Extendiendo la JMF mediante Plug-ins

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

Departamento de Ingeniería Telemática
Universidad Carlos III de Madrid

2 Índice

- Formas de extender la JMF
- El Demultiplexer Plug-In
  - Ejemplo
- Codec Plug-In
- Effect Plug-In
  - Ejemplo
- Multiplexer Plug-In
- Renderer Plug-In
- Registro de un plug-in en el PlugInManager
  - Ejemplo

3 Formas de extender la JMF

- Las formas básicas de extensión de la JMF son:
  - A bajo nivel:
    - Implementación de los interfaces de plug-in para el procesamiento a medida de las diferentes pistas en los procesadores
  - A alto nivel:
    - Implementación de nuevos **DataSources** y **DataSinks**
    - Implementación de nuevos **MediaHandlers** (Processors y Players).
- Nosotros nos vamos a centrar en este tema en los Plug-ins.
- Recordemos que hay 5 tipos de Plug-ins:
  - Demultiplexores
  - Codificadores
  - Procesadores de efectos
  - Multiplexores
  - "Renderers" o presentadores de medios

4 El Demultiplexer Plug-In

- Un **demultiplexor analiza una fuente** (DataSource) de medios con tipo de contenido como WAV, MPEG o QuickTime. Si los medios están multiplexados se separan las diferentes pistas (tracks). Se puede implementar un "Demultiplexer plug-in" para soportar un nuevo formato de fichero de medios o para realizar un nuevo demultiplexor de altas prestaciones. Si implementaramos un DataSource propio prodríamos implementar un "Demultiplexer plug-in" que trabajara con nuestro DataSource para permitir su reproducción a través de un Processor existente.
- Un Demultiplexer es un componente de procesamiento tipo "**single-input, multi-output**". Lee datos de un DataSource, extrae los tracks y los saca a la salida de forma separada.
- Un Demultiplexer es un tipo de MediaHandler y debe implementar el método MediaHandler.**setSource**. Este método es usado por un Processor para localizar un Demultiplexer que pueda manejar un tipo de DataSource. El Processor recorre la lista de Demultiplexers registrados hasta que encuentra uno que no retorna una excepción cuando se llama a setSource.
- El principal trabajo que realiza un Demultiplexer se hace en el método **getTracks**, que devuelve un array de tracks que se extraen del DataSource de entrada.

5 Implementación de un Demultiplexer

- La interfaz Demultiplexer:
  - public interface Demultiplexer extends PlugIn, MediaHandler, Duration
- Para implementar la interfaz Demultiplexer se requiere básicamente:
  - Implementar getSupportedInputContentDescriptors para devolver los formatos de entrada que soporta el demultiplexor. Por ejemplo, un demultiplexor GSM necesita avisar que soporta ficheros GSM.
  - Implementar el método MediaHandler.setSource para verificar el DataSource y determinar si el Demultiplexer puede manejar un tipo de fuente de medios (si no se lanza una IncompatibleSourceException).
  - Implementar getTracks para extraer los tracks individuales del flujo de medios.
  - Otros métodos: start, stop…

6 Ejemplo Demultiplexer

```
private static ContentDescriptor[] supportedFormat =
    new ContentDescriptor[] {new ContentDescriptor("audio.x_gsm")};

public ContentDescriptor [] getSupportedInputContentDescriptors() {
    return supportedFormat;
}
```

7 Ejemplo Demultiplexer (cont)

```
    public void setSource(DataSource source)
        throws IOException, IncompatibleSourceException {

        if (!(source instanceof PullDataSource)) {
            throw new IncompatibleSourceException("DataSource
               not supported: " + source);
        } else {
            streams = ((PullDataSource) source).getStreams();
        }


        if ( streams == null) {
            throw new IOException("Got a null stream from the DataSource");
        }

        if (streams.length == 0) {
            throw new IOException("Got a empty stream array
               from the DataSource");
        }

        this.source = source;
        this.streams = streams;

        positionable =  (streams[0] instanceof Seekable);
        seekable =  positionable && ((Seekable)
                  streams[0]).isRandomAccess();

        if (!supports(streams))
            throw new IncompatibleSourceException("DataSource not
                  supported: " + source);
    }
```


8 Ejemplo Demultiplexer (cont)

```
    public Track[] getTracks() throws IOException, BadHeaderException {

        if (tracks[0] != null)
            return tracks;
        stream = (PullSourceStream) streams[0];
        readHeader();
        bufferSize = bytesPerSecond;
        tracks[0] = new GsmTrack((AudioFormat) format,
                                 /*enabled=*/ true,
                                  new Time(0),
                                  numBuffers,
                                  bufferSize,
                                  minLocation,
                                  maxLocation
                                  );
        return tracks;
    }

// ...

    private void readHeader()
        throws IOException, BadHeaderException {

        minLocation = getLocation(stream); // Should be zero

        long contentLength = stream.getContentLength();
        if ( contentLength != SourceStream.LENGTH_UNKNOWN ) {
            double durationSeconds = contentLength / bytesPerSecond;
            duration = new Time(durationSeconds);
            maxLocation = contentLength;
        } else {
            maxLocation = Long.MAX_VALUE;
        }
```

9 Codec Plug-In

- Los "Codec plug-ins" se usan para **manejar formatos de datos comprimidos, convertir el formato de datos, o codificar datos "en crudo" a un formato comprimido**. Podemos implementar la interfaz Codec por temas de rendimiento, para soportar nuevos formatos de compresión o para convertir datos de un formato propietario a un formato estándar.
- Un Codec es un componente de procesamiento "**single-input, single-output**". Lee datos de pistas individuales, procesa los datos y los saca a la salida.
- Un "Codec plug-in" puede permitir el control externo del proceso a través de los controles de tipo EncodingControl o DecodingControl. Estos controles proporcionan un mecanismo para ajustar atributos como la tasa de marcos, la tasa de bit y el ratio de compresión. Los controles se obtienen a través del método getControls. Si un control ofrece interfaz visual, ésta se puede obtener invocando el método getControlComponent.
- Para implementar la interfaz Codec se necesita:
  - Implementar **getSupportedInputFormats** y **getSupportedOutputFormats** para indicar los formatos de entrada y salida que se soportan.
  - Permitir la selección de estos formatos mediante la implementación del método **setInputFormat** y **setOutputFormat**.
  - Implementar el método **process** para realizar la compresión o descompresión del Track de entrada.

10 Effect Plug-In

- Un "Effect plug-in" es en realidad un tipo de Codec que realiza un tipo de procesamiento sobre el Track de entrada distinto al de compresión o descompresión. Por ejemplo se puede implementar un control de volumen sobre la pista. Como el Codec, un Effect es "**single-input, single-output**".
- Un "Effect plug-in" puede usarse como pre-procesado o post-procesado. Por ejemplo, si un Processor se está usando para "renderizar" un flujo de medios el efecto será típicamente de post-procesado, pero si el Processor para guardar un medio comprimido el efecto será típicamente de pre-procesado.
- Para implementar la interfaz Effect se necesita:
  - Implementar **getSupportedInputFormats** y **getSupportedOutputFormats** para indicar los formatos de entrada y salida que se soportan.
  - Permitir la selección de estos formatos mediante la implementación del método **setInputFormat** y **setOutputFormat**.
  - Implementar el método **process** para añadir el efecto al Track de entrada.

11 Ejemplo de implementación del plug-in de efectos

```
import javax.media.*;
import javax.media.format.*;
import javax.media.format.audio.*;

public class GainEffect implements Effect {

    /** The effect name **/
    private static String EffectName="GainEffect";

    /** chosen input Format **/
    protected AudioFormat inputFormat;

    /** chosen output Format **/
    protected AudioFormat outputFormat;

    /** supported input Formats **/
    protected Format[] supportedInputFormats=new Format[0];

    /** supported output Formats **/
    protected Format[] supportedOutputFormats=new Format[0];

    /** selected Gain **/
    protected float gain = 2.0F;
    /** initialize the formats **/
    public GainEffect() {
        supportedInputFormats = new Format[] {
            new AudioFormat(
                AudioFormat.LINEAR,
                Format.NOT_SPECIFIED,
                16,
                Format.NOT_SPECIFIED,
                AudioFormat.LITTLE_ENDIAN,
                AudioFormat.SIGNED,
                16,
                Format.NOT_SPECIFIED,
                Format.byteArray
            )
        };
```


12 Ejemplo de implementación del plug-in de efectos

```
        supportedOutputFormats = new Format[] {
            new AudioFormat(
                AudioFormat.LINEAR,
                Format.NOT_SPECIFIED,
                16,
                Format.NOT_SPECIFIED,
                AudioFormat.LITTLE_ENDIAN,
                AudioFormat.SIGNED,
                16,
                Format.NOT_SPECIFIED,
                Format.byteArray
            )
        };


    }
    /** get the resources needed by this effect **/
    public void open() throws ResourceUnavailableException {
    }

    /** free the resources allocated by this codec **/
    public void close() {
    }

    /** reset the codec **/
    public void reset() {
    }

    /** no controls for this simple effect **/
    public Object[] getControls() {
        return (Object[]) new Control[0];
    }
```

13 Ejemplo de implementación del plug-in de efectos

```
/**
 * Return the control based on a control type for the effect.
 **/
public Object getControl(String controlType) {
    try {
        Class cls = Class.forName(controlType);
        Object cs[] = getControls();
        for (int i = 0; i < cs.length; i++) {
            if (cls.isInstance(cs[i]))
            return cs[i];
        }
        return null;
    } catch (Exception e) { // no such controlType or such control
        return null;
    }
}
/************** format methods *************/
/** set the input format **/
public Format setInputFormat(Format input) {
    // the following code assumes valid Format
    inputFormat = (AudioFormat)input;
    return (Format)inputFormat;
}
/** set the output format **/
public Format setOutputFormat(Format output) {
    // the following code assumes valid Format
    outputFormat = (AudioFormat)output;
    return (Format)outputFormat;
}
/** get the input format **/
protected Format getInputFormat() {
    return inputFormat;
}
```


14 Ejemplo de implementación del plug-in de efectos

```
/** get the output format **/
protected Format getOutputFormat() {
    return outputFormat;
}

/** supported input formats **/
public Format [] getSupportedInputFormats() {
    return supportedInputFormats;
}

/** output Formats for the selected input format **/
public Format [] getSupportedOutputFormats(Format in) {
    if (! (in instanceof AudioFormat) )
        return new Format[0];

    AudioFormat iaf=(AudioFormat) in;

    if (!iaf.matches(supportedInputFormats[0]))
        return new Format[0];

    AudioFormat oaf= new AudioFormat(
            AudioFormat.LINEAR,
            iaf.getSampleRate(),
            16,
            iaf.getChannels(),
            AudioFormat.LITTLE_ENDIAN,
            AudioFormat.SIGNED,
            16,
            Format.NOT_SPECIFIED,
            Format.byteArray
    );

    return new Format[] {oaf};
}
```

15 Ejemplo de implementación del plug-in de efectos

```
    /** do the processing **/
    public int process(Buffer inputBuffer, Buffer outputBuffer){

        // == prolog
        byte[] inData = (byte[])inputBuffer.getData();
        int inLength = inputBuffer.getLength();
        int inOffset = inputBuffer.getOffset();


        byte[] outData = validateByteArraySize(outputBuffer, inLength);
        int outOffset = outputBuffer.getOffset();

        int samplesNumber = inLength / 2 ;

        // == main

        for (int i=0; i< samplesNumber;i++) {
            int tempL = inData[inOffset ++];
            int tempH = inData[inOffset ++];
            int sample = tempH | (tempL & 255);

            sample = (int)(sample * gain);

            if (sample>32767) // saturate
                sample = 32767;
            else if (sample < -32768)
                sample = -32768;

            outData[outOffset ++]=(byte) (sample &  255);
            outData[outOffset ++]=(byte) (sample >> 8);

        }

        // == epilog
        updateOutput(outputBuffer,outputFormat, samplesNumber, 0);
        return BUFFER_PROCESSED_OK;
    }
```


16 Ejemplo de implementación del plug-in de efectos

```
    /**
     * Utility: validate that the Buffer object's data size is at least
     * newSize bytes.
     * @return array with sufficient capacity
     **/
    protected byte[] validateByteArraySize(Buffer buffer,int newSize) {
        Object objectArray=buffer.getData();
        byte[] typedArray;
        if (objectArray instanceof byte[]) { // is correct type AND not null
            typedArray=(byte[])objectArray;
            if (typedArray.length >= newSize ) { // is sufficient capacity
                return typedArray;
            }
        }
        System.out.println(getClass().getName()+
                           " : allocating byte["+newSize+"] ");
        typedArray = new byte[newSize];
        buffer.setData(typedArray);
        return typedArray;
    }
    /** utility: update the output buffer fields **/
    protected void updateOutput(Buffer outputBuffer,
                                Format format,int length, int offset) {


        outputBuffer.setFormat(format);
        outputBuffer.setLength(length);
        outputBuffer.setOffset(offset);
    }
}
```

17 Multiplexer Plug-In

- Un **Multiplexer** es básicamente el opuesto del **Demultiplexer**: recibe como entrada una serie de tracks de medios y los combina en un único media-stream tal como un fichero MPEG o QuickTime. El multiplexado puede también hacerse en el DataSink.
- Un Multiplexer es un componente de procesamiento "**multi-input, single-output**". Lee datos de un conjunto de tracks y saca un DataSource.
- La principal tarea del Multiplexer se realiza en la implementación del método **process**. El método getDataOutput devuelve el DataSource generado por el Multiplexer.
- Para implementar un Multiplexer se necesita:
  - Implementar **getSupportedOutputContentDescriptors** para indicar los formatos de salida que soporta el multiplexor.
  - Habilitar la selección de formato de salida mediante la implementación del método **setContentDescriptor**.
  - Implementar el método **process** para realizar el trabajo de unir los diferentes **tracks** en un único flujo de salida con el formato deseado.

18 Renderer Plug-In

- Un Renderer entrega los datos de los medios para su visualización-audición (estado final de procesamiento). Es un componente "**single-input, no output**". Los renderizadores leen los datos de un DataSource y típicamente presentan los datos al usuario (no es lo normal pero podrían usarse para entregar los datos a otro tipo de dispositivo).
- Para un renderizador de vídeo se debe implementar la interfaz VideoRenderer que extiende a Renderer para definir atributos específicos del vídeo tales como el Component (AWT) donde se renderiza el vídeo.
- El principal trabajo de un Renderer se realiza en la implementación del método process. Cuando se implementa un Renderer se necesita:
  - Implementar **getSupportedInputFormats** para indicar los formatos de entrada que soporta el **Renderer**.
  - Habilitar la selección de formato de entrada mediante la implementación del método **setInputFormat**.
  - Implementar el método **process** para realizar el trabajo de renderizar los datos del medio al dispositivo de salida.

19 Registro de un plug-in en el PlugInManager

- Para asegurarse que un plug-in está disponible para un Processor a través de la interfaz TrackControl se necesita registrarlo en el **PlugInManager**. (Los plug-ins por defecto ya están registrados automáticamente.)
- Para registrar un nuevo plug-in se usa el método **PlugInManager.addPlugIn**. Se debe llamar también al método **commit** para hacer que el registro sea permanente.
- Para desregistrar un plug-in se hace mediante la invocación del método **removePlugIn**

20 Ejemplo: registro de un nuevo Plug-in

```
// Name of the new plugin
string GainPlugin = new String("COM.mybiz.media.GainEffect");

// Supported input Formats
Format[] supportedInputFormats = new Format[] {
            new AudioFormat(
                AudioFormat.LINEAR,
                Format.NOT_SPECIFIED,
                16,
                Format.NOT_SPECIFIED,
                AudioFormat.LITTLE_ENDIAN,
                AudioFormat.SIGNED,
                16,
                Format.NOT_SPECIFIED,
                Format.byteArray
            )
};

// Supported output Formats
Format[] supportedOutputFormats = new Format[] {
            new AudioFormat(
                AudioFormat.LINEAR,
                Format.NOT_SPECIFIED,
                16,
                Format.NOT_SPECIFIED,
                AudioFormat.LITTLE_ENDIAN,
                AudioFormat.SIGNED,
                16,
                Format.NOT_SPECIFIED,
                Format.byteArray
            )
};

// Add the new plug-in to the plug-in registry
PlugInManager.addPlugIn(GainPlugin, supportedInputFormats,
                        supportedOutputFormats, EFFECT);

// Save the changes to the plug-in registry
PlugInManager.commit();
```

21 Cuestiones de repaso

- Indique cuales son las dos formas específicas que hay para extender JMF tanto a alto como a bajo nivel.
- ¿Qué cinco tipos de plug-ins existen en JMF?
- Indique cual de los plug-ins de JMF es del siguiente tipo: "uno-a-muchos".
- ¿Cuál es la función específica de un demultiplexor?
- Desde el punto de vista de la implementación de un demultiplexor, ¿Cuáles son sus dos principales métodos?
- Dibuje una clase demultiplexora sencilla, ¿ qué elementos implementa?
- ¿Cuál es la principal diferencia entre un Codec y un Efecto?
- ¿Son el codec y el efecto componentes de tipo single-input single-output?
- ¿Qué métodos hay que implementar para que funcione un plugin?
- ¿Qué tipo de procesamiento realiza un Multiplexer?
- ¿Qué tipo de procesamiento realiza un Render?
- ¿Cómo se manipulan los plugins? ¿Cómo se agregan y retiran de Plug-in manager?

22 Autoría

- Mario Muñoz Organero
- Pablo Basanta Val

  + (preguntas de refuerzo)