *Processor* y enviarlos al archivo.

1. Hacer la salida `DataSource` desde el *Processor* llamando `getDataOutput`.
2. Construir un archivo escritor `DataSink` llamando `Manager.createDataSink`.
3. Construya un archivo escritor `DataSink` llamando `Manager.createDataSink`. Pongalo en la salida `DataSource` y un `MediaLocator` que especifican la ubicación del archivo al que usted quiere escribir.
4. Llame `open` sobre el `DataSink` para abrir el archivo.
5. Llame `start` sobre el `DataSink` para comenzar a escribir los datos.
6. Espere por un `EndOfMediaEvent`, un tiempo en particular, o un evento.
7. Llame stop en el *Processor* para finalizar la captura.
8. Llame `close` en el *Processor*.

   Cuando el *Processor* esta cerrado y el `DataSink` anuncia un `EndOfStreamEvent`, llame `close` sobre el `DataSink`.

[1] Estos puntos se tratarán con mas detalle en el apéndice A del anexo.
[2] Se presentarán algunos ejemplos de esta sección en el apéndice B del anexo.