ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Desarrollo de un sistema de gestión de una parrilla de televisión

Proyecto final de carrera

Ingeniería técnica de telecomunicación, especialidad en sonido e imagen

Pamplona, 5 de Julio de 2011

Dr. Mikel Sagüés García

Hodei Altuna Cueli

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Desarrollo de un sistema de gestión de una parrilla de televisión

Proyecto final de carrera

Ingeniería técnica de telecomunicación, especialidad en sonido e imagen

Pamplona, 5 de Julio de 2011

Ignacio Echeverría Sánchez

Mikel Sagüés García

Índice

1. Introducción

En este primer punto se va a tratar de situar al lector en el punto de inicio del desarrollo del proyecto. Para ello, lo primero que se va a hacer es un breve resumen de la situación tecnológica de la empresa para la que se desarrolla. Seguidamente se explicará el modo mediante el cual se espera mejorar el sistema actual. Por último, se hace una exposición de la motivación que llevó a la aceptación de este proyecto.

El presente proyecto tiene como fin mejorar la gestión de programación de vídeos del sistema implementado para la emisión del canal de Televisión Digital Terrestre del ayuntamiento del valle de Aranguren: Aranguren TV. Dado que este sistema propone una programación de la escaleta de contenidos manual se creará una interfaz gráfica con la que editar esta programación de forma simple e intuitiva.

Para añadir una aplicación de gestión de vídeos a un sistema de emisión de un canal de televisión previo hay que considerar las posibilidades y limitaciones que este ofrece.

1.1 Sistema de emisión de TDT Aranguren TV

Este sistema se compone de un PC (sistema basado en Windows) 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.

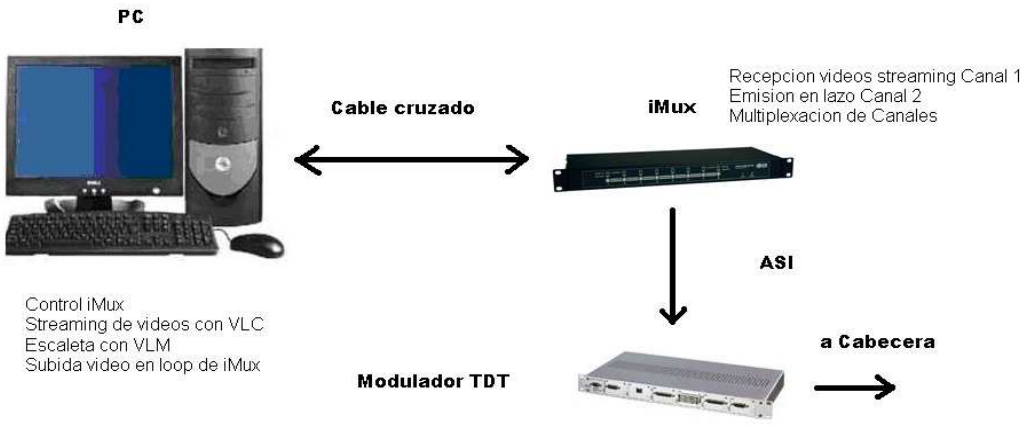


Figura 1.1 Sistema de emisión.

De esta manera, el PC se utilizará 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 mandará por *streaming* al *iMux* para que éste los adapte al formato de emisión en TDT.

- El sistema de emisión propuesto constará de 2 canales.

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

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

- Recursos Software que se utilizarán.

Como se ha comentado, el PC se utilizará para la gestión del multiplexor iMux, así como para la emisión de los contenidos por *streaming*. Para ello, el PC deberá tener instalados una serie de programas que se describirán a continuación:

-En primer lugar, se utilizará el navegador Web *Mozilla Firefox* para la gestión del iMux y para comprobar la parrilla de emisión. En dicho navegador estarán creados 2 accesos directos. Uno que dirige directamente a la interfaz Web del iMux y otro que dirige a la interfaz Web del VLM (se explica que es esto a continuación).

-En segundo lugar, deberá estar instalado el *Vídeo Lan Client* (VLC versión 0.9.4). Este programa permite hacer una emisión de vídeos por *streaming*, es decir, una emisión de vídeos a través de una red de ordenadores. El iMux es capaz de recibir varias de estas emisiones y juntarlas en una sola para crear un Multiplex. Además, el VLC posee una funcionalidad extra; el *Vídeo Lan Manager* (VLM) que permite programar la fecha y hora a la que se quiere efectuar el *streaming*, es decir, permite crear una escaleta de contenidos (parrilla de emisión).

-El VLM trabaja con comandos escritos en un archivo de texto. Para su mayor facilidad de uso se instalará también el programa *UltraEdit* que hace más amigable el tratamiento de los archivos de texto de VLM.

-Por último se han de codificar correctamente los vídeos a emitir con el programa de edición de vídeo *Premiere*. Este programa no estará instalado en el PC de gestión y emisión para una mayor estabilidad del sistema.

El proyecto que se expone a continuación se centra en la creación y la programación del archivo de configuración VLM que regirá la escaleta de emisiones.

1.2 Aplicación de gestión de vídeos

La funcionalidad del *Vídeo Lan Manager* (VLM) permite crear una escaleta de contenidos a partir de comandos escritos en un archivo de texto. El sistema implementado para Aranguren TV propone el uso del programa *UltraEdit* para la edición de este archivo. Esto conlleva tener que programar toda la escaleta manualmente, dando lugar a posibles errores humanos por la incomodidad de este método.

Para una programación cómoda e intuitiva se propone la creación de una interfaz gráfica con la que gestionar los vídeos de emisión. Para implementar esta interfaz el lenguaje de

programación utilizado será *Java* [19]. Además, para una lectura cómoda de la programación existente, esta interfaz permitirá la representación de los vídeos programados.

El segundo capítulo hace una introducción a Java. Es aquí donde se comienza a comprender las posibilidades que este programa ofrece y se crea un primer programa generador de código VLM.

Como ya se verá en el capítulo 3 la conexión entre este programa y VLC no será la propuesta en el sistema de emisión de TDT para Aranguren TV. Este cambio se debe a la poca fiabilidad que esta interfaz Web ofrece. En su lugar se utilizará la interfaz *Telnet* [15].

Otro de los inconvenientes que la versión previa para la televisión de Aranguren ofrece es el conocimiento de la duración de los vídeos. El conocimiento de los vídeos es fundamental para llevar un control exhaustivo de la escaleta de emisión. La versión previa propone nombrar los vídeos con su duración. En el capítulo cuatro se explica una alternativa eficaz a esta otra.

1.3 Motivación

La primera motivación que llevó a aceptar el desarrollo de este proyecto es su naturaleza. Se trata de una aplicación útil que va a intentar resolver un problema real de una empresa con lo que podía servir para establecer un punto de experiencia muy gratificante.

Además, el lenguaje en el que estaba propuesta la realización del mismo era del todo desconocido, por lo que suponía un doble reclamo. Primero, supone el aprendizaje de una herramienta como es Java para el futuro desarrollo profesional. Segundo porque va en consonancia con lo que pensamos es la idea de desarrollo de la carrera, formar trabajadores capaces de solucionar problemas en campos con los que pueden no estar del todo familiarizados. Personalmente, podría servir para ensalzar la autoestima obteniendo conciencia de los conocimientos adquiridos durante los años que han durado los estudios.

A estas motivaciones de carácter útil, se le une una especial ilusión que reside en la creación de una herramienta de la que alguien pueda verse beneficiado. Además esta sensación aumenta cuando se tiene en cuenta que esta ha sido creada de la nada, siguiendo los criterios que se consideraban mejores para que cumpla con su cometido.

Con todo ello, se comienza con el desarrollo del proyecto cuyo inicio se centra en la familiarización con el lenguaje de programación a usar.

2. Primeros pasos

No es posible calificar el esfuerzo dedicado en el desarrollo de un trabajo sin tener en cuenta las herramientas y los conocimientos con los que se cuenta antes de comenzarlo. Por ello conviene poner en consonancia los conocimientos sobre la materia con los que se contaba y la evolución de los mismos.

En este capítulo se va a tratar de hacer un análisis de los conocimientos con los que se contaba previamente, al comienzo de desarrollo del proyecto en el punto 2.1. En el 2.2, se pasará a hacer una pequeña introducción al lenguaje de *Java* [19]. En los siguientes apartados se detallará el camino seguido para la familiarización con dicho lenguaje (2.3) y se explicará el primer programa que se realizó con vistas a la solución (2.4).

2.1 Conocimientos previos

Es muy sencillo contar los conocimientos del lenguaje *Java*, o de cualquier otro lenguaje orientado a objetos, con los que se contaba antes de comenzar la realización de este proyecto porque eran prácticamente nulos. Es cierto que se tenían referencias de su uso web, gracias a la caracterización de su símbolo en muchas de sus aplicaciones. Pero este conocimiento y el aumento de demanda de programadores conocedores de este tipo de lenguaje gracias al auge de los teléfonos inteligentes con sistema operativo *Android,* era todo el conocimiento previo.

2.2 Introducción a Java

En este apartado se realizará una introducción a *Java* orientado a aquellos lectores que no estén familiarizados con él, a razón de hacer más comprensible el resto de la lectura

2.2.1 Introducción

Java es un lenguaje desarrollado por *Sun Mycrosistems* [6] con la intención de competir con Microsoft en el mercado de la red. Sin embargo, su historia se remonta a la creación de una filial de *Sun* (*FirstPerson*) enfocada al desarrollo de aplicaciones para electrodomésticos, microondas, lavaplatos, televisiones... Esta filial desapareció tras un par de éxitos de laboratorio y ningún desarrollo comercial.

Sin embargo, para el desarrollo en el laboratorio, uno de los trabajadores de *FirstPerson, James Gosling*, desarrolló un lenguaje derivado de *C++* que intentaba eliminar las deficiencias del mismo. Llamó a ese lenguaje *Oak*. Cuando *Sun* abandonó el proyecto de *FirstPerson*, se encontró con este lenguaje y, tras varias modificaciones (entre ellas la del nombre), decidió lanzarlo al mercado en verano de 1995.

El éxito de *Java* reside en varias de sus características. Java es un lenguaje sencillo, o todo lo sencillo que puede ser un lenguaje orientado a objetos, eliminando la mayor parte de los problemas de *C++,* que aportó su granito (o tonelada) de arena a los problemas de *C*. Es un lenguaje independiente de plataforma, por lo que un programa hecho en *Java* se ejecutará igual en un *PC* con *Windows* que en una estación de trabajo basada en *Unix*. También hay que destacar su seguridad, desarrollar programas que accedan ilegalmente a la memoria o realizar *caballos de troya* es una tarea propia de titanes.

Cabe mencionar también su capacidad *multihilo,* su robustez o lo integrado que tiene el protocolo *TCP/IP*, lo que lo hace un lenguaje ideal para Internet. Pero es su sencillez, portabilidad y seguridad lo que le han hecho un lenguaje de tanta importancia.

2.2.2 Portabilidad

La portabilidad se consigue haciendo de *Java* un lenguaje medio interpretado medio compilado. ¿Cómo funciona esto? El código fuente se compila a un lenguaje intermedio cercano al lenguaje máquina pero independiente del ordenador y el sistema operativo en que se ejecuta (llamado en el mundo Java *bytecodes*) y, finalmente, se interpreta ese lenguaje intermedio por medio de un programa denominado máquina virtual de *Java*.

Este esquema lo han seguido otros lenguajes, como por ejemplo Visual Basic. Sin embargo, nunca se había empleado como punto de partida a un lenguaje multiplataforma ni se había hecho de manera tan eficiente. Cuando Java apareció en el mercado se hablaba de que era entre 10 y 30 veces más lento que *C++.* Ahora, con los compiladores *JIT* (*Just in Time*) se habla de tiempos entre 2 y 5 veces más lentos. Con la potencia de las máquinas actuales, esa lentitud es un precio que se puede pagar sin problemas contemplando las ventajas de un lenguaje portable.

2.2.3Orientación a objetos

Dado que Java es un lenguaje orientado a objetos, es imprescindible entender qué es esto y en qué afecta a nuestros programas. Desde el principio, la carrera por crear lenguajes de programación ha sido una carrera para intentar realizar abstracciones sobre la máquina. Al principio no eran grandes abstracciones y el concepto de lenguajes imperativos es prueba de ello. Exigen pensar en términos del ordenador y no en términos del problema a solucionar. Esto provoca que los programas sean difíciles de crear y mantener, al no tener una relación obvia con el problema que representan No abstraen lo suficiente.

Muchos paradigmas de programación intentaron resolver este problema alterando la visión del mundo y adaptándola al lenguaje. Estas aproximaciones modelaban el mundo como un conjunto de objetos o de listas. Funcionaban bien para algunos problemas pero no para otros. Los lenguajes orientados a objetos, más generales, permiten realizar soluciones que, leídas, describen el problema. Permiten escribir soluciones pensando en el problema y no en el ordenador que debe solucionarlo en último extremo. Se basan en cinco características:

- Todo es un objeto**.** Cada elemento del problema debe ser modelizado como un objeto.
- Un programa es un conjunto de objetos diciéndose entre sí que deben hacer por medio de mensajes. Cuando necesitas que un objeto haga algo le mandas un mensajes. Más concretamente, ejecutas un método de dicho objeto.
- Cada objeto tiene su propia memoria, que llena con otros objetos. Cada objeto puede contener otros objetos. De este modo se puede incrementar la complejidad del programa, pero detrás de dicha complejidad sigue habiendo simples objetos
- Todo objeto tiene un tipo. En jerga POO, cada objeto es una instancia (un caso particular) de una clase (el tipo general). Lo que distingue a una clase de otra es la respuesta a la pregunta, ¿qué mensajes puedes recibir?

- Todos los objetos de un determinado tipo pueden recibir los mismos mensajes. Por ejemplo, dado que un objeto de tipo Gato es también un objeto de tipo Animal, se puede hacer código pensando en los mensajes que se mandan a un animal y aplicarlo a todos los objetos de ese tipo, sin pensar si son también gatos o no.

2.3 Tutoriales y ejercicios

En este caso, la falta de información conlleva la necesidad de una primera lectura sobre la filosofía tanto del lenguaje orientado a objetos en general como de *Java* en particular. Este primer contacto se realizó por medio de "*Thinking in Java*" de *Bruce Eckel* [3], reconocido autor de libros introductorios a distintos lenguajes de programación. Aunque aquella lectura resultó ser muy fructífera en cuanto a la idea en la que se basaba Java, el paso inmediato estaba en tutoriales y ejercicios.

Es en este punto donde hay que hacer un agradecimiento especial. La compartición de conocimientos es algo que en muchos casos cuesta tiempo y dinero, sin embargo gente anónima como el señor *niktutos* [4], sacan tiempo para la realización de tutoriales de gran nivel e incluso con la posibilidad de resolución de dudas. De hecho las herramientas más manejadas durante toda la resolución del proyecto se han basado en este concepto de compartición de conocimientos, haciendo de internet el sistema de consulta más útil con la que se ha contado.

Con estas premisas se comenzó a trabajar en una serie de pequeños programas para el desarrollo de la familiarización con *Java*, tras lo que se pasó a realizar un pequeño programa que dándole un la dirección y hora de los vídeos, hiciera la programación de los archivos VLM (ver 1.1) para su posterior lectura de VLC.

2.4 Primer generador de código

El primer generador de código se basa en una interfaz gráfica muy sencilla desde la que poder hacer una programación, evitando crear este código manualmente. El programa reclamará que se rellenen una serie de campos relacionados con la instancia que se quiere programar, generando el código que lo hace posible. Esta información se guardará en un documento de texto llamado VLM.txt por lo que será previamente borrado para evitar que el código a programar sea contaminado por programaciones antiguas.

```
String sFichero = "C:/VLM.txt";
File fichero = new File(sFichero);

if(fichero.exists()){
    fichero.delete();
}
```

Todo este proceso es precedido por la creación de la interfaz gráfica, que requiere un orden. Obviamente cada objeto que forma parte de él tiene un nombre, para poder más tarde realizar las comunicaciones entre ellos. Este orden es establecido mediante *GridBagConstraints*, método que permite asignar un tamaño a los objetos y situarlos en la interfaz. Se puede observar como al objeto *gbc* del tipo GridBagConstraints se le asigna un lugar donde

comenzar, un tamaño, y el modo en el que crecerá cuando aumente o disminuya la ventana. Tras esta asignación de las propiedades que va a tener el objeto, este le es asignado a *File*, por lo que queda situado en nuestra interfaz gráfica.

```
        File = new JTextField(20);
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.gridx = 0;
        gbc.gridy = 3;
        gbc.gridheight = 1;
        gbc.gridwidth = 8;
        gbc.weightx = 1.0;
        gbc.weighty = 0.0;
        gbc.fill = GridBagConstraints.HORIZONTAL;
        add(File,gbc);
```

La idea es que al aceptar los parámetros, se genere la programación recogiendo la información introducida por el usuario. Para ello, cuando usamos la pestaña de activar, el programa leerá el texto escrito en los campos y generará dicha programación.

```
if(e.getSource()==programar){

areaProgramar.append("new prog"+elementoBroadcast.getSelectedItem()
        +" schedule date "+año.getText()+"/"+mes.getText()+"/"
        +dia.getText()+"-"+hora.getText()+":"+minuto.getText()+
        ":"+segundo.getText()+" enabled");

    if(rv.getnuevaRepeticion()==null){
    areaProgramar.append("\r\n");
    }else {
    areaProgramar.append(" period "+rv.getnuevaRepeticion()+"\r\n");
    }
```

Una vez establecida la programación de ambos campos podemos pasar a exportar el archivo, para lo que se utiliza el archivo antes creado y un método de escritura *Buffered Writter.*

```
 if(e.getSource()==exportar){

 try{
 BufferedWriter bw= new BufferedWriter(new FileWriter(sFichero));
 bw.write(areaActivar.getText()+areaProgramar.getText());
 System.out.println(str);
 bw.close();
 }catch (IOException ioe){
         ioe.printStackTrace();
 }

 }
```

Con ello, el archivo ya está generado y listo para que VLC pueda realizar su carga.

2.5 Conclusiones

En este capítulo se ha realizado una exploración de los conocimientos previos que se contaban antes de comenzar la programación. En él se retratan los pasos dados para la familiarización con el lenguaje *Java*, incluyendo una serie de ejercicios.

La realización de estos ejercicios que culminaron en este programa consiguió una familiarización con un lenguaje de programación nuevo, y el aprendizaje de algunas herramientas que serían utilizadas en el programa final para la exportación de los archivos VLM.txt.

3. La problemática VLM

3.1 Introducción

Si lo que desea es realizar la programación utilizando VLC [14], debe haber un modo de establecerla. Concretamente en Aranguren Televisión, proponía que esta parrilla, previamente programada en un archivo de configuración VLM [16], fuese cargada en el programa VLC.

Un archivo de configuración VLM es un archivo de texto donde se almacena la programación de los comandos que controlan el funcionamiento de VLC. VLM (*VideoLan Manger*) es una herramienta integrada en VLC diseñada para controlar múltiples volcados con un solo proceso de VLC. Permite realizar *streaming* múltiple y vídeo bajo demanda (*VoD*). Así, es posible crear puntos temporales en los que se ejecute una acción determinada, como *play* o *stop*, además de añadir o borrar vídeos de la programación. De este modo se puede programar la escaleta de emisión previamente y el programa VLC actuará bajo su mando continuamente.

VLC presenta 4 maneras para cargar un archivo de configuración VLM al VLC.

3.1.1 Activar la configuración mediante el reinicio de VLC

Cambiando las propiedades del programa VLC para que al ejecutarse cargue el archivo de configuración VLM. Esto requiere que este archivo siempre este guardado con el mismo nombre y en la misma dirección. Es posible configurar el PC para que cada vez que se arranque ejecute VLC, cargando así el archivo de configuración VLM.

3.1.2 Crear configuración VLM desde VLC

Accediendo al configurador VLM desde la pestaña HERRAMIENTAS > CONFIGURACION VLM > IMPORTAR es posible introducir la programación manualmente. Esto lo hace poco práctico, porque, aunque permite cargar archivos VLM, no se puede manejar desde otra interfaz.

3.1.3 A través de la interfaz Web

Antes de usar este método hay que habilitar la interfaz Web [16] en VLC. Una vez habilitada se podrá acceder a él escribiendo en el navegador http://localhost:8080/vlm.html. Para cargar el archivo de configuración VLM se debe copiar el su contenido en la interfaz Web y mandársela.

3.1.4 Mediante la interfaz Telnet

Primeramente, hay que habilitar la interfaz Telnet [16]. Para hacer la conexión con el software VLC se requiere un cliente *Telnet* [17], programa de emulación de terminal para teleproceso adaptado al protocolo TCP/IP. A través del cliente *Telnet* controlamos VLC mediante comandos VLM.

3.1.5 Análisis de los métodos de conexión

La versión previa para Aranguren Televisión proponía el uso del primer método, resultando ser muy limitado, ya que para ello hacía falta reiniciar el programa VLC. De este modo, en algún momento de la emisión habría que detenerla y reiniciarla cuando no hubiese ningún vídeo en reproducción. Es por ello que este método no resulta demasiado elegante.

Por otra parte, el modo de cargar el archivo VLM a través de la interfaz Web resulta poco práctica. Además de que el hecho de tener que introducir el código VLM manualmente, la interfaz Web no es fiable, suele fallar a la hora de cargar el archivo de configuración de VLM. Por estos motivos este método no se utilizará.

Finalmente, cargar el archivo de configuración VLM mediante la interfaz *Telnet* tiene el inconveniente de depender de un cliente *Telnet*.

3.2 Solución

Después de la investigación de todos los métodos (apartado 3.3) se obtuvo la solución. La interfaz *Telnet* es la herramienta más potente que VLC ofrece para cargar el archivo de configuración VLM, además de ofrecer mayor facilidad para el control de la herramienta VLM.

3.2.1 ¿Cómo funciona la interfaz Telnet?

Para empezar, Telnet es un programa que permite conectar dos ordenadores a través de la red. Uno de ellos será el servidor, que es el que espera a que otro se conecte a él (en este caso VLM creará un servidor de *Telnet*). El otro ordenador 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.

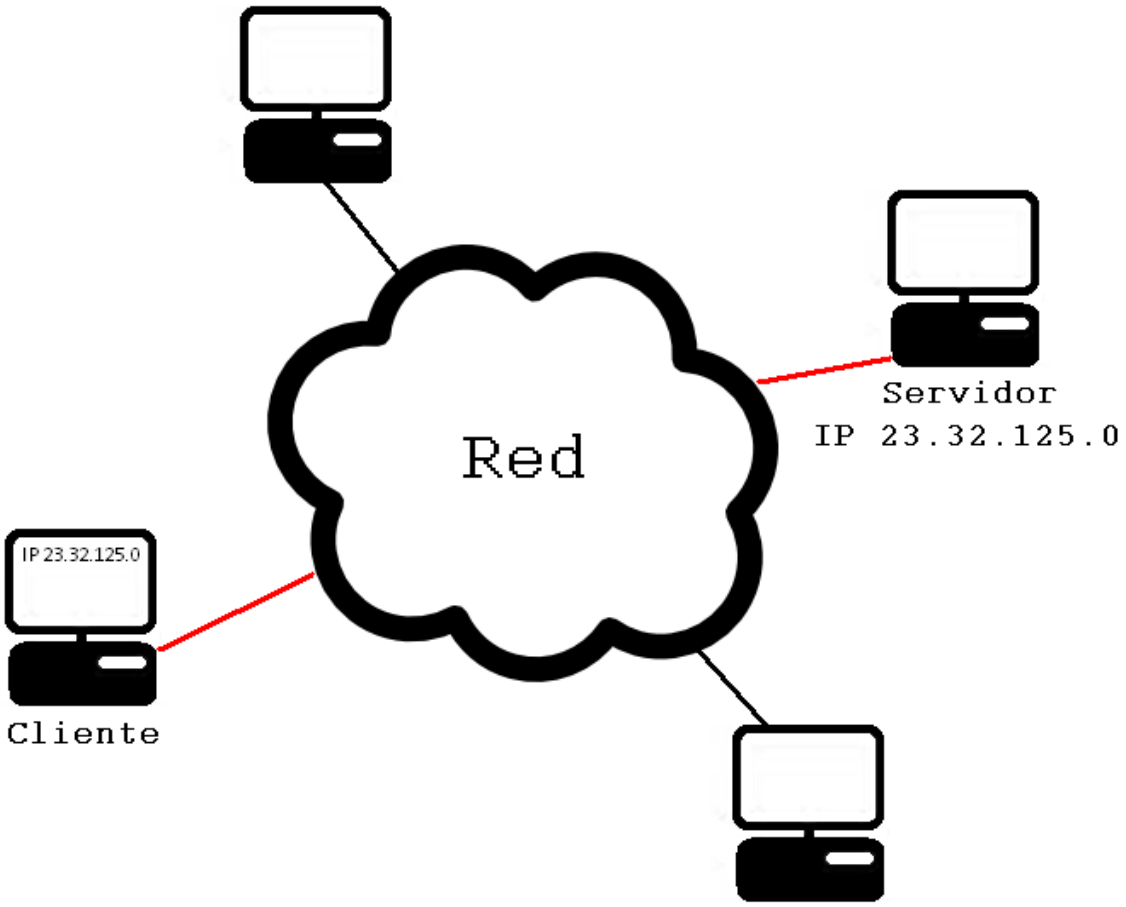


Figura 3. 1 Conexión mediante Telnet

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 salida de VLC, la gestión de datos se realizará desde la misma máquina donde el software está instalado, es decir, la red de conexión será interna.

3.2.2 Habilitar interfaz Telnet

Previamente se debe abrir una instancia VLC con la interfaz *Telnet* habilitada. Para ello existen dos posibilidades: habilitar esta interfaz desde la propia interfaz gráfica de VLC o desde línea de comandos.

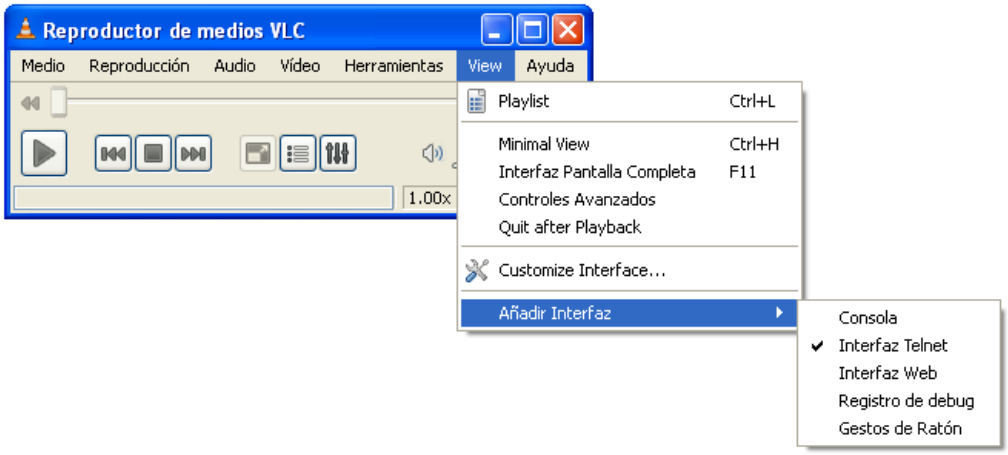


Figura 3.2 Habilitar interfaz Telnet desde interfaz gráfica de VLC

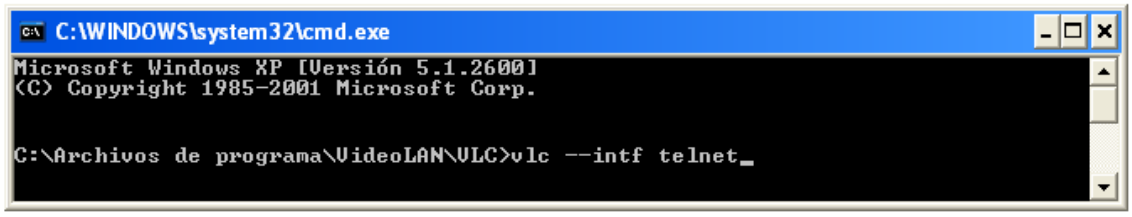


Figura 3.3 Habilitar interfaz Telnet desde línea de comandos

Hay que tener en cuenta que no todas las versiones del VLC funcionan correctamente con la conexión *Telnet*. Las antiguas versiones, no todas, no son capaces de conectarse mediante *Telnet*, mientras la más reciente (VLC 1.1.7) ejecuta de forma errónea las órdenes recibidas. La versión que finalmente hace una conexión estable y ejecuta correctamente las órdenes recibidas es la 1.0.0.

3.2.3 Conexión Telnet

Para realizar la conexión con la interfaz *Telnet* se usara un cliente *Telnet*. Windows cuenta con uno pero resulta ser bastante inestable. Dependiendo de la versión VLC que se utilice esta conexión será imposible. Para la versión elegida (1.0.0) este cliente *Telnet* funciona, aunque

con una limitación. No permite ver la escritura que se hace en él. Por este motivo se aconseja utilizar otro.

Durante la realización de este proyecto se ha utilizado el cliente *Telnet Putty*[19] . *Putty* es una implementación libre de *Telnet* para plataformas Windows y Unix. El aspecto de su interfaz es el que podemos ver en la figura 3.2.3.1, en la que se muestra también un ejemplo de cómo conectarse a la interfaz *Telnet* de VLC. En este caso se conectará al mismo equipo mediante la IP "*localhost*" (es posible conectarse a otros equipos introduciendo su IP). En el cuadro *Host Name (or IP address)* se introducirá el nombre del equipo o su dirección IP. Después de elegir el protocolo *Telnet* se escribirá el puerto en el que escucha el servidor en *Port*. En nuestro caso el puerto es el 4212. Hecho esto pulsando *Open* el programa intentará la conexión.

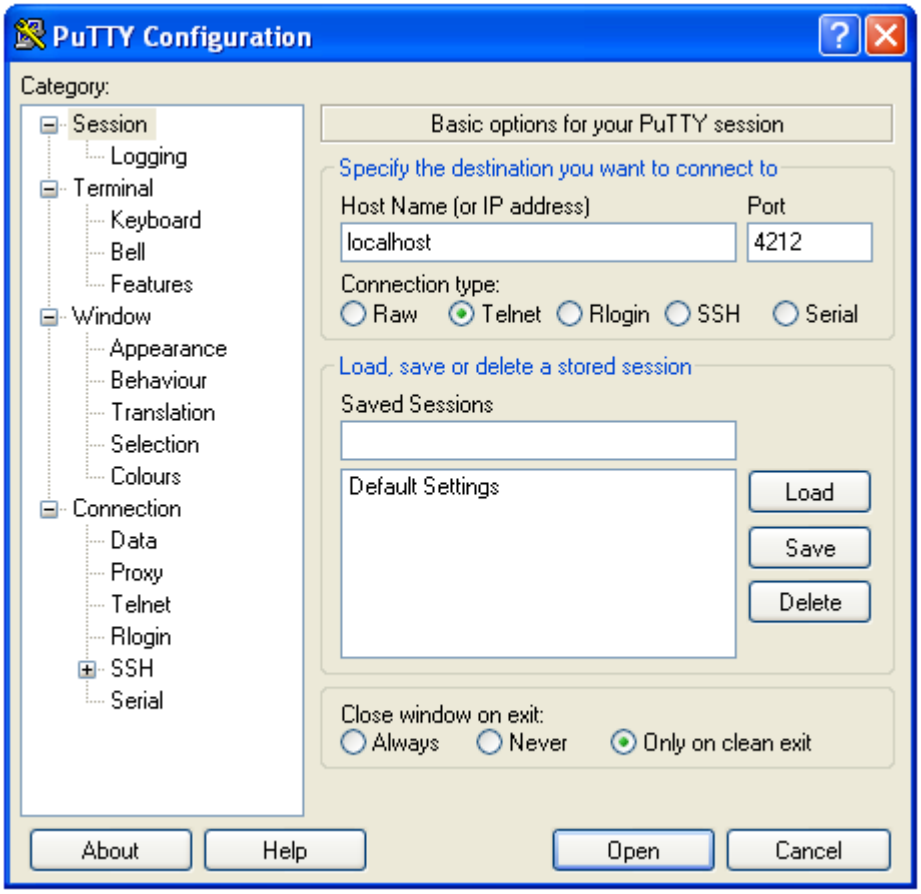


Figura 3.4 Interfaz gráfica del cliente de Telnet Putty

Una vez hecha la conexión, éste envía un mensaje pidiendo la contraseña, *Password*" (figura 3.2.3.2). La contraseña por defecto es "*admin*". Una vez conectados un mensaje de confirmación y bienvenida emergerá, *Welcome, Master*. En el caso de introducir una contraseña errónea aparecerá el mensaje *Wrong password*.

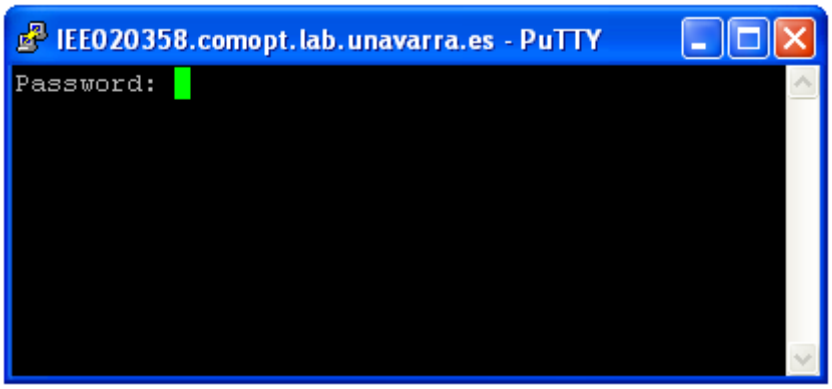


Figura 3.5 Inicio de la comunicación Telnet con VLC

A partir de aquí este cliente *Telnet* controlará la herramienta VLM del equipo remoto. Las órdenes serán enviadas a VLM. Los comandos válidos, por lo tanto, no serán ya los propios de *Telnet*, sino los de VLM.

3.2.4 Control de VLM

Antes de nada conviene recordar que la finalidad de este apartado es el de cargar un archivo de configuración VLM, aunque la interfaz *Telnet* sea capaz de mucho más. Para cargar este archivo basta con escribir, por ejemplo, el comando "*load c:\VLM.txt*" como se ve en la siguiente figura.

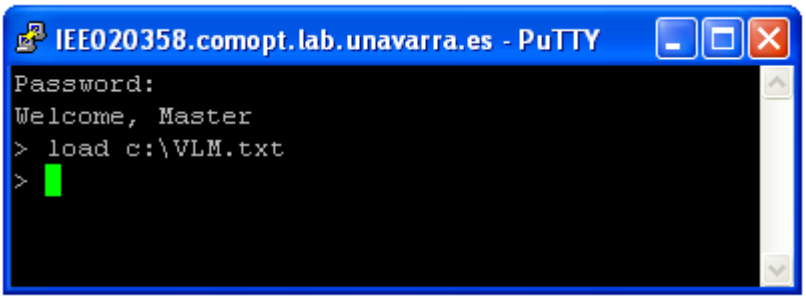


Figura 3.6 Comando para cargar el archivo de configuración VLM

VLC permite cargar varios archivos de configuración sin eliminar los anteriores, pero cuenta con una limitación. Hay que controlar que el nombre de las instancias nunca se repita. De lo contrario, VLC no cargará el archivo de configuración correctamente.

Además de cargar este archivo, esta interfaz es capaz de ejecutar varias órdenes muy interesantes, convirtiéndose así en una herramienta muy eficaz para la edición de la configuración VLM. Podemos ver la lista de estos comandos escribiendo *help*.

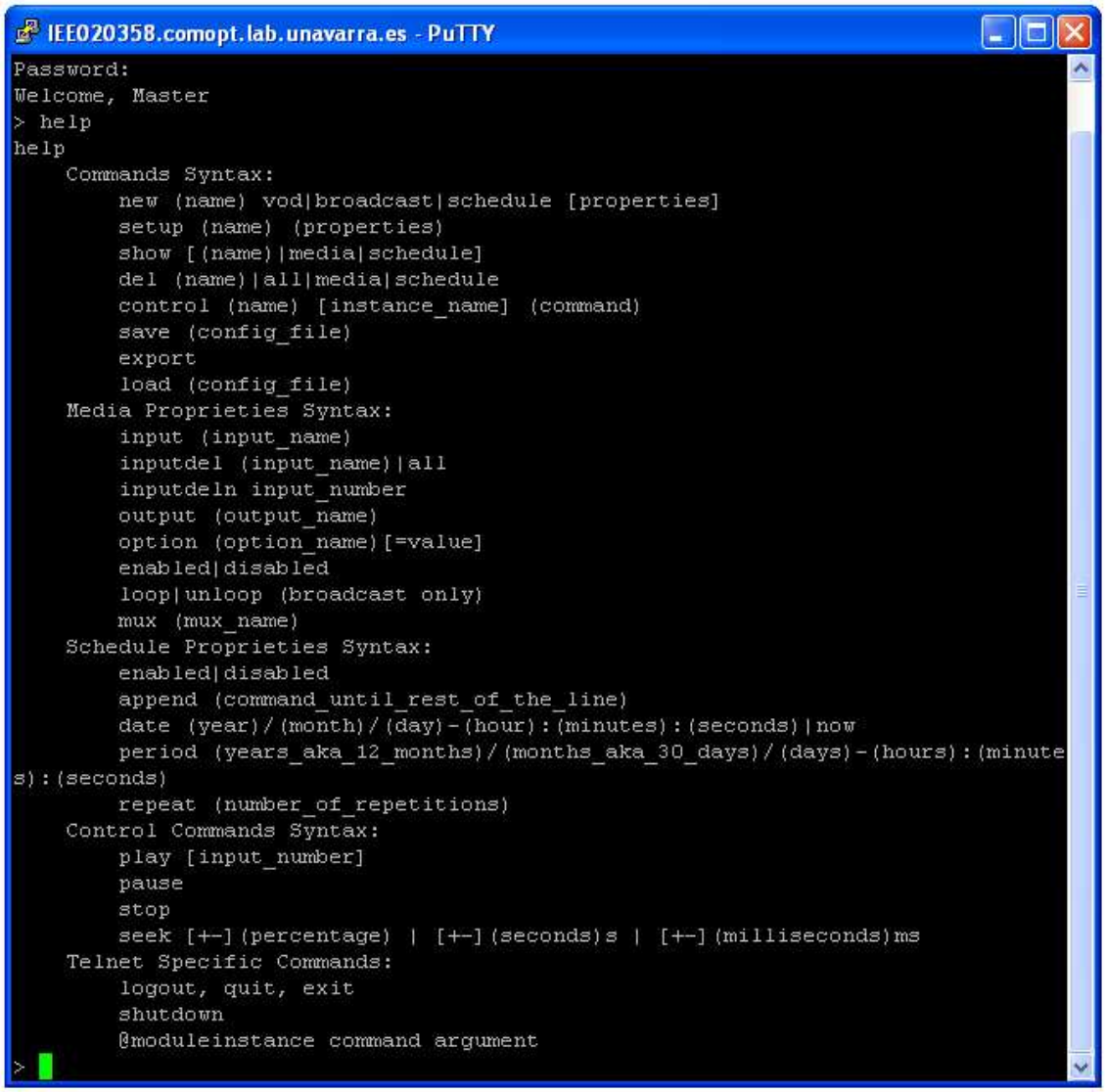


Figura 3.7 Comandos VLM

Superado el problema de cargar el archivo de configuración VLM a continuación se comentan varios comandos VLM muy útiles que esta interfaz ofrece.

- 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.

- output

  Este comando define la salida de los vídeos, es decir, permite generar cualquier tipo de flujo capaz de ser leído por VLC.

- logout

Para una desconexión segura de *Telnet* se debe usar este comando. Si esta no es la adecuada, al querer conectarse nuevamente puede no hacerlo.

3.3 Investigación previa

Antes de llegar a la solución adoptada, se estudió toda posibilidad que VLC presentaba para cargar el archivo de configuración VLM. Previo a este estudio había muchas dudas que resolver. Internet fue la herramienta que solventara todas estas gracias a la página oficial de VLC [15]. En el foro de esta página gente como *Sébastien Escuider* [20] ofrecen toda su ayuda para asimilar todos aquellos conceptos nuevos que iban apareciendo. Resulta gratificante observar como gente desconocida dona sus conocimientos en busca de un sistema en de compartición de los mismos y sin reclamar ni siquiera un agradecimiento.

Al principio se descartó el método de cargar el archivo de configuración VLM mediante el reinicio de VLC. Cabía la posibilidad de automatizar este reinicio a una hora en la que la audiencia fuese mínima, de madrugada. Este proceso no resultaba convincente ya que el acto de carga no era invisible y se conocían otros métodos que si lo eran, como el de la interfaz Web.

3.3.1 Manejo desde la interfaz Web

El primer paso fue intentar descubrir cómo la interfaz Web carga el archivo de configuración VLM mientras este está reproduciéndose. Siguiendo el código fuente de esta interfaz se descubrieron los métodos en JavaScript que utilizaba para cargar los vídeos y separar la información necesaria, pero no había ninguno de ellos que comunicara la interfaz con el programa, por lo que se dedujo que la comunicación entre ellos no pasaba por los archivos de JavaScript. Esto trunco la idea de enviar el archivo de configuración VLM directamente a los archivos JavaScript encargados de comunicarse con VLC, ya que resulto ser imposible.

Hasta este punto, siempre se utilizaba la versión VLC 0.9.4 ya que era la versión utilizada en la versión anterior para la Televisión de Aranguren. Visto que esta versión era bastante limitada para cargar el archivo de configuración, se comenzó a estudiar la última versión, VLC 1.1.7. Esta contaba con una herramienta esperanzadora que la anterior versión no tenía, el configurador VLM.

3.3.2 Crear archivo de configuración VLM desde VLC

El configurador VLM controla directamente la herramienta VLM de VLC. Es capaz de de editar la configuración VLM pero es muy práctica. Toda la información para la programación de una escaleta de emisión debe introducirse manualmente, es decir, vídeo a vídeo. Además de que esta interfaz gráfica resulta poco intuitiva no permite ver la programación ya existente.

Esto limita la intención de este proyecto por el modo incomodo de introducción de vídeos. Habría que desfragmentar toda la información de programación de cada vídeo y enviársela a VLC. Resulta una tarea inútil sabiendo que VLC es capaz de cargar toda una programación de varios vídeos a la vez. Así que se comenzó con la búsqueda de cualquier otra vía. Así que se comenzó con el estudio de la interfaz *Telnet*.

3.3.3 Manejo desde la interfaz Telnet

Cabe comentar que el manejo de la interfaz *Telnet* fue una de las primeras opciones de estudio. En aquel momento, la conexión entre el cliente *Telnet* y VLC era imposible, aun siguiendo las directrices que se encontraban en Internet. Dado el poco conocimiento respecto a este tema, no se llegó a la explicación de la imposibilidad de conexión. Se tuvo que fracasar en otras vías de estudio para volver a esta, eso sí, después de cambiar la versión de VLC utilizada.

Con la última versión de VLC, la 1.1.7, desapareció el problema de la conexión. No todas las versiones de VLC son capaces de conectarse mediante *Telnet*. De modo que una vez conectados se comenzó con la comprensión de este nuevo método. El cliente *Telnet* que Windows ofrece no es muy práctico porque no permite el visionado de la escritura. Solo basta con utilizar cualquier otro.

3.4 Conclusiones

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*.

4. Duración final de los vídeos

En este capítulo se va a presentar la necesidad y solución que acarreó la extracción de la duración de un vídeo sin abrirlo. En primer término, en el capítulo 4.1, se discute sobre la necesidad propia de la obtención de la duración de un vídeo. En el 4.2 se cuenta la solución que adopta el programa final para lograr obtener dicha información y en el 4.3 se realiza un repaso de la investigación que llevó a adoptar esta medida.

4.1 Motivo del interés de la extracción de la duración

Hay que recordar al empezar este punto, que en la versión previa que utilizaba Aranguren Televisión para su programación, una de las condiciones indispensables era poner la duración del vídeo junto a su nombre para hallarla de forma sencilla. A pesar de que era una de las opciones que se podía mantener simplificando la programación del sistema gestor, parecía bastante obvio que no era la solución más profesional, así que se comenzó a estudiar cómo se extraer la duración de un vídeo sin abrirlo. Conocer cuando acaba un vídeo programado es una herramienta que en el programa final va a ser muy usada, y a ella solo se puede llegar conociendo la duración de dicho vídeo, y la hora a la que se quiere programar, que viene dada por el interés del usuario. En la interfaz final la idea era que hubiera un tipo de calendario donde poder situar las instancias a sus horas de forma que ocupe el rango adecuado de franja horaria en que se va a reproducir, para poder tener una idea a primera vista de cómo queda la programación de cada día o semana. Esto lo hace posible el conocimiento de la duración, pero no solo eso, también que el sistema de interponer un *loop* de anuncios entre los vídeos programados son fácilmente automatizable con esta información e incluso podemos hacer un sistema de control de solapamiento entre vídeos para evitar los errores humanos en la programación. Por todo ello, la extracción del tiempo que dura un vídeo era un punto imprescindible que debía contemplar la versión final del programa.

Este proceso debía cumplir con una serie de requisitos, siendo el esencial el de sacar la duración de un archivo *mpeg* sin tener que abrirlo. Es importante además que la precisión de este sistema sea la mayor posible y que sea rápido dejando las menores huellas. Además, como posibilidad, se tratará de adoptar al sistema de compatibilidad con otros archivos.

4.2 Solución adoptada

FFmpeg [1] es una colección de software libre desarrollado para Linux pero que puede ser compilado para la mayoría de sistemas operativos. Sus opciones permiten grabar o convertir archivos multimedia entre múltiples opciones.

4.2.1 FFmpeg

Llegamos a la solución abordando el problema de forma indirecta gracias al uso de *ffmpeg*, que viene dado como un sistema de programación ejecutable en línea de comandos que sirve para realizar transformaciones de los vídeos. Para esta transformación, es necesario ejecutar el programa diciéndole el vídeo de entrada, el formato de entrada y salida y el vídeo de salida.

```
ffmpeg -i C:\maestro.mpeg -timecode_frame_start 0 -vcodec mpeg2video -an output.m2v
```

Figura 4.1 programación de ffmpeg

Esto te devuelve un nuevo vídeo y el número de *frames* totales, con lo que podemos sacarla fácilmente teniendo en cuenta la cantidad de *frames* por segundo. Esta solución no era del todo aceptable, ya que la generación de un nuevo vídeo no es uno de los objetivos y además la conversión es lo demasiado lenta, con lo que se provocaría un tiempo de espera demasiado grande para el proceso que realiza.

Se abordó el problema tratando de obtener la misma información sin obtener ningún tipo de salida ni conversión. Para ello se trató de ejecutar el archivo sin especificar salida, esperando que realizara un análisis del archivo multimedia en cuestión.

El resultado es un error en el que se realiza un análisis de los principales atributos del vídeo entre los que se encuentra la duración con una precisión que llega hasta la centésima de segundo, como puede verse en la *figura 4.2.* Es posible que este análisis se muestre para que el usuario sea capaz de hallar el motivo de dicho error.

Figura 4.2 salida error de FFmpeg

4.2.2 Archivos batch

La solución al problema estaba muy cerca, pero se debía obtener una manera de ejecutar en java *ffmpeg* desde línea de comandos y poder sacar la respuesta de la misma ante el vídeo a analizar, tratando de extraer la duración. Para ello se utilizó la programación de archivos *batch*. Este tipo de archivos ejecutan órdenes directamente a línea de comandos y creándolos y ejecutándolos desde Java se podía obtener la duración de un vídeo. Para llegar a la solución final no bastaba con realizar uno solo de esos archivos sino dos. El primero de ellos, realizaría la ejecución de *ffmpeg*, siendo el vídeo de entrada el vídeo deseado y forzando el error como hemos visto con anterioridad. Pero la salida de esta ejecución no es la información esperada ya que la línea de comandos discierne entre respuesta y error y siendo nuestra información un error con tan solo la ejecución del archivo *batch* no es suficiente. Para subsanar este pequeño

escollo, recurrimos a un segundo archivo *batch*. Este reclamará la actuación del primero y, haciendo de tubería, guardará la salida en dos archivos de texto, el primero, prescindible, con la salida que ya obteníamos antes y el segundo con la salida de error. Esta segunda será leída por Java, buscando línea a línea la duración y obteniendo el preciado resultado final.

4.2.3 Esquema final

Con estas herramientas se realizó un pequeño programa ejercicio en el que la clase que nos ocupa, y que formará parte del programa final con algún cambio, es ExtraerDuración [REF 9]. Se puede observar en el esquema de la *figura 4.3* cómo tan solo con recibir la dirección absoluta del vídeo del que deseamos la información este es capaz de extraer la duración. La propia clase es la encargada de crear los archivos, ejecutarlos y borrarlos al final y la línea de comandos permanece invisible durante el proceso, prácticamente inmediato, por lo que se trata de un proceso muy limpio a pesar de su complejidad.

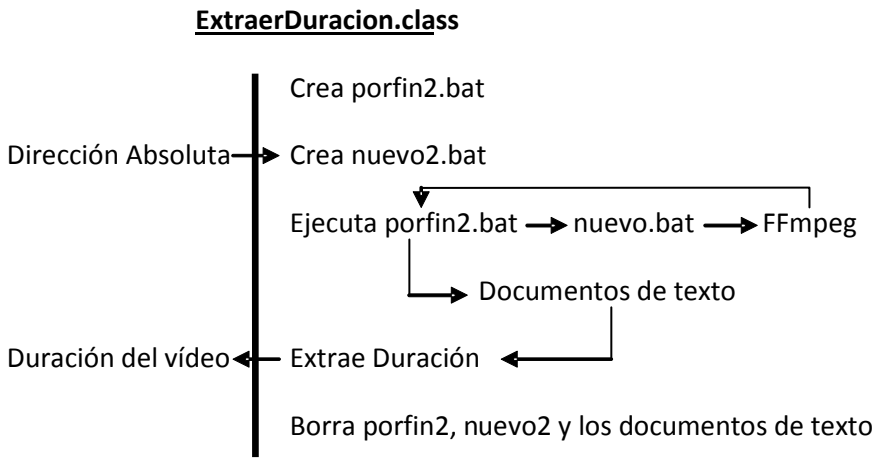


Figura 4.3, diagrama de extracción de Duración

4.2.4. Conclusiones

El proceso de extracción de la duración con el uso de *FFmpeg* y archivos *batch*, cumple con los objetivos requeridos, obteniéndolos en un proceso limpio que no deja huellas y suficientemente rápido para que no se haga molesto. Además tiene compatibilidad con otros formatos como *avi* o *mpeg* por lo que se trata de un sistema muy dinámico.

El programa final, por lo tanto, requiere tener *ffmpeg* para poder realizar su tarea, pero este programa es un ejecutable que no requiere instalación, tan solo deberá saber dónde se encuentra para ejecutarlo.

El proceso quizá no sea tan limpio como es deseable, pero esta complejidad se queda de puertas adentro, es decir, el usuario no será consciente de ello por lo que resulta, a fin de cuentas, un proceso limpio.

4.3 Investigación previa

El enrevesamiento de la solución adoptada para obtener la duración de los vídeos, hace entrever que no fue una respuesta directa, sino que responde a una investigación de cómo conseguirlo, durante la que se fueron analizando las distintas posibilidades para obtener el

resultado final. La mayor parte del tiempo de estudio se dedicó a discurrir cuáles podían ser esas fuentes a estudiar

Tenemos que tener claro que lo esencial es sacar la duración de un archivo *mpeg*, por lo que nos centramos en lograrlo y dejamos de lado por un momento la compatibilidad, objetivo en principio secundario. Partimos de la base de que es posible, ya que cualquier programa de reproducción multimedia es capaz de representar la duración un archivo de vídeo en el mismo instante en que este es abierto. Esto nos lleva a pensar que la información de duración se encuentra en la cabecera del archivo por lo que tratamos de de estudiar la cabecera de un archivo *Mpeg* para buscar el lugar donde se encuentre dicha información. *VISUALmpeg* [2] es un software que permite el análisis de cabeceras de un archivo *mpeg,* que fue utilizado para buscar dicha información. Con un primer vistazo queda claro que la duración no es una información que se encuentre en primer término, aunque el bitrate sí. La tasa de bits que podemos hallar en la cabecera principal del archivo podría ser el denominador de una división con el peso en bits del programa, con lo que hallaríamos los segundos de duración:

$$\frac{Bits}{\frac{Bits}{segundo}} = segundos$$

Esta primera posibilidad se descarta ya que las pruebas realizadas dictaminan que la precisión de la operación no es suficiente para nuestro objetivo.

En realidad la información de la duración se encuentra explicita en las cabeceras de un Archivo *mpeg*. Cada secuencia de dicho archivo tiene lo que se denomina un *GOP*, que es la cabecera de dicha secuencia. En cada una se establecen tres campos, correspondientes a horas, minutos y segundos del archivo al terminar dicha secuencia. Con ello, si buscamos la última secuencia podremos encontrar la duración total del vídeo. Aunque dicho así parece una solución limpia y sencilla, tiene dos problemas que no podemos salvar. El primero y más importante es que hay que encontrar la última secuencia, para lo que se tendría que ir analizando cada secuencia, haciendo que el proceso tarde demasiado. La segunda es que esto obligaría a un estudio diferenciado para cada tipo de archivo, condenando el programa a no tener compatibilidad de forma directa con ellos, objetivo secundario pero verdaderamente útil.

La última de las opciones que barajamos fue la de contar el número total de *frames* que forman el vídeo. Con esta información y los *frames* por segundo que presenta es fácil sacar la duración del vídeo, usando una fórmula similar a la que vimos con la tasa de bits, con la mayor precisión posible y es en este punto en el que por primera vez se comenzaron a hacer pruebas con *ffmpeg*, que devolvía el número de *frames* de los que consta el vídeo en cuestión, pero que como se ha visto en el punto anterior, tardaba demasiado.

5. Interfaz gráfica

El desarrollo de una interfaz gráfica potente compite en importancia con la manejabilidad y la fiabilidad de las operaciones internas que este 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.

Durante este capítulo se va a dar cuenta de las posibilidades barajadas para la creación de esta interfaz, en el primer apartado, explicando la motivación de la solución dada, en el segundo y la estructura seguida para su desarrollo, en el último.

5.1 Posibilidades barajadas

El desarrollo de bibliotecas para realizar distintos tipos de proyectos es uno de los puntos fuertes de Java. Por ello, la cantidad de librerías diferentes que permiten desarrollar una interfaz gráfica son prácticamente infinitas. Cuando se comenzó la búsqueda de la (o las) librería que iba a ser usada, se establecieron unos criterios de búsqueda. El primero de ellos se centraba el propio abanico de posibilidades que tenía dicha herramienta, es decir, que era capaz de hacer. El segundo, y no menos importante, tenía más que ver con la cantidad de información que se podía encontrar sobre ella, ya que el estudio de las posibilidades de una librería desde cero podía prorrogar la programación de la misma hasta un límite inaceptable.

La idea para la interfaz gráfica estaba basada en las agendas tipo *Google Calendar* [8]. El grueso de la interfaz debía constar de un calendario en el que poder viajar fácilmente entre fechas y una agenda de tipo "*time table*" en el que aparecieran los días de la semana y las horas, pudiendo ahí situar las instancias cargardas.

La búsqueda de ejemplos o aplicaciones parecidas a la idea que se tenía formó parte en gran medida del tiempo dedicado a la elección del medio para su generación. Lo cierto es que la cantidad de programas de ese estilo es muy escasa y en ninguno de los casos libre, por lo que enseguida se dictaminó la necesidad de su creación partiendo de una librería especializada en el desarrollo de interfaces gráficas.

Java FX [5] es una familia de productos de *Sun Mycrosistems* [6], 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 [6] para competir con *Flash* de *Adobe* [7].

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. En la figura 5.1 puede observarse un ejemplo da aplicación programada en Java, mostrando las posibilidades de JavaFX.

Una respuesta simple era mantener el desarrollo con AWT y Swing, las dos librerías principales con las que desarrollar interfaces gráficas en Java. No es que sus posibilidades fueran escasas, pero tras realizar un estudio de la posibilidad de hacer la interfaz mediante tablas se desechó esta posibilidad ya que no parecía ofrecer resultados muy llamativos.

Figura 5.1 Aplicación mediante JavaFX

Otras posibilidades como *Interfacsia* [13] fueron descartadas rápidamente por la gran falta de información que tenían. Con ello se continuó la búsqueda de una librería de manejo sencillo, buenos resultados y de la que se pudiera obtener suficiente información como para que la programación resultara más fácil.

5.2 SWT

SWT [10] es una librería para construir interfaces gráficas desarrollada por el proyecto Eclipse [9]. En ella las interfaces tendrán el mismo aspecto que las ventanas en sus sistemas operativos, además ofrece grandes posibilidades y cuenta con una documentación y tutoriales muy buenos. Todo indicaba una correcta adaptación de esta librería a los intereses que requiere la interfaz por lo que se comenzó a trabajar en la familiarización con esta herramienta.

Como se ha comentado se cuenta con gran cantidad de información para la familiarización con SWT, la mayor parte de ellos en internet. Dos de ellos sirvieron especialmente para solucionar los requerimientos del producto final y son los de *zetcode* [11] y *java2s* [12]. Estos no sólo sirvieron para poder aprender a funcionar con la librería, además lograron dar una mejor perspectiva de las posibilidades que daba la misma.

5.3 Desarrollo de la interfaz

La interfaz gráfica, como se ha comentado, debe constar de dos elementos fundamentales. El primero es un calendario y el segundo un horario en el que se representarán las instancias de los eventos programados.

Habiendo eliminado la posibilidad de realizar el horario mediante el uso de tablas, la posibilidad que barajamos fue crear el sistema de forma directa. Si lo que se necesitaba era una tabla en forma de horario, la mejor opción era la de generarla por medio de dibujos. Si en algo destaca SWT frente al resto de librerías estudiadas es en la facilidad y las posibilidades que alberga para pintar, así que mediante el dibujo de formas se comenzó a trabajar en el nuevo calendario.

Lo primero es el comenzar pintando formas sencillas, en este caso, un sencillo cuadrado de esquinas redondeadas

```
Color colorhorario1 = new Color(e.display, 208,208,191);
e.gc.setBackground(colorhorario1);
e.gc.fillRoundRectangle(30, 25, 210, 300, 15, 15);
colorhorario1.dispose();
```

REF 4

Esto se puede usar para hacer otras muchas formas, resultando la *figura 5.2* un ejemplo de varios tipos de figuras de programación muy simple.

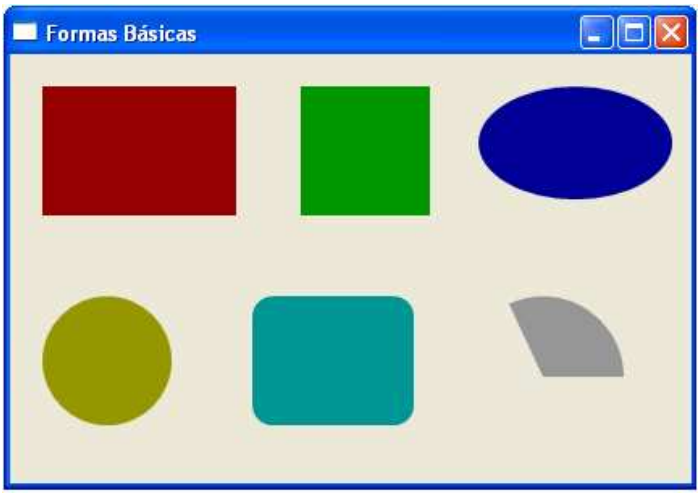


Figura 5.2 creación de formas con SWT

También contamos con otro sistema que permite pintar letras, es decir, escribir textos dando la posibilidad de asignar la fuente, el tamaño, si es negrita, itálica e incluso dotarlo de color. Con este sistema es posible escribir manteniendo el color del fondo que este establecido en este punto permitiéndonos hacer títulos y encabezamientos mucho más atractivos.

```
Font font2 = new Font(e.display,"Purisa", 8, SWT.BOLD);
e.gc.setFont(font2);
e.gc.drawText(i+":00", 175+100, 65+15+30*i);
font2.dispose();
```

REF 4

Con este sistema podemos crear figuras cada vez más complejas para ir acercándonos a la interfaz gráfica que queremos. En la *figura 5.3* se muestra un paso intermedio entre las figuras simples y la interfaz gráfica final.

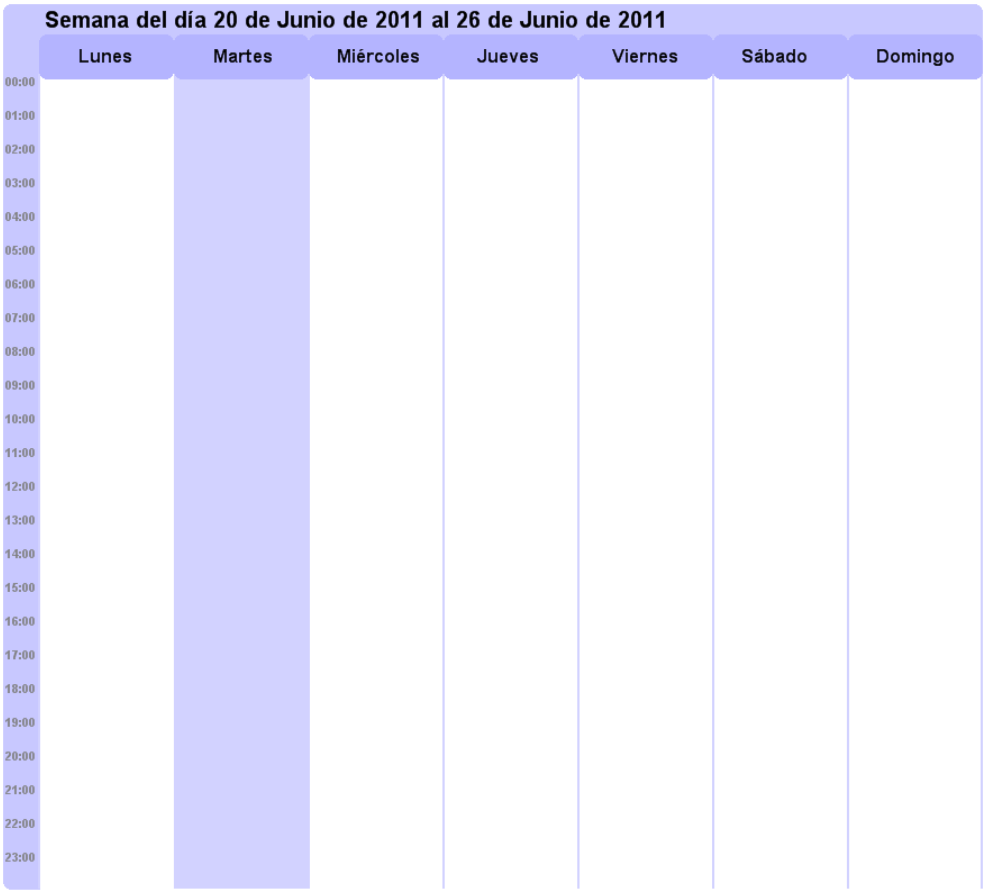


Figura 5.3, desarrollo del horario mediante SWT

Con estas herramientas, como se puede observar, es posible la creación de un horario visualmente atractivo y que cumpla con las necesidades que el programa tiene. Para cumplir con la parte fundamental de la interfaz gráfica tan solo queda el calendario. Uno de los objetos predefinidos de SWT no es otro que un calendario, que viene con tan solo instanciarlo, nos permite su uso, muy versátil y atractivo, como se puede observar en la figura 2.4.

```
dateTime = new DateTime(getShell(), SWT.CALENDAR);
```

REF 4

Figura 5.4 Calendario

5.4 Conclusiones

En este capítulo se ha presentado una discusión sobre la mejor forma de implementar la interfaz gráfica del programa en Java, culminando con el método que se utilizará para su creación.

La librería SWT da las herramientas necesarias para que la interfaz gráfica del programa sea lo suficientemente atractiva e intuitiva para cumplir los objetivos que se habían marcado al principio.

6. Programa final

A continuación, se realiza una explicación del funcionamiento interno del programa final. Para facilitar su comprensión, el capitulo se divide en dos partes principales, una perteneciente al propio funcionamiento interno y otra con la explicación de la programación de la interfaz gráfica. Es obvio que ambos están interconectados por lo que en ambas partes habrá referencias a la otra, pero estas serán superficiales, quedando mejor retratadas en su punto particular. Tras explicar los pormenores de la programación se realizará una explicación de la de la comunicación entre clases que se desarrolla en el programa, haciendo después hincapié en la problemática que conlleva la liberación de memoria en Java.

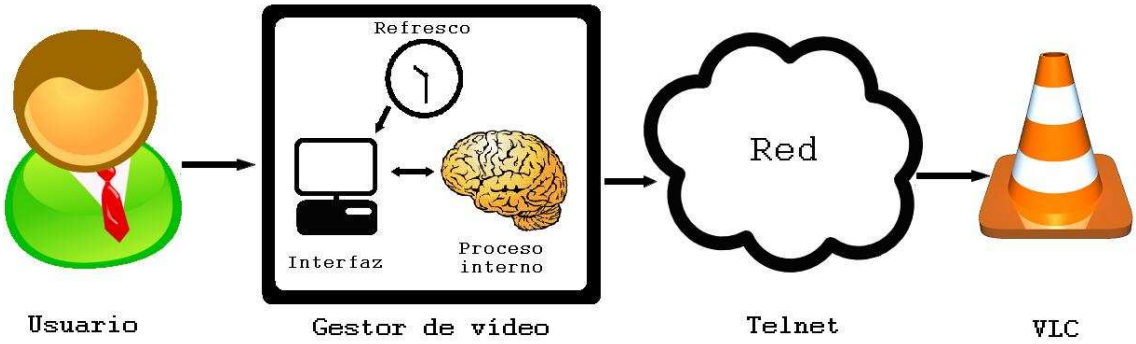


Diagrama 6.1 Modo de funcionamiento

El gestor de vídeo programado se divide en la parte de la interfaz y los procesos internos. Cuando el usuario realiza los cambios pertinentes, utilizando la interfaz, esta se pone en contacto con los procesos interno. El procesamiento culmina con la carga mediante *Telnet* de los vídeos programados y el proceso interno devuelve a la interfaz los cambios, produciendo su actualización. Además, un reloj interno renueva la interfaz periódicamente. Por tanto, el usuario puede hacer los cambios que desee y ve como estos son realizados, aunque todo el proceso sea oculto a su percepción.

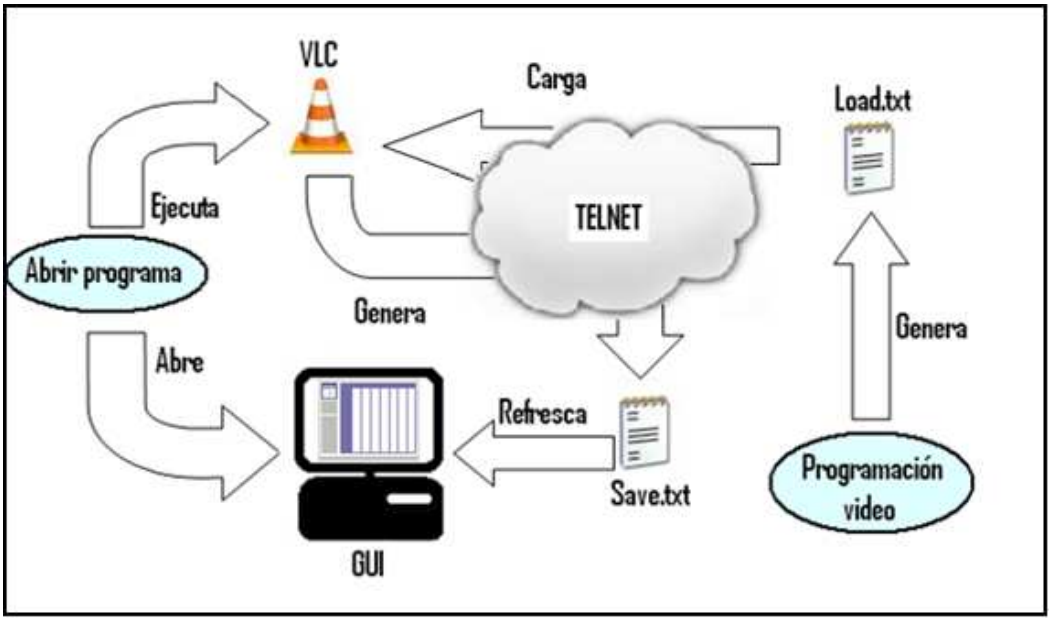


Diagrama 6.2 Esquema general

En el diagrama *6.2* puede observarse, de forma muy simplificada, como funciona el programa. Dividimos en dos acciones posibles. La primera, abrir programa, abre el programa con lo que se abre una nueva interfaz gráfica y se abre una instancia de VLC. Cuando un vídeo es programado, la segunda acción, genera un archivo (load.txt) con el código de programación que debe leer VLC. Utilizando la herramienta *Telnet* se cargará este en VLC, tras lo que se reclamará una renovación del archivo de programación (save.txt). Este contiene la información con toda la programación cargada en la instancia de VLC hasta el momento en que se carga. Finalmente, de este documento se saca la información que sirve para que se repinte la interfaz gráfica.

Aunque en realidad el funcionamiento es algo más complicado que lo explicado aquí, la filosofía de actuación del programa es bastante fiel a este esquema. Se puede observar cómo funcionan las tripas del programa, que realizan todas las funciones necesarias para conseguir generar el documento que después utilizará la interfaz gráfica para repintarse.

A continuación se expone un diagrama de flujo con el que se trata de explicar el funcionamiento de modo más detallado.

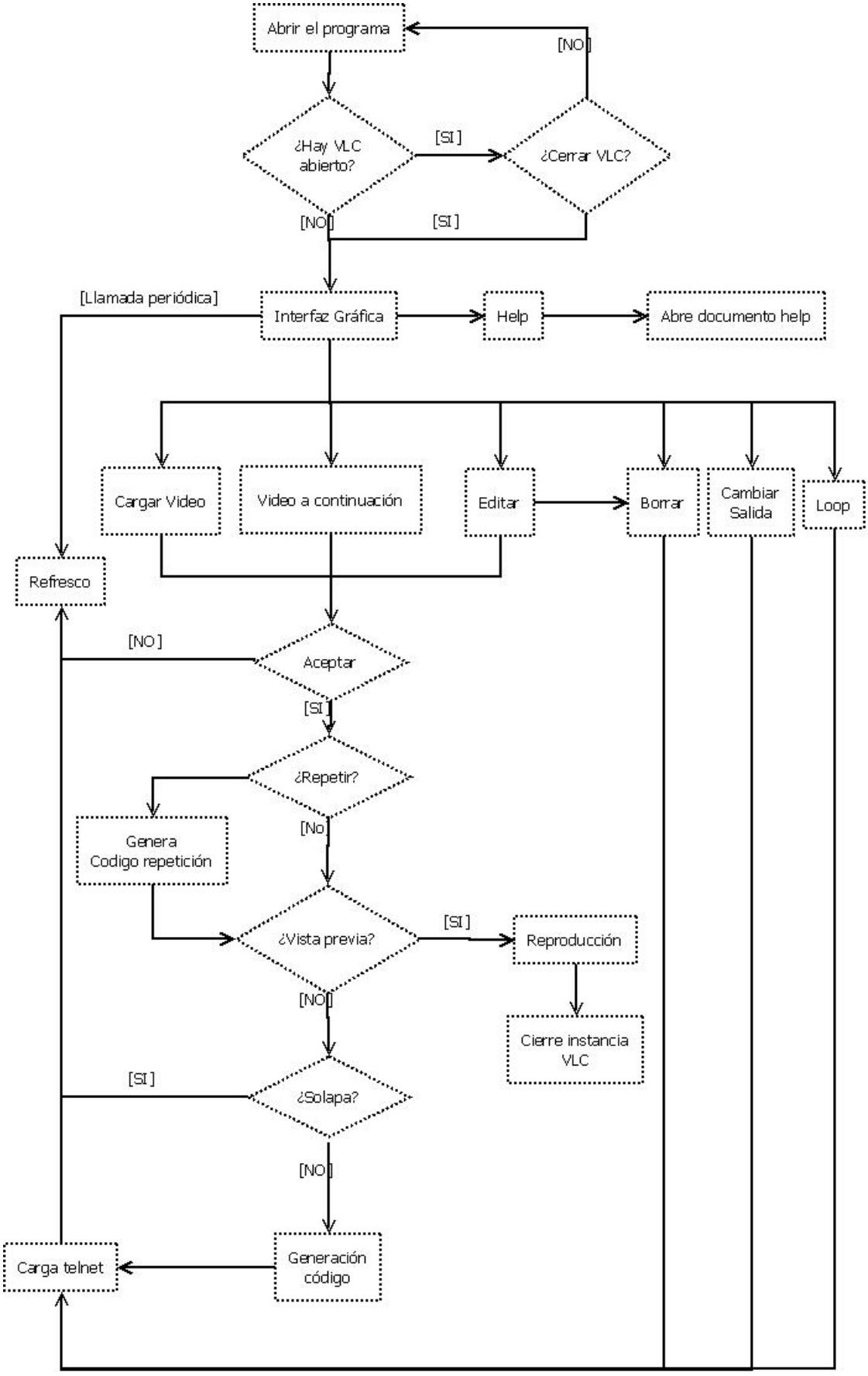


Diagrama 6.3 Diagrama de flujo del programa

6.1 Funcionamiento interno

Aquí se trata de explicar el funcionamiento interno del programa. Para simplificar la explicación, se dividirá el apartado en diferentes tareas que realiza, pero de nuevo nos encontramos con el problema de la interconexión entre todos ellos. Por ello según avance la explicación se pondrán referencias a tareas antes explicadas, tratando de evitar referenciar tareas de futuro detallado, para que tenga sentido.

6.1.1 Ejecución del programa

En el programa la ejecución no es inmediata, sino que debe de tener en cuenta ciertos aspectos que serán comentados en este punto. Aunque muchos de ellos permanecen ocultos al usuario, son de vital importancia para el correcto funcionamiento del mismo.

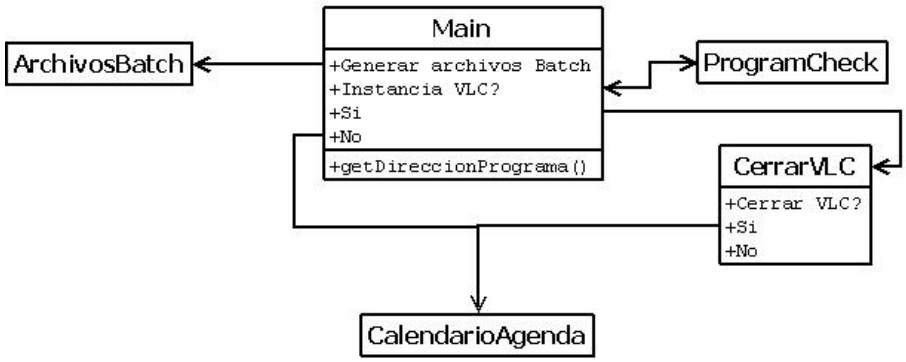


Diagrama 6.4 ejecución del programa

6.1.1.1 Direcciones relativas

El programa requiere el uso de direcciones relativas para poder ser usado en cualquier terminal sin necesidad de que el sistema esté organizado de la misma forma que lo está en el terminal que fue programado. EL uso de direcciones relativas, en lugar de absolutas, resuelve este problema, pero las órdenes que se dan directamente a línea de comandos, caso por ejemplo de la duración de los vídeos (*ver 4.2*), requieren que estas sean absolutas. Para ello, los archivos que vayan a ejecutarse se crearán o incluirán en una carpeta de recursos dentro de la carpeta del programa [REF 2]. Con este sistema, conociendo la ubicación exacta dónde está el programa, es sencillo conocer las direcciones. *Java* da esta información con una orden muy sencilla.

```
private static String direccionPrograma = System.getProperty("user.dir");
```

REF 4

6.1.1.2 Cierre de otras instancias

Como se ha visto en capítulos anteriores, el programa requiere tener una instancia de VLC [14] abierta con la opción de la interfaz *Telnet* activada. El problema viene cuando hay más de una instancia de VLC abierta, ya que al enviar la orden mediante *Telnet* [15] de que realice la carga del archivo de configuración VLM [16], este no sabe a cuál de las instancias debe cargarlo, por lo que suele solucionar el problema realizando esta carga de forma aparentemente aleatoria.

Este problema nos obliga a un doble requerimiento al ejecutar el programa. El primero es que debe haber una y solo una instancia de VLC abierta, el segundo, que esta tenga la interfaz *Telnet* activada. Para ello, la opción adoptada fue la de cerrar todas las instancias abiertas antes de la ejecución del programa y generar una nueva asegurando que este activada la opción *Telnet* en ella.

Para ello, se debe realizar una búsqueda de los procesos de VLM abiertos para cerrarlo. *Visual Basic* [17] es un lenguaje de programación orientado a objetos. El programa crea un archivo escrito en lenguaje *Visual Basic* que realiza una búsqueda de los procesos VLC instanciados. Recurrir a *Visual Basic* proporciona una herramienta que dinamiza el programa.

```
String vbs = "Set WshShell = WScript.CreateObject(\"WScript.Shell\")\n"
           + "Set locator = CreateObject(\"WbemScripting.SWbemLocator\")\n"
           + "Set service = locator.ConnectServer()\n"
           + "Set processes = service.ExecQuery _\n"
           + " (\"select * from Win32_Process where name='" + process +"'\")\n"
           + "For Each process in processes\n"
           + "wscript.echo process.Name \n"
           + "Next\n"
           + "Set WSHShell = Nothing\n";

fw.write(vbs);
fw.close();
```

REF 5

Una vez este está creado, se ejecutará, generando una salida en la que se pone en manifiesto la existencia o no de un proceso VLC. Ante este resultado, se dará a elegir la posibilidad de cerrar estas instancias y abrir el programa o por el contrario no abrirlo.

```
    //CHEQUEAMOS SI HAY UNA INSTANCIA ABIERTA DE VLC
    boolean result = pc.isRunning("vlc.exe");
    //SI ESTA ABIERTA SE ACCIONA VLC PARA CERRARLA
    //Y ABRIR UNA INSTANCIA CON EL TELNET ACTIVADO
    if (result == true){
        cv.setVisible(true);
        //Se abre la ventana para aceptar o no cerrar los procesos
        //abiertos de VLC
```

REF 4

Si no se acepta, no se abre el programa. Si se acepta, el programa crea y ejecuta un archivo *batch* para que se abra una nueva instancia con la interfaz *Telnet* activada, de modo que será esta interfaz la que esté comunicándose con el programa. Lo mismo ocurre si no hay más instancias VLC abiertas, el programa simplemente abre VLC de la manera que se ha descrito.

```
//ARCHIVO BATCH QUE LANZA EL VLC Y ACTIVA TELNET
String cmd;
  cmd = "\""+Main.getdireccionPrograma()+"\\recursos\\vlc telnet.bat"+"\"";
  try {
  Runtime r2 = Runtime.getRuntime();
  Process p2 = r2.exec(cmd);
```

REF 4

Con esto, se puede abrir el programa con la convicción de que sólo una instancia de VLC permanecerá abierta, y que esta tiene la interfaz *Telnet* activada. Como se ha observado, la ejecución de VLC requiere la ejecución de un archivo *batch* previamente generado en la carpeta de recursos del programa, por lo que el programa debe ser capaz de generarlo, como se verá a continuación.

6.1.1.3 Generación de archivos Batch

Ya ha sido comentada, con anterioridad, y se verá en capítulos posteriores, la necesidad en ciertos momentos del uso de archivos *Batch* para ejecutar ciertas órdenes en línea de comandos. Como se ha explicado, se requiere en ciertos momentos que *Java* los genere en su propia carpeta de recursos [REF 2]. Al dar comienzo al programa, automáticamente se generan los archivos necesarios en dicha carpeta con el mismo sistema usado previamente para generar el archivo *visual Basic* (ver 6.1.1.2).

```
String vlcFichero  = Main.getdireccionPrograma()+"\\recursos\\vlc telnet.bat";

 try {
     BufferedWriter bw2 = new BufferedWriter(new FileWriter(vlc));
     bw2.write("@ECHO OFF" +
             "\ncd \""+Main.getdireccionPrograma()+"\\recursos\\VLC\\\"" +
             "\nvlc --intf telnet" +
             "\nexit");
     bw2.close();
```

REF 6

Este esquema se repite hasta la creación de todos los archivos *Batch* que en adelante nos sean necesarios. En particular, en la clase de generación de *Batch (ArchivosBatch [REF 16])* se generan los archivos que requiera el programa, pero sólo aquellos que no dependen del vídeo que quiere ser generado. Los archivos generados aquí son los necesarios para la obtención de los números PID (ver 6.1.7) y el que lanza VLC con la interfaz *Telnet* activada.

6.1.2 El manejo de las fechas

Si lo que se quiere es realizar un programa de generación de escaletas para la programación de vídeos, el manejo de fechas y horas es una cuestión absolutamente necesaria. El programa debe saber tanto la fecha y la hora que es en cada momento, como saber moverse por las diferentes fechas en las que el usuario decida navegar para instanciar vídeos en el futuro. Para ello, nada más arrancar el programa, todo el contenido visual se centra en el día que se inicia.

```
FyHA.FechaActual();
año = Integer.parseInt(FyHA.getAñoActual());
mes = Integer.parseInt(FyHA.getMesActual());
dia = Integer.parseInt(FyHA.getDiaActual());
```

REF 7

Esta información es requerida a una clase que se dedica exclusivamente a dar la fecha y la hora del momento en que se pida. Esta clase se denomina *FechayHoraActual* [REF 8] e instancia un método que devuelve la fecha o la hora en que se lo pidan.

```
Calendar c = Calendar.getInstance();
añoActual=Integer.toString(c.get(Calendar.YEAR));
mesActual=Integer.toString(c.get(Calendar.MONTH));
diaActual=Integer.toString(c.get(Calendar.DATE));
```

REF 8

Las variables año, mes y día son las que el programa entiende como fecha a estudiar. Su inicialización con la fecha del momento en que comienza a trabajar el programa, responde a que el usuario no ha podido elegir la fecha en que quiere centrar su atención.

Como se verá en el capítulo de la interfaz gráfica (concretamente en el 6.2.2) el usuario puede seleccionar mediante el uso de un calendario la fecha que quiere revisar. Esta selección conlleva el cambio de las variables año, mes y día, con lo que se cambiara la fecha en la que se centra el estudio.

```
dateTime.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent e) {
        String fecha = dateTime+"";

        mes = Integer.parseInt(fecha.substring(fecha.indexOf("{")+1,
                fecha.indexOf("/")));
        año = Integer.parseInt(fecha.substring(fecha.indexOf("}")-4,
                fecha.indexOf("}")));
        dia = Integer.parseInt(fecha.substring(fecha.indexOf("/")+1,
                fecha.indexOf("}")-5));
```

REF 7

En el código anterior se muestra lo que ocurre cuando una fecha es seleccionada. La variable fecha contiene la fecha seleccionada con el uso del calendario y por comodidad para otros procesos, se desglosa en año, mes y día, con lo que cambiamos la fecha de estudio como ya se ha comentado. Cuando esto ocurre se repinta la interfaz gráfica, con lo que los videos instanciados en la semana de la fecha anterior, se cambian por los instanciados la semana de la nueva fecha elegida.

6.1.3 Cargar instancias

El programa desarrollado ofrece varias formas de crear una instancia de un video. A continuación se enumeran todas ellas.

- Forma estándar: en esta se debe de buscar el video en cuestión, poner el día y la hora que se desee y aceptar.
- Método rápido: determina que el día para instanciarlo es el que el usuario esta eligiendo en ese momento.
- Repeticiones: la clase Repetir crea una nueva instancia por cada repetición elegida (ver 6.1.8).
- Forma editar y crear vídeo a continuación: estas actúan del mismo modo que la forma estándar (ver 6.2.3).

Ahora interesa comprender el recorrido que sigue el vídeo desde que sus parámetros son elegidos hasta que se genera el archivo de texto que posteriormente se cargará mediante *Telnet* (6.1.13). Tan sólo se van a explicar aquí la programación de eventos simples, dejando otros un poco más complejos, como las repeticiones o el estudio del solapamiento, para los capítulos siguientes.

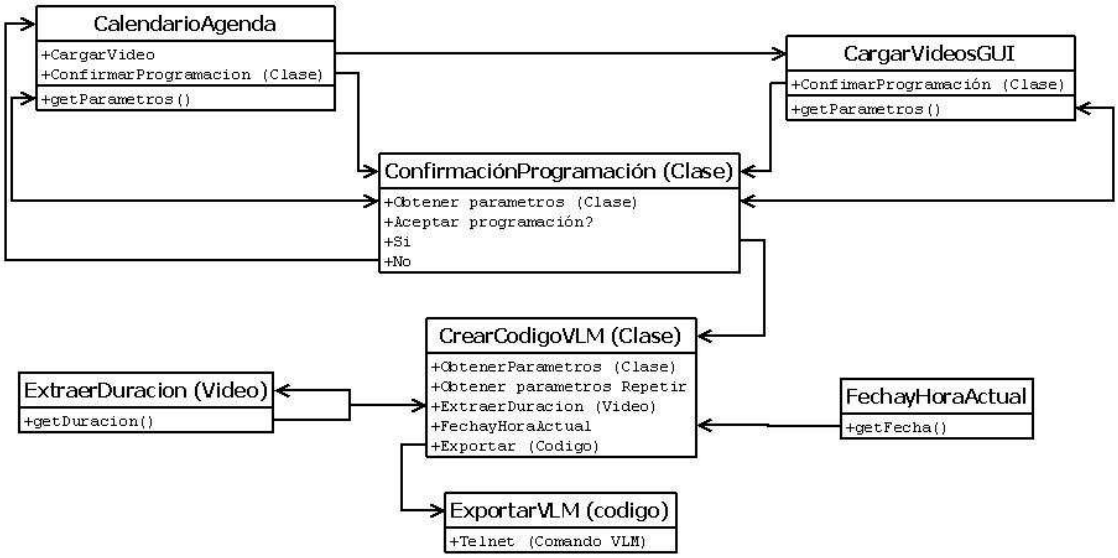


Diagrama 6.5 Cargando instancias

En el *diagrama 6.5* podemos observar las clases que participan en la creación del archivo de texto que se cargará en VLC y cómo la información va recorriéndolas. *CalendarioAgenda* y *CargarVideosGUI* son las que se ocupan de la recopilación de información para la programación del video a la hora de emisión. Desde estas clases se puede realizar la llamada para cargar el evento, que pasa por un filtro para su confirmación. Una vez confirmado, se crea el código y se exporta. En el proceso de creación del código, se analiza la duración del vídeo.

6.1.3.1 Duración del vídeo

Una vez se da por hecho que se han elegido los parámetros que formarán parte de la programación del vídeo, el paso inmediato es la extracción de su duración. Esto ocurre porque

se desea establecer un principio y un final para el evento en cuestión, y sin la duración esto no es posible. Conviene recordar que en el capítulo (4.2) se describe de forma detallada el método utilizado para la obtención de dicha información, si bien se describirá aquí brevemente en forma de resumen para que sea recordada.

Se comienza teniendo en cuenta la necesidad del uso del software *ffmpeg* [1], que se ejecuta forzándolo al error, obteniendo así un resumen de los parámetros del video. La consecución de estos parámetros requiere el uso de dos archivos *Batch*, uno que ejecute *ffmpeg* asignándole el análisis al vídeo que está siendo tratado y otro que ejecuta este archivo y envía la salida y el error a dos documentos de texto. El documento que contiene la salida de error será el utilizado para la extracción de la duración.

Lo primero a tener en cuenta es que el software *ffmpeg,* que recordemos no requiere instalación, debe estar situado en la carpeta del programa [REF 3]. Con ello, el programa deberá generar los archivos *Batch* necesarios para esta extracción. Como hemos visto con anterioridad, esto se realiza de forma sencilla. El programa genera los archivos adecuando la escritura al vídeo en cuestión.

```
try {
    BufferedWriter bw2 = new BufferedWriter(new FileWriter(sFichero));
    bw2.write("CALL "+"\""+Main.getdireccionPrograma()+"\\recursos\\"
            +"nuevo2.bat\"> "+"\""+Main.getdireccionPrograma()+"\\recursos\\"
            +"salida.txt\" 2> "+"\""+Main.getdireccionPrograma()+"\\recursos\\"
            +"nuevasalida.txt\"\r\nexit");
    bw2.close();

}catch (IOException ioe){
ioe.printStackTrace();
}
try {
    BufferedWriter bw3 = new BufferedWriter(new FileWriter(mFichero));
    bw3.write("cd \""+Main.getdireccionPrograma()+
            "\\ffmpeg\"\r\n\r\nffmpeg.exe -i "+"\""+Archivo+"\"");
    bw3.close();

}catch (IOException ioe){
ioe.printStackTrace();
}
```

REF 9

Una vez creados, ejecuta el archivo Batch encargado de analizar el vídeo seleccionado. Como hemos visto, este se ejecuta, haciendo su llamada al otro y generando los archivos de texto de los que después se extraerá la duración.

```
p=r.exec("cmd /C \""+Main.getdireccionPrograma()+"\\recursos\\"
        +"porfin2.bat"+"\"");
```

REF 9

Los archivos de texto han sido generados, una vez ejecutado el archivo, en la misma carpeta de recursos. Ahora extraeremos la duración. Para ello, se instancia el archivo del que se recogerán los datos y realizamos una lectura del mismo, por medio de un *BufferedReader* [18]. El BufferedReader es una clase que permite la lectura de un archivo de texto dado, línea a línea. Esto sirve para buscar en cada línea la palabra *Duration,* hasta que el archivo de texto no tenga

más líneas, momento en el que el bucle terminará. Cuando esta palabra se encuentra, el programa se queda con los valores de la duración, que más tarde serán reclamados para la creación del código.

```
while((linea=br.readLine())!=null)
    //busqueda de la palabra Duration
    if(linea.indexOf("Duration")!=-1){
        duracion=linea.substring(linea.indexOf("Duration")+10,
                linea.indexOf("Duration")+21);
        durHora= duracion.substring(0,2);
        durMin= duracion.substring(3,5);
        durSeg= duracion.substring(6,10);

    }
```

REF 9

Una vez hecho esto, sólo queda realizar un borrado de los archivos creados para garantizar la limpieza del sistema.

```
}finally{
    try{
        if( null != fr ){
            fr.close();
        }
        }catch (Exception e2){
            e2.printStackTrace();
        }if(generadorBat.exists()){
            generadorBat.delete();
        }if(llamaffmpeg.exists()){
            llamaffmpeg.delete();
        }if(archivo.exists()){
            archivo.delete();
        }if(salida.exists()){
            salida.delete();
        }
    }
```

REF 9

Asegurada ya esta limpieza, la duración ha sido extraída con éxito y se puede proceder a la generación del código propiamente dicha.

6.1.3.2 Generación del código de programación

La generación del código apenas ha cambiado respecto al del primer generador de código (ver 2.4). Una vez se tienen los parámetros de programación del vídeo en cuestión, solo hace falta traducirlo todo al lenguaje que VLC entiende.

Se escribirán tres parámetros en lo referente para cada vídeo. El primero para instanciar el vídeo y su salida, el segundo para programar la hora a la que comienza y el tercero para pararlo al final de su reproducción.

```
new 20110619190247 broadcast disabled loop
setup 20110619190247 input "C:\test.avi"
setup 20110619190247 output #display

new prog20110619190230 schedule date 2011/6/19-19:2:36 enabled
setup prog20110619190230  append control 20110619190230 play

new fin20110619190230 schedule date 2011/6/19-19:2:53 enabled
setup fin20110619190230 append control 20110619190230 stop
```

Cuando se crean las instancias hay que tener cierto cuidado con los nombres que se les pone. Si el usuario desea hacer una nueva instancia con un nombre igual al de otra que ya está cargada, este proceso fallará y no se realizará la carga correctamente. Para evitar este error, el nombre de las instancias es autogenerado por el programa y este les da a las instancias como nombre la fecha y la hora exacta del momento en que es programado. Esto logra que jamás exista la posibilidad de que se repitan dos nombres. Para esto, realiza una llamada a *FechayHoraActual*.

```
FyHA.FechaActual();
nombreInstancia = FyHA.getDiaActual()+FyHA.getMesActual()+
FyHA.getAñoActual()+FyHA.getHoraActual()+FyHA.getMinutoActual()+
FyHA.getSegundoActual();
```

REF 10

Una vez ya está elegido el nombre, solo queda escribir el código. Este paso es muy simple y se realiza de la misma forma que en el primer programa generador de código (ver 2.4).

```
FyHA.FechaActual();
nombreInstancia = FyHA.getDiaActual()+FyHA.getMesActual()+
FyHA.getAñoActual()+FyHA.getHoraActual()+FyHA.getMinutoActual()+
FyHA.getSegundoActual();

instancia = "new "+nombreInstancia+" broadcast enabled\r\n"
+"setup "+nombreInstancia+" input \""+filePath+"\"\r\n"
+"setup "+nombreInstancia+" output "+SS.getPrametros()+"\r\n\r\n";

schedule = "new prog"+nombreInstancia+" schedule date "
           +fechaInicio+"-"+horaInicio+" enabled\r\n"
+"setup prog"+nombreInstancia+" append control "+"Loop stop\r\n"
+"setup prog"+nombreInstancia+" append control "+nombreInstancia+" play\r\n";

finSchedule = "new fin"+nombreInstancia+" schedule date "+
           c.get(Calendar.YEAR)+"/"+(c.get(Calendar.MONTH)+1)+
           "/"+c.get(Calendar.DATE)+"-"+c.get(Calendar.HOUR_OF_DAY)+":"+
           c.get(Calendar.MINUTE)+":"+c.get(Calendar.SECOND)+" enabled\r\n"
+"setup fin"+nombreInstancia+" append control "+nombreInstancia+" stop\r\n"
+"setup fin"+nombreInstancia+" append control "+"Loop play\r\n";
```

REF 10

6.1.3.3 Generación de archivo de texto

Una vez el código ha sido escrito, se deberá generar el documento de texto que después VLM carga. Esta operación es sencilla, como ya pudo verse en el apartado *2.4.* El archivo se creará en la carpeta de recursos del programa.

```
String sFichero =Main.getdireccionPrograma()+"\\recursos\\"+"PRUEBA.txt"
File fichero = new File(sFichero);
  if(fichero.exists()){
      fichero.delete();
  }
try{
    BufferedWriter bw= new BufferedWriter(new FileWriter(sFichero));
    bw.write(Codigo);
    bw.close();
```

REF 11

6.1.4 Generación del save.txt

Debe de existir un método de control sobre los elementos instanciados. Sería un error pintar como programados todos los eventos que son programados por el usuario pues, en caso de error, no quedaría constancia del mismo, dando a pensar al usuario de que todo se reproducirá de la forma correcta.

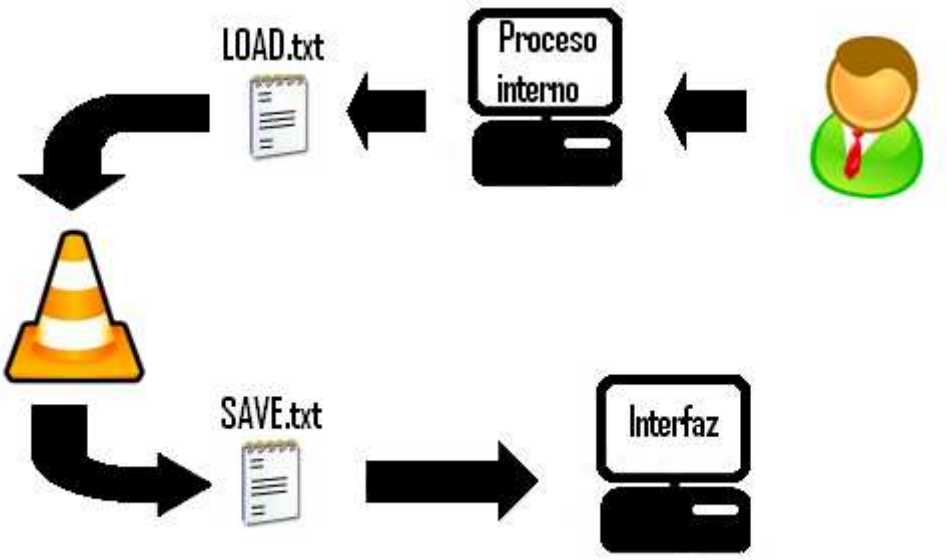


Diagrama 6.6 Generación del archivo de control

Esto se logra por medio de un archivo al que llamaremos save.txt. Cuando el usuario carga la programación, los procesos internos logran generar el archivo que *Telnet* cargará en VLC. Como ya se ha explicado en el apartado 3.2.4, la interfaz *Telnet* cuenta con varios comandos VLM muy prácticos. Una de ellos, el comando *save*, genera un archivo de texto con el código VLM que VLC tiene programado. Por tanto, se generará un archivo con la programación que se tiene después de la nueva carga. En caso de que haya existido algún error durante la carga del archivo, la instancia no será programada y VLC no la representará en el archivo de texto. Como

basamos lo representado en la interfaz en este archivo, el usuario será consciente de que no se reproducirá.

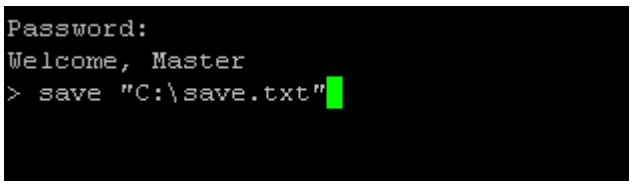


Figura 6.7 Generación del documento en telnet

Cada vez que el usuario cambie algún aspecto de la programación, un nuevo archivo será creado, por lo podremos mantener la interfaz gráfica siempre actualizada.

6.1.5 Extracción de las instancias

Cuando el archivo de texto con la programación ha sido generado, el programa debe leerlo para dar cuenta de las instancias programadas. Se procesará el archivo, leyendo su contenido y creando una matriz con los datos de los vídeos.

6.1.5.1 Generación de la matriz de parámetros

La entrada a la clase *ExtraeEventos* [REF 19] es una fecha. Dada esta fecha, el programa realizará una lectura realizando una búsqueda de eventos programados esa fecha. Dado el nombre de estos elementos podemos extraer toda la información que precisemos para mostrársela al usuario. En concreto, vamos a extraer el nombre del vídeo, la fecha y hora de inicio, la fecha y hora de fin del vídeo y el nombre de la instancia.

```
while((linea=br.readLine())!=null){
    if(linea.indexOf(fechaBuscador)!=-1){
    nombreEventos = linea.substring(linea.indexOf("new ")+8,
            linea.indexOf(" schedule"));

    horaComienzo = linea.substring(linea.indexOf("-")+1,
            linea.indexOf(" enabled"));
```

REF 12

6.1.5.2 Orden de la matriz

La matriz se va creando conforme el programa va encontrando las diferentes instancias. Esto quiere decir que la matriz estará ordenada en la misma forma que los elementos han sido programados. Sin embargo, a la hora de realizar ciertas representaciones es una gran ventaja que los datos de la matriz sean transmitidos cronológicamente, luego deberemos ordenar la matriz una vez esta termine de estar generada.

Para ordenar cronológicamente la matriz de eventos del día seleccionado, se usa la librería `java.util.Collections`. Esta contiene distintos métodos para el control de listas, como escoger el elemento mayor u ordenar de menor a mayor. El comando *Collections.sort(Vector)* es el encargado de ordenar el vector deseado de menor a mayor. Hay que tener en cuenta que la ordenación que ejecuta el método *sort* es en base al valor de los caracteres *ASCII*.

Como se comenta anteriormente, la finalidad de este apartado es ordenar cronológicamente la matriz de parámetros obtenida de un día deseado. El método a seguir para este fin se basa en la ordenación de un vector. Para ello, se generan dos vectores (en este caso llamados *Sin* y *Lista*) donde se introducirá el inicio de emisión de todos los vídeos programados el día seleccionado. Después, se ordena el vector *Sin*. Hay que tener en cuenta que mediante la ordenación de tiempos mediante sus caracteres ASCII el orden logrado es el deseado.

Lista [] = (21:30, 12:00, 00:00, 04:00)

Sin [] = (00:00, 04:00, 12:00, 21:30)

Con estos dos vectores se consigue un nuevo vector (en este caso llamado *orden*) donde se almacenan las posiciones de los elementos del vector sin ordenar en base al vector ordenado. Es decir, primero se obtienen los elementos del vector ordenado uno a uno, se compara con los elementos del vector no ordenado y se almacena su posición.

orden [] = (3, 4, 2, 1) = (Lista[3], Lista[4], Lista[2], Lista[1])

Una vez logrado este vector sólo, falta aplicar el orden regido por este a toda la matriz de eventos.

```
Collections.sort(Sin);
if (Sin.size()!= 0){
}

for(int q = 0;q<Sin.size();q++){
    orden.add(Lista.indexOf(Sin.elementAt(q)));

}
matrizEventosOrdenados = new String[getnumeroDeEventos()-1][9];
int cont = 0;
for (int p=0;p<orden.size();p++){
    cont = (Integer) orden.elementAt(p);

    matrizEventosOrdenados[p][0]= matrizEventos[cont][0];
    matrizEventosOrdenados[p][1]= matrizEventos[cont][1];
    matrizEventosOrdenados[p][2]= matrizEventos[cont][2];
    matrizEventosOrdenados[p][3]= matrizEventos[cont][3];
    matrizEventosOrdenados[p][4]= matrizEventos[cont][4];
    matrizEventosOrdenados[p][5]= matrizEventos[cont][5];
    matrizEventosOrdenados[p][6]= matrizEventos[cont][6];
    matrizEventosOrdenados[p][7]= matrizEventos[cont][7];
    if((cont%2)!=0){
        matrizEventosOrdenados[cont][8]=0+"";

    }else{

        matrizEventosOrdenados[cont][8]=1+"";
    }

}
```

REF 12

6.1.6 Extracción de fecha de los días de la semana

Como se observará en el apartado sobre la interfaz gráfica, la agenda representa los eventos producidos durante una semana. Se ha observado que la clase que se dedica a extraer los

eventos del archivo de texto que contiene la programación, requiere la información de la fecha de la que se desea la información. Cuando se selecciona una fecha, todos los eventos de la semana han de ser representados, luego se requiere esta información.

```
c.set(CalendarioAgenda.getAño(),CalendarioAgenda.getMes(),
        CalendarioAgenda.getDia());

diaSemana=c.get(Calendar.DAY_OF_WEEK);
dial = diaSemana;
if(diaSemana == 1){
    c.add(Calendar.DATE, -6);
    LunesDia=c.get(Calendar.DATE);
    LunesAño=c.get(Calendar.YEAR);
    LunesMes=c.get(Calendar.MONTH);
}else {
    c.add(Calendar.DATE, -diaSemana+2);
    LunesDia=c.get(Calendar.DATE);
    LunesAño=c.get(Calendar.YEAR);
    LunesMes=c.get(Calendar.MONTH);

}
```

REF 12

En la clase calendar, el método *Day of week* te devuelve un entero para señalar el día de la semana. La semana en este calendario comienza el domingo y acaba el sábado. Esto significa que deberemos discernir, para el cálculo de la fecha del lunes, entre cuando el día señalado es domingo y el resto. Una vez se calcula la fecha completa del lunes, es inmediato sacar las fechas del resto de la semana.

6.1.7 Vista previa

Conviene en muchos de los casos tener la opción de hacer un visionado de aquello que posteriormente se va a instanciar, por lo que el programa cuenta con la opción.

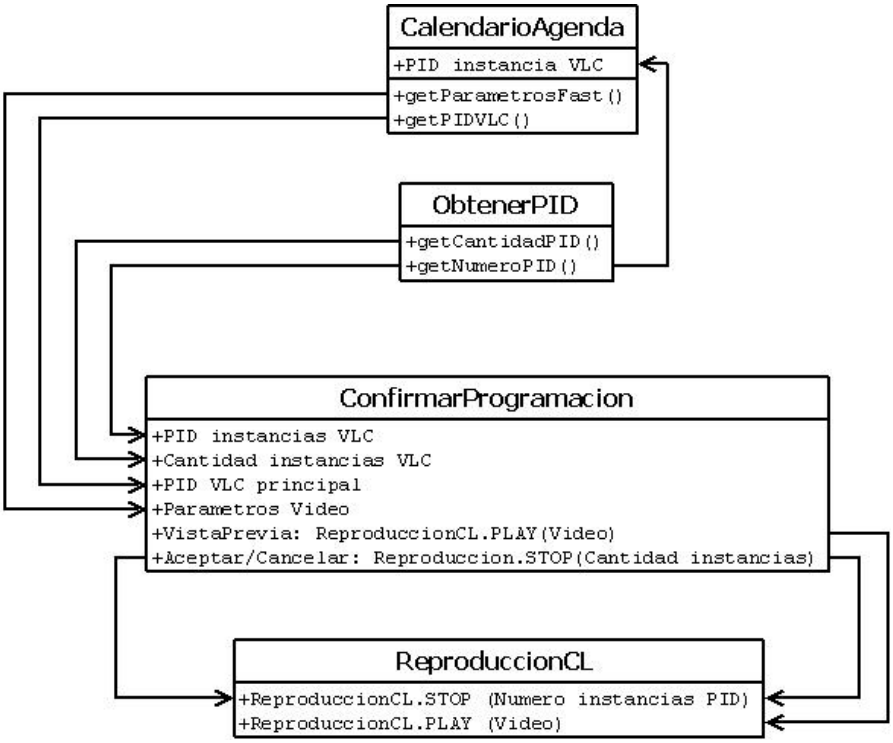


Diagrama 6.8 Vista previa

Hay que recordar que el programa no permite que dos instancias VLC estén abiertas, lo que complica la posibilidad de realizar la vista previa. Cuando una instancia de VLC es abierta, a esta se le asigna un número de identificación para diferenciarlo del resto de procesos del mismo tipo que se esté dando en el momento. Este número se denomina número PID.

Este número PID se encuentra ejecutando una sencilla orden en la línea de comandos.

Figura 6.9 Búsqueda de procesos en línea de comandos

Esta orden hace que salgan todos los procesos abiertos con sus correspondientes PID.

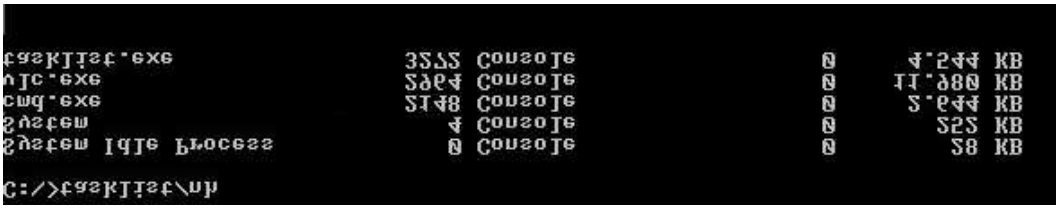


Figura 6.10 Lista de procesos

Como podemos ver en el apartado *6.1.1*, durante la ejecución del programa se fuerza a que no exista más de una instancia VLC abierta a la vez. Cuando esto se produce, el programa busca entre los procesos el número de identificación de esta instancia VLC.

```
private static String PIDVLC = ObtenerPID.getNumerosPID().
                substring(ObtenerPID.getNumerosPID().indexOf(" 1:")+
                        3,ObtenerPID.getNumerosPID().indexOf("."));
```

REF 4

En la ventana de confirmación de programación existe un botón que da la posibilidad de realizar el visionado del vídeo que se va a instanciar. Al pulsarlo, se genera un archivo batch (ver 4.2.2) que abre el vídeo en cuestión en una ventana de VLC. Para pararlo, tanto al aceptar o al cancelar en la ventana de confirmación de programación, se lanza un archivo batch que realiza la búsqueda de procesos VLC y elimina aquellos cuyo PID no coincida con el que tenemos guardado en la memoria, correspondiente al que hemos abierto junto al programa.

6.1.8 Repeticiones

En algunos casos, un mismo vídeo puede que se quiera repetir diariamente, semanalmente o con un periodo concreto de tiempo. El programa ofrece la posibilidad de realizar fácilmente este tipo de eventos.

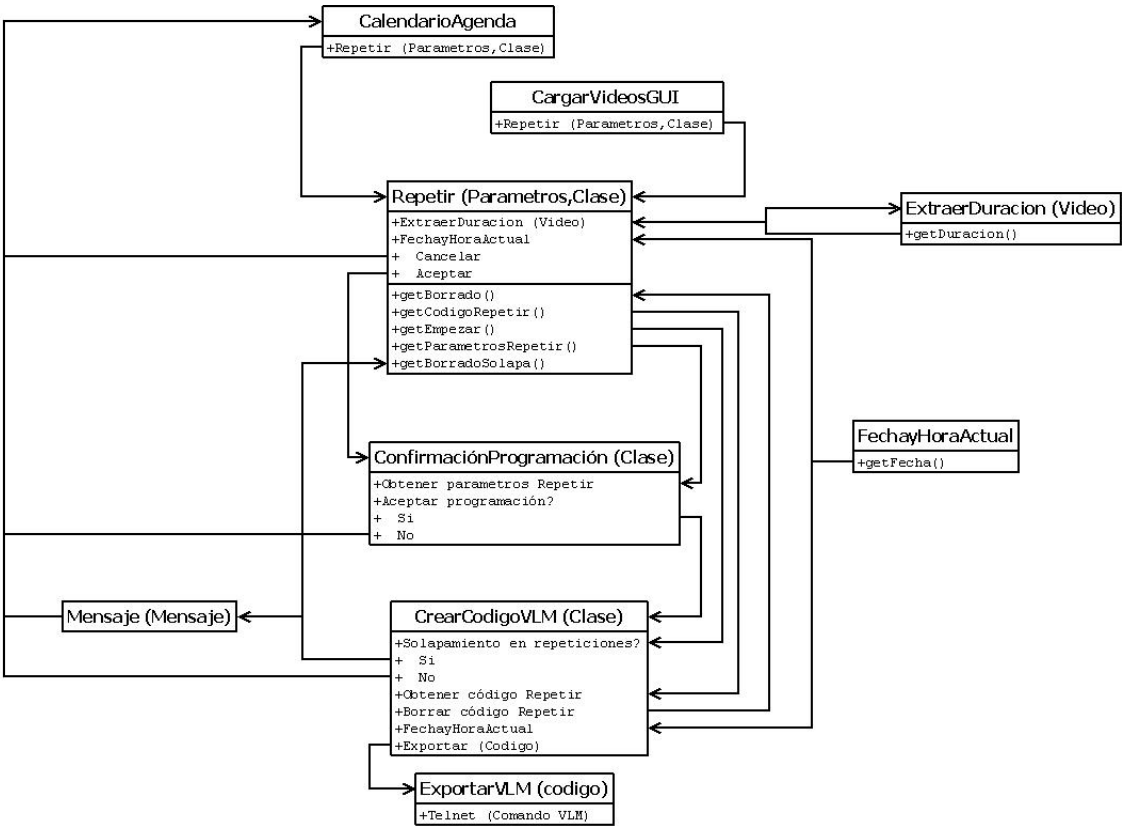


Diagrama 6.11 Repetir

Existen dos formas de instanciar repeticiones de un mismo vídeo. La primera, utilizando el parámetro *repeat,* que VLC comprende, generando una sola instancia con las repeticiones oportunas. Esto hace que las todas las repeticiones se comprendan como una sola instancia de modo que, por ejemplo, no se podrá particularizar el borrado de una de las repeticiones, sino que se borrarán todas. Además, sería un poco más complicada la búsqueda de las instancias de un día, teniendo en cuenta que se debería calcular cuándo se realizan las repeticiones para saber si coincide alguna con el día señalado. Esto se debe a que el vídeo se instanciaría con una sola fecha, señalando el número de repeticiones y el periodo de tiempo entre ellas.

Por todo ello, se opta por la segunda de las formas de instanciar repeticiones. Esta es generando una nueva instancia por repetición. De esta forma el tratamiento de cada tipo de repetición será personal y se podrán editar o borrar de forma singular. Además no se debe variar la forma en que el programa busca las instancias de cada día.

Si recordamos la generación del código de programación, a cada instancia se la daba el nombre de la fecha y la hora exacta del momento en que son programadas. Al generar instancias repetidas, tenemos el problema de instanciar en el mismo momento exacto todas ellas. Para solucionar este pequeño escollo, se le añade a la fecha el valor que tiene un contador que aumenta por cada instancia programada.

Cuando se genera el código del primer vídeo, se le da a la clase encargada de él el código de las repeticiones y esta los añade al archivo que luego cargará VLM como si se tratase de un grupo de instancias inconexas que se declaran al mismo tiempo.

6.1.9 Opciones de salida

Todo el programa realizado sirve como controlador del reproductor VLC. Este permite el envío de datos por *streaming* a otros terminales, de modo que puede utilizarse como servidor para desarrollar una televisión local. Este es precisamente el método que se utiliza en Aranguren Televisión para el envío de contenidos. Para ello hace falta seleccionar ciertas opciones además de determinar la dirección y el puerto al que quiere ser enviado. Esto quiere decir que en el programa se debe poder elegir esta información, por lo que se programó una clase encargada de cambiar las opciones de salida que tenía el programa.

Cuando se escribe el código que se cargará en VLC, este recoge la salida a la que se desea transmitir la instancia en cuestión. Por defecto, esta salida realiza una representación en el propio terminal en que se da la programación, pero puede ser cambiada. Se sobreentiende que cuando se desea cambiar esta salida, se desea que todos los elementos vayan en la misma dirección, por lo que también se han de cambiar las instancias que antes habían sido programadas. Para ello, se realiza una lectura del archivo que contiene toda la programación incluida hasta el momento, buscando todas las instancias que han sido realizadas. Cuando ya se conocen, mediante *Telnet* se cambian las salidas de todas las instancias.

```
while((linea=br.readLine())!=null)
    if(linea.indexOf("output")!=-1){
        longitud = linea.length();
       output = linea.substring(linea.indexOf("output ")+7,longitud);
```

REF 13

En caso de que cambiemos la salida, el programa debe cambiar esta. Para ello se realiza una búsqueda de todas las instancias programadas y usando *Telnet* se les asigna a todas las instancias la nueva salida.

6.1.10 Loop

En muchos casos, se requiere disponer de un grupo de anuncios que se reproduzca en el espacio entre vacío entre los vídeos instanciados. Estos anuncios vienen dados como un vídeo en bucle que se repite hasta que el próximo vídeo salga. Aunque el objetivo principal del programa sea servir de ayuda para la programación de elementos para esta televisión en particular, la filosofía es que pueda servir a diferentes usuarios. Por ello, el programa debe contemplar la posibilidad de que se quiera incluir este bucle de anuncios entre los vídeos que se reproducen, pero también la posibilidad de que no se desee.

Por ello se crea la ventana de *loop*. Esta ventana ofrece la posibilidad de hacer *loop* y seleccionar el vídeo que servirá de bucle. Para habilitar o deshabilitarlo existe un botón que se presenta como deseleccionado cuando el *loop* está deshabilitado y seleccionado cuando está habilitado.

Si se recuerda la generación del código (ver 6.1.3) se observa que para cada código se que se va a cargar en VLC se generaba en tres partes. La primera de ellas para generar la instancia, la segunda para el principio y la tercera para el fin. El truco que se utiliza para generar el *loop* consiste en que siempre se va instanciar el fin del *loop* cuando se genere el principio del vídeo y el principio del *loop* cuando se genere el final. Cuando el programa es creado lo primero que se crea es la instancia del *loop*, pero esta está deshabilitada y sin vídeo programado. Cuando el usuario decide seleccionar la opción para que se reproduzca el vídeo en bucle, se habilita el loop y se la añade el vídeo en cuestión.

6.1.11 Solapamiento

Uno de los grandes objetivos que tenía el programa era evitar los posibles errores humanos que se pueden dar en la actual forma de programación de elementos. Un error que se puede dar de forma muy simple es el solapamiento entre vídeos, es decir, que mientras un vídeo se esté reproduciendo, otro este programado para comenzar a reproducirse.

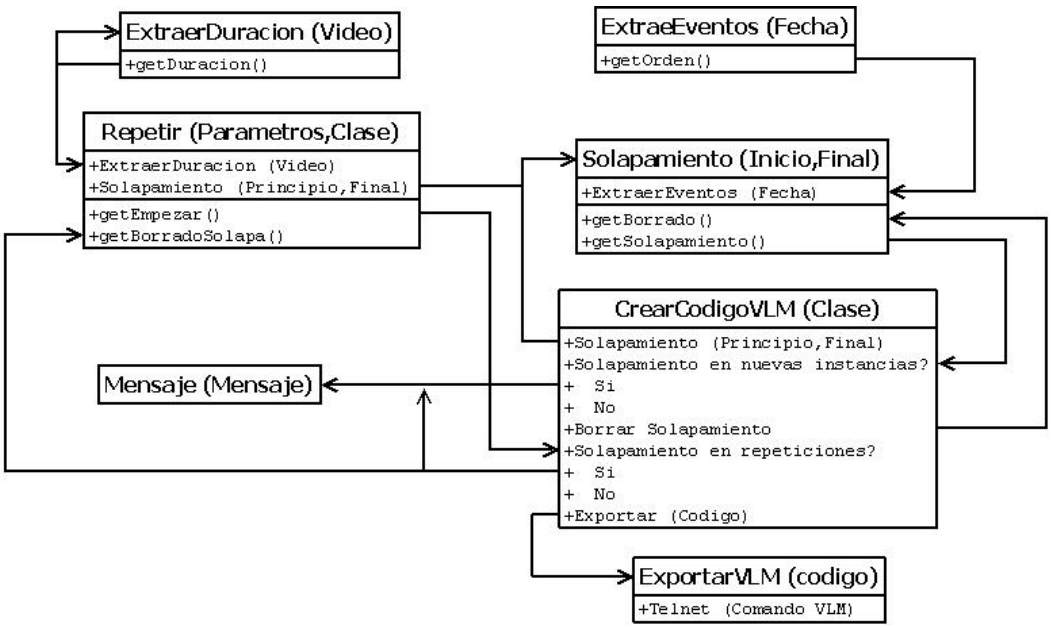


Diagrama 6.12 Solapamiento

Para evitar esto, se programó una clase de control antes de que el la instancia sea enviada a VLC. La idea es reducir el solapamiento a tres condiciones de forma que si no se cumple ninguna de ellas, se podrá decir que no existe solapamiento. Estas tres condiciones son las siguientes:

- El principio del vídeo esté entre el principio y el final de otro vídeo.
- El final del vídeo esté entre el principio y el final de otro vídeo
- El principio del vídeo esté antes que el principio del otro vídeo y el final del vídeo esté más tarde que el final del otro vídeo.

Para generar el código VLM y calcular el posible solapamiento es necesario conocer los tiempos de comienzo y fin del vídeo en cuestión. La clase *CrearCodigoVLM* [REF 10] es la encargada del cálculo del final del video y, por tanto, es la que obtiene la información necesaria para que la clase *Solapamiento* [REF 18] calcule el solapamiento.

Esta clase coteja el tiempo de comienzo y fin del vídeo con los tiempos de todas las instancias programadas el mismo día que este. Si alguna de las condiciones antes mencionadas se cumple habrá solapamiento y, después de un mensaje de aviso, esta instancia se eliminará. En caso contrario se podrá crear la instancia sin ningún tipo de problema.

6.1.12 Limpieza

Tal y como está concebido el programa, al cabo del tiempo el archivo de texto que contiene la programación cargada en VLC iría creciendo sin límite hasta tener demasiada información, haciendo más lento el proceso de generar una instancia. Para evitar esto, el programa va cotejando qué hora es. Cuando pasa la medianoche y comienza un nuevo día, son eliminados todos los elementos que terminaban el día concluido.

En el apartado 6.2.4 se explicará la razón de la existencia de un sistema multihilo. Un sistema multihilo permite que en un programa se realicen varias tareas de forma paralela. En el caso que nos ocupa, se realiza un hilo para el programa en general, y otro que sirve para llevar un control de la hora y que tiene forma de bucle. Este bucle se ejecuta cada dos minutos. Cuando este bucle se ejecuta, se comprueba si se ha cambiado de día, caso en el que se realiza la tarea de limpieza.

6.1.13 Telnet

Nada de lo mencionado anteriormente es posible si no se establece una comunicación entre programa y VLC. En el capítulo 3 se realiza un análisis de la forma en que se puede establecer esta conexión y de por qué *Telnet* es la mejor forma de hacerlo.

El programa debe abrir una conexión *Telnet* con VLC sin necesidad de utilizar ninguna aplicación externa. Para ello se realiza una clase que lo haga. La clase *Telnet* [REF 14] recibe de quien reclame su actuación la orden de lo que debe hacer. Esta establece la conexión, introduce la contraseña, realiza la función que se le reclama y cierra la conexión.

```
final String Proceso = new String(Codigo);
try {
    socket = new Socket(host, port);
    BufferedReader r = new BufferedReader(new InputStreamReader
            (socket.getInputStream()));
    BufferedWriter w = new BufferedWriter(new OutputStreamWriter
            (socket.getOutputStream()));

    readLines(r, 0);
    writeLine(w, password);
    readLines(r, 1);
    writeLine(w, Proceso);
    readLines(r,1);
    writeLine(w, LOGOUT);
    readLines(r, 1);
```

REF 14

6.2 Interfaz gráfica

Todo el entramado interno que requiere el programa debe convertirse en un lenguaje sencillo e intuitivo cuando se trata de la interfaz gráfica del programa. En este capítulo se va a explicar cómo esta es generada, haciendo referencia a elementos internos del programa que hacen posible la representación de la programación instanciada en VLC.

6.2.1 Agenda

La agenda es la parte de la interfaz donde se representan los eventos programados. Además es la parte de la interfaz más compleja, relegando a un segundo plano al resto de sus partes. Se podría decir que el resto del programa trabaja para el correcto funcionamiento de esta parte de la interfaz.

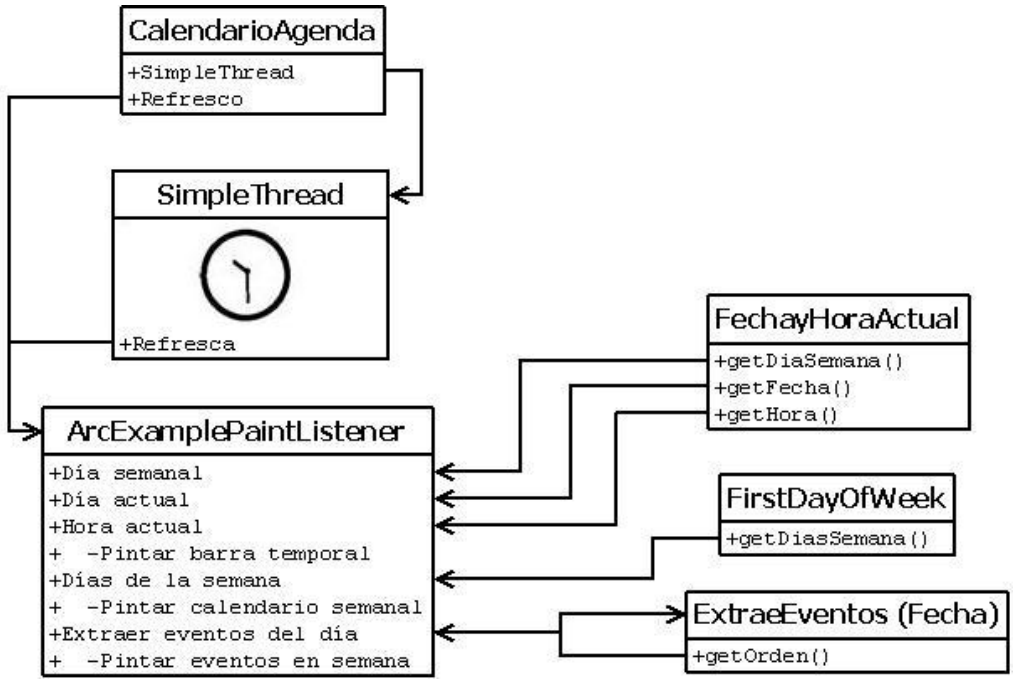


Diagrama 6.13 Interfaz gráfica

Como ya se ha dicho en el aparatado 4.2, la librería escogida para crear la interfaz gráfica es SWT. Gracias, sobretodo, a su método para pintar formas, se convierte en la librería ideal para la realización del trabajo que se está desempeñando. El método escogido para pintar la agenda se basa en el dibujo de formas, creando una estructura básica en la que después se sobrepondrán los vídeos programados, creando un efecto que permite conocer la programación de la semana dada con un simple vistazo.

ArcExamplePaintListener implementa un evento *PaintEvent* que permite el pintado de la interfaz. Este evento es utilizado por todos los métodos que completan el pintado de la interfaz. La clase tiene un controlador creado para servir como directriz del orden en que se ejecutan los diferentes pintados que componen el resultado final.

Cuando se realiza el refresco de la pantalla, esta clase es llamada de nuevo, realizando así una representación actualizada de los elementos. Este se refresca periódicamente además de cada vez que el archivo que contiene la programación sufre algún cambio.

```
private class ArcExamplePaintListener implements PaintListener {

        public void paintControl(PaintEvent e) {
        recuadroHorario(e);
        borde(e);
        horario(e);
        diasSemana(e);
        pintarEventos(e);
        lineasHoras(e);
        minutoAminuto(e);
        e.gc.dispose();
        System.gc();

    }
```

REF 7

6.2.1.1 Representación general

Para comenzar se debe crear la estructura. El método, como se ya se ha explicado consiste en dibujar elementos que, juntándolos permiten crear una estructura semejante al de una agenda semanal.

```
      Color colorhorario1 = new Color(e.display, 208,208,191);
      e.gc.setBackground(colorhorario1);
      e.gc.fillRoundRectangle(30, 25, 210, 300, 15, 15);
      colorhorario1.dispose();
```

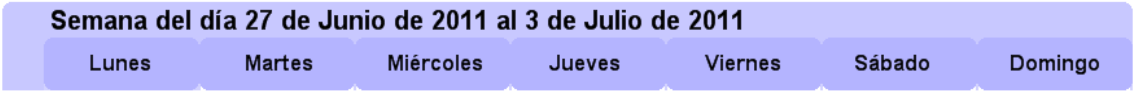
REF 7

Este método consigue la creación de cuadrados de esquinas redondeadas que será la pieza básica sobre la que se construirá el dibujo. Con esto y el método *drawText*, construimos la interfaz gráfica con los días de la semana y las horas escritos.

Debe de haber un indicador de la semana en la que estamos, por eso, *ArcExamplePaintListener*, la clase encargada de pintar la agenda, pide a *FirstDayOfWeek* la información sobre la fecha del lunes y del domingo y crea un texto que las indica.

```
e.gc.drawText(FirstDayOfWeek.getSemana(),210+100,5+15);
```

REF 7



Cuando se genera la ventana, el programa extrae los elementos de la semana entera para posteriormente pintarlos en la interfaz.

```
for(int n=0;n<7;n++){

    if(n==0){
        EE.Extraccion(FDW.getLunes());
    }if(n==1){
        EE.Extraccion(FDW.getMartes());
```

REF 4

La matriz que recoge de *ExtraerEventos* contiene la información de la hora de inicio y la hora de final y lo pasa a píxeles para saber desde donde hasta donde tiene que pintar el evento correspondiente. Para que no se pinten eventos seguidos del mismo color, lo que puede llevar a error por parte del usuario, se mete para cada vídeo un booleano que según su valor pintará el evento de un color u otro.

6.2.1.2 Representación de los eventos que empiezan un día y acaban otro

El problema llega cuando comienza un vídeo rondando la media noche y termina de madrugada. Esto conllevaría que la hora de final sea anterior a la hora de inicio por lo que tendríamos un error a la hora de pintar el evento.

Para evitar este error, cuando la hora de final es anterior a la hora de comienzo, el programa detecta que este evento terminará el día siguiente. Causa por la que pintará el evento que esté instanciado hasta el final del día, y en el día siguiente desde el principio hasta el final del vídeo.

```
if(finalin<principio){
    e.gc.setFont(font2);
    e.gc.fillRoundRectangle(205+100+n*120, principio+70+15, 120,
            720-principio, 30, 30);
    if(n!=6){
        e.gc.fillRoundRectangle(205+100+(n+1)*120, 70+15, 120,
                finalin, 30, 30);
```

REF 7

Esto tiene un nuevo contrapunto cuando es el domingo el día que empieza el vídeo que acaba el lunes. Como se puede imaginar no sirve con pintar el día siguiente el final, pues en dibujo de la agenda no hay día siguiente, por lo que pintaría sobre el fondo de la interfaz. Por ello, se realizó un método para que el programa sepa cuándo es domingo y cuando es lunes, y en el caso de que se produzca este error, el domingo pintará hasta el final y nada más y el lunes buscará eventos que terminen el lunes, para pintarlos también.

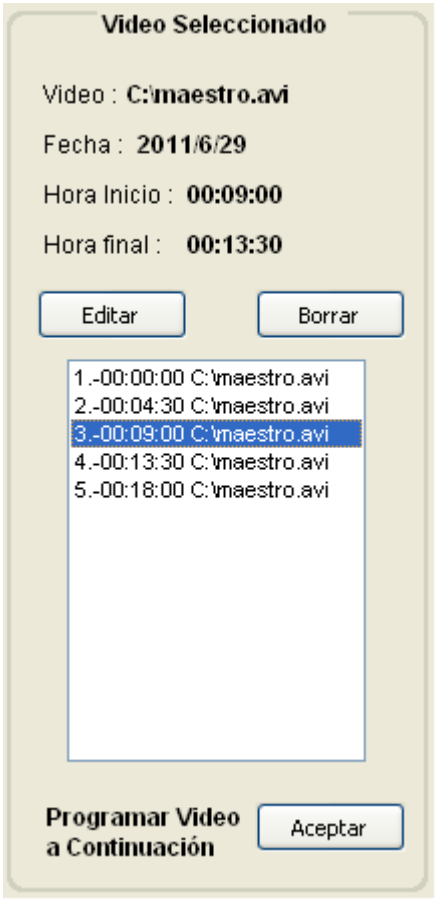
6.2.1.3 Eventos que duran poco tiempo

Existe un problema con los eventos que duran poco tiempo. Por causas lógicas la agenda tiene una extensión límite, por lo que los eventos muy cortos de tiempo no quedan representados de forma correcta.

Para solucionarlo el primer paso es limitar el tamaño mínimo de los vídeos. Esto quiere decir que cuando se presente un vídeo cuya duración real es menor de media hora, se representará en la interfaz gráfica como si durase media hora.

El tema del color de los eventos nos viene muy bien en este punto ya que ante la acumulación de elementos de corto periodo de tiempo se produce un cebrado que indica al usuario que hay una alta cantidad de elementos de cortos en un lugar de la interfaz en concreto.

Lo cierto es que este sistema no da información suficiente, pero ante la imposibilidad de mejorar el sistema visual, se incorpora una lista de los elementos ordenados cronológicamente para mayor información. Cuando se pincha uno de los elementos de la lista, se relata información sobre esa instancia en particular.



6.2.2 Calendario

Una de las grandes ventajas que ofrece la librería SWT para la creación de interfaces gráficas es que tiene implementados gran cantidad de objetos que hacen más fácil la labor del programador. En este caso, con una sencilla instrucción SWT genera un calendario interactivo para poder realizar las pertinentes instancias en los días que se desee.

```
dateTime = new DateTime(getShell(), SWT.CALENDAR);
```

REF 7

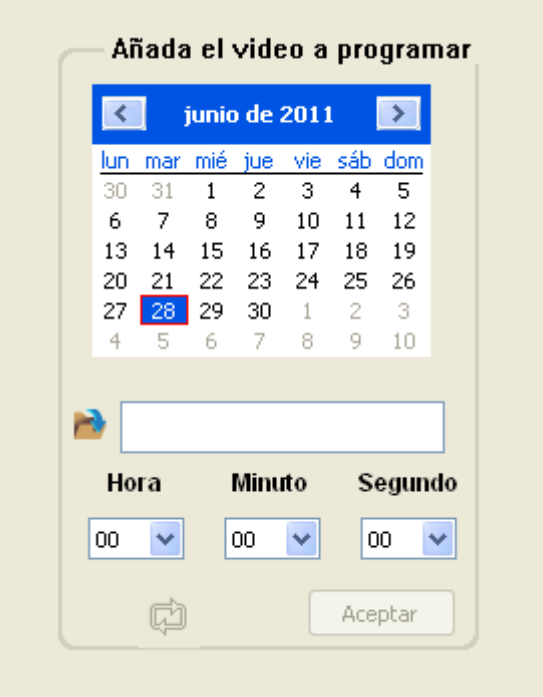
Esta sencilla orden genera el calendario que se va a utilizar en la interfaz gráfica del programa

Al pulsar cualquier fecha, el calendario te devuelve la fecha que has pulsado. La clase encargada de pintar escuchará está este cambio y viajará hasta la semana a la que pertenece. La variable referente a la fecha cambia al pulsar otra fecha diferente a la dada, por lo que se realiza un refresco en la clase encargada de pintar. Cuando se vuelve a pintar, los días de la semana habrán cambiado por lo que la agenda muestra lo que acontece los días alrededor del día elegido.

6.2.3 Formas de añadir un vídeo

Se ha diseñado el programa para que, cuando se desee realizar una acción, existan varios caminos. El programa posibilita distintos modos para la programación de vídeos, que es la acción principal. Dependiendo de la intención del usuario, a la hora de introducir nuevos videos, se ofrecen ciertos “atajos” para facilitar esta programación.

Dado que la tarea que más se repetirá es la de añadir videos, en la propia interfaz gráfica principal del programa existe una zona para realizar esta acción de modo sencillo y rápido.

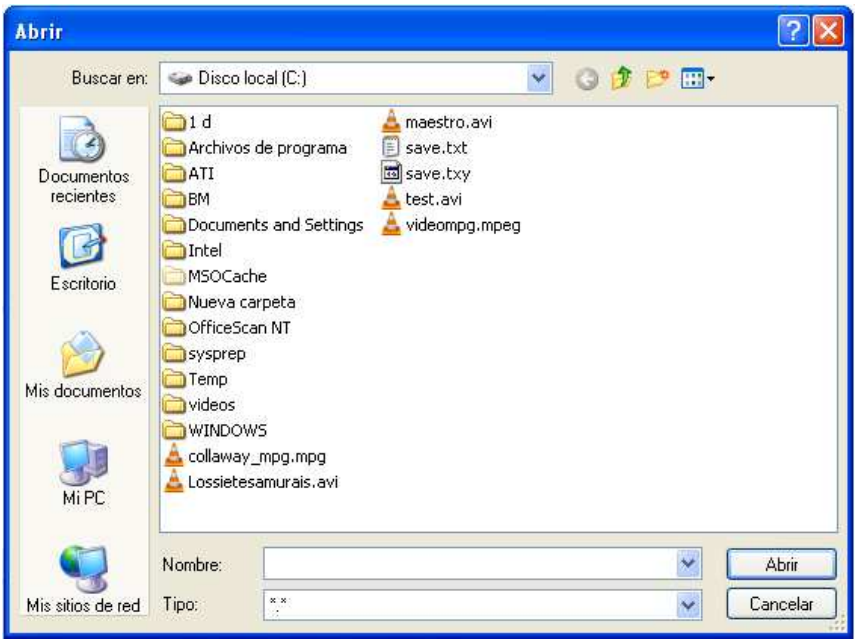


Otro de los objetos que SWT nos proporciona de forma sencilla es un explorador para seleccionar el vídeo que queramos programar. Este objeto es utilizado tanto en la interfaz principal contenida en la clase CalendarioAgenda, como en la interfaz para cargar vídeos CargarVideosGUI o en Loop.

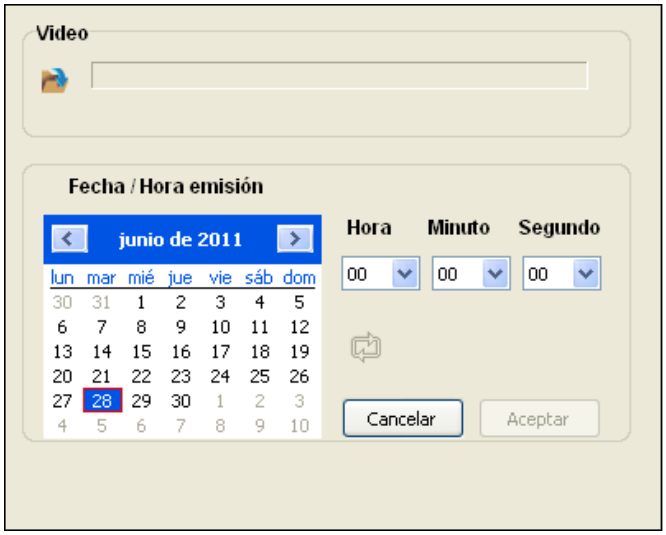
```
FileDialog buscador = new FileDialog(getShell(), SWT.OPEN);
```

REF 4

Con esta sencilla operación, al pulsar la imagen de la carpeta se abre un buscador.



Además de esta forma sencilla, se crea una interfaz gráfica para añadir vídeos. El funcionamiento básico es el mismo, pero se añade para comodidad del usuario.



Como ya hemos visto, existe una lista que representa los elementos seleccionados del día que está señalado en el calendario. Cuando se pulsa sobre uno de estos días, se accede a un resumen de los parámetros de dicho vídeo, pero además se puede realizar tres opciones, borrado, editado y añadir vídeo a continuación.

Adelantándose a la intención del usuario, el programa facilita dos modos de programación muy útiles. En muchas ocasiones, el usuario deseará editar el horario de emisión de un video en particular. Para ello, existe el botón *Editar.* La interfaz gráfica que se abre al elegir esta opción es la misma que la interfaz para abrir vídeos. La diferencia es que se deben inicializar los campos de fecha en el calendario, hora y vídeo con los valores establecidos en el vídeo original.

Otra de las intenciones del usuario sería la de poner un vídeo a continuación del seleccionado en la lista de videos programados. En este caso, los campos de la fecha y la hora deben ser las del final del vídeo escogido.

Todo esto significa que a la hora de crear la interfaz se debe discernir entre los tipos de programación posibles. Para ello se le envía un valor entero a la clase que desarrolla la interfaz gráfica. Si este es un 1, esta clase sabrá que se trata de una instancia normal y no establece los cambios. Si se trata de un 2, sabrá que lo que se desea es editar un vídeo ya programado, por lo cual primero reclama la información del vídeo original y en caso de aceptar, realizará el borrado mediante *Telnet* del elemento y realizará su nueva programación. En caso de ser un 3, el elemento se programará un vídeo a continuación por lo que se reclamará la información del vídeo original, usando para la programación los datos del fin del vídeo. El código a continuación pertenece al segundo caso.

```
CargarVideosGUI CV = new CargarVideosGUI(getDisplay(),2);
```

REF 4

Como se ha comentado, cuando se llama a la clase encargada de la creación de la interfaz, sabe que se trata de un editado.

```
public CargarVideosGUI(Display display2,int tipo) {
if(tipo==2){
    estamosEditando=true;
    tipoEditar();
}else if(tipo==3){
    tipoContinuacion();
}else {
    tipoGeneral();
```

REF 15

Como podemos observar esto nos lleva al método tipo editar, donde se establecen en la interfaz los datos pertenecientes al principio del vídeo.

El programa desarrollado crea nuevas instancias indirectamente para ahorrar trabajo al usuario, como a la hora de la introducción de repeticiones o selección del video bucle. Estas se explican posteriormente en los apartados 6.1.8 y 6.1.10.

6.2.4 Representación de la hora

Cuando el usuario observa la interfaz gráfica debe haber un elemento indicativo del día y la hora que es, para que este sepa inmediatamente en qué momento se encuentra la reproducción programada, cuánto tiempo queda para los siguientes eventos, cuáles son los eventos que se reproducirán a continuación, etc.

La mecánica para representar la hora es similar a la que se utiliza para representar los vídeos programados. Debe haber un mecanismo que sea capaz de establecer la hora actual y convertir esa hora a píxeles para su representación. Obviamente para recabar esa información realizamos una llamada a *FechayHoraActual*.

```
FyHA.FechaActual();

int distancia = Integer.parseInt(FyHA.getHoraActual())*30+
            Integer.parseInt(FyHA.getMinutoActual())/2;
```

REF 4

Esta programación suscita dos problemas. El primero es que se debe de saber situar en qué semana estamos situados, ya que la línea que marca la hora no debe de ser pintada otra semana que no sea la actual. Para ello, establecemos una variable estática que marca la fecha del lunes del día actual. Cuando se selecciona una fecha en nuestro calendario, se realiza una comprobación si la fecha del lunes de la semana del día escogido coincide con la variable estática antes establecida y de no ser así, la línea no se pintará.

```
if(comprobarLunes1.equals(comprobarLunes2) || comprobarLunes2 == null){
```

REF 4

Como se puede observar, también cabe la posibilidad de que no se establezca un valor para la segunda fecha, es decir, que no se haya seleccionado ninguna fecha en el calendario. Esto ocurre en la práctica únicamente en el momento en que se inicia el programa.

El segundo de los errores tiene que ver con el uso que se le dé al programa. Como hemos explicado anteriormente, el programa sólo se refresca cuando se cambia algo de la programación, con lo que si nada cambia la interfaz gráfica permanece estable por lo que la hora no avanzaría. Por tanto se debe establecer un modo de refrescar cada cierto tiempo esta interfaz independientemente del uso que se le dé.

Para ello se usa un sistema multihilo, es decir, se genera un nuevo hilo que correrá paralelamente al resto del programa y que cada dos minutos refrescará la pantalla, con lo que podremos tener una indicación siempre actualizada de la hora que es. Este hilo paralelo es creado en la clase *SimpleThread* [REF 7].

6.2.5 Liberación de la memoria

El programa generado tiene como una de sus más importantes necesidades la de permanecer abierto largos periodos de tiempo. Por eso se hace importante la realización de pruebas de estrés que dejen evidencia de que es lo suficientemente robusto para cumplir esta función. En este punto se trata de explicar cuál es el problema por lo que esto no es del todo posible.

La realización de las pruebas de estrés determinan que el programa tiene un tiempo de duración limitado y termina por fallar debido a la sobrecarga de memoria que requiere. Cuando se trata con Java, una de las grandes diferencias respecto a otros lenguajes como C++ es la manera en que la memoria es liberada. En C++ se debe instanciar el borrado de elementos para realizar dicha liberación de modo que los procesos que ya no se van a volver a usar quedan eliminados para que no se produzca un aumento progresivo de la memoria que acabe por colapsar el sistema.

En caso de Java, tienen un mecanismo que se dedica a realizar este tipo de borrado. Su nombre es *Garbage collector.* El beneficio y el problema de este sistema es el mismo. Por un lado realiza sin necesidad de programación el borrado de estas variables cuando lo cree necesario, facilitando la programación para el usuario de este lenguaje de programación. El problema es que si la memoria sigue creciendo es difícil saber dónde se encuentra el problema, ya que el *Garbage collector*, sólo borra lo que cree que debe y es difícil encontrar todos los puntos que se le escapan. Es en este punto donde se sitúa el problema del programa en cuestión. A pesar de que modificando el modo de instanciar el resto de las clases y tratando de que cada variable sea declarada nula cuando ha realizado su cometido ha conseguido que esta fuga de memoria sea frenada de un modo claro, no se ha conseguido pararlo de un modo absoluto, por lo que la memoria requerida para su funcionamiento sigue creciendo aunque de un modo lento. Esto conlleva que a la larga el programa falla siendo un punto a mejorar en el futuro.

7. Conclusiones y líneas futuras

Tras el desarrollo y la explicación del programa, se procede a realizar un proceso de reflexión sobre el programa y el camino que ha llevado hasta su realización. Esta operación se realizará tratando de dar un punto de vista lo más imparcial posible, ya que es la única manera de que sea provechoso.

Dividiremos estas en dos apartados, uno referente a las conclusiones de la realización del proyecto propiamente dichas y otro con posibles mejoras y líneas futuras para la mejora del mismo.

7.1 Conclusiones

Desde el punto de vista personal, conviene hacer una reflexión sobre si la realización del proyecto ha cumplido con las expectativas que motivaron su desarrollo. Cuando se comenzó a realizar, estas motivaciones eran algo etéreas, pues no se disponía de la información suficiente como para saber si el camino hasta la resolución final iba a generar los puntos positivos que previamente se le atribuían.

Cuando un estudiante comienza su vida laboral, al menos en el campo de los estudios que realiza, suele haber sido escasa. Este desconocimiento suele llevar a cierto temor ante la falta de un punto de referencia. Con el desarrollo del proyecto se ha logrado conseguir solucionar el problema real de una empresa de actual funcionamiento. Esto ha logrado que se sea consciente del valor de los conceptos aprendidos y de su utilidad en la vida laboral.

En alguna ocasión durante el periodo universitario, se ha comentado que el trabajo de un ingeniero es la resolución de problemas de la rama propia de sus estudios. Es obvio que en un periodo relativamente corto es imposible contar con un conocimiento detallado de todo el mundo que engloba la imagen y el sonido en el mundo de las telecomunicaciones. Esto deriva en que el ingeniero debe ser capaz de amoldar sus capacidades al problema que se le presente, aunque necesariamente no tenga un conocimiento amplio de este en particular. Desarrollar una interfaz para la gestión de la programación de una televisión local mediante un lenguaje a priori desconocido es un ejemplo de campo en el que no se tenía un conocimiento amplio. El haber conseguido finalmente superar los obstáculos nos hace conscientes de nuestra capacidad para ello.

Esta ha sido una de las tareas más complejas con las que ha habido enfrentamiento. Quizá no por la facilidad o complejidad del producto que se busca, sino por tener una libertad considerable para seleccionar la herramienta que lo haga posible. Esta libertad ha logrado concienciar sobre la importancia del análisis previo de la herramienta a utilizar que requiere la resolución de problemas. Si el problema de la liberación de la memoria en Java hubiese sido conocido, por ejemplo, los errores después surgidos quizás hubieran sido subsanados con cierta facilidad.

Desde el punto de vista técnico, la familiarización con un lenguaje de programación nuevo, como Java, ha sido muy positivo. Además de por el lenguaje en sí, por el descubrimiento de las amplias posibilidades de los lenguajes orientados a objetos.

Además, el uso de librerías externas supone un punto de aprendizaje que puede ser usado con otros muchos lenguajes de programación o entornos de desarrollo de los mismos.

Se ha logrado un programa dinámico que no se cierra a las limitaciones de un lenguaje de programación. En caso de ser necesario, se ha recurrido a otros, caso de *Visual Basic* y la programación de archivos *Batch*.

En cuanto a la experiencia con *Telnet*, supone una herramienta muy potente tanto para este tipo de aplicaciones, como para establecer dispositivos de control remoto.

A pesar del pensamiento general de que VLC no es más que un reproductor multimedia, este proyecto ha ayudado a darnos cuenta y a aprender a utilizar todo el potencial que este software contiene.

En conclusión, el trabajo realizado ha cumplido las expectativas personales que se marcaron al comienzo del mismo. Con la salvedad de la pérdida de memoria, también logra superar las necesidades técnicas que se tenían.

7.2 Líneas futuras

Aunque se cumplan los requisitos, el programa alberga una posibilidad de mejora clara. Durante la programación del mismo han ido surgiendo ideas para su desarrollo. Son estas ideas las que se exponen a continuación.

Existe, en la actualidad, la posibilidad de programar una instancia antes de la hora que es en el momento. Esta instancia es inservible y sólo acumula código que después afectará al tiempo de ejecución de las órdenes que se le den al programa. Este, por tanto, debería imposibilitar el instanciado de elementos antes de la hora que es en el momento de hacerlo.

Como hemos visto, existe la posibilidad de realizar una previsualización de los vídeos que se van a cargar. Esta se realiza abriendo una nueva instancia VLC. Sería más elegante que en un futuro esa visualización se realice en la misma ventana de confirmación donde se da lugar a la posibilidad de realizarla.

A pesar del intento de que el proceso de cerrado de instancias VLC que se previsualizan sea invisible, este continúa siendo visible, lo que da lugar a que resulte sucio. Este cierre debería ser invisible.

Al editar una instancia se realiza el borrado de la misma, tras lo que se instancia en el nuevo horario convenido. El problema es que si se detecta un solapamiento de esta nueva instancia, no se realizará, pero el anterior ya ha sido borrado. Debería, al menos, darse la opción de que no se borre la original cuando no se va a consolidar la nueva instancia.

La extracción de la duración es un proceso rápido, pero no instantáneo. Se debería trabajar en acercarse lo más posible a esta instantaneidad.

Se recomienda la futura implementación de un sistema Opencaster [21] que facilite el envío por *streaming* de elementos ya multiplexados.

Anexo 1: Manual de uso

1 Apertura

El programa requiere una instancia de VLC específica para el correcto funcionamiento de éste. Si en el momento de la apertura existe otra, se cerrará toda instancia VLC existente y abrirá la que el programa necesite.

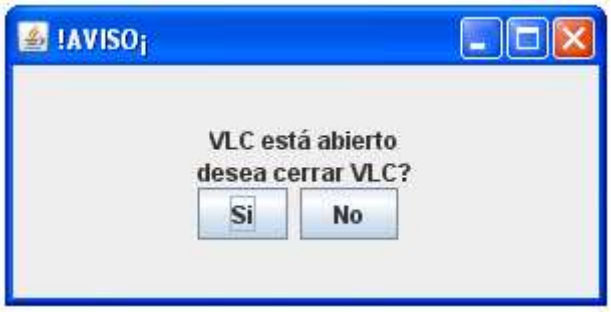


Figura 1. Ventana confirmación de apertura

Antes de ello, una ventana de aviso dará la opción de mantener la instancia antigua sin abrir el programa o, por el contrario, cerrar todas las instancias de VLC y comenzar con el programa.

2. Visionado de vídeos programados

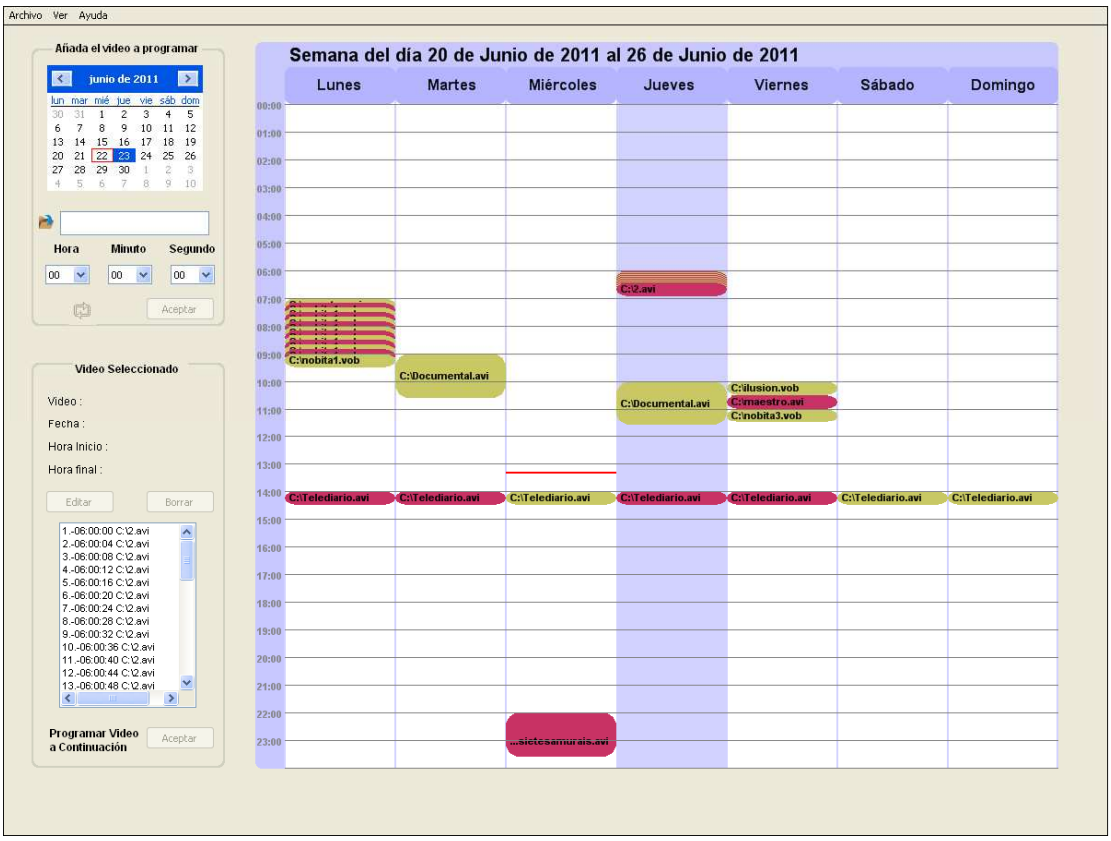


Figura 2.1. Interfaz gráfica de visionado

Como se puede observar en la figura anterior, la ventana para el visionado de los vídeos programados consta de dos partes. La primera representa la programación existente la semana del día seleccionado en el calendario de la parte superior izquierda.

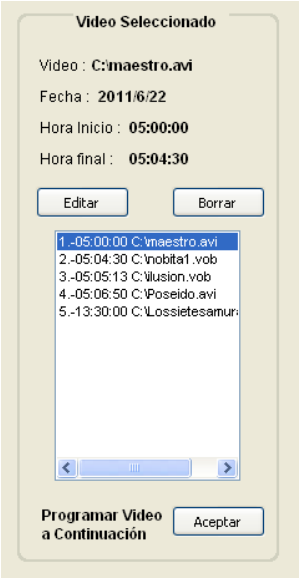


Figura 2.2. Lista vídeos programados el día seleccionado

La segunda representa en una lista todos los vídeos programados el día seleccionado. Esta es muy útil para el visionado de los vídeos de corta duración casi imperceptibles en la agenda semanal. Sobre esta se ofrece la información de programación del vídeo seleccionado.

3. Programación de un vídeo

Los vídeos se programan uno a uno y pueden generarse de varias maneras. La primera es desde el apartado de programación de vídeo de la ventana principal. Aquí sólo se pueden programar vídeos en el día seleccionado. Este apartado permite la elección del vídeo a programar, la hora de reproducción y las repeticiones de este.

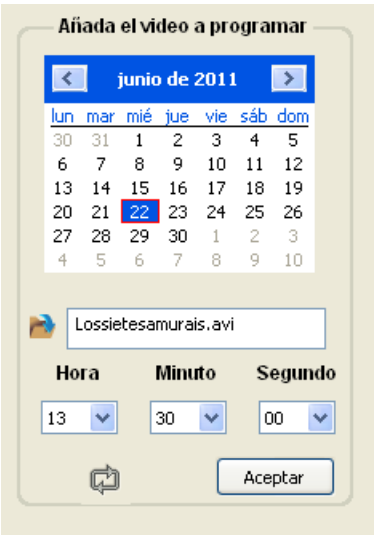


Figura 3.1. Programación de vídeo

La segunda es desde el menú Archivo \ Añadir Vídeo.

Esta ventana permite la elección del vídeo, del día, de la hora y, si se desea, de las repeticiones.

Figura 3.2. Programación de vídeo

Una vez aceptada la programación una ventana pedirá la confirmación. Esta informa de todas las características y da la opción de previsualizar el vídeo seleccionado, con el fin de asegurar una correcta reproducción.

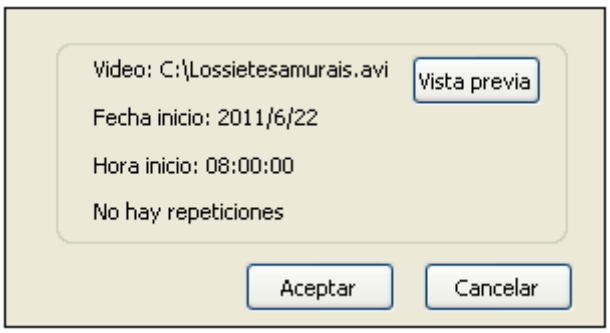


Imagen 3.3. Confirmación de programación

4. Repetir

La ventana de repeticiones permite escoger el periodo y el número de repeticiones. Una vez escogidos, informa del momento en que la última repetición será emitida.

Figura 4. Ventana de repeticiones

Por cada repetición introducida el programa generará una instancia de emisión nueva e independiente de la original.

5. Edición

En la parte inferior derecha de la ventana principal se encuentran las opciones de *Editar*, *Borrar* y *Crear vídeo a continuación* del seleccionado.

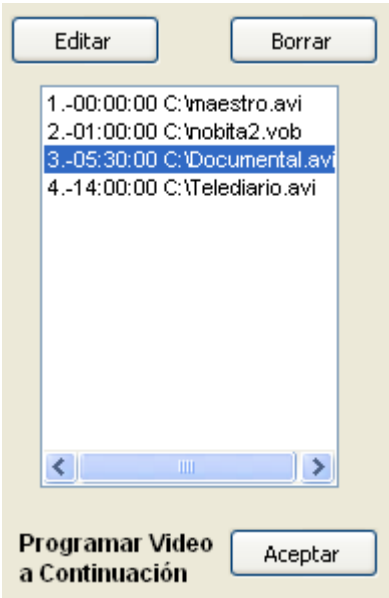


Figura 5.1. Opciones de edición de vídeo

Para escoger el vídeo a editar sólo hay que seleccionarlo en la lista de los vídeos programados el día seleccionado (apartado 2) y pulsar el botón *Editar*. Emergerá una ventana de creación de vídeo con toda la información del vídeo, donde se podrá cambiar toda característica deseada.

Pulsando *Borrar* el vídeo seleccionado se borrará. Antes de ello, una ventana de aviso pedirá la confirmación de este borrado.

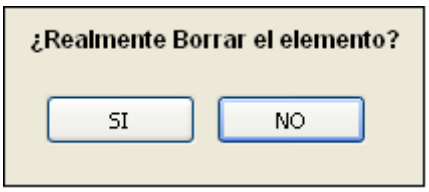


Figura 5.2. Ventana de confirmación de borrado

Aceptar programación a continuación programará un vídeo consecutivo al seleccionado. Para ello, emergerá una ventana con las opciones de día y hora fijadas. Esta permite la elección del vídeo y la generación de repeticiones.

6. Día actual

El menú Ver \ Día actual posiciona el calendario en el día actual.

7. Loop

La opción de *Loop* permite que un vídeo deseado se reproduzca continuamente mientras no haya vídeos programados. En el menú Archivo \ Loop se encuentran las opciones de activar el bucle y escoger el vídeo.

En el caso de activar el bucle en el momento que no haya ningún vídeo programado este comenzará su reproducción instantáneamente. Del mismo modo, si la acción es desactivar el *Loop* mientras este se reproduce, esta reproducción se cancelará en el mismo momento.

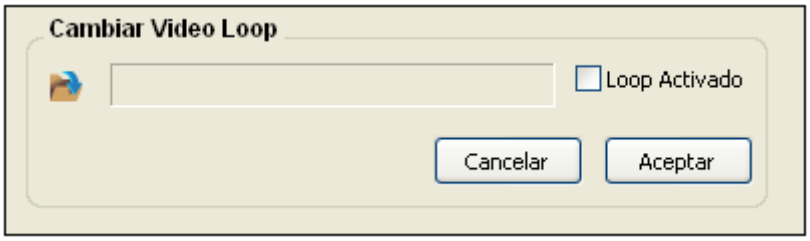


Imagen 7. Ventana de Loop

8. Configuración de salida

En el menú Archivo \ Configuración \ Salida podemos escoger el formato y el tipo de emisión, además del lugar de salida. El cambio de salida afectará a todos los vídeos programados.

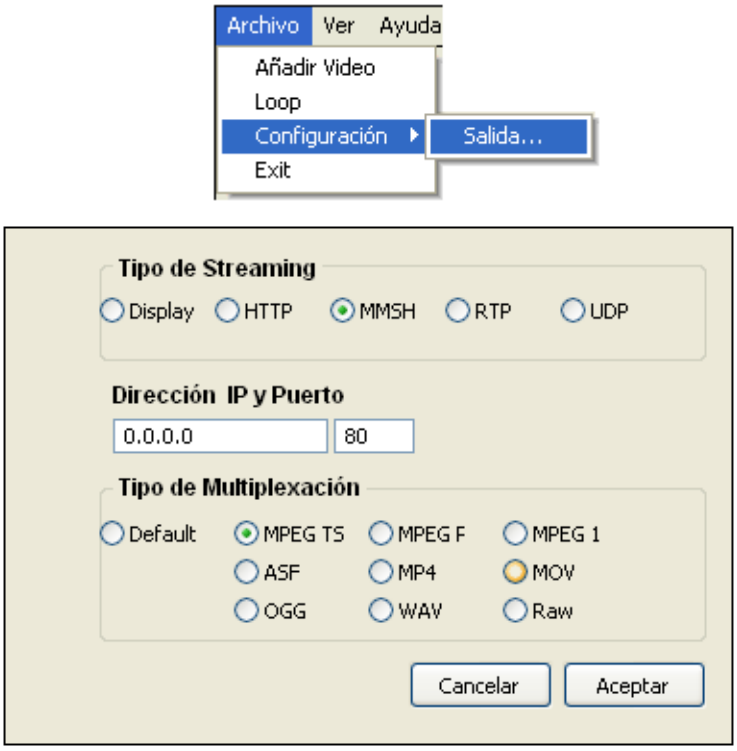


Imagen 8. Ventana de salida

9. Ayuda

El menú Ayuda \ Mostrar ayuda abre el archivo PDF con el manual de uso.

10. Exit

El cierre del programa se encuentra en el menú Archivo \ Exit. Antes del cierre completo del sistema una ventana de aviso pedirá la confirmación del cierre.

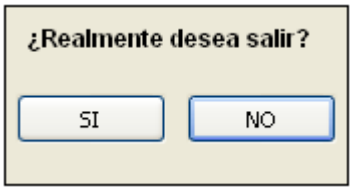


Imagen 10. Ventana confirmación de cierre

11. Instalación

En la carpeta Software del proyecto se incluye el software necesario para el correcto funcionamiento del programa.

- El programa requiere la instalación de la edición estándar de Java, Java SE. Para ello basta con instalar el archivo “jdk-6u22-windows-i586.exe” dado que el programa es desarrollado para plataformas Windows. En caso de utilizar otro sistema operativo se debe descargar la versión Java compatible con el sistema operativo en cuestión. Esto se hace en www.java.sun.com
- La versión de VLC que se ha utilizado para que todo funcione correctamente es VLC 1.0.0 que se incluye en la carpeta de software. Existen versiones en las que esto no funciona, así se debe instalar esta versión.
    - La instalación puede hacerse de dos formas. La primera y recomendada es instalar VLC en la carpeta de recursos, de modo que quede la carpeta VLC dentro de la carpeta de recursos. Dentro de la carpeta VLC estarán todos los archivos, incluido el VLC.exe.
    - Instalar VLC en la carpeta que se desee y copiar la carpeta VLC completa dentro de la carpeta de recursos.

o

Anexo 2: Bibliografía y Referencias

Bibliografía

[1] FFmpeg www.ffmpeg.org

[2]VISUALmpeg www.mpeg-analyzer.com

[3] Bruce Eckel "Thinking in Java"

[4] Tutorial de nikkutos, http://www.youtube.com/watch?v=JbcB3AUwVBY

[5] Java FX http://javafx.com/

[6] Sun Mycrosistems, empresa que creó el lenguaje Java, comprada después por Oracle. http://www.oracle.com/us/sun/index.html

[7] Adobe http://www.adobe.com/es/

[8] Google Calendar www.google.com/calendar

[9] Eclipse www.eclipse.org

[10] SWT www.eclipse.org/swt

[11] tutorial de zetcode http://zetcode.com/tutorials/javaswttutorial/

[12] tutorial de java2s http://www.java2s.com/Tutorial/Java/0280__SWT/Catalog0280__SWT.htm

[13] Interfsacsia http://www.superstable.net/interfascia/

[14] VLC www.videolan.org

[15] Telnet http://es.kioskea.net/contents/internet/telnet.php3

[16] VLM http://www.videolan.org/doc/streaming-howto/en/ch05.html

[17] visualBasic http://msdn.microsoft.com/es-es/library/2x7h1hfk%28v=vs.80%29.aspx

[18] Buffered Reader http://download.oracle.com/javase/1.4.2/docs/api/java/io/BufferedReader.html

[19] Java http://www.java.com/es/

[20] Sebastien Escuder

[21] Opencaster http://www.avalpa.com/

Referencia en la memoria

[REF 1] Código\ src

[REF 2] Código\recursos

[REF 3] Código\ffmpeg

[REF 4] Código\src\Main.java

[REF 5] Código\src\ProgramCheck. java

[REF 6] Código\src\ArchivosBatch. java

[REF 7] Código\src\CalendarioAgenda.java

[REF 8] Código\src\FechayHoraActual.java

[REF 9] Código\src\ExtraerDuracion.java

[REF 10] Código\src\CrearCodigoVLM

[REF 11] Código\src\ExportarVLM

[REF 12] Código\src\FirstDayOfWeek.java

[REF 13] Código\src\SettingsSalida.java

[REF 14] Código\src\Telnet.java

[REF 15] Código\src\CargarVideosGUI.java

[REF 16] Código\src\ArchivosBatch.java

[REF 17] Código\src\ConfirmacionProgramacion.java

[REF 18] Código\src\Solapamiento.java

[REF 19] Código\src\ExtraeEventos.java