ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Desarrollo de un sistema de gestión de una parrilla de contenidos de vídeo y audio en red

---

Proyecto final de carrera

Ingeniería técnica de telecomunicación, especialidad en sonido e imagen

Alain Chas Arizcuren

Mikel Sagüés García

Pamplona, 5 de julio de 2013

Índice

1 Introducción

1.1 Objeto del proyecto

Implementar una aplicación que permita gestionar, a través de una escaleta gráfica, los contenidos de una emisión de vídeo y audio en red, utilizando la tecnología de streaming proporcionada por el software VLC [4].

La aplicación debe contener una interfaz gráfica programada en Java, y que provea al usuario las herramientas suficientes para la creación de la emisión. Además, debe implementarse un visor de contenidos que permita al usuario ver qué contenidos hay programados en qué instantes.

1.2 Motivación y punto de partida

El primer motivo de solicitar este proyecto ha sido las ganas de aplicar los conocimientos sobre Java adquiridos en los meses anteriores en un programa de considerable envergadura, experiencia que pone de manifiesto la utilidad de tales conocimientos y los asienta. Lejos de parecerse a pequeños ejercicios relativos a partes muy específicas del lenguaje, un proyecto de estas características obliga al desarrollador a utilizar casi todo lo aprendido, y a ingeniárselas ante el difícil reto de crear de la nada un software mínimamente competitivo.

1.3 Contexto y antecedentes

Es importante contextualizar bien el presente proyecto, de modo que el lector comprenda con exactitud cuál ha sido el verdadero objetivo de éste.

1.3.1 Proyecto previo

Se partió de un proyecto anterior, realizado por Hodei Altuna Cueli e Ignacio Echeverría Sánchez [3], cuyo objetivo era crear una aplicación que permitiera gestionar de forma visual y sencilla la programación de vídeos del sistema implementado para la emisión del canal de televisión del ayuntamiento del valle de Aranguren: Aranguren TV.

1.3.2 Aranguren TV

El sistema físico del que se dispone en el valle de Aranguren para la emisión del canal de TDT puede observarse en la figura 1.1 (téngase en cuenta que los modelos que aparecen en las imágenes que componen la figura no se corresponden con los verdaderos, siendo el gráfico meramente ilustrativo).

Este sistema se compone de un PC conectado en red con el multiplexor *iMux* (sistema basado en Linux) a través de un cable cruzado Ethernet. El *iMux* es un multiplexor de canales de TDT (junta varios canales digitales en una sola salida digital con el formato adecuado de la TDT) que permite incluir aplicaciones interactivas. A la salida del i*Mux* se encuentra un modulador TDT (COFDM) cuya salida contendrá los canales listos ya para su emisión.

El PC se utiliza para, por un lado, gestionar la configuración del *iMux* a través de su interfaz Web y, por otro, para crear una escaleta (parrilla) de contenidos que se van a emitir. Siguiendo la escaleta, los manda por *streaming* al *iMux* para que éste los adapte al formato de emisión en TDT.

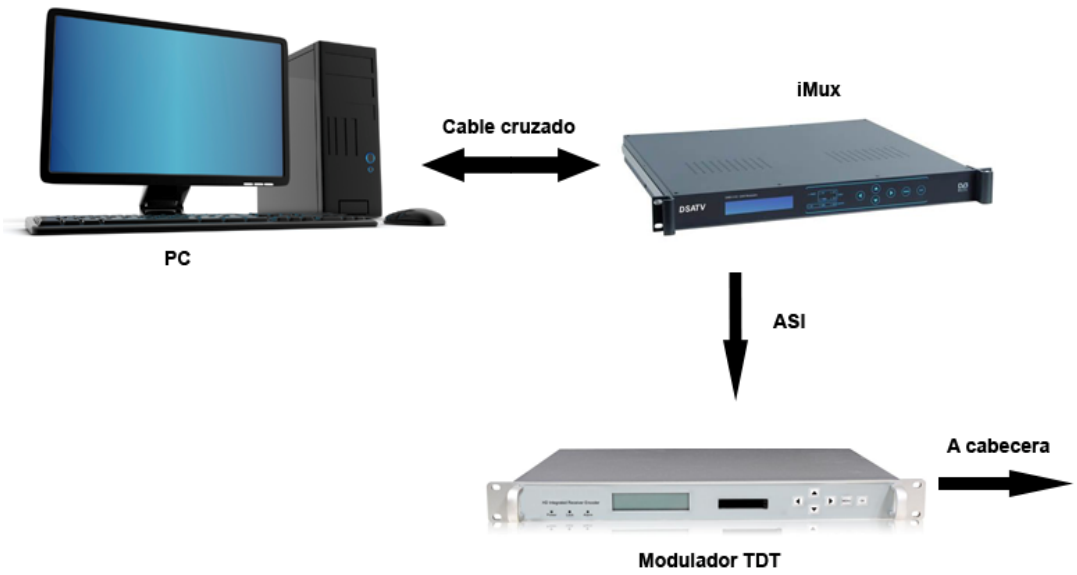


Figura 1.1 Aranguren TV

El sistema de emisión propuesto consta de 2 canales:

a) Un primer canal que contiene un vídeo en lazo (*Loop*) y que se emite en todo momento. Este vídeo muestra la información que el ayuntamiento del Valle de Aranguren quiera transmitir a sus ciudadanos. Este vídeo debe estar alojado en el propio iMux.

b) Otro canal donde se emiten los contenidos editados por Aranguren TV para su emisión a ciertas horas programadas. Los contenidos se envían desde el PC por *streaming* al iMux, ya que existe un programa capaz de enviarlo de manera programada, siguiendo una parrilla de contenidos.

El proyecto de Hodei e Ignacio se centró en este último canal; más concretamente, en crear una aplicación que corriera sobre el PC, que permitiera gestionar la emisión de vídeos gráficamente, y que proporcionara una salida por streaming, la cual se encargará de leer el iMux.

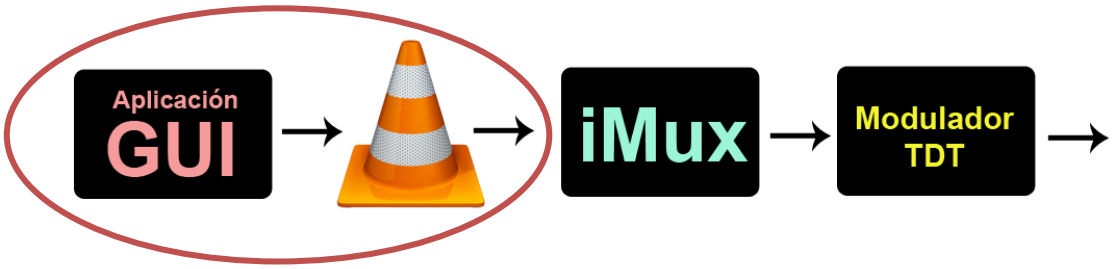


Figura 1.2 Aranguren TV (2)

En el proyecto no se actuó sobre el iMux (tampoco con el modulador), y su alcance se limitó a la realización del streaming de vídeo. Puede verse un esquema sencillo de lo mencionado en la figura 1.2, donde el círculo rojo representa tal alcance, y en la que puede observarse el icono del software VLC, que fue parte vital del proyecto.

1.3.3 Nuevo proyecto, nueva dirección

Este proyecto ha sido, por tanto, una continuación; sin embargo, ha cogido una dirección más amplia, dejando la solución para el valle de Aranguren como caso particular. Tal como se ha mencionado en el apartado 1.1, se ha pretendido lograr un software que permita gestionar y emitir un canal de TV en red, vía streaming.

1.4 Mejoras incorporadas en la nueva versión

A continuación se hace un listado de ellas:

- Estabilidad computacional.
- Posibilidad de guardar y recuperar el proyecto.
- Sistema de guardado de un fichero de seguridad del proyecto.
- Visor de la escaleta mejorado: ahora pueden verse todos los vídeos programados y el espacio temporal exacto que ocupa cada uno.
- Interfaz gráfica de usuario (GUI) con más opciones: tablas para visualizar aquellos vídeos importados, tanto si tienen un horario programado como si no, listado de la emisión de la semana seleccionada, etc.
- Mejor rendimiento del visor de la escaleta: la actualización gráfica de la programación es “instantánea” y no produce parpadeo.
- Explorador de archivos integrado en la GUI: ya no hay que abrir el explorador cada vez que se quiere importar un vídeo a emitir.
- Opción de transducción del stream: se da la posibilidad de realizar una conversión de códec de vídeo y audio en “tiempo real”.
- Opción de inicio y parada de la emisión.
- Filtrado de ficheros de vídeo y audio: no se permite la importación al proyecto de archivos que no sean de vídeo o de audio.
- Compatibilidad con la última versión del software VLC (versión 2.0.5 Twoflower, durante el desarrollo de la aplicación).
- No mueren todas las instancias VLC al cerrar la aplicación, sólo aquellas asociadas a la ejecución de ésta.

Debe mencionarse que se ha eliminado la opción de dar periodicidad a la emisión de un vídeo (por ejemplo, todos los días a las 12:00h), ya que se ha considerado que no es una opción demasiado útil.

1.4.1 El porqué de una nueva versión

El principal motivo es la inestabilidad de la aplicación previa. Una fuga de memoria provoca que tal aplicación (programada en Java) haga uso acumulativo de RAM hasta que la Java Virtual Machine (JVM) decide parar su ejecución para no dañar otros procesos de la computadora. A pesar de los intentos de depuración que se realizaron, no se consiguió hallar la fuente del problema.

Otras razones de la realización de una nueva versión son la falta de prestaciones que sí se han incorporado en ésta, y que se han mencionado al inicio de este apartado.

1.4.2 Partes recicladas de la versión anterior y modificaciones

Aunque la aplicación del proyecto previo ha sido remodelada, ésta ha servido de modelo y alimentación de la nueva, manteniendo su estructura general.

A continuación se detallan aquellas partes y mecanismos del sistema que se han mantenido en la nueva versión:

- Uso de la consola del sistema para instanciación del VLC y FFmpeg [5].
- Comunicación vía telnet entre aplicación y VLC.
- Uso de la librería gráfica de Java SWT [7].
- Uso de archivos con código VLM que se cargan en el VLC.

Por otro lado, el único cambio significativo de la nueva versión (aparte de las mejoras mencionadas al inicio del apartado) ha sido:

- Desarrollar la aplicación para Linux (concretamente, Ubuntu 12.04 LTS) en vez de para Windows.

2 Problemas a afrontar y soluciones

Aunque realmente los problemas fueron afrontados en el proyecto previo, es conveniente resumir en qué consistieron éstos y qué solución se adoptó, con la pertinente modificación en el proyecto actual.

La principal tarea fue encontrar el modo concreto de decirle al VLC [4] qué contenidos debía emitir y cuándo. Se conocía la capacidad de servidor de streaming del VLC (y esta era la principal apuesta), pero no la forma de gestionarla. Se optó por el estudio de la herramienta VLM (Video Lan Manager) que integra el propio VLC, la cual permite, mediante un tipo de código específico, gestionar el horario de los contenidos a emitir (entre otros). Los resultados fueron satisfactorios y por ello se ha decidido utilizar esta opción en el proyecto actual.

Las tareas que continuaron a la de la programación de contenidos a emitir fueron: conseguir hallar la duración de los vídeos y decidir qué librería gráfica de Java utilizar en la aplicación.

2.1 Código VLM

El uso de esta opción ha requerido el estudio previo de la herramienta, cuya característica principal es el uso de un lenguaje específico: el código VLM.

A pesar de no haber mucha documentación, el lenguaje no es demasiado extenso, y uno puede familiarizarse rápidamente con él. Es suficiente la documentación que puede hallarse en la web del proyecto VideoLan.

Desde el punto de vista de la gestión de contenidos, el lenguaje cuenta con 3 elementos esenciales: los media y los Schedule.

2.1.1 Media

Los media se componen de una entrada (por ejemplo, un fichero de vídeo en el PC) y de una salida (que define cómo y a dónde se quiere enviar el stream).

Hay dos tipos de media:

- *vod*: estos son usados para vídeo bajo demanda, y no toman parte dentro del alcance de este proyecto. Son “medias” que se envían al cliente que los solicite.
- *broadcast*: estos “medias” se acercan a lo que sería un canal de TV, ya que empiezan a emitirse cuando el administrador lo desea, y el cliente los recibe si quiere (pero no tiene control sobre ellos).

A continuación podemos ver el ejemplo de declaración de un elemento media de tipo broadcast:

```
new channel1 broadcast enabled
setup channel1 input http://host.mydomain/movie.mpeg
setup channel1 output
#rtp{mux=ts,dst=239.255.1.1,port=5004,sdp=sap://,name="Channel 1"}
```

Pasos para la configuración, mediante código VLM, de un broadcast:

1. Se declara un nuevo elemento mediante la etiqueta “new”, se le da el nombre deseado (channel1) y se define de qué tipo es (broadcast en este caso). Luego, se activa mediante la etiqueta “enabled” (con “disabled” quedaría desactivado).

2. Se configura la entrada del broadcast declarado mediante “setup”, nombre del broadcast e “input”, poniendo al final la URI del recurso (puede ser un archivo local, en nuestra computadora).

3. Se configura la entrada del broadcast declarado mediante “setup”, nombre del broadcast y “output”; ahora, en vez de añadir el origen, añadimos el destino, que posee variables tales como: protocolo (rtp en este caso), tipo de encapsulación del stream (ts), IP destino (239.255.1.1), puerto (5004), etc.

2.1.2 Schedule

Los schedule son aquellos elementos de tipo calendario, que obligan al VLC a realizar determinada acción cuando se alcanza cierta hora en cierto día.

A continuación puede verse ver el ejemplo de declaración de un elemento schedule:

```
new sched_1schedule enabled
setup sched_1date2013/05/15-12:00:00
setup sched_1 append control channel1 play
```

Pasos para la configuración, mediante código VLM, de un schedule:

1. Se declara un nuevo elemento mediante la etiqueta “new”, se le da el nombre deseado (sched_1 en este caso), se pone schedule y se activa (con “enabled”).

2. Se configura la fecha y hora del Schedule con “setup”, nombre del schedule, ”date”; se declara, después, la fecha y la hora a la que se quiera que el evento tenga lugar.

3. Se asigna un comportamiento para cuando se lance el evento; en este caso se ha querido que el broadcast antes declarado (channel1) comience a emitirse. Esto se consigue mediante “setup”, nombre del Schedule, “append”, “control” y el comando de control que se desea (normalmente “nombre del broadcast” + “play” para iniciar la emisión o “nombre del broadcast” + “stop” para detenerla).

2.1.3 Conclusión

Con los dos elementos analizados en el apartado anterior (broadcast y schedule) puede generarse un fichero de texto que contenga toda la programación de un canal de TV.

Por otro lado, VLM permite la carga de un fichero con programación en tal código (punto 2.2.2 de la memoria).

El uso de la herramienta VLM es, por tanto, una solución viable y efectiva ante el problema de cómo hacer que el contenido se emita de una forma organizada.

2.2 Conexión con herramienta VLM

Otra decisión a afrontar fue qué vía utilizar para conectar con esta herramienta:

- Por interfaz web.
- Por medio de la interfaz gráfica del VLC.
- Vía telnet.

Esta última fue la elección adoptada, por ofrecer sencillez de control de la herramienta VLM y por su eficacia. Hodei e Ignacio concluyeron lo siguiente:

"El proceso de cargar el archivo de configuración VLM mediante la interfaz Telnet es sin duda el más adecuado. Este es capaz de ejecutar la carga durante la reproducción de la escaleta de emisión en cualquier momento y sin necesidad da detenerla.
Además, presenta varias opciones de control de la herramienta VLM. De este modo, toda edición de la configuración VLM, como borrar una programación de un vídeo o cambiar el tipo de salida, serán muy útiles en el futuro.
El programa requiere el uso de una instancia VLC con la interfaz Telnet habilitada y un programa que actúe como cliente Telnet." [3]

Para este proyecto se ha optado por el uso de la interfaz telnet como vía de comunicación con la herramienta VLM, manteniendo la estructura de conexión del proyecto previo, puesto que da un resultado satisfactorio.

2.2.1 VLM vía telnet

Telnet es un protocolo de red que permite conectar un ordenador a otro de modo que el primero (cliente) pueda manejar al segundo (servidor).

En este caso VLM crea un servidor de *Telnet*. Quien conecte con él a través de la interfaz telnet será el cliente, y es el que solicita conexión al servidor. Mediante *Telnet*, el cliente puede controlar el servidor, por lo que es usado para realizar acciones de manera remota. En este caso VLC toma como entrada de datos la IP "*localhost*" y el puerto 4212 por defecto. Asignando la salida del cliente *Telnet* del mismo modo que la entrada de VLC, la gestión de datos se realiza desde la misma máquina donde el software está instalado, es decir, la red de conexión será interna.

La forma de instanciar VLC con su interfaz telnet activa ha sido a través de la Shell del sistema operativo, con el comando "*vlc --intf telnet*", tal como se muestra en la figura 2.1.

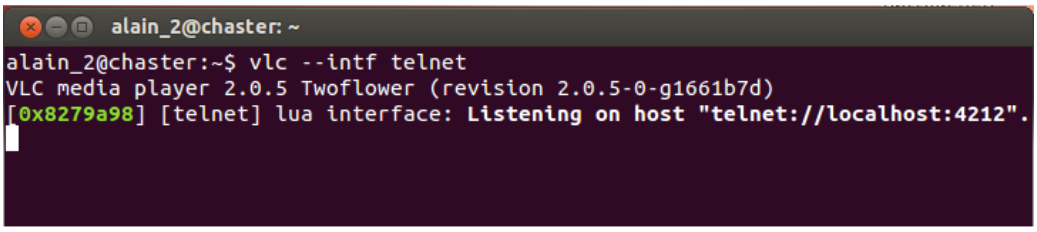


Figura 2.1 Instanciación de VLC con interfaz telnet activa

Para conectarse basta con escribir en la línea de comandos "*telnet dirección_IP_del_servidor puerto_conexión*". En la figura 2.2 puede verse el resultado del caso particular en que el servidor telnet (el propio VLC) se está ejecutando en la misma máquina, con puerto de escucha 4212. La respuesta del servidor es la petición de una contraseña (que por defecto es "*admin*") y posteriormente un mensaje que dice "*Welcome, Master*". Una vez en este punto, podemos solicitar al servidor VLC que realice cualquiera de los comandos VLM citados en el punto 2.2.2.

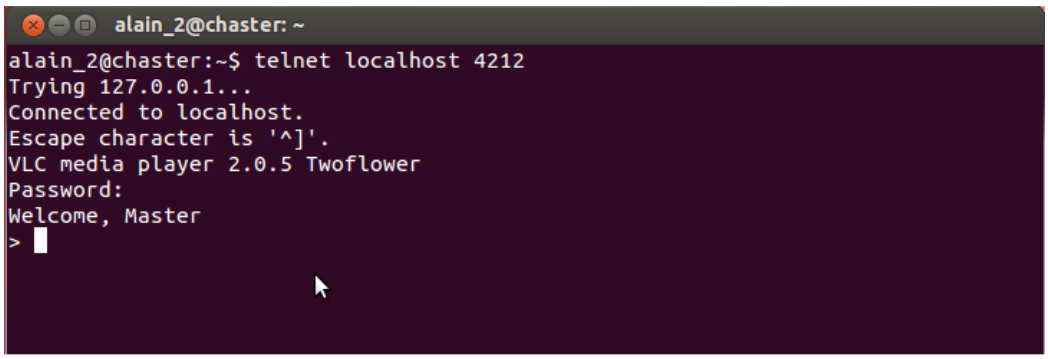


Figura 2.2 conexión con VLC, vía telnet

2.2.2 Comandos que ofrece la interfaz telnet

Aunque el código VLM visto antes (relativo a los broadcast y schedule) puede enviarse a modo de comando a la herramienta VLM, resulta más cómodo y óptimo hacérselo llegar mediante la carga de un fichero.

La interfaz telnet del VLM ofrece una lista de comandos para controlar la configuración y monitorización de la emisión que realiza VLC. Los comandos que tienen relevancia en el proyecto son los siguientes:

- *load*: permite cargar un fichero con configuración VLM en el VLC.
- *save*: el comando *save* genera un archivo de configuración de toda la programación guardada en VLC. Esto permitirá controlar que la programación es la adecuada en todo momento.
- *del*: con *del* se puede borrar todo aquel vídeo ya programado que se desee. Para ello hay que especificar el nombre de la instancia que VLC tiene asignado a tal vídeo.
- *logout*: para una desconexión segura de *Telnet* se debe usar este comando. Si ésta no es la adecuada, al querer conectarse nuevamente puede no hacerlo.
- *show*: muestra los elementos (broadcast y Schedule) cargados en el VLC y el estado de cada uno, de modo que se puede saber que vídeo se está emitiendo.

2.3 Duración de los vídeos

Esta es una característica indispensable en el proyecto, ya que debe conocerse para poder crear de forma correcta los elementos Schedule anteriormente comentados. Se necesita saber cuándo debe pararse la emisión de un vídeo tras comenzar ésta. Acompañando a la anterior exigencia, la idea del programa es crear un visor de contenidos programados (una escaleta gráfica) de forma que el usuario pueda ver qué franjas horarias están ocupadas por algún vídeo ya programado. La única conclusión es que debe hallarse un método eficaz de obtención de la duración de un vídeo.

Nuevamente, en el proyecto previo, se barajaron diversas opciones hasta alcanzar una solución aceptable:

- La primera consistió en intentar obtener la duración de un archivo de vídeo por medio del análisis de los datos de su cabecera.
- La segundo consistió en intentar obtener la duración mediante la obtención de los frames por segundo del vídeo y el nº de frames de éste, y relacionando ambos parámetros.
- La tercera fue una solución totalmente diferente, que consistió en hacer uso de un software externo llamado FFmpeg [5] (se detalla en el apartado 3.2.3).

La solución adoptada fue el uso del software, ante la imposibilidad de éxito con los dos primeros métodos.

En el proyecto actual se opta, nuevamente, por la solución del proyecto previo, ya que los resultados son difíciles de mejorar. Por un lado, FFmpeg es capaz de dar la duración del vídeo con resolución de centésima de segundo (resolución superior a la necesaria para el proyecto, que es de un segundo, debido a que VLC así lo limita). Por otro, posee un gran conjunto de librerías de códecs, por lo que es capaz de leer prácticamente cualquier formato de uso habitual.

Figura 2.3 Obtención de información de un vídeo con FFmpeg

En la figura anterior se puede observar cómo FFmpeg devuelve (entre otros datos) una duración de 00:03:02 segundos y 48 centésimas ante la petición de información de cierto vídeo, por línea de comandos.

2.4 Elección de la librería gráfica

El desarrollo de una interfaz gráfica potente compite en importancia con la manejabilidad y la fiabilidad de las operaciones internas que ésta realiza. Una buena interfaz debe ser simple, intuitiva y dinámica, a fin de que el usuario en cuestión sea capaz de exprimir sus posibilidades sin necesidad de revisar el manual de usuario repetidamente.

2.4.1 Opciones barajadas en proyecto previo

En el proyecto previo se barajaron las siguientes opciones:

- Búsqueda de aplicaciones libres del tipo "Google Calendar", que se pudieran utilizar directamente en el proyecto o de las cuales se pudieran extraer propiedades que facilitaran la creación de un calendario de contenidos gráfico. Se descartó esta opción por la falta de software libre relacionado con esta idea.

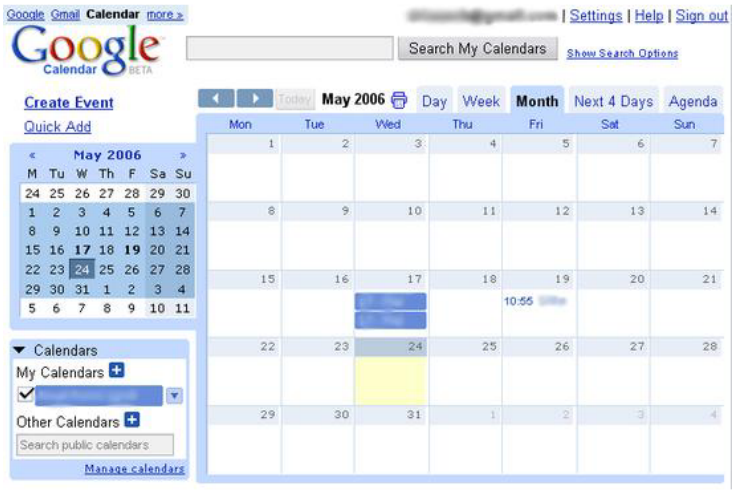


Figura 2.4 Google calendar

- *Java FX*: es una familia de productos de Sun Mycrosistems, en principio creada para aplicaciones web con características y capacidades de aplicaciones de escritorio, aunque está orientado a interfaces altamente animadas. En el fondo es la respuesta de Sun Mycrosistems para competir con Flash de Adobe. Se descartó la opción debido a la siguiente razón, citada en el proyecto previo.

  "El estudio de las posibilidades de Java FX para el desarrollo de la interfaz gráfica hizo ver que la finalidad de los productos estaba muy por encima de la idea de desarrollo de interfaz con la que se contaba. La decisión fue apartar la vista por la complejidad del mismo tratando de buscar una solución más acorde con nuestra necesidad."

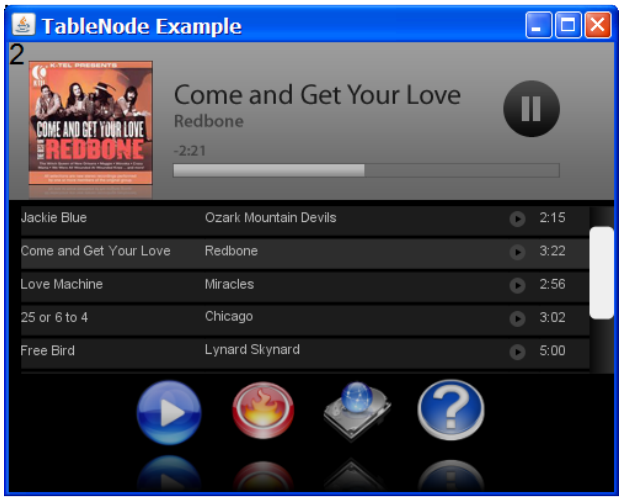


Figura 2.5 Java FX

- *Swing y AWT*: son las dos librerías gráficas principales de Java, prácticamente independientes al sistema operativo, y que permiten una visualización exacta independientemente de la plataforma. Se utilizan ambas, ya que se complementan para extender su funcionalidad. Ofrecen un conjunto de herramientas suficiente para crear interfaces competitivas, pero los resultados son poco vistosos, y éste fue el principal motivo por el que se desechó esta opción.

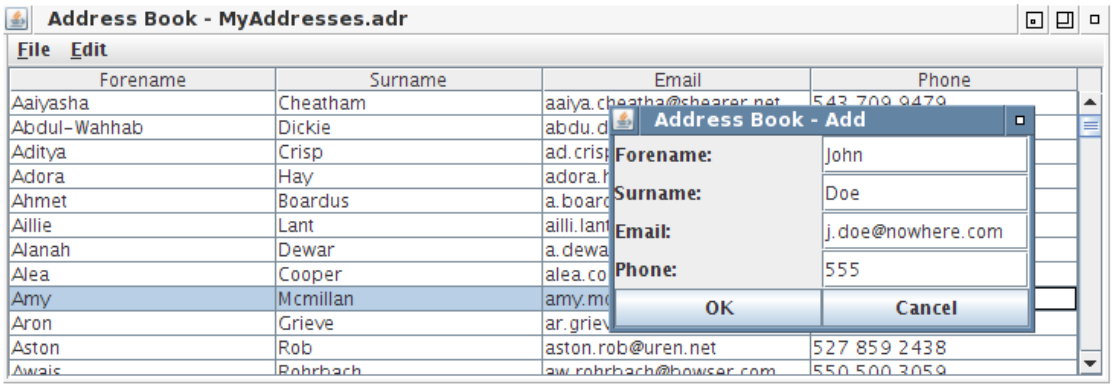


Figura 2.6 Swing y AWT

- *Interfacsia*: una librería gráfica para con poca presencia en la comunidad Java y sobre la cual no existe apenas documentación; se descartó su uso por falta de ésta.

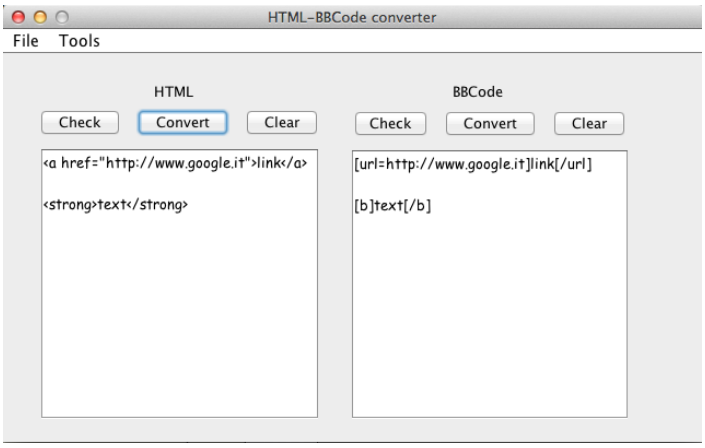


Figura 2.7 Interfacsia

- *SWT* [7]: es una librería para construir interfaces gráficas desarrollada por el proyecto Eclipse[1] [6]. Con ella se consigue que las interfaces tengan el mismo aspecto que las ventanas y los componentes de sus sistemas operativos; además ofrece grandes posibilidades y cuenta con diversa documentación y multitud de tutoriales.

[1] Eclipse: es un programa informático compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados.

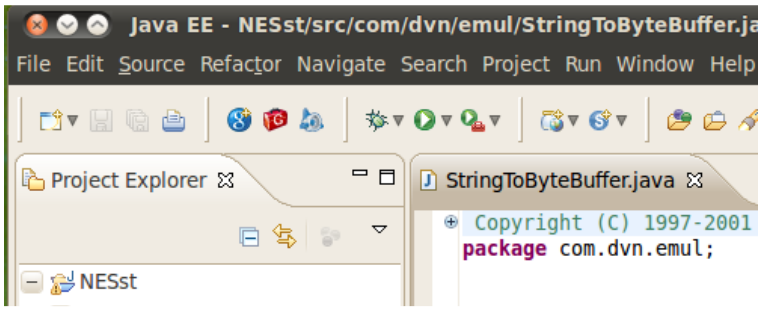


Figura 2.8 SWT

2.4.2 Opciones barajadas para la nueva versión

Se decidió realizar un nuevo intento con java FX, debido a que es una tecnología reciente, y habiendo pasado dos años desde la anterior versión de la aplicación se creyó que su uso y documentación habrían aumentado notablemente.

Tras varias pruebas, análisis de snippets[2], y vídeos demostrativos de la tecnología se concluyó que:

a) A pesar de ser una tecnología potente y apuesta sólida por parte de Oracle, no parece resultar una buena opción para aplicaciones de escritorio, sino más bien para aplicaciones de internet enriquecidas; es decir, para las tareas que viene haciendo flash (entre otros) en los últimos años.

b) Posee buenas herramientas de dibujo y animación, que superan con creces las exigencias de dibujo de un calendario de contenidos gráfico simple. Sin embargo, no posee una librería tan completay sencilla de widgets gráficos (botones, tablas, listas, etc.) como SWT o Swing.

Descartada la tecnología Java FX, la segunda apuesta para el proyecto fue SWT, y con ella se ha proseguido hasta su conclusión.

Aunque SWT es parecida, en cuanto a prestaciones, a Swing y AWT, se consiguen resultados mucho más vistosos, ya que los componentes (botones, campos de texto, etc.) cogen el aspecto por defecto del sistema operativo, y se pierde esa sensación de “programa Java” que se tiene al utilizar Swing, que gráficamente es más plano y aburrido.

2.4.3 Elección final: librería gráfica SWT

SWT es una librería gráfica para crear interfaces gráficas en Java (como Swing o Awt) que crea a través de JNI (Java Native Interface) interfaces nativas del Sistema Operativo en donde se ejecute la aplicación SWT en cuestión. Esto quiere decir que con el mismo código se visualiza en cada Sistema Operativo la ventana como si hubiera sido creada para ese SO en específico.

Por ejemplo:

[2] Fragmentos de código que generalmente se usan a modo de ejemplo o demostración.

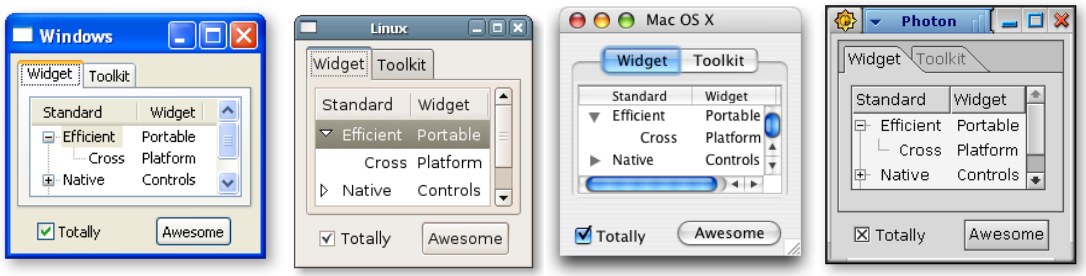


Figura 2.9 SWT en diferentes SO

Ventajas:

- Ya que crea nativamente la GUI dependiendo del SO, es más rápido que Swing.
- Consume menos recursos que Swing.
- La interfaz gráfica se verá como las demás aplicaciones del SO.
- Está en constante desarrollo.

Desventajas:

- Mucha menor cantidad de documentación que para Swing.
- Se tiene que agregar la biblioteca al proyecto a desarrollar; no viene por defecto en las librerías de Java.

Algunas aplicaciones creadas con SWT:

- Eclipse IDE[3] [6] (Uno de los más completos IDEs para desarrollo en Java, entre otros).
- Azureus (Uno de los más conocidos clientes torrent).

SWT se compone de algunos principales elementos, que interactúan entre ellos para lograr la interfaz nativa que tanto le caracteriza:

- *JNI*: Java Native Interface, es el primer y más bajo nivel, que interactúa directamente con el Sistema Operativo.
- *Display*: esta clase de SWT es la responsable de traducir o comunicar la Clase Shell con el JNI.
- *Shell*: es una clase de SWT, que representa una ventana, siendo la responsable de administrar los widgets (componentes) que se tienen en la misma. Se tiene una instancia de la clase Shell por cada ventana en el proyecto. Para poder instanciar la clase Shell se debe hacer referencia a un Display anteriormente instanciado.

La forma de trabajar de la librería SWT consiste en un único hilo, que monitoriza constantemente 2 factores. El primero es si la ventana declarada se ha cerrado; en tal caso termina el código relativo a la GUI, desechando el objeto Display antes mencionado. El

[3] Entorno de desarrollo integrado. Es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien puede utilizarse para varios.

segundo es si ha ocurrido algún evento en la GUI (por ejemplo, un clic de ratón en alguno de los widgets de la ventana); en este caso se llama al manejador[4] correspondiente a tal evento.

A continuación se muestra un ejemplo de código Java que crea una ventana de nombre "Primera GUI en SWT", y se hace hincapié en ciertas partes del código:

```
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
publicclass PrimeraGUIenSWT {
      publicstaticvoid main(String str[]){
            Display display =new Display();
            Shell shell =new Shell();
            shell.setText("Primera GUI en SWT");
            shell.setVisible(true);
            shell.setSize(400,300);
            shell.open();
            while(!shell.isDisposed())
            {
                  if(!display.readAndDispatch())
                        display.sleep();
            }
            display.dispose();
      }
}
```

a) Se importan las clases necesarias al proyecto:

```
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
```

b) Se instancia primero un objeto de la clase Display y luego otro de la clase Shell:

```
Display display =new Display();
Shell shell =new Shell();
```

c) Se da ciertas propiedades al Shell (en este caso, le damos nombre, la hacemos visible y le asignamos dimensión en píxeles) y finalmente se "abre" (esto provoca que aparezca la ventana):

```
shell.setText("Primera GUI en SWT");
shell.setVisible(true);
shell.setSize(400,300);
shell.open();
```

d) Se crea un bucle del que sólo se sale cuando la ventana es cerrada. Este bucle espera a que ocurran eventos en los widgets declarados dentro de la ventana, y llama a los manejadores pertinentes cuando tales eventos (por ejmplo, un clic de ratón) suceden. Cuando se sale del bucle, se desecha el display antes creado.

[4] Se entiende como manejador, en este contexto, al objeto Java que implementa una interfaz de escucha de determinados eventos responsable de responder ante éstos.

```
while(!shell.isDisposed())
	{
		if(!display.readAndDispatch())
			display.sleep();
	}
display.dispose();
```

3 Estructura general del sistema de gestión

En este capítulo se presenta la estructura general del proyecto, sin entrar a detalles demasiado técnicos (tales como código, configuraciones, etc.).

Se explica brevemente qué software entra en juego, qué funcionalidades tiene y cuáles de ellas se han usado en la aplicación. También se especifica cómo interactúan las diferentes piezas de software entre sí, con el fin de generar una idea generalizada, pero clara y concisa a la vez, del funcionamiento del sistema de gestión.

3.1 Esquema gráfico

Se presenta en la figura 3.1 la estructura básica del sistema de gestión de la parrilla de TV, con el objetivo de que los apartados siguientes sean más digeribles y fáciles de entender para el lector, nada más que remitiéndose a esta imagen.

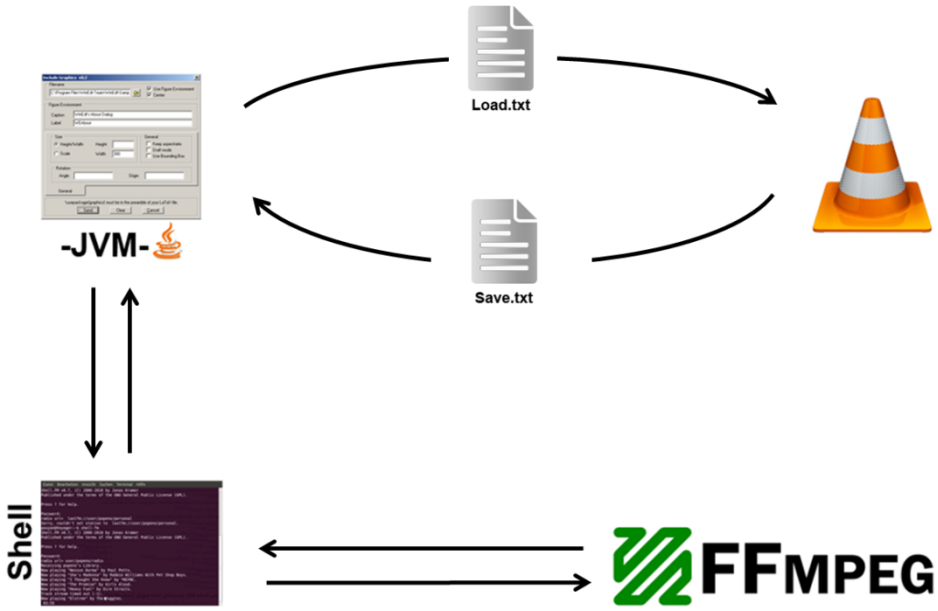


Figura 3.1 Estructura general del sistema de gestión

En la figura, que muestra la estructura interna de la aplicación de forma generalizada, pueden observarse los siguientes elementos:

- *Aplicación Java*: corre sobre la JVM. A lo largo de la memoria se hará distinción entre "aplicación", que engloba todas las piezas de software en interacción, y "aplicación Java", que hace referencia a la parte de la aplicación programada en Java. Es la encargada de generar una GUI y de procesar la información que el usuario transmite a través de ésta, así como de conectar con las diferentes piezas de software para poder crear un sistema funcional.

- *Consola/terminal/shell*: todas hacen referencia al intérprete de comandos del sistema operativo.

- FFmpeg [5]: se detalla en el punto 3.2.3.

- VLC [4]: se detalla en el punto 3.2.4.
- 2 ficheros de texto, de nombre Load.txt y Save.txt.

3.2 Software requerido

Es importante tener en cuenta que el sistema operativo (SO) sobre el que se ha desarrollado (y para el que está dirigido) este proyecto ha sido la distribución GNU/Linux Ubuntu, en su versión 12.04 LTS, aunque también ha sido objeto de éste ser suficientemente versátil como para ejecutarse correctamente en otras versiones del SO.

A continuación se lista y se explica el software que compone el proyecto.

3.2.1 JDK

Es la infraestructura software y conjunto de herramientas que nos permiten crear y correr aplicaciones programadas en Java en nuestro equipo. Contiene la Java Virtual Machine (JVM), que es la máquina virtual que permite a los programas Java (Java bytecode) correr bajo cierta máquina física. Nos servirá para correr la aplicación, puesto que está programada en lenguaje Java.

3.2.2 Shell

La shell es un intérprete de comandos nativo del SO (en el caso de GNU/Linux); es la herramienta que permite comunicarse con el núcleo del sistema operativo mediante la línea de comandos.
Entre sus cometidos dentro del proyecto están:

- Instanciar VLC y FFmpeg cuando sea necesario.
- Encontrar procesos asociados a VLC.
- Matar procesos asociados a VLC.
- Comprobar disponibilidad de puertos.

3.2.3 FFmpeg

FFmpeg es una colección de software libre que puede grabar, convertir (transcodificar) y hacer streaming de audio y vídeo. Está orientado a su uso a través de la línea de comandos, de modo que se hace uso de la shell para comunicarse con este software. Es utilizado, dentro del proyecto, para tareas tales como obtener información acerca de los vídeos (duración, formato contenedor, códec, etc.), así como para determinar si un determinado fichero del ordenador es vídeo, audio o ninguno de ellos.

3.2.4 VLC

VLC media player es un reproductor y framework multimedia libre de código abierto desarrollado por el proyecto VideoLan.

Posee la capacidad de streaming, además de una herramienta para la gestión de ésta (llamada VLM, de "Video Lan Manager"); se puede interactuar con ella a través de una interfaz telnet del propio VLC. Se hace uso de tal herramienta en este proyecto.

3.3 Comunicaciones entre software

Se detallan a continuación los tipos de comunicación que se llevan a cabo dentro del esquema general (figura 3.1).

3.3.1 Comunicación entre aplicación Java y Shell

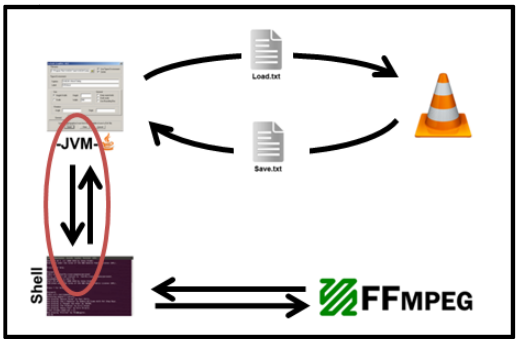


Figura 3.2 Conexión aplicación Java-Shell

Se consigue a través de servicios que nos provee la clase "Process" de Java (deriva de java.lang.Object).

Es una comunicación bidireccional, ya que desde la interfaz gráfica (GUI) se le dice a la terminal (o shell) qué instrucción debe ejecutar, y posteriormente se lee la salida que ésta da.

El envío sirve para:

- Instaciar VLC.
- Llamadas a FFmpeg.
- Solicitar los puertos activos al SO.
- Matar procesos VLC.

La recepción sirve para leer la respuesta proveniente de FFmpeg y procesarla.

Ha de tenerse en cuenta que esta comunicación es un proceso transparente para el usuario, ya que la Shell corre internamente sobre el SO, sin necesidad de mostrar resultados gráficamente.

3.3.2 Comunicación entre shell y FFmpeg

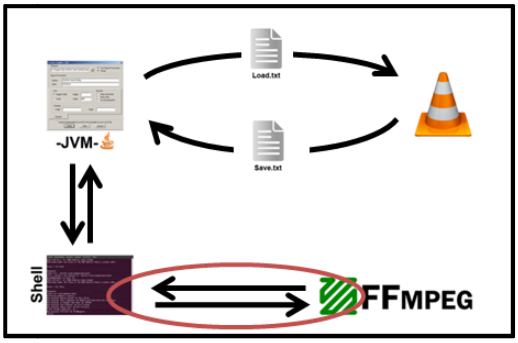


Figura 3.3 Conexión Shell-FFmpeg

La shell instancia FFmpeg (que no tiene GUI) con información añadida y recibe su salida, por lo que también es considerada una comunicación bidireccional. Dado que el proceso de la shell es transparente para el usuario, al final aparenta ser una comunicación directa entre la aplicación Java y el FFmpeg, del que tampoco tiene por qué ser consciente el usuario.

En resumen, toda la comunicación referente al FFmpeg resulta en un proceso automático y transparente para el usuario, que simplemente ve cómo campos de información de la GUI acerca del vídeo importado son rellenados por el propio programa.

En la siguiente figura (3.4) puede observarse un ejemplo de la respuesta dada por el software FFmpeg tras ser solicitada información acerca de cierto vídeo, a través de la línea de comandos (se ha realizado esta prueba desde la versión gráfica de la shell; este proceso no se vería al realizarse desde la aplicación Java). Comprobamos que tras varios datos no relevantes para este caso, se aportan datos como duración, tipo de códec, etc.

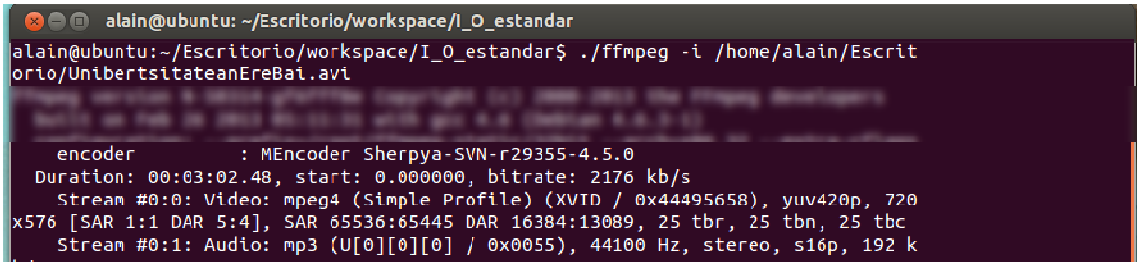


Figura 3.4

3.3.3 Comunicación entre aplicación Java y VLC

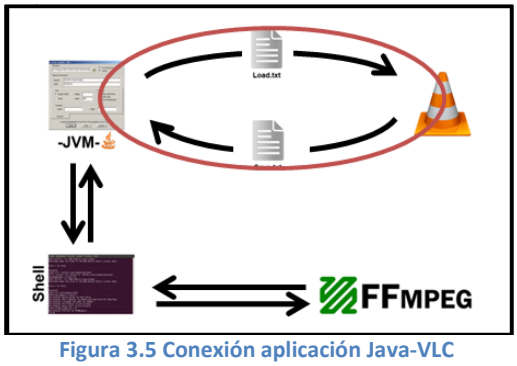


Figura 3.5 Conexión aplicación Java-VLC

Prerrequisitos:

- VLC debe haber sido instanciado con interfaz telnet activa.
- Debe haberse establecido conexión telnet.

A partir de tal conexión (que se describe más adelante en la memoria) se puede considerar una comunicación unidireccional, ya que se da órdenes a la herramienta VLM del VLC, y se confía en que ésta cumpla con su cometido.

Básicamente, los comandos que se utilizan y que interpretará la herramienta VLM son:

- load “Load.txt”
- save “Save.txt”

En los puntos 3.3.4 y 3.3.5 se detalla qué es cada uno de los archivos “.txt”’, y qué funciones cumple dentro del proyecto.

3.3.4 Load.txt

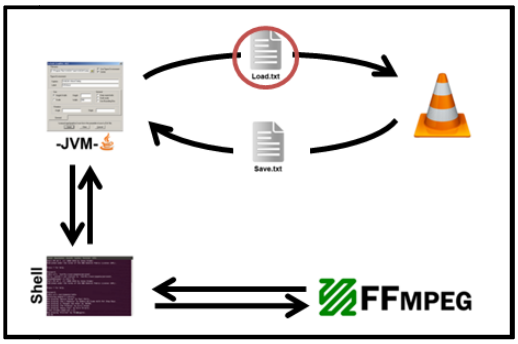


Figura 3.6 Load.txt

Load.txt es el fichero de texto en el que debe aparecer toda la programación/configuración para el streaming de vídeo escrita de tal modo que la herramienta VLM del VLC pueda interpretarla.

En este fichero tan sólo hay información del tipo visto en el apartado 2.1 (Código VLM).

El objetivo de la aplicación es crear este fichero, valiéndose de la entrada de datos por parte del usuario (tipo de stream, hora de inicio, duración, etc.).

Se carga vía telnet en el VLC, y este último adquiere el comportamiento que se le haya especificado en el fichero.

3.3.5 Save.txt

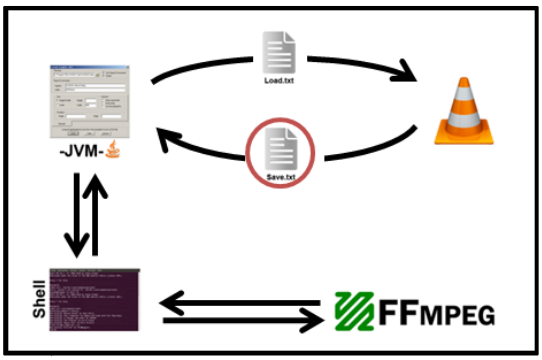


Figura 3.7 Save.txt

Es el fichero que podemos pedir a la herramienta VLM que genere; muestra la configuración de todo el streaming programado en ese instante.

En teoría debiera tener la misma configuración (acerca del streaming) que Load.txt, y sirve para asegurar que la configuración se ha cargado correctamente.

Sirve, una vez procesado por la aplicación, para refrescar la parte de la GUI que muestra los contenidos programados gráficamente, así como para el refresco del listado de vídeos por fecha y hora. También sirve para guardar la configuración del proyecto, a modo de copia de seguridad, y poder cargarlo en una nueva instancia de la aplicación.

4 Modelado de la aplicación Java con UML

En este capítulo se explica qué elementos se han utilizado para la elaboración de un modelo en el cual se ha basado, posteriormente, la aplicación programada en Java. Los apartados de este capítulo muestran un listado de componentes que intentan describir, en la medida de lo posible, la estructura y comportamiento de la aplicación.

Debido al lenguaje y a su filosofía (orientación a objetos), se consideró oportuno la realización de un modelo UML [2] que describiera un sistema basado en objetos, interactuando entre sí.

UML (Lenguaje Unificado de Modelado) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados.

En este proyecto han tomado relevancia dos de las características principales del UML; por un lado los llamados “elementos estructurales”, que ayudan a comprender la estructura de objetos que componen el sistema desde un punto de vista estático; por otro lado los “casos de uso”, que ayudan a ver el sistema desde un punto de vista más dinámico, mediante comportamientos y acciones de éste.

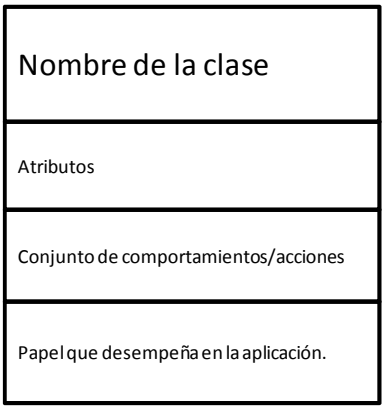
4.1 Elementos estructurales

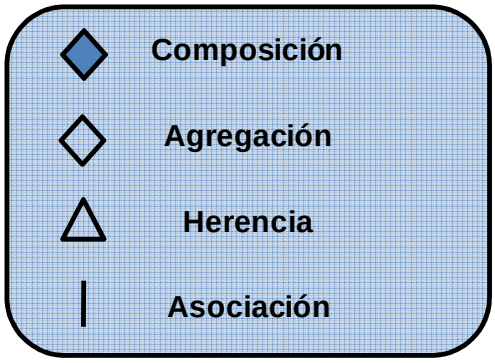
Los elementos estructurales, como su propio nombre indica, son aquéllos que dan estructura al sistema, y que en conjunción, consiguen formar un todo.

En el paradigma orientado a objetos, cada ente que posee atributos y/o comportamientos es denominado clase. De cada clase pueden instanciarse objetos, que son casos particulares de la clase; es decir, poseen valores específicos para los atributos ésta.

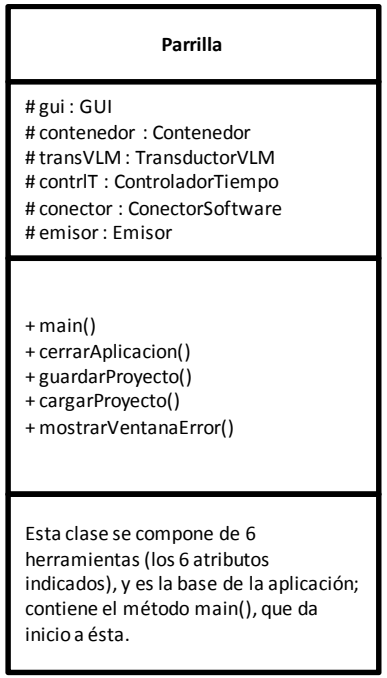
La clase puede verse como un tipo de variable, y el objeto como la variable en cuestión, la cual posee un determinado valor.

No es objeto de la presente memoria describir las normas del lenguaje, o el modo en que se representan sus componentes, por lo que se pasa directamente a la presentación de diagramas. Aun así, cabe destacar el modelo de diagrama utilizado para la representación de las clases, así como la leyenda para algunos diagramas venideros:





4.1.1 La clase Parrilla



La clase Parrilla se compone de 6 objetos de 6 clases diferentes (que se describen en puntos posteriores). Es el punto de entrada de la aplicación, ya que ésta se inicia con el método main(), declarado en esta clase.
Posee algunos métodos de carácter general en la aplicación:

- *cerrarAplicación*: encargado de matar los procesos asociados a VLC instanciados desde la interfaz y de liberar memoria, cuando la aplicación se cierre.
- *guardarProyecto*: encargado de proporcionar la opción de seleccionar un destino y guardar un fichero de texto con la configuración de la emisión.
- *cargarProyecto*: método que permite cargar un fichero de texto con formato adecuado (el guardado con la función anterior) y recuperar la información de la emisión que previamente se haya guardado.
- *mostrarVentanaError*: permite mostrar ventanas de aviso con mensajes personalizados para el cada caso en particular.

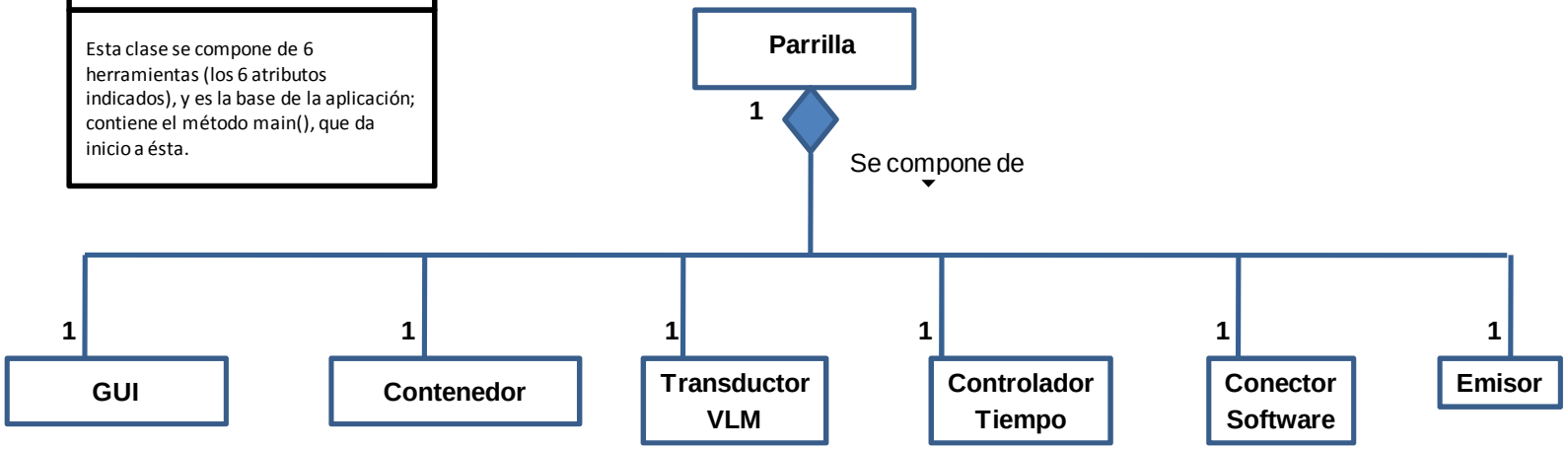
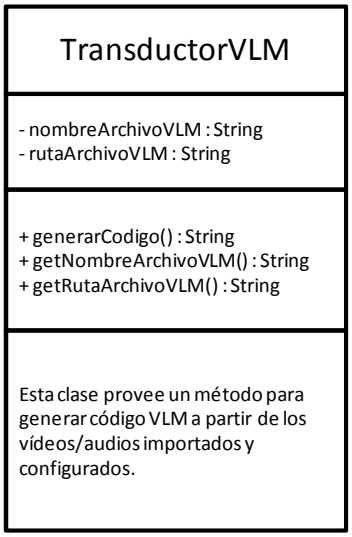


Figura 4.1 Estructura UML de la aplicación Java

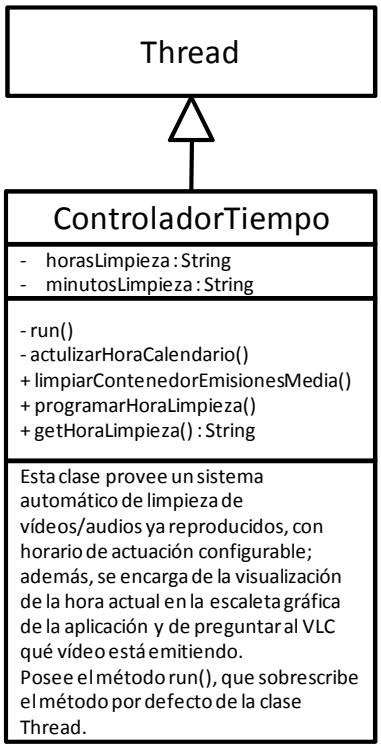
4.1.2 La clase TransductorVLM



La instancia de esta clase es la encargada de proveer un método general de escritura de código VLM; concretamente, *generarCodigo()*.

Los atributos de ruta y nombre del archivo VLM son relativos al fichero Load.txt, mencionado en el punto 3.3.4.

4.1.3 La clase ControladorTiempo



La clase ControladorTiempo deriva de la clase Thread de Java, que permite iniciar un nuevo hilo[5] de ejecución, independiente de aquél en el que fue creado. El método *run()* inicia la ejecución del hilo.

La instancia de esta clase cumple con varios propósitos dentro de la aplicación:

a) Monitorizar si la instancia VLC asociada a la aplicación (instancia que hace de servidor de streaming) sigue viva.

b) Llevar el control de la hora del sistema, y limpiar e contenedor de emisiones ya reproducidas (si procede); método *limpiarContenedoremisionesMedia()*.

c) Monitorizar cuál es el vídeo que se está emitiendo.

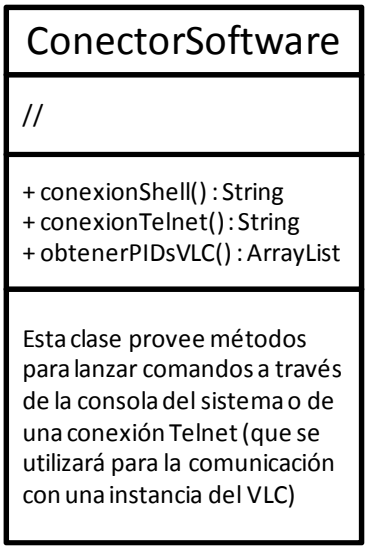
d) Crear fichero de seguridad de la emisión (transparente para el usuario).

e) Dibujar hora en pantalla; método *actualizarHoraCalendario().*

[5] Un hilo es simplemente una tarea que puede ser ejecutada al mismo tiempo con otra tarea. Los distintos hilos de ejecución comparten una serie de recursos tales como el espacio de memoria, los archivos abiertos, situación de autenticación, etc.

Los métodos *programarHoralimpieza()* y *getHoraLimpieza()* sirven para asignar una nueva hora de actuación (por defecto, las 00:00h) y ver la hora programada para la actuación, respectivamente.

4.1.4 La clase ConectorSoftware



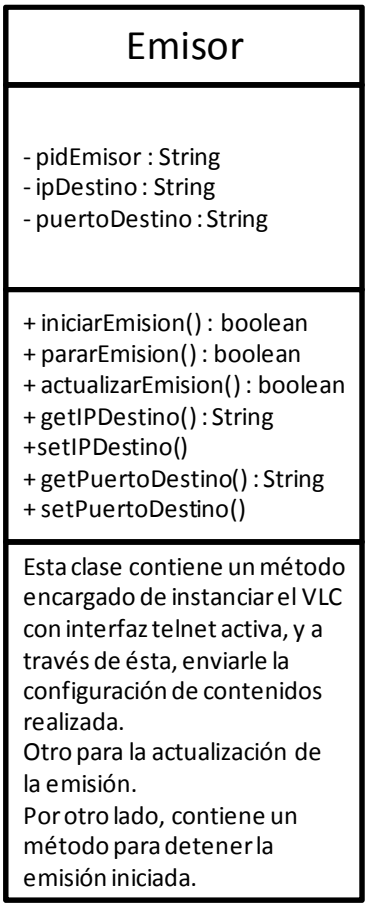
Es una clase sin atributos, y pone a disposición de los objetos de las demás clases 3 métodos de utilidad para comunicarse con el diverso software que toma parte en el sistema:

- *conexionShell()*: permite conectar con la consola del sistema; tan sólo hay que decirle al método qué comando se quiere que la Shell ejecute.

- *conexionTelnet()*: permite conectar con la interfaz telnet de la instancia de VLC que hace de servidor de streaming; hay que indicar el comando a ejecutar por parte del VLC.

- *obtenerPIDsVLC()*: método de utilidad para diversas situaciones, que devuelve todos los identificadores de procesos relativos a VLC. Sirve, por ejemplo, para determinar qué instancias de VLC hay activas antes de crear una por parte de la aplicación.

4.1.5 La clase Emisor



La instancia de esta clase posee los comportamientos y parámetros necesarios para un correcto inicio y direccionamiento de la emisión.

- *iniciarEmision()*: crea una nueva instancia de VLC y asigna el PID de tal proceso a *pidEmisor*.

- *pararEmision()*: mata el proceso relativo al PID *pidEmisor*.

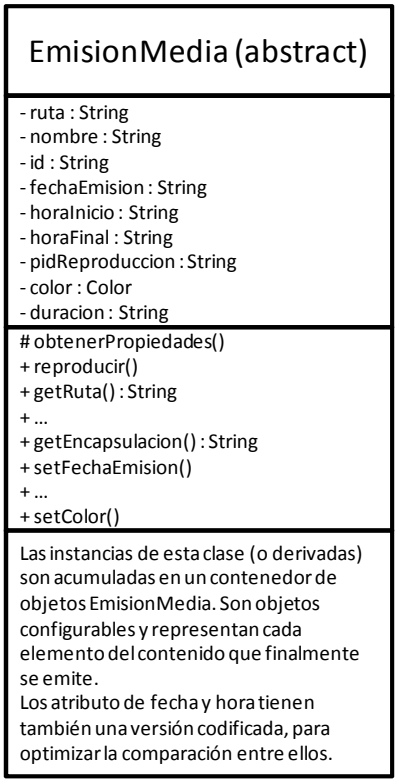
- *actualizarEmision()*: vuelve a generar el código VLM (que se guarda en Load.txt) y lo carga en el VLC (por medio de la herramienta VLM).

- ipDestino: contiene la IP a la que debe ir dirigido el stream emitido por el VLC (se declara en el Load.txt).

- *puertoDestino*: contiene el puerto al que debe ir dirigido el stream emitido por el VLC (se declara en el Load.txt).

Los atributos *ipDestino* y *puertoDestino* se asignan y consultan con los métodos “set” y “get”, respectivamente.

4.1.6 La clase abstracta EmisionMedia



Ésta puede considerarse la clase con mayor personalidad en el sistema de gestión, ya que representa a cada vídeo/audio importado en el proyecto.

Realmente, cuando se "importa un vídeo/audio" en el proyecto (en esta aplicación), no se importa como tal un fichero de vídeo o de audio, sino que se crea una representación de tal fichero; esa representación viene dada por una instancia de esta clase. Es decir, importar un vídeo/audio en este proyecto es igual a crear un objeto de tipo EmisionMedia.

Los atributos de la clase son características que representan tanto al vídeo/audio como a su posterior emisión. Así pues, pueden verse las siguientes características:

- *ruta*: es la ruta en la que se encuentra el fichero de vídeo o audio importado.

- *nombre*: nombre del fichero.

- *id*: un identificador único para cada objeto EmisionMedia instanciado.

- *fechaEmision*: es la fecha en la que se hace streaming del fichero.

- *horaInicio*: hora en la que empieza a hacerse streaming del fichero.

- *horaFinal*: hora en la que acaba el streaming del fichero.

- *pidReproduccion*: PID de la instancia VLC en la que se hace preview del fichero (sólo si el usuario solicita el preview).

- *color*: color que representa a la emisión del fichero en la escaleta gráfica de contenidos. El usuario puede seleccionar el que desee desde una paleta de colores; se ve más adelante en la memoria.

- *duracion*: duración del vídeo/audio importado.

Los métodos de la clase son los necesarios para establecer y consultar cada atributo ("set" y "get", respectivamente). También hay dos métodos con una finalidad diferente:

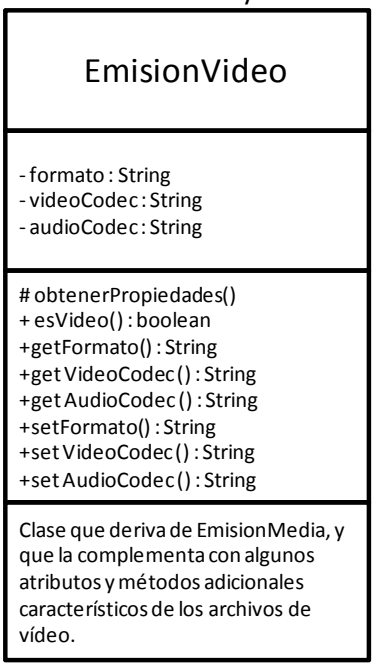
- *reproducir()*: es el que realiza el preview del fichero, y asigna el PID de la instancia VLC al atributo pidReproduccion.

- *obtenerPropiedades()*: es un método abstracto, por lo que el código lo deben implementar aquellas clases que deriven de ésta. Su finalidad está dirigida a obtener los valores de los atributos de la clase relativos al fichero de vídeo/audio (ruta, nombre, id y duración).

4.1.7 Las clases derivadas de EmisionMedia

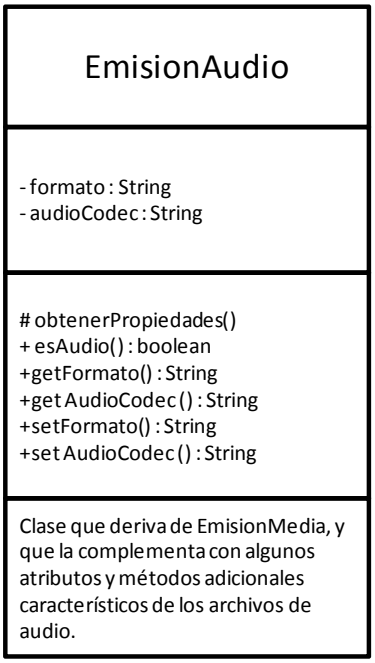
Dado que EmisionMedia es una clase abstracta, no pueden crearse objetos de ella. Sí, en cambio, de aquellas clases que deriven (es decir, sean hijas) de EmisionMedia y que implementen los métodos abstractos de ésta. Sólo hay un método abstracto en EmisionMedia, que es el último visto, obtenerPropiedades().

Son dos las clases que cumplen los requerimientos del párrafo anterior, y han sido bautizadas como EmisionVideo y EmisionAudio.



Es la clase que se utiliza cuando se importa un vídeo a la aplicación; se crea un objeto de esta clase y se llama al método *obtenerPropiedades()* para obtener los siguientes atributos: ruta, nombre, id, duración, fomato, videoCodec y audioCodec. Debe tenerse en cuenta que los atributos del cuadro de la izquierda se suman a los de la clase padre (los de EmisionMedia).

En cuanto a los métodos, la única novedad con respecto a su clase padre es que implementa el código de *obtenerPropiedades()* y que incorpora la función *esVideo()*, un método estático para determinar si cierto fichero es un archivo de audio o no. Finalmente, se añaden los métodos correspondientes a la asignación y consulta de los atributos añadidos por la propia clase (los ya mencionados "set" y "get").



Es la clase análoga a EmisionVideo pero para los ficheros de audio.

Las diferencias son que no posee el atributo *codecVideo*, y que en vez del método *esVideo()* implementa *esAudio()*, para determinar si un determinado fichero es de audio o no.

4.1.8 La clase Contenedor

Contenedor
- contenedorMedia : ArrayList
+ insertarEmisionMedia() : boolean + borrarEmisionMedia() : boolean + getContenedorMedia() : String + vaciarContenedor()
Provee un contenedor para los objetos EmisionMedia que se importan al proyecto.

Clase cuya función es ser el almacén temporal de los objetos EmisionMedia; se entiendo por objetos Emisionmedia a los EmisionVideo y emisionAudio, que son los realmente instanciables.

El atributo de tipo Arraylist es su principal característica, donde se añaden los objetos Emisionmedia cuando se han creado, y de donde se eliminan cuando ya han sido emitidos o borrados por el usuario.

Sus métodos están dirigidos a las operaciones de inserción y eliminación de objetos:

- insertaremisionmedia(): inserta un objeto Emisionmedia en el arrayList.
- borrarEmisionMedia(): elimina cierto objeto Emisionmedia del arrayList.
- getContenedorMedia(): entrega un listado con el contenido del contenedor.
- vaciarContenedor(): vacía todo el contenedor.

A modo orientativo se presenta el siguiente diagrama, que muestra la relación entre los Objetos Emisionmedia y el contenedor:

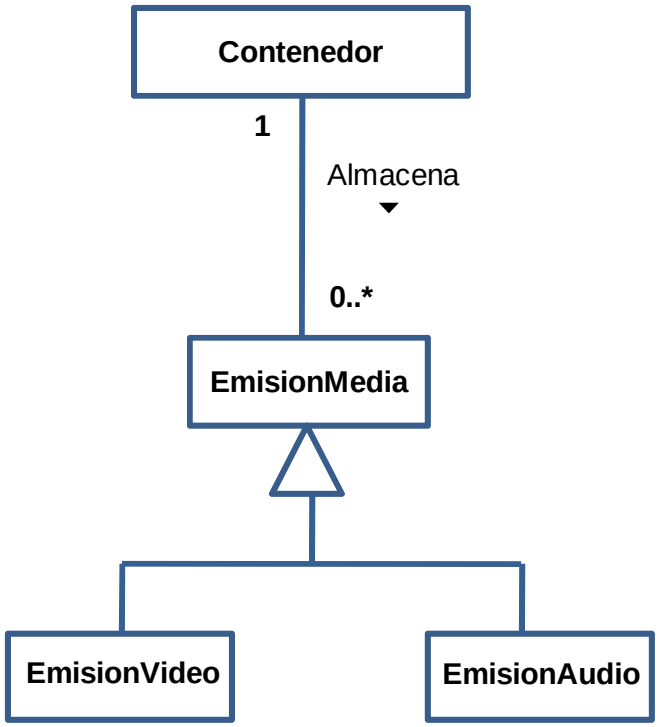


Figura 4.2 contenedor de objetos EmisionMedia

4.1.9 La clase GUI

GUI
<<Objetos gráficos>> # widget 1 : Widget … # widget N : Widget <<Manejadores de eventos>> # manejador1 : Manejador1 … # manejadorM : ManejadorM
- iniciarInterfaz() - crearContenidos()
Declara y crea los objetos gráficos necesarios (controles, contenedores, etc.) que componen la interfaz gráfica, así como los objetos manejadores, que son los encargados de responder a los eventos generados por los elementos gráficos. Es una clase que se mantiene "a la escucha".

Aunque se detalla más adelante en la memoria, cabe mencionar que la instancia de la clase GUI es la encargada de crear todo el sistema gráfico de la aplicación.

Los widgets son objetos de la librería SWT [7], y son elementos gráficos que se visualizan dentro de la ventanas del programa. Cada uno tiene funcionalidades especializadas. Son ejemplos de widgets las tablas, botones, menús, etc.

Los métodos de esta clase son:

- iniciarInterfaz(): crea un objeto de tipo Display y otro de tipo Shell (explicados el el punto 2.4.3), elementales para la inicialización de la ventana gráfica de la aplicación.

- crearContenidos(): instancia los widgets que finalmente se ven en la ventana gráfica.

La parte gráfica de la aplicación se analiza con más detalle en un capítulo posterior.

ManejadorXX
- gui : GUI
<<métodos declarados en las interfaces de escucha que la clase implemente>> + Métodos que se consideren necesarios.
Esta clase implementa las interfaces de escucha necesarias (e.g. SelectionListener) para manejar eventos generados por Widgets gráficos de una determinada zona de la interfaz gráfica (zona XX, donde XX puede ser, por ejemplo, la parte del explorador de archivos de la GUI). Habrá un manejador por cada zona caracterizada en la GUI.

Son clases que implementan interfaces de escucha de eventos para hacerse cargo (o manejar) de los eventos generados por los widgets de ciertas partes de la GUI.

La "XX" debe sustituirse por el nombre de la parte de la GUI sobre la cual actúe la clase manejadora en cuestión. De este modo, son clases manejadoras, por ejemplo: ManejadorMenu, ManejadorExplorardor, etc.

Se detallen en un capítulo posterior, por lo que no se entra al detalle en este punto.

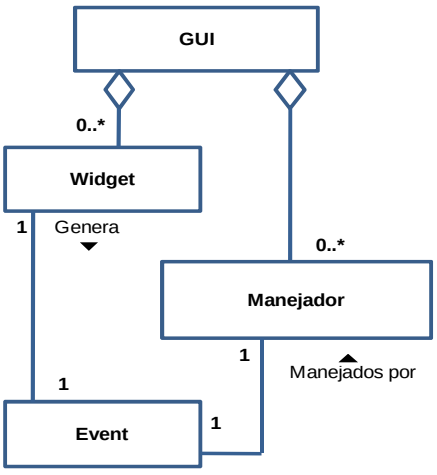


Figura 4.3 composición clase GUI

4.1.10 Gráfico-resumen de las clases del sistema de gestión

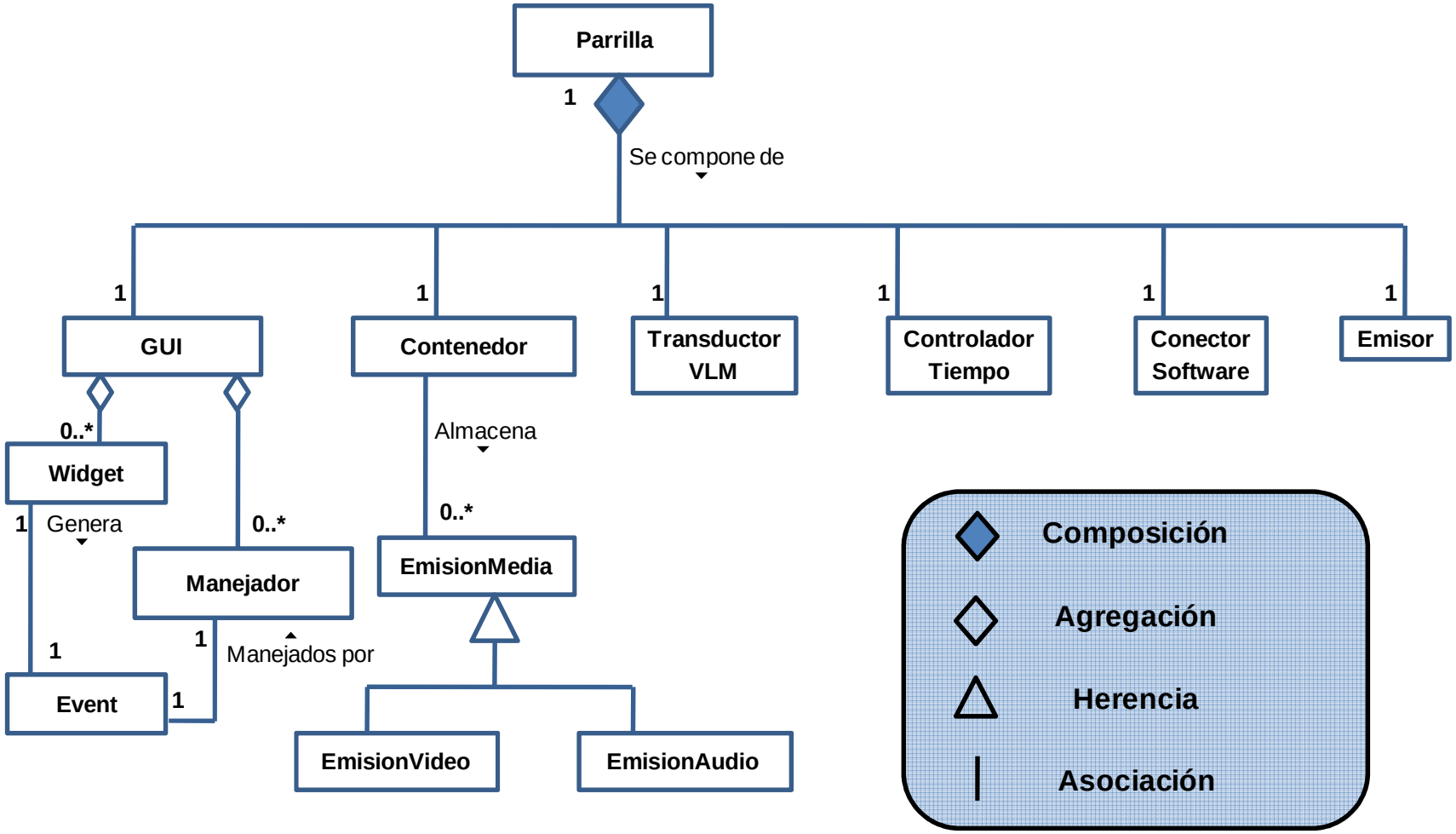


Figura 4.4 resumen modelo UML de la aplicación Java

4.2 Casos de uso

Los casos de uso son conjuntos de acciones que se dan en un determinado contexto o situación, y que tienen por objeto dar a conocer el comportamiento del sistema que se quiere describir. Normalmente, la acción que desencadena un caso de uso viene dada por un actor, que suele ser un individuo u otro sistema.

Un ejemplo de sistema puede ser una lámpara con un único interruptor de on/off; para tal sistema los dos casos de uso principales serían "pulsar on" y "pulsar off", con sus correspondientes sucesos asociados (la bombilla recibe corriente, da luz, etc.). Sin embargo, los casos de uso pueden extenderse, obteniendo, por ejemplo, "pulsar on con la lámpara no enchufada". El actor en este caso sería la persona que está ante la lámpara, dispuesta a usarla.

La descripción completa del comportamiento de un sistema requiere, por lo general, de un gran número de casos de uso, y puede acabar en un trabajo tedioso. Para la descripción de la aplicación Java se ha optado por el análisis de un conjunto básico de casos de uso, que pueden verse en la siguiente figura:

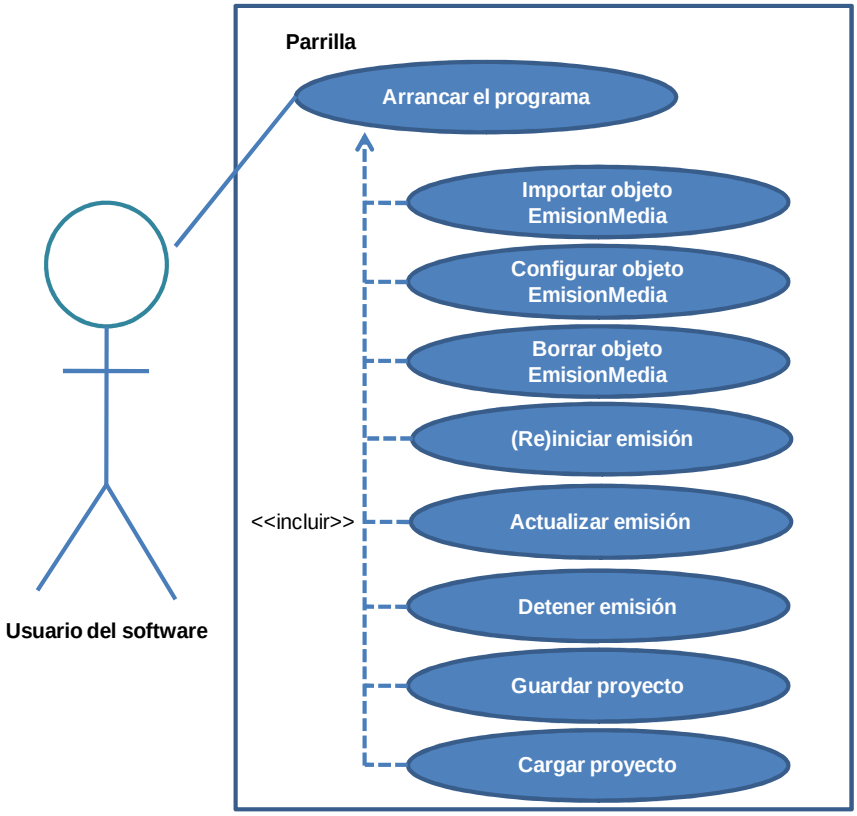


Figura 4.5 casos de uso

Todos los casos de uso indicados incluyen, implícitamente, que el programa esté en marcha.

A continuación se muestran todos los casos de uso de forma desglosada, mediante diagramas de actividades, que sirven a modo de diagramas de flujo del programa. Se incorpora junto a cada diagrama las clases que toman parte en el caso de uso; mediante un código de colores se asocian los métodos de estas clases a cada actividad del diagrama (es decir, si un método posee el mismo color que una actividad, ese método ha sido ejecutado para esa acción.

4.2.1 Arrancar el programa

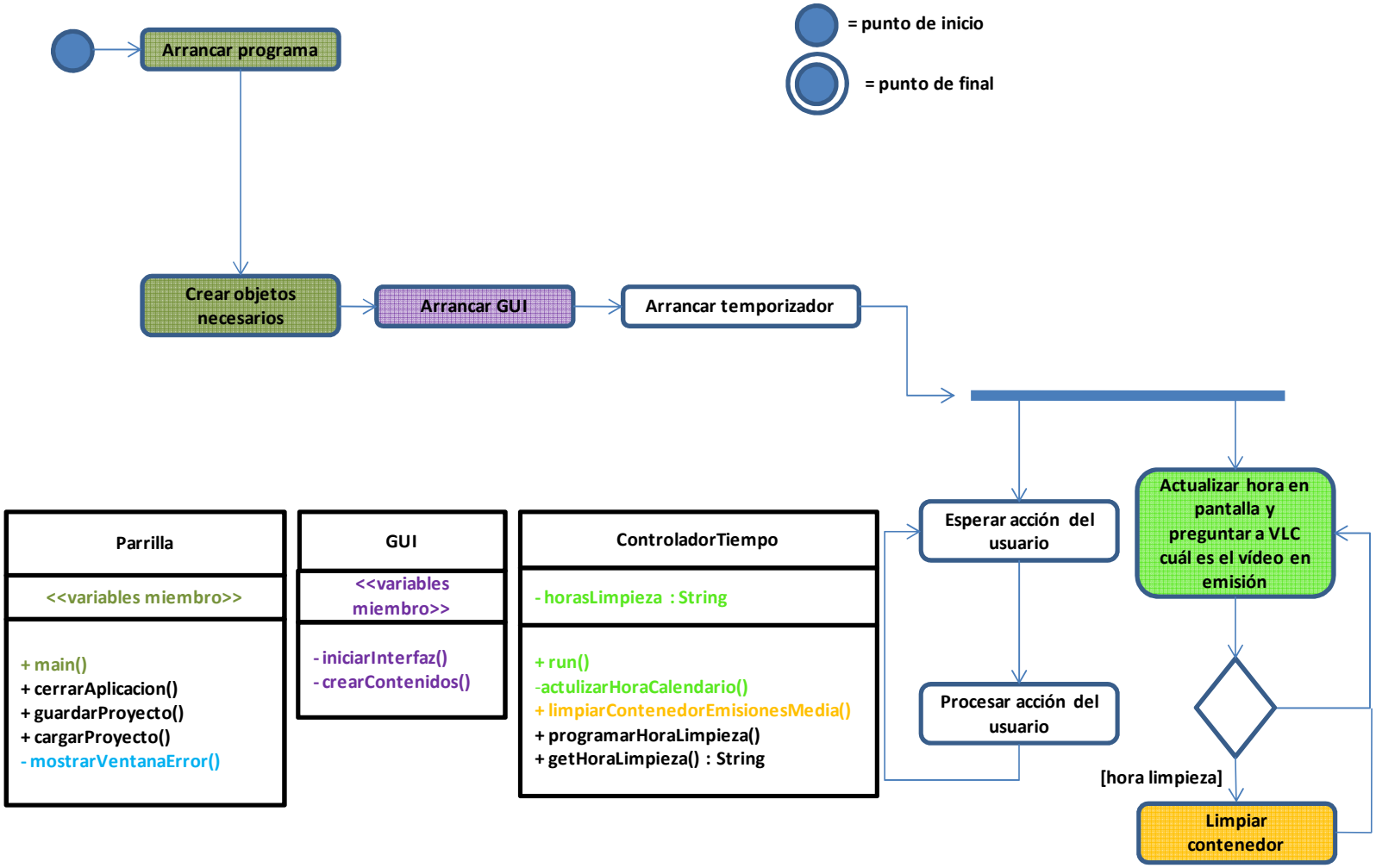


Figura 4.6

4.2.2 Importar objeto EmisionMedia

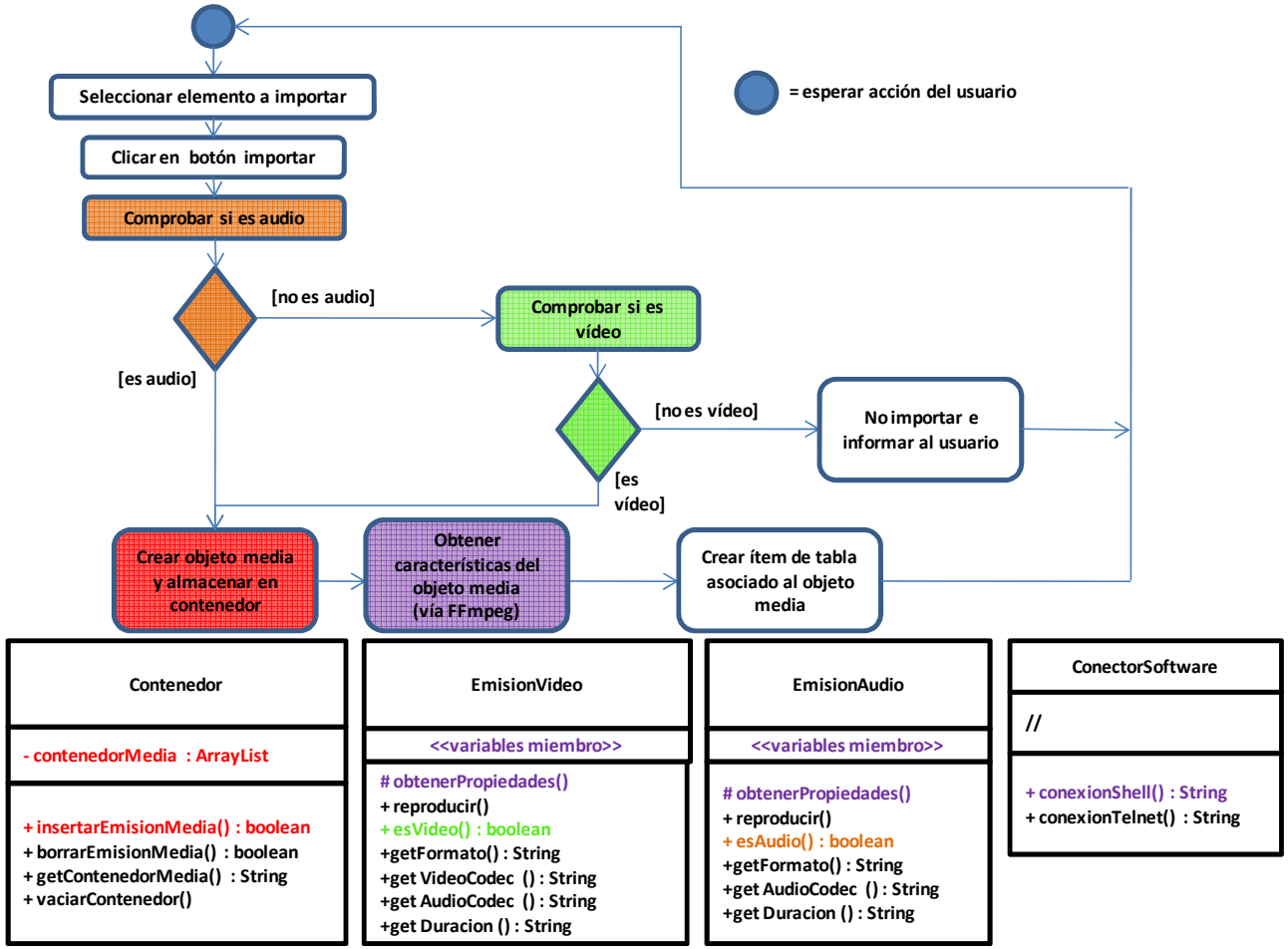
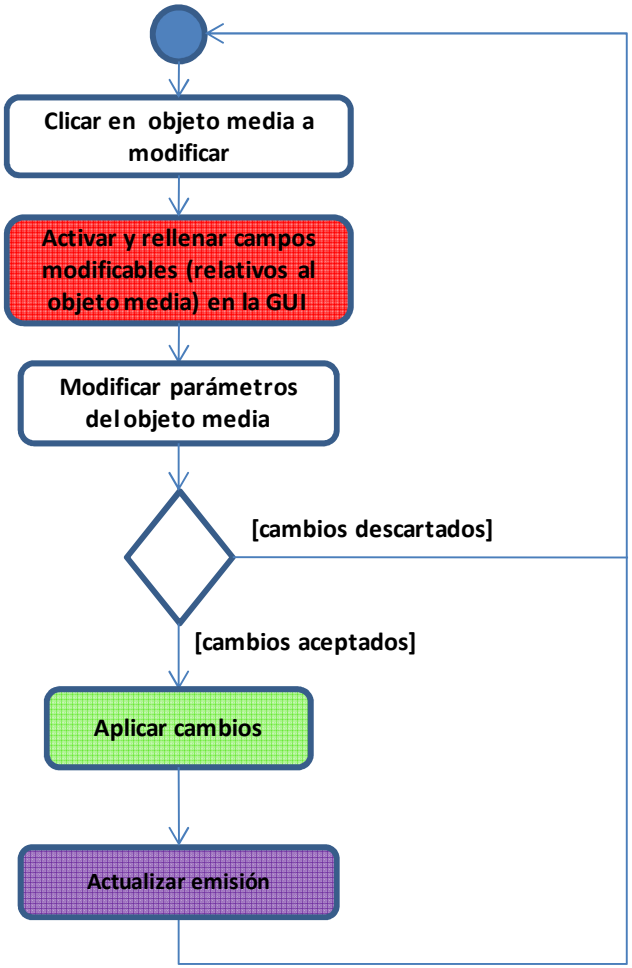
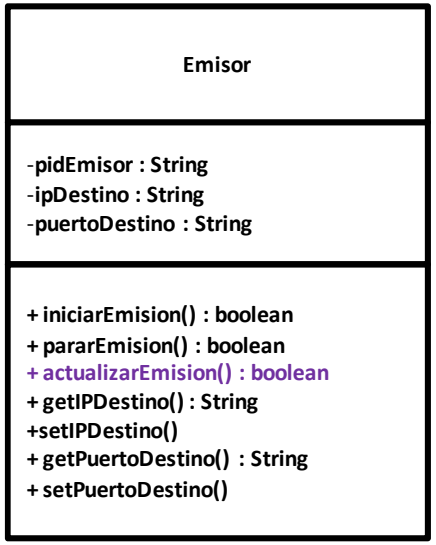


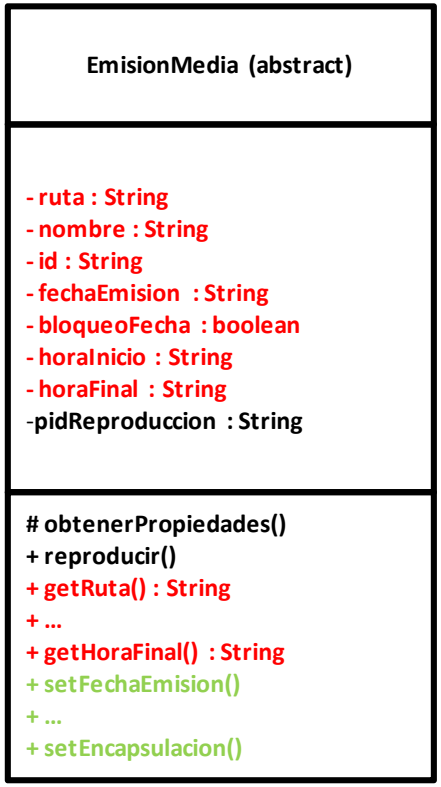
Figura 4.7

4.2.3 Configurar objeto EmisionMedia



= esperar acción del usuario

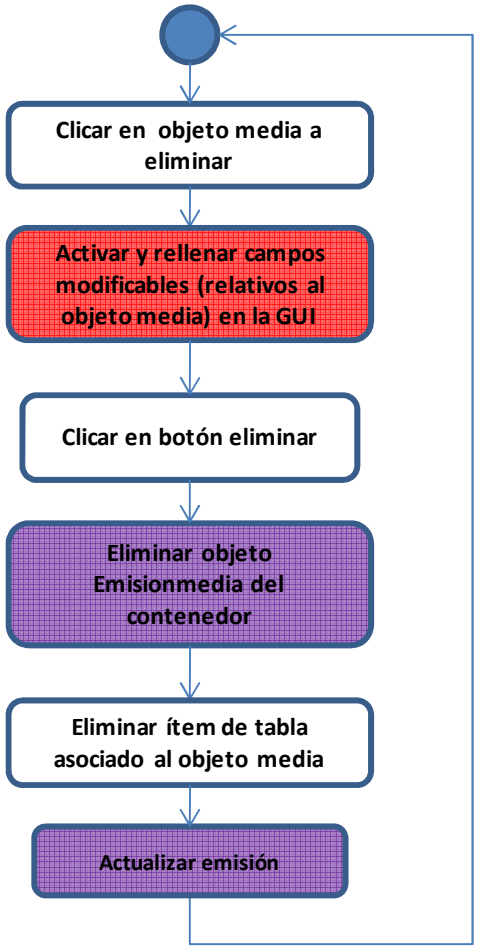




*ruta, nombre y ID no son atributos modificables

Figura 4.8

4.2.4 Borrar objeto emisionMedia



= esperar acción del usuario

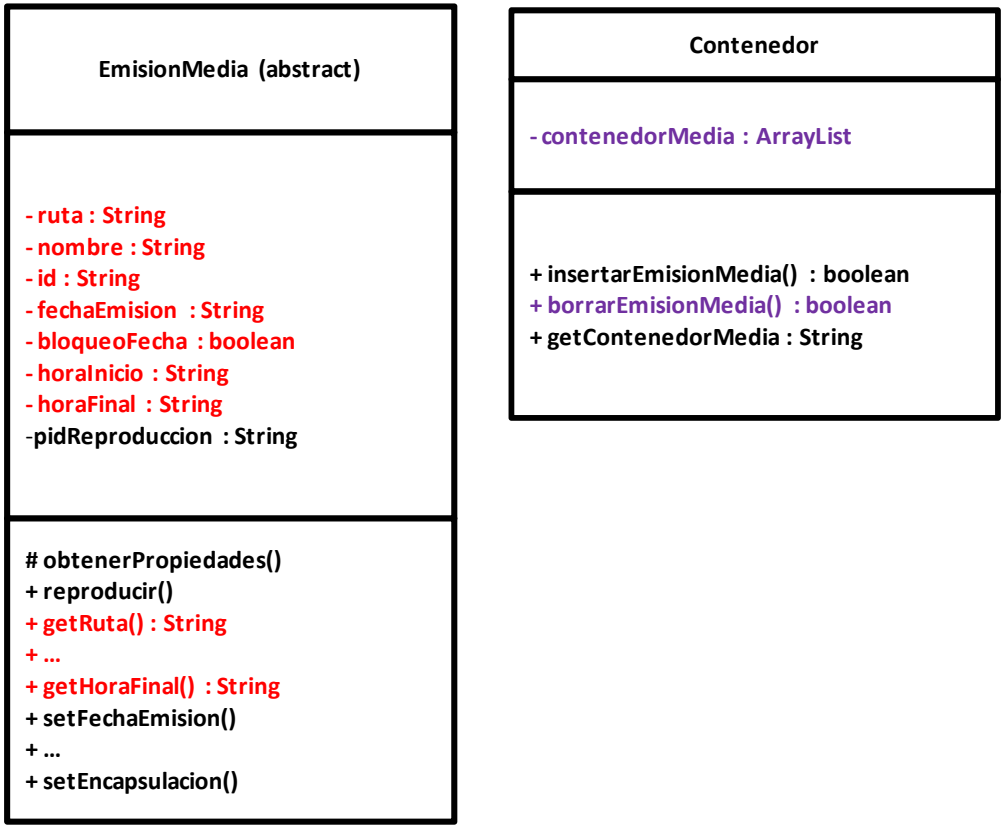


Figura 4.9

4.2.5 (Re)iniciar emisión

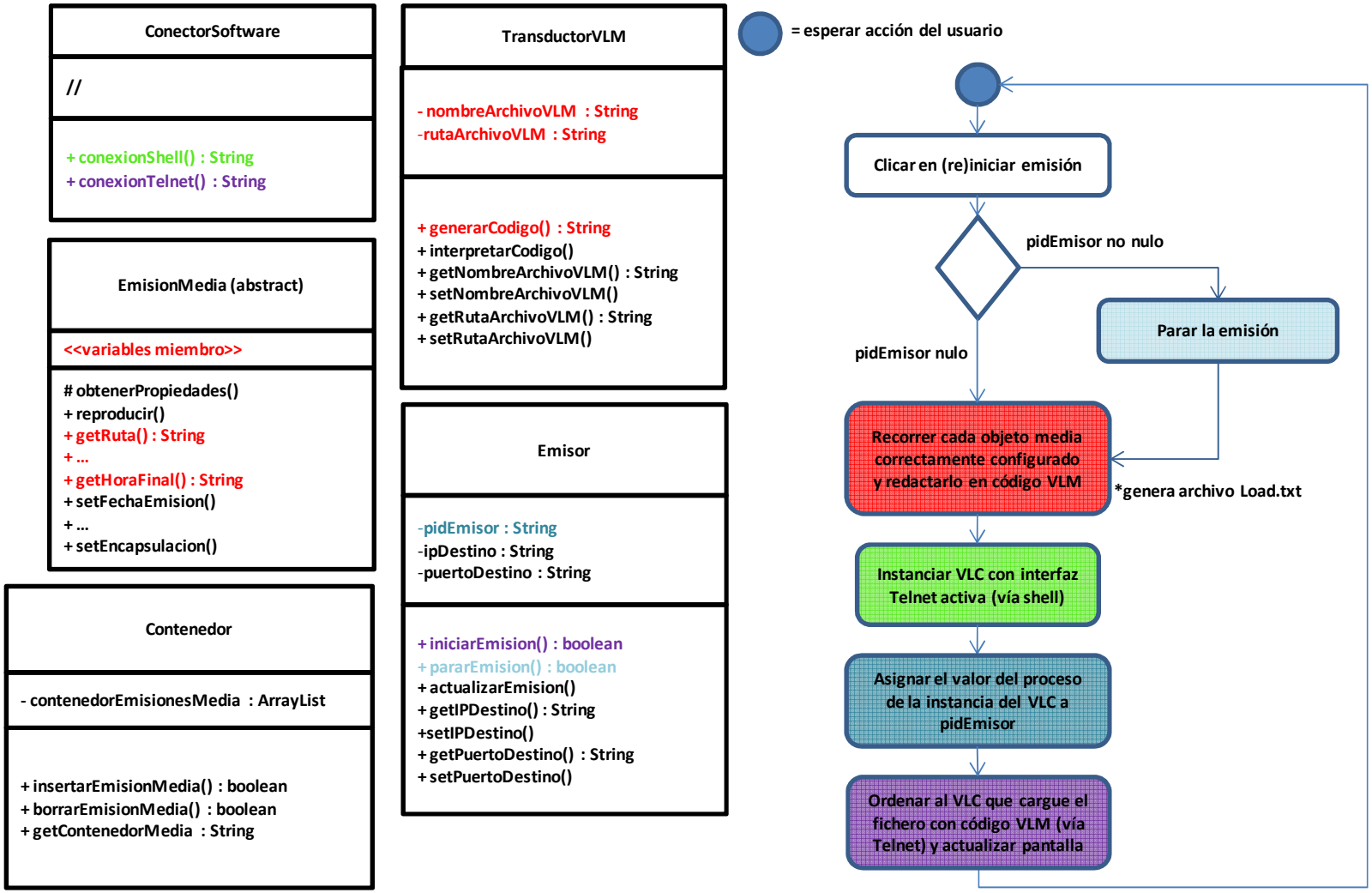


Figura 4.10

4.2.6 Actualizar emisión

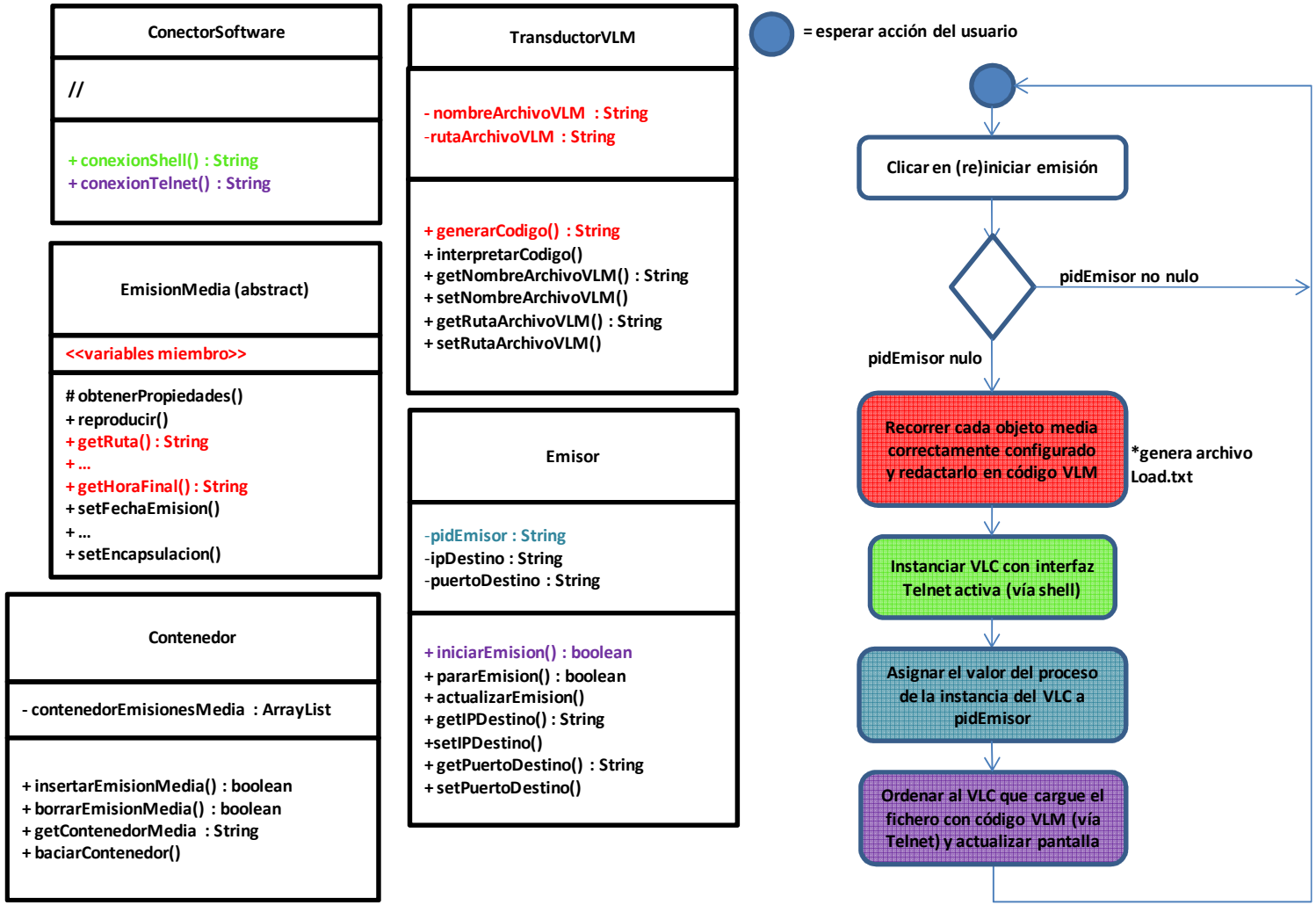


Figura 4.11

4.2.7 Parar emisión

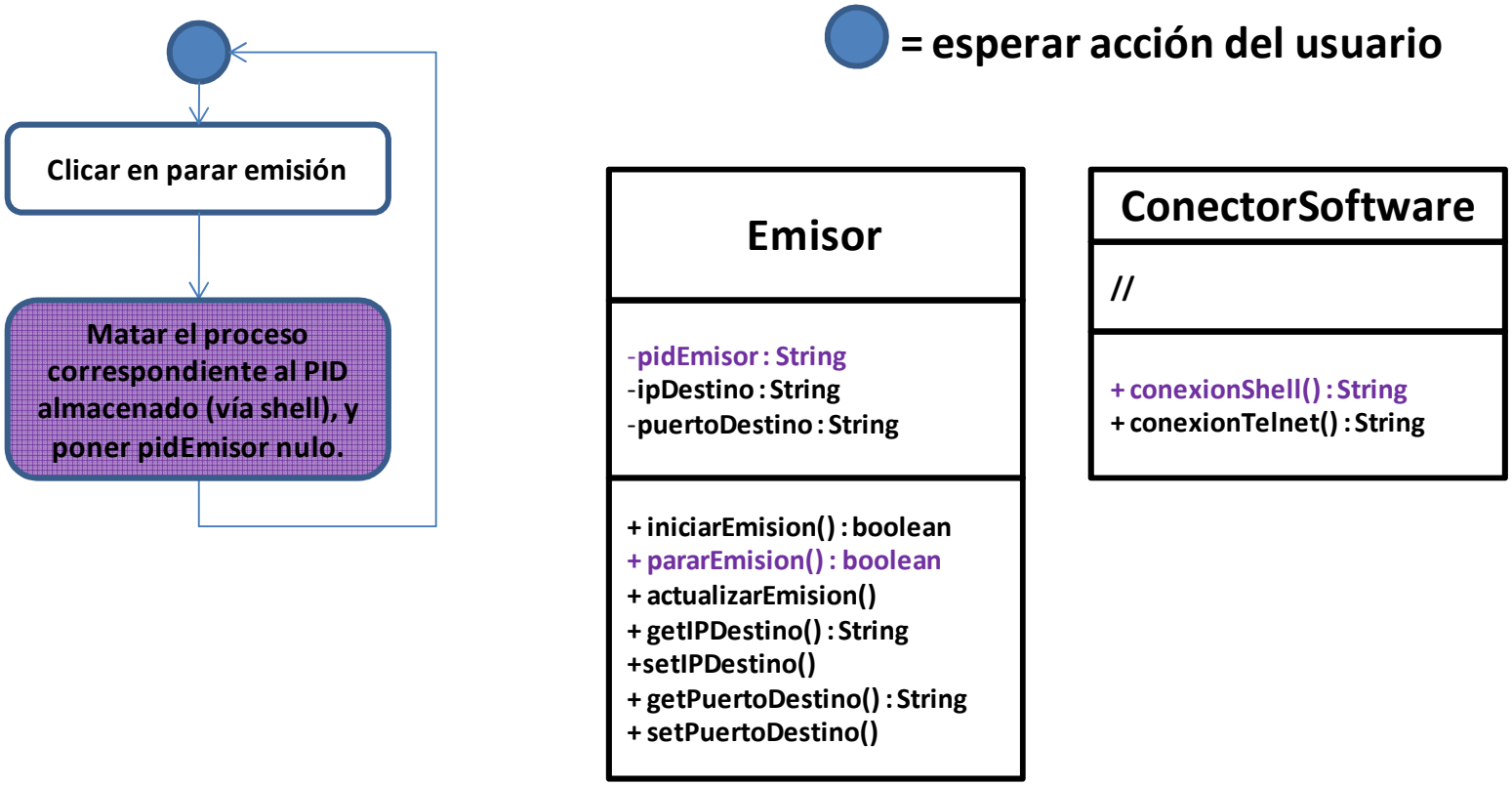
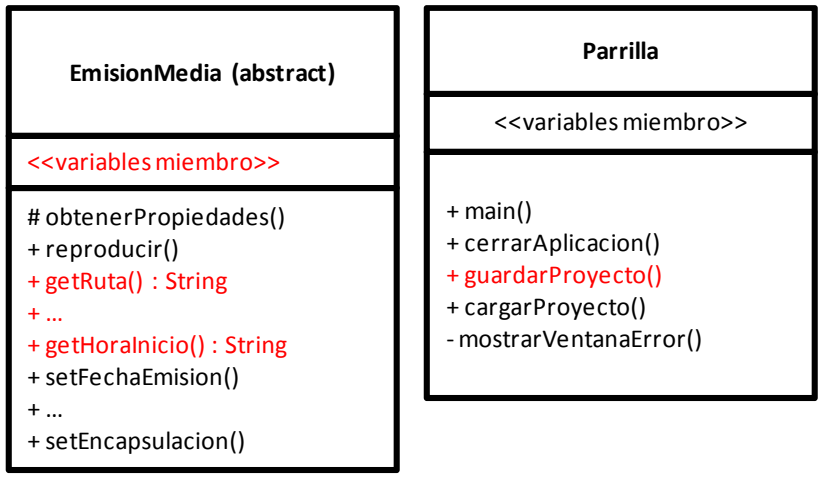


Figura 4.12

4.2.8 Guardar proyecto



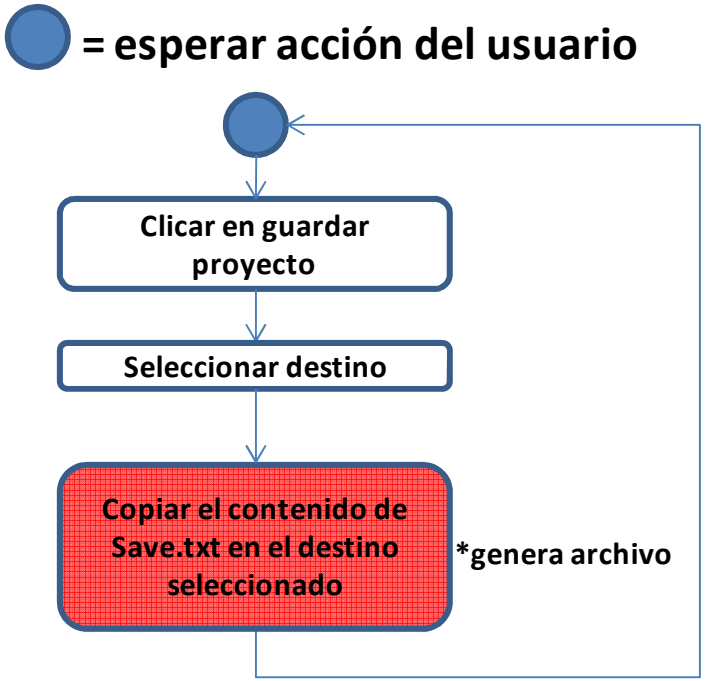


Figura 4.13

4.2.9 Cargar proyecto

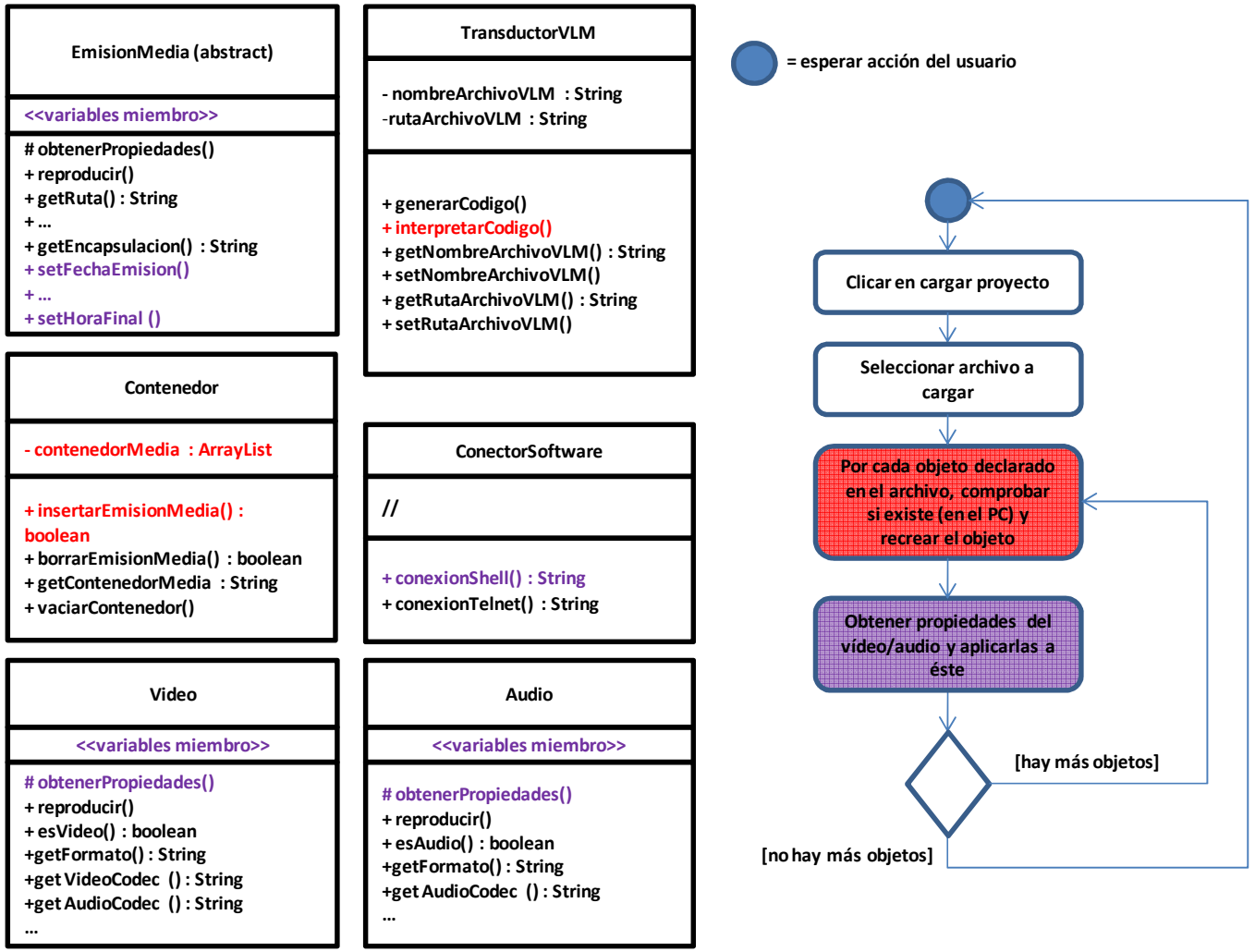
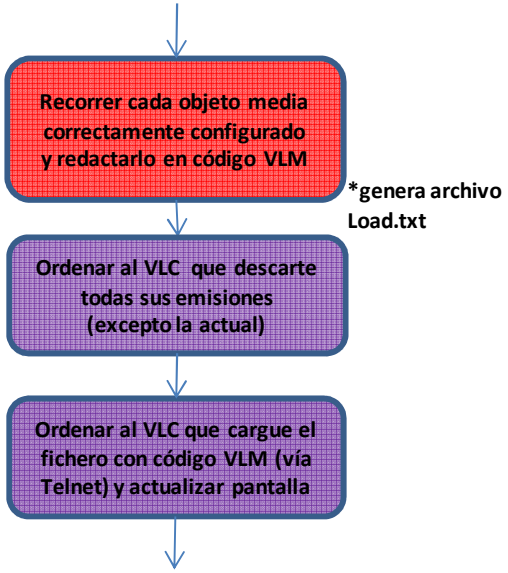
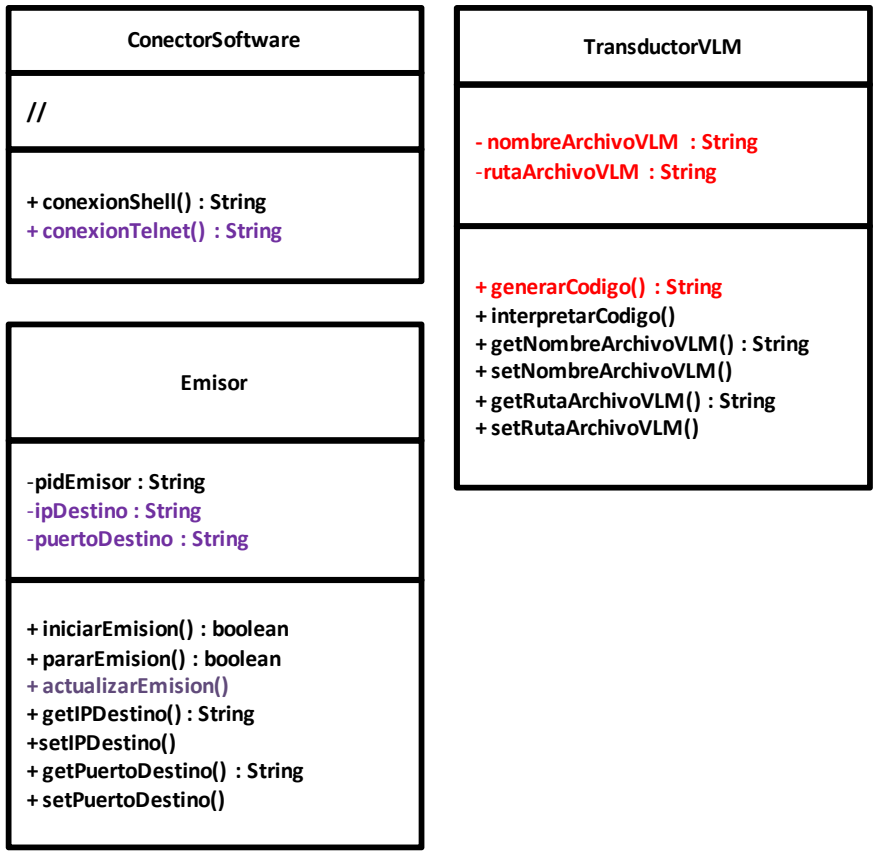


Figura 4.14

4.2.10 Observaciones sobre el método *actualizarEmision()*





Este método aprovecha la instancia del VLC previa, y es a ésta a la que ordena cargar una configuración diferente.

Con objeto de no perder continuidad en la emisión, no se le obliga al VLC a parar la reproducción del vídeo/audio que esté emitiéndose.

Este método también es llamado al cambiar la IP de destino de la emisión o el puerto.

Figura 4.15

5 Código de la GUI de la aplicación

Una de las características esenciales de este proyecto ha sido cumplir con la necesidad de crear un visor de contenidos. Aunque escribir código VLM es costoso y enrevesado, es una posibilidad ante la necesidad de controlar una programación de contenidos. Sin embargo, no ver qué vídeo o audio está ocupando cierto espacio temporal dificulta muchísimo el trabajo, y requiere gran cantidad de consultas al código VLM para ver si se están solapando potenciales emisiones.

Lo que se ha pretendido con la aplicación Java ha sido solventar los dos problemas presentados en el párrafo anterior. Por un lado, se ha buscado crear un visor de contenidos rápido, sencillo e intuitivo. Por otro lado, proveer al usuario de las herramientas suficientes para gestionar correctamente la importación de vídeos/audios y la configuración general de la emisión, buscando siempre la sencillez y el modelo intuitivo.

Las prestaciones que se han incorporado en la interfaz, y que se ven en apartados de este capítulo, han sido las siguientes:

- Explorador de archivos.
- Tablas para los vídeos importados.
- Modificadores de la emisión de cada vídeo.
- Modificadores de la emisión general.
- Visor de contenidos programados.
- Listado de contenidos programados.
- Modificador de la hora de limpieza.

A continuación se muestra una vista general de la aplicación, en la que se detalla cada parte que compone la interfaz general y su función principal. Cada parte viene redondeada por un color diferente:

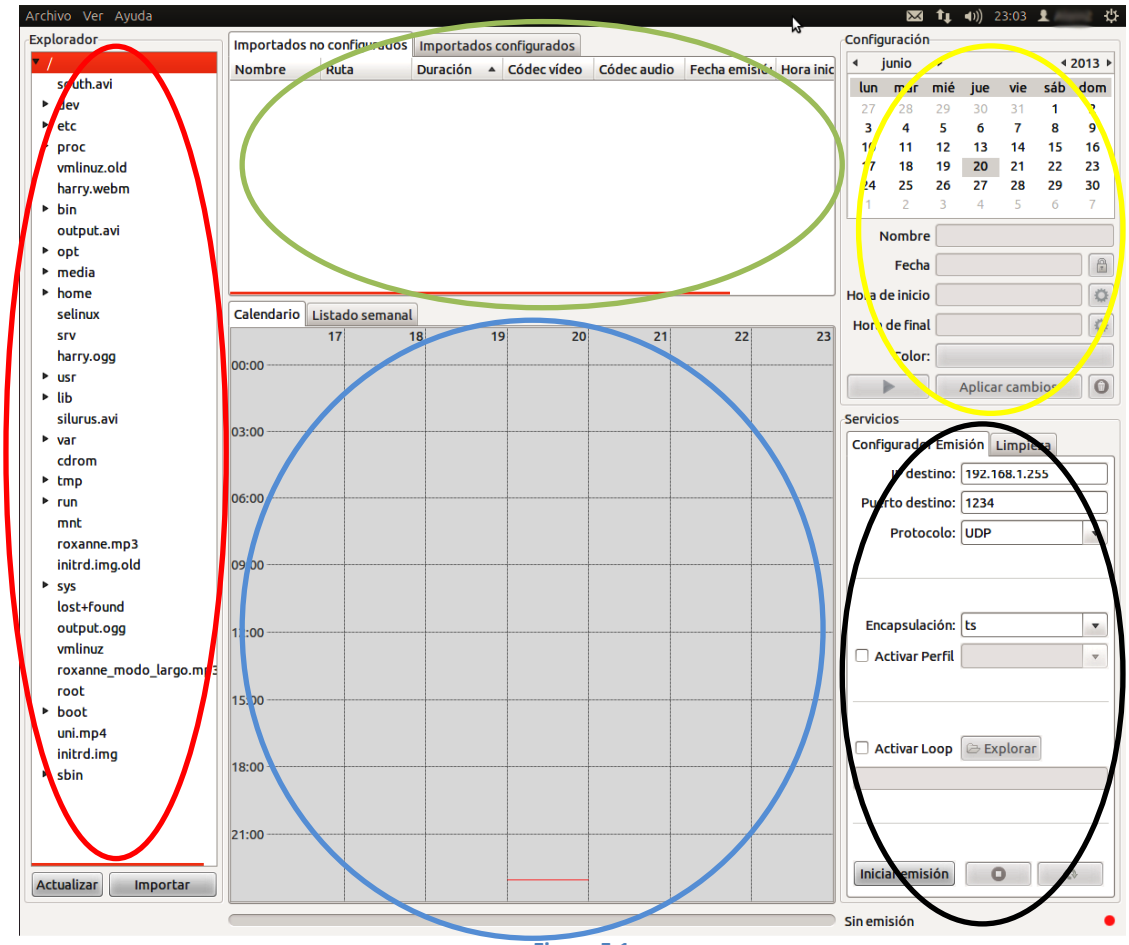


Figura 5.1

Por colores:

- **Rojo**: es el explorador de ficheros; sirve para buscar un fichero de vídeo e importarlo al proyecto.
- **Verde**: es la tabla donde se visualizará la información de cada fichero importado. Cada fichero se corresponderá con una entrada en la tabla.
- **Azul**: es la sección gráfica donde se podrá contemplar la programación correspondiente al día seleccionado en el calendario de la zona superior-derecha, así como de los 3 días previos y los 3 posteriores.
- **Amarillo**: es el campo de opciones donde se configura la fecha y hora de emisión del vídeo seleccionado en la tabla, así como la hora de final.
- **Negro**: es el campo de opciones donde se configuran los parámetros de emisión, tales como protocolo a utilizar, encapsulación del stream, etc.

Figura 2

5.1 Plantilla

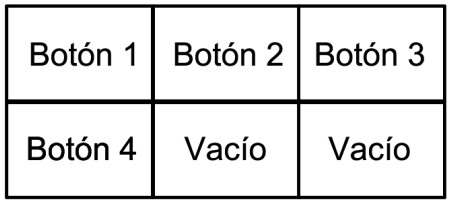
Al crear una interfaz gráfica debe pasarse por etapas de diseño hasta alcanzar el producto visual deseado, sin embargo, la librería gráfica puede limitar la imaginación del diseñador si no posee herramientas suficientemente avanzadas para establecer y distribuir todos los widgets (o elementos gráficos) en pantalla.

En este caso no ha habido exigencias gráficas excesivas, y el planteamiento inicial se ha podido llevar a cabo correctamente, mediante una distribución matricial. Aunque SWT provee varios tipos de "disposiciones" (algo así como plantillas de distribución de widgtes gráficos) diferentes, se ha optado por la "GridLayout": disposición en cuadrícula.

5.1.1 GridLayout

Es una de las diversas opciones de "Layout" que provee SWT. Consiste en tratar la ventana gráfica de la aplicación como una matriz a la que uno asigna un número de columnas, mientras que las filas se añaden sólo si son necesarias. Es decir, se comienza dando un número de columnas fijo (por ejemplo 3) y si se el número de widgets sobrepasa tal número, se crean nuevas filas hasta satisfacer la necesidad. La figura 5.1 muestra el comportamiento de GridLayout.

Ventana con GridLayout de 3 columnas: 2 botones añadidos.



Ventana con GridLayout de 3 columnas: 4 botones añadidos.

Figura 5.2 Ejemplo GridLayout (1)

Aunque a priori parece un sistema bastante limitado, el hecho de poder utilizar widgets contenedores (que envuelven a otros widgets, y que además pueden poseer su propio Layout) le da una versatilidad enorme a la herramienta. Si se quisiera añadir 4 botnes en la casilla vacía del primer ejemplo de la figura 5.1, con una disposición de 2 filas y 2 columnas, habría que llevar a cabo los siguientes pasos:

- Aparte de los dos botones (columnas 1 y 2), declarar un widget contenedor (se posicionaría en la columna 3 automáticamente).
- Al widget contenedor asignarle un GridLayout de 2 columnas (que sería independiente del de la ventana principal).
- Declarar 4 botones internos al wiget contendor; se posicionarían los botones con una disposición de 2x2 automáticamente, tal como muestra la figura 5.2.

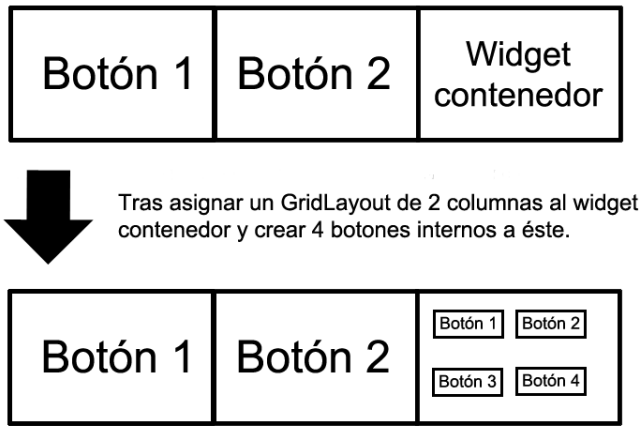


Figura 5.3 Ejemplo GridLayout (2)

GridLayout, técnicamente, es una clase de la librería SWT, y se puede asignar a todos aquellos widgets gráficos (que también son clases de SWT) que tengan la capacidad de albergar otros widgets; por ejemplo, la clase Shell o la clase Composite. Shell, como ya se ha mencionado anteriormente en la memoria, representa una ventana en la interfaz gráfica, mientras que Composite no es más que un contenedor para albergar widgets, especialmente útil en el contexto descrito en la figura 5.2.

```
shell = new Shell();
shell.setSize(1024, 880);
shell.setText("Parrilla TV");
shell.setLayout(new GridLayout(3, false));
```

La última línea muestra la forma de asignar un Layout de tipo GridLayout a la ventana creada.

5.2 Disposición de la interfaz gráfica

En la aplicación se ha utilizado para la ventana principal un GridLayout de 3 columnas. Tras la adición de los widgets que se han considerado necesarios, la ventana principal ha quedado estructurada como una matriz 4x3 (4 filas y 3 columnas).

La clase Gridlayout va de la mano con la clase GridData, la cual permite coger ciertos comportamientos especiales a los objetos internos en una ventana o contenedor que se estrucutre por medio de GridLayout. De este modo, un widget determinado que en teoría sólo debiera ocupar una casilla de la cuadrícula, puede abarcar 2 filas y 2 columnas (por ejemplo) si el usuario así lo desea. En la clase GridData también se determina si un objeto debe estirarse/encogerse si la ventana o contenedor en la que está inmerso cambia de tamaño, entre otros muchos parámetros.

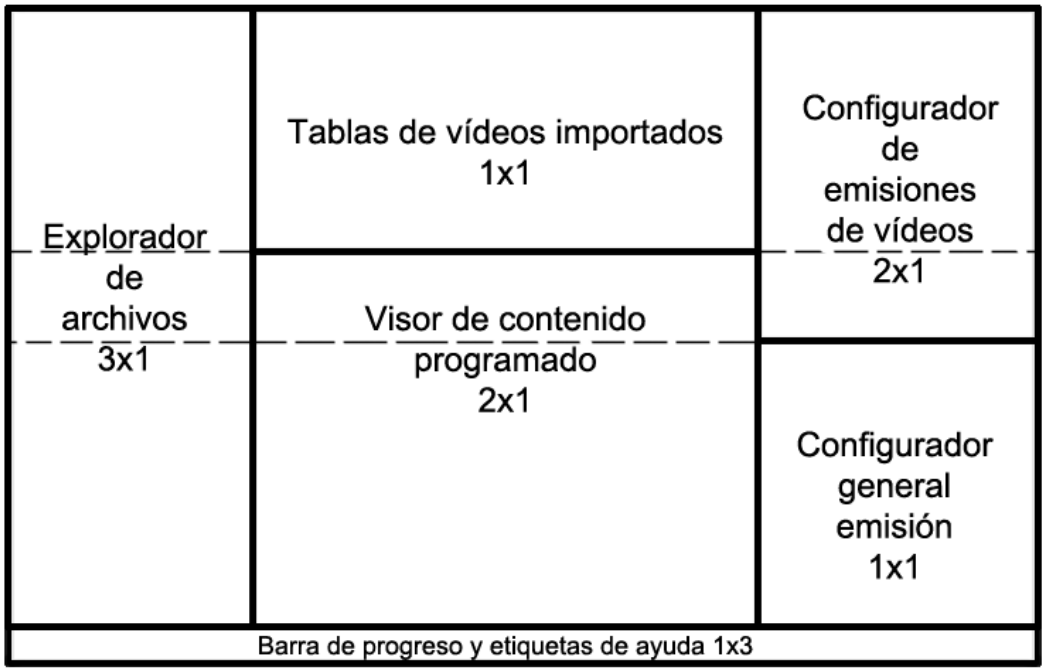


Figura 5.4 Estructura de la GUI

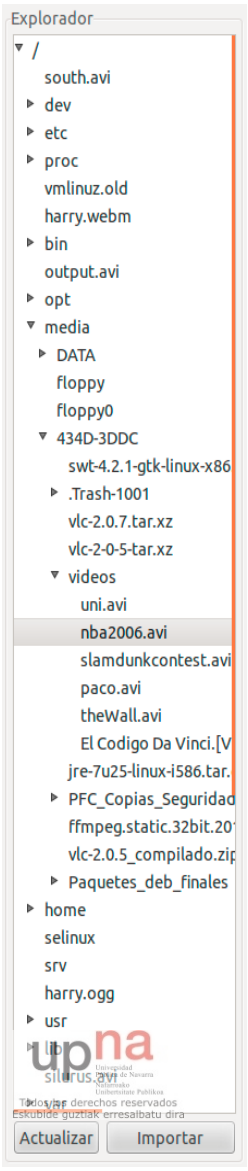


Figura 5.5 Explorador

5.2.1 Grupo explorador de archivos

```
//*****Grupo de Explorador;
grpExplorador = new Group(shell, SWT.NONE);
grpExplorador.setLayout(new GridLayout(2, false));
grpExplorador.setLayoutData(new GridData(SWT.FILL, SWT.FILL, false, true, 1,
3));
grpExplorador.setText("Explorador");
...
tree = new Tree(grpExplorador, SWT.VIRTUAL | SWT.BORDER);
...
bActualizar = new Button(grpExplorador, SWT.NONE);
...
bImportar = new Button(grpExplorador, SWT.NONE);
...
```

Group es un widget contenedor, que envuelve a otros widgets. Gráficamente, los rodea con una línea y permite asignarles una etiqueta.

```
grpExplorador.setLayout(new GridLayout(2, false));
```

La línea de arriba asigna un gridLayout de 2 columnas al grupo.

```
grpExplorador.setLayoutData(new GridData(SWT.FILL, SWT.FILL, false,
true, 1, 3));
```

Esto hace que rellene su casilla horizontal y verticalmente (parámetros FILL), que se estire/encoja el grupo si la ventana padre cambia de tamaño (en este caso sólo verticalmente) y que el grupo ocupe 1 columna y 3 filas.

```
tree = new Tree(grpExplorador, SWT.VIRTUAL | SWT.BORDER);
...
bActualizar = new Button(grpExplorador, SWT.NONE);
...
bImportar = new Button(grpExplorador, SWT.NONE);
```

Se crea un árbol y dos botones internos al grupo. Se hace que el árbol de directorios ocupe 2 columnas (mediante GridData) para lograr tal distribución de widgets.Mediante GridData, también se le asigna al árbol de directorios una anchura “a poder ser” de 200px.

El grupo de explorador tiene su propia clase ManejadorExplorador, creada expresamente para atender a los eventos generados por el árbol de directorios y los botones del grupo.

5.2.2 Tablas de vídeos importados

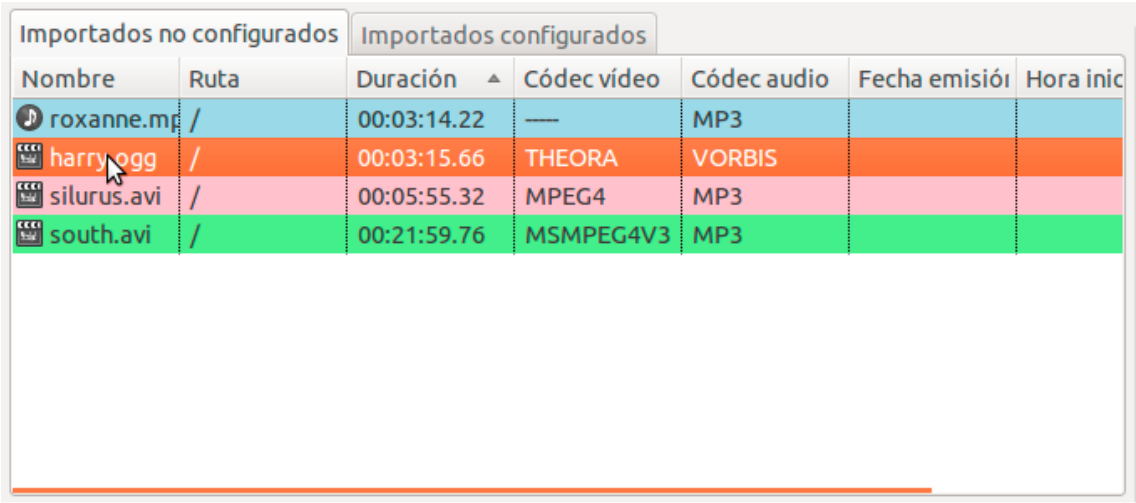


Figura 5.6 Tablas de vídeos/audios importados

```
tabFolderImportados = new TabFolder(shell, SWT.NONE);
tabFolderImportados.setLayout(new GridLayout(1, false));
GridData gd_tabFolderImportados = new GridData(SWT.FILL, SWT.FILL, true, false, 1, 1);
gd_tabFolderImportados.heightHint = 260;
tabFolderImportados.setLayoutData(gd_tabFolderImportados);
tabFolderImportados.addListener(SWT.Selection, manejadorTabla);

tbtmImportadosNoConfigurados = new TabItem(tabFolderImportados, SWT.NONE);
tbtmImportadosNoConfigurados.setText("Importados no configurados");

tbtmImportadosConfigurados = new TabItem(tabFolderImportados, SWT.NONE);
tbtmImportadosConfigurados.setText("Importados configurados");
```

En la figura 5.5 puede apreciarse como hay dos pestañas, la de “importados no configurados” y la de “importados configurados”. Cada una de ellas es un objeto TabItem, que se crea dentro de un objeto tabFolder (contenedor de TabItems: pestañas gráficas) que a su vez ya se había creado dentro de la ventana principal (un objeto Shell).

```
gd_tabFolderImportados.heightHint = 260;
```

“heightHint” es una propiedad de la clase Griddata, y permite asignar la altura deseada para un determinado widget. Esta altura fuerza a la fila a tener tal altura, siempre que sea posible. Si otro widget de la misma fila declarara un heightHint de 500px, la fila, siempre que fuera posible, adoptaría una altura de 500px (satisface al que solicita mayor tamaño).

```
tabFolderImportados.addListener(SWT.Selection, manejadorTabla);
```

La línea de arriba asigna un determinado manejador como objeto encargado de procesar los eventos generados el objeto TabFolder. En este caso, un objeto de la clase ManejadorTabla, exclusivamente creado para hacerse cargo de los eventos de esta parte de la interfaz gráfica.

5.2.3 Grupo de configuración de horarios

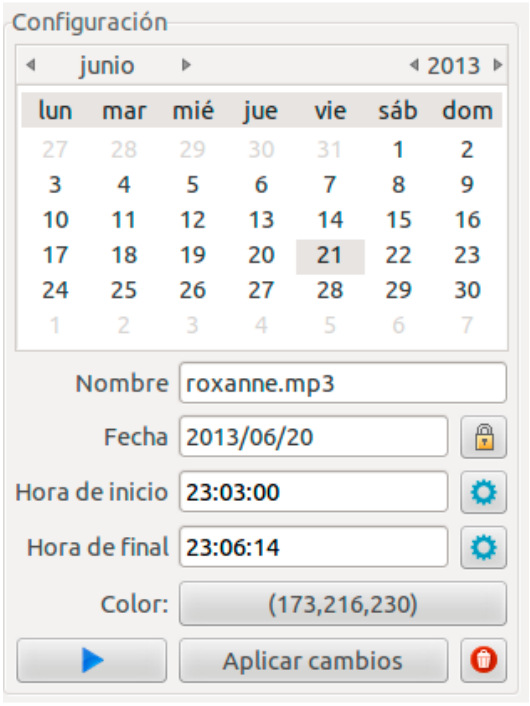


Figura 5.7 calendario y opciones de configuración

```
dateTime = new DateTime(grpConfiguracion, SWT.BORDER | SWT.CALENDAR);
dateTime.setLayoutData(new GridData(SWT.FILL, SWT.FILL, false, false, 3, 1));
...
lblNombre = new Label(grpConfiguracion, SWT.NONE);
…
txtNombre = new Text(grpConfiguracion, SWT.BORDER);
…
bEliminar = new Button(grpConfiguracion, SWT.NONE);
try{
	Image image = new Image(display, new
	ImageData("./media/images/delete_icon.png"));
	bEliminar.setImage(image);
} catch (SWTException swte){
	System.out.println("Hubo un error SWT!!");
}
```

La clase DateTime pertenece a SWT, y permite una visualización tal como la que aparece en la figura 5.6. Es un calendario seleccionable y que permite navegar en el visor de contenidos entre un día y otro, aparte de servir para asignar fechas a la emisión de cierto vídeo/audio.

Label es la clase que permite generar etiquetas gráficas como las que se ven en la figura de arriba (por ejemplo, *Nombre*, *Fecha*, etc.).

Text crea un campo de texto y Button un botón.

Si se quiere añadir una imagen a un botón hay que crear primero un objeto Image, y pasarle la ruta del icono. Posteriormente, asignarle el objeto Image al objeto botón.

El objeto que se encarga de procesar los eventos generados por esta parte de la interfaz es una instancia de la clase ManejadorConfiguración.

5.2.4 Grupo de servicios

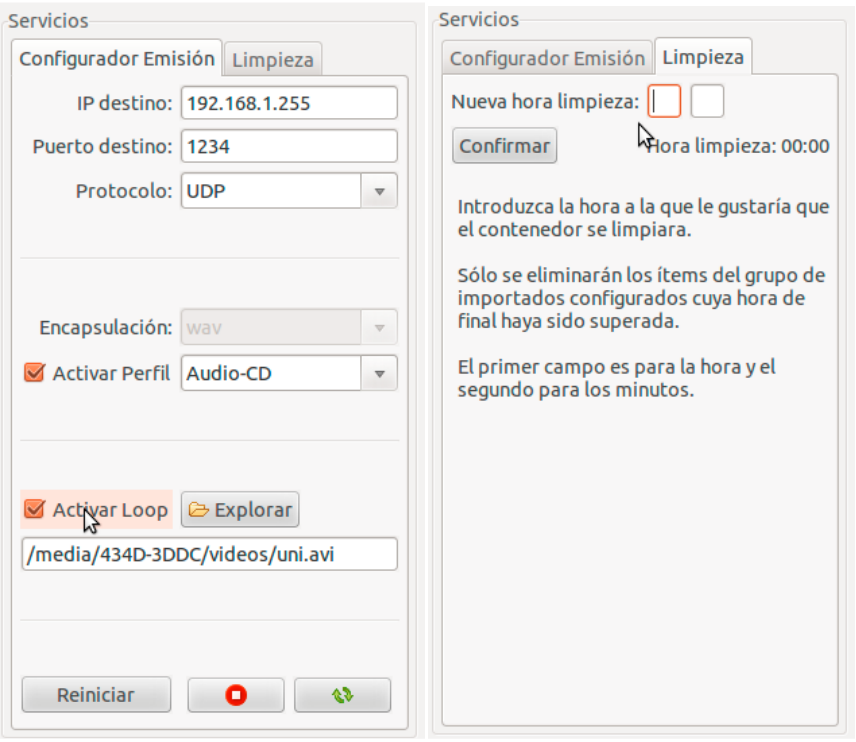


Figura 5.8 Configurador de la emisión y ventana del sistema de limpieza

La única novedad en este grupo es la aparición de dos Combo-boxes, dos botones de tipo check-Button y separadores:

```
...
separador1 = new Label(cmpConfiguradorEmision, SWT.SEPARATOR | SWT.HORIZONTAL);
...
cmbEncapsulacion = new Combo(cmpConfiguradorEmision, SWT.NONE);
cmbEncapsulacion.setItems(new String[] {"ts", "webm", "ogg", "ffmpeg{mux=flv}",
"ps","mpjpeg","wav","flv","mpeg1","mkv","raw","avi","asf"});
...
bCheckPerfil = new Button(cmpConfiguradorEmision, SWT.CHECK);
bCheckPerfil.setText("Activar Perfil");
...
```

Los Combos son campos de texto con un desplegable de opciones que hay que concretarle después de su creación. En el código de arriba se puede ver cómo se le ha añadido un conjunto de strings predeterminado, que son las opciones que luego despliega cuando el usuario lo solicita.

El botón de tipo *check* no tiene mayor misterio que el paso de una constante SWT.CHECK al constructor del objeto Button.

El separador es un Label a cuyo constructor se pasa la constante SWT.SEPARATOR, y la orientación deseada (en este caso SWT.HORIZONTAL).

Los objetos que se hacen cargo de los eventos generados por los widgets de la pestaña de configuración y la pestaña de limpieza son, respectivamente, instancias de las clases ManejadorEmisor y ManejadorLimpieza.

5.2.5 Visor de contenidos programados

El visor ha sido una de las partes centrales de la interfaz, adoptando un protagonismo especial sobre los demás elementos.

Consta de un objeto TabFolder (contenedor de pestañas gráficas). Una de las pestañas hace verse un Composite (contenedor de widgets) vacío, el cual sirve de lienzo para dibujar (mediante las herramientas de dibujo de SWT) los contenidos de vídeo/audio programados. La otra pestaña da acceso a un listado de programación; básicamente muestra la misma información que el Composite, pero en vez de a modo de escaleta gráfica, a modo de listado.

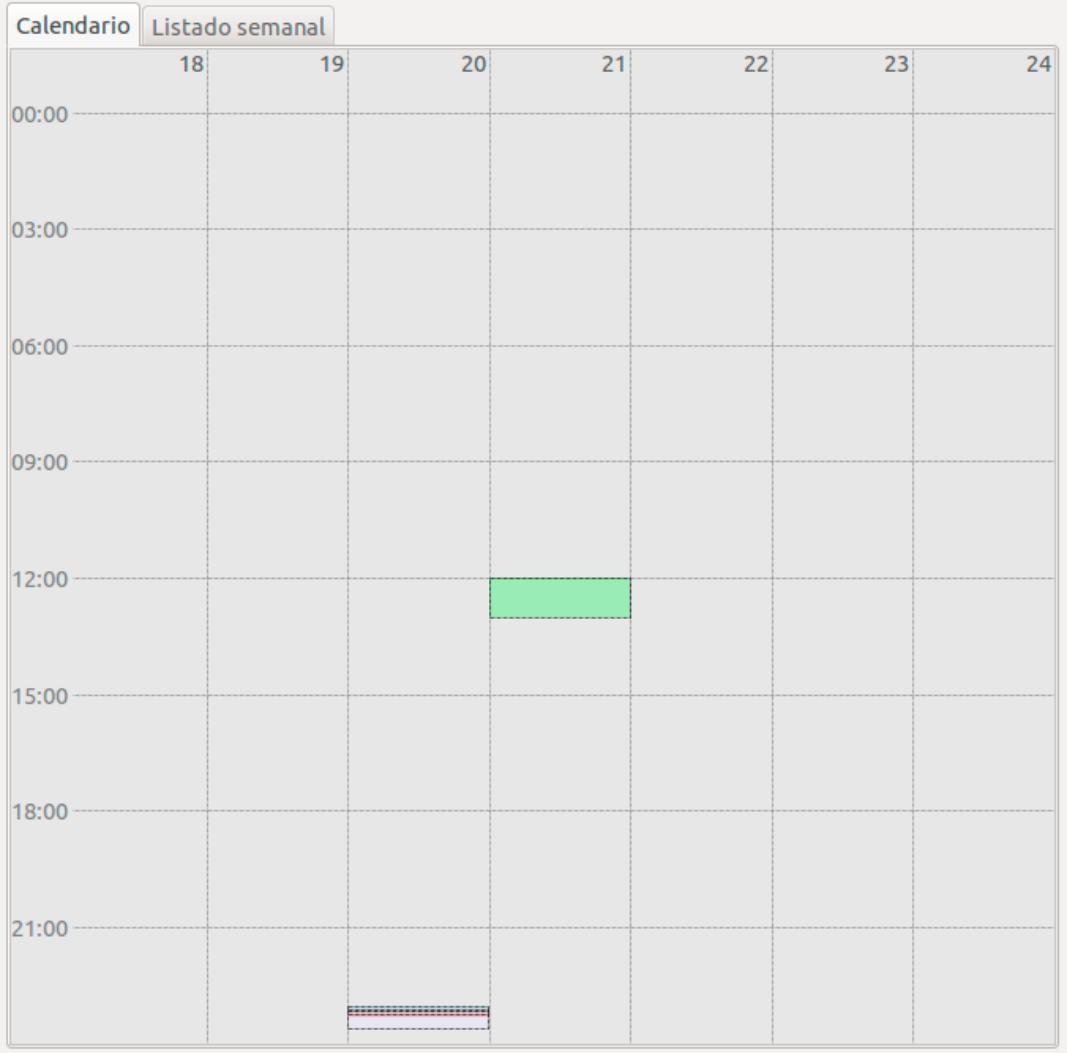


Figura 5.9 Visor de contenidos programados

Los cuadros de colores corresponden a vídeos o audios ya programados y que ocupan esa franja horaria. La división horaria es en bloques de 3 horas. El día central de la visualización (día 21 en la figura 5.8) es el seleccionado en el calendario de la figura 5.6, viéndose también los 3 días previos y los 3 posteriores.

El visor se redibuja cada vez que se realiza algún cambio en la emisión (por ejemplo, cambio de hora de un vídeo) y cada vez que cambia de tamaño la ventana. También se encarga de refrescarla el bucle de control realizado por la instancia de la clase ControladorTiempo.
Quien se encarga de los eventos generados por el visor es una instancia de la clase ManejadorDibujo.

5.2.6 Barra de progreso y etiquetas de ayuda

Figura 5.10 Barra de progreso y etiquetas informativas

Las etiquetas no son más que objetos Label que se actualizan en determinadas ocasiones. La barra de progreso es un objeto de la clase ProgressBar, que se utiliza para mostrar al usuario el progreso del inicio de la emisión, que llegar a tardar varios segundos.

```
...
progressBar = new ProgressBar(shell, SWT.NONE);
progressBar.setLayoutData(new GridData(SWT.FILL, SWT.CENTER, false, false, 1, 1));
...
```

5.2.7 Menú superior

Es un conjunto de widgets que no afecta a la distribución matricial 4x3 explicada al comienzo de este apartado. El menú se añade a la ventana y se integra de tal forma que no afecta a la disposición implantada por el objeto GridLayout.

```
...
mntmArchivo = new MenuItem(menu, SWT.CASCADE);
mntmArchivo.setText("&Archivo");
mntmVer = new MenuItem(menu, SWT.CASCADE);
mntmVer.setText("&Ver");
mntmAyuda = new MenuItem(menu, SWT.CASCADE);
mntmAyuda.setText("&Ayuda");

subMenuArchivo = new Menu (shell, SWT.DROP_DOWN);
mntmArchivo.setMenu(subMenuArchivo);

mntmGuardar = new MenuItem (subMenuArchivo, SWT.PUSH);
mntmGuardar.setText("Guardar proyecto");
mntmGuardar.addSelectionListener(manejadorMenu);
mntmGuardar.setAccelerator(SWT.MOD1  + 's');
mntmCargar = new MenuItem (subMenuArchivo, SWT.PUSH);
mntmCargar.setText("Cargar proyecto");
mntmCargar.addSelectionListener(manejadorMenu);
mntmCargar.setAccelerator(SWT.MOD1  + 'a');
...
```

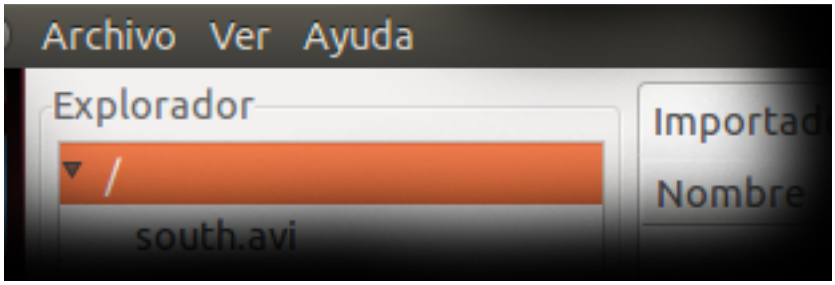


Figura 5.11 Menú superior

```
mntmArchivo = new MenuItem(menu, SWT.CASCADE);
mntmArchivo.setText("&Archivo");
```

Crea un objeto MenuItem que se visualiza directamente en el menú. Añadiendo "&" al nombre, se permite la adición de un menú a este objeto, de modo que pueda tener un menú desplegable en cacada, al hacer clic sobre él. También es necesario el paso de la constante SWT.CASCADE al constructor del objeto MenuItem.

```
subMenuArchivo = new Menu (shell, SWT.DROP_DOWN);
mntmArchivo.setMenu(subMenuArchivo);

mntmGuardar = new MenuItem (subMenuArchivo, SWT.PUSH);
mntmGuardar.setText("Guardar proyecto");
mntmGuardar.addSelectionListener(manejadorMenu);
mntmGuardar.setAccelerator(SWT.MOD1  + 's');
mntmCargar = new MenuItem (subMenuArchivo, SWT.PUSH);
mntmCargar.setText("Cargar proyecto");
mntmCargar.addSelectionListener(manejadorMenu);
mntmCargar.setAccelerator(SWT.MOD1  + 'a');
```

Se crea un nuevo menú (*subMenuArchivo* en este caso) cuyo constructor recibe la constante SWT.DRP_DOWN, que le asigna el comportamiento en cascada. Se le asigna al ítem antes creado (el relativo a *Archivo*) el menú creado. Finalmente, se crean los ítems colgantes y se añaden al menú creado.

El menú posee su propia clase ManejadorMenu para procesar los eventos generados por los ítems de menú.

6 Código interno de la aplicación

Este capítulo se ha dedicado al análisis de las partes más relevantes de código, procurando que el lector pueda hacerse la idea de cómo se ha desarrollado el software.

No se ha hecho hincapié en las partes de código relativas a la GUI, ya que se han comentado en el capítulo previo.

6.1 Constructor de parrilla y main()

```
public class Parrilla {

	//Declaramos las 6 herramientas indispensables del sistema (un
	objeto de cada tipo);
	protected static GUI gui;
	protected static Contenedor contenedor;
	protected static TransductorVLM transVLM;
	protected static ControladorTiempo contrlT;
	protected static ConectorSoftware conector;
	protected static Emisor emisor;

	private Parrilla(){
		//Ahora se crean los objetos esenciales de la clase Parrilla:
		contenedor = new Contenedor();
		transVLM = new TransductorVLM();
		conector = new ConectorSoftware();
		emisor = new Emisor();
		gui = new GUI();

		cerrarAplicacion();
	}

	//Método de entrada a todo el sistema:
	public static void main(String[] args) {
		new Parrilla();
	}
...
```

El único cometido del método main() es crear un objeto de la propia clase parrilla.

Los objetos declarados son *static* debido a que por su carácter de herramienta dentro del sistema, deben poder ser fácilmente accesibles desde otras clases.

El objeto relativo a la clase ControladorTiempo se crea dentro de la clase GUI, después de que se hayan creado todos los widgets de la interfaz gráfica; de lo contrario se da lugar a errores, ya que el objeto de tipo ControladorTiempo intenta acceder a widgets aún no creados. Esto ocurre porque el objeto de clase ControladorTiempo inicia un hilo paralelo al de la interfaz gráfica, y rápidamente le solicita a éste información de algunos widgets que debe monitorizar; al no estar éstos creados todavía, se generan errores en el hilo SWT (el de la GUI).

Puede verse lo mencionado en el siguiente fragmento de código, en el que se crea el objeto ControladorTiempo después de haberse creado los contenidos de la GUI (los vistos en el capítulo anterior):

```
public void iniciarInterfaz() {
	display = Display.getDefault();

	//createContents() crea el contenidos de la GUI (los widgets)
	createContents();
	shell.open();
	shell.layout();

	//Se crea el objeto contrlT aquí para poder pasarle la
	referencia a este objeto GUI y para que no haya errores debido
	al intento de acceso a widgets no creados.
	Parrilla.contrlT = new ControladorTiempo(this);
	Parrilla.contrlT.start();

	while (!shell.isDisposed()) {
		if (!display.readAndDispatch()) {
			display.sleep();
		}
	}
}
```

6.2 Importación de un vídeo/audio

```
private void importarEmisionMedia(){

                    if(EmisionVideo.esVideo(ruta+nombreVideo)){
                          EmisionVideo eVideo = new EmisionVideo(ruta, nombreVideo);
                          Parrilla.contenedor.insertarEmisionMedia(eVideo);

                          TableItem tableItem = new
                          TableItem(gui.tableNoConfigurados, SWT.NONE);;

                          tableItem.setText(0, eVideo.getNombre());
                          tableItem.setText(1, eVideo.getRuta());
                          tableItem.setText(2, eVideo.getDuracion());
                          tableItem.setText(3, eVideo.getVideoCodec());
                          tableItem.setText(4, eVideo.getAudioCodec());

                          Device device = Display.getCurrent ();
                           tableItem.setImage(new
                          Image(device,"./media/images/video_icon.png"));
                          //con objeto de crear un 'enlace' entre el tableItem y el
                          objeto EmisionMedia al que representa.
                          tableItem.setData(eVideo);
                    }
                    else if(EmisionAudio.esAudio(ruta+nombreVideo)){
                          EmisionAudio eAudio = new EmisionAudio(ruta, nombreVideo);
                          Parrilla.contenedor.insertarEmisionMedia(eAudio);
                          Parrilla.contenedor.getContenedorEmisionesMedia();

                          TableItem tableItem = new
                          TableItem(gui.tableNoConfigurados, SWT.NONE);

                          tableItem.setText(0, eAudio.getNombre());
                          tableItem.setText(1, eAudio.getRuta());
                          tableItem.setText(2, eAudio.getDuracion());
                          tableItem.setText(3, eAudio.getVideoCodec());
                          tableItem.setText(4, eAudio.getAudioCodec());

                          Device device = Display.getCurrent ();
                           tableItem.setImage(new
                          Image(device,"./media/images/music_icon.png"));

                          //con objeto de crear un 'enlace' entre el tableItem y el
                          objeto EmisionMedia al que representa.
                          tableItem.setData(eAudio);
                    }

      }
```

Este método se encuentra en la clase ManejadorExplorador. Lo primero que se hace es comprobar si el fichero dado por *ruta+nombre* (parámetros que se obtienen del árbol de directorios) es un vídeo, y de no serlo, si es audio. En caso de ser alguno de los dos, se crea el objeto EmisionMedia pertinente, se introduce en el contenedor de objetos Emisionmedia, y después se crea un ítem de tabla asociado, asignando a cada columna es valor correspondiente (nombre, ruta, duración, etc.). El ítem se visualiza, tras su creación, tal como se muestra en la figura 5.5.

6.2.1 Los métodos esVideo() y esAudio()

Son métodos estáticos de las clases EmisionVideo y EmisionAudio, respectivamente. Son necesarios a la hora de importar un fichero al proyecto.

```
public static boolean esVideo(String rutaMasNombre){

	String ffout = "";

	//ffout es la salida que devuelve ffmpeg tras la consulta.
	ffout = Parrilla.conector.conexionShell("./ffmpeg -i
	"+rutaMasNombre);

	//Se pasa a mayus. Para evitar problemas.
	ffout = ffout.toUpperCase();

	//Uso de expresiones regulares para hallar la duración del
	vídeo (si la hay); se busca un formato hh:mm:ss
	Pattern pattern;
	Matcher matcher;

	pattern =
	Pattern.compile("DURATION:\\s+\\w{2}:\\w{2}:\\w{2}");
	// Se crea el Matcher a partir del patron, la cadena como
	parámetro.
	matcher = pattern.matcher(ffout);

	// Si se encuentra el patrón:
	if(matcher.find()){

		//La suma del tiempo debe ser > 1 seg.!! Para evitar
		importar ficheros que sean una imagen.
		int segundosDuracion =
		Integer.parseInt(ffout.substring(matcher.end()-2,
		matcher.end()));
		int minutosDuracion =
		Integer.parseInt(ffout.substring(matcher.end()-2,
		matcher.end()));
		int horasDuracion =
		Integer.parseInt(ffout.substring(matcher.end()-2,
		matcher.end()));
		int duracionTotal =
		horasDuracion*3600+minutosDuracion*60+segundosDuracion;

		if(ffout.contains("BITRATE") && duracionTotal>1){

			if(ffout.contains("VIDEO")){
				return true;
			}
		}
	}

	return false;
}
```

Si se encuentran (en la respuesta del ffmpeg acerca del fichero) las etiquetas “bitrate” y “video”, y además la duración es mayor a 1 seg., el fichero es de tipo vídeo; por tanto, se puede importar. El sistema es análogo para los ficheros de audio.

6.2.2 Obtención de propiedades de los vídeos/audios

Se consigue mediante expresiones regulares, tal y como se ha hecho en el apartado anterior. Se vuelve a consultar la información que devuelve el ffmpeg acerca de un fichero y se buscan determinadas etiquetas. A continuación el ejemplo para la obtención de la duración:

```
	Pattern patron = Pattern.compile("DURATION:.+\\d+:\\d+:\\d+.\\d+");
	// Se crea el Matcher a partir del patron, la cadena como parametro
	Matcher encaja = patron.matcher(ffout);

	//es necesario llamar al método find antes de invocar el método
	group!!
	// una nueva llamada a find encuentra el sig. substring que
	corresponde a la expresión regular;
	if(encaja.find()){
		this.setDuracion(encaja.group());
		patron = Pattern.compile("\\d+:\\d+:\\d+.\\d+");
		encaja = patron.matcher(this.getDuracion());
		encaja.find();
		this.setDuracion(encaja.group());
	} else {
		this.setDuracion("No se puede hallar");
	}
```

El método find() busca una correspondencia del patrón en cierto string (y devuelve true si la hay), mientras que group() junta el conjunto de caracteres que se corresponde con el patrón.

6.3 Método conexiónShell()

Aunque ya se ha aplicado el método conexiónShell() de la clase ConectorSoftware en los fragmentos de código previos, no se ha analizado todavía. Para la instanciación de software (generalmente FFmpeg) ha resultado vital en la aplicación. A continuación se detalla el método, cuyo núcleo reside en la clase Process.

```
public String conexionShell(String comando){
	String s = null;
	StringBuilder sb = new StringBuilder();
	try{
		//La siguiente línea permite lanzar comandos a la shell del
		SO:
		Process process = new ProcessBuilder(comando.split("
		")).start();
		//Se canaliza la entrada de 'process' (que es la salida
		estándar del SO si todo va bien) hacia la aplicación:
		BufferedReader results = new BufferedReader(new
		InputStreamReader(process.getInputStream()));
		while((s = results.readLine())!=null){
			sb.append(s);
			sb.append("\n");
		}
(continúa en página siguiente)
```

```
(continuación)
            //Se canaliza la entrada de 'process' (que es la salida de
            error del SO si algo ha ido mal) hacia la aplicación:
            BufferedReader errors = new BufferedReader(new
            InputStreamReader(process.getErrorStream()));
            while((s = errors.readLine())!=null){
                  sb.append(s);
                  sb.append("\n");
            }
            } catch(IOException ioe){
                  sb.append(ioe.toString());
            }
            return sb.toString();
      }
```

El método split() separa una cadena (String) mediante el patrón indicado como argumento de entrada (en este caso un espacio vacío: " "). La clase Process posee el método start(), que permite la ejecución del comando en consola. Un ejemplo de comando podría ser "killall vlc", que mataría todos los procesos activos de VLC corriendo sobre el SO.

6.4 Instanciando VLC

Se consigue a través del método iniciarEmisión(). Éste pertenece a la clase Emisor y es encargado de instanciar VLC con interfaz telnet activa. Con ayuda del método actualizarEmision() carga, de paso, la configuración correspondiente a ese instante en el VLC (mediante la herramienta VLM). En este caso concreto se utiliza la clase Runtime para comunicarse con la terminal del sistema; lo que permite esta clase es no tener que esperar a que el comando enviado a la terminal termine su ejecución (lo que provocaría el estancamiento de la aplicación durante la ejecución del VLC).

```
String cmd = "vlc --intf telnet --telnet-port "+Parrilla.emisor.contPuertos;
Runtime.getRuntime().exec(cmd);
```

Parrilla.emisor.contPuertos es una variable que determina qué puerto debe usarse para la interfaz telnet, después de una comprobación previa de los puertos en uso.

Un ejemplo de comando que finalmente se envía a la terminal es:

vlc --intf telnet --telnet-port 4212

6.5 Conexión con VLC vía telnet

Después de la instanciación de VLC con Puerto telnet activo, debe cargársele la configuración (Load.txt). De esto se encarga, entre otras cosas, el método actualizarEmision() de la clase Emisor.

```
Parrilla.conector.conexionTelnet("load ./vlm/Load.txt");
```

El punto fuerte reside en el método conexionTelnet() de la clase ConectorSoftware:

```
public String conexionTelnet(String comando) throws IOException{

	//Se declaran y asignan los parámetros de conexión; Emisor.contPuertos
	es la variable que determina a qué puert se debe realizar la conexión.
	String host = "localhost";
	int port = Emisor.contPuertos;
	String login = "root";
	String password = "admin";

...

	try {
		try{
			socket = new Socket(host, port);
		}
		catch (Exception e){
			e.printStackTrace();
		}
		BufferedReader r = new BufferedReader(new
		InputStreamReader(socket.getInputStream()));
		BufferedWriter w = new BufferedWriter(new
		OutputStreamWriter(socket.getOutputStream()));

		//Se realiza un establecimiento de conexión mediante envío y
		recepción de datos.
		readLines(r, 0, DEBUG);
		writeLine(w, password, CRLF);
		readLines(r, 1, DEBUG);
		//Se envía el comando telnet (es el argumento de entrada del
		método).
		writeLine(w, comando, CRLF);
		readLines(r,1, DEBUG);
		writeLine(w, LOGOUT, CRLF);
		readLines(r, 1, DEBUG);
...
```

Se puede, por tanto, cargar la configuración generada por medio del comando “load Load.txt” al método conexionTelnet(). Del mismo modo se puede conectar para solicitar la ejecución del comando “save Save.txt”, y hacer que VLC genere el archivo con la configuración que tiene cargada.

6.6 Dibujar contenido

Dibujar el contenido programado requiere ciertas etapas. Por un lado, debe dibujarse la base: dar color de fondo y formas deseadas al lienzo sobre el que se dibujan, posteriormente, los vídeos/audios programados. Después, debe hacerse un recorrido exhaustivo de los objetos EmisionMedia correctamente configurados y convertirlos en un dibujo de tipo rectangular, que tenga una altura determinada en función de la hora de inicio y final del objeto.

Para poder dibujar y pintar sobre el lienzo (que es el widget contenedor de nombre cmpCalendario) hay que hacer uso de un objeto de clase GC de la librería SWT. Éste puede ser instanciado directamente como atributo del evento que provoca la llamada al método de dibujo.

```
public void paintControl(PaintEvent arg0) {

		if(arg0.getSource().equals(gui.cmpCalendario)){

			Rectangle clientArea = gui.cmpCalendario.getClientArea();

			//Se obtienen la anchura y altura de la casilla que el widget
			ocupa en la cuadrícula:
			int width = clientArea.width;
			int height = clientArea.height;

			//Se asigna nivel de transparencia a la herramienta de dibujo
			(entre 0 y 255);
			arg0.gc.setAlpha(160);
			//Se asigna anchura de línea;
			arg0.gc.setLineWidth(1);
			//Se asigna tipo de línea (en este caso punteada);
			arg0.gc.setLineStyle(SWT.LINE_DASH);
			//Se asigna color a la herramienta de dibujo:
			arg0.gc.setForeground(gui.display.getSystemColor(SWT.COLOR_DARK
			_GRAY));

			//Se comienza por dibujar las líneas verticales:
			//arg0.gc.drawLine(int x1, int y1, int x2, int y2);
			arg0.gc.drawLine((width-35)/7+35,0,(width-35)/7+35,height);
			arg0.gc.drawLine(2*(width-35)/7+35,0,2*(width-35)/7+35,height);
			arg0.gc.drawLine(3*(width-35)/7+35,0,3*(width-35)/7+35,height);
			arg0.gc.drawLine(4*(width-35)/7+35,0,4*(width-35)/7+35,height);
			arg0.gc.drawLine(5*(width-35)/7+35,0,5*(width-35)/7+35,height);
			arg0.gc.drawLine(6*(width-35)/7+35,0,6*(width-35)/7+35,height);

...
			//Se dibuja el texto que al lado de las líneas horizontales:
			arg0.gc.setAlpha(255);
			arg0.gc.drawText("00:00 ",0,32);
			arg0.gc.drawText("03:00 ",0,(height-tamFuente-30)/8+32);
			arg0.gc.drawText("06:00 ",0,2*(height-tamFuente-30)/8+32);
			arg0.gc.drawText("09:00 ",0,3*(height-tamFuente-30)/8+32);
			arg0.gc.drawText("12:00 ",0,4*(height-tamFuente-30)/8+32);
			arg0.gc.drawText("15:00 ",0,5*(height-tamFuente-30)/8+32);
			arg0.gc.drawText("18:00 ",0,6*(height-tamFuente-30)/8+32);
			arg0.gc.drawText("21:00 ",0,7*(height-tamFuente-30)/8+32);
...
			//Se crea un objeto lector del fichero y se determina la hora a
			buscar.
			File f = new File( "./vlm/Save.txt" );
			BufferedReader entrada;
			try {
				entrada = new BufferedReader( new FileReader( f ) );

				//Se obtiene fecha del dia seleccionado en el calendario
				String fecha =
				gui.dateTime.getYear()+"/"+(gui.dateTime.getMonth()+1)+"
				/"+gui.dateTime.getDay();
...
```

Después se hace uso de expresiones regulares para encontrar la fecha indicada en el fichero Save.txt, ya que de esta forma pueden obtenerse todos aquellos vídeo/audios programados para ese día. No obstante, también se buscan las emisiones correspondientes a los 3 días previos y a los 3 posteriores. Una vez hallada la emisión de un vídeo/audio, se procede a dibujar el rectángulo correspondiente:

```
//Se asigna el color de fondo de la herramienta de dibujo:
arg0.gc.setBackground(eMedia.getColor());
//Se rellena el rectángulo con el color de fondo:
arg0.gc.fillRectangle(0*(width-35)/7+35+5, 40, (width-35)/7-5,
(int)(factorAltura*eMedia.getHoraFinalCodificada()));
//Se dibuja el borde del rectángulo con el color de foreground:
arg0.gc.drawRectangle(0*(width-35)/7+35+5, 40, (width-35)/7-5,
(int)(factorAltura*eMedia.getHoraFinalCodificada()));
```

Este es un caso particular de dibujo; lo importante es tener en cuenta que el método de dibujo funciona del siguiente modo:

```
arg0.gc.drawRectangle(int x0, int y0, int anchura, int altura);
```

x0: nº pixel horizontal en el que empieza el rectángulo.
Y0: nº pixel vertical en el que empieza el rectángulo.
Anchura: anchura del rectángulo.
Altura: altura del rectángulo.

6.7 Generador de código VLM

Se genera el código mediante el método generarCodigo() de la clase TransductorVLM. En resumen, lo que se hace es sobrescribir el fichero Load.txt cada vez que se llama a este método, escribiendo en él strings predefinidos en función de los parámetros de emisión y horarios configurados. Grosso modo, lo que se hace es recorrer la tabla donde se acumulan los vídeos/audios importados y configurados, redactándose varias líneas por cada uno de ellos:

```
for(int i=0; i<gui.tableConfigurados.getItems().length; i++){
      EmisionMedia eMedia =
      (EmisionMedia)gui.tableConfigurados.getItem(i).getData();
      //Se declaran nuevos broadcast/schedule en el código VLM:
      codigo += "new "+eMedia.getID()+" broadcast enabled"+"\n";
      codigo += "setup "+eMedia.getID()+" input
      "+"\""+eMedia.getRuta()+eMedia.getNombre()+"\""+"\n";
      codigo += "setup "+eMedia.getID()+" output ";
      codigo += "#";
...
      codigo += "new prog"+eMedia.getID()+" schedule date
      "+eMedia.getFechaEmision()+"-"+eMedia.getHoraInicio()+" enabled"+"\n";
      codigo += "setup prog"+eMedia.getID()+" append control loopVideo
      stop"+"\n";
      codigo += "setup prog"+eMedia.getID()+" append control
      "+eMedia.getID()+" play"+"\n";
      codigo += "new prog"+eMedia.getID()+"_2 schedule date
      "+eMedia.getFechaParada()+"-"+eMedia.getHoraFinal()+" enabled"+"\n";
      codigo += "setup prog"+eMedia.getID()+"_2 append control
      "+eMedia.getID()+" stop"+"\n";
...
```

6.8 Consulta de vídeo en emisión

Consiste en conectar con el VLC vía telnet y decirle que muestre los medias que tiene configurados. Posteriormente, se analiza cuál de ellos está es esta "playing".

```
String respuestaVLC;
	try {
		respuestaVLC = Parrilla.conector.conexionTelnet("show media");
			if(respuestaVLC.contains("playing")){
...
```

6.9 Limpieza contenedor

Consiste en recorrer la tabla de importados configurados y eliminar aquellos objetos EmisionMedia cuya hora de final programada sea inferior a la hora actual del sistema.

```
gui.display.syncExec(new Runnable() {
	public void run() {

	TableItem[] items = gui.tableConfigurados.getItems();
	ArrayList<Integer> arrayOfIndexes = new ArrayList<Integer>();

	//recorrer tabla de importados configurados:
	for(int i=0; i<items.length; i++){

		EmisionMedia eMedia = (EmisionMedia)items[i].getData();
...
		//Borrar del contenedor objeto Emisionmedia:
		if(eMedia.gethoraFechaFinalCodificada()<horaFechaSistemaCodificada){
			Parrilla.contenedor.borrarEmisionMedia(eMedia);
			Parrilla.contenedor.getContenedorEmisionesMedia();
...
```

"syncExec(Runnable r)" es la forma de conseguir acceder a los widgets creados en un determinado hilo SWT desde otro hilo diferente. Lo que se consigue con este método es decirle al hilo SWT: "cuando tengas tiempo ejecuta las siguientes acciones". Tales acciones las declaramos dentro del método run() de un objeto Runnable, que le pasamos como argumento.

Hay otro método similar: asyncExec(Runnable r). Este último continúa la ejecución del hilo en el que fue declarado el método, sin esperar a que el hilo SWT termine las acciones indicadas por el programador.

7 Manual de la aplicación

7.1 Introducción

Este manual tiene por objeto explicar de un modo claro y conciso el uso del programa Parrilla TV para una correcta puesta en marcha de un canal de contenidos de vídeo y audio en red.

La aplicación, de nombre Parrilla TV, es un gestor de emisión de contenido de vídeo y audio en red, sirviéndose del software VLC para las funciones de servidor de streaming posteriores. Dicho de un modo más sencillo, la aplicación permite configurar la hora a la que se quiere que cierto vídeo (o audio) se emita, al igual que si se tuviera que organizar la programación de un canal de TV.

A partir de este punto se usará solamente "vídeo" cada vez que se mencione un objeto que se pueda emitir, en vez de usar la expresión "vídeo o audio"; por tanto, se asume que todo lo aplicable a vídeo puede aplicarse a audio.

7.2 Espacio de trabajo

Es la interfaz gráfica que ve el usuario, y es el entorno mediante el cual éste podrá configurar toda la emisión. A continuación se detallan las 5 partes principales del espacio de trabajo.

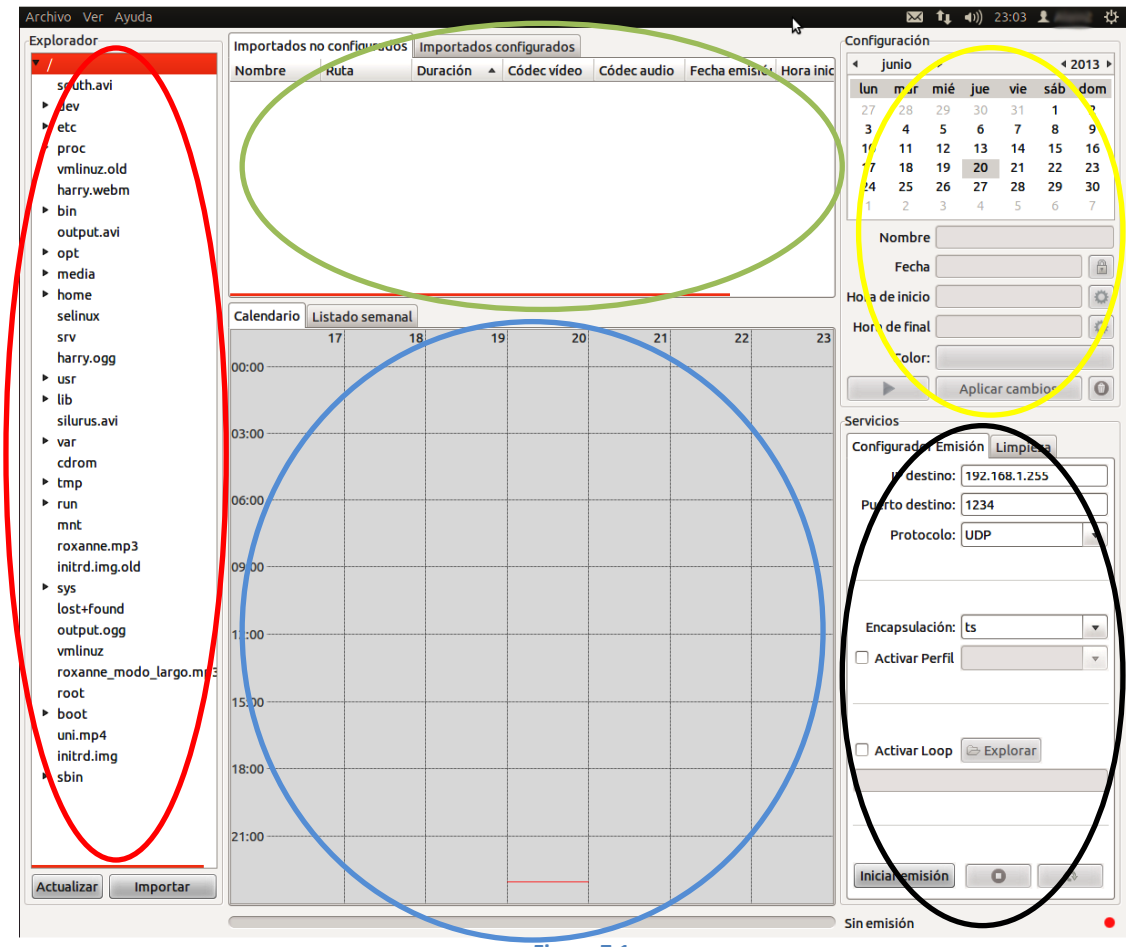


Figura 7.1

Por colores:

- **Rojo**: es el explorador de ficheros; sirve para buscar un fichero de vídeo e importarlo al proyecto.
- **Verde**: es la tabla donde se visualizará la información de cada fichero importado. Cada fichero se corresponderá con una entrada en la tabla.
- **Azul**: es la sección gráfica donde se podrá contemplar la programación correspondiente al día seleccionado en el calendario de la zona superior-derecha, así como de los 3 días previos y los 3 posteriores.
- **Amarillo**: es el campo de opciones donde se configura la fecha y hora de emisión del vídeo seleccionado en la tabla, así como la hora de final.
- **Negro**: es el campo de opciones donde se configuran los parámetros de emisión, tales como protocolo a utilizar, encapsulación del stream, etc.

7.3 Configuración e inicialización de la emisión

7.3.1 Búsqueda de un vídeo a emitir e importación al proyecto

Para este paso se usará el explorador. Éste comienza por la raíz de directorios "/" en sistemas Linux. Una vez hallado el vídeo a incorporar al proyecto, basta con clicar sobre él y pulsar el botón importar. Si se quiere actualizar cierto directorio hay que clicar sobre él y pulsar el botón actualizar. Pueden verse los botones mencionados en la figura 2.

Se puede utilizar la tecla intro para importar; es útil si queremos importar varios vídeos de un mismo directorio, junto con el uso de las flechas arriba y abajo.

Al importar un vídeo, se crea una entrada asociada a éste en la tabla de importados (Véase figura 3). De primeras, aparecerá en la tabla de la pestaña de no-configurados, y pasará a la de configurados una vez se le hayan asignado correctamente ciertos parámetros (se verán a continuación).

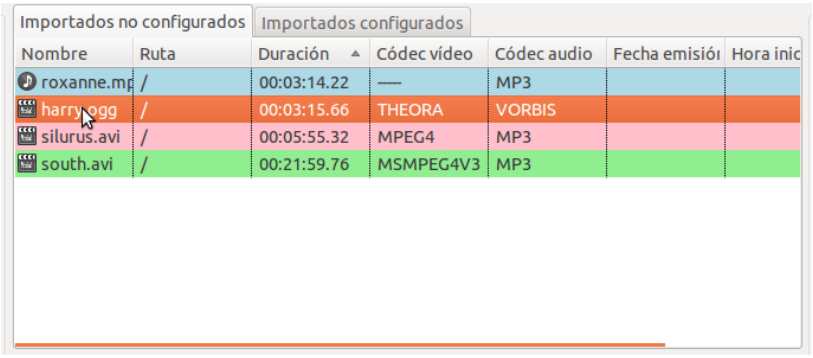


Figura 7.3

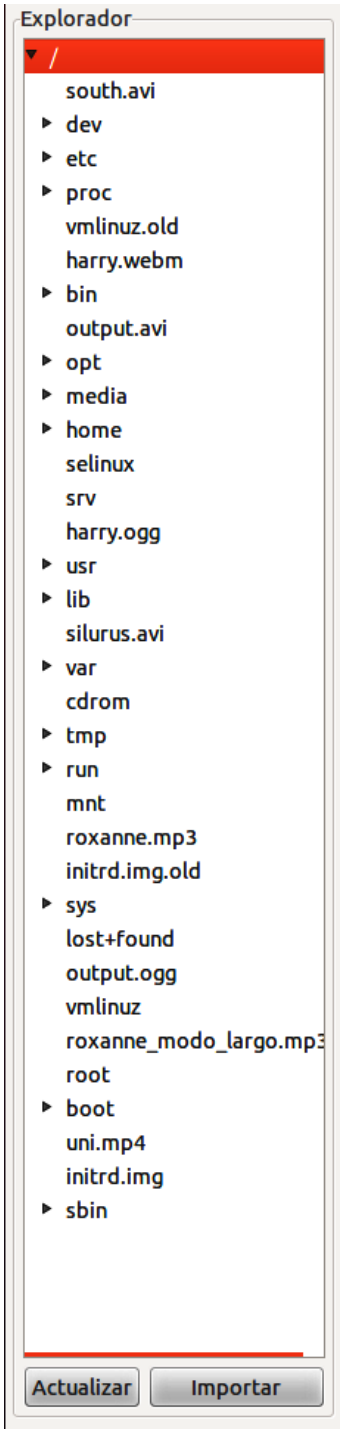


Figura 7.2

7.3.2 Configuración de la emisión del vídeo

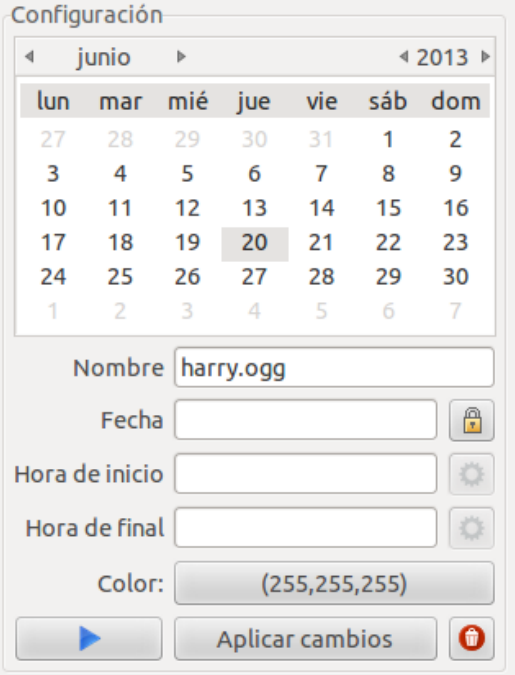


Figura 7.4

El siguiente paso es asignar a cada entrada de la tabla de importados no-configurados los siguientes parámetros:

- *Fecha*: se asigna seleccionando un día en el calendario de la parte superior. El botón con icono de candado sirve para bloquear el campo de fecha, y que al seleccionar un día en el calendario, éste no se modifique. La razón de existir de este botón es que el calendario tiene, también, la función de seleccionar el día que se quiere visualizar en el panel gráfico (figura 1, color azul).
- *Hora de inicio*: en este campo debe escribirse la hora de inicio deseada para el vídeo seleccionado de la tabla (harry.ogg en la figura 4). La hora debe estar escrita en formato hh:mm:ss, de lo contrario aparecerá en rojo y no se aplicará el cambio.
- *Hora de final*: al igual que la hora de inicio, debe escribirse en formato hh:mm:ss, e indica la hora a la que finalizará el streaming del vídeo.
- *Color*: puede cambiarse el color de fondo de la entrada de tabla (en la figura 3 se ven entradas con diferentes colores). Esto afectará en la visualización de emisión asociada a esta entrada en el visor (figura 11).

Los 3 parámetros mencionados son suficientes para la correcta configuración de una entrada de tabla. Una vez se pulse el botón aplicar cambios, éstos tendrán efecto (si ningún campo está en rojo), y la entrada de tabla pasará de la de no-configurados a la de configurados (véase figura 5).

Los botones restantes en el grupo de configuración (figura 4) tienen la siguiente función:

- *Botón autocompletar*: es el botón que aparece al lado de hora de inicio y hora de final; su función es la de autocompletar la hora ajustándose a la duración del vídeo. Funciona en ambos sentidos; es decir, se puede usar para autocompletar la hora de final si la de inicio fue correctamente escrita, o viceversa. Se activará cuando alguno de los campos de hora haya sido escrito y sea correcto.
- *Botón de play*: sirve para hacer un preview del vídeo asociado a la entrada de tabla que se está editando. Se abre una instancia de VLC independiente que reproduce tal vídeo.
- Botón de borrado: tiene el icono de una papelera, y sirve para eliminar la entrada de tabla que se está editando.

Hay dos abreviaturas de teclado para una entrada de tabla seleccionada. La primera es que si se pulsa intro, el vídeo se reproducirá, al igual que sucede con el botón de play. La segunda es que pulsando la tecla spr, la entrada de tabla se borra, al igual que sucede con el botón con el icono de la papelera.

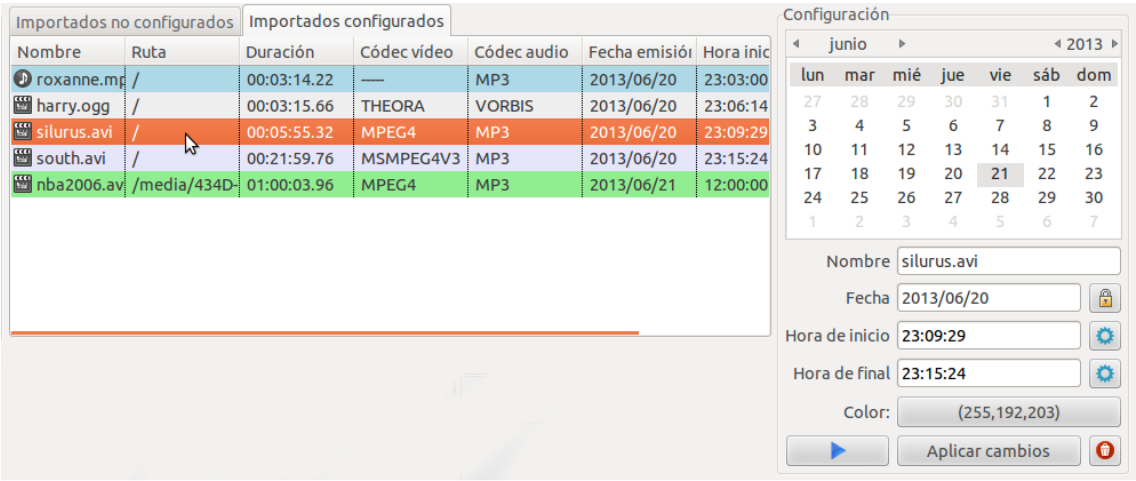


Figura 7.5

7.3.3 Parámetros de la emisión

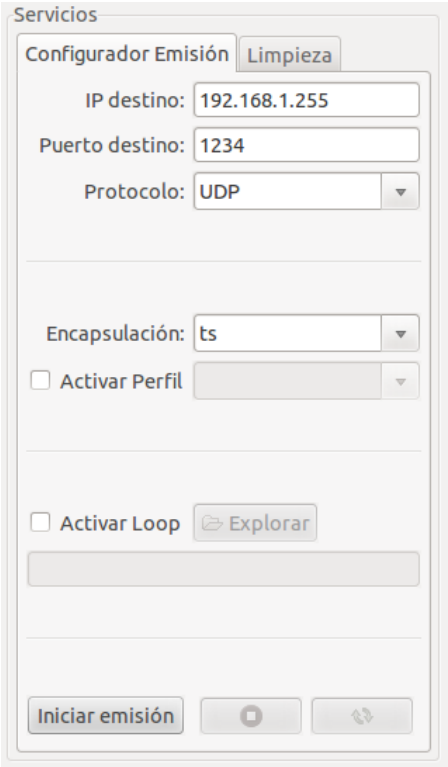


Figura 7.6

En este punto debe seleccionarse:

- *IP destino*: es la dirección IPv4 a la que irá dirigido el stream del servidor VLC que se iniciará a continuación. Puede hacerse broadcast utilizando una IP específica para tal cometido. Este campo no tendrá trascendencia si se selecciona protocolo HTTP o RTSP, ya que en tal caso, el stream no se generará hasta que el cliente lo "solicite" al servidor.
- *Puerto destino*: es el parámetro que acompaña a la IP anterior en los casos de protocolo UDP y RTP. Por el contrario, para los casos de HTTP y RTSP, será el puerto al que solicitar el stream (junto a la IP del PC en el que corremos la aplicación). En ambos casos sirve para completar el socket IP+puerto.
- *Encapsulación*: es el tipo de encapsulación que se quiere dar al stream.

Hasta ahí el conjunto mínimo de campos a rellenar. Como optativos quedan los siguientes:

- *Perfil*: permite seleccionar un tipo predeterminado de transcodificación a utilizar sobre los vídeos que se emitan. La transcodificación consta de una conversión a cierto códec de vídeo y audio, y una encapsulación concreta.
- *Loop*: permite seleccionar un vídeo que se emitirá en lazo, de forma repetida, durante todos los periodos en los que no haya ningún otro vídeo emitiéndose.

Finalmente, los botones de inicio, parada y actualización de la emisión (respectivamente, en la figura 6, parte inferior):

- *Iniciar emisión*: al pulsar sobre este botón se genera una instancia de VLC (invisible para el usuario), el cual hará de servidor. Éste es verdadero encargado del streaming, y cuando uno desea recibir cierto stream, debe solicitárselo (véase último punto del manual: recepción del stream con VLC).
  El proceso de instanciación del VLC puede tardar unos segundos, y el progreso se puede ir viendo en la barra inferior de la interfaz gráfica y en una etiqueta en la parte inferior derecha (figura 7). Una vez se haya completado, se estará en plena emisión.
  Al completarse la instanciación (y, por tanto, estar en emisión) el botón cambia de nombre a "Reiniciar", y su función pasa a ser la de parar y actualizar (ambas en uno) la emisión, tal como se ve a continuación.

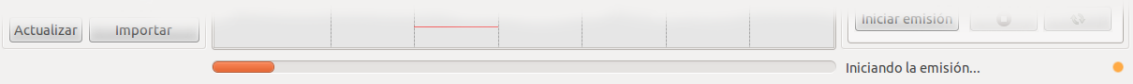


Figura 7.7

- *Parar emisión*: detiene en seco la emisión, matando el proceso VLC del párrafo anterior.
- *Actualizar emisión*: sirve para refrescar la emisión, de modo que se reciban los nuevos cambios aplicados en la pestaña de "Configurador Emisión" (figura 6). Si sólo se cambia el Loop, la emisión no requiere un reinicio, pero de lo contrario sí.
  Es conveniente utilizar este botón si se considera que el VLC no ha cargado la configuración correctamente, independientemente del motivo.

7.4 Sistema de limpieza

Es un sistema de autolimpieza por parte del programa que consiste en borrar todas las entradas de tabla con emisiones de vídeos ya caducadas, es decir, cuya hora de final configurada sea inferior a la hora presente.

Por defecto, la limpieza se realiza a las 00:00, pero la hora y el minuto se pueden cambiar desde la pestaña "Limpieza" de la interfaz (figura 8).

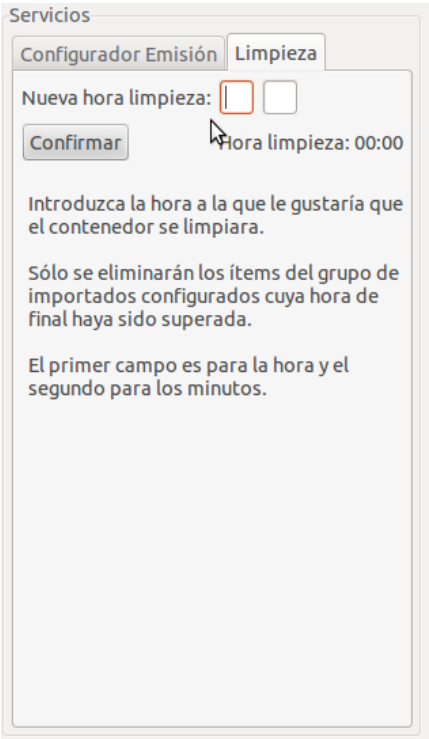


Figura 7.8

7.5 Guardado y carga del proyecto

Estas opciones son accesibles desde el menú de la aplicación (parte superior de la interfaz, figura 9), clicando en "Archivo".

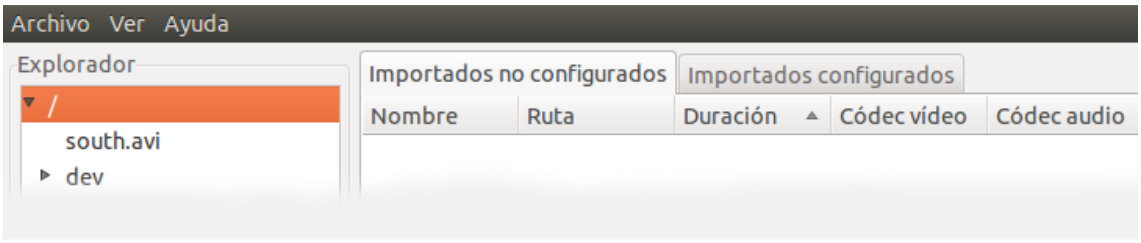


Figura 7.9

Las opciones que aparecen son:

- *Guardar proyecto*: sólo accesible una vez se ha iniciado la emisión. Guarda la información de todos los vídeos a emitir, y permite una recuperación posterior.
- *Cargar proyecto*: sólo accesible con la emisión parada (o sin iniciar). Busca en el PC los vídeos presentes en el archivo guardado y regenera las entradas de tabla pertinentes, directamente en la tabla de la pestaña de importados-configurados.

7.6 Visualización del contenido programado

Hay 3 formas de ver el contenido programado:

1. *Visualización gráfica directa*: mediante el panel gráfico (figura 1, color azul), donde aparecerán los contenidos programados para el día seleccionado en el calendario (figuran 4, parte superior), los 3 días previos y los 3 posteriores.

2. *Listado semanal*: Es prácticamente igual a la opción anterior, pero en forma de listado, mostrando la hora exacta de inicio y la hora exacta de final. Las emisiones aparecen en orden. Puede verse a continuación, en la figura 10.

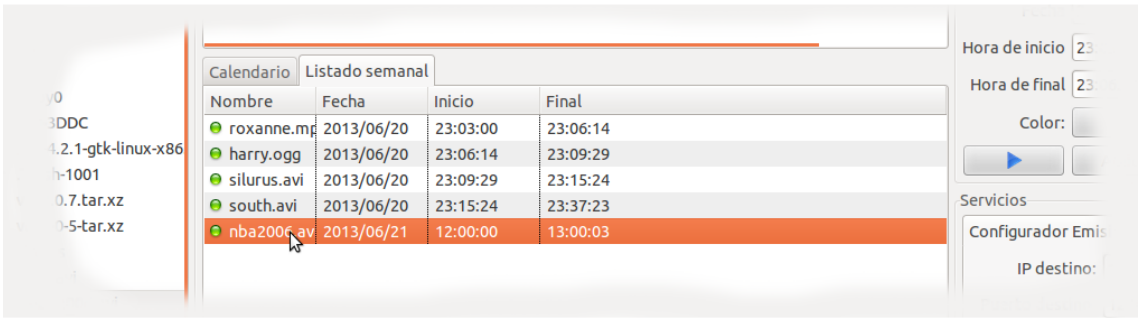


Figura 7.10

3. *Código VLM*: este método es para aquél que haya trabajado con código VLM alguna vez. Es el que se carga en el VLC que hace de servidor, cada vez que realizamos algún cambio en la emisión. Podemos ver el código, donde se encuentra toda la información de la emisión haciendo clic en la opción "ver" del menú de la aplicación y seleccionando "Listado VLM cargado". Aparecerá una ventana emergente, mostrando algo semejante a lo mostrado en la figura 10.

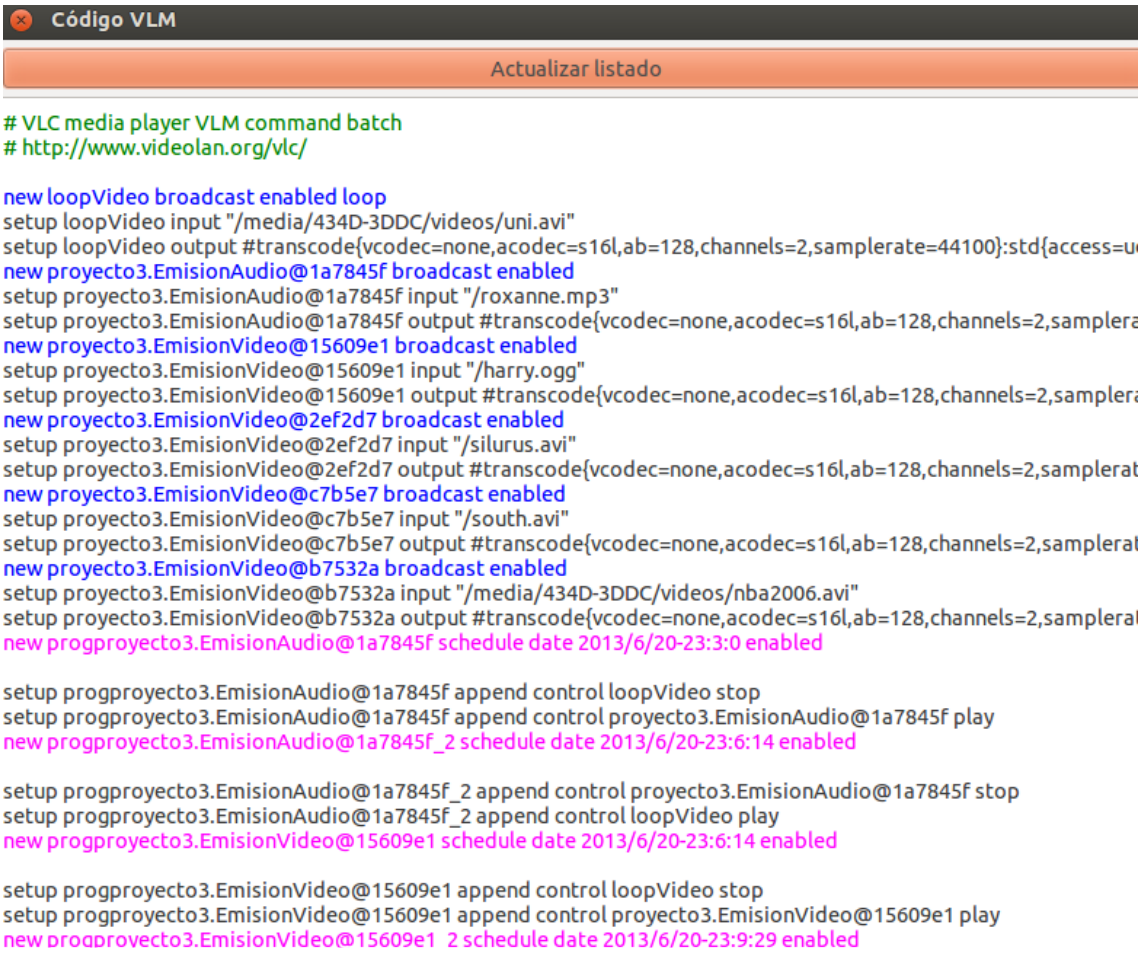


Figura 7.11

Truco importante: La segunda opción (Listado semanal), además, proporciona un método para configurar con mayor rapidez y comodidad una ráfaga de emisiones de vídeos consecutivas. El método consiste en clicar en una entrada de tabla sin configurar (tal como sería el caso de harry.ogg, en la figura 4) y posteriormente en hacer doble clic sobre un elemento del listado (por ejemplo, nba2006.avi de la figura 10); lo que sucede es lo siguiente:

1. El campo de fecha de la figura 4 coge el valor de la fecha del elemento del listado (en este caso, 21/06/2013. Dicho de otro modo, se escribe 21/06/2013 en el campo de fecha de la figura 4.
2. La hora de final (en este caso las 13:00:03) se escribe automáticamente en el campo de hora de inicio de la figura 4.

De esta forma, con un solo clic más (el de la opción de autocompletar hora de final de la figura 4) habremos acabado la configuración de la emisión de harry.ogg. Bastaría clicar en botón aplicar de la figura 4 para que la emisión se cargara y apareciera en la lista de la figura 10, justo a continuación de nba2006.avi.

7.7 Consejos

- Es recomendable (aunque no necesario) haber iniciado la emisión antes de configurar ninguna entrada de tabla. De esta forma evitará que aparezcan constantes mensajes

de aviso advirtiéndole de que la emisión no está iniciada, y además podrá ser consciente en todo momento de los vídeos correctamente configurados (figura 1, color azul, o figura 10).

- Si intenta hacer cambios significativos en la emisión, tales como modificar una entrada de tabla cuyo vídeo asociado está en emisión en ese instante, causará la aparición de mensajes de aviso en modo de ventanas emergentes. Haga caso a lo que éstas dicen antes de continuar.
- Si cree que la programación no se ha cargado en el VLC correctamente (dado que no aparece nada en el visor (figura 1, color azul), asegúrese de que en el calendario (figura 4, parte superior) esté seleccionado el día que usted quiere visualizar. Si el problema no es ese, pruebe a actualizar la emisión. Si perdura, pruebe a reiniciarla.

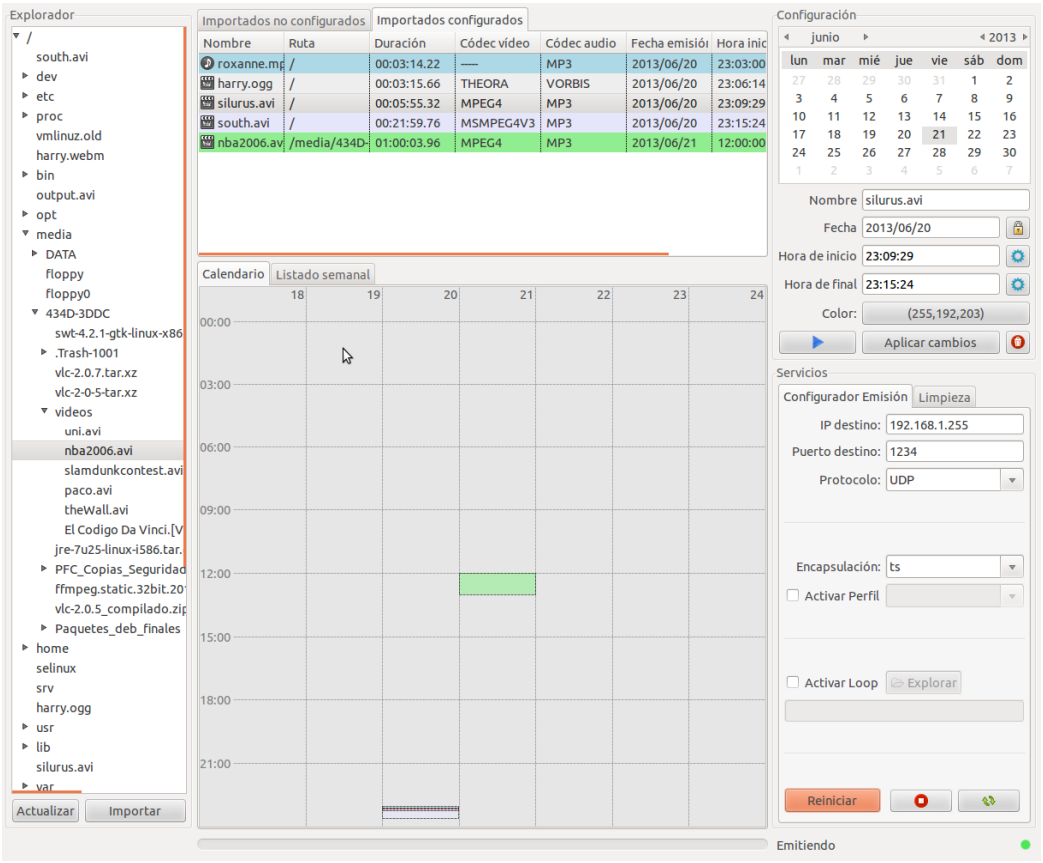


Figura 7.12 – Emisión activa con contenidos programados.

8 Conclusiones y líneas futuras

Las conclusiones generales tras la realización del proyecto han sido:

- Se ha cumplido el objetivo inicial, que era crear una versión estable de la aplicación previamente desarrollada.

- Se ha ido más allá, incorporando una amplia serie de mejoras, listadas en el apartado 1.4.

- A nivel personal, el proyecto ha supuesto el salto de madurez que el desarrollador pretendía, asentando conocimientos sobre el lenguaje de programación java. Este salto se ha logrado tras poner en juego la mayoría de técnicas previamente adquiridas (mediante lectura y realización de ejercicios de diversas fuentes, principalmente el "Piensa en Java" de Bruce Eckel [1]).

- Como desarrollador amateur, un proyecto de estas características da la oportunidad de hacerse a la idea de cómo están desarrollados muchos de los softwares comerciales.

- La realización de este proyecto sirve para poder afrontar, posteriormente, otros trabajos de desarrollo de software (si se diera el caso).

- Otra ventaja de realizar este proyecto en concreto ha sido la posibilidad de aprender características sobre streaming.

A continuación se listan las características del proyecto que podrían mejorarse:

- La detección del vídeo en emisión corre a cargo de un hilo en bucle infinito. Aunque el bucle se recorre varias veces en un segundo, se da lugar a márgenes temporales de error en tal detección.

- Un vídeo o audio de duración igual o mayor a 24 horas produciría comportamientos extraños en el programa, en la parte del visor de contenidos y configuración de horarios.

- Cuando se cierra la ventana principal de la aplicación, el hilo del bucle infinito mencionado muere por errores al no poder acceder a los widgets gráficos. Sería aconsejable un mecanismo de finalización del hilo independiente de errores.

- Que el sistema de limpieza actuara constantemente, eliminando los objetos EmisionMedia si poseen una hora de final inferior a la hora actual del sistema. Actualmente, se actúa una vez cada 24 horas, en un instante programable por el usuario.

- En el visor de contenidos programados, dar información acerca de cada cuadro dibujado; por ejemplo, que se muestre la información del vídeo al pasar el ratón por encima. Actualmente, el sistema de dibujo del visor no posee ninguna característica "interactiva", siendo puramente visual.

A continuación se listan características que podrían añadirse al proyecto:

- Aprovechar la funcionalidad del software FFmpeg para incorporar un conversor de formatos a la interfaz gráfica.

- Incorporar algún mecanismo para ver el streaming que se esté realizando en la propia GUI, en vez de tener que hacer uso de una instancia VLC externa.

- Aprovechar la opción de "elementos de tipo VoD" del VLM, y crear la función de servidor de streaming bajo demanda. Esta característica está más cerca de ser un proyecto independiente que de ser un "añadido" al proyecto actual.

9 Bibliografía

[1] Eckel, Bruce. *Piensa en Java: cuarta edición*. Prentice Hall, 2007.
[2] Schmuller, Joseph. *Aprendiendo UML en 24h*. Prentice hall, 2001.
[3] Hodei Altuna Cueli, Ignacio Echeverría Sánchez. *Desarrollo de un sistema de gestión de una parrilla de televisión* (Proyecto final de carrera, 2011).

Webs de apoyo:
[4] www.videolan.org
[5] www.ffmpeg.org
[6] www.eclipse.org
[7] www.eclipse.org/swt