JConferencia: Audioconferencia en Java

Redes Globales

Gerardo Parra Juan de la Cruz

Prof. Dr. Vicente González Ruiz

Índice

1. Introducción **1**

2. Algunos conceptos de JMF **2**

3. Arquitectura de JConferencia **3**
3.1. DataSource 3
3.2. Processor 4
3.3. DataSink 5
3.4. Player 5

4. El programa JConferencia **5**

1. Introducción

En el presente documento, vamos a ver cómo se ha implementado y cómo funciona un sistema de audioconferencia creado con el lenguaje de programación Java y con una expansión del mismo, creado por la empresa creadora de Java, SUN, llamado Java Media Framework (a partir de ahora, JMF).

El porqué de utilizar el lenguaje Java en lugar de C o C++: su sencillez. El lenguaje Java presenta una abstracción bastante mayor con respecto a C, por lo tanto, nos evita tener que tratar a un nivel mas bajo con algunas de las características del sistema, como puede ser la tarjeta de sonido.

Así, contrariamente a lo que pueda pasar con C, nos olvidamos de cómo hacer que suene nuestra tarjeta de sonido, dejándolo todo en manos del paquete JMF. Esto nos evita tener que reinventar la rueda, de modo que podamos ir más allá en su programación, sin tener que perder tiempo y esfuerzo en programar algo que ya está programado.

Además, una de las ventajas de trabajar con Java es la cualidad el mismo de ser independiente de la plataforma en la que se ejecute o compile. De este modo, igual podemos utilizar nuestro sistema de audioconferencia en la plataforma Windows, como en GNU/Linux o en

MacOS. Esto es en teoría. En la práctica, he podido experimentar que, aunque SUN distribuye el JMF para GNU/Linux, además de para Windows, en GNU/Linux no se ha podido hacer que funcione. Sin embargo, en Windows si, por lo tanto, la ejecución de este programa queda limitada a las plataformas Windows.

2. Algunos conceptos de JMF

Java Media Framework (JMF), proporciona una arquitectura unificada y un protocolo de mensajes para la gestión de la adquisición, procesado y envio de datos multimedia. JMF está diseñado para soportar la mayoría de los tipos de contenidos multimedia estandar, tales como AIFF, AU, AVI, GSM, MIDI, MPEG, QuickTime, RMF y WAV.

De este modo, JMF nos proporciona una serie de clases que nos facilitan el acceso a la tarjeta de sonido, tanto para la reproducción como la captura. Además, nos permite también la captura y reproducción de vídeo, pudiéndose hacer así un sistema de videoconferencia, el cual queda fuera de los objetivos que se persiguen en esta práctica.

Vamos a presentar algunos conceptos necesarios para poder trabajar con el JMF. Para empezar, tenemos lo que se llama la clase **Player**, la cual va a ser la encargada de reproducir los medios de audio y vídeo. Para nuestro caso, será la clase encargada de reproducir todo lo que nos llegue de nuestro interlocutor.

Por otro lado tenemos la clase **Processor**. Esta clase es un procesador que tendrá como misión tratar el sonido, convirtiéndolo en algún formato de audio. Es aquí donde se aplicará el correspondiente *codec* para hacer que la transmisión sea lo más fluida posible, ya que comprimirá el sonido en algún formato que ocupe menos que el capturado por el micrófono.

Otra clase importante es la llamada **DataSource** que va a ser la encargada de capturar todo lo que entre por nuestro micrófono. De igual manera, podría capturar por una webcam o un archivo de audio que tuviéramos en el disco duro, como puede ser cualquier MP3. El DataSource será la entrada de un Processor o un Player.

Por último, tenemos la clase **DataSink**, la cual sirve para dirigir la salida de un Processor o un Player hacia la red o al disco duro, o a otro Processor u otro Player.

Podemos ver esto de otra forma. Por ejemplo, los reproductores de casette o los reproductores de video domésticos, proporcionan un modelo de grabación, procesado y presentación de multimedia. Cuando reproducimos una pelicula usando un video, proporcionamos un flujo de datos multimedia al video, insertando una cinta de video. EL video lee e interpreta los datos de la cinta y envia la señal apropiada para la televisión y los altavoces.

JMF usa el mismo modelo básico. Una fuente de datos encapsula el flujo de datos como una cinta de video y un reproductor propociona el procesado y el mecanismo de control similar al del video. La reproducción y la captura de audio y video con JMF requiere de las entradas y las salidas apropiadas tales como micrófonos, cámaras, altavoces y monitores. (Ver Figura 1)

Las fuentes de datos y los reproductores son partes integrantes de la API de alto nivel de JMF para la manipulación de la captura, la presentación y el procesado de los datos multimedia. JMF también proporciona una API de más bajo nivel que soporta la integración de componentes de procesado personalizados y extensiones.

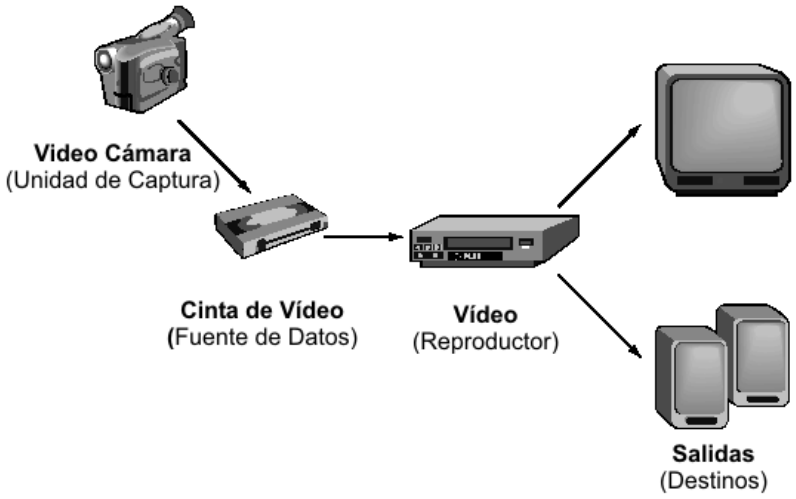

Figura 1: Comparación entre JMF y un sistema de vídeo doméstico

3. Arquitectura de JConferencia

El sistema de audioconferencia que se ha creado consta de un programa que es a la vez cliente y servidor. Servidor porque será quien envie datos hacia otra sesión de este mismo programa en cualquier otro ordenador y cliente porque recibirá aquellos datos que envie el otro programa. De este modo, podrán intercambiar datos, que en nuestro caso serán sonidos, a través de la red.

Por tanto, necesitamos un sistema de captura por el micrófono. Luego tendremos que procesar los datos y enviarlos. Mientras se hace esto, también deberemos recibir los datos del otro programa desde la red, procesarlos y mostrarlos al usuario, es decir, reproducir el sonido que nos viene desde el otro ordenador al que estamos conectados.

Entrando más en detalle, vamos a ver cómo creamos cada uno de los componentes que hemos citado arriba.

3.1. DataSource

Para comenzar, debemos capturar el audio que entre por el micrófono. Para ello, debemos crear un objeto de la clase *DataSource*. La clase DataSource será la que determine la entrada del Player o del Processor. Así, la configuraremos para que tome los datos que entren por el microfono. Para ello, debemos obtener una lista de los dispositivos de captura de audio que tiene el sistema. Para ello, determinamos que queremos un medio de captura que sea para audio, del tipo Linear, con una tasa de bits de 8000, que cada muestra sea representada por 8 bits y en formato monoaural (es decir, con un solo canal). Así, mediante la siguiente linea, obtenemos un listado de los dispositivos que cumplen estos requisitos:

```
Vector deviceList = CaptureDeviceManager.getDeviceList(new
AudioFormat(AudioFormat.LINEAR,8000,8,1));}
```

donde CaptureDeviceManager.getDeviceList obtiene una lista de dispositivos de captura que cumplen los requisitos especificados en el constructor del objeto de la clase AudioFormat.

Así que tomamos el primer elemento, que siempre será (en caso de un sistema Windows) el sistema de captura *DirectSoundCapture*. Ahora, solo nos queda asociar este dispositivo con

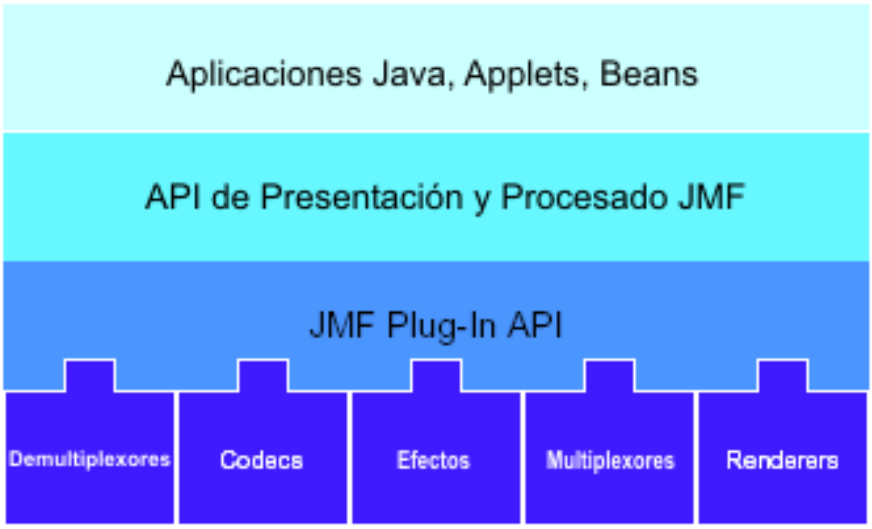

Figura 2: Arquitectura de Capas de JMF

un objeto de la clase DataSource tal y como sigue:

```
DataSource source = Manager.createDataSource(di.getLocator());
```

donde di.getLocator() devuelve el dispositivo antes obtenido y es el método `Manager.createDataSource()` el que asocie este dispositivo con el DataSource.

3.2. Processor

Una vez que podemos capturar, debemos tratar el audio para que su transmisión por la red sea lo más fluida posible. Para ello, debemos sacrificar un poco la calidad y aplicar algún *codec*. Así, como se ha visto en el apartado anterior, vamos a trabajar con sonido que está capturado con 8000 muestras de 8 bits cada una. Aquí ya hemos perdido calidad, pues la mejor calidad para que el sonido sea de calidad excelente (como es la calidad del sonido de un CD) requiere 44100 muestras de 16 bits cada una y en sonido estereo. Pero como vamos a transmitir voz y este tipo de comunicación no requiere una calidad de CD, podemos rebajarla hasta adquirir una calidad parecida al teléfono (aunque la que conseguimos es superior).

Así, vamos a codificar el sonido para ganar aun más en velocidad de transmisión, ya que reducimos el tamaño de los paquetes que se deben enviar. Para esto, se han probado una serie de codecs que implementa JMF y se ha elegido uno que ofrece una calidad bastante aceptable y una tasa de bits tambien aceptable. Así, mientras formatos como U_LAW o MPEG conseguian una tasa de bits alrededor de los 60 kbits/seg, con el formato elegido, que es G723 conseguimos alrededor de 6 kbits/seg, por tanto, en cada segundo tenemos que enviar casi 10 veces menos bits. A pesar de la codificación, la calidad sigue siendo aceptable y presenta los parámetros que se definen en la siguiente linea que hemos utilizado para especificar dichas características de codificación:

```
private static final Format[] FORMATS = new Format[] {
    new AudioFormat(AudioFormat.G723_RTP,8000,16,1)};
```

Una vez que tenemos esto, tenemos que crear el Processor que será el que aplique esta codificación al sonido capturado. Para eso, le damos como argumentos en la creación de dicho Processor el DataSource obtenido anteriormente y el formato en el que vamos a convertir esos datos:

```
processor = Manager.createRealizedProcessor(new ProcessorModel
(ds,FORMATS,CONTENT_DESCRIPTOR));
```

La constante CONTENT_DESCRIPTOR especifica el formato en que vamos a transmitir los datos, que es utilizando el protocolo RTP (Real-Time Protocol, Protocolo en Tiempo Real).

3.3. DataSink

Ya tenemos los datos capturados y codificados. Ahora necesitamos que salgan de nuestro ordenador. Para eso, utilizamos un objeto de la clase DataSink que será el encargado de enviar por la red estos datos.

Lo único que tenemos que hacer es llamar a un método de la clase Manager (al igual que hacíamos con el DataSource) para asignarle una dirección donde enviar los datos y asociarle los propios datos. Todo esto se hace en la siguiente línea:

```
dataSink =
Manager.createDataSink(processor.getDataOutput(),destino);
```

donde processor.getDataOutput() devuelve los datos de salida del procesador (el sonido ya codificado) y destino guarda la dirección de destino a donde tiene que enviar los datos.

Para que comience la captura, codificación y transmisión de los datos, debemos llamar al método dataSink.start() el cual creará un hilo que haga todo este trabajo.

La dirección de destino llevará la forma rtp://x.y.z.w:puerto/audio de modo que especificamos la dirección IP, el puerto al que enviaremos los datos y el tipo de datos que son.

3.4. Player

La siguiente parte que analizamos es el Player, ya que mientras estamos enviando datos, tambien los estamos recibiendo (ya que es una comunicación full-duplex). Así, crearemos un Player al que le diremos qué datos tiene que reproducir. Obviamente, los datos que debe reproducir son los provenientes de su otro interlocutor. Para ello crearemos el Player dándole la dirección de donde vienen esos datos.

```
player = Manager.createRealizedPlayer(origen);
```

Así, para que comience a reproducir los datos, solo habrá que llamar al método player.start() el cual se encargará de capturar los datos de la dirección especificada por la variable origen, los decodificará y los reproducirá.

Los datos de origen, los toma desde el puerto local especificado en el código (en este caso se ha utilizado el puerto 49150). Por tanto, nosotros colocamos los datos en el puerto del otro interlocutor y tomamos los que nos envia del mismo puerto pero en nuestro ordenador.

Para obtener nuestra IP se ha utilizado un método que proporciona Java en uno de sus paquetes.

4. El programa JConferencia

Cuando arrancamos el programa JConferencia, ya sea mediante el archivo JAR, utilizando la línea de comandos:

java -jar JConferencia.jar

o haciendo doble click en el archivo ejecutable para Windows se nos presenta la pantalla que vemos en la figura 3.

Como vemos, tenemos un cuadro de texto en el que introducimos la dirección IP en el formato x.y.z.w sin especificar puerto ni tipo de datos, ya que eso lo hemos dejado en manos

Figura 3: Pantalla que muestra el programa JConferencia

del programador, no del usuario. Así, basta con pulsar el botón Conectar para conectarnos al otro ordenador a la espera de que él tambien ejecute el programa, introduzca nuestra IP y pulse el botón.

Una vez que ocurre esto, se produce una comunicación fluida y con un retraso aceptable para poder llevar una conversación sin ningun problema.

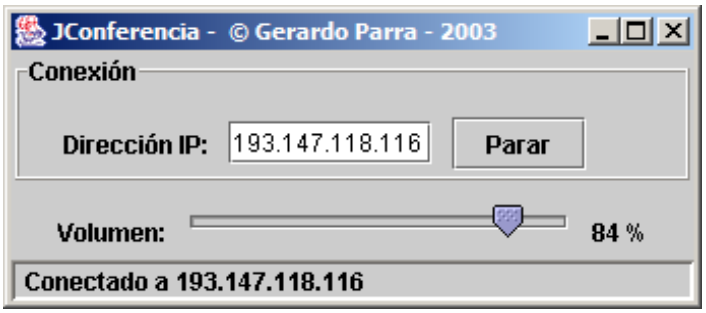

Figura 4: JConferencia conectado a otro ordenador en plena comunicación

Como podemos ver en la figura 3 y 4 también disponemos de una barra deslizante que nos permite controlar el volumen al que oímos lo que nos llega de la red.

Para terminar la comunicación, basta con pulsar el boton Conectar, ahora llamado Parar. Así, podemos especificar una nueva IP y comenzar una nueva comunicación.