Escola Politècnica Superior de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL DE FI DE CARRERA

TÍTOL DEL TFC : Diseño e Implementación de Módulos para un Sistema de TV P2P I

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Telemàtica

AUTOR: Lluís Martínez Recasens

DIRECTOR: Sergio Machado Sánchez

DATA: 27 de abril de 2008

Títol : Diseño e Implementación de Módulos para un Sistema de TV P2P I

Autor: Lluís Martínez Recasens

Director: Sergio Machado Sánchez

Data: 27 de abril de 2008

Resum

El trabajo de fin de carrera trata del diseño e implementación de módulos de un Sistema P2P para la transmisión de video desarrollado en Java. Los módulos desarrollados corresponden a la parte reproductora y transcodificadora del sistema, y para ello se ha implementado una librería Java que incluye un reproductor utilizable en aplicaciones con interfaz de usuario gráfica y un acceso nativo a los funcionalidades de reproducción y transcodificación. El VideoLAN se caracteriza por reproducir video y audio como cualquier reproductor, y además, permite hacer envíos de flujo de video desde una máquina hacia otra para luego poderlo reproducir localmente o difundirlo hacia otra máquina.
Este estudio permite añadir funcionalidades típicas de un reproductor a un Sistema P2P. Las aplicaciones desarrolladas en Java, que dependan de un reproductor externo para visualizar el video que se esté bajando desde la red, permitiría que en cualquiera de estas aplicaciones se pudiera incrustar el VideoLAN sin necesidad de otro reproductor para ver el vídeo.

Title : Modules Design and Implementation for a P2P System I

Author: Lluís Martínez Recasens

Director: Sergio Machado Sánchez

Date: April 27, 2008

Overview

The following report relates to the design and the implementation of modules of a P2P System for the transmission of videos developed in Java. The developed modules belong to the previewing and transcodification parts of the system. To do so, a Java library has been implemented. This one contains a player that can be used in applications with graphical user interface and a native access to the functions of previewing and transcodification. The VideoLan can preview video and audio as any player. Moreover, it can send streams of videos from one computer to another, so these can be played localhost or sent to another computer.
This report enables to add the functions of a player to a P2P system. Usually, the applications developed in Java depend on an external player to view a video downloaded from the Internet. But, with VideoLAN set in any of these applications, the external player will not be needed.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE CUADROS

INTRODUCCIÓN

La distribución de contenidos en vivo mediante redes Peer-to-Peer (P2P) de *streaming* es una aplicación emergente en Internet de la que se espera sea el próximo tráfico dominante de las comunicaciones IP [1]. Existen varios sistemas que proporcionan este servicio tales como CoolStreaming [2], PPLive [3] y SopCast [4].

En el departamento de Ingeniería Telemática de *l'Escola Politécnica Superior de Castelldefels* se está desarrollando uno de estos sistemas utilizando para la implementación el lenguaje de programación Java. Dicho sistema de transmisión de vídeo consta básicamente de un servidor de canal, que acomoda el flujo de vídeo original (una película almacenada en un DVD, un canal de televisión TDT, una webcam personal, etc.) a unas unidades que se intercambiarán los usuarios que se conecten a la red P2P. Obsérvese que el usuario no sólo es un mero reproductor de información sino que participa activamente en la difusión de la misma.

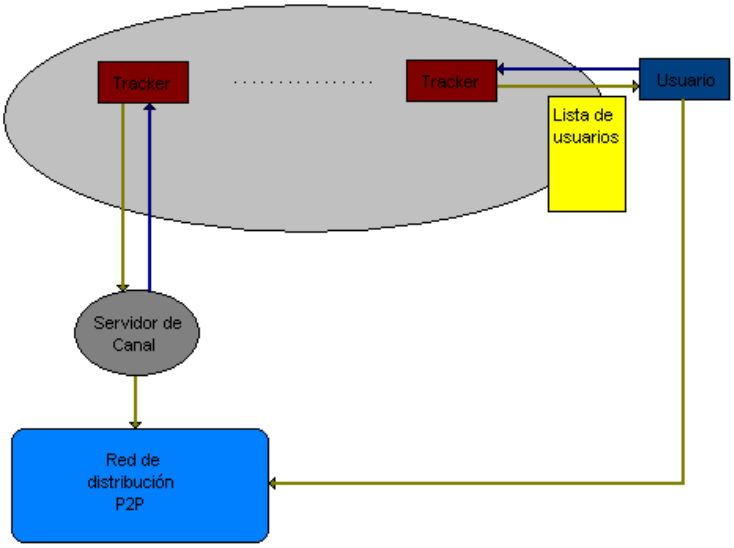

Figura 1: Red P2P

En la Figura 1 se muestran los componentes del sistema de distribución en desarrollo. Un peer accede al sistema a través de un componente denominado *tracker*[1]. El *tracker* posee una lista de canales que se han dado de alta en el servicio y, opcionalmente, una base de datos de usuarios registrados por si se desease implementar algún servicio que requiriese dicha información como, por ejemplo, un sistema de pago por visión. Una vez que el usuario escoge un canal se conecta a la red de distribución P2P formada para tal fin.

[1]La traducción en castellano del término *tracker* sería rastreador aunque preferimos utilizar la nomenclatura inglesa por su amplio uso.

La transmisión del flujo de vídeo entre los peers de la red de distribución P2P a nivel de protocolos de transporte, puede utilizar cualquiera de los protocolos de la capa de transporte de la pila TCP/IP:*User Datagram Protocol (UDP)* [5], *Transmission Control Protocol (TCP)* [6] y *Real Time Protocol (RTP)* [7] [8]. Además se hace necesario la existencia de un protocolo de señalización entre peers que les permita realizar peticiones de descarga sobre unidades definidas del flujo de vídeo.

Una parte fundamental del sistema radica en el elemento que acomoda el flujo de vídeo original a las unidades de transmisión a intercambiar por los peers. Asimismo, también se necesita un elemento que sea capaz de recuperar el flujo de vídeo original a partir de dichas unidades y entonces reproducir el vídeo.

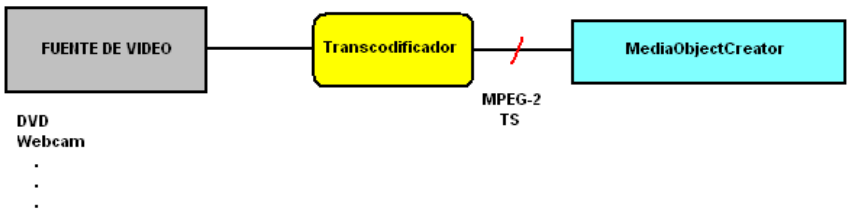

Figura 2: Fuente de unidades de transmisión

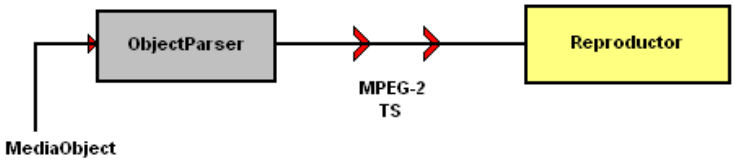

Figura 3: Receptor de unidades y reproducción

La Figura 2 muestra el esquema del primer elemento, que consta de la fuente de video, un transcodificador que transcodifica la señal original a un flujo de video MPEG-2 Transport Stream [9] y un elemento, denominado MediaObjectCreator, que crea las unidades de intercambio entre peers a partir del flujo MPEG-2 TS. En la Figura 3 se muestra el elemento terminal que a partir de las unidades de intercambio, denominadas MediaObject, recupera el flujo MPEG-2 TS que puede ser reproducido por el reproductor.

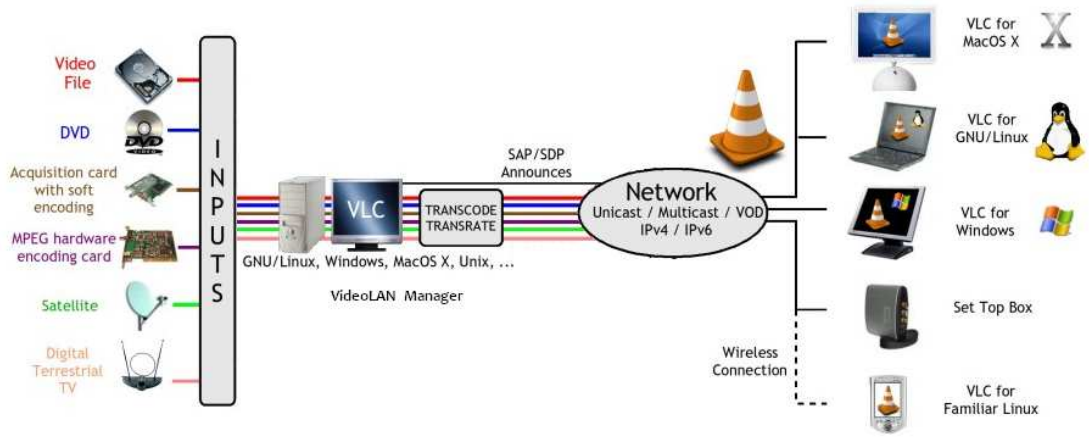

Figura 4: Capacidades del *VideoLAN*

El VideoLAN es una aplicación que puede realizar tanto la tarea de transcodificar como la de reproduccir, tal y como se puede observar en la Figura 4. El objetivo primordial de este TFC ha sido el desarrollo de una librería Java que acceda nativamente a la API [2] del VideoLAN distribuída en una librería C denominada libvlc. En la página oficial de la aplicación [10] se puede descargar tanto la versión compilada como el código fuente que es el que nosotros utilizaremos para la implementación de la librería Java.

La memoria del TFC queda organizada de la siguiente manera. Primeramente, en el Capítulo 1 se detalla la propuesta del TFC, incluyendo una explicación sobre el funcionamiento básico de la aplicación VideoLAN así como de las API's proporcionadas tanto en la librería libvlc como en el control ActiveX. El Capítulo 2 detalla el acceso Java al control ActiveX proporcionado por VideoLAN y la implementación de una clase que permite a otras aplicaciones clientes de nuestra librería disponer de un reproductor flexible y de gran riqueza gráfica. En el Capítulo 3 se explica cómo se ha realizado la librería Java que permite acceder a la funcionalidad completa proporcionada por la librería de VideoLAN libvlc. Finalmente, realizaremos las conclusiones de este TFC.

[2] API: *Application Programming Interface*, Interfaz de Programación de Aplicaciones

CAPÍTULO 1. VISIÓN GENERAL

En este TFC se desarrollan dos módulos para un sistema de transmisión de vídeo P2P. Por un lado, desarrollaremos la parte de acceso a la funcionalidad transcodificadora del VideoLAN y, por otro lado, su funcionalidad reproductora de tal modo que se pudiese empotrar en interfaces de usuario programadas en Java desarrolladas en cualquiera de las tres API's que dispone el lenguaje para tal fin: *Abstract Window Toolkit AWT* [11], Swing [12] y *Standard Widget Toolkit SWT* [13], las dos primeras distribuídas en el kit de desarrollo Java proporcionado por Sun Microsystems [14] y la última desarrollada por IBM dentro del proyecto Eclipse [15]. El acceso desde Java a la funcionalidad nativa de la librería libvlc se realiza utilizando JNI (Java Native Interface) [16], que es el mecanismo que proporciona el lenguaje Java para invocar desde sus programas a funciones de librerías nativas. Sin embargo, el acceso al control ActiveX queda restringido únicamente a aplicaciones desarrolladas en SWT, ya que ésta es la única de las tres API's gráficas de Java que lo permite implementar de modo sencillo, tal y como se explica en el Capítulo 2.

1.1. VideoLAN

El proyecto VideoLAN es un software libre distribuido bajo licencia GPL que integra diversos *codecs* de audio y video, así como diferentes tipos de contenedores y diversos protocolos de *streaming*. También puede ser utilizado como servidor tanto en transmisión unicast como multicast, en IPv4 o IPv6. Utiliza la librería de *codecs* libavcodec del proyecto FFmpeg [17] para manejar los formatos de audio y video con los que trabaja, empleando la librería de descifrado DVD libdvdcss para poder reproducir los DVDs cifrados.

Es uno de los reproductores más independientes, en cuanto a plataforma se refiere, con versiones para Linux, Microsoft Windows, Mac OS X, BeOS, BSD, Pocket PC y Solaris. En Windows, Linux y en algunas otras plataformas, VideoLAN incluye un plug-in Mozilla, que permite ver algunos archivos Quicktime y Windows Media en páginas web sin tener que utilizar un reproductor externo. A partir de la versión 0.8.2 en adelante, VideoLAN distribuye un plugin ActiveX, que permite ver vídeo incrustrado en páginas HTML cuando se navega con Internet Explorer.

1.2. Librería libvlc

La librería libvlc representa el API subyacente de VideoLAN. El reproductor VideoLAN, denominado VLC, es de hecho un simple envoltorio de acceso a la librería libvlc. Los desarrolladores pueden utilizar la librería libvlc para aprovechar las complejas funcionalidades implementadas por VideoLAN. La librería libvlc se distribuye como una librería compartida, lo que permite al desarrollador de aplicaciones acceder a la funcionalidad del VideoLAN sin tener que empezar a codificarla él mismo desde cero.

La librería libvlc se compone de ocho módulos, de los cuáles hemos desarrollado el ac-

ceso Java para siete de ellos dejando fuera la funcionalidad de trazas ya que en Java encontramos potentes librerías de trazas como, por ejemplo, Log4J [18].

1.2.1. Manejo de excepciones.

El manejo de excepciones consiste en la funcionalidad que proporciona la librería para el tratamiento de errores. Para ello se define la siguiente estructura:

```
struct libvlc_exception_t {
   int b_raised;
   char *psz_message;
};
typedef struct libvlc_exception_t libvlc_exception;
```

Está formada por el entero, b_raised, que se usa para determinar si se ha producido una excepción tomando el valor uno cuando ocurre una excepción y un cero en caso contrario. En el caso de que se produzca algún error el mensaje de error queda almacenado en la cadena psz_message.

El prototipo de las funciones definidas en este módulo, y cuya descripción se detalla en el Cuadro 1.1 son:

```
void libvlc_exception_init( libvlc_exception *p_exception);

int libvlc_exception_raised( libvlc_exception *p_exception);

void libvlc_exception_raise( libvlc_exception *p_exception,
char*psz_format, ...);

void libvlc_exception_clear( libvlc_exception *p_exception);

char* libvlc_exception_get_message( libvlc_exception
*p_exception);
```

Función	Descripción
libvlc_exception_init	Inicializa la excepción.
libvlc_exception_raised	Permite comprobar si se ha producido una excepción.
libvlc_exception_clear	Hace un reset de la excepción para ser reutilizada.
libvlc_exception_get_message	Se obtiene el mensaje de la excepción.

Cuadro 1.1: Descripción de las funciones relacionadas con el manejo de excepciones.

1.2.2. Núcleo de acceso al VideoLAN.

Las funciones de esta parte son las más importantes, pues son las que permiten crear o destruir instancias de la aplicación VideoLAN.

El resto de los módulos que existen dependen mucho de la instancia que se ha generado, es decir, si antes no se ha creado una instancia, no se va a poder hacer nada en cualquiera de los otros módulos ya que se utiliza siempre desde una instancia.

Las funciones definidas en este módulo, y cuya descripción se detalla en el Cuadro 1.3, son:

```
typedef struct libvlc_instance_t libvlc_instance_t;

libvlc_instance_t *libvlc_new( int , char **,

void libvlc_destroy( libvlc_instance_t *);

int libvlc_get_vlc_id( libvlc_instance_t *p_instance);
libvlc_exception_t *);
```

Función	Descripción
libvlc_instance_t	Estructura de que representa una instancia de la aplicación *VLC*.
libvlc_new	Crea e inicializa una nueva instancia.
libvlc_destroy	Destruir una instancia.
libvlc_get_vlc_id	Obtiene el id de la instancia.

Cuadro 1.3: Descripción de las funciones relacionadas con el núcleo.

1.2.3. Manejo de la lista de reproducción.

Estas funciones incluyen todos los métodos relacionados con la funcionalidad de control de la reproducción que podemos usar en cualquier sistema operativo. Es decir, permite que podamos controlar cualquier reproducción en curso, añadir o eliminar elementos de la lista de de reproducción de la aplicación.

En el Cuadro 1.5 encontramos la descripción de las funciones definidas para este módulo:

```
void libvlc_playlist_play( libvlc_instance_t *, int , int ,
char**,libvlc_exception_t *);

void libvlc_playlist_pause( libvlc_instance_t *,
libvlc_exception_t*);

int libvlc_playlist_isplayling( libvlc_instance_t
*,libvlc_exception_t *);

int libvlc_playlist_items_count( libvlc_instance_t
*,libvlc_exception_t *);
```

```
int libvlc_playlist_stop(libvlc_instance_t*,libvlc_exception_t *);

int libvlc_playlist_next(libvlc_instance_t*,libvlc_exception_t *);

int libvlc_playlist_prev(libvlc_instance_t*,libvlc_exception_t *);

int libvlc_playlist_clear(libvlc_instance_t*,libvlc_exception_t *);

int libvlc_playlist_add( libvlc_instance_t
*, const char *, const char *,libvlc_exception_t *);

int libvlc_playlist_add_extended( libvlc_instance_t
*, const char *, const char *, int, const char **,
libvlc_exception_t *);

int libvlc_playlist_delete_item( libvlc_instance_t
*, int, libvlc_exception_t *);

libvlc_input_t *libvlc_playlist_get_input(libvlc_intance_t
*,libvlc_exception_t*);

typedef struct libvlc_input_t libvlc_input_t;
```

Función	Descripción
libvlc_playlist_play	Inicia una reproducción de un elemento.
libvlc_playlist_pause	Pone a pausa el elemento que se está reproduciendo.
libvlc_playlist_isplaying	Comprueba si hay alguna reproducción en curso.
libvlc_playlist_items_count	Informa de la cantidad de elementos en la lista.
libvlc_playlist_stop	Detiene la reproducción en curso.
libvlc_playlist_prev	Carga el anterior elemento de la lista respecto al actual.
libvlc_playlist_next	Carga el siguiente elemento de la lista respecto al actual.
libvlc_playlist_clear	Elimina todos los elementos de la lista.
libvlc_playlist_add	Añade un elemento al final de la lista.
libvlc_playlist_add_extended	Añade un elemento al final de la lista con opciones adicionales.
libvlc_playlist_delete_item	Elimina un elemento de la lista.
libvlc_playlist_get_input	Obtiene el elemento de entrada que se está reproduciendo.
libvlc_input_t	Estructura de los elementos de entrada.

Cuadro 1.5: Descripción de las funciones relacionadas con el manejo de la lista de reproducción.

1.2.4. Manejo del audio.

En cualquier aplicación multimedia nos hace falta que podamos controlar el nivel de audio, pues bien, la librería proporciona funciones que nos permiten controlar el volumen en

la aplicación. Permitiendo que se controle el nivel de audio e incluso poder controlar el estado mudo.

Las funciones definidas en este módulo, y cuya descripción se detalla en el Cuadro 1.7, son:

```
void libvlc_audio_toggle_mute(libvlc_instance_t*,
libvlc_exception_t*);

vlc_bool_t libvlc_audio_get_mute(libvlc_instance_t*,
libvlc_exception_t*);

void libvlc_audio_set_mute( libvlc_instance_t*, vlc_bool_t ,
libvlc_exception_t*);

int libvlc_audio_get_volume(libvlc_instance_t*,libvlc_exception_t*);

void libvlc_audio_set_volume(libvlc_instance_t*,int ,
libvlc_exception_t*);
```

Función	Descripción
libvlc_audio_toggle_mute	Pone el audio en estado mudo.
libvlc_audio_get_mute	Permite conocer el estado mudo.
libvlc_audio_set_mute	Activa o desactiva el mudo.
libvlc_audio_get_volume	Obtiene el nivel del volumen que el valor oscila de uno a cien.
libvlc_audio_set_volume	Permite modificar el nivel del volumen.

Cuadro 1.7: Descripción de las funciones relacionadas con el manejo del audio.

1.2.5. Manejo del video.

Las funciones definidas en esta parte controlan el *renderizado* del video en curso, es decir, se hace el tratamiento gráfico de la secuencia de las imágenes que se disponen una tras otra a la tasa de imágenes determinadas. También es posible personalizar el tamaño de la ventana en la que se representa el video, teniendo en cuenta tanto el ancho como la altura. Igualmente, se puede controlar la opción de pantalla completa pudiendo activarla o desactivarla, según se desee. Las funciones definidas en este módulo, y cuya descripción se detalla en el Cuadro 1.9, son:

```
vlc_bool_t libvlc_input_has_vout(libvlc_input_t*,
libvlc_exception_t*);

float libvlc_input_get_fps (libvlc_input_t*,libvlc_exception_t*);

void libvlc_toggle_fullscreen(libvlc_input_t ,libvlc_exception_t);
```

```
void libvlc_set_fullscreen(libvlc_input_t*,int,
libvlc_exception_t*);

int libvlc_get_fullscreen(libvlc_input_t*,libvlc_exception_t*);

int libvlc_video_get_height( libvlc_input_t*,
libvlc_exception_t*);

int libvlc_video_get_width(libvlc_input_t*,libvlc_exception_t*);

char *libvlc_video_get_aspect_ratio(libvlc_input_t*,
libvlc_exception_t*);

void libvlc_video_set_aspect_ratio(libvlc_input_t*,
char*,libvlc_exception_t*);

void libvlc_video_take_snapshot(libvlc_input_t*,char*,
libvlc_exception_t*);

int libvlc_video_destroy( libvlc_input_t*,libvlc_exception_t*);

void libvlc_video_resize(libvlc_input_t*,int,int,
libvlc_exception_t*);

typedef int libvlc_drawable_t;

int libvlc_video_reparent(libvlc_input_t*,libvlc_drawable_t,
libvlc_exception_t*);

void libvlc_video_set_parent(libvlc_instance_t*,
libvlc_drawable_t,libvlc_exception_t*);

void libvlc_video_set_size( libvlc_instance_t*,int,int,
libvlc_exception_t*);

typedef struct
{
    int top,left,bottom,right;
} libvlc_rectangle_t;

void libvlc_video_set_viewport(libvlc_instance_t*,const
libvlc_rectangle_t*,const libvlc_rectangle_t*,libvlc_exception_t*);
```

1.2.6. VideoLAN Manager.

Este módulo corresponde a toda la funcionalidad de transcodificación y servidor de *streaming*, permitiendo la transmisión de flujo transcodificado a través de una red IP. En el sis-

Función	Descripción
libvlc_toggle_fullscreen	Permite poner el video a pantalla completa.
libvlc_set_fullscreen	Activa o desactiva la opción de pantalla completa.
libvlc_get_fullscreen	Permite saber si está activa la opción de pantalla completa.
libvlc_video_get_height	Devuelve el valor de alto del video en curso.
libvlc_video_get_width	Devuelve el valor de ancho del video en curso.
libvlc_video_get_aspect_ratio	Permite saber la tasa.
libvlc_video_set_aspect_ratio	Permite modificar la tasa.
libvlc_video_take_snapshot	Permite hace una captura instantánea del video.
libvlc_video_destroy	Elimina la entrada.
libvlc_video_resize	Reconfigura la ventana del video.
libvlc_video_reparent	Cambia el *parent* para la salida de video en curso.
libvlc_video_set_parent	Modifica el *parent* por defecto al video de salida.
libvlc_video_set_size	Modifica el tamaño por defecto del video saliente.
libvlc_rectangle_t	Define la estructura del rectángulo para *viewport*.
libvlc_video_set_viewport	Pone la salida de vídeo viewport para una salida de vídeo sin ventanas

Cuadro 1.9: Descripción de las funciones relacionadas con el manejo del video.

tema en desarrollo la idea es que el módulo de creación de unidades MediaObjectCreator, recibirá el flujo transcodificado a través de una interfaz local.

Las funciones definidas en este módulo, y cuya descripción se detalla en el Cuadro 1.11 son:

```
void libvlc_vlm_add_broadcast( libvlc_instance_t*,char*,char*,
char*,int ,char**,int ,int ,libvlc_exception_t*);

void libvlc_vlm_del_media(libvlc_instance_t*,char*,
libvlc_exception_t*);

void libvlc_vlc_set_enabled(libvlc_instance_t*,char*,int ,
libvlc_exception_t*);

void libvlc_vlm_set_output(libvlc_instance_t*,char*,char*,
libvlc_exception_t*);

void libvlc_vlm_set_input(libvlc_instance_t*,char*,char*,
libvlc_exception_t*);

void libvlc_vlm_set_loop(libvlc_instance_t*,char*,int ,
libvlc_exception_t*);

void libvlc_vlm_change_media(libvlc_instance_t*,char*,char*,
char *,int ,char**,int ,int ,libvlc_exception_t*);

void libvlc_vlm_play_media(libvlc_instance_t*,char*,
```

```
libvlc_exception_t*);

void libvlc_vlm_stop_media(libvlc_instance_t*,char*,
libvlc_exception_t*);

void libvlc_vlm_pause_media(libvlc_instance_t*,char*,
libvlc_exception_t*);
```

Función	Descripción
libvlc_vlm_add_broadcast	Permite hacer un *broadcast* con una entrada.
libvlc_vlm_del_media	Elimina un *broadcast*.
libvlc_vlc_set_enabled	Permite o no un *broadcast*.
libvlc_vlm_set_output	Poner una salida para un *broadcast*.
libvlc_vlm_set_input	Poner una entrada para un *broadcast*.
libvlc_vlm_set_loop	Permite activar reproducción constante.
libvlc_vlm_change_media	Permite editar el *broadcast*.
libvlc_vlm_play_media	Empieza a reproducir el *broadcast*.
libvlc_vlm_stop_media	Para el *broadcast* en curso.
libvlc_vlm_pause_media	Pone en pausa el *broadcast*.

Cuadro 1.11: Descripción de las funciones relacionadas con el VideoLAN Manager.

1.2.7. Manejo de la señal de entrada.

Estas funciones permiten recopilar información sobre el estado de la reproducción de un video o una canción como, por ejemplo, el tiempo de reproducción total, el tiempo de reproducción actual. Además, para el caso de que el video fuera bajo demanda nos permitiría implementar funciones avanzadas del tipo, adelantar la reproducción, rebobinar, etc.

Las funciones definidas en este módulo, y cuya descripción se detalla en el Cuadro 1.13 son:

```
void libvlc_input_free(libvlc_input_t *);

vlc_int64_t libvlc_input_get_length( libvlc_input_t *,
libvlc_exception_t *);

vlc_int64_t libvlc_input_get_time( libvlc_input_t *,
libvlc_exception_t*);

void libvlc_input_set_time( libvlc_input_t *, vlc_int64_t,
libvlc_exception_t *);

float libvlc_input_get_position( libvlc_input_t *,
libvlc_exception_t *);
```

```
void libvlc_input_set_position( libvlc_input_t *, float,
libvlc_exception_t *);

vlc_bool_t libvlc_input_will_play( libvlc_input_t *,
libvlc_exception_t *);

float libvlc_input_get_rate( libvlc_instance_t *,
libvlc_exception_t*);

void libvlc_input_set_rate( libvlc_instance_t *, float,
libvlc_exception_t*);

int libvlc_input_get_state( libvlc_instance_t *,
libvlc_exception_t*);
```

Función	Descripción
libvlc_input_free	Deja libre la entrada.
libvlc_input_get_length	Obtenemos la duración del elemento de entrada.
libvlc_input_get_time	Obtenemos el tiempo del instante de reproducción.
libvlc_input_set_time	Modificar el tiempo de la reproducción.
libvlc_input_get_position	Obtener la posición de la reproducción en curso.
libvlc_input_set_position	Modificar la posición de la reproducción.
libvlc_input_will_play	Permite saber si se reproducirá.
libvlc_input_get_rate	Permite saber la tasa.
libvlc_input_set_rate	Modificar la tasa.
libvlc_input_get_state	Permite saber el estado.
libvlc_input_has_vout	Comprueba si la entrada tiene un video de salida.
libvlc_input_get_fps	Permite saber si se está transcodificando.

Cuadro 1.13: Descripción de las funciones relacionadas con el manejo de la entrada.

1.3. Control ActiveX

1.3.1. ¿Qué es un control ActiveX?

ActiveX es un Component Object Model (COM) [19] desarrollado por Microsoft para plataformas Windows. El software basado en la tecnología ActiveX es difundido mediante un plugin [1] para Internet Explorer, los controles ActiveX se basan en aplicaciones ejecutables desde páginas web.

[1] Plugin: conocido como addin, add-in, addon o add-on, es una aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica.

Antes de la tecnología ActiveX, dos estándares fueron predominantes en Microsoft Windows. Uno de ellos fue Object Linking and Embedding (OLE) [20], y el otro fue COM. Ambos sistemas fueron diseñados para la interoperatividad de software, el primero se centraba en la comunicación y el segundo en la aplicación. En el 1996, Microsoft combinó las tecnologías pasando a ser ActiveX.

La suite de Microsoft Office es un ejemplo de software compatible con ActiveX. Con el uso de determinados lenguajes de programación el programador puede manipular documentos y hojas de cálculo desde código como si se tratara de utilizar directamente las aplicaciones.

Un control ActiveX es un componente reutilizable que implementa la interfaz IDispatch [2] [21]. Los componentes no llegan a ser una aplicación, sino que proporciona un pequeño bloque que puede ser compartido por software diferente. El hecho de que los botones tengan el mismo aspecto en casi cualquier programa en una plataforma, es un claro ejemplo de componente reutilizable que no se limita sólo a los controles ActiveX.

Los controles ActiveX pueden ser comparados a algunos applets de Java, ya que ambas tecnologías actúan como una capa de abstracción entre el desarrollador y el sistema operativo. Los applets de Java permiten funcionar en cualquier plataforma, en cambio, los componentes ActiveX sólo son compatibles con el sistema operativo Microsoft Windows. Otra gran diferencia entre ellos, es que los controles ActiveX pueden ser usados para hacer ataques informáticos.

Los controles ActiveX pueden ser escritos en MFC, ATL, C++, Borland Delphi y Visual Basic. Los ejemplos más comunes de los controles ActiveX son el botón comando, cuadro de lista, cuadros de diálogo e incluso el navegador Internet Explorer.

1.3.2. Descripción del control ActiveX del VLC

La aplicación VideoLAN para la plataforma Windows incorpora como opción instalar el control ActiveX, que permite incrustar el VideoLAN en un navegador web y en aplicaciones.

Los controles ActiveX tienen definidas unas propiedades que son propias para cada uno, en el caso del control ActiveX de VideoLAN tiene definidas las propiedades en el Cuadro 1.15.

Las propiedades definidas para el control ActiveX permiten obtener información, ya sea, si hay algún elemento para reproducir, que elemento se está reproduciendo. Son informaciones típicas de un reproductor.

Como en el caso de las propiedades, los controles ActiveX tienen definidos unos métodos que les permiten que tengan alguna funcionalidad, en el caso del control ActiveX de VideoLAN tiene definido los métodos en el Cuadro 1.17.

Los métodos definidos corresponden a funciones que puede hacer cualquier reproductor que permita ver vídeo. Es decir, podemos dar inicio a una reproducción, pasar al siguiente

[2] IDispatch: Es una interfaz que introduce el protocolo OLE Automation

Nombre	Tipo	get o set	Descripción
Length	Integer	get	Devuelve la longitud del video cargado.
PlaylistCount	Integer	get	Devuelve el número de items de la lista.
PlaylistIndex	Integer	get	Devuelve el índice del item que se reproduce
AutoLoop	Boolean	get/set	Determina si el item se tiene que volver a reproducir.
AutoPlay	Boolean	get/set	Determina si el reproductor cuando carga un item o una lista de reproducción tiene que reproducirlo automáticamente.
Volume	Integer	get/set	Indica el volumen actual, el valor va de cero a 100.
MRL	Integer	get/set	Devuelve la MRL del fichero cargado.
Time	Integer	get/set	Tiempo transcurrido desde el inicio de la reproducción.
Showdisplay	Boolean	get/set	Muestra o esconde el control de la vista.
Playing	Boolean	get	Devuelve si hay alguna MRL en reprodcucción.
Position	'real'	get/set	
VersionInfo	String	get	Devuelve la versión de la aplicación.

Cuadro 1.15: Propiedades

Nombre	Tipo	Descripción
setVariable	method	Asigna un valor a una variable que está definida en libvlc.c.
getVariable	method	Devuelve el contenido de una variable definida en libvlc.c.
pause	method	Pone a pause
play	method	Es el play como en cualquier reproductor, si no hay ningún ítem no hace nada.
playFaster	method	Permite que el video vaya más rápido.
playSlower	method	Hace ir más lento el vídeo.
stop	method	Para la reproducción en curso.
shuttle	method	Especifica los segundos.
playlistClear	method	Hace una limpieza de la lista de reproducción.
playlistNext	method	Selecciona el siguiente item de la lista de reproducción.
playlistPrev	method	Selecciona el elemento previo de la lista de reproducción.
addTarget	method	Reemplaza la lista de reproducción en curso con una *uri*.
toggleMute	method	Permite quitar o poner el volumen.
fullscreen	method	Permite el modo *fullscreen*.

Cuadro 1.17: Métodos

elemento de la lista de reproducción, etc.

Conociendo las propiedades y las funciones del control ActiveX podremos desarrollar en Java una aplicación que nos permita incrustar el VideoLAN permita y que permite controlarlo mediante el control ActiveX.

CAPÍTULO 2. INCRUSTADO DEL CONTROL VLC ACTIVEX EN JAVA SWT

En este capítulo se trata de hacer una explicación de cómo se ha podido desarrollar una aplicación con controles *ActiveX*. En nuestro caso, los controles a los que queremos tener acceso son los de la aplicación VideoLAN, mediante el plugin ActiveX del VideoLAN.

2.1. Introducción al desarrollo de aplicaciones con SWT

El proyecto SWT está integrado como parte de la API del plugin de Eclipse, pero para el desarrollo de aplicaciones autónomas es mejor utilizar el proyecto SWT de eclipse que se puede descargar de *http://www.eclipse.org/swt*.

Los pasos a seguir son los siguientes:

- Importar el proyecto SWT en el área de trabajo.

  En primer lugar hemos de descargar el proyecto SWT de Eclipse para poder utilizar la API, se ha de escoger el que podamos utilizar para nuestra plataforma de desarrollo.

  Terminada la descarga del proyecto, se ha de importar éste a nuestra área de trabajo. Para importar el proyecto en el Eclipse hemos de ir al menú *Archivo*, seleccionar *Importar* y luego ir a la opción *Existing Projects Into Workspace*, como se muestra en las Figuras 2.1 y 2.2.

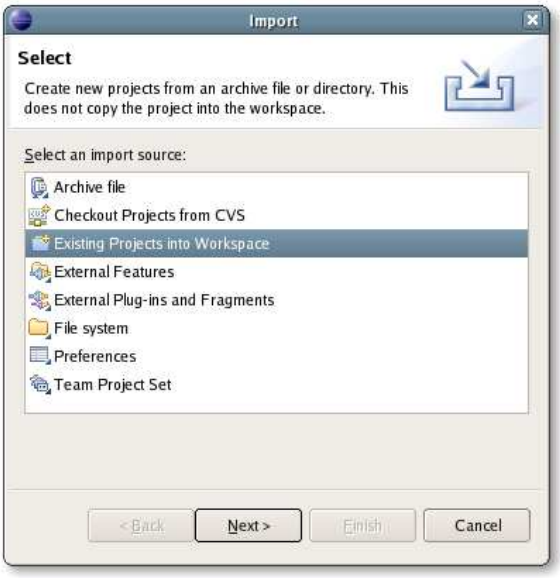

Figura 2.1: Diálogo de Eclipse para importar nuevos proyectos

Después de importar el proyecto SWT en el área de trabajo encontraréis un proyecto

Figura 2.2: Diálogo de Eclipse para selección del proyecto

nuevo con el nombre de *org.eclipse.swt*, significará que se ha importado correctamente.

- Ejemplo de uso del proyecto SWT de Eclipse.

  En un proyecto Java en el cual queremos usar la API de SWT se ha de crear una dependencia del proyecto SWT que hemos importado en el paso anterior. Para ello hemos de acceder a las propiedades del proyecto Java y seleccionar la opción de *Java Build Path*, en este punto se ha de añadir el proyecto con el nombre org.eclipse.swt, como se muestra en la Figura 2.3.

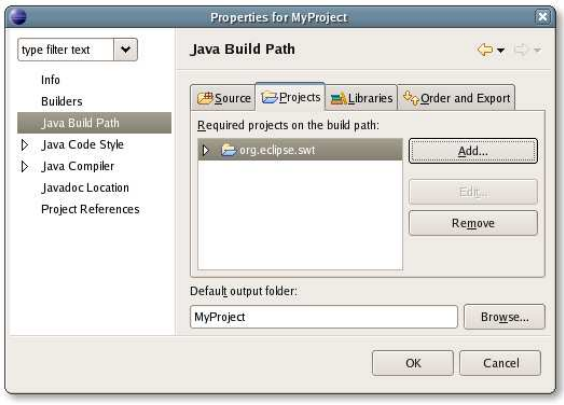

Figura 2.3: Diálogo propiedades del proyecto Java

La Figura 2.4, representa un ejemplo sencillo del uso del proyecto SWT, el resultado de la ejecución crea una ventana donde aparece el mensaje *Hello Word*.

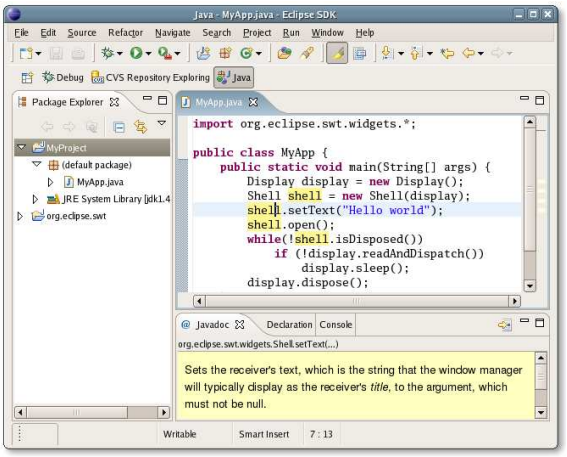

Figura 2.4: Ejemplo de aplicación usando la API de SWT

2.2. Manejo del Control ActiveX

Para poder trabajar con el plugin ActiveX de VideoLAN a través de Java, usamos unas clases que existen en la *API* de *SWT*, que son:

- OleFrame
- OleControlSite
- OleAutomation

Con estas tres clases el mecanismo para tener acceso a la API del plugin ActiveX de VideoLAN, lo primero de todo es crear un contenedor donde se van a contener los controles del plugin, el contenedor lo obtenemos a partir de la clase OleFrame. Cuando tenemos el contenedor creado podemos agregar un un sitio de control del plugin mediante la clase OleControlSite, donde uno de los parámetros que se le pasa al constructor es el plugin que queremos usar. En nuestro caso, usaremos el plugin ActiveX de VideoLAN.

Para tener acceso al mecanismo de los métodos de la API del plugin ActiveX de VideoLAN, utilizamos la clase OleAutomation que al final nos permitirá el acceso a las funciones de la aplicación. Estas funcionalidades son las que hemos comentado en el Capitulo 1, son las típicas funcionalidades de cualquier tipo de reproductor.

Esta explicación se entenderá mejor con el ejemplo siguiente:

```
public class ejemploOle {
   OleFrame frame = new OleFrame(this, style);
   try{
        OleControlSite site = new OleControlSite(frame,
       SWT.NONE, "VideoLAN.VLCPlugin.1");
        OleAutomation auto = new OleAutomation(site);
   }catch(SWTException e){
        System.out.println("Unable to open VideoLAN.VLCPlugin.1");
```

```
        return;
    }
}
```

Hasta aquí hemos explicado cómo tener un escenario para el acceso a las propiedades y a los métodos del plugin ActiveX de VideoLAN. Ahora bien, el objeto que hemos creado con la clase OleAutomation, nos permitirá modificar las propiedades e invocar los métodos referentes al plugin.

2.2.1. Modificación o recuperación de las propiedades del control ActiveX

Existen dos métodos de la clase OleAutomation que son esenciales, con ellos podremos inicializar o modificar y recuperar algunas propiedades del control:

```
    getProperty
    setProperty
```

Para empezar cualquier aplicación ActiveX, se han de inicializar algunas propiedades del control ActiveX, por ejemplo, en nuestro caso como que se trata de una aplicación que reproduce videos, antes de empezar deberíamos añadir un elemento, como se muestra en el ejemplo:

```
public void setMRL(String mrl){
    setProperty("MRL", new Variant((String mrl));
}
private setProperty(String property, Variant variant){
    int []rgdispid = auto.getIDSOfName(new String[]{property});
    int dispIdMember = rgdispid[0];
    auto.setProperty(dispIdMember,variant);
}
```

Si quisiéramos recuperar la información de alguna propiedad del control ActiveX, se haría como se muestra en el siguiente ejemplo:

```
public void getMRL(String mrl){
    return getProperty("MRL").getString();
}
private getProperty(String property){
    int []rgdispid = auto.getIDSOfName(new String[]{property});
    int dispIdMember = rgdispid[0];
    auto.getProperty(dispIdMember);
}
```

2.2.2. Invocación de métodos del control ActiveX

La invocación de los métodos del control ActiveX permitirá darle funcionalidad a la aplicación. Usaremos un método de la clase OleAutomation que se encargará de hacer la invocación , el método utilizado es:

```
invoke
```

En el siguiente ejemplo el método que se invoca es el de reproducción de un elemento, del siguiente modo:

```
public void play(){
    invokeMethod("play");
}

private Variant invokeMethod(string method){
   int []rgdispid = auto.getIDSOfName(new String[]{property});
   int dispIdMember = rgdispid[0];
   return auto.invoke(dispIdMember);
}
```

El método *play* es uno de los métodos que hemos nombrado en el Cuadro 1.17 del Capítulo 1 en la Sección 1.3., pues bien, el resto de los métodos que aparecen se harían como en este ejemplo.

Teniendo implementado los propiedades y los métodos del control Active de VideoLAN, podemos desarrollar en Java una aplicación sencilla que permita la reproducción de un elemento, como se muestra en la Figurar 2.5.

Figura 2.5: Reproducción a partir del control *ActiveX*

2.3. Implementación de un Composite SWT con el control ActiveX del VideoLAN integrado

Esta sección supone que el lector está familiarizado con el desarrollo de aplicaciones SWT y la terminología de programación Java, que puede encontrar más información en la página oficial de SWT de Eclipse [22] o también en la página de desarrollo de SWT de IBM [23].

Una vez que tenemos acceso a los métodos ofrecidos por el control ActiveX del VideoLAN, podremos ver cómo se ha integrado la parte gráfica con la parte funcional del control y explicaremos cómo se han tratado los eventos que se han implementado dentro del reproductor.

En la Sección 2.2. hemos desarrollado el control ActiveX para una aplicación Java en el que se podía ver el vídeo sin ningún control sobre él. En esta Sección vamos a desarrollar un panel de botones en el que podremos controlar el video que se esté reproduciendo.

2.3.1. Parte gráfica de los botones

Las imágenes para cada botón se han obtenido de la distribución con las fuentes del VideoLAN, usándose los mismos iconos que utiliza la propia aplicación para el sistema operativo *Mac OS*. Para el diseño del panel de botones del reproductor se definió la clase PlayerControlComposite, heredera de la clase Composite, definiendo internamente un Canvas[1] con unas dimensiones fijas para luego pintar en esa zona los botones. En ese mismo Canvas se añade un objeto de la clase PanelCanvas, que simplemente sirve para definir la imagen de fondo del panel de botones que es la que se muestra en la Figura 2.6.

Figura 2.6: Imagen del fondo del panel de botones

Para empezar a dibujar se ha definido la clase BotoneraAction que se encarga de pintar los botones y tratar los eventos que a cada uno de ellos se le han asignado. Los botones que se han definido son objetos de la clase ButtonImage, ésta clase hereda de la clase Canvas e implementa el método de la interfaz PaintListener, que nos permitirá pintar y hacer repintados. Cada uno de estos botones se le han definido dos imágenes, excepto al botón reproducir que son cuatro imágenes, ya que, para que el botón tenga el efecto visual de pulsado con una sola imagen no basta, a diferencia de los objetos de la clase Button.

Los estados se tratan con el pulsado del ratón encima del botón. Si pulsamos encima y no soltamos el botón se muestra una imagen distinta a la que hay cuando no se está pulsando el botón. Con el botón reproducir usamos dos imágenes más porque cuando el reproduc-

[1] Canvas es un lienzo donde se permite pintar en ella y donde se obtienen eventos.

tor está reproduciendo un fichero de vídeo, pasa a ser el botón pausa del reproductor y el botón reproducir se vuelve a poner cuando se ha terminado cualquier reproducción.

Los estados se han definido dentro de la clase ButtonImage, en el Cuadro 2.2 se detalla la descripción de los diferentes estados.

Función	**Descripción**
STATE_DOWN	Estado cuando se pulsa el botón.
STATE_UP	Estado por defecto y después de ser pulsado el botón.
STATE_RUNNING_UP	Estado por defecto cuando hay una reproducción en curso, sólo para el caso que se trate del botón reproducir.
STATE_RUNNING_DOWN	Estado cuando se pulsa el botón sólo para el caso que se trate del botón reproducir.

Cuadro 2.2: Descripción de los diferentes estados.

En el siguiente ejemplo de código el objeto botonera es el Canvas que hemos definido y en el que estamos dibujando los botones. Entonces el objeto *botonera* es el que está a la espera de cualquier evento del ratón y poderlo tratar. A todos los botones se le está haciendo el mismo tratamiento de eventos, es decir, cuando llega un evento que significa que se tiene pulsado un botón, el evento se trata en el método mouseDown cambiando el estado a STATE_DOWN. Cuando se deja de pulsar el botón llega otro nuevo evento que es tratado en el método mouseUp, modificando otra vez el estado a STATE_UP. Los estados STATE_RUNNING_UP y STATE_RUNNING_DOWN nos permiten diferenciar si existe alguna reproducción en curso, sólo para el botón reproducir, porque cuando existe una reproducción el botón se encuentra en el estado STATE_RUNNING_UP, y la imagen que aparece es la del botón pausa y no la imagen del botón reproducir.

```
botonera.addMouseListener(new MouseListener(){
        public void mouseDown(MouseEvent e) {
            stop.setState(ButtonImage.STATE_DOWN);
        }
        public void mouseUp(MouseEvent e) {
            stop.setState(ButtonImage.STATE_UP);
        }
});
```

Las imágenes que representan los botones del panel del reproductor son las que se muestran en el Cuadro 2.3.

La Figura 2.7 muestra la ditribución resultante de los botones en el panel y la Figura 2.8 muestra el efecto del boton de parada cuando está pulsado.

		Play
		Pause
		Avance
		Relentir
		Stop
		Siguiente
		Anterior

Cuadro 2.3: Botones parte del reproductor

Figura 2.7: Panel de botones

Figura 2.8: Panel de botones con el efecto del botón *stop*

2.3.2. Parte gráfica del control volumen

Como se puede apreciar en las Figuras 2.7 y 2.8 el panel de control incluye la parte gráfica del control volumen como en cualquier reproductor. La parte gráfica del volumen es la más complicada de todo el panel de botones al estar compuesta por cinco imágenes. Tenemos una imagen que representa la barra de recorrido, dos imágenes que representan la bola que se desplaza en la zona del recorrido, se utilizan dos para poder hacer el efecto de selección. Por último tenemos dos imágenes más que simplemente indican el volumen mínimo y máximo. No hay que olvidar que existe además una zona de texto donde se indica el tanto por ciento de nivel de volumen, en el Cuadro 2.4 se muestran las cinco imágenes utilizadas para el control del volumen.

En el diseño gráfico de la parte del volumen hemos implementado una clase llamada VolumenCanvas. Ésta adquiera de la clase Canvas sus métodos y propiedades, y se inserta

Barra de recorrido	Defecto	Selección	Máximo volumen	Sin volumen
				

Cuadro 2.4: Imágenes para la parte del volumen

un objeto de ésta en la clase PlayerControlComposite, que es la clase donde se centra la composición del panel de control. En la clase VolumenCanvas hemos implementado los eventos de la interfaz PaintListener, MouseListener y MouseMoveListener. De la interfaz PaintListener implementamos el método PaintControl que nos permite pintar la parte gráfica y que se haga el repintado correctamente si hay algún cambio en alguna imagen.

```
paintControl
```

La implementación de la interfaz MouseListener nos obliga a implementar tres métodos, pero únicamente implementaremos dos, dejando mouseDoubleClick con una implementación vacía ya que no nos interesa tratar ese evento. Con los métodos mouseDown y mouseUP podremos tratar el efecto cuando se selecciona la imagen de la bola, y también trataremos el efecto cuando se pulsa dentro del recorrido de la barra.

```
mouseDown

mouseUp

mouseDoubleClick
```

Por último, tenemos también la implementación de la interfaz MouseMoveListener en la que se implementa el único método mouseMove. Permite el desplazamiento de la bola cuando se selecciona con el ratón en el recorrido de la barra, típica propiedad de un *slider*.

```
mouseMove
```

2.3.3. Control de gráfico para seleccionar un elemento

Por último, falta por tratar el botón que selecciona una entrada para el reproductor, como se detalla en el Cuadro 2.5. El efecto del botón es el mismo que se ha implementado para los casos del control del reproductor.

		ButtonURL

Cuadro 2.5: Botón de añadir un elemento

Cuando se selecciona el botón se abre un diálogo en el que se permite configurar un elemento de entrada. Hay dos opciones, una es insertando una dirección IP donde se

esté enviando un vídeo por la red, y otra es seleccionando un fichero desde el disco duro, desde un *Pendrive* o desde una entrada de *CD/DVD*. El diálogo que se muestra es el de la Figura 2.9.

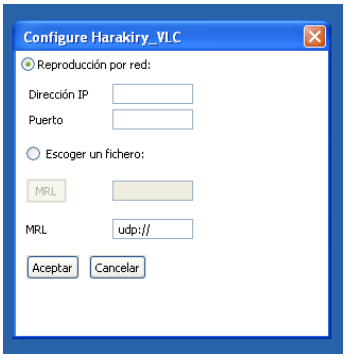

Figura 2.9: Diálogo de configuración

Después de configurar un elemento de entrada, el diálogo se cierra al pulsar el botón *Aceptar* volviendo al reproductor. Esta información aparece escrita en un *display* que se ve representado por la Figura 2.10.

Figura 2.10: Display

El display únicamente está diseñado para mostrar esta información, pero también podría utilizarse para mostrar el tiempo que lleva reproduciéndose el vídeo o una barra de desplazamiento indicando la posición de la reproducción, utilizando las funciones que se han comentado en la Subsección 1.3.2. del Capítulo 1.

En la Figura 2.11 se puede ver una captura de una aplicación que utiliza el composite desarrollado reproduciendo un fichero local.

Figura 2.11: Reproducción en curso

CAPÍTULO 3. USO DE LA LIBRERÍA LIBVLC

A lo largo del capítulo hablaremos de cómo acceder desde Java a los métodos nativos de la aplicación VideoLAN mediante una librería que utiliza para invocarlos JNI. La librería libvlc es el núcleo de la aplicación VideoLAN, en ella encontramos todas las definiciones de las estructuras, variables y los métodos existentes. En el Capítulo 1 en la Sección 1.2. hemos nombrado la estructura interna de la librería que serán llamadas desde la librería generada por nosotros.

3.1. Explicación de la parte nativa

A diferencia del control ActiveX que sólo se puede ejecutar en la plataforma Windows, el desarrollo de esta librería permitirá que sea usada tanto en Linux como en Windows.

Para el desarrollo de la librería se utiliza el lenguaje *C* ya que permite tener acceso a los métodos de la librería libvlc. Dado que la compilación de un programa o librería en C depende de cada plataforma, el código fuente estará obligado a diferenciarse según el sistema operativo en el que estemos haciendo la compilación.

3.1.1. Organización del código

Para tener un código ordenado y que se pueda entender por otras personas que no lo han implementado, tanto la parte de C como la de Java hemos modulado el código en función de los módulos del la librería libvlc que hemos tratado.

Para la parte de C, se ha generado diferentes cabeceras para los ficheros fuentes, en cada una implementa las funciones que existen para los módulos de la librería libvlc, tal y como se muestra en el Cuadro 3.2.

Módulo libvlc	**Fichero fuente**	**Fichero cabecera**
Núcleo de acceso al VideoLAN	core	entel_upc_bjvlc_JVLC
Manejo de la lista de reproducción	playlist	entel_upc_bjvlc_Playlist
Manejo del audio	audio	entel_upc_bjvlc_Audio
Manejo de la señal de entrada	input	entel_upc_bjvlc_Input
Manejo del video	video	entel_upc_bjvlc_Video
VideoLAN Manager	vlm	entel_upc_bjvlc_VLM

Cuadro 3.2: *Organización del código*

En cuanto a la parte de Java, lo hemos modulado también en función de los módulos tratados, coincidiendo con la parte de C. Siendo así más claro para trabajar entre los dos lenguajes. En el Cuadro 3.4 están detalladas las clases que implementan los métodos y en cada una de ellos hemos pensado que era conveniente tratar por separado las

excepciones que aparezcan durante la ejecución del código. En el Apéndice B tenemos representado el diagrama de clases del código Java desarrollado.

Módulo libvlc	**Clase**
Núcleo de acceso al VideoLAN	JVLC JVLCException
Manejo de la lista de reproducción	Playlist PlaylistException
Manejo del audio	Audio AudioException
Manejo de la señal de entrada	Input InputException
Manejo del video	Video VideoException
VideoLAN Manager	VLM VLMException

Cuadro 3.4: *Organización del código*

3.1.2. Plataforma *Windows*

En el desarrollo de la librería hemos usado como entorno de desarrollo *Integrated Development Environment(IDE)* [24] *Microsoft visual C++ 6.0* [25]. Existe el compilador *Minimalist GNU for Windows MinGW* [26], que es una versión del *GNU Compiler Collection (GCC)* [25] de *Linux* pero para *Windows*, permite compilar librerías y ejecutables, pero para usuarios de Windows el Microsoft Visual C++ es mucho más utilizado y fácil de compilar.

Las librerías dinámicas para Windows se llaman *Dynamic Linking Library* [27], su extensión es *.dll* y será el tipo de librería que vamos a desarrollar, ya que la JNI, tanto en Linux como en Windows requiere que la implementación nativa se distribuya en forma de librería dinámica.

Cuando se genere el proyecto desde el IDE Microsoft Visual C++ hay que añadir la librería libvlc y el directorio *plugins* dentro del directorio del proyecto que se ha creado. La librería y el directorio *plugins* se encuentran en el código fuente del VideoLAN. Este paso es necesario para compilar nuestra librería porque existen dependencias entre ellos dos. También hay que tener en cuenta que algunas librerías de Java se han de incluir en la configuración del proyecto. El primer paso consiste en acceder a la configuración como se muestra en la Figura 3.1 y el segundo como se indica en la Figura 3.2, se ha de poner la ruta del directorio donde se encuentra la librería *jawt.lib*, para que se pueda compilar la parte gráfica del código.

Para invocar las funciones de una dll necesitamos crear un alias para cada una de ellas en un fichero de cabecera. Estas definiciones se han declarado en el fichero de cabecera

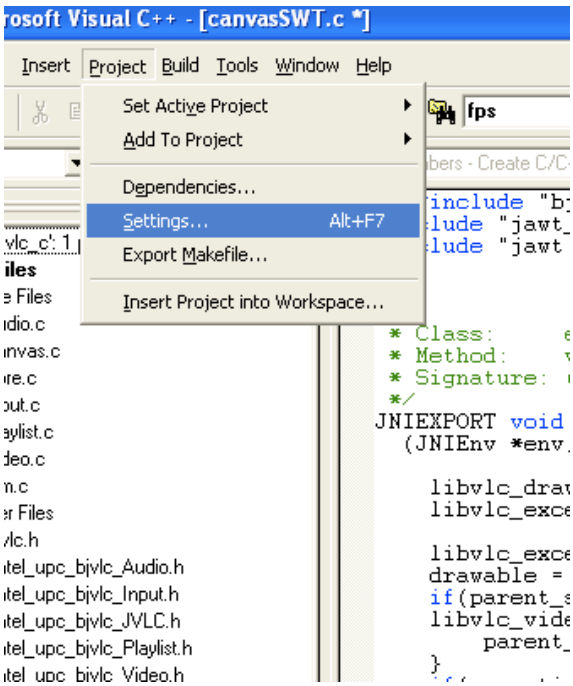

Figura 3.1: Acceso a la configuración del Visual C

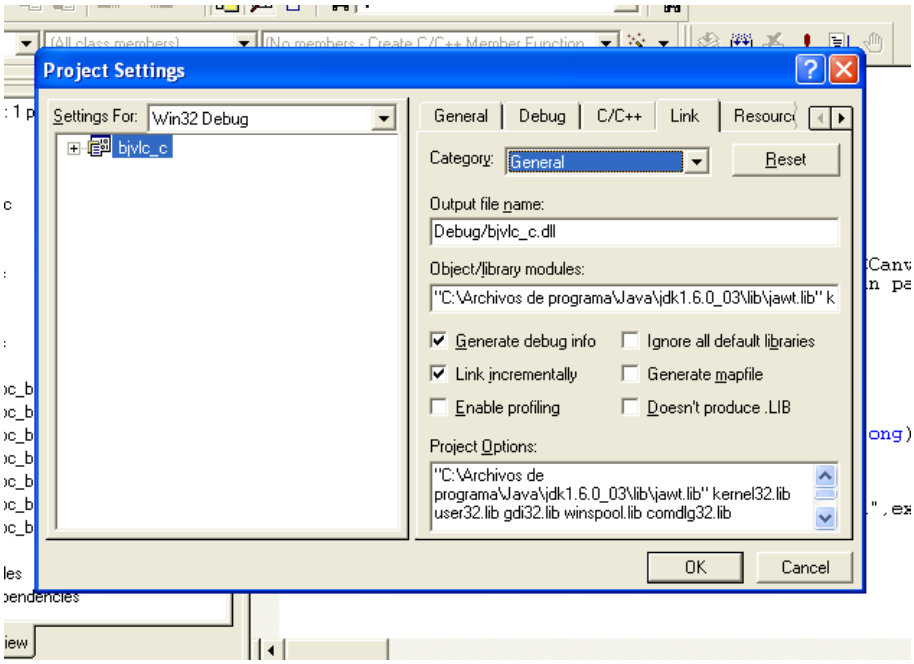

Figura 3.2: Configurador de la pestaña Link del Visual C

bjvlc .h. Para distinguirlas se han usado los mismos nombre que en la librería libvlc con el sufijo FUNC, a continuación viene un ejemplo.

```
typedef void (*libvlc_playlist_playFUNC)(libvlc_instance_t*,
int item,int argc,char **options, libvlc_exception_t*);

libvlc_playlist_playFUNC libvlc_playlist_play;
```

Cuando se trabaja con librerías dinámicas primero se han de cargar, para que se pueda trabajar con cualquier tipo de estructura, variable o función. En lenguaje C la carga de una librería dinámica se hace mediante el método LoadLibrary poniendo como parámetro el nombre de la librería, y devolviendo un puntero a la zona de memoria donde el sistema ha cargado la librería.

En nuestro código implementamos tres métodos que consisten en hacer una carga de la librería, inicializar las funciones y luego poder liberarla. El tipo de variable que usamos en *C* para almacenar la instancia de la librería libvlc es del tipo HINSTANCE. En este ejemplo se muestra como se carga una librería mediante el método LoadLibrary y la variable hInstLibvlc recibe una instancia de ella.

```
void loadLibrary(){
    HINSTANCE hInstLibvlc = LoadLibrary("libvlc.dll");
}
```

Para poder cargar y ejecutar una función utilizamos la función de la API GetProcAddress, permitiendo obtener un puntero de la función de la librería libvlc tal que como se puede observar en el siguiente ejemplo:

```
libvlc_playlist_play = (libvlc_playlist_playFUNC)GetProcAddress(
hInstLibvlc,"libvlc_playlist_play");
```

Todas las demás funciones de la librería exportada se cargan de modo equivalente. El código está implementado en el fichero core.c del Cuadro 3.2.

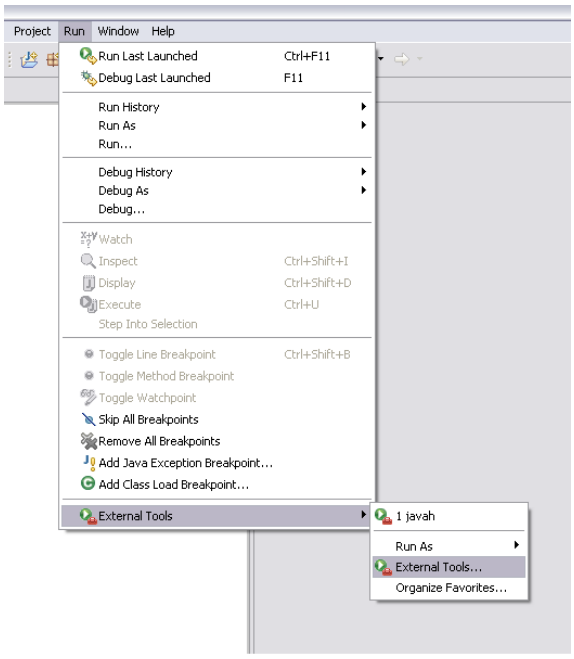

Figura 3.3: Acceso al configurador de herramientas externas de Eclipse

En cada uno de los ficheros cabecera del Cuadro 3.2 hay las declaraciones del código nativo que se ha generado a partir de JNI's. Los hemos generado mediante la aplicación

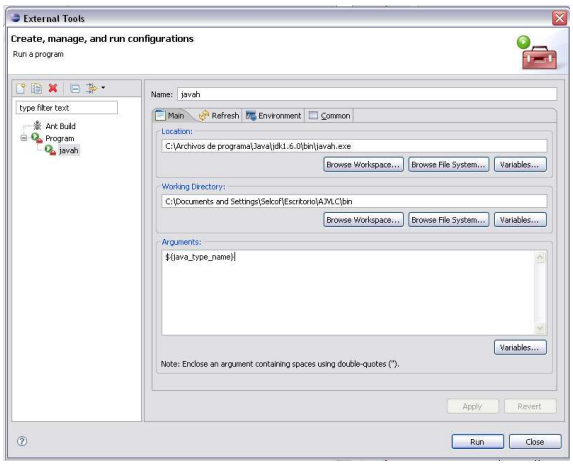

Figura 3.4: Diálogo de external tools de Eclipse

Eclipse usando como herramienta externa llamada javah [14], en la Figura 3.3 se muestra como acceder a la configuración que corresponde a la Figura 3.4, donde completamos los campos, en Location hemos de poner la ruta donde se encuentra el ejecutable javah, en Working Directory hemos de poner la ruta del directorio de nuestro proyecto y luego hay que poner como argumentos los que aparecen. Las declaraciones que tenemos en los headers, que son generadas mediante la herramienta javah, las implementamos en sus respectivas fuentes como se indica en los siguientes ejemplos:

- Ejemplo de un fragmento de código de la cabecera entel_upc_bjvlc_JVLC donde se muestra el resultado de haber utilizado el javah:

```
JNIEXPORT jlong JNICALL Java_entel_upc_bjvlc_JVLC_newInstance
(JNIEnv *, jobject, jobjectArray);
```

- En el siguiente ejemplo corresponde a un fragmento de un fichero fuente donde se hace la implementación de la función del ejemplo anterior:

```
JNIEXPORT jlong JNICALL Java_entel_upc_bjvlc_JVLC_newInstance(
JNIEnv *env, jobject jobj, jobjectArray args){
    int argc;
    const char **argv;
    int i;
    jstring jstr;
    libvlc_instance_t * instance;

    libvlc_exception_t exception;

    if(hInstLibvlc == NULL){
        loadLibrary();
        printf("Librería cargada.\n");
    }
    libvlc_exception_init(&exception);

    argc = (int)(*env)->GetArrayLength(env, (jarray)args);
```

```
    argv = (const char **) malloc(argc * sizeof(char*));
    for(i=0; i < argc; i++){
        jstr = (jstring)(*env)->GetObjectArrayElement(env, args, i);
    argv[i] = (*env)->GetStringUTFChars(env, jstr, NULL);
    }

    instance = libvlc_new(argc, (char**)argv, &exception);
    if(exception.b_raised == EXCEPTION_RAISED){
        JNU_ThrowByName(env, "entel/upc/bjvlc/JVLCException",
        exception.psz_message);
    }

    return (long)instance;
}
```

Aquí es donde implementamos el método y donde hacemos las llamadas a las funciones de la librería libvlc. En el ejemplo se trata del método que nos va a generar una instancia de la aplicación VideoLAN. Los parámetros que se pasan a la función no son del mismo tipo que los que se usan desde Java, con lo que se tienen que hacer conversiones de tipo. En el resto de funciones que se han implementado dentro de las diferentes fuentes, las excepciones las tratamos por igual, iniciando la excepción y luego comprobando si ha habido alguna.

Para tratar las excepciones en JNI hacemos uso del siguiente método que lo hemos tomado del libro de JNI [16]:

```
void JNU_ThrowByName(JNIEnv *env, const char *name, const char *msg){
    jclass cls = (*env)->FindClass(env, name);
    if(cls != NULL){
        (*env)->ThrowNew(env, cls, msg);
    }

    (*env)->DeleteLocalRef(env, cls);
}
```

Cualquier excepción que ocurra dentro de la librería durante la ejecución de la aplicación, con este método podremos saber de donde proviene y el contenido del mensaje de la excepción. Hemos creado una jerarquía de excepciones en Java como se detalla en la Figura 3.5, que representa un diagrama de clases.

Una de las complicaciones es mostrar el vídeo desde la parte nativa. Para hacer los renderizados del video nativamente se utilizó la librería jawt. En Windows la estructura que se utiliza para renderizar nativamente el pintado del vídeo se llama JAWT_Win32DrawingSurfaceInfo. Esta estructura nos proporciona la información de la ventana para luego renderizar desde una aplicación gráfica diseñada con la API de AWT.

Otra posibilidad de renderizar el vídeo, se ha conseguido para aplicaciones diseñada con la API de SWT. A diferencia que en el otro caso, no hace falta utilizar ninguna estructura

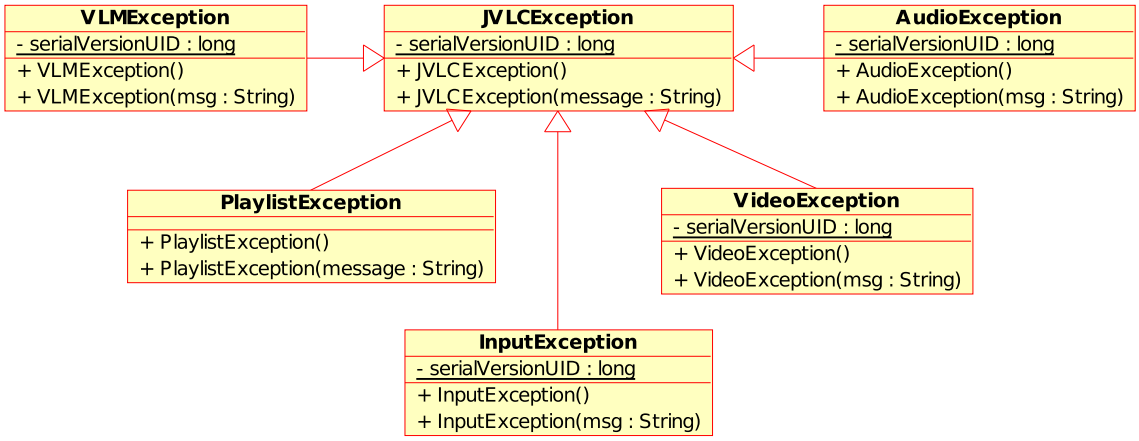

Figura 3.5: Diagrama de la jerarquía de excepciones

especial que nos de información de la ventana. Lo único que se hace es indicarle al método nativo mediante un parámetro, la ventana donde se tiene que hacer el pintado.

3.1.3. Plataforma *Linux*

En el entorno Linux se ha trabajado sobre la distribución *Debian* [28] y se ha tenido que preparar el escenario para poder compilar el código en lenguaje C y para Java. Para el lenguaje C el compilador por excelencia es el *GCC*, permitiendo generar librerías dinámicas llamadas *Shared Object* [29] con extensión *.so*. Para la parte de *Java* hemos utilizado *Eclipse* como en *Windows*.

A diferencia de Windows, no hace falta hacer todo el proceso de cargar la librería desde el código como hemos visto en la Sección 3.1.2., ni inicializar las funciones, porque en Linux la compilación del VideoLAN permite que al pasar por parámetro como librería el vlc, cuando se compila, se considera como una exportación de las funciones del VideoLAN. Para ello el Makefile [1], que usamos para la compilación de la librería, se le pasa por parámetro el nombre y la ruta donde se encuentre ubicado el vlc, como se muestra el siguiente ejemplo.

```
all:
    gcc -lvlc ... -shared -I"/usr/include/vlc" ... -o ljvlc.so core.c
    playlist.c ...
```

Para compilar la librería hay que tener en cuenta la ruta donde se encuentra ubicada la *Java Runtime Environment(JRE)* [14], también se añadirá en la configuración del fichero Makefile. Para el tratamiento de la parte gráfica se tienen que utilizar las librerías X11, que son las que permiten pintar en UNIX. Por tanto, en el Makefile se tiene que añadir la ruta donde se ubiquen awt,jawt y mawt, y pasarlas también como parámetro.

La estructura JAWT_X11DrawingSurfaceInfo es la que utiliza Linux como parte gráfica del código y tiene la misma finalidad que hemos explicado para el caso de Windows en la

[1] Makefile: es un archivo con las ordenes para compilar un ejecutable o una librería

Sección 3.1.2..

3.2. Carga de la librería desde *Java*

La parte de Java es multiplataforma, es decir, el código que se usa es el mismo para Windows y Linux.

En Java tenemos definidos los métodos nativos que son llamados desde la librería que hemos estado explicando en Sección 3.1.. Para cargar la librería desde Java se utiliza un método de la API declarándolo del siguiente modo:

```
static{
    System.loadLibrary("bjvlc");
}
```

Al método tal como se indica sólo se le pasa como parámetro el nombre de la librería que queremos usar. En nuestro caso, la librería resultante la hemos llamado *bjvlc*, y en ella tenemos declarados los métodos que usaremos desde el código Java, en el Apéndice tenemos la Sección A donde se detallan con una breve descripción los métodos que se usan desde Java hacia C. En el Apéndice B tenemos un diagrama de clases de Java.

3.2.1. Java Bindings libvlc

Después de realizar el estudio de cómo poder acceder a las funciones de la API de la librería libvlc mediante JNI, ahora es el momento de diseñar un reproductor que implemente la parte funcional del VideoLAN.

Para la parte gráfica vamos a reutilizar el diseño realizado para el reproductor que utiliza los controles ActiveX que se ha explicado en el Capítulo 2.

Los controles que se van a implementar para el reproductor son los que se han explicado en la Sección 1.2. del Capítulo 1, donde en el Apéndice A encontramos las relaciones de las funciones entre Java y C. Ahora sólo se ha de asignar cada control a cada elemento del reproductor del mismo modo que se ha hecho en el Capítulo 2.

El resultado será un simple reproductor en el que se podrá ver vídeo desde cualquier dispositivo, ya sea un fichero de disco, un DVD o desde un flujo que se envía por la red.

A diferencia del control ActiveX que no podíamos desarrollar como aplicación un servidor de video, mediante la librería libvlc vamos a poder desarrollarlo. El módulo comentado en la Sección 1.2.6. en el Capítulo 1, es el que permite que podamos enviar flujo de video por la red. La Figura 3.6 muestra el resultado de la aplicación del servidor de video.

La aplicación de servidor de video, es una aplicación sencilla desarrollada en Java utilizando la API SWT. Como se puede observar en la Figura 3.6 se han de completar unos

Figura 3.6: Aplicación del servidor de video

campos que caracterizan cada transmisión de video. En primer lugar, a una transmisión se le asigna un nombre para que se distinga de otras transmisiones. Lo más importante del servidor es la información que se introduce como entrada y como salida. La entrada es el elemento que vamos a enviar a través de la red, y la salida representa hacía donde vamos a enviar el flujo de video.

En la aplicación aparecen el botón reproducir y parada, no tienen nada de especial, son simples botones de reproducción.

El servidor podría ser más complejo, pero nos hemos limitado a que fuera simple, sin ninguna complejidad. Por ejemplo, cuando se añade un elemento sólo damos opción a que se añada un elemento local, sin embargo, existe la opción de que un elemento de entrada sea un flujo que es enviado por la red. En el caso de un elemento de salida, damos opción a que se introduzca una dirección IP sin modificar el tipo de protocolo, sin embargo, acepta varios tipos de protocolos.

CAPÍTULO 4. CONCLUSIONES

Ha sido interesante participar en el desarrollo de uno de los módulos de un Sistema P2P. Realmente he aprendido cosas que durante estos años de la carrera no había tenido ocasión de poder realizar ni practicar.

Me he basado en las fuentes gráficas que VideoLAN distribuye, que son propias del sistema operativo MAC Os. En cuanto al diseño del reproductor de la parte de los botones, me he basado en la versión de VideoLAN que existe para MAC. Sin embargo, hay una gran diferencia entre ambas, y es que en mi caso no reproduzco el video en una ventana externa como lo hace en el MAC, sino en la misma aplicación.

Me hubiera gustado hacer también algún montaje de red P2P para el tratamiento de flujo del video que manda el VideoLAN hacia una máquina destino, más que nada porque podría haber visto alguna cosa más sobre estos sistemas de red.

Los sistemas P2P pienso que son el futuro, es decir, el pensar que pueden llegar a tener infinidad de canales que la televisión digital no seria capaz de abarcar es muy atractivo. Pero la gran desventaja frente a la televisión digital es que no ofrece una gran calidad de imagen. Es una de las grandes desventajas porque al no poder disponer de un gran ancho de banda, o sea, una velocidad no muy alta, hace que se apliquen compresiones de video repercutiendo en la calidad de imagen.

BIBLIOGRAFÍA

[1] S. Cherry. The battle for broadband [Internet Protocol television]. *IEEE Spectrum*, 42(1):24–29, January 2005.

[2] X. Zhang, J.C. Liu, B. Li, , and P. Yum. CoolStreaming/DONet: A data-driven overlay network for efficient live media streaming. In *Proc. IEEE INFOCOM*, volume 3, pages 2102–2111, Miami, FL, USA, March 2005.

[3] PPLive. http://www.pplive.com.

[4] SopCast. http://www.sopcast.org.

[5] J. Postel. User datagram protocol, http://www.ietf.org/rfc/rfc0768.txt, August 1980.

[6] California 90291 Information Sciences Institute University of Southern California 4676 Admiralty Way Marina del Rey. Transmission control protocol darpa internet program protocol specification, http://www.ietf.org/rfc/rfc793.txt, September 1981.

[7] R. Frederick y V. Jacobson H. Schulzrinne, S. Casner. Rtp: A transport protocol for real-time applications, July 2003.

[8] H. Schulzrinne y S. Casner. Rtp profile for audio and video conferences with minimal control, July 2003.

[9] Peter Schirling. Mpeg-2 systems white paper ps and ts, October 2005.

[10] VideoLAN. Videolan - vlc media player, April 2001.

[11] Sun Microsystems. The awt in 1.0 and 1.1, April 1999.

[12] Sun Microsystems. Creating a gui with jfc/swing, April 1999.

[13] Eclipse. Swt: The standard widget toolkit, February 2007.

[14] Sun Microsystems. Sun developer network.

[15] Eclipse, 2000.

[16] Sheng Liang. *The Java Native Interface Programmer Guide and Specification*. Addison-Wesley, 1999.

[17] Ffmpeg, December 2004.

[18] Log4j.

[19] Com: Component object model technologies.

[20] Object linking and embedding.

[21] Idispatch interface.

[22] Activex support in swt.

[23] Integrate activex controls into swt applications.

[24] Integrated development environment.

[25] Microsoft Corporation. Microsoft visual c++, 1994-1998.

[26] Colin Peters. Minimalist gnu for windows, July 1998.

[27] Brent et al Rector. *Win32 Programming*. Addison-Wesley, 1997.

[28] Debian.

[29] David A. Wheeler. Program library howto, April 2003.

APÈNDIXS

APÉNDICE A. DESCRIPCIÓN DE LAS FUNCIONES

En esta sección hacemos una descripción de los métodos que se han implementado en cada una de las clases que aparecen en el diagrama de clases, haciendo referencia a la llamada que hacen hace el código nativo.

A.1. Descripción de los métodos de la clase JVLC

Método Java:	
Método nativo:	newInstance(String[] args
Función libvlc:	libvlc_new (**int** , **char**∗∗, libvlc_exception_t ∗)
Descripción:	Función que genera una instancia de la aplicación.

Método Java:	destroyJVLC()
Método nativo:	destroy(**long** instance)
Función libvlc:	libvlc_destroy (libvlc_instance_t ∗)
Descripción:	Función que destruye la instancia.

Método Java:	get_vlc_id_JVLC()
Método nativo:	get_vlc_id (**long** instance)
Función libvlc:	libvlc_get_vlc_id (libvlc_instance_t ∗)
Descripción:	Función que devuelve el identificador de la instancia.

En la clase JVLC se tratan los métodos nativos que corresponden al source *core*, es el eje central del reproductor, por tanto, es el que.

A.2. Descripción de los métodos de la clase Playlist

Método Java:	add(String uri, String name)
Método nativo:	add(**long** instance,uri,name)
Función libvlc:	libvlc_playlist_add (libvlc_instance_t *,**const char***,**const char***, libvlc_exception_t *)
Descripción:	Función que añade un elemento a la lista.
Método Java:	play()
Método nativo:	play(**long** instance,**int** item,String []options)
Función libvlc:	libvlc_playlist_play (libvlc_instance_t *,**int**,**int**,**char****, libvlc_exception_t *)
Descripción:	Función que permite reproducir cualquier elemento.
Método Java:	stop()
Método nativo:	stop(**long** instance)
Función libvlc:	libvlc_playlist_stop (libvlc_instance_t *, libvlc_exception_t *)
Descripción:	Función que permite para la reproducción en curso.
Método Java:	pause()
Método nativo:	pause(**long** instance)
Función libvlc:	libvlc_playlist_pause (libvlc_instance_t *, libvlc_exception_t *)
Descripción:	Función que permite reproducir cualquier elemento.
Método Java:	prev()
Método nativo:	prev(**long** instance)
Función libvlc:	libvlc_playlist_prev (libvlc_instance_t *, libvlc_exception_t *)
Descripción:	Función que carga el elemento anterior.
Método Java:	next()
Método nativo:	next(**long** instance)
Función libvlc:	libvlc_playlist_next (libvlc_instance_t *, libvlc_exception_t *)
Descripción:	Función que carga el siguiente elemento.
Método Java:	deleteItem(**int** idItem)
Método nativo:	delete(**long** instance,idItem)
Función libvlc:	libvlc_playlist_next (libvlc_instance_t *,**int**, libvlc_exception_t *)
Descripción:	Función que elimina un elemento.
Método Java:	isPlaying()
Método nativo:	add(**long** instance)
Función libvlc:	libvlc_playlist_isplayling (libvlc_instance_t *, libvlc_exception_t *)
Descripción:	Función que comprueba si hay alguna reproducción en curso.
Método Java:	itemsCount()
Método nativo:	itemsCount(**long** instance)
Función libvlc:	libvlc_playlist_items_count (libvlc_instance_t *, libvlc_exception_t *)
Descripción:	Función que informa de cuantos elementos hay en la lista.

A.3. Descripción de los métodos de la clase Video

Método Java:	destroyVideo()
Método nativo:	_destroyVideo(**long** instance)
Función libvlc:	libvlc_video_destroy (libvlc_instance_t *, libvlc_exception_t *)
Descripción:	Función que elimina la entrada.

Método Java:	getFullscreen()
Método nativo:	_getFullscreen(**long** instance)
Función libvlc:	libvlc_get_fullscreen (libvlc_instance_t *, libvlc_exception_t *)
Descripción:	Función que comprueba si está activa la opción de pantalla completa.

Método Java:	setFullscreen(**boolean** fullscreen)
Método nativo:	_setFullscreen(**long** instance,**boolean** fullscreen)
Función libvlc:	libvlc_set_fullscreen (libvlc_instance_t *, **int**, libvlc_exception_t *)
Descripción:	Función que activa o desactiva la opción de pantalla completa.

Método Java:	toggleFullscreen()
Método nativo:	_toggleFullscreen(**long** instance)
Función libvlc:	libvlc_toggle_fullscreen (libvlc_instance_t *, libvlc_exception_t *)
Descripción:	Función que pone a pantalla completa la ventana del video.

Método Java:	getHeight()
Método nativo:	_getHeight(**long** instance)
Función libvlc:	libvlc_video_get_height (libvlc_instance_t *, libvlc_exception_t *)
Descripción:	Función que informa del tamaño en alto de la ventana.

Método Java:	getWidth()
Método nativo:	_getWidth(**long** instance)
Función libvlc:	libvlc_video_get_width (libvlc_instance_t *, libvlc_exception_t *)
Descripción:	Función que informa del tamaño del ancho de la ventana.

Método Java:	getSnapshot(String filepath)
Método nativo:	_getSnapshot(**long** instance,String filepath)
Función libvlc:	libvlc_video_take_snapshot(libvlc_instance_t *,**char***, libvlc_exception_t *)
Descripción:	Función que que permite hacer una captura de la ventana.

Método Java:	reparent(Component c)
Método nativo:	_reparent(**long** instance,Component c)
Función libvlc:	libvlc_video_reparent (libvlc_instance_t *, libvlc_drawable_t , libvlc_exception_t *)
Descripción:	Función que modifica el parent para la salida del video.

Método Java:	setSize(**int** width,**int** height)
Método nativo:	_setSize(**long** instance,**int** width,**int** height)
Función libvlc:	libvlc_video_set_size (libvlc_instance_t *,**int**,**int**, libvlc_exception_t *)
Descripción:	Función que modifica el tamaño de la ventana del video de salida.

A.4. VLM

Método Java:	addBroadcast(String name,String input,String ouput,String[] options, **boolean** enabled,**boolean** loop)
Método nativo:	addBroadcastC(**long** instance,String name,String input,String output, **boolean** enabled,**boolean** loop)
Función libvlc:	libvlc_vlm_add_broadcast(libvlc_instance_t *,**char***,**char***,**char***, **int**,**char****,**int**,**int**, libvlc_exception_t *)
Descripción:	Función que crea un broadcast con una entrada.
Método Java:	deleIteMedia(String name)
Método nativo:	deleteMediaC(**long** instance,String name)
Función libvlc:	libvlc_vlm_del_media(libvlc_instance_t *,**char***, libvlc_exception_t *)
Descripción:	Función que permite eliminar un broadcast.
Método Java:	setEnabled(String name,**boolean** enabled)
Método nativo:	setEnabledC(**long** instance,String name,**boolean** enabled)
Función libvlc:	libvlc_vlm_set_enabled(libvlc_instance_t *,**char***,**int**, libvlc_exception_t *)
Descripción:	Función que permite hacer un broadcast.
Método Java:	setOutput(String name,String output)
Método nativo:	setOutputC(**long** instance,String name,String output)
Función libvlc:	libvlc_vlm_set_output (libvlc_instance_t *,**char***,**char***, libvlc_exception_t*)
Descripción:	Función que permite poner una salida al broadcast.
Método Java:	setInput(String name,String input)
Método nativo:	prev(**long** instance,String name,String input)
Función libvlc:	libvlc_vlm_set_input (libvlc_instance_t *,**char***,**char***libvlc_exception_t*)
Descripción:	Función que permite poner una entrada al broadcast.
Método Java:	setLoop(String name,**boolean** loop)
Método nativo:	setLoopC(**long** instance,String name,**boolean** loop)
Función libvlc:	libvlc_vlm_set_loop (libvlc_instance_t *,**char***,**int**, libvlc_exception_t *)
Descripción:	Función que activa o desactiva la reproducción constante.
Método Java:	changeMedia(String name,String input,String ouput,String[] options, **boolean** enabled,**boolean** loop)
Método nativo:	changeMedia(**long** instance,String name,String input,String output, **boolean** enabled,**boolean** loop)
Función libvlc:	libvlc_vlm_change_media(libvlc_instance_t *,**char***,**char***,**char***, **int**,**char****,**int**,**int**, libvlc_exception_t *)
Descripción:	Función que permite editar el broadcast y modificar los parámetros.
Método Java:	playMedia(String name)
Método nativo:	playMediaC(**long** instance,String name)
Función libvlc:	libvlc_vlm_play_media(libvlc_instance_t *,**char***, libvlc_exception_t *)
Descripción:	Función que inicia la reproducción del broadcast.
Método Java:	stopMedia(String name)
Método nativo:	stopMediaC(**long** instance,String name)
Función libvlc:	libvlc_vlm_stop_media(libvlc_instance_t *,**char***, libvlc_exception_t *)
Descripción:	Función que para la reproducción en curso.
Método Java:	pauseMedia(String name)
Método nativo:	pauseMediaC(**long** instance,String name)
Función libvlc:	libvlc_vlm_pause_media(libvlc_instance_t *,**char***, libvlc_exception_t *)
Descripción:	Función que pone en pausa el broadcast.

A.5. Descripción de los métodos de la clase Audio

Método Java:	getMute()
Método nativo:	_getMute(**long** instance)
Función libvlc:	libvlc_audio_get_mute (libvlc_instance_t *, libvlc_exception_t)
Descripción:	Función que informa el estado del *mute*.

Método Java:	setMute(**boolean** value)
Método nativo:	_setMute(**long** instance,**boolean** value)
Función libvlc:	libvlc_audio_set_mute (libvlc_instance_t *, vlc_bool_t , libvlc_exception_t)
Descripción:	Función que permite activar o desactivar el *mute*.

Método Java:	toggleMute()
Método nativo:	_toggleMute(**long** instance)
Función libvlc:	libvlc_audio_toggle_mute (libvlc_instance_t *, libvlc_exception_t)
Descripción:	Función que quita o pone el volumen.

Método Java:	getVolumen()
Método nativo:	_getVolumen(**long** instance)
Función libvlc:	libvlc_audio_get_mute (libvlc_instance_t *, libvlc_exception_t)
Descripción:	Función que nos devuelve el nivel del volumen.

Método Java:	setVolumen(**int** volumen)
Método nativo:	_setVolumen(**long** instance,**int** volumen)
Función libvlc:	libvlc_audio_get_mute (libvlc_instance_t *, **int** volumen, libvlc_exception_t)
Descripción:	Función que permite modificar el nivel del volumen.

A.6. Descripción de los métodos de la clase Input

Método Java:	getLength()
Método nativo:	_getLength(**long** instance)
Función libvlc:	libvlc_input_get_length (libvlc_input_t *, libvlc_exception_t *)
Descripción:	Función que informa de la duración del elemento de entrada.
Método Java:	getTime()
Método nativo:	_getTime(**long** instance)
Función libvlc:	libvlc_input_get_time (libvlc_input_t *, libvlc_exception_t *)
Descripción:	Función que informa del instante de tiempo de reproducción..
Método Java:	getPosition ()
Método nativo:	_getPosition (**long** instance)
Función libvlc:	libvlc_get_position (libvlc_input_t *, libvlc_exception_t *)
Descripción:	Función que informa de la posición de la reproducción.
Método Java:	setTime(**long** time)
Método nativo:	_setTime(**long** instance,**long** time)
Función libvlc:	libvlc_input_set_time (libvlc_input_t *, vlc_int64_t , libvlc_exception_t *)
Descripción:	Función que modifica el tiempo de la reproducción.
Método Java:	setPosition (**float** position)
Método nativo:	_getHeight(**long** instance,position)
Función libvlc:	libvlc_input_set_position (libvlc_input_t *, **float** , libvlc_exception_t *)
Descripción:	Función que modifica la posición de la reproducción.
Método Java:	getFPS()
Método nativo:	_getFPS(**long** instance)
Función libvlc:	libvlc_input_get_fps (libvlc_input_t *, libvlc_exception_t *)
Descripción:	Función que permite saber si se está transcodificando.
Método Java:	isPlaying ()
Método nativo:	_isPlaying (**long** instance)
Función libvlc:	libvlc_input_will_play (libvlc_input_t *, libvlc_exception_t *)
Descripción:	Función que permite saber si se reproducirá.
Método Java:	hasVout()
Método nativo:	_hasVout(**long** instance)
Función libvlc:	libvlc_input_has_vout (libvlc_input_t *, libvlc_exception_t *)
Descripción:	Función que comprueba si la entrada tiene un video de salida.

APÉNDICE B. DIAGRAMA DE CLASES

En esta sección mostramos el diagrama de clases, como todo diagrama de clases su función es describir la estructura de un sistema mostrando sus clases, atributos y las relaciones que hay entre ellos.

El diagrama está formado únicamente de la parte de Java que corresponde al tratamiento nativo y sus métodos.

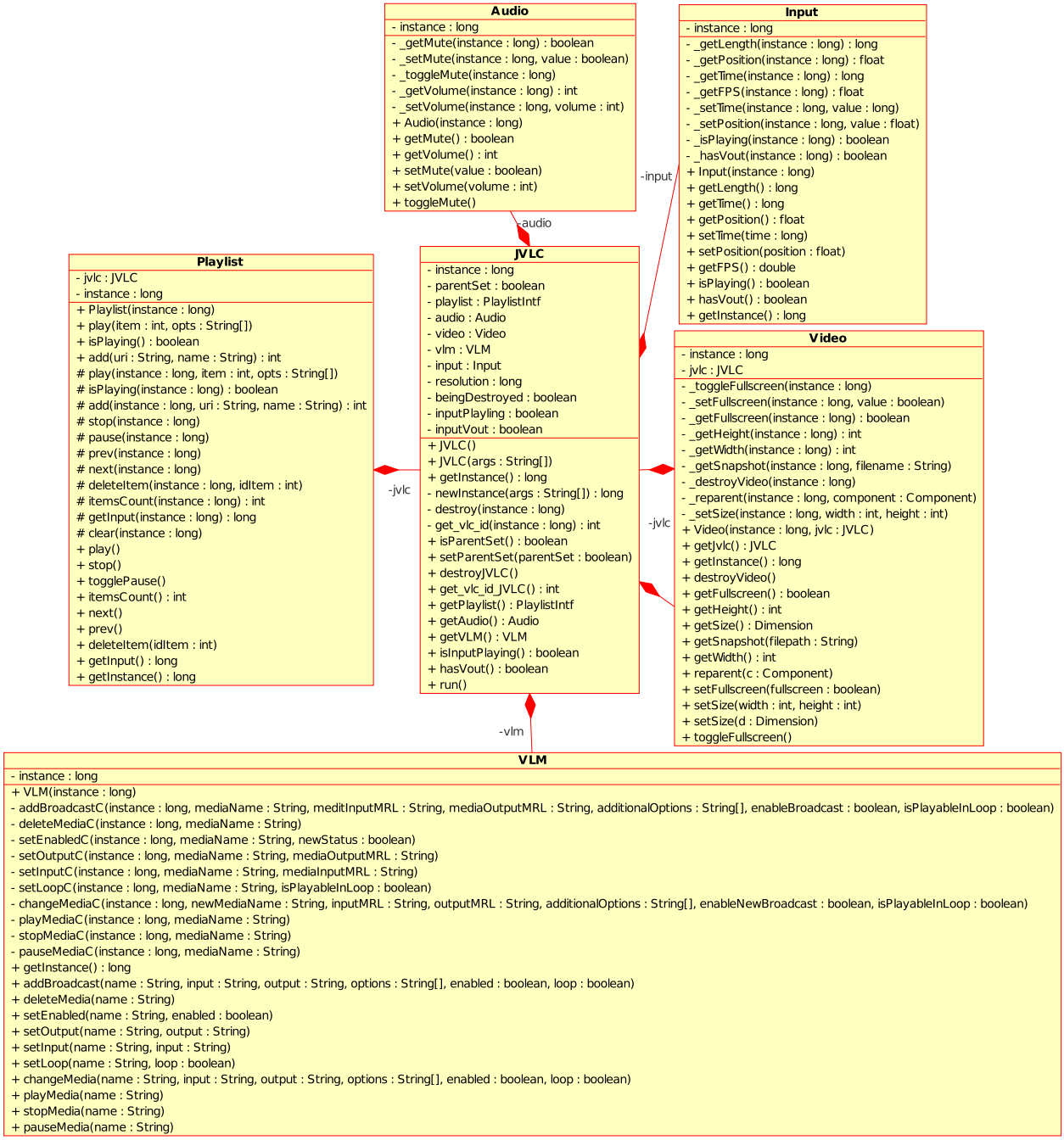

Figura B.1: Diagrama de clases