



UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

Curso 2013-2014

**PROCESADO DE SEÑALES
PARA ELIMINAR LA VOZ
ORIGINAL Y SUSTITUIRLA
POR VOZ GRABADA EN
KARAOKE**

Alumno: Pancorbo Rubio, Daniel.

Tutor: Vera Candeas, Pedro.

Departamento: Ingeniería de Telecomunicación.

Junio, 2014

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

Curso 2013-2014

**PROCESADO DE SEÑALES
PARA ELIMINAR LA VOZ
ORIGINAL Y SUSTITUIRLA
POR VOZ GRABADA EN
KARAOKE**

Alumno: Pancorbo Rubio, Daniel

Tutor: Vera Candeas, Pedro

Dept.: Ingeniería de Telecomunicación

Visto Bueno a la defensa del TFG

Firma del autor/a



PR

Firma del Tutor/a

NOMBRE
VERA
CANDEAS
PEDRO - NIF
26228400M

Digitally signed by
NOMBRE VERA CANDEAS
PEDRO - NIF 26228400M
DN: c=ES, o=FNMT,
ou=FNMT Clase 2 CA,
ou=501072765,
cn=NOMBRE VERA
CANDEAS PEDRO - NIF
26228400M
Date: 2014.06.20
12:07:35 +02'00'

A mi familia, amigos y pareja por estar siempre a mi lado y apoyarme en los momentos difíciles.

Índice General

I Memoria	8
1. Introducción	9
1.1 Contexto.....	9
2. Resumen	11
2.1 Generalidades	11
2.2 Busqueda de contenidos e investigación	12
2.3 Desarrollo del algoritmo e integración con el servidor web.....	13
3. Objetivos.....	15
4. Antecedentes.....	18
4.1 Antecedentes del karaoke	18
5. Materiales y Métodos	21
5.1 Estado del arte del karaoke.....	21
5.1.1 Estado del arte del karaoke	21
5.1.2 Estado del arte de los algoritmos de separación de fuentes sonoras	23
5.1.3 Estado del arte de los servidores web.....	24
5.2 Desarrollo del TFG	26
5.2.1 Archivos enviados por el usuario	27
5.2.1.1 Archivo de texto	27
5.2.1.2 Archivo de audio	31

5.2.2 Desarrollo de una página web como interfaz de usuario.....	31
5.2.3 Implementación en JSP de Servlet Java	32
5.2.4 Comunicación entre Java y Matlab	36
5.2.5 Introducción al programa desarrollado.....	38
5.2.6 Integración de la información extraída del programa UltraStar	41
5.2.7 Desarrollo de modelos basados en el algoritmo NMF	42
5.2.7.1 Descripción general de los modelos implementados.....	42
5.2.7.2 Primer modelo basado en NMF, percusivo/armónico	45
5.2.7.3 Segundo modelo basado en NMF, vocal/instrumental	54
5.2.7.4 Reconstrucción de la señal.....	59
6. Resultados y Discusión	61
6.1 Evaluación y resultados	61
7. Conclusiones.....	66
7.1 Conclusiones generales y problemas encontrados.....	66
8. Lineas de futuro	69
II Pliego de condiciones	71
9. Pliego de condiciones	72
9.1 Requisitos Hardware.....	72
9.2 Requisitos Software	72
III Presupuesto	74
10. Presupuesto	75
10.1 Recursos humanos	75
10.2 Recursos materiales	75
10.3 Presupuesto final.....	76

IV Anexos	77
11. Manual de usuario	78
11.1 Guía de utilización del servicio	78
12. Manual técnico	82
12.1 Instalaciones previas.....	82
12.1.1 Instalación JDK 8	83
12.1.2 Instalación Apache Tomcat 7	84
12.1.3 Instalación codificador LAME	88
12.2 Documentación Técnica	89
V Referencias Bibliográficas	123

Índice de tablas

6.1 Base de datos usada para obtener los resultados	61
6.2 Medidas obtenidas	63
6.3 Comparación de resultados entre distintos algoritmos	64
10.1 Presupuesto recursos humanos	75
10.2 Presupuesto recursos materiales	75
10.3 Presupuesto final.....	76

Índice de figuras

5.1 Interfaz de usuario del software UltraStar.....	23
5.2 Fases del proceso	26
5.3 Procesos en el servidor web.....	35
5.4 Fases de Matlabcontrol.....	37
5.5 Primera parte, programa desarrollado en MATLAB.....	39
5.6 Segunda parte, programa desarrollado en MATLAB	40
5.7 Intercambio de información entre los modelos desarrollados	44
5.8 Ilustración de las restricciones de dispersión y suavidad	47
11.1 Sección Principal	78
11.2 Sección “Procesa tu canción”	79
11.3 Sección “Acerca de”	79
11.4 Página de descarga.....	80
12.1 Terminal Ubuntu.....	82
12.2 Instalación JDK	83
12.3 Página principal Apache Tomcat.....	86
12.4 Host Manager	87

PARTE I

MEMORIA

CAPÍTULO 1

Introducción

1.1 Contexto

Admitamos que a todos nos gusta cantar. Cantamos en el coche, en la ducha y mientras cocinamos, pero no hay nada como estar en un karaoke, tomar el micrófono y sacar al cantante que todos llevamos dentro. El karaoke es una forma popular de entretenimiento donde los cantantes no profesionales cantan con un acompañamiento musical, en la que los vocales del cantante original están ausentes.

En general, la música se distribuye de manera que todos los instrumentos y los vocales de los cantantes están mezclados en una pista monofónica o polifónica. Esto lleva a que su uso no sea recomendable para el Karaoke. Además, es imposible de proporcionar a los usuarios todas sus canciones favoritas, ya que la producción de las pistas instrumentales para karaoke es realizada en estudios profesionales, con el correspondiente coste en tiempo y dinero. Por este motivo, existen métodos que producen el acompañamiento instrumental directamente de las pistas ya mezcladas.

En la actualidad, existe una gran variedad de software relacionado con el Karaoke, desde videojuegos en las principales plataformas de videoconsolas, como SingStar para PlayStation 3 o Lips para Xbox 360, hasta programas de karaoke desarrollados para los distintos sistemas operativos para computadoras, como es el caso de UltraStar, software multiplataforma.

Es objeto de este Proyecto el desarrollo de un algoritmo que permita extraer el acompañamiento de las pistas polifónicas, cuya inclusión dentro de un servicio web harán posible ofrecer a los usuarios del programa UltraStar, una experiencia diferente para probar sus cualidades como cantantes sin perder la esencia del karaoke.

CAPÍTULO 1 – INTRODUCCIÓN

CAPÍTULO 2

Descripción

2.1 Generalidades

Este Trabajo Fin de Grado tiene como objetivo el desarrollo de un algoritmo capaz de realizar la separación de una canción ya producida en estudio, en las dos partes fundamentales que la componen, la parte vocal y la parte instrumental.

Asimismo, se ha extendido este fin hacia la realización de un servicio web que complementa a un determinado software de Karaoke, el programa UltraStar. En este software, los usuarios tienen la libertad de usar sus copias de canciones descargadas en el programa. Ellos mismos crean los diferentes archivos necesarios que acompañan a sus canciones para hacerlo posible, entre los que se incluye: un archivo de texto, el cual es fundamental, un posible video acorde con la canción, una portada, etc. Sin embargo, no disponen de las pistas instrumentales por separado que les proporcionarían una experiencia más cercana a la realidad.

Así pues, el resultado final de este TFG debe ser un servicio web a su disposición y accesible desde cualquier sistema, que realice esta tarea y que junto a sus canciones y sus archivos de texto, de los cuales se extraerá cierta información necesaria para procesar los archivos de audio, les permita obtener estas pistas para su posterior uso. Es imprescindible pensar en los formatos permitidos para su envío y posterior procesado por este. Se decidió utilizar los formatos más populares de texto y audio usados por los usuarios de UltraStar, MP3 o WAV como formatos de audio y TXT como formato de texto.

En las siguientes páginas de que consta este documento se supone que el lector dispone de unos conocimientos mínimos de programación orientada a objetos, en concreto Java, y de la programación en lenguaje matemático Matlab, así como del tratamiento de la señal.

2.2 Búsqueda de contenidos e investigación

Se ha realizado una búsqueda exhaustiva de información a través de internet de libros y documentos que aporten conocimientos y conceptos sobre los distintos algoritmos de separación de fuentes sonoras, que permitieran la realización de este TFG de forma exitosa.

Además, se han estudiado distintas formas de mejorar el algoritmo seleccionado

A continuación, se ha realizado un estudio que permitiese complementar el algoritmo seleccionado con la información procedente del programa UltraStar.

Asimismo, se han estudiado las distintas posibilidades de desarrollo del servidor web dentro del sistema operativo Linux, así mismo como la manera de extender las capacidades de interacción del servidor web con los clientes.

Posteriormente, se han investigado las distintas posibilidades de integración del algoritmo desarrollado con un servidor web, de forma que la creación del servicio web fuera posible.

También, se llevó a cabo la búsqueda de información de los distintos lenguajes de programación para la creación de una página web desde la que acceder al servicio, creando un entorno amigable e intuitivo para los usuarios del programa UltraStar.

Finalmente, tras la integración de todos estos elementos, se ha llegado a la creación del servicio web final, alcanzando la correcta ejecución del TFG.

2.3 Desarrollo del algoritmo e integración en el servidor web

Para el desarrollo de este TFG se ha utilizado el programa Eclipse Kleper (para Linux), con el cual se programarán los distintos archivos JSP implementados en este TFG, además de Matlab R2013a, siguiendo los siguientes pasos hasta su realización final:

- Análisis de las posibilidades dadas tras seleccionar el algoritmo de separación de fuentes sonoros para su posterior mejora usando la información del programa UltraStar.
- Implementación del algoritmo seleccionado con las respectivas mejoras dentro del entorno de programación Matlab.
- Selección del servidor web ha integrar en el sistema operativo Linux, así como la programación en Java del Servlet necesario para que el usuario pueda interaccionar con el servidor.
- Creación de una clase sencilla en Java que permita la conexión entre el Servlet y Matlab para procesar las peticiones de los usuarios.
- Desarrollo de una página web accesible desde cualquier rincón de internet, para acceder al servicio implementado

CAPÍTULO 2 – DESCRIPCIÓN

CAPÍTULO 3

OBJETIVOS

El objetivo principal de este TFG consiste en la separación de la voz del cantante original del acompañamiento instrumental. Para ello, es necesario conocer a priori los intervalos de la pista de audio donde podemos encontrar música instrumental o voz cantada más acompañamiento.

A continuación se enumeran una serie de objetivos secundarios que se pretenden alcanzar con el presente TFG. Así como nuevos objetivos que no eran objeto de este TFG, pero los cuales han proporcionado un desarrollo más extenso y enriquecedor del mismo, superando las fronteras establecidas para este.

- Conocer los fundamentos teóricos del procesado de señales, relacionados con este TFG
- Conocer los distintos métodos de separación de fuentes sonoras y selección de uno de ellos que permita su desarrollo.
- Programar el algoritmo seleccionado dentro del entorno de programación matemático Matlab.
- Complementar el algoritmo desarrollado con la información extraída del fichero de texto que acompaña a las canciones empleadas en el programa UltraStar, para su posterior uso con el mismo.
- Conocer el funcionamiento de los distintos software que implementan la función de servidor web así como las distintas posibilidades que ofrece.
- Comprensión más profunda de las distintas posibilidades del lenguaje de programación Java orientadas a su uso en servidores.

- Estudiar distintos métodos de conexión que permitan la comunicación entre el programa Matlab y el servidor web.
- Integrar de forma conjunta todos estos conocimientos para la correcta ejecución del TFG, permitiendo crear el servicio web deseado para los usuarios del programa UltraStar

CAPÍTULO 3 - OBJETIVOS

CAPÍTULO 4

ANTECEDENTES

4.1 Antecedentes del Karaoke

La palabra “Karaoke” se remonta del japonés. Viene dada por “Kara”, abreviatura de “Karappo” cuyo significado es vacío y “Oke” abreviatura de “okesutura” cuyo significado es orquesta. Así pues, Karaoke significa “orquesta vacía”.

La historia del karaoke viene de Japón, en donde como en el resto del mundo la entretenimiento a base de música en reuniones y comidas siempre ha sido popular. La primera máquina de karaoke fue introducida por el cantante Daisuke Inoue, en 1971. Él comenzó rentando estas máquinas, que funcionaban con monedas, a diversos establecimientos, y esta forma de entretenimiento comenzó a popularizarse. Las primeras máquinas utilizaron cintas para las grabaciones y más adelante el sistema se implementó con CD's, Laserdisks y posteriormente con DVD.

En el siglo XIX, en Francia surgió un tipo de entretenimiento parecido llamado goguette. Esta modalidad de karaoke consistía en, utilizando la melodía original de una canción que era interpretada en directo, los asistentes de las cantinas y sala de variedades modificaban la letra para así realizar composiciones humorísticas o críticas políticas. Esta modalidad entrañaba una mayor riqueza literaria y artística.

Una máquina de karaoke básica contiene una entrada de audio, un modificador de tono y una salida de audio. También suelen disponer de una pantalla en la que se muestra las letras de las canciones sincronizadas con la música para ayudar al cantante. En la actualidad, el desarrollo de la tecnología, permite la inclusión del karaoke en dispositivos móviles, tales como Smartphones o Tablets, ordenadores portátiles o de sobremesa, con distintos tipos de software desarrollados para los diferentes sistemas operativos, por ejemplo, ‘UltraStar’.

El fenómeno también se ha extendido dentro del mundo de las videoconsolas, con el desarrollo de juegos dentro de las plataformas creadas por Sony, Nintendo o Microsoft, como es el caso de ‘SingStar’.

CAPÍTULO 4 – ANTECEDENTES

CAPÍTULO 5

Materiales y Métodos

5.1 Estado del arte

En este apartado, se va a realizar una breve revisión de cómo se encuentra el mundo del software relacionado con el karaoke, así como de diversas técnicas relacionadas con la separación de fuentes sonoras.

5.1.1 Estado del arte del karaoke

En el mercado actual podemos encontrar muchos tipos de software relacionados con el Karaoke, algunos en modalidad de pago y otro cuya distribución y uso es gratuita. Algunos de ellos permiten cantar con la voz del cantante original atenuada o silenciada, como es el caso de SingStar para PlayStation que permite modular el volumen de la voz local llegando a resultar esta imperceptible o Lips para Xbox donde se tiene la posibilidad de desactivar la voz del cantante principal de las canciones precargadas en las distintas versiones distribuidas de este juego. El caso de UltraStar, software de karaoke multiplataforma, presenta en este punto, su aspecto más negativo, dada la imposibilidad de atenuar o silenciar la voz principal de las canciones que en él se usan.

A continuación se describen los tipos de karaoke anteriormente mencionados, además de otros modelos de karaoke que están presentes en la actualidad.

- **SingStar™:** SingStar constituye una serie de juegos no gratuitos de karaoke publicado por Sony Computer Entertainment Europe para su plataforma PlayStation 2 y recientemente también para PlayStation 3. Estos juegos son distribuidos en DVD junto a par de micrófonos más un convertidor USB. El objetivo del juego es cantar una canción, lo más parecido a la original en cuanto a tono y melodía para ganar más puntos. Así mismo, las canciones son reproducidas sin ser alteradas, permaneciendo la voz original del cantante.
- **Redkaraoke:** es una web dedicada al mundo de la canción, con la peculiaridad de que permite cantar y realizar grabaciones a través de la propia web, sin

necesidad de ningún tipo de software adicional. Es el primer karaoke online en España. Así mismo permite la reproducción vía web de ficheros de karaoke. kar, él estándar en el mundo del karaoke que permite utilizar estos ficheros para grabar canciones. Así mismo, este tipo de karaoke no permite a los usuarios utilizar sus canciones compradas para su adaptación al servicio; tampoco permite aplicar ningún tipo de efecto sobre las pistas proporcionadas; el sistema de captación de voz se limita al uso de los microfonos integrados en los ordenadores de los usuarios.

- **Lips:** Lips constituye una serie de juegos de karaoke disponibles para su compra publicados por Microsoft Game Studios en el año 2008 y cuya disponibilidad es global y su distribución se realiza en DVD. Estos juegos están acompañados de unos micrófonos inalámbricos que permiten realizar gestos que el juego nos indicara durante la partida. Permite una gran variedad de modos de juego y la opción de elegir entre un solo jugador o múltiples jugadores. En este juego, se permite tanto el uso de las canciones precargadas en el mismo como el uso de nuestras propias canciones, sin embargo para estas últimas, no estarán disponibles sus letras para su visualización durante las partidas.
- **UltraStar Deluxe:** UltraStar Deluxe constituye una evolución del software UltraStar lanzado en el año 2007. Se trata de un software gratuito de código abierto disponible para ordenadores bajo la licencia Freeware, que permite ser modificado por sus usuarios con el fin de satisfacer los deseos de los mismo. La experiencia de juego es similar a la de otros productos comerciales, como es el caso de SingStar™ desarrollado por Sony Computer, el cuál es exclusivo de la plataforma Playstation®. UltraStar deluxe permite a sus usuarios crear sus propias canciones. Así mismo, existe una comunidad alrededor de este software, donde sus usuarios crean temas para el programa y también comparten las canciones que ellos mismos han creado, evitando la necesidad de comprar caros DVDs o CDs, como ocurre en los demás software. Con relación ha este TFG, este software no permite la supresión de la voz del cantante. Por este motivo, se ha elegido este software para el desarrollo de este TFG.

En la siguiente figura, podemos ver la interfaz de usuario del karaoke UltraStar, durante una partida de un solo jugador.



Figura 5.1 – Interfaz de usuario del software UltraStar

Como podemos comprobar, existen distintos modelos de negocio relacionados con el karaoke, ya sean videojuegos, software de ordenador, servicios web, etc. En la actualidad, los videojuegos de karaoke, en concreto SingStar con más de veinte millones de copias vendidas, lideran este mercado como consecuencia de las elevadas ventas que ha experimentado el mundo de las videoconsolas en el último siglo. Por otro lado, servicios como Redkaraoke o el software UltraStar, quedan en una posición menos privilegiada, aunque con una base de usuarios en crecimiento continuo.

5.1.2 Estado del arte de los algoritmos de separación de fuentes sonoras

En este apartado se realizará una breve descripción de los distintos métodos de separación de fuentes sonoras que podemos encontrar en la actualidad, ICA, SCA, NMF y MCA. Cada uno de estos métodos explota diferentes características de las señales como la independencia mutua, la no negatividad, dispersión, suavidad o combinaciones de las mismas para obtener las fuentes y reducir la influencia del ruido e interferencias.

- **Análisis de Componentes independientes (ICA):** Este método para la extracción de fuentes se basa en la independencia estadística de las mismas. Así pues, las filas de la matriz S que representan a cada una de las fuentes a extraer deberán ser estadísticamente independientes.

- **Factorización No Negativa de Matrices (NMF):** Se basa en la no negatividad de las muestras de las fuentes a extraer. En el mundo real existen infinidad de escenarios en los que los datos son no negativos y sus componentes subyacentes solo tienen sentido físico cuando se da esta no negatividad.
- **Análisis de Componentes Dispersos (SCA):** en este tipo de algoritmos se considera que las fuentes a extraer son del tipo sparse. Una señal es del tipo sparse cuando la mayoría de sus muestras son cero o próximas a cero, en cuyo caso se dice que son muestras inactivas, y solo un porcentaje de las mismas toman valores significativos, en cuyo caso se dicen que son muestras activas.
- **Análisis de Componentes Morfológicas(MCA):** la separación de las fuentes se basan en la diversidad morfológica de las mismas.

5.1.3 Estado del arte de distintos softwares para servidores web

A continuación, se enumeran distintos tipos de software que actúan a modo de servidores web, así como una breve descripción de sus características.

- **Apache Tomcat:** Es un servidor web multiplataforma programado en Java con soporte de servlets y JSPs. Se trata de un software de código abierto desarrollado por Apache Software Foundation. Incluye un compilador Jasper, que compila JSPs convirtiéndolas en servlets. Implementa un servidor web HTTP que permite la ejecución de código en Java. Cada petición HTTP a Tomcat se procesa en un hilo separado, lo que permite el manejo simultáneo de varias sesiones.
- **JBoss:** Es un servidor web de aplicaciones Java EE de código abierto implementado en Java, lo que permite su disponibilidad en cualquier sistema operativo. Ofrece una plataforma de alto rendimiento para aplicaciones de e-business.

- **Jetty**: Es un contenedor de Servlets y Servidor HTTP escrito en Java ultraligero con licencia Open Source, su condición de aplicación ligera la hace perfecta para embeberla dentro de grandes aplicaciones Java. Su tamaño permite ofrecer Servicios Web en aplicaciones independientes.

5.2 Desarrollo del TFG

El resultado final de este Trabajo Fin de Grado ha sido un servicio web que complementa al software de karaoke UltraStar, accesible desde cualquier dispositivo con conexión a internet mediante un navegador web, implementado sobre Apache Tomcat 7.

En este servicio, los usuarios del programa UltraStar deben enviar el archivo de audio correspondiente a la canción que desean procesar ya sea en formato MP3 o en formato WAV, y el archivo de texto con la información de dicha canción creado por ellos mismos, o bien descargado de internet. Tras ser procesado, el usuario podrá descargar un archivo comprimido en formato ZIP, el cual contendrá la correspondiente canción procesada que deberá sustituir por la canción original.

En el siguiente diagrama de flujo se enumeran las distintas fases de forma general por las que transcurre todo el proceso.

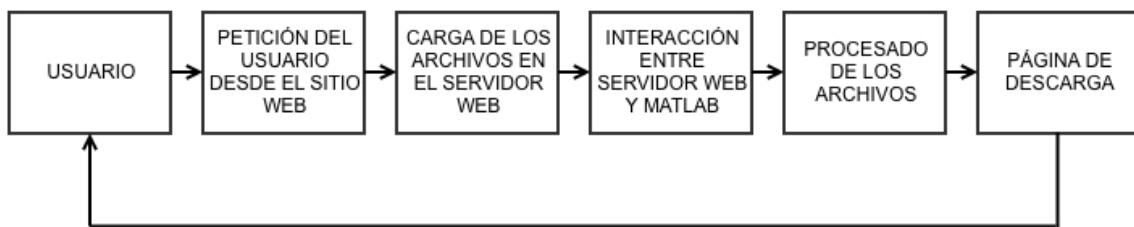


Figura 5.2 – Fases del Proceso

5.2.1 Archivos enviados por el usuario

5.2.1.1- Archivo de texto

Este archivo es el que hace posible la compatibilidad con el programa UltraStar. Para su creación, debe seguirse el protocolo definido en la plataforma del programa, véase [10] y que, a continuación, vamos a describir.

El siguiente texto es un ejemplo de la información contenida en el archivo de texto y el formato que siguen las notas de sincronización.

```
#TITLE:Yesterday
#ARTIST:The Beatles
#LANGUAGE:Other
#GENRE:Other
#MP3:The Beatles - Yesterday.mp3
#COVER:The Beatles - Yesterday [CO].jpg
#BACKGROUND:The Beatles - Yesterday [BG].jpg
#RESOLUTION:0
#BPM:96
#GAP:6450
: 0 2 7 Yes
: 2 1 5 ter
: 3 13 5 day
- 15
: 20 2 9 all
: 22 2 11 my
: 24 1 13 trou
: 25 3 14 bles
: 28 2 16 seemed
: 30 2 17 so
: 32 5 16 far
: 37 2 14 a
: 39 9 14 way
- 44
E
```

#TITLE

Etiqueta de carácter obligatorio. Se corresponde al título de la canción. Hay canciones que se las conoce por un título distinto al que les puso su creador originalmente, así que en estas ocasiones se pondrá poner entre paréntesis y a continuación el nombre conocido.

#ARTIST

Etiqueta de carácter obligatorio. Representa al artista que interpreta la canción. Si no conoce el nombre del artista, puede usarse el nombre de una serie o película que esté relacionada con la canción.

#LANGUAGE

Etiqueta de carácter opcional. Corresponde al idioma principal de la canción. Tiene una gran utilidad y es recomendable su uso para la clasificación de canciones.

#EDITION

Etiqueta de carácter opcional. Es la edición a la que pertenece la canción. Es lo que se utiliza para clasificar las canciones que han sido convertidas desde los SingStar.

#GENRE

Etiqueta de carácter opcional. El género de la canción ya sea Rock, Pop, Country, etc. Generalmente, no suele usarse correctamente por lo que no es de gran utilidad para la clasificación de canciones.

#YEAR

Etiqueta de carácter opcional. Corresponde al año de lanzamiento comercial de la canción o el disco del cual forma parte.

#CREATOR

Etiqueta de carácter opcional. Nombre o apodo del creador de la canción.

#MP3

Etiqueta de carácter obligatorio. Indica el nombre del archivo de audio que contiene la canción. Suele tener el mismo formato que las etiquetas #ARTIST y #TITLE. Un ejemplo podría ser: The Beatles - Yesterday.mp3

#COVER

Etiqueta de carácter opcional. Indica la carátula del disco o single en el que salió la canción. Puede tener diversos formatos, aunque el más popular es JPEG. En el caso de que haya otra imagen de fondo para la canción se añade un espacio seguido de [CO] al final para distinguirlas.

#BACKGROUND

Etiqueta de carácter opcional. Indica la imagen que sustituirá en el fondo si no se usa video.

#VIDEO

Etiqueta de carácter opcional. Es el video que se mostrara de fondo mientras se canta la canción. En ocasiones se le añade al final del nombre el videogap que tiene la canción.

#VIDEOGAP

Etiqueta de carácter opcional. Es el número de segundos que va a empezar el video antes o después de que comience la canción.

#BPM

Etiqueta de carácter obligatorio. Son los golpes por minuto (Beats Per Minute en inglés) de la canción, es decir la velocidad de la canción. Se calcula con ciertos programas directamente del archivo de audio. Se trata de un valor comprendido entre 150 y 400.

#GAP

Etiqueta de carácter obligatorio. Es la pausa en milisegundos desde que comienza a sonar la canción hasta que se comienza a cantar. Suele calcularse a ojo.

#START

Etiqueta de carácter opcional. Es el número de segundos en el que va a empezar la canción.

#END

Etiqueta de carácter opcional. Es el número de milisegundos en el que va a terminar la canción.

#RELATIVE

Etiqueta de carácter opcional. Se establece a “YES” cuando queremos indicar que en cada línea que se escriba las notas comenzarán desde la posición cero y no desde la posición que le debería corresponder si se cuenta desde el inicio de la canción.

Dando por terminada la descripción de las etiquetas del archivo de texto. Ahora se va a realizar una explicación de cómo debemos expresar las notas y la letra para obtener una correcta sincronización. Tomaremos como referencia la siguiente línea del texto.

: 0 2 7 Yes

A continuación, se explicará el significado de cada valor que aparece en la anterior línea.

El primer símbolo (:) hará referencia al tratamiento que tendrá la nota cantada.

En el segundo valor (0), encontramos un número que nos indica el instante temporal en el que se empieza a ejecutar la nota. Se trata de un valor definido en tiempo de negra, donde debemos tener en cuenta que en un minuto hay tantos tiempos de negra como el valor del BPM multiplicado por cuatro, ya que en un compás entran cuatro negras.

El tercer valor (2), nos indica la nota en escala MIDI a la que se debe cantar. Dado que no hay información de la escala, se contemplarán 12 notas MIDI.

El cuarto valor (7), indica la duración de la nota en las mismas unidades que la posición temporal.

El ultimo valor (“Yes”), indica la sílaba o sílabas que se corresponderán con dicha nota.

5.2.1.2 Archivo de Audio

El archivo de audio deberá tener el mismo nombre que el archivo de texto antes mencionado. El formato de audio más usado es MP3; sin embargo, UltraStar también es compatible con diversos formatos entre los que se incluye WMA, WAV, AAC y FLAC.

En el caso de usar un archivo de audio codificados en MP3, este deberá tener una frecuencia de muestreo inferior a 48.000Hz. El bitrate no es un punto de preocupación, pero no se recomienda usar canciones codificadas con un bitrate mayor a 720kbps dado que surgen errores en la reproducción.

5.2.2 Desarrollo de una página web como interfaz de usuario

Para la realización de este TFG, se ha diseñado una página web que actuará como intermediaria entre el servicio que se ofrece y el cliente o usuario que desea acceder a él.

Para el desarrollo de la página web se han tomado como referencia el lenguaje de marcas HMTL5, véase [11], con el cuál se ha estructurado la página en diversas secciones. Una vez estructurada la página, es necesario dar un formato a la misma. Para esto se ha usado CSS3, una hoja de estilo, un mecanismo simple que describe cómo se va a mostrar un documento en la pantalla, cómo se va a ser pronunciaba la información presente en ese documento a través de un dispositivo de lectura, proporcionando un control total sobre el estilo y formato del sitio web.

Para ofrecer un sitio más amigable al usuario, se han definido marcos de texto con ciertos colores llamativos. Así mismo se han creado efectos en el sitio web que lo hacen

más amigable de cara al usuario, como translaciones en el mensaje superior o ventanas deslizantes para cada una de las secciones.

Para el envío de los archivos requeridos, se ha optado por usar un formulario web, que mediante el método POST de HTTP, enviará ambos archivos al servidor y será el encargado de llamar al Servlet para su almacenamiento.

Una vez acabado todo el proceso, se mostrará una nueva página donde el usuario tendrá la posibilidad de descargar su canción procesada dentro de un contendor ZIP. El sitio web se a desarrollado en un contendor JSP llamado “index.jsp”, dada la compatibilidad de Apache Tomcat con este tipo de formatos. Este archivo JSP, se encargará de pasar los archivos enviados por el usuario al Servlet implementado en el archivo “procesawavytxt.jsp”.

5.2.3 Implementación en JSP de Servlet en Java

Para manejar las peticiones procedentes del formulario incluido en el sitio webs necesario implementar un Servlet, véase [6] y [9], que pueda manejarlas y que permita almacenar los archivos recibidos dentro del servidor.

Para el desarrollo del Servlet se ha optado por usar contenedores JSP, dentro de los cuales se ha hecho uso de distintas clases Java, así como de los correspondientes métodos necesarios para llegar a un funcionamiento correcto del mismo.

El Servlet ha sido implementado dentro del archivo JSP llamado “procesa_wavytxt.jsp”. En caso de seleccionar la opción de lengua inglesa en el sitio web, la comunicación se realizará con el archivo “procesa_wavytxt_english.jsp”, el cual hace uso de las mismas clases y métodos que “procesa_wavytxt.jsp”, pero retornando una página en inglés.

Ha sido necesario descargar la librería “commons-fileupload-1.2.2.jar” distribuido por Apache Commons FileUpload TM, que provee soporte para la carga de varios archivos dentro de un Servlet, junto con la librería “commons-io-2.4.jar”, indispensable para el funcionamiento de la primera. Además es necesario instalar el kit llamado Java

Development Kit (JDK) provisto por Oracle® para el sistema operativo Linux. De esta forma, será posible acceder a la clases necesarias para el desarrollo del Servlet.

La implementación realizada hasta llegar al fin propuesto sigue así, las distintas clases Java utilizadas, así como sus métodos, vienen descritas en el manual técnico, página 91 a 109.

- En cada nueva sesión, se tomará un identificador de sesión llamado “id_sesion”.
- Se crea un directorio de trabajo donde serán almacenados los archivos enviados. Para cada nueva sesión, los archivos se ubicarán en carpetas distintas.
- Mediante el método “isMultipartContent (request)” de la clase “ServletFileUpload”, se comprobará la existencia de una petición HTTP que contenga la carga de archivos. Para una comprobación exitosa, la petición debe ser enviada con el método POST y usando el tipo de contenido “multipart / form-data” dentro de un formulario diseñado en html.
- Ante una respuesta afirmativa, mediante el uso de las clases “DiskFileItemFactory ()” y “ServletFileUpload ()”, se establecerán los parámetros para la subida de los archivos.
- Una vez establecida la configuración, se procesará la petición para extraer los archivos y se creará una lista con una referencia a cada uno de ellos, haciendo uso del método “parseRequest(request)”.
- Utilizando los métodos “iterator()”, “hasnext()” y “next()” de la clase “Iterator()”, se recorrerá dicha lista.
- En cada iteración, se tomará el nombre completo de cada archivo con el método getName(); seguido, se comprobará su extensión usando el método “indexOf()”, con tres posibles coincidencias, “.wav” , “.mp3” o “.txt”.

- En caso de coincidencia, el archivo será guardado dentro del directorio de trabajo, dentro de una carpeta cuyo nombre coincidirá con el identificador de la sesión actual con el método “write()”.
- Si el archivo de audio y el archivo de texto son encontrados, pasaremos a la siguiente fase, comunicarnos con el programa Matlab. Para este fin, se ha diseñado una clase llamada “llamarfuncionMatlab(path, id_sesion, wavname, txtname)”.
- Una vez se ha finalizado el procesado de los archivos, se retornará la página de descarga de los mismos.
- Por último, el directorio donde se han almacenado los archivos subidos será eliminado. Para ello, se hará uso de los métodos “listFiles()” y “delete()”, borrando cada archivo de forma individual y, finalmente, el directorio temporal donde estaban almacenados.

A continuación, se mostrará un diagrama de flujo del proceso anteriormente descrito, en el cual se indica las distintas fases por las que transcurre el proceso principal en cada nueva sesión creada.

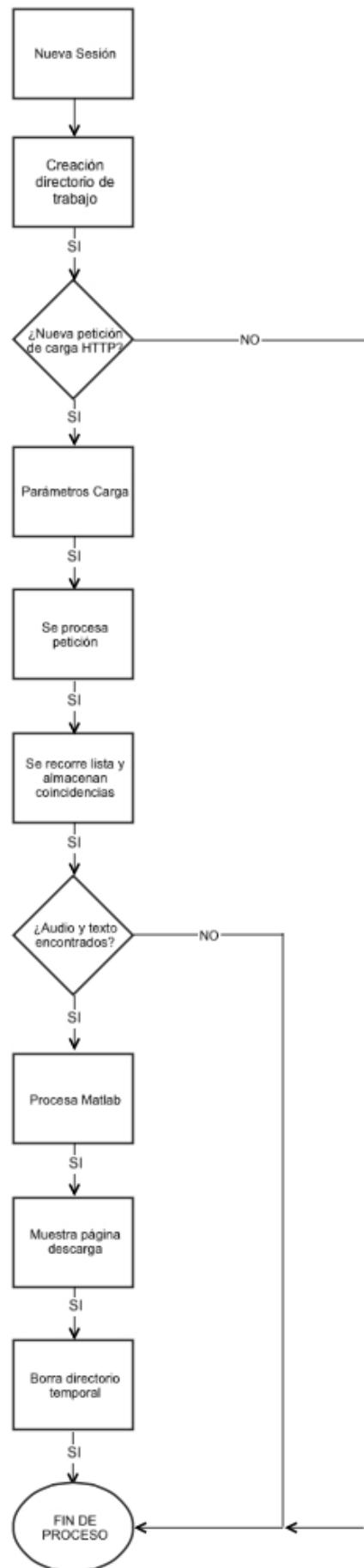


Figura 5.3 – Procesos en el Servidor Web

5.2.4 Comunicación entre el JAVA y Matlab

Para la comunicación entre el Servlet en Java y el programa Matlab, se encontraron varias posibilidades. Se encontraron distintas APIs programadas en Java que podían cumplir con este cometido; sin embargo, sólo la API llamada “Matlabcontrol” permite su uso con las versiones más recientes del programa Matlab. Esto llevó a descartar automáticamente las demás.

La API Matlabcontrol, véase [7], en su versión más reciente, la 4.1.0, fue lanzada bajo la licencia “New BSD License”. Permite evaluar comandos de Matlab en Java como si se tratase de la “Command Windows” del propio Matlab, mediante el método “eval(string)”. También permite seleccionar y obtener variables.

Para la comunicación entre ambos, esta API hace uso de la JMI (Java MATLAB Interface). Matlab ya dispone el contenedor JMI.jar necesario en su directorio principal, ante esto, no es necesario el uso de ninguna otra dependencia como es el caso de otro tipo de API.

Se ha creado un Package llamado “llamarMatlabpack”, dentro del cual podemos encontrar la clase “llamarMatlab”. Dentro de esta clase, se han programado dos métodos, ‘llamarfuncionMatlab(String path, String id_sesion, String wavname, String txtname)’ que es el encargado de establecer la comunicación con Matlab, y el método ‘getOutname()’ que devuelve el nombre del contenedor ZIP que devuelve Matlab tras procesar los archivos.

Las distintas clases Java, así como sus métodos, se detallan en el manual técnico, páginas 110 a 123. Para implementar la comunicación el método ‘llamarfuncionMatlab()’ procede como sigue.

- En primer lugar, mediante los métodos “setHidden()”, “setProxyTimeout()”, “setMatlabLocation()” y “build()” que pertenecen a la clase “Builder()”, se establecerán una serie de opciones, para la creación posterior del proxy.

- Después, junto con las opciones anteriores se crea el proxy encargado de establecer la conexión con Matlab, haciendo uso de la clase “MatlabProxyFactory(options)” y el método “getProxy()”.
- Una vez creado el proxy, se seleccionan las variables que necesitará la función de Matlab que implementara el algoritmo de separación de fuentes de audio. Con este fin, se usa el método “setVariable()”. Los parámetros necesarios serán, “path” , “id_sesion” , “wavname” y “txtname”.
- Tras esto, ya estamos en disposición de llamar a nuestra función programa en Matlab; usaremos el método “eval()” para llamarla. Una vez finalizado el proceso, se devolverá el nombre del fichero ZIP producido.
- Finalmente, podremos obtener esta variable mediante el método “getVariable()”.

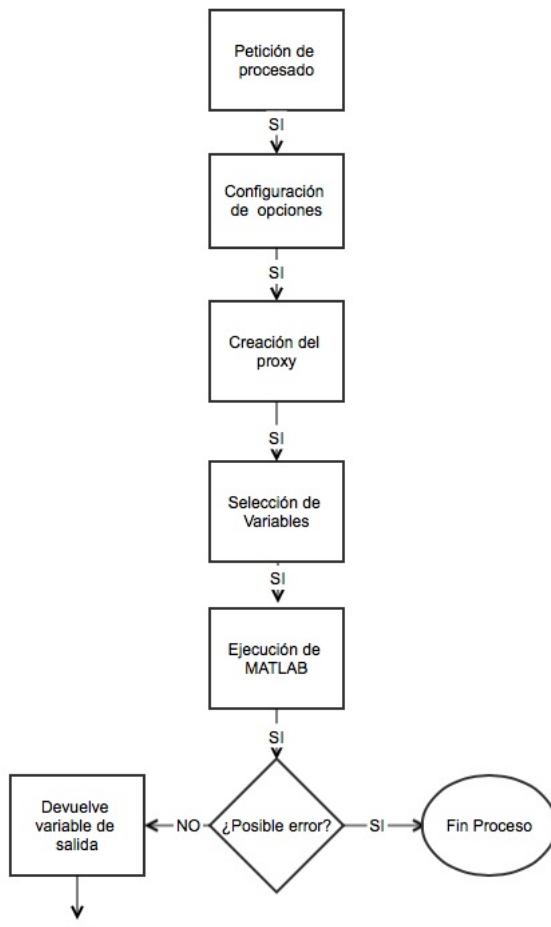


Figura 5.4 – Fases de Matlabcontrol

5.2.5 Introducción al programa desarrollado

En una pista de audio mezclada, podemos encontrar dos tipos de zonas principales, zona vocal/instrumental y zona instrumental. La integración del programa UltraStar dentro del algoritmo programado, ha provisto a este del etiquetado, es decir, la capacidad de determinar la ubicación de zonas instrumentales dentro de una canción.

Resulta interesante conocer aquellas zonas de la canción donde únicamente encontramos información de fuentes instrumentales. Esta información ayudará al entrenamiento de bases armónicas/percusivas en aquellas zonas instrumentales que la canción presente.

En el programa desarrollado, se ha utilizado un primer modelo basado en el algoritmo NMF con una serie de restricciones que permite el entrenamiento de bases percusivas y armónicas, que serán almacenadas para su uso posterior.

Una vez las bases han sido entrenadas con el primer modelo, son usadas en un segundo modelo del algoritmo NMF, donde se realiza la separación vocal/instrumental. La información de estas bases ayuda al entrenamiento de nuevas bases armónicas/percusivas en aquellas zonas de la canción donde podemos encontrar tanto fuentes vocales como instrumentales, permitiendo una separación más exitosa entre ambos tipos.

Finalmente, las señales $x_v(t)$, $x_p(t)$ y $x_h(t)$, son sintetizadas mediante los espectrogramas X_v , X_p y X_h , calculados como el producto de las respectivas bases factorizadas vocales, percusivas y armónicas. Se utiliza una máscara Wiener, para determinar en cada punto de la canción cual es la componente predominante.

En las siguientes páginas, se muestra un diagrama de flujo con todas las fases que componen el programa que ejecuta Matlab cuando se establece la conexión con el servidor web.

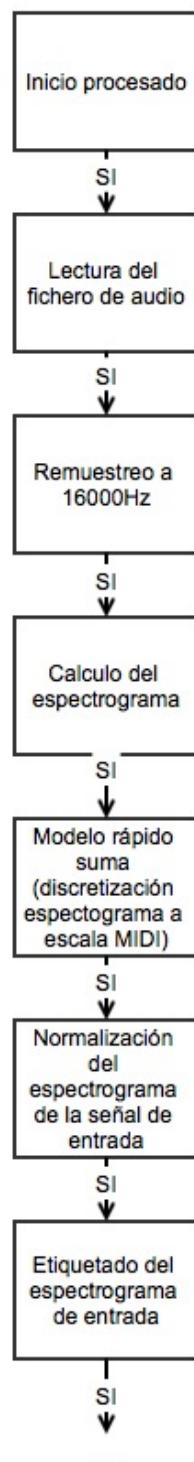


Figura 5.5 – Primera parte programa desarrollado en MATLAB

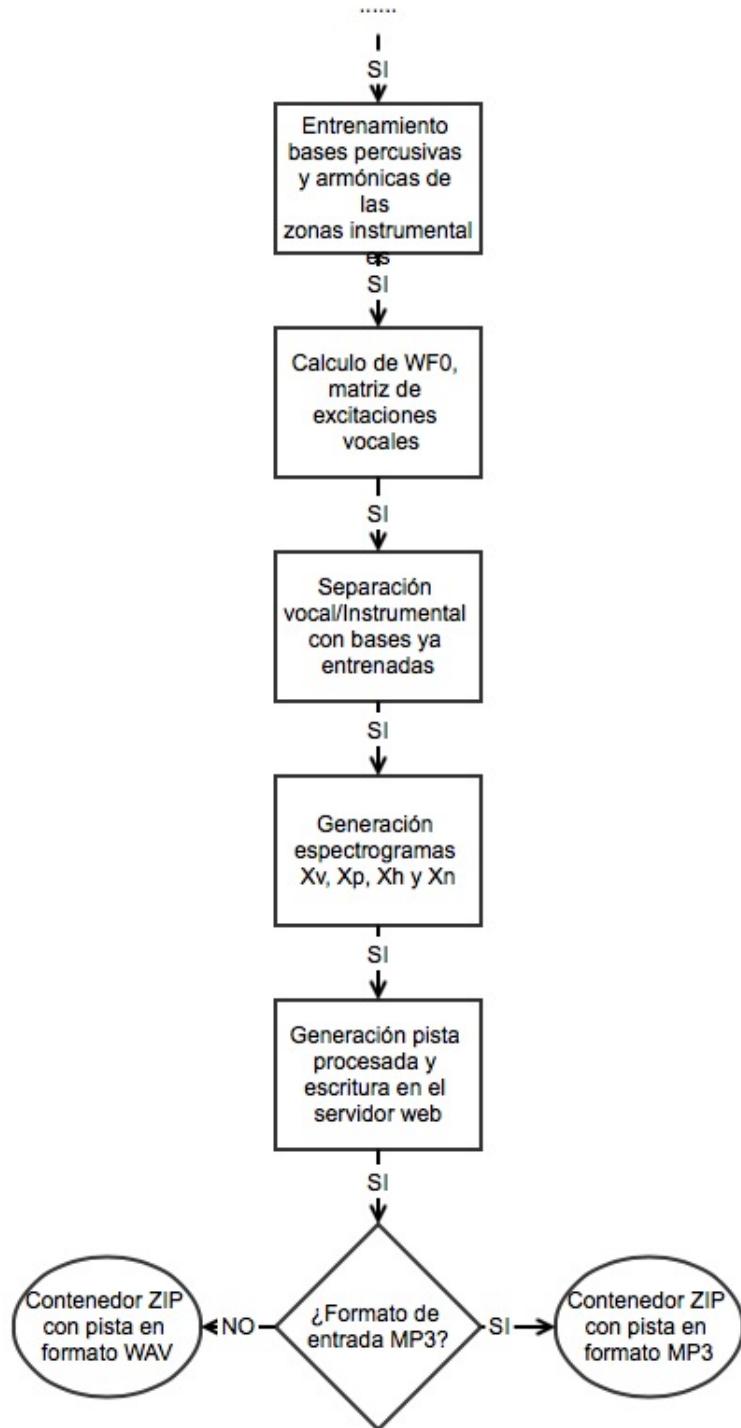


Figura 5.6 – Segunda parte programa desarrollado en MATLAB

5.2.6 Integración de la información extraída del programa UltraStar

En relación con el programa UltraStar, una vez el usuario ha enviado los archivos de audio y texto, podemos extraer información bastante útil de este último para el procesado de la canción. El contenido de este archivo ya ha explicado al comienzo de este capítulo.

El archivo de texto de UltraStar, contiene información sobre la posición y duración de cada nota dentro de la pista. De esta manera, gracias a la diferencia de tiempos entre notas, pueden encontrarse aquellas zonas donde sólo aparecen fuentes instrumentales. Sin embargo, una diferencia muy pequeña no es útil ya que no permitiría que las bases lleguen a la convergencia. Por este motivo, se tomarán como zonas instrumentales aquellas cuya duración supere un cierto umbral.

Para extraer la información de archivos de texto, Matlab viene provisto de determinadas funciones como es el caso de ‘load()’. Sin embargo, estas sólo permiten extraer la información del archivo siempre y cuando esta tenga una estructura homogénea.

El archivo de texto procedente del programa UltraStar, presenta una estructura bastante heterogénea, por lo que ha sido necesario desarrollar un algoritmo capaz de extraer la información necesaria de este. Esta información incluye, constante GAP, constante BPM, vector de tiempos $x_{duración}$ y vector de posiciones, $x_{posición}$.

Dado el espectrograma de entrada X , definido en la siguiente ecuación, se notará matemáticamente el proceso de etiquetado descrito.

$$X(f, t) \quad t \in t_{inst} \cup t_{instvocal} \quad (1)$$

Con t_{inst} referente a las zonas instrumentales de la canción y $t_{instvocal}$ referente a las zonas instrumentales/vocales. Así pues, el espectrograma de entrada puede descomponerse como:

$$X_{instrumental} = X(f, t_{inst}) \quad (2)$$

$$X_{instrumental/vocal} = X(f, t_{instvocal}) \quad (3)$$

5.2.7 Desarrollo de modelos basados en el algoritmo NMF

5.2.7.1 Descripción general de los modelos empleados

Para el desarrollo de este programa se han usado dos modelos distintos del algoritmo NMF. Este algoritmo se basa en la factorización de una matriz de entrada, es decir, la matriz de entrada es considerada el producto de un conjunto de matrices.

En los modelos desarrollados, se ha tomado como referencia un espectrograma X de entrada, con dimensiones frecuencia-tiempo, para la factorización. Este espectrograma es descompuesto en dos matrices distintas; por un lado, se obtiene una matriz W que se corresponde con la excitación de una fuente sonora en frecuencia; por otro lado, se obtiene una matriz H, que corresponde con la activación de esa excitación en el dominio del tiempo.

Para el primer modelo, se ha tomado que el espectrograma de entrada está formado por dos espectrogramas diferentes, uno correspondiente a la parte percusiva y otro correspondiente a la parte armónica. En este modelo, solo se realiza separación entre fuentes instrumentales, por lo que se aplica únicamente a aquellas zonas de la canción donde sólo se encuentran fuentes instrumentales; para ello se usa la información extraída del programa UltraStar. Estas dos matrices contienen las bases armónicas y percusivas, donde se encuentra la información que indica si en un determinado punto de la canción aparecen sonidos armónicos, sonidos percusivos o ambos.

Ambas bases son actualizadas mediante las llamadas reglas de actualización multiplicativas, es decir, a través de un número determinado de iteraciones la información de estas bases va actualizándose. Para el cálculo de esta información se minimiza una función de coste global, llamada D, que se compone de tres partes. La primera parte, es el llamado coste β -divergencia, que hace referencia a la similitud de la señal original con la

señal tras cada paso por el algoritmo. La segunda parte, es el costo percusivo, que toma cuenta de la cantidad de sonido percusivo que habrá en un punto de la canción. La última parte, es el costo armónico, que cuenta con la misma función que el costo percusivo pero referido a los sonidos armónicos.

Una vez estas bases son calculadas, son usadas en el segundo modelo para realizar la separación entre sonidos armónico/percusivos y vocales. Se calculan nuevas bases, esta vez en las zonas instrumentales/vocales. Para ello, en este modelo se introducen un nuevo espectrograma referente a la información vocal. Se usa un modelo excitación/filtro para estimar la parte vocal de estas zonas, es decir, a cada excitación vocal corresponde un filtro determinado que permite su extracción, para finalmente obtener el trayecto de la voz principal de la canción.

En este segundo modelo, se actualizan las bases percusivas y armónicas, así como las bases correspondientes a las activaciones de filtros y excitaciones vocales usando las reglas de actualización multiplicativas ya comentadas, en las zonas no calculadas anteriormente.

En la siguiente figura, se detalla el intercambio de información entre los dos modelos implementados, haciendo referencia a la información de entrada necesaria para el correcto funcionamiento de cada algoritmo.

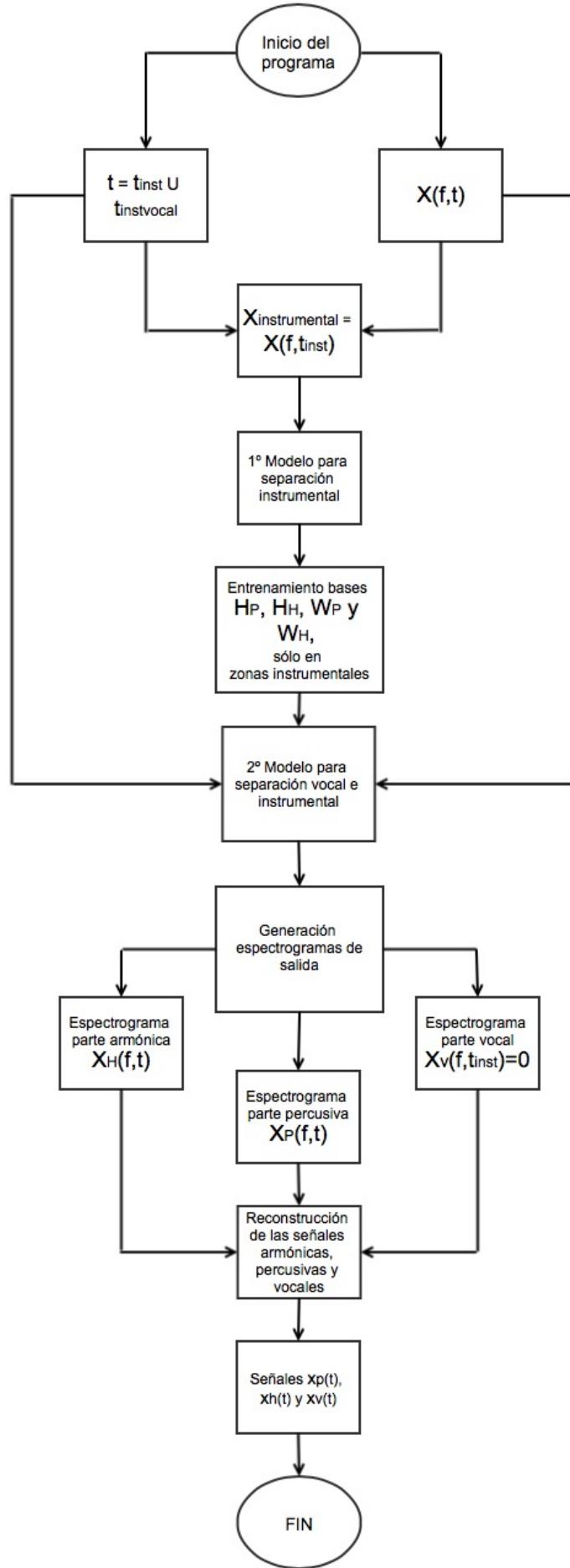


Figura 5.7 – Intercambio de información entre los modelos desarrollados
44

5.2.7.2 Primer modelo basado en el algoritmo NMF

La motivación de este modelo es realizar la separación entre fuentes armónicas/percusivas en aquellas zonas de la canción donde sólo aparecen fuente instrumentales. Para ello, el algoritmo toma como parámetro de entrada el espectrograma $X_{instrumental}$ definido en la ecuación (2). Como resultado, este modelo genera las bases percusivas, W_P y H_P , y las bases armónicas, W_H y H_H , optimizadas en las zonas sólo instrumentales de la canción.

Para este TFG, se ha implementado una modificación del algoritmo NMF (Non-Negative Matrix Factorization), véase [1], que permite el entrenamiento de bases armónicas/percusivas en aquellas zonas de una pista ya mezclada donde sólo encontramos fuentes instrumentales. Se han asumido una serie de restricciones que ofrecen ciertas ventajas respecto al algoritmo original, en concreto, consideraciones de suavidad y dispersión.

Este modelo se basa en una modificación del algoritmo NMF que no requiere ningún postprocesado o entrenamiento para distinguir entre bases armónicas y percusivas dado que la información se encuentra en el proceso de descomposición.

Se asume que los sonidos armónicos poseen cualidades de dispersión espectral y suavidad temporal, mientras que los sonidos percusivos poseen cualidades de dispersión temporal y suavidad espectral.

Entendemos los sonidos percusivos como aquellos producidos al golpear instrumentos, mientras que los sonidos armónicos se corresponden con aquellos que surgen como variaciones de frecuencias como es el caso de la voz.

El método aquí desarrollado presenta una serie de ventajas.

- Se trata de un método robusto para varias fuentes espectrales.
- Es un algoritmo eficiente y veloz en el procesamiento.
- No hay necesidad de definir umbrales.

Este método no necesita de entrenamiento ya que la información armónico/percusiva, es extraída del espectro tiempo-frecuencia que se usa en la factorización. Además, distingue automáticamente entre bases percusivas y bases armónicas.

En sonidos armónicos típicamente encontramos un espectrograma disperso en frecuencia; así mismo, para sonidos percusivos el espectrograma se caracteriza por ser impulsivo en frecuencia y disperso en tiempo.

Estas características permiten modelar sonidos armónicos usando la dispersión en frecuencia, picos espectrales, y la suavidad en tiempo, poca variación en la amplitud. Para los sonidos percusivos encontramos el caso contrario, en frecuencia la energía decrece lentamente mientras en tiempo la mayor parte de la energía se encuentra concentrada en pequeñas regiones.

En este método, no se ha considerado:

- Los fonemas vocales, pueden presentar características armónicas que se implementan en este método.
- El método no está diseñado para distinguir entre fonemas sordos (con propiedades percusivas) y fonemas sonoros (con propiedades armónicas).

En la siguiente figura, se ha analizado una pista de seis segundos en la que podemos comprobar las características anteriormente mencionadas.

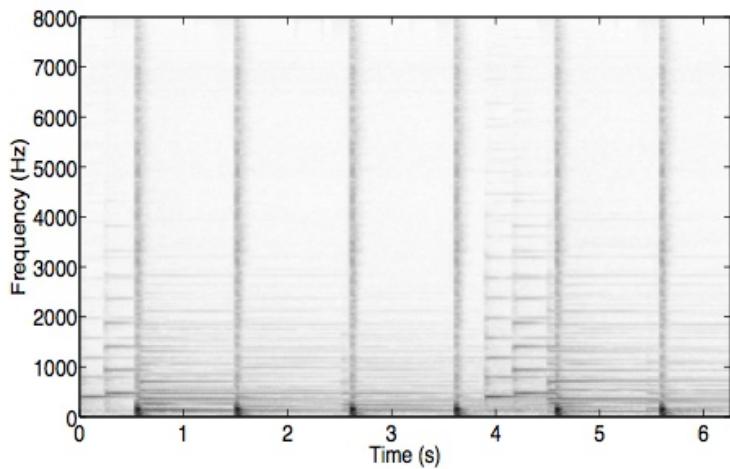


Figura 5.8 – Ilustración de las restricciones de dispersión y suavidad

El resultado de la descomposición de un espectrograma de entrada X mediante la factorización en una matriz no negativa, es dos matrices no negativas que notaremos como W y H .

$$X \approx W * H \quad (4)$$

El espectrograma magnitud X de una señal musical $x(t)$, se compone de tramas T junto con bins de frecuencia F y un conjunto de unidades tiempo-frecuencia $X_{f,t}$. Cada $X_{f,t}$ se define con un bin de frecuencia f -th y un frame de tiempo t -th, calculado usando la magnitud de la STFT, Transformada de Fourier de Tiempo Reducido, usando una ventana Hamming $w(n)$ de N muestras y un desplazamiento temporal J .

La columna i -ésima de la matriz W es una base de frecuencia que representa el patrón espectral de una fuente de sonido que está activa en el espectrograma. Así, la fila i -ésima de la matriz H , es considerada la ganancia o activación temporal que representa el intervalo de tiempo en el cuál, la base de frecuencia i -ésima está activa.

El algoritmo original del NMF, no puede usarse para distinguir entre los dos tipos de bases mencionadas, ya que sólo asegura la convergencia a mínimos locales, que permite reconstruir el espectrograma pero no distinguir entre las bases.

Es necesario normalizar la entrada para llevar a cabo la correcta separación entre fuentes armónicas/percusivas. Por lo que el espectrograma $X_{n\beta}$, se calcula con el número de tramas, el número de bins y normalización de la norma beta.

$$X_{n\beta} = \frac{X}{\sigma \cdot X_\beta} = \frac{X}{\left(\frac{\sum_{f=1}^F \sum_{t=1}^T x_{f,t}^\beta}{FT} \right)^{\frac{1}{\beta}}} \quad (5)$$

Se define una función para factorizar un espectrograma X_n como mezcla de dos espectrogramas separados X_p y X_h . Cada espectrograma separado exhibe características espectro-temporales para los sonidos armónicos/percusivos.

$$X_{n\beta} = X_p + X_h = W_{P_{F,Rp}} \cdot H_{P_{Rp,T}} + W_{H_{F,Rh}} \cdot H_{H_{Rh,T}} \quad (6)$$

$R_p \approx$ Número de componentes percusivas usadas en la factorización

$R_h \approx$ Número de componentes armónicas usadas en la factorización

La asintropía, se usa para estimar W_p , H_p , W_h y H_h minimizando una función de coste global que depende del coste β -divergencia, el coste de percusión y el coste armónico.

Se desarrolla los distintos costes anteriormente mencionados en los siguientes puntos, así como el algoritmo global NMF armónico/percusivo.

- **Coste β -divergencia:** El espectrograma normalizado de entrada $X_{n\beta}$ se construye minimizando el coste β -divergencia $d\beta(x|y)$ para los dos espectrogramas X_p y X_h .

Considerando que, $x = X_{n\beta}$ e $y = X_p + X_h$.

$$d\beta(x|y) = \begin{cases} \sum_{f=1}^F \sum_{t=1}^T \frac{1}{\beta(\beta-1)} (x_{f,t}^\beta + (\beta-1)y_{f,t}^\beta - \beta x_{f,t} y_{f,t}^{\beta-1}) \quad \beta \in (0,1) \cup (1,2] \\ \sum_{f=1}^F \sum_{t=1}^T x_{f,t} \log \frac{x_{f,t}}{y_{f,t}} - x_{f,t} + y_{f,t} \quad \beta = 1 \\ \sum_{f=1}^F \sum_{t=1}^T \frac{x_{f,t}}{y_{f,t}} - \log \frac{x_{f,t}}{y_{f,t}} - 1 \quad \beta = 0 \end{cases} \quad (7)$$

En este modelo, la aproximación de NMF usa una función objetiva que considera el costo β -divergencia y las características espectro-temporales comunes en las señales percusivas y armónicas.

- **Coste percusivo:** se toman las consideraciones de suavidad en frecuencia y dispersión en tiempo. Los sonidos percusivos suelen representarse como señales de banda ancha, con energía confinada en cortos intervalos de tiempo. Se usa una definición de suavidad espectral, SSM, asociada con la matriz W_p .

$$SSM = \frac{T}{R_p} \sum_{r_p=1}^{R_p} \frac{1}{\sigma W_{p,r_p}} \sum_{f=2}^F (W_{p,f-1,r_p} - W_{p,f,r_p})^2 \quad (8)$$

Un alto coste, se asocia con ganancias distintas de cero, suponiendo que los sonidos percusivos pueden representarse como transitorios con energías concentradas en cortos intervalos de tiempo. Este hecho se asocia con H_p .

$$TSP = \frac{F}{R_p} \sum_{r_p=1}^{R_p} \sum_{t=1}^T \left| \frac{H_{P_{rp},t}}{\sigma H_{P_{rp}}} \right| \quad (9)$$

$$\text{Con } \sigma H_{P_{rp}} = \sqrt{\frac{1}{T} \sum_{t=1}^T H_{P_{rp},t}^2} \text{ y } \sigma W_{P_{rp}} = \sqrt{\frac{1}{F} \sum_{f=1}^F W_{P_{rp},t}^2}$$

- **Coste armónico:** Para los sonidos armónicos ocurre de forma similar que en el caso anterior. Sin embargo, en este caso se toman consideraciones de dispersión en frecuencia y suavidad en tiempo.

$$SSP = \frac{T}{R_h} \sum_{r_h=1}^{R_h} \sum_{f=1}^F \left| \frac{W_{H_{f,r_h}}}{\sigma W_{H_{r_h}}} \right| \quad (10)$$

$$TSM = \frac{F}{R_h} \sum_{r_h=1}^{R_h} \frac{1}{\sigma H_{H_{r_h}}}^2 \sum_{t=2}^T (H_{H_{r_h,t-1}} - H_{H_{r_h,t}})^2 \quad (11)$$

- **Algoritmo global NMF armónico/percusivo:** la función global D, incluido el coste β -divergencia, aparece en la siguiente ecuación. Además, se definen las variables K_{SSM} , K_{TSM} , K_{TSP} y K_{SSP} , constantes que hacen referencia al peso en la función D de los sonidos armónico/percusivos con sus correspondientes restricciones de dispersión y suavidad.

$$D = d\beta \left(X_{n\beta} \left| (X_p + X_h) \right. \right) + K_{SSM} \cdot SSM + K_{TSP} \cdot TSP + K_{SSP} \cdot SSP + K_{TSM} \cdot TSM \quad (12)$$

Dado que los valores K_{SSM} y K_{TSM} no presentan diferencias significativas en su inicialización, se define un parámetro igual a estos que represente los términos correspondientes a la consideración de suavidad, K_{SM} . Igual ocurre con los términos que representan la consideración de dispersión, K_{TSP} y K_{SSP} , que serán definidos por un parámetro de igual valor a los mismos, K_{SP} .

Se usan las llamadas “reglas de actualización multiplicativas”, de forma que D no aumente, mientras se garantiza la no negatividad mediante un algoritmo de gradiente descendente, usando una correcta elección del tiempo de salto y estimación para cada parámetro escalar Z. Se expresan las derivadas parciales de la función objetiva $\frac{\partial D}{\partial Z}$, como la diferencia entre $\left[\frac{\partial D}{\partial Z}\right]^+$ y $\left[\frac{\partial D}{\partial Z}\right]^-$. En este primer modelo, se optimizan las bases W_p , W_h , H_p y H_h .

$$Z = Z \odot \frac{\left[\frac{\partial D}{\partial Z}\right]^-}{\left[\frac{\partial D}{\partial Z}\right]^+} \quad (13)$$

Sustituyendo las bases percusivas W_p y de ganancia H_p dentro de la ecuación (11), se obtienen las siguientes reglas de actualización multiplicativas para sonidos percusivos.

$$W_p = W_p \odot \frac{\left[\frac{\partial d\beta}{\partial W_p}\right]^- + K_{SSM} \left[\frac{\partial SSM}{\partial W_p}\right]^-}{\left[\frac{\partial d\beta}{\partial W_p}\right]^+ + K_{SSM} \left[\frac{\partial SSM}{\partial W_p}\right]^+} \quad (14)$$

$$H_p = H_p \odot \frac{\left[\frac{\partial d\beta}{\partial H_p}\right]^- + K_{TSP} \left[\frac{\partial TSP}{\partial H_p}\right]^-}{\left[\frac{\partial d\beta}{\partial H_p}\right]^+ + K_{TSP} \left[\frac{\partial TSP}{\partial H_p}\right]^+} \quad (15)$$

$$\left[\frac{\partial d\beta}{\partial W_P} \right]^- = [(W_P H_P + W_H H_H)^{\beta-2} \odot X_{n\beta}] H'_P \quad (16)$$

$$\left[\frac{\partial d\beta}{\partial W_P} \right]^+ = [(W_P H_P + W_H H_H)^{\beta-1}] H'_P \quad (17)$$

$$\left[\frac{\partial d\beta}{\partial H_P} \right]^- = W'_P [(W_P H_P + W_H H_H)^{\beta-2} \odot X_{n\beta}] \quad (18)$$

$$\left[\frac{\partial d\beta}{\partial H_P} \right]^+ = W'_P [(W_P H_P + W_H H_H)^{\beta-1}] \quad (19)$$

$$\left[\frac{\partial SSM}{\partial W_P} \right]_{f, rp}^- = 2F \left[\frac{W_{P_{f-1, rp}} + W_{P_{f+1, rp}}}{\sum_{j=1}^F W_{P_{j, rp}}^2} \right] + \frac{2FW_{P_{f, rp}} \sum_{j=2}^F (W_{P_{j, rp}} - W_{P_{j-1, rp}})^2}{(\sum_{j=1}^F W_{P_{j, rp}}^2)^2} \quad (20)$$

$$\left[\frac{\partial SSM}{\partial W_P} \right]_{f, rp}^+ = \frac{4FW_{P_{f, rp}}}{\sum_{j=1}^F W_{P_{j, rp}}^2} \quad (21)$$

$$\left[\frac{\partial TSP}{\partial H_P} \right]_{rp, t}^+ = \sqrt{T} \frac{H_{P_{rp, t}} \sum_{i=1}^T H_{P_{rp, i}}}{(\sum_{i=1}^T H_{P_{rp, i}}^2)^{\frac{3}{2}}} \quad (22)$$

$$\left[\frac{\partial TSP}{\partial H_P} \right]_{rp, t}^- = \frac{1}{\sqrt{\frac{1}{T} \sum_{i=1}^T H_{P_{rp, i}}^2}} \quad (23)$$

Sustituyendo las bases armónicas W_H y de ganancia H_H dentro de la ecuación (11), se obtienen las siguientes reglas de actualización multiplicativas para sonidos armónicos.

$$W_H = W_H \odot \frac{\left[\frac{\partial d\beta}{\partial W_H} \right]^- + K_{SSP} \left[\frac{\partial SSP}{\partial W_H} \right]^-}{\left[\frac{\partial d\beta}{\partial W_H} \right]^+ + K_{SSP} \left[\frac{\partial SSP}{\partial W_H} \right]^+} \quad (24)$$

$$H_H = H_H \odot \frac{\left[\frac{\partial d\beta}{\partial H_H} \right]^- + K_{TSM} \left[\frac{\partial TSM}{\partial H_H} \right]^-}{\left[\frac{\partial d\beta}{\partial H_H} \right]^+ + K_{TSP} \left[\frac{\partial TSP}{\partial H_H} \right]^+} \quad (25)$$

$$\left[\frac{\partial d\beta}{\partial W_H} \right]^- = [(W_P H_P + W_H H_H)^{\beta-2} \odot X_n] H'_H \quad (26)$$

$$\left[\frac{\partial d\beta}{\partial W_H} \right]^+ = [(W_P H_P + W_H H_H)^{\beta-1}] H'_H \quad (27)$$

$$\left[\frac{\partial d\beta}{\partial H_H} \right]^- = W'_H [(W_P H_P + W_H H_H)^{\beta-2} \odot X_n] \quad (28)$$

$$\left[\frac{\partial d\beta}{\partial H_H} \right]^+ = W'_H [(W_P H_P + W_H H_H)^{\beta-1}] \quad (29)$$

$$\left[\frac{\partial TSM}{\partial H_H} \right]_{rh,t}^- = 2T \left[\frac{H_{H_{rh,t-1}} + H_{H_{rh,t+1}}}{\sum_{i=1}^T H_{H_{rh,t}}^2} \right] + \frac{2TH_{H_{rh,t}} \sum_{i=2}^T (H_{H_{rh,i}} - H_{H_{rh,i-1}})^2}{(\sum_{i=1}^T H_{H_{rp,i}}^2)^2} \quad (30)$$

$$\left[\frac{\partial TSM}{\partial H_H} \right]_{rh,t}^+ = \frac{4TH_{H_{rh,t}}}{\sum_{i=1}^T H_{H_{rh,i}}^2} \quad (31)$$

$$\left[\frac{\partial SSP}{\partial W_H} \right]_{f,rh}^- = \sqrt{F} \frac{W_{H_{f,rh}} \sum_{j=1}^F W_{H_{j,rh}}}{(\sum_{j=1}^F W_{H_{j,rh}}^2)^{\frac{3}{2}}} \quad (32)$$

$$\left[\frac{\partial SSP}{\partial H_P} \right]_{f,rh}^+ = \frac{1}{\sqrt{\frac{1}{F} \sum_{j=1}^F W_{H_{j,rh}}^2}} \quad (33)$$

Para la actualización iterativa de las matrices W_P , H_P , W_H y H_H , se usan *MaxIter* iteraciones, en las cuales las bases armónicas y percusivas van optimizando su información hasta que llegan a converger.

5.2.7.3 Segundo modelo basado en el algoritmo NMF

Además del primer modelo desarrollado, se usa una segunda modificación del mismo para el desarrollo de este TFG, véase [1], [2], [3], [4] y [5]. La motivación de este modelo es realizar la separación entre fuentes vocales instrumentales en aquellas zonas de la canción donde fuentes de excitaciones vocales e instrumentales La canción se procesa en estas zonas según está definido en la ecuación (39). Para ello, el algoritmo toma como parámetro de entrada la información temporal de la señal de entrada $t \in t_{inst} \cup t_{instvocal}$, las bases ya entrenadas generadas por el primer modelo y el espectrograma de entrada X , definido en la ecuación (1). En cuanto a las bases de la parte vocal de la canción, se incluyen W_{F0} y H_{F0} , correspondientes a las excitaciones vocales y sus activaciones, además de W_f y H_f , correspondientes a los filtros vocales y sus activaciones.

Las bases W_f y W_{F0} , son fijadas durante todo el proceso, por lo que no son optimizadas. Dado que las zonas instrumentales están desprovistas de fuentes vocales, se establece la información de las bases de activación temporal H_{F0} y H_f como nula en ellas, igual a cero. Se evita su optimización en estas zonas de la forma:

$$H_f(ph, t_{inst}) = 0 \quad (34)$$

$$H_{F0}(ft, t_{inst}) = 0 \quad (35)$$

Asimismo, las bases percusivas, W_P y H_P , y las bases armónicas, W_H y H_H , son sólo optimizadas en las zonas instrumentales/vocales, aprovechando de esta manera la información obtenida en el primer modelo.

Tras el cálculo de las distintas bases factorizadas antes mencionadas, se calculan los tres espectrogramas de salida, X_v , X_p y X_h , correspondientes a la parte vocal, percusiva y armónica respectivamente.

$$X_v = (W_f \cdot H_f) \odot (W_{F0} \cdot H_{F0}) \quad (36)$$

$$X_p = (W_P \cdot H_P) \quad (37)$$

$$X_h = (W_H \cdot H_H) \quad (38)$$

En cuanto al desarrollo de este segundo modelo, en la factorización del espectrograma X_n , se añaden los espectrogramas X_v y X_n .

$$\begin{aligned} X_{n\beta} = & X_v + X_p + X_h + X_n = \\ & (W_{f_{F,ph}} \cdot H_{f_{ph,T}}) * (W_{F0_{F,ft}} \cdot H_{F0_{ft,T}}) + W_{P_{F,Rp}} \cdot H_{P_{Rp,T}} + W_{H_{F,Rh}} \cdot H_{H_{Rh,T}} \\ & + W_{n_{F,rn}} \cdot H_{n_{rn,T}} \end{aligned} \quad (39)$$

Donde W_{F0} constituye la matriz de excitaciones vocales y H_{F0} contiene sus activaciones. Así mismo, W_f constituye la matriz de filtros vocales y H_f sus activaciones.

Esta separación vocal, se basa en un modelo excitación/filtro, en el cual se considera que para una señal de voz sonora producida por una excitación es dependiente de una frecuencia fundamental. Estas excitaciones o fonemas, son extraídos mediante el uso de un rango discreto de filtros correspondientes a un número limitado de posibles vocales pronunciadas en la voz principal. Bajo ciertos supuestos, podríamos considerar que cada filtro estimado corresponde a un determinado fonema. Se utiliza este modelo excitación/filtro, para determinar la trayectoria de la voz principal de la canción y, así, realizar la separación entre esta y el acompañamiento instrumental.

Las bases W_{F0} y W_f serán fijadas durante todo el proceso, mientras que las bases H_{F0} y H_f se irán actualizando mediante las reglas de actualización multiplicativas, al igual que las bases percusivas y armónicas. En esta nueva modificación, aunque se han implementado

las bases para la separación de ruido, no han sido usadas, dado que emporaban el resultado del algoritmo. La restricción asumida para estas nuevas bases es la suavidad espectral. Se definen por tanto las variables K_{SPHF0} y K_{SPHF} .

El cometido de esta segunda modificación, es separar la parte vocal de los sonidos armónicos/percusivos de forma más precisa, resultando pistas instrumentales más limpias de voz. Se define la nueva función global D como,

$$D = d\beta \left(X_{n\beta} \left| (X_p + X_h + X_V) \right. \right) + K_{SSM} . SSM + K_{TSP} . TSP + K_{SSP} . SSP \\ + K_{TSM} . TSM + K_{SPHF0} . SPHF0 + K_{SPHF} . SPHF \quad (40)$$

A continuación, se obtienen las siguientes reglas de actualización multiplicativas para la base HF0.

$$H_{F0} = H_{F0} \odot \frac{\left[\frac{\partial d\beta}{\partial H_{F0}} \right]^- + K_{SPHF0} \left[\frac{\partial SPHF0}{\partial H_{F0}} \right]^-}{\left[\frac{\partial d\beta}{\partial H_{F0}} \right]^+ + K_{SPHF0} \left[\frac{\partial SPHF0}{\partial H_{F0}} \right]^+} \quad (41)$$

$$\left[\frac{\partial d\beta}{\partial H_{F0}} \right]^- = W'_{F0} [(W_f H_f) \odot (W_{F0} H_{F0} + W_f H_f + W_P H_P + W_H H_H)^{\beta-2} \odot X_{n\beta}] \quad (42)$$

$$\left[\frac{\partial d\beta}{\partial H_{F0}} \right]^+ = W'_{F0} [(W_f H_f) \odot (W_{F0} H_{F0} + W_f H_f + W_P H_P + W_H H_H)^{\beta-2}] \quad (43)$$

$$\left[\frac{\partial SPHF0}{\partial H_{F0}} \right]_{ft,t}^- = \sqrt{T} \frac{H_{F0}_{ft,t} \sum_{i=1}^T H_{F0}_{ft,i}}{(\sum_{i=1}^T H_{F0}_{ft,i})^{\frac{3}{2}}} \quad (44)$$

$$\left[\frac{\partial SPHF0}{\partial H_{F0}} \right]_{ft,t}^+ = \frac{1}{\sqrt{\frac{1}{T} \sum_{j=1}^T H_{F0,ft,j}^2}} \quad (45)$$

$f_t \approx$ Número de excitaciones vocales usadas en la factorización

Finalmente, se obtiene las siguientes reglas de actualización multiplicativas para la base HF.

$$H_F = H_F \odot \frac{\left[\frac{\partial d\beta}{\partial H_F} \right]^- + K_{SPHF} \left[\frac{\partial SPHF}{\partial H_F} \right]^-}{\left[\frac{\partial d\beta}{\partial H_F} \right]^+ + K_{SPHF} \left[\frac{\partial SPHF}{\partial H_F} \right]^+} \quad (46)$$

$$\left[\frac{\partial d\beta}{\partial H_F} \right]^- = W'_F [(W_{F0}H_{F0}) \odot (W_{F0}H_{F0} + W_fH_f + W_PH_P + W_HH_H)^{\beta-2} \odot X_{n\beta}] \quad (47)$$

$$\left[\frac{\partial d\beta}{\partial H_{F0}} \right]^+ = W'_{F0} [(W_{F0}H_{F0}) \odot (W_{F0}H_{F0} + W_fH_f + W_PH_P + W_HH_H)^{\beta-2}] \quad (48)$$

$$\left[\frac{\partial SPHF}{\partial H_F} \right]_{ph,t}^- = \sqrt{T} \frac{H_{Fph,t} \sum_{i=1}^T H_{Fph,i}}{(\sum_{i=1}^T H_{Fph,i}^2)^{\frac{3}{2}}} \quad (49)$$

$$\left[\frac{\partial SPHF}{\partial H_F} \right]_{ph,t}^+ = \frac{1}{\sqrt{\frac{1}{T} \sum_{j=1}^T H_{Fph,j}^2}} \quad (50)$$

$p_h \approx$ Número de bases vocales usadas en la factorización

5.2.7.4 Reconstrucción de la señal

La señal percusiva $x_p(t)$ se sintetiza usando el espectrograma X_P como el producto de las bases factorizadas W_P y sus activaciones H_P . Para asegurar la reconstrucción, se genera una máscara M_P usando el filtro Wiener. La información de fase se calcula multiplicando la máscara por el espectrograma complejo X_c relacionado con la señal mezclada $x(t)$. Para la señal armónica $x_h(t)$ y $x_v(t)$ se sigue el mismo procedimiento.

$$M_P = \frac{X_P}{X_P + X_H + X_v} \quad (48)$$

$$M_H = \frac{X_H}{X_P + X_H + X_v} \quad (50)$$

$$M_v = \frac{X_v}{X_P + X_H + X_v} \quad (51)$$

$$x_p(t) = IDFT(M_P \cdot X_C) \quad (52)$$

$$x_h(t) = IDFT(M_h \cdot X_C) \quad (53)$$

$$x_v(t) = IDFT(M_v \cdot X_C) \quad (54)$$

CAPÍTULO 5 – MATERIALES Y MÉTODOS

CAPÍTULO 6

RESULTADOS Y DISCUSIÓN

6.1 Evaluación y resultados

Una vez el desarrollo del algoritmo ha sido acabado, se han realizado una serie de pruebas objetivas para comprobar la calidad de separación del mismo. Para ello se han tomado diversas pistas con una duración aproximada de 4 minutos de la base de datos QUASI, véase [12].

En la siguiente tabla se detallan las pistas seleccionadas para comprobar el funcionamiento correcto del algoritmo.

Etiqueta	Autor	Nombre Canción
E01	Another Dreamer	One We Love
E02	Fort Minor	Remember the name
E03	Glen Philips	The spirit of shackleton
E04	Jims Big Ego	Mix Tape
E05	Nine Inch Nails	Good Soldier
E06	Ultimate Nz Tour	-
E07	Vieux Farka	Ana

Tabla 6.1 – Base de datos usada

Las medidas tomadas para evaluar el rendimiento del algoritmo desarrollado son:

- **SDR (Source-to-distortion ratio):** Proporciona información de la calidad de separación del algoritmo.

$$SDR = 10 \log_{10} \frac{\|s_{true}\|^2}{\|e_{interf} + e_{noise} + e_{artif}\|^2}$$

- **SIR (Source-to-interferences ratio):** Proporciona una medida de la presencia de sonidos armónicos en las señal percusiva y viceversa.

$$\text{SIR} = 10 \log_{10} \frac{\|s_{true}\|^2}{\|e_{interf}\|^2}$$

- **SAR (Source-to-artifacts ratio):** Proporciona información de los artefactos en la señal separada debido al proceso de separación y de reconstrucción.

$$\text{SAR} = 10 \log_{10} \frac{\|s_{true} + e_{interf} + e_{noise}\|^2}{\|e_{artif}\|^2}$$

Donde, s_{true} es una matriz que contiene la imagen de la fuente original, e_{noise} es una matriz que contiene la componente espacial de distorsión, e_{interf} es una matriz que contiene la componente de interferencia y e_{artif} es una matriz que contiene la componente de artefactos.

Para tomar estas medidas, se han usado $N=1024$ muestras para la STFT así como $J=512$ muestra, que optimizan la relación entre rendimiento y costo computacional. Además, se ha tomado $\beta=1.5$ dado que se obtiene el mejor compromiso entre altos y bajos cambios de energía en frecuencia. Así mismo, se ha tomado el valor $K_{SM} = 0.4$ y el valor $K_{SP} = 0.1$; estos deben ser mucho menores que uno, dado que altos valores pueden producir problemas en la reconstrucción. Finalmente el número de bases percusivas y armónicas, se ha tomado como sigue, $R_h=150$ y $R_p=60$, ya que es necesario un número más alto de componentes armónicas que de componentes percusivas para mejorar la calidad de la separación.

Además, dado que no los resultados empeoraban optimizando la base W_f , referida a los filtros vocales, se ha optado utilizar una base W_f ya entrenada.

Así pues, las medidas obtenidas se presentan en la siguiente tabla. La unidad de estos datos es el decibelio dB. Por un lado, se ha comparado la canción solo vocal original y la canción solo vocal estimada, columna izquierda. Por otro lado, se ha comparado la canción solo instrumental original y la canción solo instrumental estimada, columna derecha.

TABLA DE RESULTADOS			
Etiqueta	SDR	SIR	SAR
E01	[2.5224, 8.8681]	[6.8784, 25.4750]	[4.8713, 9.3380]
E02	[4.3488, -12.3355]	[13.1928, 2.2750]	[4.0239, -21.6277]
E03	[0.0368, 2.5521]	[1.2604, 17.4966]	[7.0664, 4.2701]
E04	[2.6023, 6.2233]	[1.2249, 33.8512]	[5.9369, 8.3303]
E05	[-1.5339, 9.1088]	[3.0318, 18.2703]	[-0.2423, 9.6017]
E06	[-3.4525, 5.3232]	[-1.7961, 30.3734]	[4.4288, 6.4779]
E07	[2.8171, 7.6102]	[6.3143, 28.8857]	[6.0793, 8.5798]

Tabla 6.2 – Medidas obtenidas

Para la estimación instrumental, podemos comprobar como la mayoría de las pistas vienen determinadas por interferencia, es decir, se ha realizado una buena separación. Sin embargo, también se ha dado algún caso en el que la separación viene determinada por artefacto, es decir, se ha realizado una mala separación debido, principalmente, a que no se ha podido obtener una cantidad suficiente de información instrumental que ayude a realizar el proceso de separación correctamente.

A continuación se realizará una comparación las medidas SDR entre estos resultados y los obtenidos por otros algoritmos relacionados con NMF, en este caso, se han tomado los datos de la tesis doctoral de Augustin Lefèvre, véase [16], para el aprendizaje de métodos para la separación de fuentes de audio de un solo canal, para el cual se han usado cinco métodos distintos basados en NMF. Usaremos las medidas obtenidas con el método “Auto” dado que también se basa en el etiquetado del acompañamiento.

Etiqueta	SDR (Nuestro método)	Método de Augustin
E01	[2.5224,8.8681]	[-5.95,7.89]
E02	[4.3488;-12.3355]	[-3.98,4.54]
E03	[0.0368;2.5521]	[-4.66,8.63]
E04	[2.6023,6.2233]	[-10.51,12.36]
E05	[-1.5339,9.1088]	[-0.14,-0.26]
E06	[-3.4525,5.3232]	[-5.55,9.23]
E07	[2.8171,7.6102]	[-1.96,5.45]

Tabla 6.3 – Comparación de resultados entre distintos algoritmos

Podemos comprobar como en la mayoría de las canciones analizadas los resultados, aunque mejores en el caso del algoritmo desarrollado por Augustin, son muy cercanos a estos. Podemos considerar, por tanto, que nuestro algoritmo ofrece buenos resultados en la separación vocal/instrumentas.

Nuestro algoritmo presenta las siguientes ventajas; por un lado, no es necesario conocer el número de instrumentos que aparecen en la pista, por otro lado, no es necesario conocer el tipo de instrumento para realizar su clasificación entre los dos tipos de bases.

CAPÍTULO 6 – RESULTADOS Y DISCUSIÓN

CAPÍTULO 7

CONCLUSIONES

Finalmente, tras comprobar el correcto funcionamiento del algoritmo diseñado y, tras la puesta en marcha del servicio web en los servidores de la Escuela Politécnica Superior de Linares, se ha alcanzado con éxito el objetivo de este TFG; creando un servicio capaz de complementar al programa de Karaoke UltraStar que es capaz de acercar al mismo a otros programas de Karaoke comerciales que si incluyen la opción de suprimir de voz del cantante principal.

El servicio final implementado, se ha puesto en manos de los usuarios del programa UltraStar a través del foro oficial de este programa, en la siguiente dirección.

<http://ultrastar-es.org/foro/viewtopic.php?f=5&t=8196>

Durante el desarrollo de este TFG, se han detectado dos problemas principales, los cuales han sido abordados de la manera más apropiada posible.

El primer problema que se presentó esta relacionado con la API “Matlabcontrol”, utilizada para la comunicación entre el servidor web y el programa Matlab. Una vez establecida la comunicación entre ambos puede ocurrir, por cualquier circunstancia, que está se pierda. Si esto llega a ocurrir, la API no es capaz de volver a establecer comunicaciones posteriores a la sesión anteriormente fallida. Tras investigar los posibles motivos relacionadas con este problema, se ha llegado a la conclusión de que este guarda relación con el puerto asignado en la comunicación entre el servidor web y la interfaz JMI de Matlab que permite su comunicación con programas programados en JAVA. Sin embargo, tras una búsqueda exhaustiva de información, no se ha encontrado una posible solución que permite solventar este problema, el cual puede llegar a poner en riesgo el correcto funcionamiento del servicio implementado.

El segundo problema presentado, tiene relación con el algoritmo de separación entre fuentes vocales y fuentes instrumentales. Durante el procesado de ciertos archivos de audio, algunos fonemas han sido erróneamente clasificados por el algoritmo como parte de

fuentes instrumentales, en concreto, parte de los fonemas sordos han sido detectados como sonidos percusivos, mientras que parte de los fonemas sonoros han sido detectados como sonidos armónicos. Esto es debido a que dichos fonemas presentan características espectro-temporales similares a los tipos de sonidos antes mencionados. Ante este problema, se han tomado distintas medidas pero ninguna sin resultado satisfactorio. Por tanto se requiere una nueva modificación del algoritmo NMF que permita distinguir entre fonemas sordos y sonoros que solvete este problema.

CAPÍTULO 7 – CONCLUSIÓN

CAPÍTULO 8

LINEAS DE FUTURO

En el siguiente apartado se enumeran las líneas de futuro a seguir en una futura continuación del trabajo realizado en este TFG.

- Extensión del tipo de ficheros de audio compatible para su procesado.
- Modificación del algoritmo NMF con restricciones de dispersión y suavidad que permita distinguir entre fonemas sordos y fonemas sonoros para solventar el problema presentado en este TFG.
- Solventar el problema relacionado con la API “MatlabControl”.
- Extensión de este servicio a otros tipos de software de Karaoke.
- Capacidad de procesamiento simultaneo de varios archivos de audio dentro de una misma sesión.
- Extensión del tipo de contenedores usados para entregar al usuario su canción procesada, ya sea RAR, 7ZIP, etc.
- Creación de un servidor alojado en la nube que proporcione canciones ya procesadas y que puedan ser usadas por otros usuarios.
- Modificación del código fuente del programa UltraStar que permita modificar el nivel de voz cantada.

CAPÍTULO 8 – LINEAS DE FUTURO

PARTE II

PLIEGO DE CONDICIONES

CAPÍTULO 9

PLIEGO DE CONDICIONES

En este capítulo se comentan los requisitos mínimos de hardware y software necesarios para el correcto funcionamiento del servicio implementado.

9.1 Requisitos Hardware

- PC con S.O. Windows 7 o superior, con memoria RAM de 4GB, disco duro de 500GB y procesador Intel Core i5 Quad Core a 2,5GHz.
- PC con S.O. Ubuntu 12.04 o superior, con memoria RAM de 4GB, disco duro de 500GB y procesador Intel Core i5 Quad Core a 2,5GHz.
- PC con S.O. Mac OS X 10.6.4 o superior con memoria RAM de 4GB, disco duro de 500GB y procesador Intel Core i5 Quad Core a 2,5GHz.
- Conexión a internet de 20MB de bajada y 1 MB de subida.

Como podemos comprobar el servicio puede ser instalado en un PC que utilice cualquiera de los sistemas operativos populares en la actualidad. Dado que no requiere una gran capacidad de procesamiento, no se necesita un equipo demasiado potente, aunque se establece un mínimo de 4GB de memoria RAM y un procesador de cuatro núcleos para minimizar el tiempo medio de procesamiento, que se encuentra entorno a cuatro minutos.

9.2 Requisitos Software

Es necesario disponer en dicho PC del servidor web Apache Tomcat 7 o superior, Matlab 2012a o superior y el kit de desarrollo JDK 8 de Java compatibles con el S.O. instalado en el equipo.

CAPÍTULO 9 – PLIEGO DE CONDICIONES

PARTE III

PRESUPUESTO

CAPÍTULO 10

PRESUPUESTO

10.1 Recursos humanos

En la tabla siguiente se realiza una valoración económica de los recursos humanos utilizados en el desarrollo de este TFG.

Concepto	Horas	Precio Unitario (euros)	Precio Total (euros)
Análisis y búsqueda de información	20 Horas	35	700
Diseño general del servicio	10 Horas	35	350
Desarrollo del Servicio	100 Horas	35	3500
Puesta en marcha y Pruebas	20 Horas	35	700
Desarrollo de la documentación	40 Horas	35	1400
Total	190 Horas	35	6650

Tabla 10.1 – Presupuesto de recursos humanos

10.2 Recursos materiales

En esta tabla se refleja el importe del software y hardware adquiridos.

Concepto	Uds.	Precio Unitario (euros)	Precio Total (euros)
Toshiba L855-11K	1	655	655
Microsoft Windows 7 Profesional	1	125	125
Ubuntu 14.04	1	0	0
JDK y JRE	1	0	0
Eclipse Kepler	1	0	0
API Matlabcontrol	1	1	0
Apache Tomcat 7.0.14	1	0	0
Matlab 2013a	1	35	35
Total			815

Tabla 10.2 – Presupuesto de recursos materiales

10.3 Presupuesto final

Finalmente, el presupuesto final aparece en la siguiente tabla.

Concepto	
Recursos Humanos	6650€
Recursos Materiales	815€
Total	7465€
Precio final = Total + I.V.A (21%)	9032,65€

Tabla 10.3 – Presupuesto final

PARTE IV

ANEXOS

CAPÍTULO 11

MANUAL DE USUARIO

11.1 Guía de utilización del servicio

Para acceder al servicio, el usuario debe introducir la siguiente dirección en la barra de direcciones de su navegador web, <http://anclas3.ujaen.es:8080/Server1>, disponible en lengua hispana y lengua inglesa. En primer lugar, al acceder a este sitio web, accederemos a la página principal. Encontramos un mensaje en la parte superior donde se nos indica cual es el cometido de este sitio. Seguido, aparece una barra horizontal donde aparecen los accesos a las distintas secciones en las que está dividida y un par de botones que permiten cambiar el idioma de la página entre inglés y español, con el fin de que los usuarios no hispanohablantes puedan comprender el contenido de la misma.

A continuación se describirá brevemente cada una de las secciones del sitio web.

- **Página principal:** En esta sección, se ha incrustado un mensaje de bienvenida, así como una imagen representativa del programa UltraStar.



Figura 11.1 – Sección Principal

- **Procesa tu canción:** Es en esta sección, donde el usuario deberá seleccionar tanto el archivo de texto y el archivo de audio a procesar, para ser enviados al servidor.

Figura 11.2 – Sección “Procesa tu canción”

- **Acerca de:** Se describe el cometido de la página.

Figura 11.3 – Sección “Acerca de”

Dentro de la sección “Procesa tu canción”, aparecerá un formulario con dos entradas bien diferenciadas. Por un lado, encontramos la opción llamada “Selecciona tu archivo WAV o MP3”, en la que el usuario deberá seleccionar el archivo de audio a procesar. Por otro lado, encontramos otra opción diferente, “Selecciona tu archivo TXT”, en la cual el usuario deberá adjuntar el archivo de texto que acompaña su canción. Una vez ambos archivos han sido seleccionados, el usuario deberá presionar en el botón “Procesar”. Mientras se transcurre el procesado de los archivos, aparecerá un ícono “loading” para indicar que el proceso se está ejecutando. Para el correcto funcionamiento del servicio, es indispensable que el usuario envíe ambos archivos, de otra forma, el servicio no podrá funcionar correctamente.

Una vez terminado el proceso, se cargará una nueva página en el navegador. El usuario deberá seleccionar sobre el apartado “Descarga tu canción procesada”. En ese momento, se descargará un archivo ZIP, cuyo nombre no coincide con el de la canción original. La siguiente figura muestra esta página.



Figura 11.4 – Página de descarga

Finalmente, deberá descomprimir el archivo que se encuentra en su interior, de mismo nombre y formato que la canción original, dentro de la carpeta donde se ubicó la misma, sobrescribiéndola si fuese necesario.

CAPÍTULO 11 – MANUAL DE USUARIO

CAPÍTULO 12

MANUAL TÉCNICO

12.1 *Instalaciones previas*

Dado que tanto el desarrollo del servicio como la implementación se han realizado bajo el uso del S.O. Ubuntu 14.04, en esta guía se describirá los distintos pasos a seguir para la correcta instalación del software necesario para el correcto funcionamiento del servicio web.

Será necesario abrir una ventana del terminal de comandos. En ella debemos introducir los comandos necesarios para realizar la instalación del siguiente software, JDK 8 de JAVA, Apache Tomcat 7 y Codificador LAME.

En la siguiente figura, se muestra como acceder a dicho terminal desde este S.O.



Figura 12.1 - Terminal de Ubuntu

12.1.1 Instalación JDK 8

Para la instalación de este software, véase [15], debemos seguir los siguientes pasos, sin alterar su orden. Su uso es imprescindible para que el servidor Apache pueda funcionar.

- 1 - usuario@X:~\$ sudo add-apt-repository ppa:webupd8team/java
- 2 - usuario@X:~\$ sudo apt-get update
- 3 - usuario@X:~\$ sudo apt-get install oracle-java8-installer

Aparecerá la siguiente ventana, en la cuál deberemos aceptar todos las condiciones y simplemente seleccionar siempre siguiente.

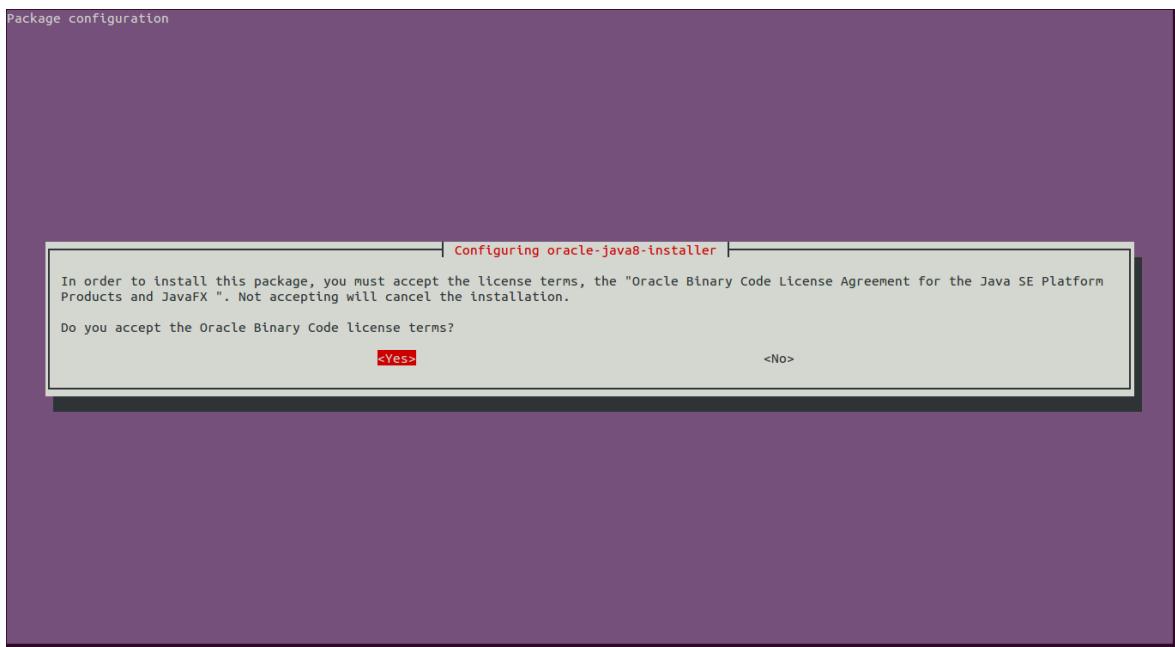


Figura 12.2 – Instalación JDK

Una vez finalizado el proceso, podremos comprobar si la instalación a concluido correctamente. Para ello, introduciremos el siguiente comando.

- 4 - usuario@X:~\$ java -version

Si en el sistema se encuentran instaladas varias versiones de JDK, debemos seleccionar la ultima versión, en este caso V8. Así pues, se introduce el siguiente comando en el terminal.

```
5 - usuario@X:~$ sudo update-java-alternatives -s java-8-oracle
```

Por último, es necesario seleccionar las variables de entorno del sistema que se corresponda con el JDK 8. Se realiza con el comando:

```
6 - usuario@X:~$ sudo apt-get install oracle-java8-set-default
```

12.1.2 Instalación de Apache Tomcat 7

En anteriores capítulos, ya se ha mencionado que la aplicación que actuará como servidor web dentro del equipo será Apache Tomcat, véase [14]. Bajo el S.O. Ubuntu 14.04, la instalación del mismo requiere una serie de pasos y posteriores configuraciones para su correcto funcionamiento.

Los comandos necesarios para la instalación de Apache Tomcat son,

```
1 - usuario@X:~$ wget http://archive.apache.org/dist/tomcat/tomcat-7/v7.0.47/bin/apache-tomcat-7.0.47.tar.gz
```

```
2 - usuario@X:~$ tar -xvf apache-tomcat-7.0.47.tar.gz
```

```
3 - usuario@X:~$ sudo mv apache-tomcat-7.0.47/usr/local/
```

Tras la instalación de apache, crearemos un script que permita poner en marcha el servidor. Para generarla y poder editarla, debemos introducir las siguientes líneas.

```
4 - usuario@X:~$ sudo vim /etc/init.d/tomcat747
```

```
5 - usuario@X:~$ gedit /etc/init.d/tomcat747
```

Este script debe contener la siguiente información.

```
#!/bin/bash
export CATALINA_HOME=/usr/local/apache-tomcat-7.0.47
PATH=/sbin:/bin:/usr/sbin:/usr/bin
start() {
    sh $CATALINA_HOME/bin/startup.sh
}
stop() {
    sh $CATALINA_HOME/bin/shutdown.sh
}
case $1 in
    start|stop) $1;;
    restart) stop; start;;
    *) echo "Run as $0 <start|stop|restart>"; exit 1;;
esac
```

Una vez realizada esta tarea, necesitamos modificar los permisos de dicho script para que pueda funcionar de manera adecuada.

```
6 - usuario@X:~$ sudo chmod 755 /etc/init.d/tomcat747
```

Con objeto de instalar nuestra sitio web y nuestro Servlet en su interior, es necesario definir el usuario y contraseña del administrador del servicio. Para ellos, dentro de la ventana terminal, introducimos el siguiente comando.

```
7 - usuario@X:~$ gedit /usr/local/apache-tomcat-7.0.47/conf/tomcat-users.xml
```

Aparecerá una nueva ventana en la que debemos introducir la siguiente información,

```
<role rolename="manager-gui"/>
<role rolename="admin-gui"/>
<user username="admin" password="admin" roles="manager-gui,admin-gui"/>
```

También, necesitamos que en caso de reinicio del equipo, el servidor web vuelva a su funcionamiento, para ello se establece su inicio en el arranque del S.O.

```
8 - usuario@X:~$ sudo update-rc.d tomcat747 defaults
```

Así pues, una vez hemos acabado, tendremos tres opciones para usar en el terminal que nos permitirán actuar sobre el servidor Apache, que nos permitirá iniciarla, pararla o resetearlo

```
9 - usuario@X:~$ sudo /etc/init.d/tomcat747 start
10 - usuario@X:~$ sudo /etc/init.d/tomcat747 stop
11 - usuario@X:~$ sudo /etc/init.d/tomcat747 restart
```

Finalmente, podemos acceder al servidor Apache desde la barra de direcciones de nuestro navegador web, accediendo a la dirección: <http://localhost:8080/>. Aparecerá una ventana como la siguiente.

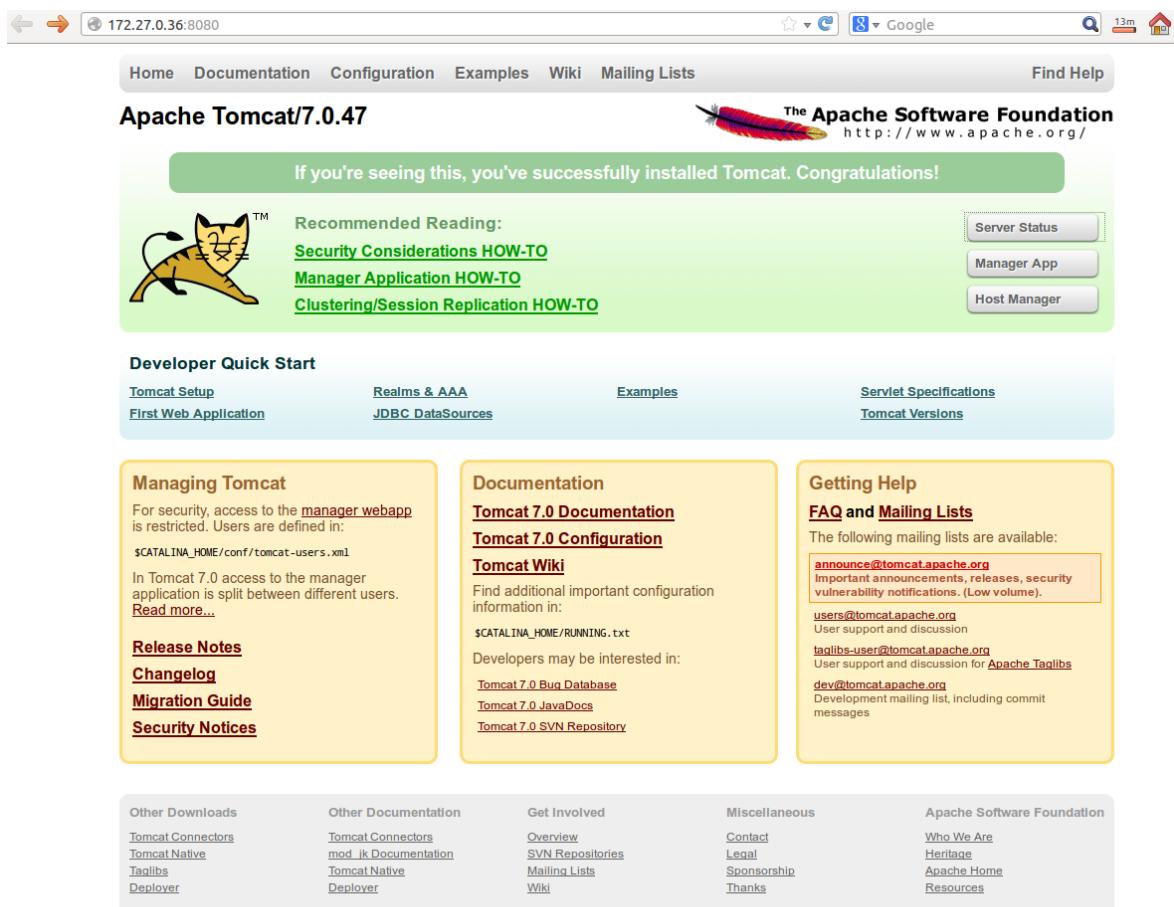


Figura 12.3 – Página principal Apache Tomcat

Una vez en esta ventana debemos seleccionar “Host Manager”, apareciendo la siguiente ventana.



Tomcat Web Application Manager

Message:

Manager						
List Applications		HTML Manager Help		Manager Help		Server Status
Applications						
Path	Version	Display Name	Running	Sessions	Commands	
/	None specified	Welcome to Tomcat	true	0	<input type="button" value="Start"/> <input type="button" value="Stop"/> <input type="button" value="Reload"/> <input type="button" value="Undeploy"/>	<input type="button" value="Expire sessions with idle ≥ 30 minutes"/>
/Server1	None specified		true	0	<input type="button" value="Start"/> <input type="button" value="Stop"/> <input type="button" value="Reload"/> <input type="button" value="Undeploy"/>	<input type="button" value="Expire sessions with idle ≥ 30 minutes"/>
/docs	None specified	Tomcat Documentation	true	0	<input type="button" value="Start"/> <input type="button" value="Stop"/> <input type="button" value="Reload"/> <input type="button" value="Undeploy"/>	<input type="button" value="Expire sessions with idle ≥ 30 minutes"/>
/examples	None specified	Servlet and JSP Examples	true	0	<input type="button" value="Start"/> <input type="button" value="Stop"/> <input type="button" value="Reload"/> <input type="button" value="Undeploy"/>	<input type="button" value="Expire sessions with idle ≥ 30 minutes"/>
/host-manager	None specified	Tomcat Host Manager Application	true	0	<input type="button" value="Start"/> <input type="button" value="Stop"/> <input type="button" value="Reload"/> <input type="button" value="Undeploy"/>	<input type="button" value="Expire sessions with idle ≥ 30 minutes"/>
/manager	None specified	Tomcat Manager Application	true	1	<input type="button" value="Start"/> <input type="button" value="Stop"/> <input type="button" value="Reload"/> <input type="button" value="Undeploy"/>	<input type="button" value="Expire sessions with idle ≥ 30 minutes"/>

Deploy					
Deploy directory or WAR file located on server					
Context Path (required): <input type="text"/> XML Configuration file URL: <input type="text"/> WAR or Directory URL: <input type="text"/> <input type="button" value="Deploy"/>					
WAR file to deploy					
Select WAR file to upload <input type="button" value="Seleccionar archivo"/> nada seleccionado <input type="button" value="Deploy"/>					

Figura 12.4 – Host Manager

En el apartado “WAR file to deploy”, deberemos seleccionar nuestro proyecto generado con Eclipse. Tras esto, seleccionamos “Deploy”. Nuestro servicio estará casi listo para su correcto funcionamiento.

A continuación necesitamos, dar permisos a la carpeta donde se guardaran los archivos procesados por el algoritmo. En este TFG, se ha realizado con el siguiente comando.

```
12 - usuario@X:~$ sudo chmod ugo+rwx /usr/local/apache-tomcat-7.0.47/webapps/Server1/
```

Finalmente, debemos ubicar la carpeta contenedora con los ficheros en formato Matlab, que contiene el algoritmo desarrollado, en el directorio que se seleccionó para su ubicación.

```
“/home/paguilar/Documentos/ficherosmatlab”
```

12.1.3 – Instalación codificador LAME

Para poder codificar los archivos procesados en formato MP3, se ha utilizado el codificador LAME dentro del propio terminal del S.O. Ubuntu. Para su instalación, solamente necesitamos ejecutar el siguiente comando.

```
1 - usuario@X:~$ sudo apt-get install lame
```

12.2 Documentación técnica

Class

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#)

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

org.apache.commons.fileupload.servlet

Class **ServletFileUpload**

```
java.lang.Object
  org.apache.commons.fileupload.FileUploadBase
    org.apache.commons.fileupload.FileUpload
      org.apache.commons.fileupload.servlet.ServletFileUpload
```

```
public class ServletFileUpload
extends FileUpload
```

High level API for processing file uploads.

This class handles multiple files per single HTML widget, sent using multipart/mixed encoding type, as specified by RFC 1867. Use `parseRequest(HttpServletRequest)` to acquire a list of `FileItems` associated with a given HTML widget.

How the data for individual parts is stored is determined by the factory used to create them; a given part may be in memory, on disk, or somewhere else.

Version:

\$Id: ServletFileUpload.java 1455949 2013-03-13 14:14:44Z simonetripodi \$

Nested Class Summary

Nested classes/interfaces inherited from class org.apache.commons.fileupload.FileUploadBase

```
FileUploadBase.FileSizeLimitExceededException, FileUploadBase.FileUploadIOException,
FileUploadBase.InvalidContentTypeException, FileUploadBase.IOFileUploadException,
FileUploadBase.SizeException, FileUploadBase.SizeLimitExceededException,
FileUploadBase.UnknownSizeException
```

Field Summary

Fields inherited from class org.apache.commons.fileupload.FileUploadBase

```
ATTACHMENT, CONTENT_DISPOSITION, CONTENT_LENGTH, CONTENT_TYPE, FORM_DATA, MAX_HEADER_SIZE,
MULTIPART, MULTIPART_FORM_DATA, MULTIPART_MIXED
```

Constructor Summary

Constructors

Constructor and Description

ServletFileUpload()

Constructs an uninitialised instance of this class.

ServletFileUpload(FileItemFactory fileItemFactory)

Constructs an instance of this class which uses the supplied factory to create `FileItem` instances.

Method Summary

Modifier and Type	Method and Description
<code>FileItemIterator</code>	<code>getIterator(HttpServletRequest request)</code> Processes an RFC 1867 compliant multipart/form-data stream.
<code>static boolean</code>	<code>isMultipartContent(HttpServletRequest request)</code> Utility method that determines whether the request contains multipart content.
<code>Map<String, List<FileItem>></code>	<code>parseParameterMap(HttpServletRequest request)</code> Processes an RFC 1867 compliant multipart/form-data stream.
<code>List<FileItem></code>	<code>parseRequest(HttpServletRequest request)</code> Processes an RFC 1867 compliant multipart/form-data stream.

Methods inherited from class org.apache.commons.fileupload.FileUpload

`getFileItemFactory, setFileItemFactory`

Methods inherited from class org.apache.commons.fileupload.FileUploadBase

`createItem, getBoundary, getFieldName, getFieldName, getFileName, getFileName, getFileSizeMax, getHeader, getHeaderEncoding, getItemIterator, getParsedHeaders, getProgressListener, getSizeMax, isMultipartContent, newFileItemHeaders, parseHeaders, parseParameterMap, parseRequest, setSizeMax, setHeaderEncoding, setProgressListener, setSizeMax`

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

ServletFileUpload

`public ServletFileUpload()`

Constructs an uninitialised instance of this class. A factory must be configured, using `setFileItemFactory()`, before attempting to parse requests.

See Also:

`FileUpload.FileUpload(FileItemFactory)`

ServletFileUpload

`public ServletFileUpload(FileItemFactory fileItemFactory)`

Constructs an instance of this class which uses the supplied factory to create `FileItem` instances.

Parameters:

`fileItemFactory` - The factory to use for creating file items.

See Also:

`FileUpload.FileUpload()`

Method Detail

isMultipartContent

```
public static final boolean isMultipartContent(HttpServletRequest request)
```

Utility method that determines whether the request contains multipart content.

Parameters:

request - The servlet request to be evaluated. Must be non-null.

Returns:

true if the request is multipart; false otherwise.

parseRequest

```
public List<FileItem> parseRequest(HttpServletRequest request)
                           throws FileUploadException
```

Processes an RFC 1867 compliant multipart/form-data stream.

Overrides:

parseRequest in class FileUploadBase

Parameters:

request - The servlet request to be parsed.

Returns:

A list of FileItem instances parsed from the request, in the order that they were transmitted.

Throws:

FileUploadException - if there are problems reading/parsing the request or storing files.

parseParameterMap

```
public Map<String,List<FileItem>> parseParameterMap(HttpServletRequest request)
                           throws FileUploadException
```

Processes an RFC 1867 compliant multipart/form-data stream.

Parameters:

request - The servlet request to be parsed.

Returns:

A map of FileItem instances parsed from the request.

Throws:

FileUploadException - if there are problems reading/parsing the request or storing files.

Since:

1.3

getitemiterator

```
public FileItemIterator getItemIterator(HttpServletRequest request)
    throws FileUploadException,
    IOException
```

Processes an RFC 1867 compliant multipart/form-data stream.

Parameters:

request - The servlet request to be parsed.

Returns:

An iterator to instances of `FileItemStream` parsed from the request, in the order that they were transmitted.

Throws:

`FileUploadException` - if there are problems reading/parsing the request or storing files.

`IOException` - An I/O error occurred. This may be a network error while communicating with the client or a problem while storing the uploaded content.

Class

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#)

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

org.apache.commons.fileupload.disk

Class DiskFileItemFactory

java.lang.Object
 org.apache.commons.fileupload.disk.DiskFileItemFactory

All Implemented Interfaces:

FileItemFactory

Direct Known Subclasses:

DefaultFileItemFactory

```
public class DiskFileItemFactory
extends Object
implements FileItemFactory
```

The default `FileItemFactory` implementation. This implementation creates `FileItem` instances which keep their content either in memory, for smaller items, or in a temporary file on disk, for larger items. The size threshold, above which content will be stored on disk, is configurable, as is the directory in which temporary files will be created.

If not otherwise configured, the default configuration values are as follows:

- Size threshold is 10KB.
- Repository is the system default temp directory, as returned by `System.getProperty("java.io.tmpdir")`.

NOTE: Files are created in the system default temp directory with predictable names. This means that a local attacker with write access to that directory can perform a TOUTOC attack to replace any uploaded file with a file of the attackers choice. The implications of this will depend on how the uploaded file is used but could be significant. When using this implementation in an environment with local, untrusted users, `setRepository(File)` MUST be used to configure a repository location that is not publicly writable. In a Servlet container the location identified by the `ServletContext` attribute `javax.servlet.context.tempdir` may be used.

Temporary files, which are created for file items, should be deleted later on. The best way to do this is using a `FileCleaningTracker`, which you can set on the `DiskFileItemFactory`. However, if you do use such a tracker, then you must consider the following: Temporary files are automatically deleted as soon as they are no longer needed. (More precisely, when the corresponding instance of `File` is garbage collected.) This is done by the so-called reaper thread, which is started and stopped automatically by the `FileCleaningTracker` when there are files to be tracked. It might make sense to terminate that thread, for example, if your web application ends. See the section on "Resource cleanup" in the users guide of commons-fileupload.

Since:

FileUpload 1.1

Version:

\$Id: DiskFileItemFactory.java 1564788 2014-02-05 14:36:41Z markt \$

Field Summary

Fields

Modifier and Type	Field and Description
static int	<code>DEFAULT_SIZE_THRESHOLD</code> The default threshold above which uploads will be stored on disk.

Constructor Summary

Constructor and Description

`DiskFileItemFactory()`

Constructs an unconfigured instance of this class.

`DiskFileItemFactory(int sizeThreshold, File repository)`

Constructs a preconfigured instance of this class.

Method Summary

Modifier and Type	Method and Description
<code>FileItem</code>	<code>createItem(String fieldName, String contentType, boolean isFormField, String fileName)</code> Create a new <code>DiskFileItem</code> instance from the supplied parameters and the local factory configuration.
<code>org.apache.commons.io.FileCleaningTracker</code>	<code>getFileCleaningTracker()</code> Returns the tracker, which is responsible for deleting temporary files.
<code>File</code>	<code>getRepository()</code> Returns the directory used to temporarily store files that are larger than the configured size threshold.
<code>int</code>	<code>getSizeThreshold()</code> Returns the size threshold beyond which files are written directly to disk.
<code>void</code>	<code>setFileCleaningTracker(org.apache.commons.io.FileCleaningTracker pTracker)</code> Sets the tracker, which is responsible for deleting temporary files.
<code>void</code>	<code>setRepository(File repository)</code> Sets the directory used to temporarily store files that are larger than the configured size threshold.
<code>void</code>	<code>setSizeThreshold(int sizeThreshold)</code> Sets the size threshold beyond which files are written directly to disk.

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Field Detail

DEFAULT_SIZE_THRESHOLD

`public static final int DEFAULT_SIZE_THRESHOLD`

The default threshold above which uploads will be stored on disk.

See Also:

[Constant Field Values](#)

Constructor Detail

DiskFileItemFactory

```
public DiskFileItemFactory()
```

Constructs an unconfigured instance of this class. The resulting factory may be configured by calling the appropriate setter methods.

DiskFileItemFactory

```
public DiskFileItemFactory(int sizeThreshold,  
                           File repository)
```

Constructs a preconfigured instance of this class.

Parameters:

`sizeThreshold` - The threshold, in bytes, below which items will be retained in memory and above which they will be stored as a file.

`repository` - The data repository, which is the directory in which files will be created, should the item size exceed the threshold.

Method Detail

getRepository

```
public File getRepository()
```

Returns the directory used to temporarily store files that are larger than the configured size threshold.

Returns:

The directory in which temporary files will be located.

See Also:

```
setRepository(java.io.File)
```

setRepository

```
public void setRepository(File repository)
```

Sets the directory used to temporarily store files that are larger than the configured size threshold.

Parameters:

repository - The directory in which temporary files will be located.

See Also:

```
getRepository()
```

getSizeThreshold

```
public int getSizeThreshold()
```

Returns the size threshold beyond which files are written directly to disk. The default value is 10240 bytes.

Returns:

The size threshold, in bytes.

See Also:

```
setSizeThreshold(int)
```

setSizeThreshold

```
public void setSizeThreshold(int sizeThreshold)
```

Sets the size threshold beyond which files are written directly to disk.

Parameters:

sizeThreshold - The size threshold, in bytes.

See Also:

```
getSizeThreshold()
```

createItem

```
public FileItem createItem(String fieldName,
                           String contentType,
                           boolean isFormField,
                           String fileName)
```

Create a new `DiskFileItem` instance from the supplied parameters and the local factory configuration.

Specified by:

`createItem` in interface `FileItemFactory`

Parameters:

`fieldName` - The name of the form field.
`contentType` - The content type of the form field.
`isFormField` - true if this is a plain form field; false otherwise.
`fileName` - The name of the uploaded file, if any, as supplied by the browser or other client.

Returns:

The newly created file item.

getFileCleaningTracker

```
public org.apache.commons.io.FileCleaningTracker getFileCleaningTracker()
```

Returns the tracker, which is responsible for deleting temporary files.

Returns:

An instance of `FileCleaningTracker`, or null (default), if temporary files aren't tracked.

setFileCleaningTracker

```
public void setFileCleaningTracker(org.apache.commons.io.FileCleaningTracker pTracker)
```

Sets the tracker, which is responsible for deleting temporary files.

Parameters:

`pTracker` - An instance of `FileCleaningTracker`, which will from now on track the created files, or null (default), to disable tracking.

java.io

Class File

[java.lang.Object](#) [java.io.File](#)

All Implemented Interfaces:

[Serializable](#), [Comparable<File>](#)

Field Summary

static String	pathSeparator The system-dependent path-separator character, represented as a string for convenience.
static char	pathSeparatorChar The system-dependent path-separator character.
static String	separator The system-dependent default name-separator character, represented as a string for convenience.
static char	separatorChar The system-dependent default name-separator character.

Constructor Summary

[File\(File parent, String child\)](#)

Creates a new `File` instance from a parent abstract pathname and a child pathname string.

[File\(String pathname\)](#)

Creates a new `File` instance by converting the given pathname string into an abstract pathname.

[File\(String parent, String child\)](#)

Creates a new `File` instance from a parent pathname string and a child pathname string.

`File(URI uri)`

Creates a new `File` instance by converting the given `file:` URI into an abstract pathname.

Method Summary

boolean	<code>canExecute()</code> Tests whether the application can execute the file denoted by this abstract pathname.
boolean	<code>canRead()</code> Tests whether the application can read the file denoted by this abstract pathname.
boolean	<code>canWrite()</code> Tests whether the application can modify the file denoted by this abstract pathname.
int	<code>compareTo(File pathname)</code> Compares two abstract pathnames lexicographically.
boolean	<code>createNewFile()</code> Atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.
static <code>File</code>	<code>createTempFile(String prefix, String suffix)</code> Creates an empty file in the default temporary-file directory, using the given prefix and suffix to generate its name.
static <code>File</code>	<code>createTempFile(String prefix, String suffix, File directory)</code> Creates a new empty file in the specified directory, using the given prefix and suffix strings to generate its name.
boolean	<code>delete()</code> Deletes the file or directory denoted by this abstract pathname.
void	<code>deleteOnExit()</code> Requests that the file or directory denoted by this abstract pathname be deleted when the virtual machine terminates.

boolean	equals(Object obj) Tests this abstract pathname for equality with the given object.
boolean	exists() Tests whether the file or directory denoted by this abstract pathname exists.
File	getAbsoluteFile() Returns the absolute form of this abstract pathname.
String	getAbsolutePath() Returns the absolute pathname string of this abstract pathname.
File	getCanonicalFile() Returns the canonical form of this abstract pathname.
String	getCanonicalPath() Returns the canonical pathname string of this abstract pathname.
long	getFreeSpace() Returns the number of unallocated bytes in the partition named by this abstract path name.
String	getName() Returns the name of the file or directory denoted by this abstract pathname.
String	getParent() Returns the pathname string of this abstract pathname's parent, or <code>null</code> if this pathname does not name a parent directory.
File	getParentFile() Returns the abstract pathname of this abstract pathname's parent, or <code>null</code> if this pathname does not name a parent directory.
String	getPath() Converts this abstract pathname into a pathname string.
long	getTotalSpace() Returns the size of the partition named by this abstract pathname.
long	getUsableSpace() Returns the number of bytes available to this virtual

	machine on the partition named by this abstract pathname.
int hashCode()	Computes a hash code for this abstract pathname.
boolean isAbsolute()	Tests whether this abstract pathname is absolute.
boolean isDirectory()	Tests whether the file denoted by this abstract pathname is a directory.
boolean isFile()	Tests whether the file denoted by this abstract pathname is a normal file.
boolean isHidden()	Tests whether the file named by this abstract pathname is a hidden file.
long lastModified()	Returns the time that the file denoted by this abstract pathname was last modified.
long length()	Returns the length of the file denoted by this abstract pathname.
String[] list()	Returns an array of strings naming the files and directories in the directory denoted by this abstract pathname.
String[] list(FilenameFilter filter)	Returns an array of strings naming the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.
File[] listFiles()	Returns an array of abstract pathnames denoting the files in the directory denoted by this abstract pathname.
File[] listFiles(FileFilter filter)	Returns an array of abstract pathnames denoting the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.
File[] listFiles(FilenameFilter filter)	Returns an array of abstract pathnames denoting the

	files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.
static <code>File</code> []	<code>listRoots()</code> List the available filesystem roots.
boolean	<code>mkdir()</code> Creates the directory named by this abstract pathname.
boolean	<code>mkdirs()</code> Creates the directory named by this abstract pathname, including any necessary but nonexistent parent directories.
boolean	<code>renameTo(File dest)</code> Renames the file denoted by this abstract pathname.
boolean	<code>setExecutable(boolean executable)</code> A convenience method to set the owner's execute permission for this abstract pathname.
boolean	<code>setExecutable(boolean executable, boolean ownerOnly)</code> Sets the owner's or everybody's execute permission for this abstract pathname.
boolean	<code>setLastModified(long time)</code> Sets the last-modified time of the file or directory named by this abstract pathname.
boolean	<code>setReadable(boolean readable)</code> A convenience method to set the owner's read permission for this abstract pathname.
boolean	<code>setReadable(boolean readable, boolean ownerOnly)</code> Sets the owner's or everybody's read permission for this abstract pathname.
boolean	<code>setReadOnly()</code> Marks the file or directory named by this abstract pathname so that only read operations are allowed.
boolean	<code>setWritable(boolean writable)</code> A convenience method to set the owner's write permission for this abstract pathname.
boolean	<code>setWritable(boolean writable, boolean ownerOnly)</code> Sets the owner's or everybody's write permission for this abstract pathname.
<code>String</code>	<code>toString()</code> Returns the pathname string of this abstract pathname.

URI	toURI() Constructs a <code>file:</code> URI that represents this abstract pathname.
URL	toURL() Deprecated. <i>This method does not automatically escape characters that are illegal in URLs. It is recommended that new code convert an abstract pathname into a URL by first converting it into a URI, via the toURI method, and then converting the URI into a URL via the URI.toURL method.</i>

Methods inherited from class java.lang.Object

[clone](#), [finalize](#), [getClass](#), [notify](#), [notifyAll](#), [wait](#), [wait](#), [wait](#)

Class

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)
Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

org.apache.commons.fileupload

Interface FileItem

All Superinterfaces:
FileItemHeadersSupport, Serializable

All Known Implementing Classes:
DefaultFileItem, DiskFileItem

```
public interface FileItem
extends Serializable, FileItemHeadersSupport
```

This class represents a file or form item that was received within a multipart/form-data POST request.

After retrieving an instance of this class from a `FileUpload` instance (see `#parseRequest(javax.servlet.http.HttpServletRequest)`), you may either request all contents of the file at once using `get()` or request an `InputStream` with `getInputStream()` and process the file without attempting to load it into memory, which may come handy with large files.

While this interface does not extend `javax.activation.DataSource` per se (to avoid a seldom used dependency), several of the defined methods are specifically defined with the same signatures as methods in that interface. This allows an implementation of this interface to also implement `javax.activation.DataSource` with minimal additional work.

Since:
1.3 additionally implements FileItemHeadersSupport

Version:
\$Id: FileItem.java 1454690 2013-03-09 12:08:48Z simonetripodi \$

Method Summary

Modifier and Type	Method and Description
void	<code>delete()</code> Deletes the underlying storage for a file item, including deleting any associated temporary disk file.
byte[]	<code>get()</code> Returns the contents of the file item as an array of bytes.
String	<code>getContentType()</code> Returns the content type passed by the browser or <code>null</code> if not defined.
String	<code>getFieldName()</code> Returns the name of the field in the multipart form corresponding to this file item.
InputStream	<code>getInputStream()</code> Returns an <code>InputStream</code> that can be used to retrieve the contents of the file.
String	<code>getName()</code> Returns the original filename in the client's filesystem, as provided by the browser (or other client software).
OutputStream	<code>getOutputStream()</code> Returns an <code>OutputStream</code> that can be used for storing the contents of the file.
long	<code>getSize()</code> Returns the size of the file item.
String	<code>getString()</code> Returns the contents of the file item as a String, using the default character encoding.
String	<code>getString(String encoding)</code> Returns the contents of the file item as a String, using the specified encoding.
boolean	<code>isFormField()</code> Determines whether or not a <code>FileItem</code> instance represents a simple form field.
boolean	<code>isInMemory()</code> Provides a hint as to whether or not the file contents will be read from memory.
void	<code>setFieldName(String name)</code> Sets the field name used to reference this file item.
void	<code>setFormField(boolean state)</code> Specifies whether or not a <code>FileItem</code> instance represents a simple form field.
void	<code>write(File file)</code> A convenience method to write an uploaded item to disk.

Methods inherited from interface org.apache.commons.fileupload.FileItemHeadersSupport

`getHeaders, setHeaders`

Method Detail

getInputStream

```
InputStream getInputStream()  
throws IOException
```

Returns an `InputStream` that can be used to retrieve the contents of the file.

Returns:

An `InputStream` that can be used to retrieve the contents of the file.

Throws:

`IOException` - if an error occurs.

getContentType

```
String getContentType()
```

Returns the content type passed by the browser or `null` if not defined.

Returns:

The content type passed by the browser or `null` if not defined.

getName

```
String getName()
```

Returns the original filename in the client's filesystem, as provided by the browser (or other client software). In most cases, this will be the base file name, without path information. However, some clients, such as the Opera browser, do include path information.

Returns:

The original filename in the client's filesystem.

Throws:

`InvalidFileNameException` - The file name contains a NUL character, which might be an indicator of a security attack. If you intend to use the file name anyways, catch the exception and use `InvalidFileNameException#getName()`.

isInMemory

```
boolean isInMemory()
```

Provides a hint as to whether or not the file contents will be read from memory.

Returns:

`true` if the file contents will be read from memory; `false` otherwise.

getSize

```
long getSize()
```

Returns the size of the file item.

Returns:

The size of the file item, in bytes.

get

```
byte[] get()
```

Returns the contents of the file item as an array of bytes.

Returns:

The contents of the file item as an array of bytes.

getString

```
String getString(String encoding)
    throws UnsupportedEncodingException
```

Returns the contents of the file item as a String, using the specified encoding. This method uses `get()` to retrieve the contents of the item.

Parameters:

`encoding` - The character encoding to use.

Returns:

The contents of the item, as a string.

Throws:

`UnsupportedEncodingException` - if the requested character encoding is not available.

getString

```
String getString()
```

Returns the contents of the file item as a String, using the default character encoding. This method uses `get()` to retrieve the contents of the item.

Returns:

The contents of the item, as a string.

write

```
void write(File file)
    throws Exception
```

A convenience method to write an uploaded item to disk. The client code is not concerned with whether or not the item is stored in memory, or on disk in a temporary location. They just want to write the uploaded item to a file.

This method is not guaranteed to succeed if called more than once for the same item. This allows a particular implementation to use, for example, file renaming, where possible, rather than copying all of the underlying data, thus gaining a significant performance benefit.

Parameters:

`file` - The `File` into which the uploaded item should be stored.

Throws:

`Exception` - if an error occurs.

delete

```
void delete()
```

Deletes the underlying storage for a file item, including deleting any associated temporary disk file. Although this storage will be deleted automatically when the `FileItem` instance is garbage collected, this method can be used to ensure that this is done at an earlier time, thus preserving system resources.

getFieldName

```
String getFieldName()
```

Returns the name of the field in the multipart form corresponding to this file item.

Returns:

The name of the form field.

setFieldName

```
void setFieldName(String name)
```

Sets the field name used to reference this file item.

Parameters:

name - The name of the form field.

isFormField

```
boolean isFormField()
```

Determines whether or not a `FileItem` instance represents a simple form field.

Returns:

true if the instance represents a simple form field; false if it represents an uploaded file.

setFormField

```
void setFormField(boolean state)
```

Specifies whether or not a `FileItem` instance represents a simple form field.

Parameters:

state - true if the instance represents a simple form field; false if it represents an uploaded file.

getOutputStream

```
OutputStream getOutputStream()  
throws IOException
```

Returns an `OutputStream` that can be used for storing the contents of the file.

Returns:

An `OutputStream` that can be used for storing the contents of the file.

Throws:

`IOException` - if an error occurs.

Class MatlabProxyFactory

java.lang.Object └ **matlabcontrol.MatlabProxyFactory**

```
public class MatlabProxyFactory
extends java.lang.Object
```

Creates instances of [MatlabProxy](#). Any number of proxies may be created with a factory.

How the proxies will connect to a session of MATLAB depends on whether the factory is running inside or outside MATLAB:

Running inside MATLAB

The proxy will connect to the session of MATLAB this factory is running in.

Running outside MATLAB

By default a new session of MATLAB will be started and connected to, but the factory may be configured via the options provided to this factory to connect to a previously controlled session.

This class is unconditionally thread-safe. Any number of proxies may be created simultaneously.

Since:

4.0.0

Nested Class Summary

static interface	MatlabProxyFactory.Request A request for a proxy.
static interface	MatlabProxyFactory.RequestCallback Provides the requested proxy.

Constructor Summary

[MatlabProxyFactory\(\)](#)

Constructs the factory using default options.

[MatlabProxyFactory\(MatlabProxyFactoryOptions options\)](#)

Constructs the factory with the specified `options`.

Method Summary

MatlabProxy	getProxy() Returns a MatlabProxy .
MatlabProxyFactory .Request	requestProxy(MatlabProxyFactory.RequestCallback callback) Requests a MatlabProxy .

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

MatlabProxyFactory

`public MatlabProxyFactory()`

Constructs the factory using default options.

Throws:

[MatlabConnectionException](#)

MatlabProxyFactory

`public MatlabProxyFactory(MatlabProxyFactoryOptions options)`

Constructs the factory with the specified `options`. Depending on the whether the factory is running inside MATLAB or outside MATLAB will determine if a given option is used.

Parameters:

`options` -

Method Detail

getProxy

`public MatlabProxy getProxy()`

throws

[MatlabConnectionException](#)

Returns a [MatlabProxy](#). If a connection cannot be established before the timeout then this method will end execution and an exception will be thrown. A timeout can be specified with the options provided to this factory. If no timeout was specified, then a default of 180 seconds will be used.

While this method blocks the calling thread until a proxy is created (or the timeout is reached), any number of threads may call `getProxy()` simultaneously.

Returns:

`proxy`

Throws:

[MatlabConnectionException](#)

```
requestProxy
```

```
public MatlabProxyFactory.Request requestProxy(MatlabProxyFactory.RequestCallback
    callback) throws
MatlabConnectionException
```

Requests a [MatlabProxy](#). When the proxy has been created it will be provided to the `callback`. The proxy may be provided to the callback before this method returns. There is no timeout. The returned [MatlabProxyFactory.Request](#) instance provides information about the request and can be used to cancel the request.

This method is non-blocking. Any number of requests may be made simultaneously from the same thread or different threads.

Returns:

request

Throws:

[MatlabConnectionException](#)

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [NESTED](#) | FIELD | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#)

DETAIL: FIELD | [CONSTR](#) | [METHOD](#)

matlabcontrol

Class MatlabProxyFactoryOptions

`java.lang.Object`

matlabcontrol.MatlabProxyFactoryOptions

```
public class MatlabProxyFactoryOptions
extends java.lang.Object
```

Options that configure how a factory operates. Use a [MatlabProxyFactoryOptions.Builder](#) to create an instance of this class.

This class is unconditionally thread-safe.

Since:

4.0.0

See Also:

[MatlabProxyFactory.MatlabProxyFactory\(matlabcontrol.MatlabProxyFact](#)
[ryOptions\)](#)

Nested Class Summary

static class	MatlabProxyFactoryOptions.Builder
	Creates instances of MatlabProxyFactoryOptions .

Method Summary

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

matlabcontrol

Class MatlabProxyFactoryOptions.Builde r

java.lang.Object [matlabcontrol.MatlabProxyFactoryOptions.Builder](#)
Enclosing class:
[MatlabProxyFactoryOptions](#)

public static class MatlabProxyFactoryOptions.Builder
extends java.lang.Object

Creates instances of [MatlabProxyFactoryOptions](#). Any and all of these properties may be left unset, if so then a default will be used. Depending on how the factory operates, not all properties may be used. Currently all properties are used only when running outside MATLAB, but future releases may add additional options.

Calls on this class may be chained together to easily create factory options.
Example usage:

```
MatlabProxyFactoryOptions options = new
MatlabProxyFactoryOptions.Builder()
.setHidden(true)
.setProxyTimeout(30000L)
.build();
```

This class is unconditionally thread-safe.

Since:

4.0.0

Constructor Summary

[`MatlabProxyFactoryOptions.Builder\(\)`](#)

Method Summary

<code>MatlabProxyFactoryOptions</code>	<code>build()</code> Builds a MatlabProxyFactoryOptions instance.
<code>MatlabProxyFactoryOptions.Builder</code>	<code>setHidden(boolean hidden)</code> Sets whether MATLAB should appear hidden.
<code>MatlabProxyFactoryOptions.Builder</code>	<code>setJavaDebugger(int portnumber)</code> Sets whether to enable use of the Java debugger on the MATLAB session using port <code>portnumber</code> .
<code>MatlabProxyFactoryOptions.Builder</code>	<code>setLicenseFile(java.lang.String licenseFile)</code>) Sets the license file used by MATLAB.
<code>MatlabProxyFactoryOptions.Builder</code>	<code>setLogFile(java.lang.String logfile)</code> Sets whether to have MATLAB log any output to the MATLAB Command Window (including crash reports) to the file specified by <code>logfile</code> .
<code>MatlabProxyFactoryOptions.Builder</code>	<code>setMatlabLocation(java.lang.String matlabLocation)</code> Sets the location of the MATLAB executable or script that will launch MATLAB.
<code>MatlabProxyFactoryOptions.Builder</code>	<code>setMatlabStartingDirectory(java.io.File dir)</code> Sets the starting directory for MATLAB.
<code>MatlabProxyFactoryOptions.Builder</code>	<code>setPort(int port)</code> Sets the port matlabcontrol uses to communicate with MATLAB.
<code>MatlabProxyFactoryOptions.Builder</code>	<code>setProxyTimeout(long timeout)</code> Sets the amount of time in milliseconds to wait for a proxy to be created when requested via the blocking method <code>MatlabProxyFactory.getProxy()</code> .

<code>MatlabProxyFactoryOptions.Builder</code>	<code>setUsePreviouslyControlledSession(boolean usePreviouslyControlled)</code> Sets whether the factory should attempt to create a proxy that is connected to a running session of MATLAB.
<code>MatlabProxyFactoryOptions.Builder</code>	<code>setUseSingleComputationalThread(boolean useSingleCompThread)</code> Sets whether to limit MATLAB to a single computational thread.

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

MatlabProxyFactoryOptions.Builder
`public MatlabProxyFactoryOptions.Builder()`

Method Detail

setMatlabLocation
`public final MatlabProxyFactoryOptions.Builder setMatlabLocation(java.lang.String matlabLocation)`
 Sets the location of the MATLAB executable or script that will launch MATLAB. If the value set cannot be successfully used to launch MATLAB, an exception will be thrown when attempting to create a proxy.

The absolute path to the MATLAB executable can be determined by running MATLAB. On OS X or Linux, evaluate `[matlabroot '/bin/matlab']` in the Command Window. On Windows, evaluate `[matlabroot '/bin/matlab.exe']` in the Command Window. The location provided does not have to be an absolute path so long as the operating system can resolve the path.

Windows

Locations relative to the following will be understood:

- The current working directory
- The `Windows` directory only (no subdirectories are searched)
- The `Windows\System32` directory
- Directories listed in the `PATH` environment variable

- App Paths defined in the registry with
`key HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths`

By default on Windows, MATLAB adds itself to the `PATH` environment variable as well as places an App Path entry in the registry so that `matlab` can be used to launch MATLAB. If this property is not set, then the `PATH` environment variable or the App Path entry will be used.

OS X

Locations relative to the following will be understood:

- The current working directory
- Directories listed in the `PATH` environment variable

On OS X, MATLAB is installed in `/Applications/` as an application bundle. If this property is not set, the executable inside of the application bundle will be used.

Linux

Locations relative to the following will be understood:

- The current working directory
- Directories listed in the `PATH` environment variable

During the installation process on Linux, MATLAB can create a symbolic link named `matlab` that can be used to launch MATLAB. If this property is not set, this symbolic link will be used.

Parameters:

`matlabLocation -`

setMatlabStartingDirectory

```
public final
MatlabProxyFactoryOptions.Builder setMatlabStartingDirectory(java.io.File
dir)
```

Sets the starting directory for MATLAB.

Parameters:

`dir -`

Throws:

`java.lang.NullPointerException` - if `dir` is null

`java.lang.IllegalArgumentException` - if `dir` does not exist or is not a directory

setHidden

```
public final MatlabProxyFactoryOptions.Builder setHidden(boolean hidden)
```

Sets whether MATLAB should appear hidden. By default this property is set to `false`. If set to `true` then the splash screen will not be shown and:

Windows

The MATLAB Command Window will appear fully minimized.

OS X

MATLAB will be entirely hidden. The MATLAB session will terminate when the Java application terminates.

Linux

MATLAB will be entirely hidden. The MATLAB session will terminate when the Java application terminates.

Parameters:

`hidden` -

setLogFile

```
public final  
MatlabProxyFactoryOptions.Builder setLogFile(java.lang.String logfile)
```

Sets whether to have MATLAB log any output to the MATLAB Command Window (including crash reports) to the file specified by `logfile`. The validity of `logfile` is not checked by `matlabcontrol`. By default output is not logged.

Parameters:

`logfile` -

setJavaDebugger

```
public final  
MatlabProxyFactoryOptions.Builder setJavaDebugger(int portnumber)
```

Sets whether to enable use of the Java debugger on the MATLAB session using port `portnumber`. The `portnumber` may be in the range 0-65535 so long as it is not reserved or otherwise in use. By default the Java debugger is not enabled.

Parameters:

`portnumber` -

Throws:

`java.lang.IllegalArgumentException` - if `portnumber` is not in the range 0-65535

setLicenseFile

```
public final  
MatlabProxyFactoryOptions.Builder setLicenseFile(java.lang.String licensefile)
```

Sets the license file used by MATLAB. By default no license file is specified. On Linux and OS X `licensefile` may have the form `port@host` or a colon-separated list of license filenames. On

Windows `licenseFile` may have the form `port@host`. Setting this option causes the `LM_LICENSE_FILE` and `MLM_LICENSE_FILE` environment variables to be ignored. The validity of `licenseFile` is not checked by `matlabcontrol`.

Parameters:

`licenseFile` -

setUsePreviouslyControlledSession

```
public final  
MatlabProxyFactoryOptions.Builder setUsePreviouslyControlledSession(boolean  
n usePreviouslyControlled)
```

Sets whether the factory should attempt to create a proxy that is connected to a running session of MATLAB. By default this property is set to `false`.

When this property is `true` all options which configure MATLAB such as being hidden or logging are ignored. The only criterion used is whether a session of MATLAB is available for connection. In order for the factory to connect to the session of MATLAB, it must know about the session. This will be the case if a factory started the session of MATLAB and that factory was configured to use the same port as specified by [setPort\(int\)](#) (or both are using the default port). The factory will only connect to a session that does not currently have a proxy controlling it from outside of MATLAB.

To determine if the proxy created is connected to an existing session of MATLAB call [MatlabProxy.isExistingSession\(\)](#). You may wish to clear MATLAB's environment using `clear`. Doing so will not in anyway interfere with `matlabcontrol` (including executing `clear java`).

If a running session of MATLAB previously loaded classes defined in the controlling application, issues can arise. If your application does send to MATLAB or retrieve from MATLAB custom `Serializable` OR `Remote` classes then these issues do not apply.

MATLAB sessions launched by `matlabcontrol` are able to load classes defined in the controlling application. When an existing session of MATLAB is connected to by a newly controlling application it will now be able to load classes defined by the newly controlling application but not the previous one. Several problems may arise due to this behavior. If an attempt is made to use a class defined in a previously controlling session that was not loaded while the application was controlling MATLAB then it will fail with a `ClassNotFoundException` if it is not also defined in the newly controlling application. If the class is defined it will fail to load it if the

serializable definition is not compatible. A similar issue is if the newly controlling application attempts to send to MATLAB an instance of a class that was also defined by the previously controlling application but the serializable definition is not compatible. These above issues can easily be encountered when developing an application while changing `Serializable` or `Remote` classes and using the same session of MATLAB repeatedly. This will particularly be the case if the classes do not define a `serialVersionUID`. If multiple instances of the same application do not vary in their definition of `Serializable` and `Remote` classes then connecting to a previously controlled session of MATLAB will not cause any issues in this regard.

Parameters:

`usePreviouslyControlled` -

setProxyTimeout

```
public final MatlabProxyFactoryOptions.BuildersetProxyTimeout(long timeout)
```

Sets the amount of time in milliseconds to wait for a proxy to be created when requested via the blocking method `MatlabProxyFactory.getProxy()`. By default this property is set to 180000 milliseconds.

Parameters:

`timeout` -

Throws:

`java.lang.IllegalArgumentException` - if `timeout` is negative

setUseSingleComputationalThread

```
public final MatlabProxyFactoryOptions.BuildersetUseSingleComputationalThread(boolean useSingleCompThread)
```

Sets whether to limit MATLAB to a single computational thread. By default this property is set to `false`.

Parameters:

`useSingleCompThread` -

setPort

```
public final MatlabProxyFactoryOptions.BuildersetPort(int port)
```

Sets the port matlabcontrol uses to communicate with MATLAB. By default port 2100 is used. The port value may not be negative. It is recommended to be in the range of 1024 to 49151, but this range is not enforced. The port should be otherwise unused; however, any number of `MatlabProxyFactory` instances (even those running in completely separate applications) may use the same port.

A `MatlabProxyFactory` will only be able to connect to a previously controlled running session that was started by a factory using the same port.

Parameters:

port -

Throws:

`java.lang.IllegalArgumentException` - if port is negative

build

public final [MatlabProxyFactoryOptions](#)**build()**

Builds a `MatlabProxyFactoryOptions` instance.

Returns:

factory options

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#)

[FRAMES](#) [NO FRAMES](#)

DETAIL: FIELD | CONSTR | [METHOD](#)

matlabcontrol

Interface MatlabOperations

All Known Subinterfaces:

[MatlabProxy.MatlabThreadProxy](#)

All Known Implementing Classes:

[LoggingMatlabProxy](#), [MatlabProxy](#)

public interface **MatlabOperations**

Operations which interact with a session of MATLAB.

WARNING: This interface is not intended to be implemented by users of `matlabcontrol`. Methods may be added to this interface, and these additions will not be considered breaking binary compatibility.

Since:

4.1.0

Method Summary

void	eval (java.lang.String command) Evaluates a command in MATLAB.
void	feval (java.lang.String functionName, java.lang.Object... args) Calls a MATLAB function with the name <code>functionName</code> , returning the result.
java.lang.Object	getVariable (java.lang.String variableName)

	Gets the value of <code>variableName</code> in MATLAB.
<code>java.lang.Object[]</code>	<code>returningEval(java.lang.String command, int nargout)</code> Evaluates a command in MATLAB, returning the result.
<code>java.lang.Object[]</code>	<code>returningFeval(java.lang.String functionName, int nargout, java.lang.Object... args)</code> Calls a MATLAB function with the name <code>functionName</code> , returning the result.
<code>void</code>	<code>setVariable(java.lang.String variableName, java.lang.Object value)</code> Sets <code>variableName</code> to <code>value</code> in MATLAB, creating the variable if it does not yet exist.

Method Detail

eval

```
void eval(java.lang.String command) throws
MatlabInvocationException
```

Evaluates a command in MATLAB. This is equivalent to MATLAB's `eval('command')`.

Parameters:

`command` - the command to be evaluated in MATLAB

Throws:

[MatlabInvocationException](#)

returningEval

```
java.lang.Object[] returningEval(java.lang.String command,  
int nargout) throws
MatlabInvocationException
```

Evaluates a command in MATLAB, returning the result. This is equivalent to MATLAB's `eval('command')`.

In order for the result of this command to be returned the number of arguments to be returned must be specified by `nargout`. This is equivalent in MATLAB to the number of variables placed on the left hand side of an expression. For example, in MATLAB the `inmem` function may be used with either 1, 2, or 3 return values each resulting in a different behavior:

```
M = inmem; [M, X] = inmem; [M, X, J] = inmem;
```

The returned `Object` array will be of length `nargout` with each return argument placed into the corresponding array position.

If the command cannot return the number of arguments specified by `nargout` then an exception will be thrown.

Parameters:

command - the command to be evaluated in MATLAB

nargout - the number of arguments that will be returned from evaluating command

Returns:

result of MATLAB command, the length of the array will be nargout

Throws:

[MatlabInvocationException](#)

feval

```
void feval(java.lang.String functionName,  
           java.lang.Object... args) throws MatlabInvocationException
```

Calls a MATLAB function with the name `functionName`, returning the result. Arguments to the function may be provided as `args`, but are not required if the function needs no arguments.

The function arguments will be converted into MATLAB equivalents as appropriate. Importantly, this means that `astring` will be converted to a MATLAB `char` array, not a variable name.

Parameters:

`functionName` - the name of the MATLAB function to call

`args` - the arguments to the function

Throws:

[MatlabInvocationException](#)

returningFeval

```
java.lang.Object[] returningFeval(java.lang.String functionName,  
                                  int nargout, java.lang.Object... args)  
throws MatlabInvocationException
```

Calls a MATLAB function with the name `functionName`, returning the result. Arguments to the function may be provided as `args`, but are not required if the function needs no arguments.

The function arguments will be converted into MATLAB equivalents as appropriate. Importantly, this means that `astring` will be converted to a MATLAB `char` array, not a variable name.

In order for the result of this function to be returned the number of arguments to be returned must be specified by `nargout`. This is equivalent in MATLAB to the number of variables placed on the left hand side of an expression. For example, in MATLAB the `inmem` function may be used with either 1, 2, or 3 return values each resulting in a different behavior:

```
M = inmem; [M, X] = inmem; [M, X, J] = inmem;
```

The returned `object` array will be of length `nargout` with each return argument placed into the corresponding array position.

If the function is not capable of returning the number of arguments specified by `nargout` then an exception will be thrown.

Parameters:

`functionName` - the name of the MATLAB function to call

`nargout` - the number of arguments that will be returned

by `functionName`

`args` - the arguments to the function

Returns:

result of MATLAB function, the length of the array will be `nargout`

Throws:

[MatlabInvocationException](#)

setVariable

```
void setVariable(java.lang.String variableName,  
java.lang.Object value) throws MatlabInvocationException
```

Sets `variableName` to `value` in MATLAB, creating the variable if it does not yet exist.

Parameters:

`variableName` -

`value` -

Throws:

[MatlabInvocationException](#)

getVariable

```
java.lang.Object getVariable(java.lang.String variableName)  
throws MatlabInvocationException
```

Gets the value of `variableName` in MATLAB.

Parameters:

`variableName` -

Returns:

`value`

Throws:

[MatlabInvocationException](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#)

[FRAMES](#) [NO FRAMES](#)

DETAIL: FIELD | CONSTR | [METHOD](#)

CAPÍTULO 12 – MANUAL TÉCNICO

PARTE V

BIBLIOGRAFÍA

REFERENCIAS BIBLIOGRAFÍCAS

- [1] Canadas Quesada, F.J., Vera Candeas, P., Ruiz Reyes, N., et al. *PERCUSSIVE/HARMONIC SOUND SEPARATION BY NON-NEGATIVE MATRIX APPROXIMATION WITH SMOOTHNESS/SPARNESS CONSTRAINTS*. Journal of New Music Research, 2012 .
- [2] Durrieu, J.L., Thiran, J.P. *A Graphical User Interface For Musical Audio Source Separation*. Ecole Polytechnique Fédérale de Lausanne. 2012.
- [3] Durrieu, J.L., Richard, G., David, B. & Févotte, C. *Source/Filter Model for Unsupervised Main Melody Extraction From Polyphonic Audio Signals*. 2009, 2-7.
- [4] Durrieu, J.L. & Thiran, J.P. *Musical Audio Source Separation Base on User-Selected F0 Track*. Tel-Aviv, Israel, 2012, 1-5.
- [5] Ryyynänen, M., Virtanen T., Paulus J., & Klapuri A. *ACCOMPANIMENT SEPARATION AND KARAOKE APPLICATION BASED ON AUTOMATIC MELODY TRANSCRIPTION*. Tampere University of Technology, Institute of Signal Processing, 2008.
- [6] García, J., Rodríguez, J.I., et al. *Aprenda Java como si estuviere en primero*. Tecnum, Sebastián. 2000.
- [7] *API Matlabcontrol 4.1.0*. (2013). <http://code.google.com/p/matlabcontrol>.
- [8] *Apache Tomcat*. (2014). <http://tomcat.apache.org>
- [9] *Apache Commons FileUpload*. (2014).
<http://commons.apache.org/proper/commons-fileupload/>

- [10] *UltraStar Deluxe*. (2014). <http://ultrastardx.sourceforge.net>
- [11] Gauchat, J.D. *El gran libro de HTML5, CSS3 y Javascript*. 2013.
- [12] *QUASI Database – A musical audio signal Database for Source Separation*. (2012).
<http://www.tsi.telecom-paristech.fr/aao/en/2012/03/12/quasi/>
- [13] *Eclipse Kepler* (2013). <http://www.eclipse.org/kepler/>
- [14] *Instalación de Apache Tomcat 7 bajo S.O. Ubuntu*. (2014).
<http://www.krizna.com/ubuntu/install-tomcat-7-ubuntu-12-04/>
- [15] *Instalación de JDK 8 bajo S.O. Ubuntu*. (2014).
<http://www.enqlu.com/2014/03/how-to-install-oracle-java-78-jdk-and.html>
- [16] Augustin Lefèvre. *Dictionary learning methods for single-channel audio source separation*. Tesis doctoral. Octubre 2012.