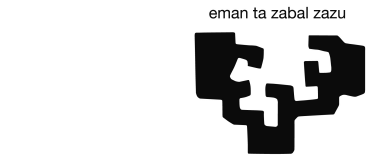

Grado en Ingeniería Informática

Ingeniería del Software

Proyecto de Fin de Grado

Gestor de Metadatos Extendido para el Reproductor Multimedia VLC

Autor

Asier Santos Valcárcel

2017

Resumen

Esta documentación corresponde a la memoria del proyecto *Gestor de Metadatos Extendido para el Reproductor Multimedia VLC*, que tenía como objetivo desarrollar una funcionalidad adicional en el software VLC. Se ha implementado un gestor de metadatos extendido, capaz de editar manual o automáticamente las etiquetas de múltiples ficheros al mismo tiempo. Todo ello ha sido implementado bajo los estándares impuestos por el equipo de desarrollo oficial de VLC, sin añadir nuevas dependencias al software original y con la intención de que dicha funcionalidad pueda y sea incluida en la versión oficial. Se han usado las herramientas provistas por VLC junto con C++ y el framework multiplataforma Qt. El software ha pasado por un proceso de betastesting tras el cual ha sido corregido y mejorado.

El proyecto ha sido llevado a cabo en el marco del Trabajo de Fin de Grado del grado en Ingeniería Informática de la UPV/EHU.

A lo largo del documento el autor, Asier Santos Valcárcel, hace referencia a sí mismo como “el autor” y en ocasiones “el desarrollador” en apartados relacionados con la implementación.

Índice general

Índice de figuras

Índice de tablas

1. CAPÍTULO

Introducción

1.1. Antecedentes

1.1.1. Motivación

A menudo es fácil olvidar el trabajo que tantísimos desarrolladores han dedicado para que cualquier persona pueda usar las tecnologías y plataformas actuales. Muchos de esos desarrolladores ni siquiera han obtenido beneficio de ello, lo han hecho de manera altruista para acercar al público general las maravillas de la computación.

Ante la situación de tener que desarrollar un proyecto como éste, se muestra una oportunidad de aportar un grano de arena a todo ese trabajo acumulado y de alguna manera saldar la deuda que todos los usuarios tienen con esos desarrolladores open source.

Los ejemplos de software open source son muchos, pero uno de los más conocidos para el usuario final es VLC.

1.1.2. ¿Qué son VideoLan y VLC?

VideoLAN es una organización sin ánimo de lucro enfocada a la creación de frameworks y aplicaciones multimedia open source. Su principal proyecto, VLC (o VideoLan Client), es un reproductor multimedia multiplataforma, gratuito y compatible con prácticamente

todos los formatos y códecs existentes. Más allá de la reproducción local, también es capaz de emitir y alimentarse de retransmisiones de red.

Figura 1.1: Logo de VideoLan.

VLC es un gran ejemplo de proyecto open source. Un software complejo, creado por y para la comunidad con un trabajo constante. Es uno de los reproductores multimedia más famosos a nivel mundial y cuenta con millones de usuarios (28.280.000 descargas en el momento de redacción de este documento). Esto también hace que un proyecto a desarrollar sobre él tenga un alcance potencial masivo.

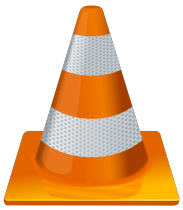

Figura 1.2: Logo de VLC.

Desde su primera versión publicada en febrero de 2001, ha estado en constante desarrollo, gestionado activamente por miembros de la organización en conjunto con la comunidad. Actualmente, la última versión estable de VLC a la hora de redactar este documento es la 2.2.6 y se está trabajando en la versión 3.0.

El software puede ser no solo encontrado en su web, sino también en los principales mercados de aplicaciones oficiales como Google Play, iTunes, Windows Store, etc. Esto significa que cumple los rigurosos estándares impuestos por las empresas que gestionan dichas tiendas.

1.1.3. Relación entre VLC y el público general

Pese a que la mayoría de los usuarios de Windows tienen una copia de VLC, no muchos conocen su nombre o no les consta que lo están usando, lo único que ven es que cuando abren cualquier archivo multimedia, siempre funciona. Sin embargo basta con preguntar coloquialmente: "¿No te suena el icono de un cono de obras?" (figura 1.2) y todo el

mundo lo reconoce. Podría decirse que su funcionamiento es tan bueno, que se ha vuelto transparente para el usuario.

2. CAPÍTULO

Definición del proyecto

2.1. Objetivos

El objetivo es desarrollar una versión funcional de un módulo integrable en la última versión de desarrollo de VLC [Wikipedia Community, 2017] que añada una funcionalidad inexistente y de interés para el usuario medio.

Al comienzo del proyecto la idea principal para el módulo era una interfaz para facilitar el streaming de contenido local a dispositivos UPnP disponibles en la red de área local; sin embargo en la primera fase del proyecto queda patente que las funcionalidades integradas por VLC no son suficientes y la idea requerirá más tiempo del disponible, por lo que se pasa a la que será la idea principal del proyecto: un gestor de metadatos extendido para ficheros de audio (comúnmente conocidos como tags). La funcionalidad ha sido solicitada por múltiples usuarios en el foro de VLC a lo largo de los años tal y como se pueden ver en las referencias [“m33p”, 2009], [“andrea.carlon”, 2012] y [“unique_name”, 2014].

Teniendo en cuenta la cantidad de usuarios activos de VLC, el plugin a desarrollar puede alcanzar cotas de uso muy elevadas. Su distribución apenas tendría coste (se haría a través de la infraestructura ofrecida por VLC) y los usuarios tendrían el software sin tener que pasar por un proceso de instalación específico, fomentado y facilitando así su uso. Las facilidades mencionadas pueden hacer que el usuario se decante por el plugin frente a software *stand-alone* (o independiente).

El trabajo a realizar tiene una complejidad muy superior a la que podría tener el mismo

proyecto si se llevase a cabo fuera de VLC por los siguientes motivos:

- El software deberá usar las funciones de su API (incluso cuando no sean las más prácticas).
- Se deberá evitar crear nuevas dependencias del proyecto (o si es necesario, usar librerías multiplataforma).
- Deberá ceñirse a un estilo de código específico.
- Deberá hacerse uso de los lenguajes y herramientas ya integradas en VLC (cuando a día de hoy probablemente haya idiomas que ofrezcan más facilidades).

El proceso pretende llevarse desde la mayor profesionalidad posible, tratando de gestionar el posible contacto con la organización VideoLAN de manera adecuada y haciendo un software no solo funcional, sino comprensible, eficiente y fácil de mantener.

2.2. Alcance

El proyecto que conforma el TFG es en realidad parte de un proyecto personal más grande y ambicioso, que es al mismo tiempo un reto profesional para el autor. Dicho proyecto probablemente requiera más de las 300 horas en las que se estima la duración del Trabajo de Fin de Grado, por lo que es necesario separar el alcance del Trabajo de Fin de Grado del alcance del proyecto completo.

2.2.1. Alcance del Trabajo de Fin de Grado

El alcance se ha planteado de manera conservadora debido a la falta de experiencia tanto en proyectos de este tamaño como en el ámbito del desarrollo de software profesional.

Se creará un módulo integrable en la última versión de desarrollo de VLC, tratando de no añadir nuevas dependencias al proyecto. El módulo constará de una interfaz gráfica en inglés, será compatible al menos con sistemas operativos Linux y se tratará de que también lo sea para Windows. Centrándose en las funcionalidades, a continuación se listan las mínimas:

- Cargar datos desde la lista de reproducción actual.

- Cargar datos desde un explorador de ficheros.
- Buscar metadatos automáticamente (Búsqueda rápida).
- Permitir el editado manual de datos en la interfaz.
- Editar datos manualmente.
- Guardar los metadatos obtenidos, añadidos y/o editados permanentemente.
- Mostrar una ventana con texto de ayuda.
- Visualizado de la imagen de carátula de cada fichero en la GUI.

Las siguientes funciones serán implementadas, si el tiempo lo permite:

- Buscar metadatos automáticamente (Búsqueda avanzada y consulta por pasos).
- Ver y cambiar la imagen de la carátula (o artwork/cover) del fichero.
- Análisis del nombre del fichero para la búsqueda de metadatos.
- Búsquedas en múltiples bases de datos.

Durante el desarrollo del proyecto surgen nuevas ideas, las cuales no están listadas. Desde el comienzo del proyecto se decide no incluirlas en el alcance para evitar una sobrecarga de tareas. Ello no excluye que algunas de las ideas hayan sido implementadas en el proceso por ser sencillas. En el apartado de conclusiones (capítulo 7) pueden verse las funcionalidades que se han dejado para la segunda iteración del proyecto (fuera del marco del Trabajo de Fin de Grado).

Al finalizar la primera iteración del desarrollo se llevará a cabo un programa de betatesting tras el cual se recogerán errores y sugerencias, que se corregirán en base al tiempo restante disponible.

El despliegue y publicación del software no será abarcado en esta fase, ya que no se espera obtener una versión lista para la distribución hasta pasada la segunda iteración.

2.2.2. Alcance del Proyecto Completo

El objetivo final es corregir todos los posibles errores encontrados en la primera iteración, así como las funcionalidades que se hayan dejado por implementar por falta de tiempo, para así poder enviar un código completamente funcional al equipo oficial de VLC para su evaluación. Si se considera de interés por su parte, deberá llevarse a cabo una revisión del software por ambas partes para asegurarse que cumple todos sus estándares. Finalmente, se pretende que el código sea incluido en las futuras versiones estables de VLC.

El despliegue y publicación del software correrá a cargo del equipo de VLC, si éste considera que el proyecto es de interés.

2.3. Exclusiones del alcance

Queda excluido del alcance del Trabajo de Fin de Grado la traducción de la interfaz más allá del inglés. Esto se debe a que durante la segunda iteración el prototipo puede cambiar significativamente y el esfuerzo de traducción podría ser en vano.

No se implementará una versión compatible con sistemas Mac OS debido a que requiere el uso de distintas tecnologías, lo cual conllevaría tener que crear todo un módulo nuevo estrictamente para dicho sistema operativo. Esto alargaría significativamente tanto la fase de aprendizaje como la de desarrollo, lo que podría generar no llegar a un producto viable o tener que prescindir de la fase de betatesting.

No se crearán sets de pruebas automáticas o regladas para el marco del TFG, ya que la creación de dichas pruebas requiere de un aprendizaje adicional y potencialmente pueden quedar inservibles por los cambios que se vayan a llevar a cabo en la segunda iteración. Esto no conlleva que no se hagan pruebas, especialmente en la fase de betatesting.

También queda excluida la compilación y distribución del software para sistemas operativos Windows. El motivo es que dicha compilación puede ser muy problemática y obtenerlo restaría mucho tiempo de desarrollo. Una descripción más extensa puede ser encontrado en la sección 5.1.3.

3. CAPÍTULO

Gestión

3.1. Planificación

Se hizo una planificación inicial a modo de referencia, ya que sin conocer las herramientas que se iban a usar y sin estar definida completamente la función del módulo, hacer una planificación precisa era imposible. Pronto se demostró que dicha planificación no iba a poder ser cumplida, ya que el autor aceptó un trabajo y la presidencia de una junior empresa, las cuales redujeron significativamente el tiempo disponible para el proyecto.

Vista la situación se probó con un nuevo enfoque: la planificación no se hizo en base a fechas, sino en base al tiempo a invertir en cada tarea. De esa forma se le dedicaba al proyecto el tiempo disponible de manera irregular, sin llegar a perder la perspectiva del avance global del proyecto. Las tareas requerían ser abarcadas en determinado orden, el cual puede verse en al figura 3.1.

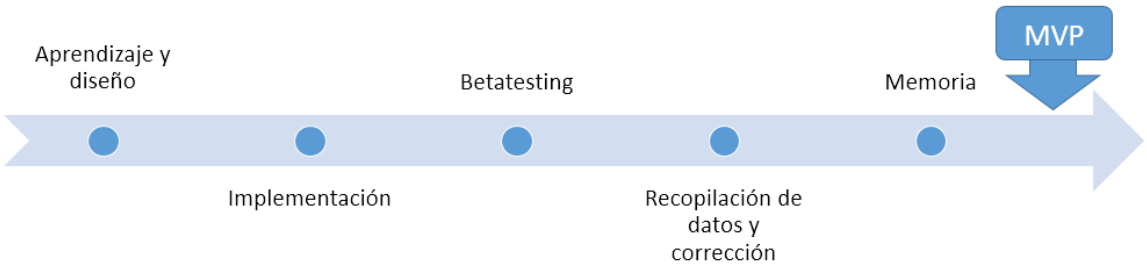

Figura 3.1: Progresión requerida en el proyecto.

En la tabla 3.1 puede observarse el tiempo reservado a cada tarea y el tiempo real requerido para cada una. Se han añadido las subtareas en las que se han ido dividiendo según avanzaba el proyecto. Los datos en rojo son aquellos que han requerido más tiempo del planificado, los verdes son aquellos que han requerido menos y los naranjas aquellos a los que se les asignó tiempo fuera de la planificación inicial (y por lo tanto no han sido sumados al cómputo total de la planificación).

Tarea	**Subtarea**	**H. planificadas**	**H. invertidas**
Compilación		15	23
	Sistemas Linux	6	3
	Sistemas Windows	9	20
Aprendizaje		12*	46,5
	Módulo Hello World	4	4,5
	API LUA	3	3,5
	LUA y C	4	5,5
	Módulo GUI en VLC	6	20
	Qt y C++	6	6
	API VLC	4	7
Desarrollo		125	111
	Lógica de Negocio	70	64
	GUI	35	38
	Corrección de bugs	20	9
Betatesting		30	25
	Empaquetado	10	9
	Reclutamiento	2	2
	Preparación de material	13	10
	Sesiones	5	4
Memoria		70	65
	Redacción	40	53,5
	Adecuación y corrección	25	9,5
	Maquetado	5	2
Documentación		15	12
Gestión		5	3
Reuniones		8	6,5
Otros		20	16
	Planificación	10	11
	Colchón para emergencias	10	5
Total		300	308

Tabla 3.1: Planificación por horas.

3.2. Plan de Gestión

A continuación se describe el plan de gestión aplicado al proyecto.

3.2.1. Gestión del tiempo

En esta sección se muestra un listado de los hitos más relevantes del proyecto:

- **Comienzo del proyecto:** 11 de octubre de 2016
- **Comienzo de la fase 1:** 10 de enero de 2017
- **Finalización del la fase 1 y comienzo de la fase 2:** 24 de febrero 2017
- **Finalización del la fase 2 y comienzo de la fase 3:** 12 de mayo de 2017
- **Finalización del la fase 3 (comienzo del betatesting y redacción de la memoria):** 21 de junio de 2017
- **Finalización del betatesting:** 3 de julio de 2017
- **Finalización de la primera versión de la memoria:** 2 de agosto de 2017

3.2.2. Gestión de riesgos

Dado que el software a desarrollar va a ser open source y este documento va a ser público, el filtrado de información no supondría un problema. Sin embargo, la pérdida de la información generada sí es un riesgo significativo, por lo que gestionar las copias de seguridad del proyecto es importante. Teniendo en cuenta lo anterior se opta por mantener toda la información generada en servicios en la nube conocidos, con los que el riesgo de pérdida de la información almacenada en sea muy reducida. Las plataformas específicas se describen más adelante, en la sección 3.3. La información a proteger puede dividirse en los siguientes tipos.

- Código generado.
- Ficheros de gestión.

- Ficheros varios generados durante el desarrollo.
- Memoria en formato LaTeX.

Adicionalmente, se tienen copias locales del contenido almacenado en los servicios en la nube. En este proyecto se han usado dos equipos principalmente, alternando entre ellos al menos una vez a la semana por limitaciones técnicas de los mismos. Esto conlleva que ambos equipos tuviesen una copia de los ficheros con una antigüedad máxima de una semana y en caso catastrófico de pérdida de la nube, se seguiría contando con las dos copias locales.

3.2.3. Seguimiento

El seguimiento del tiempo invertido se hace con la herramienta web de PrimaERP (figura 3.2), en la que se va registrando todo el tiempo invertido. La herramienta permite establecer un cronómetro cuando se empieza una tarea y detenerlo cuando se hace una pausa o se termina e ir escribiendo comentarios sobre lo que se ha hecho en ese tiempo. Siendo meticuloso y constante como se ha sido y dado que la granularidad es muy alta, el margen de error de los tiempos medidos es ínfima, pudiendo ser de apenas minutos. La herramienta también permite asociar los registros de tiempo a determinadas tareas, a las cuales también se les pueden poner planes de tiempo a invertir. Al registrar en PrimaERP la planificación hecha en el apartado 3.1 se obtiene una visión global del avance del proyecto.

3.2.4. Control

Dado que el alcance del PFG establecido ofrece cierto margen y que la finalización del mismo no significa el fin del proyecto, se pueden ir equilibrando los tiempos dedicados de manera que el tiempo total invertido se acerque a las 300 horas planificadas y las tareas incompletas se abarquen en la segunda iteración (fuera del marco del TFG).

Tal y como se puede ver en la tabla 3.1, se le ha tenido que dedicar mucho más tiempo del planificado al aprendizaje por motivos que se describen el la sección 5. Esto ha hecho que se tuviese que reducir el tiempo dedicado al desarrollo y betatesting.

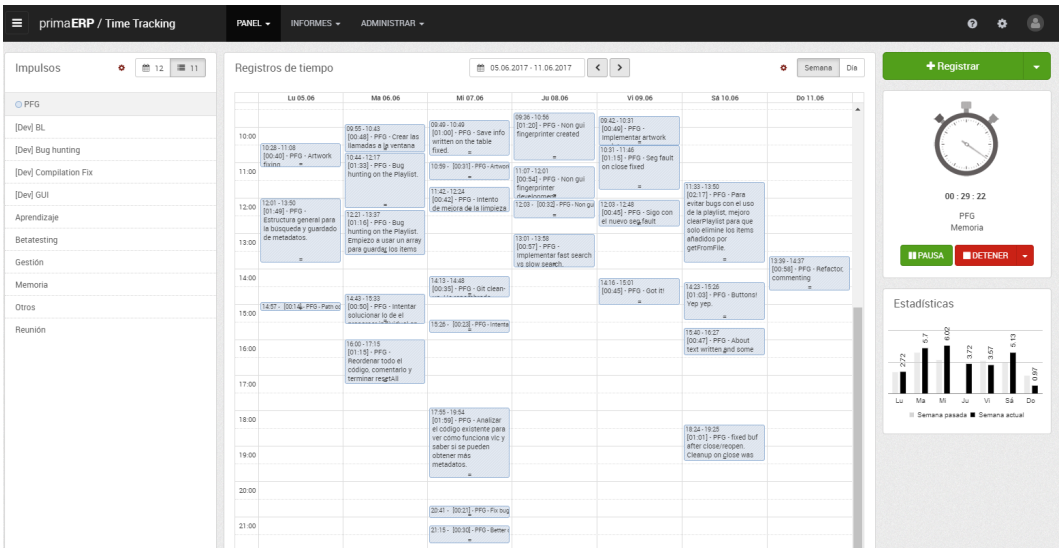

Figura 3.2: Servicio *primaERP* en uso.

3.2.5. Decisiones tomadas

A continuación, un resumen de las decisiones más significativas tomadas durante el proyecto.

3.2.5.1. Compilación para Windows

La compilación de VLC para Windows se muestra problemática desde el primer momento (una descripción extensa del problema puede ser encontrada en la sección 5.1.3). Una vez superado el umbral definido en la planificación se debe decidir si se le asigna tiempo adicional o si se descarta, lo cual significará que la fase de betatesting será más complicada. Se decide mantener el planteamiento inicial de desarrollar la aplicación para tanto sistemas Windows como sistemas Linux, pero sin abarcar la compilación y testing para Windows en la primera iteración. Las consecuencias de esta decisión en la fase de betatesting pueden verse descritas en la sección 6.

3.2.5.2. Tipo de módulo a desarrollar

Los módulos de VLC pueden ser desarrollados dentro del árbol de VLC (también denominado *in-tree*) o fuera de él (*out-of-tree*) [VideoLAN Organization, 2017a]. Las ventajas del desarrollo *in-tree* son las siguientes:

- El equipo de VLC revisará el código, lo cual es una gran oportunidad de mejora y aprendizaje.
- El módulo podrá ser traducido a múltiples idiomas por la comunidad de VLC.
- El módulo podrá ser distribuido a través de la web oficial de VLC, contando con su respectiva infraestructura de control.
- El módulo será compatible con múltiples versiones de VLC y para múltiples arquitecturas.

Las ventajas del desarrollo *out-of-tree* son las siguientes:

- La compilación es mucho más rápida.
- Puede usarse cualquier sistema de control de versiones.
- No es necesario ceñirse a las licencias de VLC.
- El módulo puede ser publicado en cualquier momento, sin tener que ceñirse al plan de publicación del equipo de VLC.
- Pueden usarse idiomas que no están oficialmente soportados por VLC (en teoría).
- Pueden usarse librerías en las que VLC no podría o sería inadecuado que dependiese.

Si bien el desarrollo *out-of-tree* parece ofrecer más libertad a la hora del desarrollo, el objetivo del proyecto es desde el comienzo que el módulo sea parte de VLC y no una herramienta que añadirle. Esto se debe a que ya existen herramientas externas con las funciones planteadas por el proyecto, el objetivo es hacerlo en VLC para que éste también sea capaz. Por ello se opta por el desarrollo *in-tree*.

3.2.5.3. Funcionalidad del módulo a desarrollar

Con apenas 12 horas de desarrollo invertidas en la idea de crear un módulo capaz de emitir contenido a dispositivos de la área local usando UPnP, ya se puede apreciar que VLC no cuenta con las funciones esperadas y que el desarrollo va a ser significativamente más complejo de lo pensado inicialmente, por lo que se decide cambiar la funcionalidad

del módulo a un gestor de metadatos capaz de buscar en internet los tags de múltiples ficheros de audio simultáneamente, así como modificarlos manualmente. Una descripción más extensa de lo aquí descrito puede ser encontrada en la sección 5.2.1.

3.2.5.4. Elección de bases de datos para la consulta de metadatos

Habiendo analizado múltiples alternativas, se decide que se priorizará la implementación de la base de datos *acoustID* haciendo uso de la librería *Chromaprint* y posteriormente, si el tiempo restante lo permite, se implementará la búsqueda a través de una o varias de las bases de datos con consulta vía texto. Una descripción más extensa de lo aquí descrito puede ser encontrada en la sección 4.2.

3.2.5.5. Abandono de LUA y adopción de C++ y Qt

Durante la fase 2 del desarrollo se descubre que el equipo de VLC ya ha implementado parte de lo que iba a ser el programa y que la API de VLC para LUA va a ser insuficiente para poder darle un valor añadido suficiente al módulo para el éxito del proyecto, por lo que se decide abandonar la rama de desarrollo en LUA, migrar lo posible a C++ y Qt y así comenzar la fase 3 del desarrollo. Una descripción más extensa de lo aquí descrito puede ser encontrada en la sección 5.2.3.1.

3.2.5.6. Empaquetado y distribución en el betatesting

Dada la complejidad de VLC y sus múltiples dependencias, el empaquetado de la última versión de desarrollo se presenta complicada desde el comienzo. Una vez se alcanzan las 7 horas de las 10 reservadas para el empaquetado sin apenas haber avanzado, se decide no crear un paquete que cumplimente todos los requisitos de Debian y conformarse con uno más simple pero que a su vez su instalación sea más compleja. Esto requiere una posterior gestión adicional del material y las instrucciones provistas a los testers. Una descripción más extensa de lo aquí descrito puede ser encontrada en la sección 6.1.2.

También se decide cambiar la ruta de instalación del paquete a */EMM_BETA/* para evitar conflictos con las posibles copias de VLC oficiales que puedan tener los testers.

3.2.6. Idioma

Toda la interacción entre el autor y el tutor de este trabajo había sido en euskera hasta la definición del proyecto y por ello se decidió mantenerlo así.

Por limitaciones del autor a la hora de redactar texto técnico, se opta por desarrollar este documento en español. Lo mismo se aplica para la defensa del proyecto.

Dado que el idioma oficial de desarrollo y documentación del software sobre el que se va a trabajar (VLC) es el inglés, todo el código ha sido desarrollado en dicho idioma, haciendo así el proyecto trilingüe.

3.2.7. Gestión de las reuniones

El autor termina las últimas clases de su grado en Ingeniería Informática en Donostia/San Sebastián durante el desarrollo de este proyecto. Siendo el autor de Vitoria/Gasteiz, éste se vuelve a su ciudad de origen, lo que supone un problema a la hora de establecer reuniones con el tutor, por lo que se acuerda la posibilidad de mantener reuniones telemáticas a través de la plataforma de videoconferencia gratuita Google Hangouts cuando los temas a tratar no requieran de interacción física. Las actas pueden ser encontradas en el anexo D.

3.3. Recursos

A continuación se describen los recursos y herramientas usados para los distintos apartados del proyecto.

3.3.1. Tecnologías escogidas

A lo largo del desarrollo se prueban distintos enfoques que requieren de diferentes lenguajes de programación para ser implementados. A continuación se presenta un listado de los lenguajes y frameworks usados:

3.3.1.1. C

VLC está mayormente implementado en C por su versatilidad a los niveles más bajos. Al ser uno de los lenguajes más extendidos, cuenta con librerías para prácticamente cualquier

funcionalidad imaginable en distintas variedades, desde las más completas, pasando por las librerías multiplataforma, hasta las versiones más livianas.

3.3.1.2. C++

Si bien en VLC predomina C, el apartado de la interfaz visual está desarrollado mayormente en C++ por las facilidades ofrecidas para interactuar con el framework QT. Las librerías base de C++ incluyen las principales funciones propias de los idiomas orientados a objetos, pero requieren de grandes cantidades de memoria para operar, por lo que no se usan en VLC (aunque cuenta con su propia implementación básica de algunos elementos avanzados no incluidos en C, como las listas).

3.3.1.3. Qt

En las versiones de Windows y Linux de VLC se usa el framework Qt para la creación de interfaces gráficas. Cuenta con dos licencias, una siendo de uso comercial con funciones extendidas y la otra gratuita estando bajo licencias GPL que obligan a que el software sea open source. La segunda es usada por VLC y será también usada para este proyecto.

3.3.1.4. LUA

LUA es un idioma de scripting extensible a través de enlaces o "bindings" a otros idiomas como C, lo cual le ofrece una versatilidad excepcional. Si bien es un idioma interpretado, puede ser compilado para obtener mejores tiempos de carga y ejecución. VLC cuenta con una API para LUA que cuenta con elementos gráficos básicos.

3.3.2. Entorno de desarrollo

A continuación se describen las herramientas usadas para crear el entorno de desarrollo.

3.3.2.1. VirtualBox

La compilación de VLC en su versión más reciente requiere de librerías experimentales y potencialmente inestables que podrían afectar el correcto funcionamiento del sistema operativo. Por este motivo se opta por crear un entorno aislado, únicamente dedicado al

proyecto. Para ello se usa el software para creación de máquinas virtuales VirtualBox, por ser open source y gratuito (a excepción de determinadas funciones que no son necesarias para este proyecto).

3.3.2.2. Debian

El equipo oficial de VLC recomienda la compilación en sistemas Linux incluso para las versiones de Windows (práctica denominada como Cross-Compiling).

"Las distribuciones Ubuntu (basadas en Debian) son muy completas, versátiles y son más fáciles de desplegar. Sin embargo, Ubuntu incluye en sus repositorios software estable en el momento de su desarrollo, lo que hace a veces complicado obtener las últimas versiones disponibles (por poder ser inestables y no haber sido supervisadas por el equipo de Ubuntu). Obtener las últimas versiones es algo indispensable en este proyecto, por lo que se decide usar Debian, la distribución desde la que desciende Ubuntu, la cual no cuenta con la limitación mencionada.

3.3.2.3. Git + GitHub + GitKraken

El proyecto a realizar será open source, por lo que se opta por el sistema de control de versiones más extendido, Git, a través del servicio GitHub, el servidor para proyectos open source por excelencia. Para facilitar el uso de Git se opta por usar GitKraken, un software que ofrece una interfaz gráfica agradable para la gestión de repositorios Git, contando también con un visualizador de la representación gráfica del flujo del repositorio. Su interfaz simple, la integración con GitHub y la licencia gratuita para uso no comercial lo hacen la opción más atractiva.

3.3.2.4. Atom

En el proyecto se trabajará mayormente con C/C++, pero no se limita a ello, por lo que es necesario un IDE o editor de texto versátil. Dado que la compilación de VLC es compleja debido a todas sus dependencias, el uso de un IDE como Visual Studio, Eclipse u otros puede potencialmente generar más problemas de los que soluciona. Es por eso que se opta por usar una terminal para la compilación y un editor de texto para escribir el código.

Siendo Atom compatible nativamente con C/C++ y teniendo la opción de instalar paque-

tes adicionales que aumenten su versatilidad (snippets para nuevos idiomas, previsualizadores de código, etc.) es el editor perfecto para este proyecto.

3.3.2.5. Qt Designer

A la hora de crear interfaces haciendo uso del framework Qt la manera más sencilla de hacerlo es a través de una de las herramientas de desarrollo provistas por la empresa. *Qt Designer* es la más sencilla de dichas herramientas, enfocada más a la creación de interfaces y menos a proyectos complejos, por lo que es la que ha sido escogida para este proyecto.

3.3.2.6. Librerías y toolchain de VLC

A través de los repositorios de software de las distribuciones Debian, obtener las herramientas de desarrollo y compilación (o toolchain) para sistemas Linux es trivial. Este proceso se describe más adelante, en la sección 5.1.2.

En el caso de sistemas Windows el proceso es más complejo ya que requiere de una compilación individual de las librerías. El proceso se describe en la sección 5.1.3.

3.3.3. Seguimiento y control

A continuación se describen las herramientas usadas para llevar a cabo el seguimiento y control del proyecto.

3.3.3.1. PrimaERP

Debido a que el tiempo invertido en el proyecto no sigue un horario establecido, se requiere de una herramienta de seguimiento precisa que facilite anotar las tareas y el tiempo invertido. Se opta por analizar únicamente aquellos servicios que tienen (al menos) una interfaz web y que son gratuitos para la situación bajo la que se desarrolla este proyecto. Finalmente se opta por PrimaERP.com por su interfaz agradable y simple, sus opciones para visualizar el progreso de tareas y por las facilidades para exportar los datos generados a formato Spreadsheet.

3.3.4. Documentación

A continuación se describen las herramientas usadas para la creación de este documento.

3.3.4.1. ShareLatex

Este proyecto se ha desarrollado desde múltiples dispositivos por lo que se han priorizado las herramientas con soporte web, para evitar tener que instalar y mantener muchas herramientas. Tras un análisis de los servicios online para escritura de documentación en LaTeX, se opta por ShareLatex, por no contar con limitaciones de espacio y tener una interfaz agradable y liviana.

3.3.4.2. Modelio y sequencediagram.org

Modelio es una herramienta open source diseñada para el modelado en distintos estándares como UML, BPMN, MDA, SysML, etc. Ha sido usada para desarrollar los diagramas UML incluidos en este documento. En un punto del proyecto se carece de los equipos del desarrollador por causas de peso mayor, por lo que se termina generando parte de los diagramas UML en sequencediagram.org, un servicio web extremadamente básico.

3.3.4.3. Draw.io

Para la creación de esquemas y gráficos genéricos y sencillos (como los de las figuras 5.4 y 5.5) la herramienta online Draw.io es la mejor opción. Los ficheros creados pueden gestionarse desde Google Drive, se pueden exportar las imágenes creadas a múltiples formatos y es completamente gratuito.

3.3.4.4. Google Drive

La documentación generada a lo largo del proyecto, así como otros ficheros de interés, han sido almacenados en el servicio en la nube Google Drive. Se ha escogido dicho servicio por ser gratuito, conocido para el autor, por tener mucha capacidad de almacenamiento y por ofrecer funcionalidades útiles para las fases de betatesting (como la compartición de múltiples ficheros sin necesidad de pasar por un proceso de registrado).

4. CAPÍTULO

Diseño

4.1. Arquitectura

El módulo, al estar integrado en VLC, tendrá que usar los lenguajes de programación compatibles con la API provista. Además, como pretende estar incluido en la distribución oficial (ser un módulo *in-tree*), no podrá limitarse a hacer uso de lenguajes compatibles con la API, sino que tendrá que usar aquellos en los que está construido VLC. Tal y como se describirá más adelante, los lenguajes disponibles son C, C++ y LUA.

Respecto a la API, se deberá de usar aquella provista por VLC. Si hiciese falta, para la obtención de información online, se usarán APIs web, que puedan ser accedidas con las herramientas provistas por VLC.

Al ser un módulo mayormente gráfico, también se deben de usar las herramientas en las que está basado VLC. Las principales opciones, tal y como se describe más adelante, son LUA (haciendo uso de una API) y el framework Qt.

4.2. Bases de datos

Tras un análisis rápido en la web, se encuentran múltiples bases de datos y métodos de búsqueda, de las cuales las más prometedoras se encuentran listadas y clasificadas en la tabla 4.1:

Modo de uso	Base de datos
Consulta vía texto	freeDb MusicBrainz Discogs
Parseado de contenido web	Amazon iTunes
Consulta vía hashing	acoustID

Tabla 4.1: Bases de datos para la obtención de metadatos clasificadas por modo de uso.

La consulta vía texto consiste en dar una cadena de caracteres a la base de datos que facilite identificar la pista. Los datos a buscar podrían ser los metadatos con los que ya cuente la pista o el nombre del fichero. Su implementación es relativamente sencilla y eficaz, pero cuenta con dos grandes limitaciones:

1. Un fichero con metadatos y/o nombre erróneos hará que la búsqueda devuelva información equivocada.
2. Un fichero sin metadatos y con un nombre no significativo no podrá ser identificado usando estas bases de datos.

Las bases de datos enfocadas al uso desde la web son más complejas de usar en proyectos como este. La implementación consistiría en efectuar una petición HTTP al servidor para que este devuelva una página web completa. Después habría que buscar entre los elementos de la web los datos de interés. Esto se debe a que estas bases de datos están enfocadas al uso comercial de sus propias webs y no a la consulta de terceros, aunque son toleradas. La ventaja de estas bases de datos es que son actualizadas activamente por las empresas que las gestionan y cuentan con mucha información.

Finalmente la base de datos con técnicas de hashing, *acoustID*, ofrece la posibilidad de hacer un análisis de los datos contenidos en el fichero. Haciendo uso de una librería llamada *Chromaprint* se puede aplicar un algoritmo a un fichero, el cual analiza el espectrograma del audio contenido. Dicho espectrograma es teóricamente el mismo o al menos muy similar incluso cuando el fichero ha sido codificado con distintos métodos. Del espectrograma obtiene una cadena de caracteres única que identifica la pista procesada, de manera similar a las técnicas de hashing tradicionales. El algoritmo es incluso capaz de identificar versiones de estudio y directos como la misma pista en los casos en los que el audio es muy nítido. A esta técnica se le llama “fingerprinting” o creación de huella

dactilar. Una explicación más extensa del proceso puede encontrase en la entrada de la bibliografía [Lalinský, 2011]. La cadena de caracteres obtenida a través de *Chromaprint* es enviada a la base de datos en una consulta para así obtener los metadatos asociados a la pista. Muchos de los registros de *acoustID* están al mismo tiempo ligados a la base de datos *MusicBrainz*, haciéndola así más completa y versátil.

De todos los métodos, el más prometedor es el provisto por *acoustID*, seguido de aquellas bases de datos cuya consulta se hace a través de texto y por último las que requieren un parseado de contenido web. Por lo tanto se decide que se priorizará la implementación de *acoustID* haciendo uso de *Chromaprint* y posteriormente, si el tiempo restante lo permite, se implementará la búsqueda a través de una o varias de las bases de datos con consulta vía texto.

4.3. Diseño gráfico

Se comienza por crear un boceto en papel de la interfaz. Dicho boceto es refinado con la ayuda de Israel Betolaza Hinojosa, estudiante de Grado en Creación y Diseño en la UPV/EHU. El resultado del trabajo conjunto puede verse en la figura 4.1.

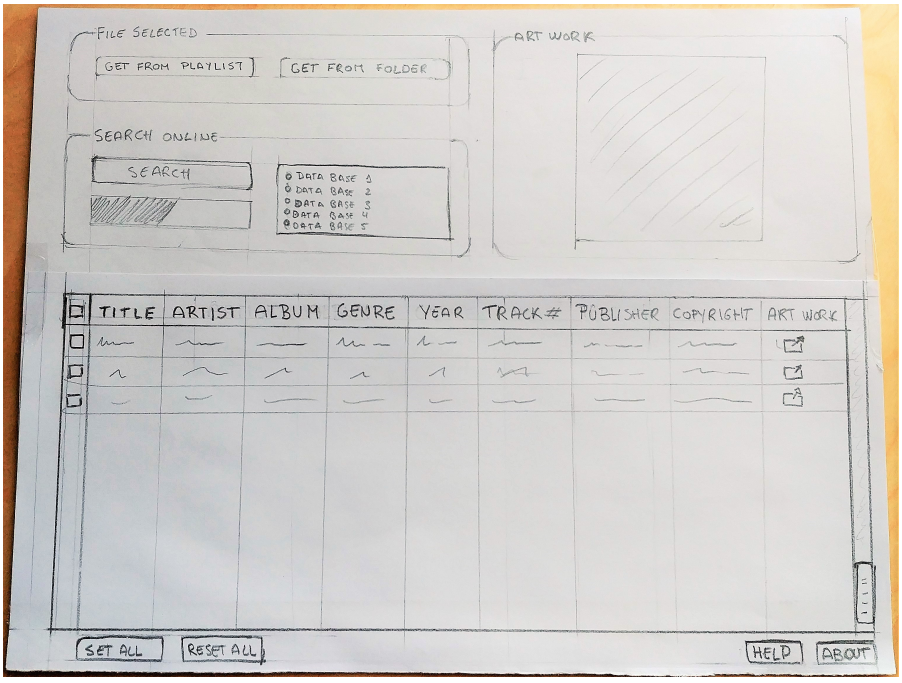

Figura 4.1: Boceto de la interfaz para el *Gestor de Metadatos Extendido*.

Se decide que añadir a todos los botones un icono puede facilitar la comprensión de las funciones de cada uno. Se procede entonces a analizar los iconos ya incorporados en VLC. Muchos de ellos son útiles y se reutilizan para el módulo, sin embargo faltan imágenes para representar los siguientes conceptos:

1. Guardado.
2. Búsqueda automática.
3. Limpieza o borrado de la tabla.

Tras reflexionar sobre imágenes con significado universal, se decide asignar las siguientes imágenes a los valores:

1. Un disquete.
2. Una lupa.
3. Un cepillo o una escoba.

Se pide entonces asistencia, una vez más, a Israel Betolaza Hinojosa para que diseñe los iconos/imágenes necesarios. Tomando como referencia los iconos ya existentes en VLC se topan las siguientes limitaciones:

- Deben tener un tamaño de 16x16 píxeles.
- Deben ceñirse a la estética simple y minimalista de VLC.

El reducido tamaño de las imágenes hace que un diseño convencional no sea suficiente, por lo que se baraja aplicar *Pixel Art*. Siendo una técnica desconocida para Israel Betolaza, este procede a practicar libremente, para finalmente llegar a los resultados de las figuras 4.2a, 4.2b y 4.2c en apenas una hora.

(a) Guardado. **(b)** Búsqueda. **(c)** Limpieza.

Figura 4.2: Iconos creados para la interfaz.

Finalmente se le pide a Israel Betolaza que diseñe una imagen a modo de logo que represente el Gestor de Metadatos Extendido a desarrollar. Procede a crear una serie de bocetos y tras decantarse conjuntamente por uno de ellos, se refina y se obtiene la imagen de la figura 4.3. Se crean dos formatos, uno de alta resolución para su uso dentro de la propia ventana del módulo y otra en resolución de 16x16 píxeles a modo de icono de su entrada de menú.

Figura 4.3: Logo para el Gestor de Metadatos Extendido.

El logo que se muestra en la figura 4.3 está compuesto por dos elementos: el planeta Tierra (que representa la búsqueda de información en Internet) y dos corcheas (que representan la música a procesar). Respetando la estética propia de VLC se usan 3 colores que abundan en la interfaz: negro, gris y blanco.

4.4. Diseño técnico

Tal y como se describirá en el capítulo 5, a lo largo del proyecto se trata de obtener el módulo usando distintas tecnologías, todas ellas desconocidas al comenzar el proyecto. En esta sección se muestra un resumen del diseño final, el cual fue hecho durante la tercera fase del desarrollo (sección 5.3).

Una vez analizadas las capacidades de las herramientas a usar y comprendidas las limitaciones a superar, se procede a modelar los casos de uso a implementar. En la figura 4.4 puede verse el diagrama de casos de uso desarrollados.

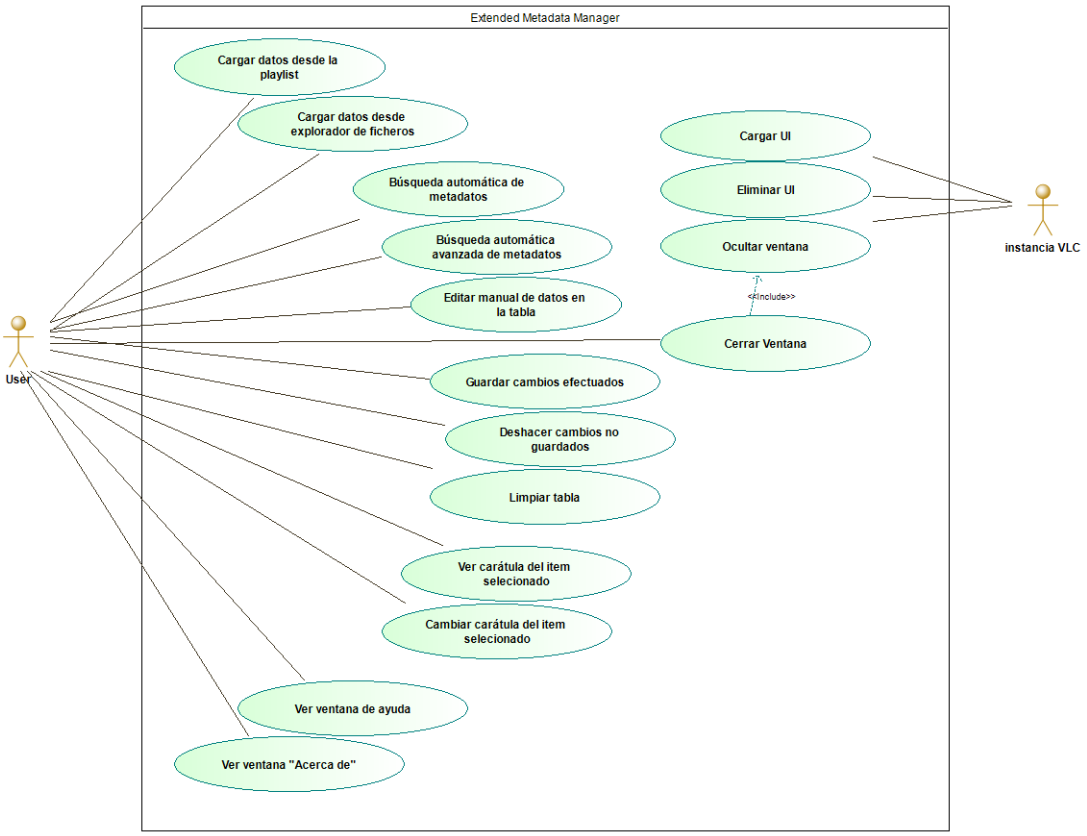

Figura 4.4: Diagrama de casos de uso para el Gestor de Metadatos Extendido.

La mayoría de los casos de uso siguen un patrón similar debido al uso de la API de VLC (más información sobre la API en la sección 5.3.2.2). En la figura 4.5 se muestra a modo de ejemplo el diagrama de secuencia del caso de uso “Búsqueda automática de metadatos”, por ser un buen referente. El resto de los casos son similares o más simples.

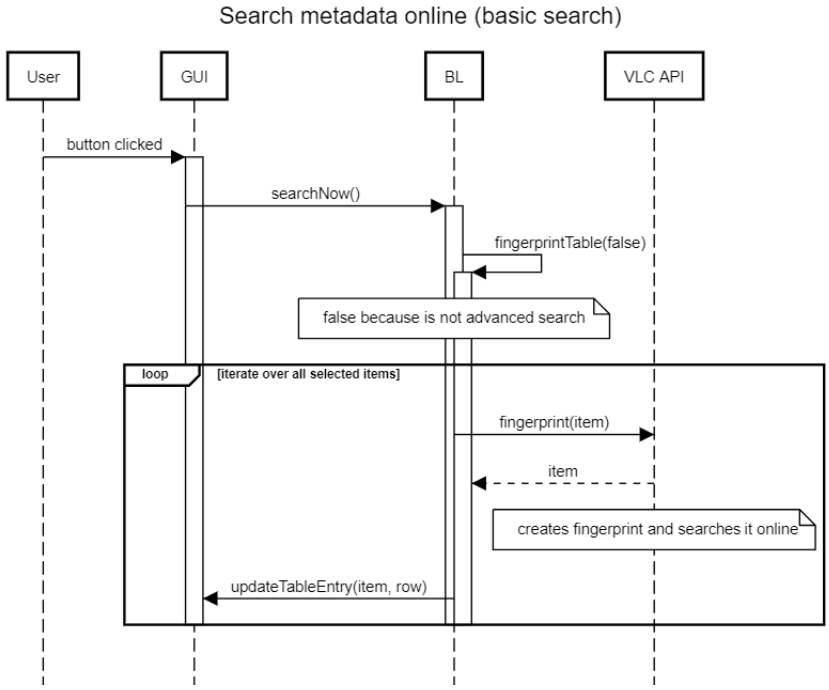

Figura 4.5: Diagrama de secuencia para “Búsqueda automática de metadatos”.

5. CAPÍTULO

Desarrollo del proyecto

En este capítulo se describen las distintas fases del desarrollo y algunos de los procesos llevados a cabo. Con el afán de simplificar la explicación, se da por hecho que los comandos descritos son para sistemas basados en Debian con la herramienta `SUDO` instalada y adecuadamente configurada.

5.1. Primera Fase - Aprendizaje

A continuación se describe la fase inicial del desarrollo, que se centra en el aprendizaje de las tecnologías y herramientas necesarias.

5.1.1. Obtención del código

El equipo de VideoLAN cuenta con múltiples repositorios públicos para todos sus proyectos. Los únicos relevantes para este proyecto son los de VLC. Si bien el oficial se aloja en `https://git.videolan.org`, su clon en la plataforma GitHub (disponible en `https://github.com/videolan/vlc`) es más cómodo por ser un entorno más conocido para el desarrollador. La versión en GitHub es actualizada con menos frecuencia y puede tener hasta un día de retraso, pero es un margen insignificante para el proyecto y cuenta con la posibilidad de obtener una versión más reciente desde el repositorio oficial si se configura este último como un origen adicional en las copias locales del desarrollador.

El primer paso es obtener la herramienta Git (si no se tiene ya) con el siguiente comando:

```
sudo apt-get install git
```

Puede obtenerse una copia local del repositorio con el siguiente comando de terminal (se requiere tener instalado el software de control de versiones git):

```
git clone https://github.com/videolan/vlc
```

Si bien los repositorios mencionados antes son públicos, los permisos de escritura están desactivados por defecto, para que solo el equipo oficial pueda hacer modificaciones. Sin embargo, Github permite crear un Fork, o variante del proyecto, capaz de contener su propio código y pudiendo incorporar en cualquier momento las modificaciones hechas en el original. Se procede a crear un Fork disponible en el siguiente enlace: https://tiny.cc/emm_for_vlc. El repositorio puede ser clonado localmente con el siguiente comando:

```
git clone https://github.com/ASantosVal/vlc-extension-trials
```

Independientemente de qué repositorio se use, una vez obtenido el código hay que ejecutar los siguientes comandos para prepararlo:

```
cd carpetaConElCodigo
./bootstrap
```

Completados estos pasos ya se tiene una copia con todo el código necesario y puede pasarse a la compilación.

5.1.2. Compilado para sistemas Linux

El equipo de VideoLAN deja constancia en la documentación pública que tienen una falta de personal dedicado al documentado y que por ello las guías pueden estar incompletas y/o desactualizadas. Esto conlleva que efectivamente la mayoría de las guías requieran de ajustes o pasos adicionales para ser válidas. Se ha tomado como referencia la guía disponible en la bibliografía [VideoLAN Organization, 2017d], sin embargo no se ciñe estrictamente a ella.

Una vez se tiene una copia local del código, el siguiente paso es obtener las herramientas necesarias para la compilación:

```
sudo apt-get install build-essential pkg-config libtool automake autopoint gettext
```

También es necesario obtener las librerías necesarias. Existen dos maneras de obtenerlas: a través de repositorio o con el método *"Contrib"*. El primero es trivial, basta con el siguiente comando (teniendo los repositorios de desarrollo o *"-dev"* activados):

```
sudo apt-get build-dep vlc
```

El método *"Contrib"* consiste en compilar localmente las librerías necesarias, incluidas en el código fuente. Esto puede ser problemático, pues las versiones incluidas son para la última versión estable y no para la de desarrollo (sobre la que se va a trabajar en este proyecto). Por ello no ha sido usada en el proyecto, pero con carácter informativo, el proceso sería el siguiente (estando dentro de la carpeta que contiene el repositorio):

```
cd contrib
mkdir native
cd native
../bootstrap
make
```

Una vez se tienen las herramientas necesarias, el proceso de compilado es fácilmente reconocible para personas que han compilado antes herramientas en sistemas Linux:

```
./configure
make
```

El comando /configure; prepara la compilación, activando o desactivando funciones del software. También comprueba que el sistema cuenta con todas la librerías necesarias. Si alguno de los pasos mencionados en esta sección no se ha llevado a cabo correctamente, este comando probablemente devolverá un error. Proyectos grandes como éste no cuentan con un fichero *Makefile*, en su lugar incluyen una plantilla llamada *Makefile.in*, la cual es tomada por el comando mencionado para generar un fichero *Makefile* específico para el sistema en el que se encuentra.

El comando `make` toma el fichero Makefile generado y procede a compilar todo el proyecto. La falta de librerías o el uso de versiones incorrectas también puede causar que este proceso falle.

Cabe mencionar que se puede modificar la ruta de instalación de VLC cambiando el

comando al siguiente (siendo RUTA_DESEADA la ruta real donde se desea que sea instalado):

```
./configure --prefix=RUTA_DESEADA
```

De cara al betatesting, usar la ruta predeterminada de VLC puede generar conflictos con los testers que cuenten ya con una copia oficial, por lo que más adelante se decide que a lo largo del proyecto la ruta de instalación siempre será */EMM_BETA/*.

5.1.3. Compilado para sistemas Windows

Al igual que en la sección 5.1.2, las guías oficiales están desactualizadas y pese a haberse tomado como referencia, los pasos descritos a continuación no son exactamente los mismos. La información de referencia ha sido tomada de la entrada de la bibliografía [VideoLAN Organization, 2017e].

Hay varias maneras de compilar VLC para sistemas Windows aunque se pueden reducir a dos tipos: compilación en Windows o compilación desde Linux (denominado *Cross-Compile*). Solo el método *Cross-Compile* es oficialmente compatible. Para la compilación desde Linux se usa el software *MinGW* y para la compilación desde Windows *MSYS+MinGW* o *Cygwin*. Dado que el autor sabía de la complejidad de configuración de herramientas como *MSYS* y *Cygwin* en Windows y que ya se cuenta con un entorno de desarrollo Linux, se escoge el método *Cross-Compile*.

El proceso de obtención del software necesario es similar al descrito en el apartado anterior (sección 5.1.2). Se requiere mucho del software ya mencionado, junto con la herramienta MinGW. A continuación los comandos para la obtención de *MinGW*:

Máquinas de desarrollo con arquitectura de 64 bits:

```
apt-get install gcc-mingw-w64-i686 g++-mingw-w64-i686 mingw-w64-tools
```

Máquinas de desarrollo con arquitectura de 32 bits:

```
apt-get install gcc-mingw-w64-x86-64 g++-mingw-w64-x86-64 mingw-w64-tools
```

El siguiente paso es obtener las herramientas necesarias para la compilación, muchas de las cuales ya se tendrán instaladas si se ha llevado a cabo lo descrito en la sección 5.1.2:

```
apt-get install lua5.2 libtool automake autoconf autopoint make qt4-dev-tools qt5-default git
    subversion cmake cvs wine64-development-tools libwine-dev zip p7zip nsis yasm ragel ant
    default-jdk protobuf-compiler gettext pkg-config bzip2 dos2unix
```

Hecho lo anteriormente descrito, puede procederse a la compilación en sí. Es necesario para ello entender el concepto de *Host Triplet*. El *Host Triplet* es una cadena de texto que especifica al toolchain las características tanto de la máquina compiladora como de la máquina destino. Por ejemplo, si la máquina compiladora está basada en Debian con arquitectura de 64 bits y la máquina de destino es un Windows de 32bits, el *Host Triplet* será el siguiente:

```
i686-w64-mingw32
```

Sin embargo con la misma máquina compiladora pero con una máquina de destino es un Windows de 64bits, el *Host Triplet* será el siguiente:

```
x86_64-w64-mingw32
```

El *Host Triplet* se usará en los comandos descritos a continuación. Deberán sustituirse las palabras “HOST-TRIPLET” por el texto específico que describa la máquina objetivo deseada.

Si bien en la compilación para sistemas Linux las librerías podían obtenerse desde un repositorio, en el caso de Windows es necesario compilar cada una de ellas. Para ello se usa el método “Contrib” mencionado en la sección 5.1.2. Existen dos variantes de dicho método: *Prebuilt* o *Manually built*.

La variante “Prebuilt” es significativamente más rápida pero cuenta con la limitación de que es únicamente compatible con la última versión estable (2.2.x) y por lo tanto no vale para versiones antiguas ni de desarrollo. Dicho método, por lo por lo tanto, queda descartado para este proyecto. Los comandos necesarios serían los siguientes (ejecutándolos desde la carpeta con el código):

```
1 mkdir -p contrib/win32
2 cd contrib/win32
3 ../bootstrap --host=HOST-TRIPLET
4 make prebuilt
```

La variante “Manually built” es lenta y potencialmente problemática. El proceso puede

fallar por diversos motivos y encontrar una solución puede ser complicado, ya que los errores los darán las propias librerías. Los comandos necesarios serían los siguientes:

```
mkdir -p contrib/win32
cd contrib/win32
../bootstrap --host=HOST-TRIPLET
make fetch
make
```

En el proyecto este punto generó muchos problemas ya que la compilación no llegaba a completarse correctamente y por lo tanto el siguiente paso (la compilación del propio VLC) no pudo finalizarse. Una vez se sobrepasó el umbral de las 20 horas invertidas y habiendo conseguido satisfactoriamente la compilación para sistemas Linux, se decidió eliminar la compilación de la versión de Windows del alcance del TFG y consecuentemente del programa de betatesting. Esto generó la necesidad de gestión adicional en la fase de betatesting (descrito en la sección 6). Pese a no completarse con éxito, en esta sección se terminará la descripción del proceso.

Una vez se cuenta con las librerías, el siguiente paso es la compilación del propio VLC. Sin embargo se han de hacer unas pequeñas correcciones antes. Las máquinas de compilación de 64 bits deben borrar ciertos ficheros:

```
rm -f ../i686-w64-mingw32/bin/moc ../i686-w64-mingw32/bin/uic ../i686-w64-mingw32/bin/rcc
```

También es necesario crear un enlace simbólico a las librerías compiladas:

```
ln -sf 'HOST-TRIPLET' ../i686-w64-mingw32
```

Completado esto se procede a volver a la carpeta raíz del proyecto y se prepara el árbol de ficheros:

```
cd -
./bootstrap
```

A continuación se configura la compilación (proceso equivalente a `./configure` usado en la compilación para Linux):

```
mkdir win32 && cd win32
export PKG_CONFIG_LIBDIR=\$HOME/vlc/contrib/HOST-TRIPLET/lib/pkgconfig
../extras/package/win32/configure.sh --host=HOST-TRIPLET --build=x86_64-pc-linux-gnu
```

El último comando requiere tanto del *Host Triplet* como de las características del entorno de compilado. El código provisto es para máquinas Linux de 64 bits.

Finalmente se ha de compilar y empaquetar el software:

```
make
make package-win-common
```

Esto debería generar un directorio llamado *vlc-x.x.x* (dependiendo de la versión que se haya compilado) con los binarios de VLC ya ejecutables.

5.1.4. Ejecución e instalación

Una vez completado el proceso de compilación, basta con ejecutar el siguiente comando para ejecutar VLC, estando en la carpeta que contiene el código:

```
1 ./vlc
```

También se puede instalar el software para que no sea necesario desplazarse a la carpeta a la hora de ejecutarlo. El comando necesario es el siguiente:

```
sudo make install
```

Finalizada la instalación, puede ejecutarse la copia de VLC con:

```
vlc
```

VLC cuenta con múltiples opciones que se le pueden añadir a la hora de ejecutarlo. La gran mayoría no son de interés para el contexto de este proyecto, pero hay uno que sí es extremadamente útil. Dicha opción activa el modo debug, con el cual se pueden ver en la terminal todos los mensajes de debug y errores leves (los graves siempre se muestran). Esto es esencial para encontrar la causa de posibles errores tanto en el desarrollo, como en la fase de testing. Independientemente de si está instalado o no (la diferencia será si el comando lleva por delante "./" o no), puede ejecutarse de la siguiente manera:

```
vlc -vvv
```

5.1.5. Creación de un módulo Hello World

A continuación se describen los pasos necesarios para implementar un módulo básico en VLC.

5.1.5.1. Código

Tomando como referencia la guía (altamente desactualizada) disponible en la entrada de la bibliografía [VideoLAN Organization, 2017a], se procede a crear un módulo básico cuya única funcionalidad sea escribir en la terminal el mensaje "Hello World".

Los módulos están escritos en C o C++, dependiendo de sus necesidades. Para este ejemplo se usará C. El fichero con el código ha de empezar con un descriptor del módulo, en el que se declare su nombre, descripción, tipo de funcionalidad (o "capability"), categoría en la que se clasifica y posibles variables de entrada que se quiera usar:

```
vlc_module_begin()
    set_shortname(N_("Hello"))
    set_description(N_("Hello interface"))
    set_capability("interface", 0)
    set_callbacks(Open, Close)
    set_category(CAT_INTERFACE)
    add_string("hello-who", "world", "Target", "Whom to say hello to.", false)
vlc_module_end ()
```

El siguiente paso es crear los métodos a ejecutar en la apertura y cerrado del módulo:

```
1 static int  Open ( vlc_object_t * ){ [...] };
2 static int  Close ( vlc_object_t * ){ [...] };
```

El código completo puede ser encontrado en la sección del anexo B.1.

5.1.5.2. Declaración

Es necesario declarar el fichero de manera que sea compilado junto con VLC. Se ha de encontrar el fichero *Makefile.am* (el cual es la plantilla en base a la que se creará el fichero *Makefile* cuando se ejecute el comando `./configure`) en el subdirectorio en el que se vaya a situar el módulo. Esto dependerá de su función. Para este ejemplo consideraremos que el módulo Hello creado será de tipo "control", por lo que el código se situará en

la carpeta *./modules/control/hello/* y el fichero *Makefile.am* a editar será el situado en *./modules/control/*. Se deben añadir las siguientes líneas:

```
libhello_plugin_la_SOURCES = control/hello/hello.c
libhello_plugin_la_CFLAGS = $(AM_CFLAGS)
libhello_plugin_la_LIBADD = $(AM_LIBADD)
libhello_plugin_la_DEPENDENCIES =
# Always compile the hello module:
control_LTLIBRARIES += libhello_plugin.la
```

El módulo Hello no requiere de librerías adicionales, por lo que con la configuración mencionada es suficiente. Sin embargo, si el módulo requiriese de nuevas librerías, dicha dependencia debería ser configurada en el fichero *Makefile.am* y la compilación del módulo debería ser añadida al fichero *./Makefile.am* encontrado en la raíz del proyecto.

5.1.5.3. Compilado y Ejecución

Si los anteriores pasos se han seguido correctamente, para compilar bastará con seguir los pasos descritos en la sección 5.1.2. A continuación un resumen de los comandos:

```
./configure
make
```

Finalmente el módulo puede ser ejecutado con el siguiente comando:

```
vlc --intf hello
```

5.1.6. Resumen

En esta fase se ha conseguido la versión compilada de VLC en su última rama de desarrollo para sistemas Linux y se ha prescindido de la versión para Windows. También se ha conseguido crear con éxito un módulo básico dentro del árbol de VLC. El coste temporal ha sido de aproximadamente 28 horas de 20 planificadas, habiendo requerido casi un 50 % más del tiempo planificado y habiendo tenido que eliminar partes del alcance.

Con los conocimientos adquiridos, puede procederse a desarrollar el módulo.

5.2. Segunda Fase - de UPnP a fanTAGstic

A continuación se describe la fase 2 del proyecto, en la que se comienza a implementar un módulo avanzado para VLC.

5.2.1. Análisis

Una vez se cuenta con la capacidad de compilar VLC y se comprende el funcionamiento básico de los módulos se procede a analizar las herramientas provistas por VLC. Se comienza por buscar las funcionalidades de streaming disponibles a través de la interfaz gráfica. Tras un vistazo general se observa que cuenta con capacidad de consumir y emitir muchos protocolos de streaming, pero las funcionalidades con UPnP son muy reducidas. En la figura 5.1 pueden verse los protocolos disponibles.

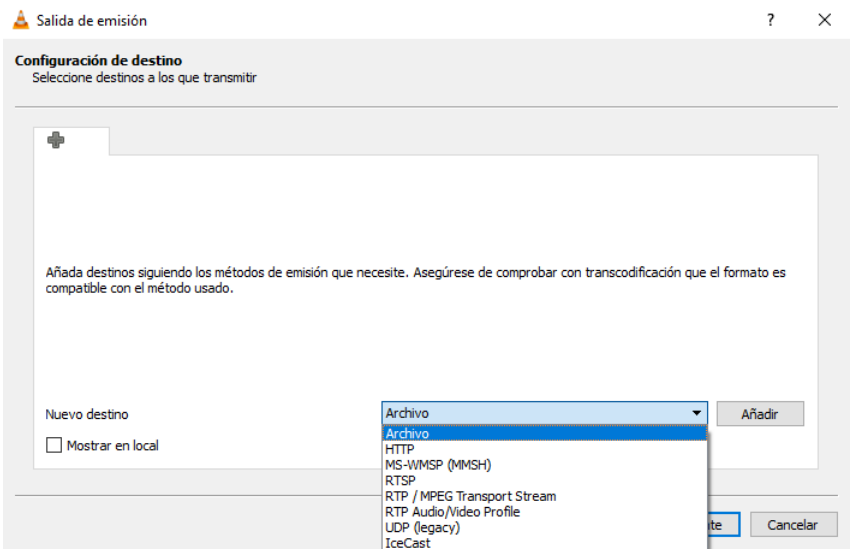

Figura 5.1: Protocolos de Streaming disponibles en VLC.

El análisis de la API de VLC lleva a una conclusión similar. VLC es capaz de emitir contenidos de manera abierta y es capaz de consumir multimedia de servidores tanto de la área de red local como de internet, pero no tiene implementado ningún método para emitir y controlar dispositivos UPnP. Es posible buscar dispositivos de red y obtener su información, pero los pasos de enlace, emisión y control no están implementados.

Lo anterior conllevaría que la tarea de crear una interfaz para facilitar el streaming a dispositivos UPnP en red requiere en realidad de dos módulos: uno capaz de gestionar y

administrar las conexiones y otro que haga de interfaz gráfica de la primera. Dado que el protocolo UPnP es complejo y de muy bajo nivel, el módulo de control y gestión por sí solo es una tarea ardua que podría no llegar a completarse en el tiempo disponible, además de contar con el handicap de tener que buscar librerías que el equipo de VLC pueda probar para su uso oficial.

Siendo las posibilidades de fracaso del proyecto relativamente elevadas y no viendo necesidad de ceñirse a la idea original del módulo, se decide cambiar la funcionalidad del módulo a un gestor de metadatos para ficheros de audio extendido, capaz de procesar múltiples ficheros al mismo tiempo y hacer una búsqueda de los metadatos en bases de datos online.

El siguiente paso a seguir es conseguir que un módulo básico como el mencionado en la sección 5.1.4 muestre una ventana. Una vez más, no habiendo documentación oficial sobre las opciones disponibles, se ha de buscar la información por otro medios. Existe un canal IRC en la red Freenode a través del cual se puede contactar con miembros tanto del equipo como de la comunidad de desarrolladores involucrados en el proyecto. Los datos necesarios para acceder a dicho canal pueden encontrase a continuación:

```
Server: irc.freenode.org
Cannel: #videolan
URL: https://webchat.freenode.net
```

También existe un foro en el que se pueden publicar dudas tanto de usuario como de desarrollo. Se publica una pregunta breve acerca de las opciones disponibles pero tras una semana de espera , se decide probar por el canal IRC. El resultado no es mucho mejor. El canal está mayormente en silencio, recibiendo mensajes automatizados sobre las compilaciones automáticas del servidor oficial de VLC. Solo esporádicamente se intercambian mensajes entre personas y por lo general los usuarios conectados solo responden a mensajes dirigidos directamente a ellos. Se procura obtener respuesta sin enviar demasiados mensajes, para que no sea considerado spam. Finalmente, después de haber solicitado la información más de 10 veces, un usuario anónimo responde con un texto muy breve, dando a entender que las opciones son las siguientes: API dialog, API LUA o Qt. También recomienda evitar Qt por ser más complejo.

Se procede entonces a analizar las tres opciones. La *API dialog* es sencilla y permite crear pequeñas ventanas, pero está mayormente enfocada a intercambios de información breve

como mensajes, preguntas de "sí o no" o solicitud de credenciales. Tras hacer una pequeña prueba de concepto se ve en seguida que no es una opción válida.

Descartada la primera se continúa analizando tanto la *API* para *LUA* como *Qt*. Qt es un framework específicamente diseñado para la creación de interfaces, con kits de desarrollo muy completos y una wiki extensa, sin embargo apenas se puede encontrar información sobre cómo hacer que *Qt* y VLC interactúen. Sobre la *API LUA* sin embargo sí se encuentra más información.

LUA es un idioma de scripting que se ejecuta fuera de VLC, pero interactúa con él a través de una API específica para dicho idioma. La API en cuestión cuenta con múltiples métodos para la creación de elementos gráficos, desde ventanas completas hasta listas y elementos formados con HTML, pasando por botones y cuadros de texto. El módulo no ha de ser registrado para su compilación por ser un idioma de scripting.

Contando con poca información sobre Qt (y la recomendación del usuario anónimo del canal IRC de evitarlo) y habiendo visto múltiples ventajas del LUA y su API, se toma la decisión de desarrollar el módulo en LUA. Viendo otros módulos LUA existentes, puede observarse que los autores tienden a poner nombres personalizados a sus módulos y no se ciñen a la seriedad de una entrada de menú estándar, por lo que decide darle al módulo un nombre de manera provisional: **fanTAGstic**. El nombre es un juego de palabras entre la palabra *fantastic* (fantástico en inglés) y la palabra *tag* (etiqueta), que es el nombre coloquial para los metadatos de ficheros de audio.

5.2.2. Aprendizaje

En esta sección se describe el aprendizaje adicional requerido para completar la fase.

5.2.2.1. Creación de un módulo LUA

No existe una guía oficial de cómo desarrollar un módulo LUA, sin embargo existen pequeñas piezas de información distribuidas en el repositorio oficial de VLC, además de algunos módulos ya implementados. Para este proyecto ha hecho falta analizar el código ya existente y deducir su funcionamiento para poder replicarlo. A continuación se describen los pasos esenciales a la hora de crear un módulo con LUA que se han descubierto.

De manera similar al módulo Hello.c (sección 5.1.5) se ha de comenzar poniendo una descripción del módulo a crear. A continuación, un ejemplo:

```
function descriptor()
  return { title = 'fanTAGstic',
           version = '0.1',
           author = 'Asier Santos',
           capabilities = {}   }
end
```

El código mostrado define la función indispensable `descriptor()`, que devolverá a VLC la información de título (en este ejemplo *fanTAGstic*), la versión (*0.1*), el autor (*Asier Santos*) y las capacidades específicas del módulo (en este caso ninguna).

El siguiente paso es crear las funciones básicas para su lanzado y cerrado:

```
function activate()
  [...]
end

function close() --Closing [triggered by clicking 'X']
  [...]
end

function deactivate()
  close()
end
```

El código contenido en `activate()` será ejecutado cuando se abra el módulo desde la GUI de VLC. `close()` se ejecutará cuando la aplicación se cierre adecuadamente, como por ejemplo pulsando en la "X" de la ventana si es una interfaz gráfica, mientras que `deactivate()` se ejecutará si se cierra debido a un error o si VLC es cerrado durante el uso del módulo. Dado que en las dos situaciones anteriores se acostumbra llevar a cabo las mismas acciones puede hacerse que el uno llame al otro para ahorrar líneas, tal y como se puede ver en la función `deactivate()`.

El objetivo de este proyecto es crear una interfaz visual, por lo que se requiere que el módulo cree una ventana. El momento correcto para hacer esto es durante la carga, por lo que habrá que modificar `activate()`:

```
function activate()
  dlg = vlc.dialog('TITULO')
end
```

De esta manera VLC creará la ventana con el texto "TITULO" en la barra superior. En la variable `dlg` se contiene la ventana, a la cual se le pueden añadir elementos o widgets para

completar la interfaz. Esto se obtiene usando la API de VLC para LUA, la cual sí tiene una pequeña guía, aunque está desactualizada. Puede ser encontrada en el repositorio oficial en la ruta *vlc/share/lua/README.txt* [VideoLAN Organization, 2017b]. A modo de ejemplo, se puede añadir un botón añadiendo una línea de código:

```
function activate()
  dlg = vlc.dialog('TITULO')
  dlg:add_button('TEXTO', funcion, 1, 1, 1, 1)
end
```

En este caso se creará un botón con el texto "TEXTO" que al ser pulsado lanzará la función llamada `funcion()`. Los siguientes números representan la posición en la que se encontrará el botón y su tamaño. Se han de interpretar en el siguiente orden: fila, columna, ancho y alto. Esto es aplicable a prácticamente todos los widgets disponibles a través de la API. El resultado de lo descrito puede verse en la figura 5.2.

Figura 5.2: Ejemplo de ventana en LUA usando la API de VLC.

Una vez se tiene el código bastará con poner los ficheros en la ruta *vlc/share/lua/extensions/*. Al ser un idioma de scripting, una vez se tiene VLC compilado no es necesario volver a compilarlo cada vez que se haga un cambio, aunque sí existe la posibilidad de compilar el código LUA para ganar rendimiento. Si el código no cuenta con errores significativos, podrá ser ejecutado desde una entrada de menú generada en *barra superior* → *view* → *nombre del módulo*. En caso de que VLC no sea capaz de cargarlo, la terminal mostrará una serie de mensajes de error. La entrada de menú generada por el código de ejemplo provisto puede verse en la figura 5.3.

5.2.2.2. Vinculación entre C y LUA

Una excelente descripción y guía de este tema puede ser encontrada en la documentación oficial de LUA (capítulo 24, [Roberto Ierusalimschy, 2016]). Sin embargo, a continuación, se provee una breve descripción.

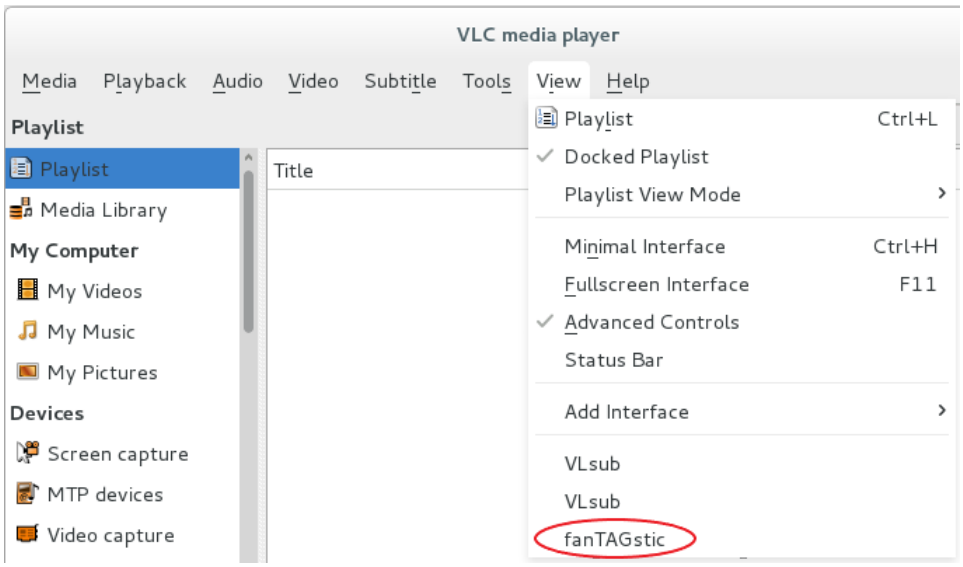

Figura 5.3: Ejemplo de entrada de menú en VLC de un módulo LUA.

LUA, por ser un idioma relativamente reciente, tiene pocas librerías disponibles, pudiendo así desalentar a los desarrolladores. Para suplir el problema se provee de una API para el idioma C. Dicha API ofrece la posibilidad de intercambiar datos entre ambos idiomas de manera sencilla, haciendo así que LUA pueda valerse de las muchísimas librerías disponibles para C con simplemente enlazar ambos idiomas.

El principal problema es la conversión de datos, ya que LUA cuenta con variables complejas que C no es capaz de reconocer de manera nativa. Esto se soluciona con una serie de funciones que convierten las variables más básicas a C obteniéndolas desde una pila provista por LUA. Las variables disponibles son las siguientes:

- Boolean
- Number (en C `double`)
- String (en C `const char *`)
- Longitud de String (en C `size_t`)

5.2.2.3. Enlazado entre LUA y C

A continuación un ejemplo práctico del uso de la API:

Es necesario crear un fichero para el código en C y registrar en él las funciones que se quieren vincular. Se comienza por dar nombre al enlace de los dos idiomas (o binding). Se requiere una función cuyo nombre sea el compuesto de `luaopen_ + rutaDelFichero`. Para este ejemplo se considerará que el código se incluye en el árbol de VLC, por lo que su nombre refleja la ruta en la que se encuentra (*share/lua/extensions/libs/foo.c*). Dentro de dicha función, se llama a la función `lua_register()`, la cual requiere de 3 entradas: la variable `lua_State` provista por la API, el nombre que tendrá la función en LUA y el nombre bajo el que está registrada en C.

```
1 int luaopen_share_lua_extensions_libs_foo (lua_State *L){
2     lua_register(
3         L,                    /* Lua state variable */
4         "fooInLUA",           /* func name as known in Lua */
5         fooInC                /* func name in this file */
6     );
7     return 0;
8 }
```

Por lo tanto, cuando se desee acceder a la función desde LUA se llamará a través de `fooInLUA()` mientras que en C la función a llamar se denominará `fooInC`. Tal y como se observa en el ejemplo, no se definen las variables que se intercambiarán. Esto se ha de gestionar dentro de `fooInC`.

5.2.2.4. Intercambio de variables

A la hora de enlazar LUA y C, LUA exporta las variables de la llamada a una pila accesible desde C. LUA no comprueba el número de variables que serán recibidas por C, por lo que es tarea del desarrollador comprobar el número, tipo y orden de las variables pasadas a la llamada. C deberá entonces tomar los elementos de la pila y convertirlas al tipo de variable adecuado.

A continuación dos secciones de código de ejemplo, una con la llamada a la función desde LUA y otra con la función de enlace en C, que analizará el intercambio de variables:

Código LUA:

```
1 require("share.lua.extensions.libs.foo") /*Import code*/
2 fooInLUA("textVar", 56)
```

Código C:

```
static int fooInC(lua_State *L){
    double number = (double)lua_tonumber(L, -1);  /*Get 2nd arg, string */
    char *text = (char*)lua_tostring(L, -2);  /*Get 1st arg, string */

    return 0;   /* no return values */
}
```

En la sección de LUA puede observarse primero cómo se importa el código en C (el cual ha debido ser compilado antes), y posteriormente se llama a la función registrada, `fooInLUA()`, pasando dos variables.

En la sección de C puede verse la implementación de la función `fooInC`, la cual solamente recibe la variable `L`, cuyo uso es el de acceder a la API de LUA. A continuación se obtienen las dos variables enviadas desde LUA (un número y una string). Los métodos `lua_tonumber` y `lua_tostring` toman el objeto `lua_State`, acceden a la pila de LUA y obtienen los objetos de las posiciones 1 (la cumbre o *top*) y 2 respectivamente. Las variables se añaden a la pila en el orden en el que son enviadas, es decir, con la llamada `fooInLUA("textVar", 2)` se consigue que la string `"textVar"` entre primero y posteriormente el valor `"56"` (una representación gráfica del proceso puede encontrarse en la figura 5.4). Al obtener los valores de la pila en C es necesario hacer un cast al tipo de dato que es (de ahí el añadir `(double)` y `(char*)`).

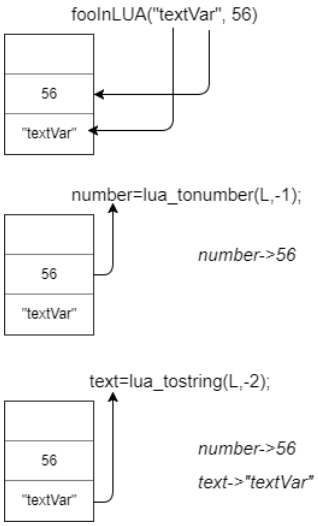

Figura 5.4: Representación de la pila de LUA en la API de C sin variable de respuesta.

Finalmente el código provisto devuelve el valor 0 (`return 0`) ya que no pretende devolver ninguna variable a LUA. Sin embargo, si no fuese el caso, debería primero meter la variable a devolver en la pila y posteriormente devolver el valor 1:

```
1 static int fooInC(lua_State *L){
2     double number = (double)lua_tonumber(L, -1);  /*Get 2nd arg, string */
3     char *text = (char*)lua_tostring(L, -2);  /*Get 1st arg, string */
4
5     lua_pushstring (L, "returnVal");
6     return 1;   /* there is a return value */
7 }
```

El código devolverá la cadena `"returnVal"` y podrá ser almacenada en una variable en LUA al hacer la llamada a la función de enlace. Del mismo modo que se ha devuelto una cadena de caracteres, se podría devolver un número con `lua_pushnumber`, un booleano con `lua_pushboolean` o el valor `NULL` (en LUA `NIL`) con `lua_pushnil`. La representación de la pila de LUA para el código con respuesta puede encontrarse en la figura 5.5.

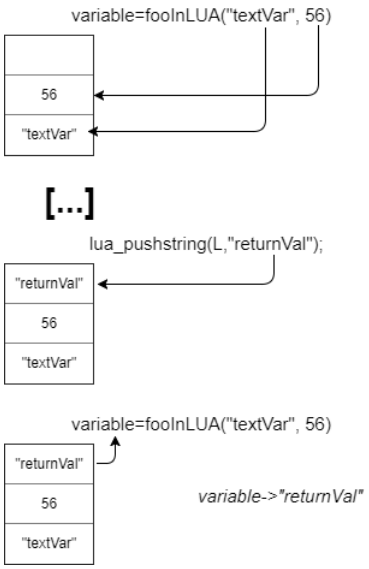

Figura 5.5: Representación de la pila de LUA en la API de C con variable de respuesta.

5.2.2.5. Uso de la API de VLC para LUA

La API para LUA es mucho más reducida que la de C, pero no parece ser insuficiente. La documentación sobre dicha API, la cual está desactualizada pero no en exceso, puede ser encontrada en la ruta *vlc/share/lua/README.txt* [VideoLAN Organization, 2017b]

del repositorio oficial. A continuación se puede encontrar una descripción de las funciones de interés para el proyecto:

5.2.2.5.1. Dialog

Permite crear ventanas, esconderlas, destruirlas, ponerles título y añadirle múltiples tipos de widgets.

5.2.2.5.2. Input

Permite crear y gestionar elementos de reproducción, los cuales contarán con los metadatos a procesar.

5.2.2.5.3. Messages

Permite enviar mensajes a los logs y a la terminal de VLC. Los mensajes disponibles son los de debug, alerta, error e información.

5.2.2.5.4. Objects

Permite obtener objetos relevantes de VLC, como su propia entidad o la lista de reproducción.

5.2.2.5.5. Playlist

Puede ser usada para gestionar la lista de reproducción, así como obtener y procesar los elementos que la conforman.

5.2.2.5.6. Strings

Facilita el procesado de información textual como URIs, URLs, XMLs, etc.

5.2.3. Implementación

Se decide crear una rama en Git para la implementación en LUA, llamada *LUA-main*, en la que se vaya incluyendo el código funcional, y ramas especificas para determinadas funciones como *LUA-setMeta_FIX* o *LUA-viaHash*. La última versión del código puede verse en `http://tiny.cc/lua_branch`.

Para familiarizarse con el desarrollo en LUA y su API de VLC, se procede a implementar una ventana básica que contengan la información principal, para más adelante plantear la mejor manera representarla (cuando el desarrollo esté en un punto más maduro). El resultado puede verse en la figura 5.6.

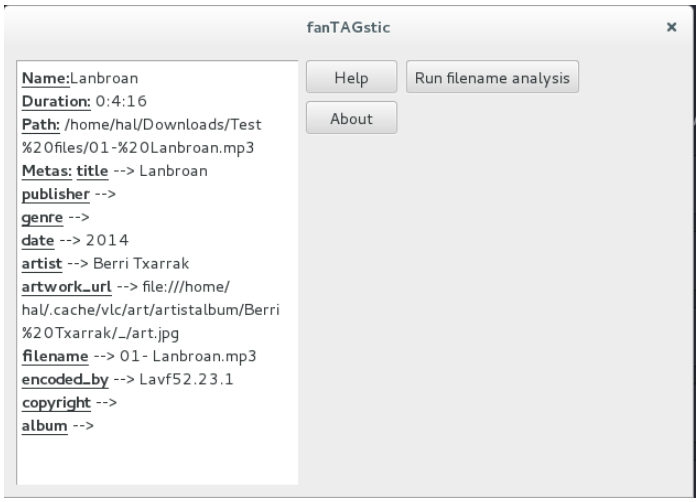

Figura 5.6: Ventana principal de *fanTAGstic* durante el comienzo de su desarrollo.

Adquiridas unas destrezas básicas en el uso de elementos para la interfaz gráfica, se decide trabajar la parte lógica con una función más compleja: la obtención de los metadatos a través del análisis del nombre del fichero. Se procede a implementar una ventana para ello (figura 5.7). La función busca los datos del artista y título de la canción en el nombre, separado por caracteres como guiones o múltiples espacios seguidos, para finalmente preguntarle al usuario cuál de los valores encontrados es el artista y cuál el título. El código específicamente encargado del análisis puede ser encontrado en el anexo B.4.

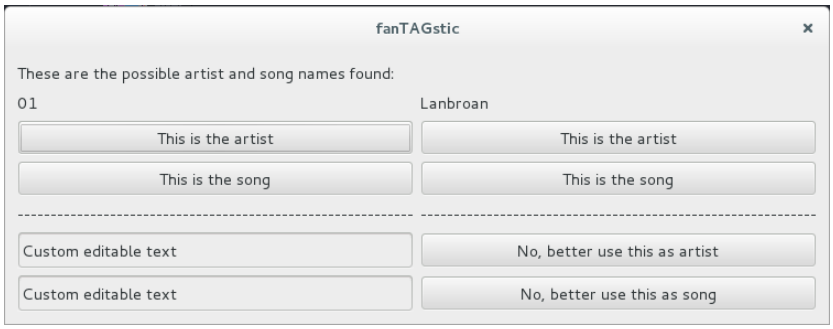

Figura 5.7: Ventana para el análisis del nombre del fichero en *fanTAGstic* durante el comienzo de su desarrollo.

Durante la implementación de la función anterior se encuentra un problema: la API de VLC para LUA permite editar los metadatos, pero no guardarlos en el fichero de origen, solo en la lista de reproducción. Esto significa que el cambio solo sería útil para usuarios que siempre hagan uso de VLC para la escucha de su música. Se decide que esto no es suficiente, por lo que se procede a obtener y usar una librería liviana para el procesado de los metadatos.

Si bien lo óptimo sería hacer uso de una librería implementada en LUA, las opciones son muy limitadas, muchas no están testeadas y son mayormente implementaciones caseras pensadas para el uso específico en determinado software. Debido a que no parece adecuado hacer depender a VLC de este tipo de librerías, se descarta la opción a favor de software implementado en C. Se opta por *id3v2lib*, una librería sencilla y sobre todo ligera capaz de procesar las etiquetas más comunes, las cuales son suficientes para este proyecto:

- Título
- Álbum
- Artista
- Artista del álbum
- Comentarios
- Año
- Número de pista
- Número del álbum
- Carátula

Para hacer uso de esta librería es necesario hacer un enlace entre LUA y C, tal y como se describe en la sección 5.2.2.2. La implementación de dicho enlace puede encontrase en el anexo B.2. Cabe mencionar que como se describirá a continuación, esta rama de desarrollo no se finalizará, por lo que el código provisto es una versión de desarrollo potencialmente incompleta. Para compilar el código provisto se requiere de un comando específico que enlace LUA y la librería *id3v2lib*:

```
gcc -Wall -shared -fPIC -o id3libBinding.so -I/usr/include/lua5.2 -llua5.2 id3libBinding.c -
    lid3v2
```

Otra limitación topada es que aunque VLC cuenta con la librería *Chromaprint*, la API de LUA no cuenta con un método para su uso, por lo que deberá ser implementado el acceso desde LUA o un enlace haciendo uso de C. *Chromaprint* incluye una herramienta de terminal llamada *fpcalc* para efectuar el proceso de *fingerprinting* (figura 5.8) cuyo uso es extremadamente sencillo, por lo que se decide intentar implementar el fingerprinting haciendo una llamada de terminal en segundo plano desde C y obteniendo el resultado, para luego ser procesado en LUA (por ser más versátil en el procesado de texto). El código utilizado puede encontrase en el anexo B.3 (una vez más, es código de desarrollo sin terminar).

Figura 5.8: Herramienta de terminal *fpcalc* en uso.

Durante el desarrollo de este método se descubre que VLC cuenta con la opción de hacer dicho fingerprinting, solo que era inestable y estaba desactivada por defecto en la configuración de la versión de desarrollo. Tras actualizar el repositorio a la última versión, la función parece funcionar correctamente, de manera más eficiente que el código implementado hasta el momento, aunque cuenta con la limitación de que solo es capaz de procesar ficheros uno a uno, haciendo muy incómodo el procesado de carpetas o listas de reproducción completas. Llegados a este punto se empieza a sospechar que la rama de desarrollo actual puede estar condenada al fracaso, por lo que se han de reevaluar las opciones.

5.2.3.1. Migrado a C++ y LUA

Contando VLC con la funcionalidad principal del módulo, para que el proyecto siga teniendo el objetivo de ser útil para los usuarios estándar y ser incluido en VLC, es necesario que el módulo tenga un valor diferencial significativo.

La manera de dar un valor diferencial al módulo es que este sea capaz de procesar y mostrar los metadatos de múltiples ficheros simultáneamente de manera sencilla. Gestionar los datos a través de una tabla parece la mejor opción, tanto por ser conceptualmente sencilla para el usuario, como por ser fácil de diseñar y gestionar desde el punto de vista del desarrollo. Tras un repaso a la API de VLC para LUA, se puede observar que no existe la opción nativa de crear tablas. Se baraja la opción de crear series de cajas de texto editables en la interfaz de manera dinámica en base al número de ficheros o una etiqueta HTML con la tabla, pero sería complejo de obtener a nivel técnico y potencialmente confuso para el usuario.

Teniendo en cuenta que el módulo ha generado nuevas dependencias para suplir las carencias de VLC y que no cuenta con la posibilidad de usar tablas, seguir con el desarrollo en LUA podría llevar el proyecto a no ser viable, o no llegar a ser lo suficientemente práctico para su incorporación a la versión oficial. Dada la situación se ha de tomar una decisión crítica de entre las siguientes:

- Arriesgarse continuando una rama cuyo éxito es improbable por sus limitaciones técnicas.
- Iniciar una rama con C++ y Qt (la cual se beneficiaría de la API de C completa y las tablas disponibles en Qt) en la que haya que volver a pasar por las fases de aprendizaje, diseño y desarrollo (aunque en menor medida), la cual tiene más potencial de éxito, pero reduce significativamente el tiempo de desarrollo.

Se decide finalmente abandonar el desarrollo en LUA, convirtiendo el código útil a C++, iniciando una nueva fase de aprendizaje, rediseñando parte del módulo y finalmente implementándolo.

En el momento en que que se decide abandonar la rama de desarrollo en LUA y pasar a Qt se ve necesario renombrar el módulo, ya que este no va a ser independiente de VLC, sino que estará integrado en sus menús. Se decide darle un nombre serio y claro, que contraste con la función ya existente en VLC: Extended Metadata Manager (o EMM). Su significado es Gestor de Metadatos Extendido.

5.3. Tercera Fase - Extended Metadata Manager

A continuación se describe la tercera y última fase del desarrollo.

5.3.1. Análisis

Siendo los requisitos, objetivos y alcance los mismos que en la fase 2 (sección 5.2), solo es necesario comprender las tecnologías C++ y Qt, junto con la manera de crear módulos gráficos en VLC. La API para C es significativamente más completa y soluciona los grandes problemas de la fase anterior (la imposibilidad de guardar metadatos a través de la API y no poder usar la API para el *fingerprinting*). Una pequeña modificación del diseño original es suficiente para poder aplicarlo en esta fase.

5.3.2. Aprendizaje

En esta sección se describe el aprendizaje adicional que se ha necesitado para completar la fase.

5.3.2.1. Creación de un módulo de interfaz gráfica

Tal y como se ha mencionado antes en este documento, la mayoría de la documentación de VLC está desactualizada o es simplemente inexistente. Eso mismo ocurre en el ámbito del desarrollo de módulos Qt: no existe ningún documento explicando el proceso, lo que ha conllevado que tenga que ser deducido aplicando ingeniería inversa al código existente. Se busca un ejemplo de módulo de interfaz lo más sencillo posible (*gototime*, disponible en la ruta *vlc/modules/gui/qt/dialogs* del repositorio oficial [VideoLAN Organization, 2016a]) y se procede a imprimir el código para poder tomar notas sobre el objetivo de cada apartado. En la figura 5.9 pueden observarse las hojas impresas junto con las notas tomadas.

Los módulos gráficos están relacionados con muchos más ficheros que los módulos estándar, ya que se han de registrar en los menús en los que se desea que aparezcan, el generador de diálogos de VLC debe saber de su existencia y a menudo usan imágenes y texto que ha de ser traducido. Dichas relaciones no están definidas en la documentación y se han tenido que deducir buscando en todo VLC palabras clave como los nombres de los ficheros, las variables no declaradas dentro del fichero y el código importado en los

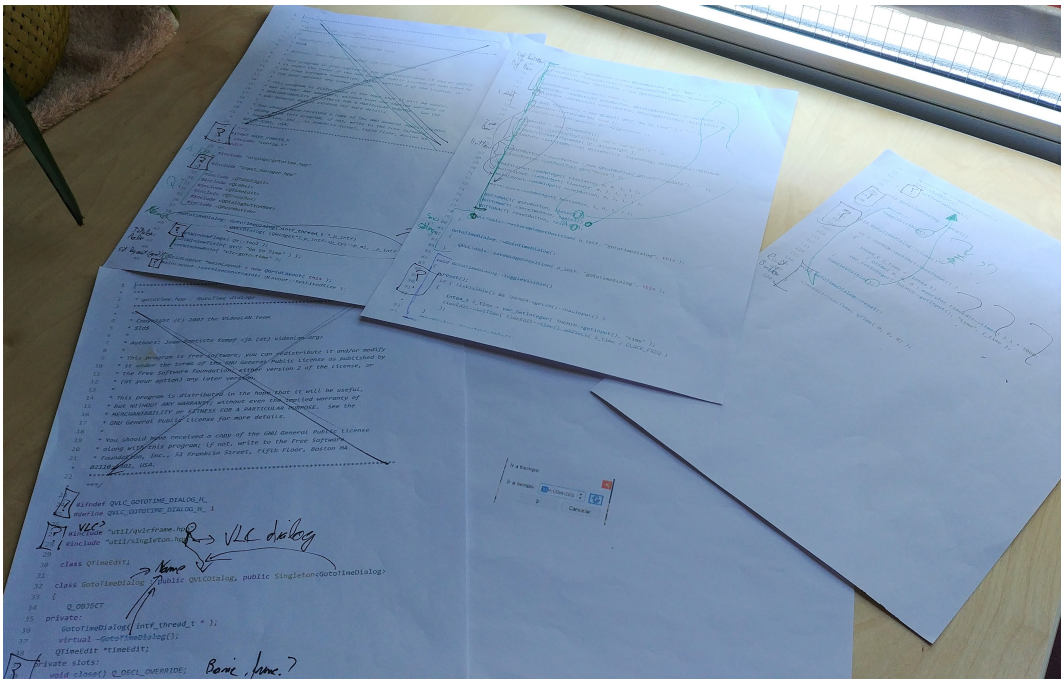

Figura 5.9: Código impreso del módulo *gototime* tras ser analizado.

ficheros del módulo. A modo de ejemplo, se incluye en la figura 5.10 un gráfico con los ficheros relacionados con el proyecto desarrollado. Cabe mencionar que no todo lo representado en 5.10 es obligatorio, pero sí sigue el orden que parece haber sido establecido por el equipo de VLC.

La mayoría de los ficheros relacionados con las interfaces creadas en Qt se encuentran en la ruta *vlc/modules/gui/qt* del repositorio oficial. Lo primero es crear los ficheros para el módulo, en este caso `extendedmetamanager.cpp` y `extendedmetamanager.hpp` (*cpp* para el código y *hpp* para los headers). Ambos ficheros contendrán la información de la ventana de diálogo, por lo que se situarán en el subdirectorio *dialog/*. De manera similar a lo descrito en la creación de un módulo LUA (sección 5.2.2.1), el código deberá contener al menos las siguientes funciones:

`extendedmetamanager.cpp`:

```
1 /* Constructor */
2 ExtMetaManagerDialog::ExtMetaManagerDialog( intf_thread_t *_p_intf)
3                 : QVLCDialog( (QWidget*)_p_intf->p_sys->p_mi, _p_intf ) {
4     QVLCTools::restoreWidgetPosition( p_intf, "ExtMetaManagerDialog", this );
5 }
6
7 /* Destructor */
```

```
8  ExtMetaManagerDialog::~ExtMetaManagerDialog() {
9      QVLCTools::saveWidgetPosition( p_intf, "ExtMetaManagerDialog", this );
10 }
11
12 /* Just closes the window (and the module itself) */
13 void ExtMetaManagerDialog::close()
14 {
15     toggleVisible();
16 }
17
18 /* Show/hide */
19 void ExtMetaManagerDialog::toggleVisible() {
20     QVLCDialog::toggleVisible();
21     if(isVisible()) //If changed to shown
22         activateWindow();
23 }
```

`extendedmetamanager.hpp`:

```
1  #ifndef QVLC_EXTMETAMANAGER_DIALOG_H_
2  #define QVLC_EXTMETAMANAGER_DIALOG_H_ 1
3
4  #include "util/qvlcframe.hpp"
5  #include "util/singleton.hpp"
6
7  class ExtMetaManagerDialog : public QVLCDialog, public Singleton<ExtMetaManagerDialog>
8  {
9      Q_OBJECT
10 private:
11     ExtMetaManagerDialog( intf_thread_t * );
12     virtual ~ExtMetaManagerDialog();
13 private slots:
14     void close() Q_DECL_OVERRIDE;
15     friend class    Singleton<ExtMetaManagerDialog>;
16 public:
17     void toggleVisible();
18 };
19
20 #endif
```

El código provisto define un constructor que se lanzará al inicio (el cual deberá encargarse de crear la interfaz), un destructor genérico y los eventos de cierre y ocultado/mostrado de la ventana, todo ello bajo la clase `ExtMetaManagerDialog`. El constructor y destructor cargan y almacenan respectivamente la posición de la ventana, para que tras haber sido cerrada, vuelva a abrirse en la misma posición y con el mismo tamaño. El módulo usa la clase `QVLCDialog`, provista por la API de VLC, sin embargo no es la única opción, también existe `QVLCFrame`. Mientras que `QVLCDialog` es una ventana en sí, `QVLCFrame`

es solo un widget y es más adecuado para paneles pequeños y mensajes puntuales, lo cual es muy escaso para la funcionalidad a implementar.

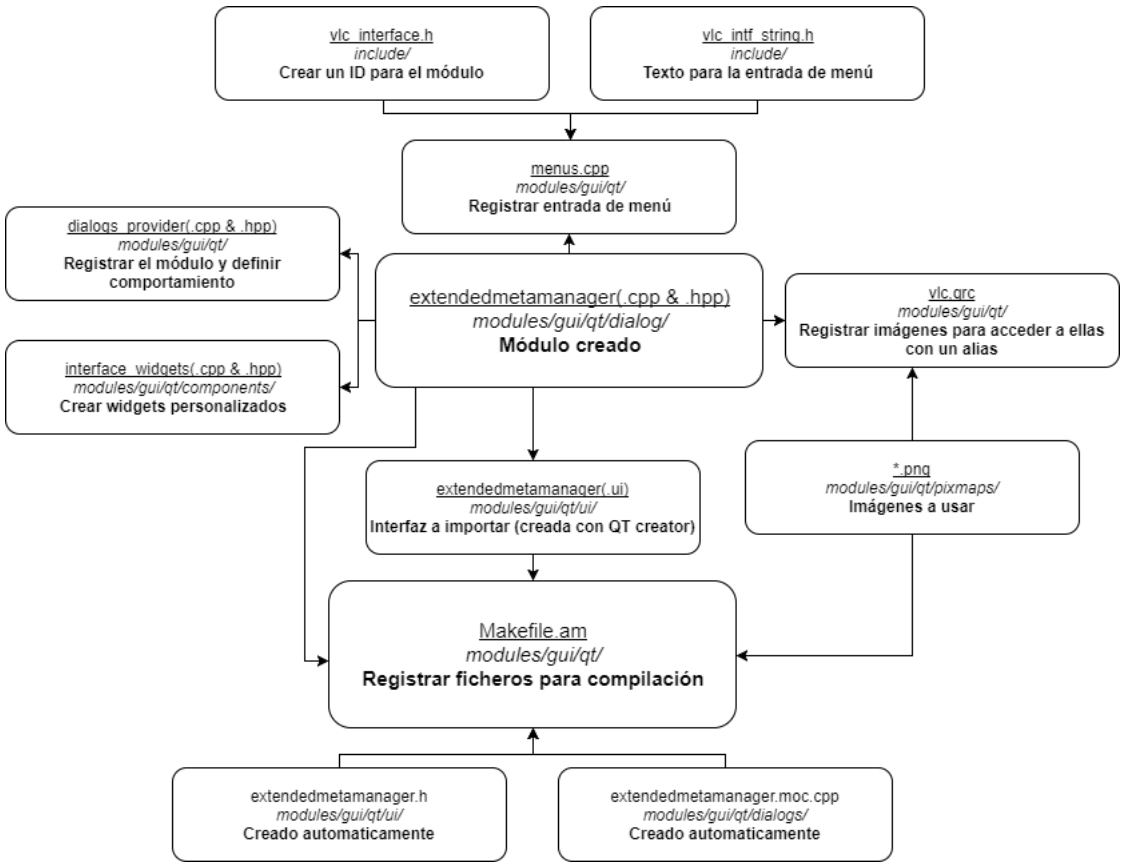

Figura 5.10: Ejemplo de dependencia de ficheros de un módulo GUI.

El código provisto ha de ser registrado en el fichero `Makefile.am` (en la ruta *vlc/modules/gui/qt*) para que sea compilado con el resto de VLC. Dicho fichero es muy extenso y un error podría llevar a que el proyecto no pueda ser compilado. Se ha de buscar la sección `libqt_plugin_la_SOURCES` y añadir el siguiente código (respetando las líneas ya existentes):

```
1 [...]
2 libqt_plugin_la_SOURCES = \
3     dialogs/extendedmetamanager.cpp dialogs/extendedmetamanager.hpp \
4     [...]
```

El siguiente paso es añadir elementos a la ventana. Esto puede conseguirse manualmente, escribiendo el código para cada elemento o widget en el programa de ejemplo provisto, pero tal y como se ve en el boceto de la interfaz (figura 4.1), el módulo va a contar con

muchos elementos, lo que conlleva que el código será muy denso y difícil de gestionar. Sin embargo, existe la alternativa de usar una de las herramientas de desarrollo para el framework Qt como *Qt Creator* o *Qt Designer* y crear la interfaz gráficamente (figura 5.11).

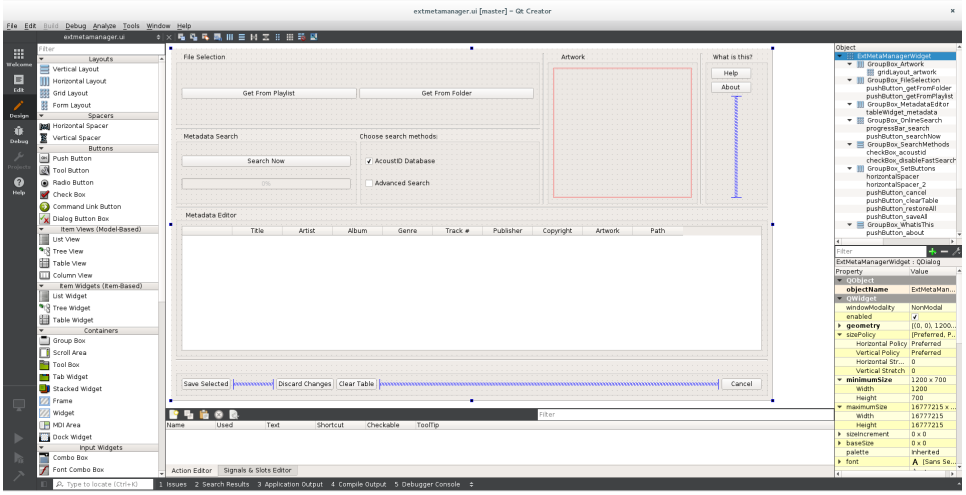

Figura 5.11: Herramienta Qt Creator.

Estas herramientas crean un fichero `.ui` que ha de ser compilado para obtener un fichero header compatible con C++. El fichero `.ui` se ha de situar en la ruta *vlc/modules/gui/qt/ui*. VLC cuenta con la capacidad de efectuar dicha conversión, si se añaden algunas líneas al fichero `Makefile.am`:

```
1 [...]
2 nodist_libqt_plugin_la_SOURCES = \
3     dialogs/extendedmetamanager.moc.cpp \
4     [...]
5
6 nodist_libqt_plugin_la_SOURCES += \
7     ui/extmetamanager.h \
8     [...]
```

El fichero `extendedmetamanager.moc.cpp` contendrá el código compilado obtenido del `.ui`. El compilador entrará en la ruta *vlc/modules/gui/qt/ui*, compilará todo lo encontrado a ficheros `.h` (de ahí la línea 7 de la sección anterior) y producirá el fichero `.moc.cpp`.

Editando los ficheros `extendedmetamanager.cpp` y `extendedmetamanager.hpp` se puede incorporar la interfaz creada a la ventana ya existente:

`extendedmetamanager.cpp`:

```
* Constructor */
ExtMetaManagerDialog::ExtMetaManagerDialog( intf_thread_t *_p_intf)
               : QVLCDialog( (QWidget*)_p_intf->p_sys->p_mi, _p_intf )
{
    msg_Dbg( p_intf, "[EMM_Dialog] Initializing" );

    /* Basic UI setup */
    ui.setupUi( this ); //setup the UI from de compiled (.h) version of de QT ui (.ui)
    setWindowFlags( Qt::Tool );
    setWindowRole( "vlc-ext-meta-manager" );
    setWindowTitle( qtr( "Extended Metadata Manager" ) );
}
[...]
```

`extendedmetamanager.hpp`:

```
#include "ui/extmetamanager.h" // Include the precompiled version of extmetamanager.ui
[...]
```

Lo anterior es suficiente para la creación del propio módulo, sin embargo para que aparezca en los menús de VLC será necesario editar el fichero `menus.cpp` (en la ruta *vlc/modules/gui/qt*). En este caso el módulo se puede considerar una "herramienta" dentro de VLC, por lo que se añadirá al menú *tools*:

```
[...]
QMenu *VLCMenuBar::ToolsMenu( intf_thread_t *p_intf, QMenu *menu )
{
    addDPStaticEntry( menu, qtr(I_MENU_EXTENDED_METADATA) , ":/EMM-icon",
        SLOT( ExtMetaManagerDialog() ), "Ctrl+Shift+I" );
[...]
```

El ejemplo registra el texto de la entrada de menú (`qtr(I_MENU_EXTENDED_METADATA)`, que será explicado más adelante), el icono para la entrada (`":/EMM-icon"`, que será también explicado más adelante), la ventana que ejecutará (`ExtMetaManagerDialog()`) y el atajo de teclado a través del cual se podrá acceder (`Ctrl+Shift+I`). El texto de la entrada, pese a poder escribirse en el mismo fichero `menus.cpp`, viendo otros ejemplos se puede ver que acostumbra a registrase en otro fichero, el cual contiene el texto de todos lo menús. Dicho fichero es `vlc_intf_string.h` en la ruta *vlc/include/*:

```
[...]
/****************** Menus *****************/
#define I_MENU_EXTENDED_METADATA  N_("Extended Metadata Manager")
[...]
```

También es necesario darle un identificador al módulo para el gestor de diálogos de VLC en el fichero `vlc_intf_string.h`, en la ruta *vlc/include/*:

```
[...]
typedef enum vlc_intf_dialog {
    [...]
    INTF_DIALOG_METAMANAGER,
[...]
```

Llegados a este punto el código se podría compilar y ejecutar desde VLC, sin embargo es común que las interfaces hagan uso de imágenes o iconos, tal y como se ha hecho en la entrada de menú. Las imágenes a usar deben situarse en la ruta *vlc/modules/gui/qt/pixmaps* y ser registradas en los ficheros `Makefile.am` y `vlc.qrc` (ambos en *vlc/modules/gui/qt*). Para este ejemplo supondremos que se ha añadido el fichero `EMM-icon.png` (figura 4.3) en la ruta *vlc/modules/gui/qt/pixmaps*:

`Makefile.am`

```
[...]
DEPS_res = \
    pixmaps/EMM-icon.png \
[...]
```

`vlc.qrc`

```
[...]
<qresource prefix="/">
    <file alias="EMM-icon">pixmaps/EMM-icon.png</file>
    [...]
</qresource>
[...]
```

El cambio hecho en `vlc.qrc` permite que se pueda acceder a la imagen a través del alias ":/EMM-icon" (tal y como se ha hecho en el fichero `menus.cpp`). Finalmente el módulo puede ser compilado (siguiendo el mismo proceso que en la sección 5.1.2) y usado a través de la entrada de menú correspondiente (figura 5.12).

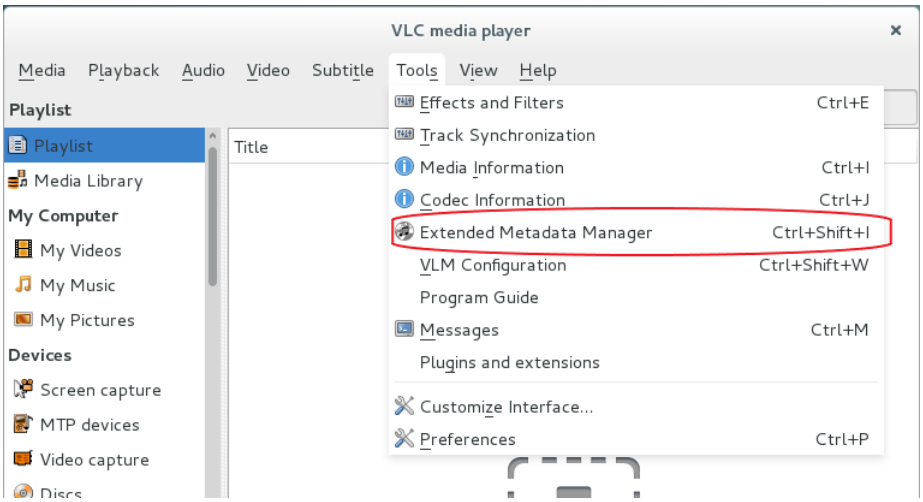

Figura 5.12: Entrada de menú para el Gestor de Metadatos Extendido.

5.3.2.2. Uso de la API de VLC

La API de VLC para C y C++ es muy extensa y la mayoría no es de utilidad para este proyecto, sin embargo en esta sección se describirán los apartados que ha sido indispensable entender para completar el desarrollo. La información de la API puede ser encontrada en la ruta *vlc/include*, en una serie de ficheros que siguen el patrón `vlc_***.h`

5.3.2.2.1. Arrays

Si bien el módulo se va a desarrollar en C++, las principales librerías incluidas en dicho idioma no son usadas por VLC, ya que requieren de grandes cantidades de memoria y por lo general se pueden solucionar con estructuras más simples y ligeras. Un ejemplo claro es el de las listas y vectores. Ambas estructuras son muy prácticas, pero solo C++ se puede beneficiar de ellas (y teniendo en cuenta que la mayoría de VLC está en C, significa hacer un uso grande de memoria para una sección pequeña). Para solucionar esto, VLC pone a disposición del desarrollador una implementación sencilla de una lista capaz de guardar cualquier tipo de objeto y con métodos para buscar o eliminar tanto a través de un índice como del objeto en sí.

5.3.2.2.2. Dialog

Permite crear pequeñas ventanas para alertas de texto, preguntas sencillas y petición de

credenciales. En este proyecto solo se usa para crear unas ventanas de ayuda (“Help”) y de información adicional (“About”) .

5.3.2.2.3. Input item

Para facilitar el uso de distintos elementos de reproducción (audio o vídeo; local o remoto; raw o comprimido; etc.) VLC ofrece el objeto `input_item_t`, el cual contiene toda la información disponible sobre el elemento. Dicho objeto se usa para interactuar con casi todas las funciones de la API. El objeto puede contar con información como los metadatos del fichero, su lugar de origen, imágenes asociadas, si está en la lista de reproducción, opciones adicionales de reproducción, etc. La API ofrece métodos para la creación, gestión, edición y borrado de los `input_item_t`.

5.3.2.2.4. Meta

Para la gestión de los metadatos de los objetos `input_item_t` VLC pone a disposición la estructura `vlc_meta_t` y una serie de getters y setters para todos los tipos de metadatos disponibles (no limitándose a los ficheros de audio, aunque en este proyecto hayan sido los únicos usados). También permite la eliminación, creación y fusión de distintos metadatos.

5.3.2.2.5. Metadata Request

Al crear un objeto `input_item_t` desde un fichero, es necesario obtener los metadatos en un paso adicional con un proceso denominado *preparsing*. Para eso VLC ofrece el método `libvlc_MetadataRequest`, el cual se encargará de obtener los metadatos y guardarlos en el objeto `input_item_t` que se le provea.

5.3.2.2.6. Playlist

VLC mantiene siempre actualizado un objeto `playlist_item_t`, el cual contiene la información sobre los elementos disponibles en la lista de reproducción. Los elementos cargados en la lista pasan automáticamente por el proceso de *parsing*, por lo que no es necesario hacerlo manualmente. La lista está implementada a modo de array de nodos que pueden a su vez tener sub-nodos, siendo cada uno un elemento de reproducción (pudiendo ser también elementos remotos obtenidos de servidores multimedia).

5.3.2.2.7. Fingerprinter

Esta sección de la API es una implementación de la librería `chromaprint` para la identificación de pistas de audio a través del servicio AcoustID (una descripción extensa de esto puede ser encontrada en la sección 4.2). Ofreciendo a la API un objeto `input_item_t`, ésta procederá a crear la cadena identificadora, buscarla en la base de datos y obtener los metadatos disponibles. En el momento de redacción de esta memoria la API solo obtiene los datos del título y artista del fichero de audio, sin embargo la devolución de los metadatos se hace a través del objeto `input_item_t`, lo que significa que si la API es extendida en el futuro, el código del módulo no tendrá que ser editado para funcionar con la nueva información disponible.

5.3.2.2.8. Messages

VLC cuenta con sus propios registros durante la ejecución del programa, los cuales pueden ser volcados a un fichero o mostrados en la terminal para ayudar a identificar problemas con el software. Para ello pone a disposición del desarrollador la posibilidad de registrar mensajes en dichos registros. Los mensajes pueden ser de error, advertencia, debug, información o carácter genérico. Esto ha sido especialmente útil durante el beta-testing, al poder crear un registro de lo que el tester ha hecho para posteriormente poder reproducir los errores que haya podido encontrar.

5.3.3. Implementación

Al igual que en el apartado análogo en la sección 5.2.3, se crea una rama principal en Git para el desarrollo (*EMM-main*) y ramas para funciones específicas como *EMM-fingerprinting* o *EMM-GUI*.

Dado que el código a desarrollar va a estar mucho más integrado en VLC que el que se desarrolló en la fase 2 (sección 5.2), se ve necesario ceñirse más a las guías de estilo de escritura de código provistas por VLC [VideoLAN Organization, 2017c]. El propio documento menciona que las normas descritas ya no son obligatorias y que prima la coherencia dentro del mismo archivo sobre las normas per se.

Aprendiendo de la fase 2, se decide mejorar el seguimiento de las tareas de desarrollo. Si bien el registro formal se sigue haciendo a través de la herramienta PrimaERP, para la gestión del desarrollo (pequeñas sub-tareas, errores a solucionar y otros datos de interés)

se incorpora el uso de una pizarra magnética junto con tarjetas de papel y post-its que facilitan la gestión y reorganización de los elementos (figura 5.13).

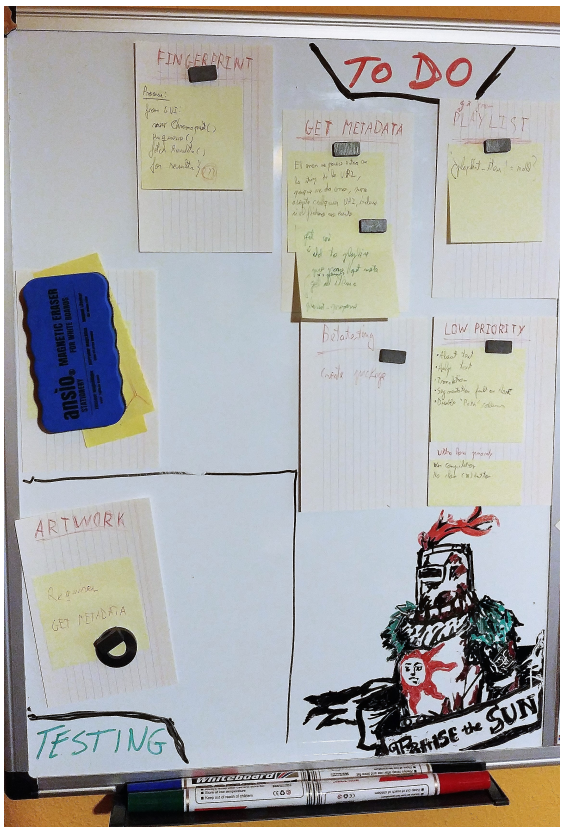

Figura 5.13: Pizarra magnética usada para el control de tareas de desarrollo.

Una vez adquiridas las destrezas básicas, el desarrollo fue fluido, constante y sin problemas dignos de ser mencionados en este documento.

6. CAPÍTULO

Betatesting

El proceso de betatesting se ha dividido en dos procesos distintos llevados a cabo simultáneamente. Cada proceso ha contado con testers de determinado nivel técnico para obtener tanto una perspectiva especializada, la cual ayude a mejorar la efectividad del código, como un punto de vista a nivel de usuario que ayude a hacer la interfaz más cómoda y comprensible.

6.1. Usuarios especializados

En esta sección se describe la fase de betatesting orientada a usuarios especializados.

6.1.1. Características de los testers

Para este proceso se buscan personas especializadas o de un alto nivel de conocimiento técnico. Para ello se dirige directamente personas que hayan cursado o estén cursando estudios del ámbito de la ingeniería, preferiblemente informática. Las capacidades que se requieren son las siguientes:

- Nivel de comprensión del inglés básico.
- Contar con un equipo o máquina virtual con sistema operativo Linux.
- Conocimientos básicos de uso de terminal e instalación desde repositorios.

- Contar con una conexión a internet para la obtención del software y emisión del feedback.

Las personas conocidas del autor que potencialmente cumplen los requisitos o pueden conocer personas que los cumplan son contactadas a través de distintos medios dependiendo del tipo de relación que les une. Se les informa de las características del betatesting y de los requisitos a cumplir y se les pide que faciliten un correo de contacto, con la promesa de que no recibirán en total más de 2 mensajes relacionados con el programa de betatesting.

Se consigue reclutar un total de 7 voluntarios, de los cuales 3 llegan a registrar al menos un error o sugerencia.

6.1.2. Empaquetado

Tal y como se describe en la sección 5.1, se considera que el tiempo requerido para obtener una versión ejecutable en Windows es excesivo y se descarta, dejando en el alcance únicamente la compilación para sistemas Linux.

El software VLC cuenta con muchas dependencias de librerías, las cuales han de ser empaquetadas con el propio software para ser distribuidas. Con cada librería adicional, el empaquetado se hace más complicado y se vuelve más probable que el proceso genere errores que eviten su finalización. Crear un paquete que cumplimente todos los requisitos establecidos por Debian requiere grandes cantidades de tiempo, tiempo que sería más productivo invertir en llevar el prototipo más lejos. Por esto finalmente se opta por hacer el empaquetado con la herramienta *checkinstall*, la cual empaqueta únicamente el software que se le provee, ignorando las dependencias y los requisitos de Debian.

Los paquetes obtenidos con *checkinstall* son más ligeros y fáciles de crear, pero generan más problemas a la hora de la instalación y ejecución del software. Esto podría conllevar un problemas entre usuarios con conocimientos medios o bajos, pero todos los betatesters de esta fase cuentan con conocimientos avanzados, por lo que se decide correr el riesgo de perder parte del feedback por incompatibilidades de los entornos de los testers, a cambio de ganar tiempo de desarrollo adicional.

Para obtener el software *checkinstall* basta con descargarlo de los repositorios de Debian:

```
sudo apt-get install checkinstall
```

Se ha de contar con una versión compilada del software a empaquetar. Una completa descripción del proceso puede encontrase en la sección 5.1.2 (no es necesario instalar el software en el equipo). Una vez se tiene el software compilado se ha de ejecutar el siguiente comando:

```
sudo checkinstall
```

En este punto *checkinstall* nos pedirá que completemos la información del paquete a crear. Por defecto la rellenará con la información disponible del software, pero es probable que requiera alguna modificación. A continuación un ejemplo de la información del paquete usado en este proyecto:

```
This package will be built according to these values:

0 -  Maintainer: [ asiersantosval@gmail.com ]
1 -  Summary: [ This package has been created to test the Extended Metadata Manager for VLC. ]
2 -  Name:    [ emm ]
3 -  Version: [ 0.1 ]
4 -  Release: [ BETA ]
5 -  License: [ GPL ]
6 -  Group:   [ checkinstall ]
7 -  Architecture: [ amd64 ]
8 -  Source location: [ https://github.com/ASantosVal/vlc-extension-trials ]
9 -  Alternate source location: [  ]
10 - Requires: [  ]
11 - Provides: [ emm ]
12 - Conflicts: [  ]
13 - Replaces: [  ]
```

Los valores pueden ser editados pulsando la numeración que precede a cada dato, pulsando ENTER e introduciendo el valor deseado. Una vez se da por buena la información, basta con volver a pulsar ENTER para comenzar el proceso, el cual es automático y puede llegar a tomar hasta 10 minutos en casos de software pesado como VLC. El proceso termina con la instalación del software en el equipo (evitable pasando el parámetro *--install=no* al comando antes descrito) y la creación del fichero *.deb* deseado en la carpeta del proyecto. El nombre del fichero dependerá de la información ofrecida en el proceso. Para el ejemplo mostrado el nombre del fichero sería el siguiente:

```
emm_0.1-BETA_amd64.deb
```

El último paso será la instalación del paquete en la máquina destino. Tal y como se ha mencionado antes en este apartado, *checkinstall* no trata correctamente las múltiples de-

pendencias del software, por lo que será necesario tener una copia funcional de la versión oficial de VLC. Dicha copia puede obtenerse con el siguiente comando:

```
sudo apt-get update && sudo apt-get install vlc
```

Para las funciones de búsqueda automática de metadatos se requiere una versión reciente de FFmpeg, la cual puede no estar disponible en los repositorios oficiales para las versiones de Ubuntu anteriores a 17.04. Esto puede causar que la aplicación se ejecute pero la funcionalidad de búsqueda no funcione (lo cual genera una serie de mensajes de error en rojo en la terminal). Este problema puede ser resuelto añadiendo el siguiente repositorio *ppa* e instalando las librerías requeridas:

```
sudo add-apt-repository ppa:jonathonf/ffmpeg-3
sudo apt update && sudo apt install ffmpeg libav-tools x264 x265
```

Una vez que se cuenta con las librerías solo falta instalar el paquete con el siguiente comando (donde "FICHERO" es el nombre del paquete *.deb*):

```
sudo dpkg -i FICHERO
```

Pese a que el nombre del paquete es *"emm"*, el nombre del software empaquetado sigue siendo *vlc*, por lo que el comando *vlc* bastará para lanzarlo. Sin embargo, al tener dos copias de VLC (la instalada con el paquete y la oficial usada para obtener las librerías), puede que la máquina opte por la copia estable en vez de la de desarrollo. Si se han seguido correctamente los pasos de la sección 5.1.2, esto puede ser evitado especificando la ruta en la que se encuentra el software. El comando es el siguiente:

```
/EMM_BETA/bin/vlc
```

Cabe mencionar que siendo una versión de desarrollo puede que ciertas dependencias no hayan sido cumplidas, haciendo el software potencialmente inestable. De ello son informados los testers tal y como se describirá en la siguiente sección.

6.1.3. Distribución

Tras la recopilación de voluntarios a través de una contacto directo, se procede a facilitarles el software y las instrucciones. Para la distribución de lo necesario se ha optado

por el servicio Google Drive. Esto se debe a que ofrece un servicio de almacenamiento en la nube rápido y versátil que permite compartir múltiples ficheros a través de un único enlace sin necesidad de que los testers posean una cuenta de Google. El último punto es importante pues no se ve adecuado requerir a los testers que se creen cuentas en servicios con condiciones que puedan no querer aceptar.

El enlace a la carpeta junto con el formulario descrito posteriormente en la sección 6.1.4 es enviado por medio de correo electrónico. El texto completo del correo electrónico enviado a los testers puede ser encontrado en el anexo C.1. Los ficheros contenidos en la carpeta son los siguientes:

6.1.3.1. LEEME.txt

Documento de texto plano que contiene la siguiente información:

- Descripción del proyecto
- Casos de uso disponibles
- Instrucciones para su instalación
- Instrucciones para su desinstalación
- Instrucciones para su ejecución
- Instrucciones para el recopilado de datos
- Instrucciones para la resolución de problemas

Este documento cuenta con mucha de la información del apartado anterior. Puede encontrarse el contenido completo del fichero en el anexo C.2.

6.1.3.2. RESUMEN_COMANDOS.txt

Documento de texto plano que contiene un resumen de los comandos necesarios para las siguientes tareas:

- Instalación

- Desinstalación
- Ejecución
- Recopilado de datos
- Resolución de problemas

Puede encontrarse el contenido completo del fichero en el anexo C.3.

6.1.3.3. Comandos_de_compilacion_para_Ubuntu.txt

Recopilación de los comandos necesarios para la compilación del proyecto en Ubuntu, para los casos en los que la resolución de problemas no sea efectiva y el tester esté dispuesto a invertir tiempo adicional en resolver el problema. Este proceso puede llegar a durar más de 20 minutos, por lo que se les advierte a todos los betatesters que no se espera de ellos llevar a cabo el proceso, pero que se agradece la dedicación si optan por llevarlo a cabo.

Puede encontrarse el contenido completo del fichero en el anexo C.4.

6.1.3.4. emm_0.1-BETA_32bit.deb

Software a probar empaquetado en formato Debian. Este fichero contiene la versión compilada para sistemas operativos con arquitectura de 32 bits.

Tal y como se describe en la sección 6.1.2, la instalación no cumplimenta los requisitos de Debian, por lo que se les solicita a los testers que lo instalen con la herramienta de terminal *dpkg* y no desde gestores de paquetes gráficos que puedan proveer sus distribuciones. También se les solicita que no compartan los paquetes provistos con terceros y que si supieran de alguien con interés en el proyecto y en probar el software, que le proveyeran el correo de contacto del desarrollador para que pudiera solicitar acceso al programa de betatesting. Esto es especialmente relevante porque el equipo oficial de VLC pide a los desarrolladores libres que no publiquen binarios de versiones de desarrollo en foros o de manera descontrolada, ya que ello ha generado situaciones de conflicto anteriormente.

6.1.3.5. emm_0.1-BETA_64bit.deb

Software a probar empaquetado en formato Debian. Este fichero contiene la versión compilada para sistemas operativos con arquitectura de 64 bits.

6.1.3.6. ALL-FILES.zip

Fichero comprimido en formato *zip* cuyo contenido son todos los ficheros mencionados anteriormente en esta sección.

6.1.4. Obtención de feedback

El objetivo del programa de betatesting era buscar bugs en el software y recoger sugerencias de mejora. Cada tester podía mandar un número indeterminado de errores y/o sugerencias y todas ellas debían ser fácilmente clasificables para su posterior evaluación. Estos son los datos de interés a recopilar:

En el caso de las sugerencias:

- Descripción de la sugerencia.
- Nombre o alias para los agradecimientos (opcional).

En el caso de los bugs:

- Descripción del bug.
- Pasos seguidos para la aparición del bug.
- Registros/logs de la terminal durante el proceso.
- Nombre o alias para los agradecimientos (opcional).

Se decide que la forma más adecuada para recopilar los datos es a través de la web, ya que además de ser extremadamente accesible no depende de las características del sistema operativo.

Google ofrece un servicio gratuito de creación de formularios web públicos llamado Google Forms. Este servicio permite compartir los formularios a través de un enlace público a un número indeterminado de usuarios, que no necesitan poseer una cuenta de Google para responder.

Las respuestas obtenidas en los formularios quedan registradas en una tabla de datos de tipo Spreadsheet en la plataforma Google Drive, fácilmente descargable y exportable a

los formatos más comunes (como .xlsx o .ods). A las respuestas se les añade la marca temporal en la que fueron registrados los datos. Los datos pueden ser reordenados en base a las respuestas de una de las preguntas realizadas para su más fácil gestión.

Por proveer un único enlace a los testers, se opta por recoger tanto los bugs como las sugerencias en el mismo formulario, haciendo opcionales los apartados específicos de cada uno. El resultado puede verse en las figuras 6.1 y 6.2.

6.1.5. Resultados

Los errores y sugerencias encontrados se muestran en las tablas 6.1 y 6.2. Se ha eliminado la información menos relevante (como los logs, los alias del tester y las fechas en la que fueron reportados) y los errores y sugerencias repetidos.

6.2. Usuarios sin conocimientos técnicos

En esta sección se describe la fase de betatesting orientada a usuarios sin conocimientos técnicos (o usuarios medios).

6.2.1. Características de los testers

Para este proceso se buscan personas sin conocimientos técnicos. El procedimiento descrito en el apartado anterior es excesivamente complejo para usuarios no especializados y tal y como se menciona en la fase 1 (sección 5.1.3) la compilación para Windows ha sido dejada fuera del alcance. Siendo Windows el sistema operativo más extendido entre usuarios de nivel técnico medio-bajo, llevar a cabo el proceso de betatesting entre este tipo de usuarios se vuelve complejo. Por ello se decide que a este grupo se le proporcionará un equipo con el software a probar, es decir, las pruebas se harán de manera presencial. Se contacta con personas cercanas al autor que cuenten con las siguientes características:

- Nivel de comprensión del inglés básico.
- Contar con la disponibilidad para probar la aplicación de manera presencial.

Se consigue reclutar un total de 11 voluntarios, de los cuales 10 llegan a completar el betatesting.

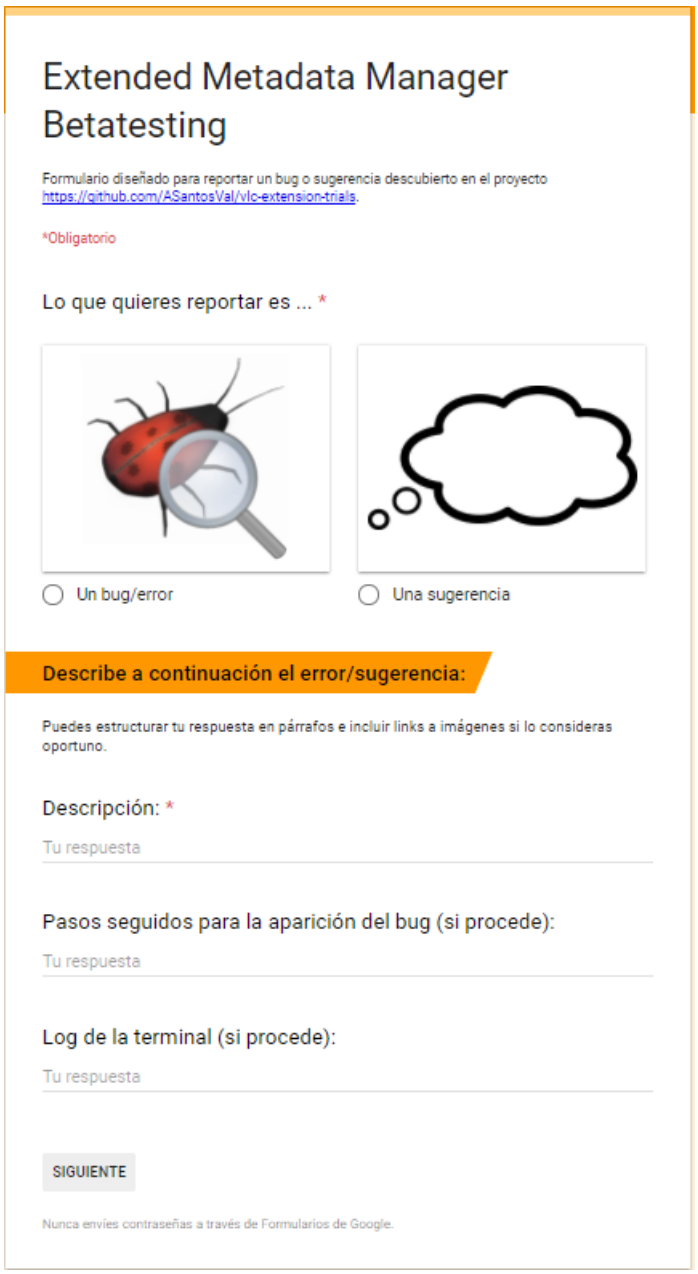

Figura 6.1: Parte 1 del formulario de recepción de feedback.

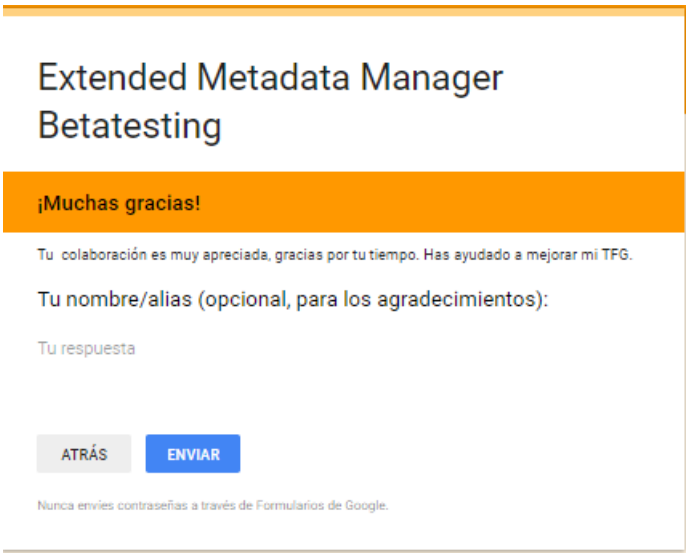

Figura 6.2: Parte 2 del formulario de recepción de feedback.

6.2.2. Proceso

Se decide agrupar a los voluntarios dispuestos a dedicar cierto tiempo a probar el software y ofrecerles una máquina provista temporalmente por el desarrollador. Esto requiere un desplazamiento del desarrollador para poder facilitar la infraestructura necesaria para las pruebas. Se prepara un ordenador portátil con una copia de la máquina virtual usada en el desarrollo y se procede a organizar encuentros en los que múltiples voluntarios puedan probar el software por turnos.

En esta fase no se busca encontrar errores en el software, sino posibles problemas de comprensión con la interfaz. Para las sesiones se les plantea una situación ficticia en la que han de suponer que se encuentran y se les explica que su objetivo es resolver el problema planteado en dicha situación a través del software provisto, sin llegar a darles instrucciones sobre el uso del programa. La descripción de la situación planteada a los testers es la siguiente:

"Imaginaos que tenéis una serie de canciones en vuestro equipo, las cuales no contienen la información del artista, álbum, año de producción, etc. En muchos casos vosotros sabéis cuál es la información correcta, pero no en todos. Lo que queréis es que el programa rellene toda la información posible por sí solo y en aquellos casos en los que la información no sea correcta o simplemente no haya sido encontrada, editarla manualmente."

Se crea una carpeta con una serie de archivos de audio cuyos metadatos han sido elimi-

nados completamente y en algunos casos el nombre del fichero ha sido cambiado por un número aleatorio, para que su identificación por parte del tester sea imposible. Se incluye entre los ficheros una pista de audio que no puede ser identificada a través de la búsqueda online para obligarles a hacer uso de la edición manual. Cada uno de los testers ha de estar aislado del resto mientras hace las pruebas, de manera que todos comiencen las pruebas sin ningún conocimiento del software. Al comienzo de la prueba se les explica dónde está la carpeta con los archivos de audio y se les da el software ya en ejecución. Se les pide que verbalicen los problemas o sugerencias que puedan ocurrírseles durante la prueba y para que los comentarios relevantes sean anotados por el autor. Al terminar cada prueba, se borran los ficheros editados por el tester y se vuelven a copiar los ficheros base.

Al finalizar la sesión se les plantea la siguiente pregunta:

"Si os encontraseis en vuestra vida con un problema como el que habéis tenido que resolver, ¿haríais uso de esta herramienta, buscaríais otra alternativa o decidiríais no invertir tiempo en resolver el problema?"

6.2.3. Resultados

Por motivos de privacidad los nombres de los testers han sido eliminados y sustituidos por números. En muchos casos los problemas conceptuales se repiten entre múltiples testers, por lo que a cada problema se le ha asignado un número que sirva a modo de alias. Los resultados pueden verse en las tablas 6.3 y 6.4.

La respuesta a la pregunta final fue "hacer uso de la herramienta" en 9 de los 10 casos y "no invertir tiempo en resolver el problema" en uno de los casos.

Descripción	**Pasos seguidos para la aparición del bug**	**Correg.**
La aplicación permite elegir ficheros que no son de audio y se bloquea al tratar de guardar los metadatos.	"Get From Folder", seleccionar una imagen y guardar los metadatos.	Sí
La aplicación peta al darle al botón de buscar información si no hay ningún fichero seleccionado.	Iniciar la aplicación y darle a buscar.	Sí
Si seleccionas un fichero de audio y limpias la tabla te deja elegir artwork aunque no haya nada seleccionado.	Iniciar la aplicación ->Seleccionar un fichero de audio ->Clear Table ->Add cover from file.	Sí
Si la playlist está vacía, pulsar el botón "Get From Playlist" y después seleccionar un fichero con "Get From Folder" hace que la aplicación se congele.	Get From Playlist (playlist vacía), Get From Folder, Seleccionar fichero.	Sí
En el tercer párrafo de la ventana de ayuda pone "Yo can choose to load them from..." falta la "u" de "You".	– –	Sí
Al abrir por primera vez (sin haber seleccionado ningún fichero de audio) y hacer doble click en el artwork se abre una ventana: "Current Media Information". No ocurre si ya se ha seleccionado un fichero.	Abrir gestor de metadatos ->Hacer doble click en el artwork.	No
Si haces doble click en "Search Now" el programa bloquea el resto de acciones (clicks), pero no las ignora y se ejecutan inmediatamente después de que la búsqueda finalice.	Pulsar varias veces "Search Now" y tratar de usar otras funciones durante la búsqueda.	No
Este bug está relacionado con la barra de progreso. Cada vez que se completa la búsqueda de los metadatos de un fichero, la cantidad que se suma al progreso es 100/TOTAL_FICHEROS, en vez de 100/TOTAL_FICHEROS_SELECCIONADOS.	Cargar múltiples ficheros, desactivar las casillas de algunos y proceder a buscar.	No
Se cierra VLC cuando añades un archivo con extensión mp3, pero que no es música.	Cambiar la extensión de un archivo jpg a mp3 y hacer la prueba.	No

Tabla 6.1: Bugs encontrados en el betatesting con usuarios especializados.

Sugerencia	Correg.
Una vez seleccionado el archivo, mostrar una referencia para poder identificarlo (ej. Título actual del archivo). Cuando tienes una lista mediana ayuda a saber qué archivos son los que no están siendo reconocidos.	No
Que haya una opción al añadir desde carpeta para que no te borre las canciones añadidas hasta el momento.	No
Si pulsas el botón "Get From Playlist" estando la playlist vacía estaría bien que se mostrase un mensaje avisando de ello (Por ejemplo..."Playlist is empty").	No
La ventana de "help" y "about" podrían ser más anchas (para reducir el número de saltos de línea y que sea más fácil de leer).	No
Estaría bien tener alguna manera de cancelar la búsqueda de metadatos (o por lo menos que se pueda cerrar la ventana sin tener que esperar a que termine).	No
Eliminar la carátula de una canción (cuando ya tiene una)	No
Cambiar varias carátulas a la vez.	No

Tabla 6.2: Sugerencias encontradas en el betatesting con usuarios especializados.

Sujeto	Conflictos encontrados
Sujeto 1	1.- Confunde el significado del botón "Change" de la columna artwork. Piensa que su uso es editar manualmente la columna en la que se encuentra. 2.- No usa la ayuda contextual ni el botón de ayuda.
Sujeto 2	*Conflictos repetidos 1 y 2. 3.- Una vez comenzada la búsqueda no es consciente de que el proceso ha comenzado y trata de usar otras funcionalidades.
Sujeto 3	*Conflictos repetidos 1 y 2. 4.- Trata de cancelar la búsqueda pero la interfaz no se lo permite.
Sujeto 4	*Conflictos repetidos 1, 2 y 4. 5.- A la hora de editar los datos manualmente, no se le ocurre que puede escribir directamente en la tabla.
Sujeto 5	6.- No consigue identificar a qué fichero pertenece cada entrada de la tabla. 7.- Busca la manera de escuchar determinada pista desde el software provisto para poder identificarla, pero este no se lo permite.
Sujeto 6	*Conflictos repetidos 1, 2 y 4.
Sujeto 7	*Conflictos repetidos 1, 2, 3, 4 y 7.

Tabla 6.3: Sesión de betatesting 1. Usuarios **no** acostumbrados al uso de equipos informáticos.

Sujeto	**Conflictos encontrados**
Sujeto 1	8.- Intenta usar atajos de teclado (Ctrl+C y Ctrl+V), pero la ventana de VLC en el fondo los captura y muestra menús no relacionados con la interfaz.
Sujeto 2	9.- Echa en falta un botón que permita deshacer el último cambio efectuado. 10.- A la hora de escoger la imagen a asignar como carátula, echa de menos poder previsualizar las imágenes a seleccionar.
Sujeto 3	11.- Expresa la falta de una notificación al cerrar que te informe de que no se han guardado los cambios.

Tabla 6.4: Sesión de betatesting 2. Usuarios **acostumbrados** al uso de equipos informáticos.

7. CAPÍTULO

Conclusiones

7.1. Resumen del trabajo efectuado

Se ha desarrollado el software denominado Extended Metadata Manager a modo de módulo de VLC, siendo compatible con las versiones de Windows y Linux. Se ha llevado un proceso de betatesting con testers de distinto nivel técnico y se ha recogido su feedback. Se han corregido los errores encontrados e implementado algunas de las sugerencias recibidas, en base al tiempo restante.

Se incluye una comparativa en imágenes entre la funcionalidad que ya incluía VLC (figura 7.1) y la desarrollada en el proyecto (figura 7.2).

Las funciones implementadas son las siguientes:

- Cargar datos desde la lista de reproducción actual.
- Cargar datos desde un explorador de ficheros.
- Buscar metadatos automáticamente (Búsqueda rápida).
- Buscar metadatos automáticamente (Búsqueda avanzada y consulta por pasos).
- Permitir el editado manual de datos en la interfaz.
- Editar datos directamente en la tabla provista por la GUI.

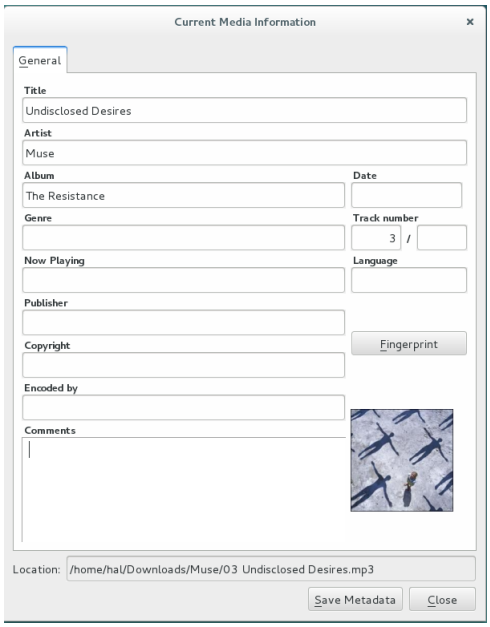

Figura 7.1: Gestor de metadatos nativo de VLC.

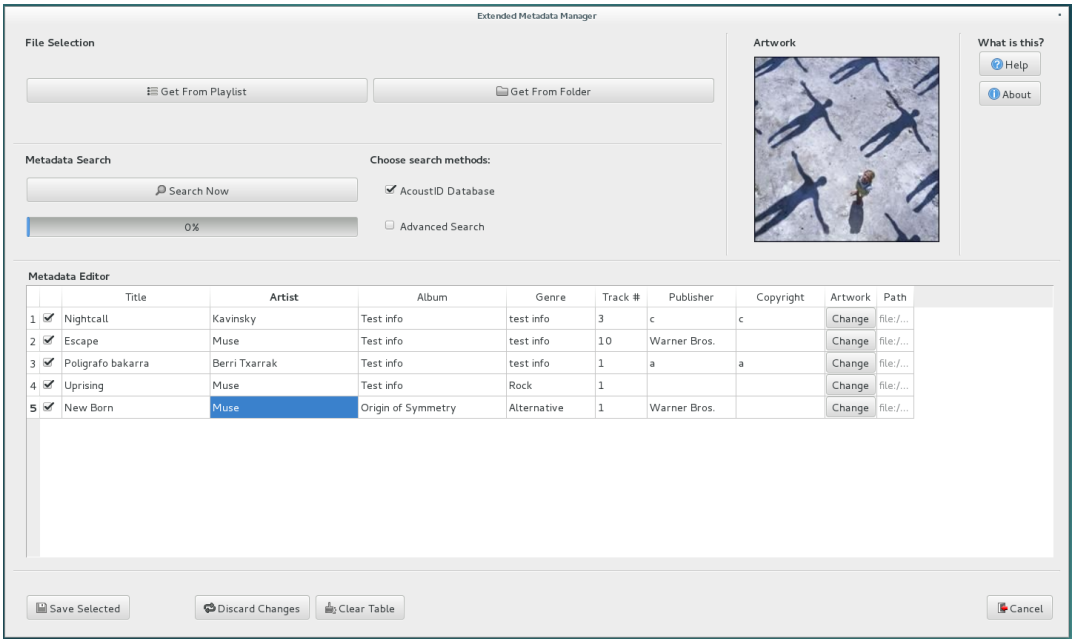

Figura 7.2: Gestor de metadatos extendido desarrollado en este proyecto.

- Guardar los metadatos obtenidos, añadidos y/o editados de manera permanente.
- Mostrar una ventana con texto de ayuda.
- Mostrar una ventana con texto “acerca de”.
- Visualizar la imagen de carátula de cada fichero en la GUI.
- Ver y cambiar artwork/cover del fichero.

7.2. Conclusiones y aprendizaje obtenido

La más notable de las conclusiones obtenidas es lo crítico que puede llegar a ser el tiempo en el ámbito de la informática. A menudo esperar a tener un producto más completo y funcional puede conllevar que la competencia publique un software similar, haciendo que el trabajo propio pierda todo su valor. El constante desarrollo de la industria y comunidad hacen que no cumplir los plazos planificados (o haberlos planificado mal) lleven el proyecto al fracaso.

También cabe mencionar la importancia de la documentación. No solo su existencia, sino también su calidad. El gran problema a gestionar en este proyecto ha sido la falta de información sobre las herramientas ofrecidas por VLC, los datos desactualizados y la falta de soporte. El proyecto probablemente podría haberse completado hasta con 80 horas menos si las fuentes de información hubiesen sido efectivas. Esto no solo lleva a valorar la propia documentación, sino también a las personas encargadas de redactarla, especialmente en proyectos open source en los que es fácil que los colaboradores tiendan más hacia el desarrollo puro.

Otra conclusión obtenida es la complejidad de crear herramientas multiplataforma usando tecnologías que no lo son. Por ejemplo, crear una aplicación móvil y de navegador puede ser sencillo si se usan frameworks que operan en múltiples sistemas de forma nativa (como Ionic), pero basar un proyecto en C y usar distintas librerías para distintos sistemas (hasta el punto de tener que crear una interfaz completamente separada para determinado sistema, como pasa con VLC en MacOS) hace que el desarrollo, gestión y mantenimiento del proyecto se vuelva extremadamente complejo.

7.3. Posibles mejoras

Las siguientes ideas estaban fuera del alcance mínimo o han surgido durante el desarrollo y no han llegado a ser implementadas, pero está planificado que se completen en la próxima iteración del proyecto.

- Capacidad de procesar carpetas enteras y sus respectivos sub-elementos.
- Listado de géneros "oficiales" disponibles.
- Traducción de la interfaz.
- Obtención de metadatos a través del análisis del nombre del fichero.
- Búsqueda en bases de datos adicionales.
- Permitir ordenado de la tabla en base a las columnas.
- Adaptar el ancho de las columnas al tamaño actual de la interfaz.
- Obtención de metadatos adicionales de la base de datos AcoustID.
- Mejoras generales a la interfaz (como mensajes de confirmación, cambio de ciertos textos, cancelado de búsqueda, etc.).
- Mostrar una referencia clara en la tabla para poder identificar los ficheros.
- Opción de añadir contenido a la tabla sin eliminar el existente.
- Opción de editar la carátula de múltiples ficheros al mismo tiempo.
- Crear un botón para deshacer el último cambio efectuado.
- Permitir la escucha de la canción asociada a la posición de la tabla seleccionada.

7.4. Futuro del proyecto

Si bien el alcance mínimo del Proyecto de Fin de Grado llegaba hasta el lanzamiento de la versión Beta y el consecuente programa de betasteting, el propio proyecto llega mucho más allá. El objetivo es corregir todos los errores obtenidos en el betatesting, así como implementar las sugerencias más interesantes, en una segunda iteración que culmine con el envío del proyecto al equipo de VLC para su potencial inserción en la versión oficial.

7.5. Agradecimientos

Este proyecto no habría sido posible sin el apoyo de muchas de las personas de mi entorno. Unas breves palabras no valen para agradecer el continuo apoyo y probablemente me deje alguien, pero lo intentaré.

Al tutor de mi proyecto, Imanol Usandizaga, por guiarme, por los buenos consejos, pero sobre todo por el entusiasmo y confianza en el proyecto.

A mis compañeros de carrera en general y en particular:
A Eneko, por la ayuda con la motivación y por ser mi gurú de C++ sin quererlo.
A Urko, por sus consejos y realismo.

A mis compañeros de Magna SIS, por entender y por el interés activo. Especialmente a Igor, por su afán por encontrar errores tanto en el código como en la memoria. Se merece su propio apartado de agradecimientos.

A Israel, por su inestimable ayuda en el bocetaje y diseño de la interfaz, por aprender PixelArt para ayudarme con los iconos y por todas las horas de trabajo conjunto, que lo hicieron más llevadero.

A todos los betatesters, por dedicar su tiempo a ayudarme a obtener un programa mejor: Jandro, Igor, Urko, Marta, María, Irati, Edurne, Leticia, Ane Irene, Israel, Mikel, Ylenia y Marina.

A toda mi familia, por la ayuda para llegar hasta este punto y por la oportunidad que me han ofrecido para demostrar lo que valgo.

Y finalmente, pero no por ello siendo menos importante, a Ane Irene, por todo. Por estar, por ser y por haber sacado tiempo pese a sus circunstancias.

Muchas gracias a todos. Este logro también es vuestro.

Anexos

A. ANEXO

Acrónimos

VLC - VideoLan Client. Reproductor multimedia multiplataforma open source.

API - Application Programming Interface. Conjunto de funciones, procedimientos y rutinas que ofrece determinado software como una capa de abstracción.

OS - Operative System.

IRC - Internet Relay Chat. Protocolo de comunicación en tiempo real a través de texto.

HTML - HyperText Markup Language. Lenguaje utilizado para la estructuración de la información, especialmente en el ámbito web.

URI - Uniform Resource Identifier. Cadena de caracteres que identifica un recurso local o de red.

URL - Uniform Resource Locator. Junto con URN (Uniform Resource Name) genera una URI.

XML - eXtensible Markup Language. Metalenguaje enfocado a la definición de marcas desarrollado por la World Wide Web Consortium.

B. ANEXO

Ejemplos de código

B.1. Módulo Hello World

El siguiente código ha sido obtenido íntegramente del sitio web disponible en la entrada bibliográfica [VideoLAN Organization, 2017a].

```
/**
 * @file hello.c
 * @brief Hello world interface VLC module example
 */
#ifdef HAVE_CONFIG_H
# include "config.h"
#endif

#include <stdlib.h>
/* VLC core API headers */
#include <vlc_common.h>
#include <vlc_plugin.h>
#include <vlc_interface.h>

/* Forward declarations */
static int Open(vlc_object_t *);
static void Close(vlc_object_t *);

/* Module descriptor */
vlc_module_begin()
    set_shortname(N_("Hello"))
    set_description(N_("Hello interface"))
    set_capability("interface", 0)
    set_callbacks(Open, Close)
```

```
25      set_category(CAT_INTERFACE)
26      add_string("hello-who", "world", "Target", "Whom to say hello to.", false)
27  vlc_module_end ()
28
29  /* Internal state for an instance of the module */
30  struct intf_sys_t
31  {
32      char *who;
33  };
34
35  /**
36   * Starts our example interface.
37   */
38  static int Open(vlc_object_t *obj)
39  {
40      intf_thread_t *intf = (intf_thread_t *)obj;
41
42      /* Allocate internal state */
43      intf_sys_t *sys = malloc(sizeof (*sys));
44      if (unlikely(sys == NULL))
45          return VLC_ENOMEM;
46      intf->p_sys = sys;
47
48      /* Read settings */
49      char *who = var_InheritString(intf, "hello-who");
50      if (who == NULL)
51      {
52          msg_Err(intf, "Nobody to say hello to!");
53          goto error;
54      }
55      sys->who = who;
56
57      msg_Info(intf, "Hello %s!", who);
58      return VLC_SUCCESS;
59
60  error:
61      free(sys);
62      return VLC_EGENERIC;
63  }
64
65  /**
66   * Stops the interface.
67   */
68  static void Close(vlc_object_t *obj)
69  {
70      intf_thread_t *intf = (intf_thread_t *)obj;
71      intf_sys_t *sys = intf->p_sys;
72
73      msg_Info(intf, "Good bye %s!", sys->who);
74
75      /* Free internal state */
76      free(sys->who);
```

```
77      free(sys);
78  }
```

B.2. Enlazado entre LUA y C para editado de metadatos

```
1   #include <id3v2lib.h>     /* id3 libraries */
2
3   #include <lua.h>        /* Lua libraries */
4   #include <lauxlib.h>
5   #include <lualib.h>
6
7
8   static int set_title_bind(lua_State *L){
9       char *text = (char*)lua_tostring(L, -1);       /*Get 2nd arg, string */
10      char *path = (char*)lua_tostring(L, -2);       /*Get 1st arg, string */
11
12      ID3v2_tag* tag = load_tag(path); // Load the full tag from the file //TODO: error handling if
          file not found
13      if(tag == NULL) {tag = new_tag();}
14
15      tag_set_title(text, 0, tag);
16      set_tag(path, tag);
17
18      return 0;   /* no return values */
19  }
20
21  static int set_album_bind(lua_State *L){
22      char *text = (char*)lua_tostring(L, -1);       /*Get 2nd arg, string */
23      char *path = (char*)lua_tostring(L, -2);       /*Get 1st arg, string */
24
25      ID3v2_tag* tag = load_tag(path); // Load the full tag from the file //TODO: error handling if
          file not found
26      if(tag == NULL) {tag = new_tag();}
27
28      tag_set_album(text, 0, tag);
29      set_tag(path, tag);
30
31      return 0;   /* no return values */
32  }
33
34  static int set_artist_bind(lua_State *L){
35      char *text = (char*)lua_tostring(L, -1);       /*Get 2nd arg, string */
36      char *path = (char*)lua_tostring(L, -2);       /*Get 1st arg, string */
37
38      ID3v2_tag* tag = load_tag(path); // Load the full tag from the file
39      if(tag == NULL) {tag = new_tag();}
40
```

```
41      tag_set_artist(text, 0, tag);
42      set_tag(path, tag);
43
44      return 0;   /* no return values */
45  }
46
47  static int set_album_artist_bind(lua_State *L){
48      char *text = (char*)lua_tostring(L, -1);       /*Get 2nd arg, string */
49      char *path = (char*)lua_tostring(L, -2);       /*Get 1st arg, string */
50
51      ID3v2_tag* tag = load_tag(path); // Load the full tag from the file
52      if(tag == NULL) {tag = new_tag();}
53
54      tag_set_album_artist(text, 0, tag);
55      set_tag(path, tag);
56
57      return 0;   /* no return values */
58  }
59
60  static int set_genre_bind(lua_State *L){
61      char *text = (char*)lua_tostring(L, -1);       /*Get 2nd arg, string */
62      char *path = (char*)lua_tostring(L, -2);       /*Get 1st arg, string */
63
64      ID3v2_tag* tag = load_tag(path); // Load the full tag from the file
65      if(tag == NULL) {tag = new_tag();}
66
67      tag_set_genre(text, 0, tag);
68      set_tag(path, tag);
69
70      return 0;   /* no return values */
71  }
72
73  static int set_comment_bind(lua_State *L){
74      char *text = (char*)lua_tostring(L, -1);       /*Get 2nd arg, string */
75      char *path = (char*)lua_tostring(L, -2);       /*Get 1st arg, string */
76
77      ID3v2_tag* tag = load_tag(path); // Load the full tag from the file
78      if(tag == NULL) {tag = new_tag();}
79
80      tag_set_comment(text, 0, tag);
81      set_tag(path, tag);
82
83      return 0;   /* no return values */
84  }
85
86  static int set_year_bind(lua_State *L){     //TODO: receive INT
87      char *text = (char*)lua_tostring(L, -1);       /*Get 2nd arg, string */
88      char *path = (char*)lua_tostring(L, -2);       /*Get 1st arg, string */
89
90      ID3v2_tag* tag = load_tag(path); // Load the full tag from the file
91      if(tag == NULL) {tag = new_tag();}
92
```

```
93      tag_set_year(text, 0, tag);
94      set_tag(path, tag);
95
96      return 0;   /* no return values */
97  }
98
99  static int set_track_bind(lua_State *L){     //TODO: receive INT
100     char *text = (char*)lua_tostring(L, -1);       /*Get 2nd arg, string */
101     char *path = (char*)lua_tostring(L, -2);       /*Get 1st arg, string */
102
103     ID3v2_tag* tag = load_tag(path); // Load the full tag from the file
104     if(tag == NULL) {tag = new_tag();}
105
106     tag_set_track(text, 0, tag);
107     set_tag(path, tag);
108
109     return 0;   /* no return values */
110 }
111
112 static int set_disc_number_bind(lua_State *L){
113     char *text = (char*)lua_tostring(L, -1);       /*Get 2nd arg, string */
114     char *path = (char*)lua_tostring(L, -2);       /*Get 1st arg, string */
115
116     ID3v2_tag* tag = load_tag(path); // Load the full tag from the file
117     if(tag == NULL) {tag = new_tag();}
118
119     tag_set_disc_number(text, 0, tag);
120     set_tag(path, tag);
121
122     return 0;   /* no return values */
123 }
124
125 static int set_composer_bind(lua_State *L){
126     char *text = (char*)lua_tostring(L, -1);       /*Get 2nd arg, string */
127     char *path = (char*)lua_tostring(L, -2);       /*Get 1st arg, string */
128
129     ID3v2_tag* tag = load_tag(path); // Load the full tag from the file
130     if(tag == NULL) {tag = new_tag();}
131
132     tag_set_composer(text, 0, tag);
133     set_tag(path, tag);
134
135     return 0;   /* no return values */
136 }
137
138 static int set_album_cover_bind(lua_State *L){     //gets the path to the file
139     char *pathToCover = (char*)lua_tostring(L, -1);       /*Get 2nd arg, string */
140     char *pathToAudio = (char*)lua_tostring(L, -2);       /*Get 1st arg, string */
141
142
143     ID3v2_tag* tag = load_tag(pathToAudio); // Load the full tag from the file
144
```

```
145     if(tag == NULL) {tag = new_tag();}
146 
147     tag_set_album_cover(pathToCover, tag);
148 
149     set_tag(pathToAudio, tag);
150 
151     return 0;   /* no return values */
152 }
153 
154 
155 /* Register this file's functions with the
156  * luaopen_libraryname() function, where libraryname
157  * is the name of the compiled .so output (and mybe the path).
158  * This function should contain lua_register() commands for
159  * each function you want available from Lua.
160  *
161 */
162 int luaopen_share_lua_extensions_libs_id3libBinding (lua_State *L){
163     lua_register(
164             L,                  /* Lua state variable */
165             "set_title",        /* func name as known in Lua */
166             set_title_bind         /* func name in this file */
167             );
168     lua_register(L,"set_album",set_album_bind);
169     lua_register(L,"set_artist",set_artist_bind);
170     lua_register(L,"set_album_artist",set_album_artist_bind);
171     lua_register(L,"set_genre",set_genre_bind);
172     lua_register(L,"set_comment",set_comment_bind);
173     lua_register(L,"set_year",set_year_bind);
174     lua_register(L,"set_track",set_track_bind);
175     lua_register(L,"set_disc_number",set_disc_number_bind);
176     lua_register(L,"set_composer",set_composer_bind);
177     lua_register(L,"set_album_cover",set_album_cover_bind);
178 
179     return 0;
180 }
```

B.3. Enlazado entre LUA y C para fingerprinting

```
1  #include <lua.h>        /* Lua libraries */
2  #include <lauxlib.h>
3  #include <lualib.h>
4 
5  #include <stdlib.h>
6 
7  #include <string.h>
8 
```

```
#include <chromaprint.h>  /* Fingerprinting libraries */


static int get_chromaprint_bind (lua_State *L){
    char *pathToAudio = (char*)lua_tostring(L, -1);       /*Get 1st arg, string */

    int system(const char *command);

    char command[256]; // = "fpcalc %s\n", pathToAudio
    strcpy(command, "fpcalc \"");
    strcat(command, pathToAudio);
    strcat(command, "\"");

    char *result = (char*) system(command);

    //TODO: return the value

    return 0;   /* no return values */
}


/* Register this file's functions with the
 * luaopen_libraryname() function, where libraryname
 * is the name of the compiled .so output (and mybe the path).
 * This function should contain lua_register() commands for
 * each function you want available from Lua.
 *
*/
int luaopen_share_lua_extensions_libs_id3libBinding (lua_State *L){
    lua_register(L,"get_chromaprint",get_chromaprint_bind);

    return 0;
}
```

B.4. Función para el análisis del nombre del fichero

```
function filename_analysis()
  fileName = GetFileName()
  bol_spaceFound = false
  int_spaceCount = 0
  int_foundSections = 1
  table_sections = {}
  table_sections[int_foundSections] = ''

  --now it will look for posible artist and song-name analysing the filename
  --a while is needed to be able to modify the index (i) and jump the space (represented as
     '%20')'
```

```
  for i=1,fileName:len() do
    current_char = fileName:sub(i,i)
    if current_char == " " then--found space, posible multiple space separator
      bol_spaceFound = true
      int_spaceCount = int_spaceCount + 1 --start counting spaces
      if int_spaceCount == 3 then --triple space found, create new section
        bol_spaceFound = false
        int_spaceCount = 0
        int_foundSections = int_foundSections + 1
        table_sections[int_foundSections] = ''
      end
    else
      --the follofing means it found a space, but was not a triple-space separator,
      --so it's added to the section
      if bol_spaceFound == true then
        table_sections[int_foundSections] = table_sections[int_foundSections]..' '
        bol_spaceFound = false
        int_spaceCount = 0
      end

      if current_char == "-" then --separator found (-), new section inititialized
        int_foundSections = int_foundSections + 1
        table_sections[int_foundSections] = ''

      elseif current_char == "." then --extension starts, ignore the rest
        break

      else --normal character, add it to the section
        table_sections[int_foundSections] = table_sections[int_foundSections]..current_char
      end

    end
  end
  return table_sections
end
```

C. ANEXO

Documentos para los betatesters

C.1. Correo enviado

```
--> Betatesting para  "Extended Metadata Manager for VLC" de Asier Santos


Estás recibiendo este mensaje porque te has mostrado dispuesto a hacer de Betatester para el proyecto "Extended Metadata Manager for VLC" de Asier Santos. Si sabes de alguien interesado/dispuesto a unirse al programa de Betatesting, por favor, no le reenvíes el mensaje, pídele que contacte conmigo a través de asiersantosval@gmail.com (o contacta tú mismo). Gracias por tu colaboración.

A continuación tienes un link con los ficheros necesarios para la instalación en maquinas Linux.

https://drive.google.com/drive/folders/0B_09pQqYeOEmdi1xTTZBTWo3OUk?usp=sharing

Se incluyen dos versiones del software, compiladas para SO de 32 y de 64 bits. El fichero LEEME.txt cuenta con las instrucciones, así como una descripción del paquete que lo acompaña. El fichero RESUMEN_COMANDOS.txt contiene un listado abreviado con los comandos necesarios. También se pueden obtener todos los anteriores en el fichero comprimido "ALL-FILES.zip".

Si encuentras algún bug o tienes una sugerencia, por favor, envíala a través del siguiente formulario:

https://docs.google.com/forms/d/1q8hLBQSCUgd3_nN5qAgPc_RZvVXsX4mBQyKiMFTVsZ4

Finalmente, agradecerte tu tiempo y dedicación. Trataré de solucionar los errores que se encuentren en la medida de lo posible, especialmente aquellos que se reporten antes del 3 de Julio de 2017.

```

```
Un saludo y gracias otra vez,
Asier Santos Valcárcel
```

C.2. LEEME.txt

```
------------------------------
-------- Introducción ---------
------------------------------
El siguiente paquete ha sido diseñado para probar el Gestor de Metadatos Extendido implementado en VLC por Asier Santos. Es una versión de desarrollo tanto del Gestor de Metadatos Extendido como del resto de VLC, por lo que no se recomienda su uso más allá del betatesting por poder ser altamente inestable.

Por favor, no extiendas este paquete, pues está basado en la rama de desarrollo de VLC y no está lista para el público general.

Puedes ver el código del proyecto en https://github.com/ASantosVal/vlc-extension-trials.

--ATENCION--
Recuerda que esta es una copia de desarrollo, asegúrate de tener una copia de seguridad de los ficheros con los que trabajes. Este software no viene con ninguna garantía, con la esperanza de que sea útil. El desarrollador no se hace responsable de los daños ocasionados a los datos que se le faciliten al software.



------------------------------
-- Descripción del software ---
------------------------------
El Gestor de Metadatos Extendido o Extended Metadata Manager (EMM) es una ventana implementada dentro de VLC (en sus interfaces para Linux y Windows) con el fin de facilitar la edición de los metadatos o tags de múltiples archivos de música al mismo tiempo. El software es capaz de buscar los metadatos adecuados en bases de datos online y ofrecértelas, para facilitar el proceso, incluso cuando no se cuenta con más información que la pista de audio en sí.



------------------------------
-------- Instalación ----------
------------------------------
El paquete puede ser instalado desde la terminal con el comando:
>sudo dpkg -i FICHERO   #donde el nombre puede ser 'emm_0.1-BETA_32bit.deb' o 'emm_0.1-BETA_64bit.deb' dependiendo de tu equipo.

No se recomienda usar gestores de paquetes gráficos (como el centro de software de Ubuntu) por no ser un paquete que cumplimente los requisitos impuestos por la mayoría de las distros (y por ende pueden ser rechazados por dichos gestores).

```

```
28 Para usar el paquete necesitarás además una copia estable de VLC obtenible en las distros basadas
       en Debian con:
29 >sudo apt-get update && sudo apt-get install vlc
30
31 Para las funciones de búsqueda automática de metadatos se requiere una versión reciente de FFmpeg
     , la cual puede no estar disponible en los repositorios oficiales para las versiones de
     Ubuntu anteriores a 17.04. Esto puede causar que la aplicación se ejecute pero la
     funcionalidad de búsqueda no funcione (también genera una serie de mensajes de error en rojo
       en la terminal). Este problema puede ser resuelto añadiendo el siguiente repositorio ppa e
     instalando las librerías requeridas:
32
33 >sudo add-apt-repository ppa:jonathonf/ffmpeg-3
34 >sudo apt update && sudo apt install ffmpeg libav-tools x264 x265
35 A continuación reiniciar su equipo.
36
37 -------------------------------
38 --------- Ejecución -----------
39 -------------------------------
40 Para ejecutar el software provisto basta con la orden desde la terminal (sin permisos de
     administrador):
41 >/EMM_BETA/bin/vlc
42 Dependiendo de la distro, puede que también cree una entrada en el menú principal, pero podría
     entrar en conflicto con la versión oficial, por lo que no se recomienda su uso.
43
44 Una vez la interfaz de VLC, puedes pulsar "Ctrl+Shift+I" para lanzar la ventana del Extended
     Metadata Manager a testear (o desde el menú "Herramientas" de la barra superior). En este
     paquete solo se encuentra su versión en inglés.
45
46 --ATENCION--
47 También se puede lanzar VLC con funciones de Debugging activadas, lo cual facilita la
     identificación de errores:
48 >/EMM_BETA/bin/vlc -vvv
49 Sin encuentras un bug, por favor, intenta reproducirlo con las opciones de debug activadas e
     incluye los registros de la terminal en el formulario provisto para reportar errores y
     sugerencias.
50
51 -------------------------------
52 ------- Desinstalación --------
53 -------------------------------
54 Elimina tu copia local con el comando:
55 >sudo dpkg -r emm
56
57 Y si por tener Ubuntu 16 o menos has tenido que instalar la librería FFmpeg, puedes eliminar el
     repositorio y software instalado con:
58 >sudo apt install ppa-purge && sudo ppa-purge ppa:jonathonf/ffmpeg-3 && sudo apt autoremove
59
60 -------------------------------
61 -------- Casos de uso ---------
62 -------------------------------
63 -Cargar datos desde la lista de reproducción o desde ficheros.
64 -Buscar metadatos automáticamente (Búsqueda rápida).
65 -Buscar metadatos automáticamente (Búsqueda avanzada y consulta por pasos).
```

```
66  -Editar datos directamente en la tabla provista por la GUI.
67  -Guardar metadatos obtenidos/añadidos/editados permanentemente.
68  -Re-cargar los datos actuales (borrar cambios efectuados, no guardados).
69  -Limpiar la tabla con datos actual.
70  -Ver y cambiar artwork/cover del fichero.
71  -Ver la ayuda o la información "acerca de".
72
73
74  ------------------------------
75  --- Resolución de problemas ---
76  ------------------------------
77
78   ---------------------------------------
79   -- No aparecen los menús mencionados --
80   ---------------------------------------
81  Si el menú "Extended Metadata Manager" no aparece o el atajo "Ctrl+Shift+I" no hace nada, lo más
        probable es que se haya ejecutado tu copia original de VLC (y no la provista por este
        paquete). Puedes comprobarlo ejecutando VLC en modo debug (vlc -vvv) y mirando los mensajes
        generados al principio. La versión oficial debería ser v2.2.* y la provista para el
        betatesting v3.*.
82
83  En tal caso prueba los siguientes pasos (teniendo ya instalado el paquete provisto):
84
85  >sudo apt-get remove vlc    #esto elimina tu copia estable de vlc
86  volver a ejecutar vlc y probar si los menús aparecen
87  >sudo apt-get install vlc   #volver a instalar VLC después de las pruebas
88
89  o
90
91  >sudo apt-get remove vlc    #esto elimina tu copia estable de vlc
92  >sudo apt autoremove        #esto elimina las dependencias del software desinstalado
93  >sudo apt-get install vlc   #volver a instalar VLC y sus dependencias
94  volver a ejecutar vlc y probar si los menús aparecen
95
96   ------------------------------------------------------------
97   -- Una vez desinstalado el paquete, mi vlc ya no funciona --
98   ------------------------------------------------------------
99  Asegúrate de haber seguir / haber seguido estos pasos:
100
101 >sudo dpkg -r emm
102 >sudo apt-get remove vlc
103 >sudo apt autoremove        #esto elimina las dependencias de el software desinstalado
104 >sudo apt-get install vlc
105 reinicia tu equipo (puede que por problemas con la caché, tu copia oficial no funcione hasta que
        reinicies)
106
107  ------------------------------------------------------
108  -- Otros problemas/lo anterior no me lo ha solucionado --
109  ------------------------------------------------------
110 La opción que mejores resultados garantiza es el compilado en tu propia máquina, pero esto
        requiere más tiempo y por eso no se ha considerado. De todas formas, si estás dispuesto a
        invertir del orden de 20 minutos, que es lo que cuesta compilarlo, se incluye un fichero ("
```

```
    Comandos de compilación para Ubuntu.txt") con lo necesario para hacer la compilación automá
    ticamente. Basta con copiar todos los comandos a la vez y pegarlos en la terminal.
111
112
113 Si lo anterior no te ha resuelto los problemas o tienes otro tipo de problema, puedes contactar
    con el desarrollador a través del siguiente correo:
114 asiersantosval@gmail.com
115
116 Se tratará de ayudar en la medida de lo posible, recordando siempre que el software viene sin
    garantías, con la esperanza de que sea útil.
```

C.3. RESUMEN_COMANDOS.txt

```
1  ------------------------------
2  -------- Instalación ----------
3  ------------------------------
4  sudo dpkg -i FICHERO
5  sudo apt-get update && sudo apt-get install vlc
6
7  #En Ubuntu 16 también es necesario:
8
9  sudo add-apt-repository ppa:jonathonf/ffmpeg-3
10 sudo apt update && sudo apt install ffmpeg libav-tools x264 x265
11 #Probablemente requiera un reinicio y el siguient comando
12 /EMM_BETA/bin/vlc  --reset-config --reset-plugins-cache
13
14 ------------------------------
15 ------- Desinstalación --------
16 ------------------------------
17 sudo dpkg -r emm
18
19 #En Ubuntu 16 también es necesario:
20
21 sudo apt install ppa-purge && sudo ppa-purge ppa:jonathonf/ffmpeg-3 && sudo apt autoremove
22
23 ------------------------------
24 --------- Ejecución -----------
25 ------------------------------
26 /EMM_BETA/bin/vlc
27
28 ------------------------------
29 -- Ejecución (con debuging) --
30 ------------------------------
31 /EMM_BETA/bin/vlc -vvv
32
33 ------------------------------
34 --- Resolución de problemas ---
```

```
------------------------------
 --------------------------------------
 -- No aparecen los menús mencionados --
 --------------------------------------
sudo apt-get remove vlc     #esto elimina tu copia estable de VLC
#volver a ejecutar vlc y probar si los menús aparecen
sudo apt-get install vlc    #volver a instalar VLC después de las pruebas
o

sudo apt-get remove vlc     #esto elimina tu copia estable de VLC
sudo apt autoremove         #esto elimina las dependencias de el software desinstalado
sudo apt-get install vlc    #volver a instalar VLC y sus dependencias

 ------------------------------------------------------------
 -- Una vez desinstalado el paquete, mi vlc ya no funciona --
 ------------------------------------------------------------
sudo dpkg -r emm
sudo apt-get remove vlc
sudo apt autoremove         #esto elimina las dependencias de el software desinstalado
sudo apt-get install vlc
#reinicia tu equipo
```

C.4. Comandos_de_compilacion_para_Ubuntu.txt

```
------------
--Build
------------

mkdir BETATESTING && cd BETATESTING &&

sudo apt-get build-dep vlc &&              #hay que activar los repositorios deb-src en /etc/apt/
     sources.list
sudo apt-get install git libtool build-essential pkg-config autoconf lua5.1 yasm wayland-
     protocols &&

git clone https://git.ffmpeg.org/ffmpeg.git ffmpeg && cd ffmpeg &&
./configure --enable-nonfree --enable-pic --enable-shared &&
make &&
sudo make install  &&
cd .. &&

git clone https://github.com/ASantosVal/vlc-extension-trials.git EMM_for_VLC &&
cd EMM_for_VLC &&
./bootstrap &&

sudo apt-get remove qt5-default qt5-qmake qtbase5-dev qtbase5-dev-tools libqt5opengl5-dev
     libqt5x11extras5-dev &&
```

```
21 sudo ./configure &&
22 sudo ./compile  &&
23 sudo apt-get install qt5-default qt5-qmake qtbase5-dev qtbase5-dev-tools libqt5opengl5-dev
       libqt5x11extras5-dev
24
25 ------------
26 --Run
27 ------------
28 ./vlc -vvv
```

D. ANEXO

Actas de reunión

D.1. Acta 11/10/2016

Fecha: 11/10/2016
Lugar: Despacho 259 de la FISS
Participantes:
-Asier Santos
-Imanol Usandizaga

Orden del día:

->Planteamiento de ideas para el TFG:
Tras plantear 2 ideas, se opta por la primera: La creación de un módulo o plug-in open source para VLC cuya funcionalidad sea, en principio, facilitar el streaming a dispositivos de red UPnP.

->Elección del idioma:
Teniendo la posibilidad de desarrollar el trabajo en Euskera, Castellano o Inglés.

->Pasos a seguir:
Primero Asier debe matricular el TFG en la plataforma GAUR. Se acuerda vovler a reunirse en unas semanas

D.2. Acta 25/10/2016

Fecha: 25/10/2016
Lugar: Despacho 259 de la FISS
Participantes:
-Asier Santos
-Imanol Usandizaga

Orden del día:

->Consejos para comenzar:
Imanol recomienda establecer un alcance mínimo y uno extendido por si da tiempo. También definir lo antes posible el SDK a usar, el repositorio, la distribución, etc. También recomienda iniciar la memoria desde el día 1 haciendo un breve diario de lo hecho cada día y especialmente los motivos a la hora de tomar decisiones.

->Pasos a seguir:
Asier debe crear una planificación inicial.

D.3. Acta 8/11/2016

Fecha: 8/11/2016
Lugar: Despacho 259 de la FISS
Participantes:
-Asier Santos
-Imanol Usandizaga

Orden del día:

->Se presentan las decisiones de Idioma del trabajo:
Pese a conocer los 3 idiomas, se opta por el Castellano, por ser en el que Asier mejor se desenvuelve para escribir documentación técnica. De todas formas, se acuerda mantener la comunicación en Euskera.

->Se presentan las decisiones de IDE:
Imanol se muestra de acuerdo con la decisión.

->Se presenta la planificación inicial:
Imanol se muestra satisfecho con lo presentado.

->Pasos a seguir:
Asier debe comenzar con la parte técnica del proyecto.

D.4. Acta 29/03/2017

Fecha: 8/11/2016
Lugar: Plataforma Hangouts (Telemática)
Participantes:
-Asier Santos
-Imanol Usandizaga

Orden del día:

->Imanol recomienda mencionar los múltiples medios de comunicación e idiomas del proyecto.

->Asier plantea crear como documento añadido una guía de compilado de VLC actualizada:
Imanol muestra interés, pero siempre y cuando haya tiempo de sobra para dedicárselo o hacerlo fuera del marco del TFG.

->Asier propone cambiar el objetivo del proyecto de un streamer UPnP a un gestor de metadatos de audio por la potencial complejidad del primero:
A Imanol le parece razonable

->Definición de un nuevo título:
Imanol no lo ve necesario en este momento. Podrá ser abarcado a la hora de solicitar la defensa.

D.5. Acta 22/05/2017

Fecha: 22/05/2017
Lugar: Plataforma Hangouts (Telemática)
Participantes:
-Asier Santos
-Imanol Usandizaga

Orden del día:

->Se informa a Imanol de que parte de la idea original ya está implementada:
Imanol propone replantear el proyecto de manera que extienda o mejore lo ya existente.

->Se propone presentar el trabajo en septiembre y no en julio tal y como se había pensado originalmente:
Imanol está de acuerdo con que es la mejor opción dada la situación.

D.6. Acta 15/06/2017

Fecha: 15/06/2017
Lugar: Despacho 259 de la FISS
Participantes:
-Asier Santos
-Imanol Usandizaga

Orden del día:

->Asier presenta una demo del proyecto:
A Imanol le parece suficiente y está de acuerdo en comenzar con el betatesting y la memoria.

->Asier quiere saber qué recomienda Imanol: hacer la memoria en Doc o en LaTeX:
Imanol propone que simplemente por haber planteado la posibilidad de hacerlo en LaTeX, ya es suficiente como para decantarse por ello.

->Imanol recomienda mencionar en la defensa las siguientes cuestiones:
-Hacer una comparación de lo existente y lo implementado.
-El potencial impacto debido al número de usuarios de VLC.
-Las limitaciones a las que se ha tenido que someter el proyecto por ser de comunidad.

->Se acuerda volver a reunirse aproximadamente el día 12 de Julio:

D.7. Acta 11/07/2017

Fecha: 11/07/2017
Lugar: Despacho 259 de la FISS
Participantes:

-Asier Santos
-Imanol Usandizaga

Orden del día:

->Con el fin de hacer la memoria más corta, Asier pregunta si es adecuado mover secciones como el aprendizaje al anexo.
-Imanol menciona que el tamaño adecuado es de entre 50 y 70 hojas, y que se puede mover al anexo lo que se considere oportuno.

->Asier pregunta si merece la pena añadir el diario informal escrito durante el proyecto, los registros de tiempo exactos y la tabla excel con los resultados del betatesting.
-Imanol propone ponerlos disponibles haciendo uso de un acortador de urls.

->Asier consulta la manera adecuada de organizar la bibliografía
-Imanol propone prescindir del nombre bibliografía y decantarse por referencias. También propone organizar las entradas en base al tipo (wikis, foros, tutoriales, videocanales, etc.)

->Asier pregunta si el glosario es necesario/recomendado.
-Imanol lo considera un extra, que si se hace está bien, pero que no se priorice.

->Asier no sabe con qué tipo de información llenar las secciones de "desarrollo" y "diseño técnico".
-Sobre el desarrollo, Imanol no ve el interés de mostrar código y propone o bien eliminarlo o bien hacer pequeñas menciones a datos de interés.
-Sobre el diseño técnico, propone poner un diagrama de secuencia a modo de referencia, pero no más.

D.8. Acta 31/07/2017

Fecha: 31/07/2017
Lugar: Plataforma Hangouts (Telemática)
Participantes:
-Asier Santos
-Imanol Usandizaga

Orden del día:

->Imanol procede a dar una serie de consejos sobre la memoria:
-Sugiere mencionar el valor del trabajo hecho por VideoLan, así como su potencial efecto.

-Remarcar la complejidad del desarrollo dentro del marco de VLC frente a una aplicación stand-alone.
-Hacer más hincapié en que el TFG es parte de un proyecto más grande y ambicioso. Mencionar pruebas, despliegue y publicación.
-Mencionar que el alcance es conservador por la falta de experiencia.
-En la planificación, incluir sólo la final, o ambas, pero no mitad y mitad (sobre el diagrama de hitos).
-Siempre incluir algo de texto entre secciones y subsecciones (no dejarlo en blanco).
-Reordenar el apartado 3 enteramente.
-Crear un apartado dedicado al Diseño, con datos como la arquitectura, uso de APIs, etc.
-Extender significativamente el apartado de conclusiones (pero no exactamente desde el enfoque de las lecciones aprendidas).
-Otras correcciones menores.

Bibliografía

[AcoustID, 2017] AcoustID (2017). Acoustid. https://acoustid.org/.

[“andrea.carlon”, 2012] “andrea.carlon” (2012). Advanced tag editor. https://forum.videolan.org/viewtopic.php?t=100184.

[Debian Wiki Team, 2015] Debian Wiki Team (2015). Checkinstall on debian wiki. https://wiki.debian.org/CheckInstall.

[Discogs, 2017] Discogs (2017). Discogs. https://www.discogs.com//.

[freeDB, 2017] freeDB (2017). freedb. http://www.freedb.org/.

[Lalinský, 2011] Lalinský, L. (2011). How does chromaprint work? https://oxygene.sk/2011/01/how-does-chromaprint-work/.

[“m33p”, 2009] “m33p” (2009). Editing multiple tracks at once. https://forum.videolan.org/viewtopic.php?t=69532.

[musicBrainz, 2017] musicBrainz (2017). musicbrainz. https://musicbrainz.org/.

[Roberto Ierusalimschy, 2016] Roberto Ierusalimschy, Luiz Henrique de Figueiredo, W. C. (2016). LUA reference manual. https://www.lua.org/manual/5.3/.

[The Qt Company Ltd., 2017] The Qt Company Ltd. (2017). Qt documentation. https://doc.qt.io/.

[“unique_name”, 2014] “unique_name” (2014). File tag batch editing. https://forum.videolan.org/viewtopic.php?t=119426.

[VideoLAN Organization, 2016a] VideoLAN Organization (2016a). Módulo gotime. https://github.com/videolan/vlc/blob/master/modules/gui/qt/dialogs/gototime.cpp.

[VideoLAN Organization, 2016b] VideoLAN Organization (2016b). Videolan documentation. https://wiki.videolan.org/Documentation.

[VideoLAN Organization, 2017a] VideoLAN Organization (2017a). Hacker guide/how to write a module. https://wiki.videolan.org/Hacker_Guide/How_To_Write_a_Module/.

[VideoLAN Organization, 2017b] VideoLAN Organization (2017b). LUA and VLC: Readme. https://github.com/videolan/vlc/blob/master/share/lua/README.txt.

[VideoLAN Organization, 2017c] VideoLAN Organization (2017c). VideoLAN documentation: Code conventions. https://wiki.videolan.org/Code_Conventions/.

[VideoLAN Organization, 2017d] VideoLAN Organization (2017d). VideoLAN documentation: Unixcompile. https://wiki.videolan.org/UnixCompile/.

[VideoLAN Organization, 2017e] VideoLAN Organization (2017e). VideoLAN documentation: Win32compile. https://wiki.videolan.org/Win32Compile/.

[Wikipedia Community, 2016] Wikipedia Community (2016). Espectrograma. https://es.wikipedia.org/wiki/Espectrograma.

[Wikipedia Community, 2017] Wikipedia Community (2017). VLC media player. https://en.wikipedia.org/wiki/VLC_media_player.