Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

PROYECTO FIN DE CARRERA

Desarrollo de una aplicación para la extracción y análisis de metainformación en dispositivos iPhone

Autor:
Javier Checa Betegón

Director:
Manuel Sánchez Chumillas

Ponente:
Álvaro Alesanco Iglesias

Centro Politécnico Superior
Ingeniería en Informática
9 de noviembre de 2011

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

PROYECTO FIN DE CARRERA

Desarrollo de una aplicación para la extracción y análisis de metainformación en dispositivos iPhone

Autor:
Javier Checa Betegón

Director:
Manuel Sánchez Chumillas

Ponente:
Álvaro Alesanco Iglesias

Centro Politécnico Superior
Ingeniería en Informática
9 de noviembre de 2011

Agradecimientos

Son muchas las personas que han contribuido a que haya llegado hasta aquí y hoy pueda leerse este documento. Nombrarlas todas es tarea complicada, así que me resignaré a mencionar sólo a algunas de las más destacables.

Gracias a mis compañeros de carrera por hacer que 5 años de estudio y levantarse temprano por la mañana se conviertan en la mejor etapa de mi vida. Sois muchos y sabéis quiénes sois

A los miembros de la rama de estudiantes del IEEE de la Universidad de Zaragoza que he tenido el placer de conocer, gracias por haberme echado una mano cuando lo he necesitado y por vuestra acertadas recomendaciones.

A Chema Alonso, por ofrecerme la idea para este proyecto y aquella vuelta en el maligno-móvil. A él, y a los compañeros de Informática64, en especial a Alejandro Martín y Manuel Sánchez, gracias por acogerme y ayudarme a dar los primeros pasos. Gracias también a Fernando Oca por sus direcciones en el análisis de los ficheros Office.

Gracias a Ismael Saad por su inestimable ayuda a la hora de desarrollar la interfaz gráfica de esta aplicación.

A mis traductoras personales, Olga Bichert y Jekaterina Vargunina, por ofrecerme su conocimiento del alemán y el ruso, respectivamente.

A Álvaro Alesanco, por orientarme durante estos meses, por ser tan cercano, aguantar mis quejas sobre los problemas que encontraba y por las veloces respuestas a mis correos.

Gracias a www.stackoverflow.com y otras comunidades que han conseguido sacarme de muchos apuros durante estos meses de trabajo.

A los creadores de las clases ZipArchive y FileReader (sec. 4.2) por facilitarme la labor y permitir que me centrara en lo verdaderamente importante.

Y, por supuesto, gracias a mi familia por los sabios consejos cuando los he necesitado y, sobretodo, por su apoyo incondicional y por creer en mí.

Índice general

1. Introducción **1**
- 1.1. Motivación . . . 1
- 1.2. Entorno de aplicación . . . 2
- 1.3. Objetivo . . . 2
- 1.4. Resumen . . . 3

2. Tecnología usada **5**
- 2.1. iOS . . . 5
- 2.2. Objective C . . . 5
  - 2.2.1. Los métodos . . . 6
  - 2.2.2. Paso de mensajes . . . 6
  - 2.2.3. Categorías . . . 6
  - 2.2.4. Internacionalización . . . 7
- 2.3. Xcode . . . 7
  - 2.3.1. Interface Builder . . . 8

3. Análisis **9**
- 3.1. Archivos de texto . . . 9
- 3.2. Archivos binarios . . . 10
- 3.3. Archivos *zip* . . . 10
- 3.4. Planificación . . . 10

4. Diseño de la aplicación **13**
- 4.1. La clase MetaAnalyzer . . . 13
- 4.2. Clases, categorías y archivos de apoyo . . . 15
  - 4.2.1. Categorías de NSString y NSData . . . 15
  - 4.2.2. Categoría de NSMutableDictionary . . . 15
  - 4.2.3. Clase ZipArchive . . . 16
  - 4.2.4. Clase FileReader . . . 16
  - 4.2.5. Archivos *strings* . . . 16
- 4.3. La interfaz gráfica . . . 16
- 4.4. Pruebas . . . 20

5. Conclusiones **21**
5.1. Problemas encontrados 21
5.2. Posibles mejoras 22
5.3. Reparto de horas invertidas 22
5.4. Opinión personal 25

A. Categorías de MetaAnalyzer **27**
A.1. Categoría 3gp 27
A.2. Categoría AIFF 27
A.3. Categoría ASF 28
A.4. Categoría gif 28
A.5. Categoría HTML 29
A.6. Categoría iWork 29
A.7. Categoría jpg 30
A.8. Categoría mov 30
A.9. Categoría mp3 31
A.10. Categoría mp4 31
A.11. Categoría Office binario 32
A.12. Categoría Office zip 33
A.13. Categoría ogg 33
A.14. Categoría pdf 33
A.15. Categoría RIFF 34
A.16. Categoría rtf 34
A.17. Categoría tiff 35
A.18. Categoría XML 35
A.19. Categoría XMP 36
A.20. Categoría *zip* 36

B. Manual de usuario **37**

C. Pruebas **49**

Capítulo 1

Introducción

La información ha jugado siempre un papel fundamental a todos los niveles y en concreto para el empresarial: datos de empleados, de clientes, de facturación, etc. Hoy en día esta importancia sigue vigente y, con el desarrollo de las nuevas tecnologías, aparecen nuevas amenazas de fuga o pérdida de esta información: ataques malintencionados por medio de virus o *troyanos*, dispositivos de almacenamiento masivo con datos sensibles sin protección o datos relevantes cedidos de manera involuntaria a través de otros archivos son sólo algunos ejemplos.

La *metainformación*, o información de la información, que de un documento podría entenderse como el nombre del autor o el programa con el que fue editado, puede resultar útil a la hora de clasificarlo y buscarlo, por ejemplo. Actualmente la mayoría de los archivos que se crean llevan algún tipo de metadato asociado desde el primer momento, lo que puede suponer un problema si no se tiene un control sobre ello, pues, al compartir un documento, se puede, además, facilitar información no deseada. Esto hace referencia a la tercera amenaza mencionada anteriormente.

Los teléfonos de empresa, en especial los *smartphones*, debido a las amplias posibilidades que ofrecen, son muy propensos a almacenar un gran volumen de datos, por lo que pueden suponer un foco importante de fuga de información.

1.1. Motivación

Poco después del comienzo del curso 2006-2007, mi primer año de carrera, tuve la oportunidad de asistir a uno de los *Code Camp* organizados por Microsoft, donde se trataron y organizaron actividades sobre temas muy interesantes, a pesar de que por aquél entonces la gran mayoría me fueran ajenos. Allí conocí a Chema Alonso, gurú de la seguridad informática que

fundó su propia empresa, Informática64, hace ya más de 11 años, y desde entonces he asistido a varias de sus charlas y mantenido el contacto con él.

A la hora de buscar un proyecto fin de carrera, puesto que ya desde pequeño encontré la criptografía y, no mucho después, la seguridad de la información, muy interesante, decidí contactar con Chema en busca de ideas. Me propuso realizar una aplicación iPhone para extraer metainformación de archivos puesto que no se contaba con herramientas con esas características. Me pareció una idea estupenda debido a todo lo que podía aprender por el camino al no saber nada de programación para dispositivos iOS (el sistema operativo utilizado por iPhone, iPod e iPad), así que acepté y poco después la propuesta de proyecto estaba entregada.

1.2. Entorno de aplicación

Tal y como se deja entrever en la introducción, los archivos pueden contener información, aunque no seamos conscientes de ello, que represente un problema de seguridad.

Para un usuario normal, puede suponer revelar qué software utiliza, con qué versión y, por ende, con qué vulnerabilidades no corregidas, su nombre real o incluso una aproximación de las horas a las que está frente al ordenador analizando las fechas de creación y modificación de sus archivos.

Si vamos al caso empresarial, además de lo mencionado para usuarios medios, hay que sumar que los archivos pueden contener información sobre clientes. Por ejemplo, muchos archivos tienen un campo "descripción" o "asunto" que puede revelar información sensible. Es más, el simple hecho de dar a conocer la versión del software usado puede exponer a la empresa a ataques o software malintencionado dirigido hacia ella, lo que puede derivar en una muy importante fuga de información.

1.3. Objetivo

Este proyecto fin de carrera busca desarrollar una aplicación para iPhone, uno de los *smartphones* más comunes del mercado, que sea capaz de analizar ficheros dentro del dispositivo en busca de información relevante.

En la actualidad no se dispone de ninguna aplicación que cumpla con estas características, por lo que el PFC se presenta como una solución para esta necesidad.

1.4. Resumen

Cuatro capítulos siguen a este, cada uno tratando un tema específico, antes de llegar a los apéndices.

- **Capítulo 2** Tecnología usada. En este apartado se explican los recursos de los que se ha dispuesto para elaborar la aplicación. Se hace referencia tanto al lenguaje utilizado como a las herramientas.
- **Capítulo 3**: Análisis del problema y de los distintos tipos de ficheros con los que se va a trabajar.
- **Capítulo 4**: Diseño de la aplicación a la vista de los recursos disponibles y la naturaleza del problema.
- **Capítulo 5**: Conclusiones finales, donde se incluyen problemas encontrados, posibles mejoras y una opinión personal.
- **Apéndice 1**: Detalles de la implementación de las categorías de la clase principal.
- **Apéndice 2**: Manual de usuario detallando cómo utilizar la aplicación.
- **Apéndice 3**: Explicación de las pruebas realizadas para comprobar el correcto funcionamiento de la aplicación.
- **Bibliografía**: Lista de libros a los que se hace referencia y enlaces con información utilizada para el desarrollo del proyecto. Todos los enlaces que aparecen han sido comprobados por última vez el 9 de noviembre de 2011.

Capítulo 2

Tecnología usada

Los dispositivos iPhone, así como los iPod e iPad de Apple utilizan iOS como sistema operativo, Objective C como lenguaje de programación y Xcode como SDK[1].

2.1. iOS

iOs proviene de la contracción de iPhone OS y es un derivado del sistema operativo utilizado por los ordenadores de Apple: Mac OS. Utiliza *Cocoa Touch* como set de herramientas para la interfaz gráfica y está escrito en C, C++ y Objective C.

2.2. Objective C

Aunque apareció en 1980 por primera vez, hoy en día ha aumentado en gran medida su popularidad debido a la programación de aplicaciones para Mac OS e iOS. Se trata de un lenguaje orientado a objetos que fue creado como un superconjunto de C con similitudes con Smalltalk. Objective C se basa en un modelo de paso de mensajes a objetos, siendo estos mensajes nombres de los métodos de los objetos junto a sus respectivos parámetros.

Si bien aprender un lenguaje nuevo siempre cuesta tiempo, uno se acostumbra rápidamente a Objective C y a su manera de hacer las cosas. Para conocer las costumbres de programación y las posibilidades de este lenguaje, a la par que se realiza una aproximación progresiva, es recomendable comenzar leyendo *Programming in Objective C 2.0* [1]. No obstante, y como el lector puede no tener el libro u otra documentación a mano, a continuación

[1] *Software Development Kit*, kit de desarrollo de software

se indican algunas de las particularidades de este lenguaje que se darán por conocidas más adelante.

2.2.1. Los métodos

Por norma general, los métodos en Objective C son muy auto-explicativos e indican sus parámetros tras símbolos de dos puntos (':') según donde proceda en el nombre. La mejor forma de entender esto es mediante un ejemplo (en la sección 2.2.2 se explica el significado del carácter '-' al principio):

```
- (void) guardaUnEntero:(int)elEntero yUnCaracter:(char)elCaracter
```

El método indicado recibe un entero y un carácter a través de sus parámetros "elEntero" y "elCaracter", respectivamente, y al finalizar no devuelve nada.

Si se quiere nombrar este método en un escrito, se indica como guardaUnEntero:yUnCaracter:. Los nombres de métodos y variables suelen comenzar con minúscula y cada nueva palabra se inicia con mayúscula para facilitar su lectura. Otra opción es usar un guión bajo ('_') para separar palabras. Habitualmente sólo los nombres de clases comienzan con mayúscula.

2.2.2. Paso de mensajes

Como ya se ha comentado, Objective C se basa en el paso de mensajes de la forma

```
[receptor mensaje],
```

donde mensaje es el nombre del método al que se llama, que sigue la estructura mencionada en la sección 2.2.1.

Los mensajes pueden enviarse a una instancia de una clase o a la clase en sí, lo que se indica en su declaración con '-' o '+', respectivamente. Se envía el mensaje a la clase cuando se quiere realizar una operación sobre ella, como por ejemplo obtener una nueva instancia.

Los métodos de instancias tienen el objetivo de asignar un valor a una propiedad de esa instancia en particular, obtenerlo, hacer que se muestre por pantalla, etc.

2.2.3. Categorías

Similar a reabrir clases en Ruby o a crear extensiones en C#, en Objective C hay formas de añadir funcionalidad a las clases. La forma de hacerlo es a través de categorías.

En las categorías se incluye la cabecera de la clase original, se declara una con el mismo nombre y se añade, entre paréntesis, el nombre de la categoría. A continuación se declaran e implementan los métodos que se desean y éstos serán añadidos a la clase original, aunque no tengamos acceso a su código.

Esto resulta útil para, por ejemplo, hacer que la clase NSString, utilizada para las cadenas de texto, tenga un método que invierta el orden de los caracteres. Todos los paquetes que incluyan la categoría podrán hacer uso de esta nueva funcionalidad.

Usar categorías también permite redefinir métodos: si se declara un método con el mismo nombre que uno que la clase posee ya, será el nuevo el que se ejecute a partir de entonces.

2.2.4. Internacionalización

Cuando uno desarrolla una aplicación de estas características, está interesado en llegar al mayor público posible, y para ello, puesto que no todo el mundo habla el mismo idioma, es interesante poder ofrecer el resultado en diferentes lenguajes. Es más, también puede ser importante que según la localización del usuario se cambie una imagen por otra, las unidades utilizadas para expresar cierta magnitud sean distintas, etc. Objective C, en conjunción con Xcode, permite hacer todo esto.

Más información acerca de este tema puede encontrarse en la documentación oficial de Apple para desarrolladores [2].

2.3. Xcode

Para programar, compilar y probar aplicaciones escritas en Objective C, ya sean para Mac OS o para iOS, se utiliza Xcode como entorno de programación. Es necesario que la versión de esta aplicación, similar a Eclipse, sea igual o superior a la 3 para contar con el simulador de iPhone que trae consigo y poder programar para este dispositivo.

La versión utilizada para desarrollar la aplicación de este proyecto es la 4.1, que en el momento de adquirirla era de pago y que posteriormente se distribuyó de forma gratuita para la versión *Lion* del sistema operativo Mac OS X. Esta versión incorpora, como se ha comentado, un simulador de iOS, entre otras cosas, y es con él que se ha probado el correcto funcionamiento de la aplicación. Unas semanas antes de la finalización de este proyecto se liberó la versión 4.2.

Xcode facilita en gran medida la programación al ofrecer funciones de autocompletar, corrección de código y compilaciones virtuales en tiempo real

para descubrir errores o problemas a la par que uno escribe. También incorpora una sección en la que se muestran todos los ficheros del proyecto, tanto recursos como de código, que pueden ser organizados en carpetas virtuales.

Además, se incluyen diversas herramientas para facilitar el desarrollo de aplicaciones, como *Interface Builder* o *Instruments*. Esta última permite supervisar diferentes parámetros como el uso de memoria, los *memory leaks*, procesos *zombie*, uso de CPU, etc.

2.3.1. Interface Builder

Es importante destacar *Interface Builder*, incluído en la herramienta, que permite programar la interfaz gráfica con sencillos "pinchar y arrastrar". Naturalmente, para tener un control más preciso de la misma hace falta escribir código, pero hace sencillo el obtener una primera aproximación.

Capítulo 3

Análisis

Una vez se conocen las herramientas de las que se dispone, y antes de comenzar a utilizarlas para implementar una solución el problema, hay que entender el problema en sí: para un archivo dado, obtener los metadatos que contiene. Esto conlleva conocer la estructura del fichero, cómo se organiza, para poder buscar la información en los lugares en los que puede estar presente.

En este proyecto se distinguen tres tipos de archivos: los archivos de texto, los archivos binarios y los archivos *zip*.

3.1. Archivos de texto

Por archivos de texto se entienden aquéllos cuyo contenido es legible si se abren con un simple editor de textos, como *wordpad*, *gedit* o *textEdit*, por ejemplo.

Pese a que los formatos de estos archivos suelen estar divididos en partes, señalizadas con *tags* o etiquetas, habitualmente no se indica el tamaño de la misma, lo que obliga realizar una lectura secuencial del fichero en busca de lo que interesa, no pudiendo, sencillamente, desplazar el lector.

Además, al tratarse de texto plano, se hace necesario realizar comparaciones entre cadenas de texto para localizar el comienzo y el final de un dato. En ocasiones esto no resulta inmediato y supone realizar numerosas comprobaciones.

Dentro de esta primera clasificación se encuentran los archivos con extensión "rtf" (*Rich Text Format*), "html" (*HyperText Markup Language*) o "xml" (*eXtensible Markup Language*)

3.2. Archivos binarios

Los archivos binarios tienen una estructura interna definida en su formato y gran parte de su contenido es ilegible porque no tiene representación como una serie de caracteres. Para explorarlos se utilizan editores hexadecimales, que muestran su contenido en hexadecimal junto a su equivalencia en texto.

Es común que aparezcan estructuras de tamaño constante o siguiendo un patrón similar a *[Tamaño][Identificador][Contenido]*, con "Tamaño" e "Identificador" de longitud constante, por lo que resulta sencillo saltar porciones de fichero que no contienen datos relevantes.

Algunos de los archivos binarios con los que se trabaja más adelante son los "mov" (formato multimedia de *QuickTime*, de Apple), "jpg" (o "jpeg", formato de compresión de imágen con pérdidas), "mp3" (formato muy extendido de compresión de audio con pérdidas) y los pertenecientes a Microsoft Office hasta la versión 2003, incluida, entre otros.

3.3. Archivos *zip*

Zip es un formato de compresión de archivos. Algunos de los archivos que la aplicación deberá analizar, como por ejemplo los pertenecientes a Micrososft Office a partir de la versión 2007, incluida, o los de iWork, paquete de ofimática de Apple, son en realidad varios archivos en uno, y utilizan *zip* para presentarse como uno solo. De hecho, cambiar la extensión de uno de estos archivos a ".zip" hará que nuestro descompresor favorito lo reconozca y nos permita ver el contenido, que no es sino una serie de carpetas y archivos.

La aparición en juego del formato *zip* hace necesario que la aplicación sea capaz de descomprimir contenedores de este tipo.

3.4. Planificación

A pesar de las similitudes que puedan aparecer entre formatos dentro de una misma clasificación, lo cierto es que sus diferencias hacen muy difícil abstraerse y generalizar el análisis de varios de ellos. Esto influirá en la forma en la que se diseñe la aplicación.

A la vista de la naturaleza del problema y de las herramientas, se decide seguir el siguiente plan:

- Recabar información básica sobre los distintos formatos, que es lo que se ha comentado ahora.

- Desarrollar una aplicación simple para explorar el sistema de archivos.
- Trabajar en una clase de análisis de archivos y profundizar más en cada formato a la hora de trabajar en él.
- Realizar pruebas de la clase.
- Desarrollar una interfaz gráfica más elaborada.
- Realizar pruebas sobre la interfaz.
- Probar la aplicación a fondo con muchos archivos.
- Redactar la memoria, que es el presente documento.

Además, y para facilitar el último paso, durante las distintas fases se irán recogiendo notas sobre los avances, problemas encontrados o posibles ampliaciones futuras.

Capítulo 4

Diseño de la aplicación

Los diferentes componentes de la aplicación implementada se pueden agrupar en tres conjuntos diferentes: los encargados de gestionar la interfaz gráfica, los pertenecientes a la clase *MetaAnalyzer*, que gestiona la extracción de la metainformación, y clases, categorías y archivos de apoyo destinadas a facilitar una determinada labor. A continuación se explican con más detalle cada una de estas partes.

4.1. La clase MetaAnalyzer

Esta clase presenta, en su cabecera, un único método, llamado analyzeFileAtPath:withError:, que devuelve un diccionario de pares clave-valor con la información obtenida o indica en su parámetro de error si algo ha ocurrido. Este archivo también declara una serie de *defines* para los códigos de error y otros en los que se crean cadenas internacionalizadas (ver sección 2.2.4) para las distintas posibles claves así como para algunos valores sencillos. Según el idioma seleccionado en el dispositivo, la aplicación mostrara el texto en un idioma u otro, siempre que la traducción esté disponible. También se define la cadena separadora que permite almacenar varios valores para una misma clave (más en 4.2.2), que en este caso es " | ".

En la implementación de la clase, se importa el fichero "MAIncludes.h", que a su vez incluye las cabeceras de todas las categorías de esta clase, que se explican en detalle en el apéndice A. Todas ellas incorporan gestión de excepciones para capturar posibles problemas durante la ejecución. Los formatos soportados en la versión de este proyecto se presentan en la tabla 4.1.

Aunque algunos formatos presentan similitudes entre ellos, ha sido necesario realizar un extenso y concienzudo estudio de cada uno para poder desarrollar sus métodos correspondientes. Además de la especificación, que

Formato	Tipo de fichero	Extensiones
3gp	Binario	.3gp
AIFF	Binario	.aif, .aiff
ASF	Binario	.wma, .wmv
gif	Binario	.gif
HTML	Texto	.htm, .html
iWork	*Zip*	.key, .pages, .numbers
jpg	Binario	.jpg, .jpeg
mov	Binario	.mov
mp3	Binario	.mp3
mp4	Binario	.mp4
Office binario	Binario	.doc, .xls, .ppt
Office Zip	*Zip*	.docx, .xlsx, .pptx
ogg	Binario	.ogg
pdf	Binario	.pdf
RIFF	Binario	.wav, .avi
rtf	Texto	.rtf
tiff	Binario	.tif, .tiff
XML	Texto	.xml
XMP	Texto	.xmp

Cuadro 4.1: Formatos soportados por la aplicación

no está disponible para todos los formatos, también se ha hecho un uso intensivo de un editor hexadecimal para comprender la estructura interna de cada tipo de fichero, ya que he en ocasiones no se sigue al pie de la letra.

En cuanto al método antes mencionado, su funcionamiento es simple: comprobar la extensión del fichero que se indica por la ruta recibida, asegurarse de que la firma del fichero corresponde con la del tipo de fichero y llamar al método de la categoría adecuada para analizarlo. Si la firma y extensión no coinciden, se llama al método analyzeFileAtPathCheckingSignature:withError:, no declarado en la interfaz, para elegir qué método usar en función de la firma.

Sólo se mantiene un método visible en la cabecera, y el resto "ocultos" en categorías, para que resulte inmediato decidir qué método llamar en caso de que la clase se reutilice en otras aplicaciones. Además, se ha implementado una categoría por formato de fichero, no por tipo, lo que podría ocasionar que no resultara trivial elegir el método correcto. Por ejemplo, los archivos *Windows Media Audio* (wma) y *Windows Media Video* (wmv) siguen el formato *ASF*, como se puede ver en la tabla 4.1.

4.2. Clases, categorías y archivos de apoyo

En ocasiones ha resultado necesario poder realizar acciones que no venían implementadas por defecto en las clases disponibles. Cuando una situación así ha aparecido, por ser algo ajeno a la clase *MetaAnalyzer*, se han creado categorías de clases ya existentes o se han añadido nuevas clases encargadas de llevar a cabo esa función.

4.2.1. Categorías de NSString y NSData

NSString es la clase que se utiliza para trabajar con cadenas de texto. Para esta aplicación, se han implementado dos categorías: una para trabajar con cadenas de caracteres hexadecimales y otra para buscar subcadenas. En la primera se pueden encontrar métodos para traducir la cadena a texto, calcular el valor numérico que representa, cambiar de *little* a *big endian* y viceversa, etc.

Por su parte, la clase NSData permite almacenar bytes de información y se ha utilizado en la lectura de ficheros. Éstos pueden almacenar información en *little* y *big endian* y en ocasiones es necesario leer los datos, cambiar su *endianness* y usarlos. Para evitar operaciones innecesarias, en lugar de interpretar la información como una cadena hexadecimal, cambiar de un formato a otro y volver a su representación en bytes, se ha creado una categoría que añada la funcionalidad de invertir el orden de los bytes.

4.2.2. Categoría de NSMutableDictionary

En la aplicación se utilizan diccionarios modificables[1]para almacenar los pares clave-valor de los metadatos encontrados.

En un mismo fichero pueden aparecer varios metadatos del mismo tipo o incluso el mismo varias veces. En el caso de esta aplicación se trabaja con cadenas, pero los diccionarios pueden contener objetos de cualquier tipo. Es por ello que por defecto la clase no incorpora un mecanismo para concatenar valores al ya existente para una clave.

En esta categoría se implementa un método que, primero, comprueba si ya había un valor asociado a la clave. En caso de no existir, crea una nueva entrada con el valor dado; si ya había algún valor presente, comprueba si alguno de los distintos valores, separados por la cadena separadora definida en la clase *MetaAnalyzer*, coincide con el que se quiere introducir y, en función de esto último, decide si concatenarlo, junto con la cadena separadora, o rechazarlo por duplicado.

[1]Como tantos otros objetos de Objective C, la versión *mutable* o modificable permite añadir, eliminar o modificar los datos que contiene

4.2.3. Clase ZipArchive

Para trabajar con archivos *zip* se hace uso de la clase ZipArchive, disponible en *Google Code* [3]. En ella se presentan diversos métodos para comprimir y descomprimir archivos, con o sin contraseña. En esta aplicación se ha hecho uso únicamente de la funcionalidad de descompresión.

4.2.4. Clase FileReader

Algunos formatos de archivos de texto utilizan saltos de línea para separar unidades. La clase FileReader, extraída del proyecto LineReader [4], implementa la lectura de líneas en ficheros de texto con un lectura por bloques. En este caso se han añadido nuevos métodos y modificado otros según las necesidades de la aplicación.

4.2.5. Archivos *strings*

Los archivos strings se utilizan para almacenar las traducciones de las cadenas internacionalizadas (sección 2.2.4). Las aplicaciones contienen uno de estos archivos por cada uno de los idiomas soportados. Si una lengua no tiene su archivo correspondiente, la aplicación muestra el texto introducido por defecto, que en este caso es en inglés.

4.3. La interfaz gráfica

La implementación de la interfaz gráfica se ha realizado con la ayuda de *Interface Builder* (sección 2.3.1) y es muy intuitiva. Al iniciar la aplicación se muestra una pantalla de presentación (figura 4.1) desde la que el usuario puede pasar a explorar el sistema de ficheros en una nueva pantalla.

En el explorador de archivos se presenta una tabla que contiene los ficheros del directorio actual, encima de la cual se indica el nombre de este último y se muestra un botón con el texto internacionalizado “Atrás” para subir un nivel. La figura 4.2 muestra un ejemplo de lo que puede encontrarse un usuario mientras navega una carpeta con distintos ficheros.

Si se pulsa sobre un directorio se accede a él, y, si se hace sobre un fichero, se crea una instancia de la clase *MetaAnalyzer* para que analice el fichero. Los metadatos encontrados se muestran a continuación en otra pantalla, agrupados según la información que representan. Por ejemplo, si se pulsa sobre el archivo “2011-08-16 13.51.42.jpg” de la figura 4.2, se obtienen los metadatos mostrados en la figura 4.3.

Figura 4.1: Menú principal de la aplicación

Si el fichero no contuviera metadatos o se produjera algún error, se mostraría un aviso por pantalla con detalles de lo ocurrido

En el apéndice B, que contiene el manual de usuario, puede encontrase un recorrido en profundidad y con más imágenes de la interfaz gráfica así como del funcionamiento de la aplicación en general.

Figura 4.2: Explorando el sistema de ficheros

Figura 4.3: Los datos se muestran en una nueva pantalla

4.4. Pruebas

Tal y como se ha indicado en la sección sobre planificación 3.4, ha habido varias fases de pruebas durante el desarrollo de la aplicación. La primera ha sido coetánea a la implementación de las distintas categorías de la clase principal y, en ella, se han creado archivos de prueba específicos para el formato en el que se estaba trabajando.

Una vez terminadas las categorías y creada la interfaz de usuario, se ha pasado a probar esta última para, finalmente, realizar pruebas en profundidad de la aplicación completa. Para ello, además de crear nuevos archivos, se han descargado otros de diferentes fuentes, que se han encontrado gracias a motores de búsqueda en la red. Además del análisis realizado por la aplicación, también se han estudiado los distintos ficheros *a mano* para comprobar que lo extraído era correcto.

Las pruebas se tratan en más detalle en el apéndice C.

Capítulo 5

Conclusiones

Finalmente, en este capítulo se van a comentar los resultados a la conclusión del proyecto, terminando con una opinión personal en la que se resumen distintos aspectos del proceso de desarrollo del proyecto.

5.1. Problemas encontrados

Durante el desarrollo del proyecto se han dado momentos en los que se ha tenido que dedicar un tiempo y esfuerzo adicional a solventar un determinado problema. Algunos de ellos se comentan a continuación junto con la solución que se les ha dado:

- En ocasiones podían aparecer varios metadatos del mismo tipo y con los métodos por defecto sólo podía guardarse el primero. Para poder almacenar varios, se creo una categoría del diccionario (ver sección 4.2.2 y se definió un separador de cadenas.
- La documentación oficial de *Microsoft* sobre los formatos de *Office* binarios resulta confusa. La facilitada por *Open Office* sobre este mismo formato resulta mucho más esclarecedora. Una vez con los conceptos claros, la documentación original se utilizó como complemento con los ejemplos que presenta.
- Muchos de los archivos *pdf* analizados no respetan el formato que se indica en la web de *Adobe*. Para soportar estos ficheros, se estudió las diferencias que estos archivos presentaban y se ha aumentó la robustez de los métodos correspondientes, extendiendo su compatibilidad a las distintas peculiaridades.
- Trabajando en la interfaz gráfica, apareció el problema de presentar metadatos en cadenas largas. Las celdas de la tabla en la que se muestran no admiten, por defecto, la posibilidad de desplazar el contenido,

por lo que parte de la información quedaba fuera del rango a mostrar. Las celdas fueron modificadas para incluir esta funcionalidad. Esto conllevó un estudio algo más profundo del funcionamiento de determinados elementos de la interfaz, pues a raíz de estos cambios surgieron otros problemas que posteriormente fueron convenientemente solucionados.

- Para la fase de pruebas, generar archivos no era una opción viable: hacían falta más y su generación desde el mismo ordenador con el mismo programa ocasionaba que los metadatos fueran siempre los mismos (exceptuando las fechas). La solución vino de mano del buscador de Google y el uso de "filetype:" para buscar archivos de un tipo determinado. Aún así, de algunos formatos ha sido especialmente complicado obtener documentos.

5.2. Posibles mejoras

La aplicación desarrollada no es perfecta, naturalmente, y si se quisiera desarrollar más, algunas de las posibles mejoras podrían incluir:

- Implementar categorías para formatos adicionales.
- Mejorar o añadir funcionalidades a algunas categorías, por ejemplo, desarrollando soporte para codificaciones adicionales: cirílico, griego, japonés, etc.
- Aumentar el número de idiomas soportados por la aplicación traduciendo los archivos *strings* (ver sección 2.2.4).
- Añadir soporte para imágenes: algunos ficheros almacenan imágenes como metadatos y resultaría interesante poder mostrarlas entre los resultados (actualmente se muestra la cadena localizada"Imagen").
- Implementar y ofrecer la limpieza de metainformación.

5.3. Reparto de horas invertidas

El comienzo de este PFC podría datarse de finales del 2010, momento en el que se contactó con Chema Alonso, se obtuvo la idea, un director de proyecto en Informática 64 y se realizó la propuesta. Sin embargo, hasta mediados Agosto del 2011, tras una mudanza a Alemania, y una vez terminadas todas las clases, tan sólo se realizó parte de la documentación sobre los distintos archivos y formatos. Esto se debió en parte a que se estaban

cursando asignaturas que requerían bastante trabajo y en parte a que la fase de documentación resultaba especialmente tediosa.

A partir de la fecha antes mencionada, se retomo el proyecto y se trabajó en él, casi diariamente, hasta los primeros días de Noviembre. Momento en el que la aplicación había superado ya la fase de pruebas y se había redactado y revisado el presente documento.

La fase de análisis fue mucho más interesante al aprender Objective C y Xcode, a los que uno se acostumbra rápidamente. La implementación, naturalmente, resultó muy satisfactoria al comenzarse a ver los resultados del tiempo invertido previamente. Debido a esto, el avance en el desarrollo de la aplicación aceleró de manera considerable durante las últimas semanas.

Para la redacción de la memoria, el presente documento, se ha utilizado LaTeX. Tampoco se había usado antes y ha sido necesario, de nuevo, documentarse y aprender antes de obtener resultados. No obstante, Álvaro Alesanco, ponente de este PFC, me envió un archivo de ejemplo en el que basarme. Esto ha permitido ahorrar tiempo en esta fase.

También ha sido posible reducir el tiempo de desarrollo al incorporar las clases “ZipArchive” y “LineReader” (secciones 4.2.3 y 4.2.4, respectivamente) para tratar con necesidades concretas que no eran el principal objetivo de esta aplicación.

En total, el PFC se ha desarrollado en poco menos de 340 horas. En la siguiente tabla (fig. 5.1) se presenta cómo se ha repartido el tiempo invertido.

Implementar la clase principal y sus categorías ha llevado algo más de 120 horas, mientras que la documentación de los distintos formatos ha requerido unas 62 horas. También han hecho falta en torno a 17 horas de documentación de Xcode y Objective C para poder trabajar cómodamente, así que la fase de “aprendizaje” en este proyecto ha sido intensa.

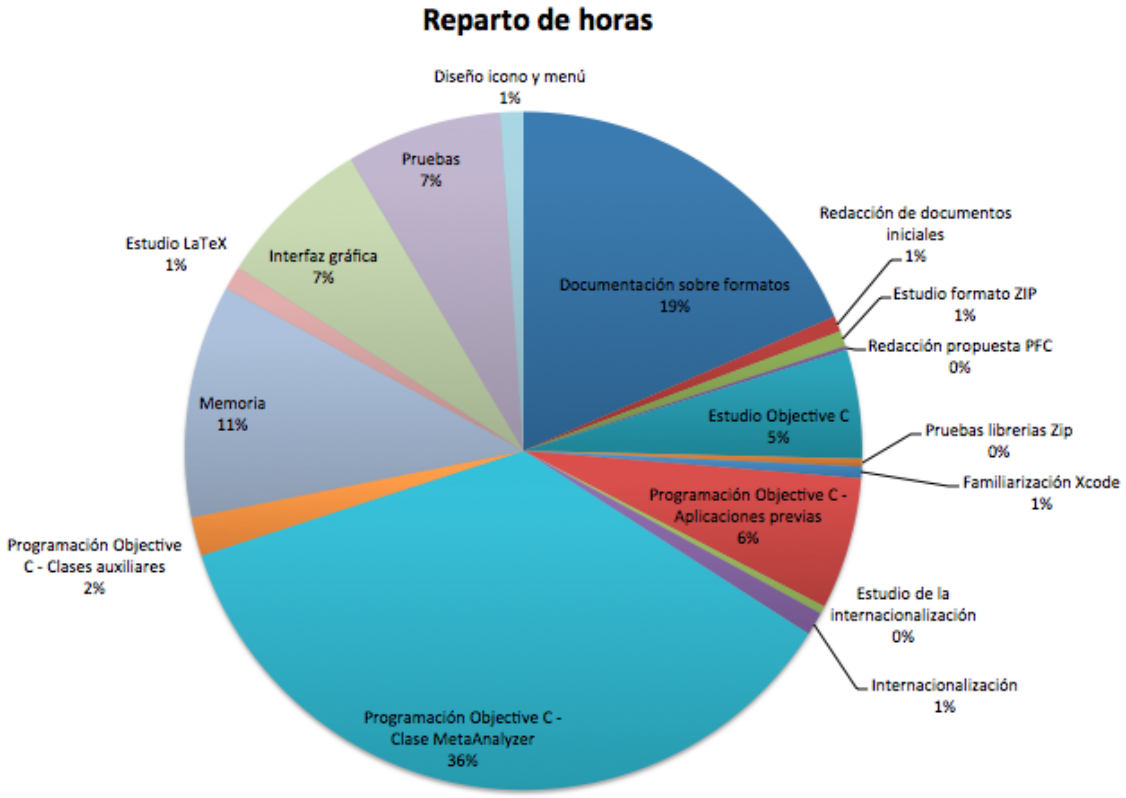

Figura 5.1: Esquema del reparto de horas

5.4. Opinión personal

Comenzar un nuevo programa suele hacerse un poco cuesta arriba, pero comenzar un proyecto en el que se ha de hacer uso de un lenguaje y unas herramientas no utilizadas hasta el momento es un reto. Si además se tiene en cuenta que este proyecto se ha realizado "a distancia", en el que no se han fijado fechas límite ni se ha controlado cuánto tiempo dedicaba a su desarrollo, también ha resultado ser un interesante ejercicio de autodisciplina.

Como puntos positivos hay que destacar el haber aprendido a utilizar un nuevo lenguaje y herramienta, el haber desarrollado una aplicación para uno de los smartphones más populares del mercado y, naturalmente, el haber obtenido experiencia en lo que supone llevar a cabo por uno mismo un proyecto de tamaño algo mayor a lo que se está acostumbrado.

Me han gustado las facilidades que ofrece Xcode para desarrollar y lo rápido que se adapta uno a programar en Objective C. Además, la solución para la internacionalización de aplicaciones me ha parecido acertada y he podido comprobar lo sencillo que resulta incorporar nuevos idiomas a un proyecto.

Por contra, a veces el contacto con la empresa no ha sido todo lo fluido y rápido que me habría gustado, pudiendo pasar semanas hasta recibir una respuesta. La fase de documentación ha sido algo tediosa porque ha tenido mucha carga y las pruebas han sido bastante repetitivas: descargar, analizar *a mano* y comprobar con la aplicación.

Sin embargo, el principal punto negativo tiene que ver con el hecho de no poder probar la aplicación en un dispositivo real si no es pagando una suscripción: a pesar de que Xcode incluye un simulador, no sé hasta qué punto se limita el uso de memoria o la velocidad de procesamiento. No obstante, puesto que se ha minimizado en la medida de lo posible el uso de recursos, se espera un funcionamiento correcto.

Apéndice A

Categorías de MetaAnalyzer

Para la clase MetaAnalyzer se ha creado una categoría por formato de archivo analizable. A continuación se detallan brevemente los formatos para los que añaden soporte.

A.1. Categoría 3gp

3gp sigue un formato de cajas muy imitar a los átomos del formato mov (sección A.8): cada una comienza con 4 bytes indicando el tamaño total de la misma seguidos de otros 4 bytes con el tipo de caja de la que se trata.

Los metadatos (si los hay), se encuentran en la caja "udta", que a su vez está contenida por la caja "track" o "moov". El método principal de esta categoría itera de caja en caja buscando "moov" y, seguidamente, busca dentro "udta" o "trak", saltando el resto. Si se encuentra con la segunda, busca dentro la primera.

Una vez la aplicación encuentra la caja, itera sobre cada una de sus subcajas, extrayendo la información. No todas las subcajas tienen el mismo formato (aunque siempre manteniendo los primeros 8 bytes como se ha comentado antes), por lo que ha sido necesario implementar varios métodos para los distintos tipos.

Por cada subcaja encontrada se actualizan los pares clave-valor del diccionario de metadatos, que es devuelto al finalizar el análisis.

Documentación sobre este formato puede encontrarse en [5].

A.2. Categoría AIFF

En esta categoría se ha implementado la extracción de metainformación tanto del formato AIFF como de AIFF-C. En ellos, al igual que en otros

formatos, el fichero se estructura en "chunks", cuyos primeros 8 bytes indican su tamaño y tipo.

En este formato los "chunks" que pueden contener datos son "COMT", "AUTH" , "NAME", "(c) " y "ANNO", por lo que el método implementado itera buscándolos. Para cada uno de ellos se extraen los datos, en forma de cadena. En el caso de los comentarios (primer "chunk" de los comentados), aparece antes la fecha en la que se realizó, a la que se da formato y se añade a la cadena. Además, pueden aparecer varios "chunks" de comentario, por lo que el diccionario se actualiza añadiendo los nuevos comentarios separados por la cadena " | ".

Documentación sobre este formato puede encontrarse en [6].

A.3. Categoría ASF

En esta categoría se incluye soporte para los archivos wma y wmv.

Cada archivo está constituido por objetos, identificados por un "GUID" (Global Unique IDentifier) de 16 bytes, divididos en 4 números. Los tres primeros, de 4, 2 y 2 bytes, respectivamente, se almacenan en *little endian*, mientras el último, de 8 bytes, en *big endian*. Adicionalmente, este formato utiliza *little endian*, por lo que hay que invertir el *endian*.

En esta categoría se define una estructura de datos para leer y almacenar los "GUID" de forma más cómoda.

Siguiendo a estos 16 bytes aparecen 8 bytes indicando el tamaño del objeto, de forma que los que no nos interesan pueden ser saltados.

Los objetos a analizar en este formato se encuentran dentro de la cabecera, o *Header Object*, y son: *Content Descriptor Object*, *Extended Content Descriptor Object*, *Content Branding Object* y *Header Extension Object*, que contiene a su vez *Metadata Object* y *Metadata Library Object*. Para cada uno de estos objetos se ha implementado un método específico que extrae los datos.

Documentación sobre este formato puede encontrarse en [7].

A.4. Categoría gif

Para implementar esta categoría se ha tomado como referencia el implementado por Sean M. Burke (sburke@cpan.org) en python [9] debido a que la documentación encontrada resultaba ligeramente confusa en algunos aspectos.

Los ficheros gif se componen de bloques, de los cuales interesa el bloque de comentarios. A diferencia de otros formatos, los bloques gif no tienen un indicador del tamaño del mismo. Además, no es sencillo identificar dónde comienza un nuevo bloque. Sin embargo, lo que sí aparece es el identificador del bloque, por lo que resulta posible saltar aquéllos que aparezcan y sean conocidos. Se han implementado métodos para dejar atrás los bloques *Image Descriptor* y *Logical Screen Descriptor*, que pueden tener un tamaño variable.

Los comentarios se encuentran en los *Extension Block* que contienen a su vez 0 ó más subbloques, precedidos por su tamaño. En este caso, sí es posible saltarlos en el caso de que no se trate de un subbloque de comentarios. Si se encuentra uno de este tipo, se extrae la información y se incorpora al diccionario, concatenando su contenido al ya existente.

Documentación sobre este formato puede encontrarse en [8].

A.5. Categoría HTML

Los archivos HTML, que son ficheros de texto, suelen tener una cabecera al comienzo donde pueden incluirse metadatos, que siguen el esquema *<meta name="clave" content="valor">*. El método implementado busca apariciones de esta cadena dentro de la cabecera y extrae las claves y valores, que añade al diccionario.

Para las claves conocidas se crea una cadena localizable como clave. Si no, se usa la clave original.

Más información sobre la cabecera de archivos HTML puede encontrarse en [10].

A.6. Categoría iWork

A diferencia del formato mov (sección A.8), también perteneciente a Apple, para el que la documentación es extensa, la información sobre los archivos pertenecientes al paquete de ofimática iWork, *Keynotes*, *Numbers* y *Pages*, es extremadamente escasa.

Para obtener algo sobre lo que trabajar se envió un correo a Apple pidiendo documentación, de quién se recibió la respuesta "the information you have requested regarding Pages, Numbers, and Keynotes file format is not a benefit of our developer programs" (la información que ha solicitado en relación a los formatos Pages, Numers y Keynotes no está disponible para nuestros programas *developer*).

Si se abre un fichero de este tipo con un editor hexadecimal, se puede ver que el comienzo coincide con el de un fichero *zip*. Es más, si se cambia la extensión al archivo, se podrá abrir con un descompresor *zip* estándar. Los archivos de iWork, juntos con los de Microsoft Office a partir de la versión del 2003 (sección A.12), utilizan este formato y, por tanto, la clase ZipArchive (sección 4.2.3).

Si se descomprime un archivo de este tipo, según de cuál se trate, aparece un archivo "index.xml" (en *Pages* y *Numer*) o "index.apxl" (en *Keynotes*). El análisis de estos archivos ha indicado que pueden contener metainformación. Además, siempre aparece en las primeras líneas la versión del software utilizado.

El método desarrollado descomprime el fichero en un directorio temporal y analiza el XML (sección A.18) correspondiente para extraer los datos, que se devuelven en un diccionario al terminar.

A.7. Categoría jpg

Los ficheros jpg están constituidos por segmentos, que comienzan con 2 bytes de identificación y otros 2 con el tamaño. Los metadatos pueden encontrase en el segmento APP1, identificado por 0xFFE1, y en el APP13, 0xFFED. El primero puede almacenar datos en el formato "Exif", para el que se utiliza un método de la categoría tiff (sección A.17), o en formato "XMP" (sección A.19); el segundo, por su parte, puede contener información añadida por la herramienta *Photoshop*. Para este segmento concreto se han creado una serie de métodos específicos dentro de la categoría.

Documentación sobre este formato puede encontrase en [11].

A.8. Categoría mov

El formato mov se construye sobre una estructura de átomos y subátomos. Cada uno de ellos comienza con 4 bytes destinados al identificador seguidos de otros 4 bytes para el tamaño, que incluye estos 8 bytes.

El átomo en el que buscar datos utiliza el identificador "meta", y puede aparecer hasta uno dentro de cada átomo "moov", "trak" y "mdia". El método principal itera sobre los átomos del fichero, saltando los que no interesan y llamando a otro método si encuentra alguno de estos tres. Este segundo método se encarga de buscar "meta" entre los subátomos. Si aparece alguno de los tres nombrados al principio, se realiza una llamada recursiva.

Una vez se ha encontrado un átomo "meta", desde un tercer método se extraen primero las claves de los metadatos, que se encuentran en el

subátomo “keys”, y, seguidamente, los valores, almacenados en “ilst”. Los datos pueden ser cadenas, enteros, flotantes, etc. El método los transforma a cadena y los incorpora al diccionario. También podría darse el caso de aparecer una imagen como metadato, pero el soporte para imágenes todavía no está implementado y por el momento sólo se muestra la cadena localizada “Imagen”.

Documentación sobre este formato puede encontrase en [12].

A.9. Categoría mp3

Los ficheros mp3, utilizados para almacenar música, suelen llevar como prefijo una cabecera de tipo ID3v2 con información sobre el mismo. La categoría incluye métodos para soportar este tipo de archivos.

Al comienzo del archivo aparece la versión de cabecera utilizada y el tamaño de la misma. Si la versión no está soportada, el análisis se detiene y se avisa al usuario. El tamaño de la cabecera viene indicado por 4 bytes, utilizándose sólo 7 bites de cada uno, descartando el de más peso. En total, 28 bites efectivos que se unen mediante desplazamientos.

La cabecera está compuesta por “frames”. Cada uno contiene un identificador de 4 bytes seguido de otros 4 indicando su tamaño. Hay varios “frames” que contienen datos que pueden resultar relevantes. Para cada tipo de ellos se ha implementado un método que extrae la información.

Adicionalmente, para el caso concreto del “frame” “TCON”, que contiene información sobre el género musical, se ha creado un archivo con una tabla con todos los géneros . Según la información sobre este formato hay 126 distintos definidos, que pueden aparecer identificados con un valor numérico entre 0 y 125, por lo que la tabla permite una traducción rápida. Por el momento no se han localizado las cadenas de texto para cada género.

Documentación sobre este formato puede encontrarse en [13].

A.10. Categoría mp4

El formato contenedor mp4 es muy similar al formato mov (sección A.8). Esto ha resultado importante porque en este caso no se disponía de documentación gratuita, solamente de la ISO asociada [14]. Haciendo uso de editores hexadecimales para analizar los distintos ficheros se han podido desarrollar los métodos específicos para este formato.

Los ficheros están organizados en átomos, que comienzan con su tamaño e identificador, ambos de 4 bytes. Interesa encontrar el átomo “meta”, contenido por “udta”. A su vez, este átomo puede estar dentro de “trak” o

"moov". También pueden darse casos en los que aparecen átomos "meta" adicionales dentro de "meco" (metadata container).

La búsqueda se realiza analizando los átomos y subátomos de forma recursiva. Una vez en "meta", se analiza una lista de ítems en el subátomo "ilst", donde, para cada átomo, se analiza el contenido, se decide si es de interés y, si lo es, se incorpora al diccionario.

Se ha observado que, en ocasiones, dentro de la lista aparece la clave "Xtra", que es en realidad un subátomo de "ilst" donde se almacenan metadatos de forma similar al formato ASF (sección A.3). Esto es así, presumiblemente, porque los datos fueron añadidos desde un ordenador con windows. No se han encontrado archivos con metadatos en este subátomo con formato distinto, pero podrían existir. La aplicación soporta sólo el aquí explicado.

A.11. Categoría Office binario

Los ficheros de Micrososft Office hasta la versión del 2003 utilizan un formato binario que se asemeja mucho al sistema de archivos FAT: el fichero contiene *streams* a las que se les asigna un determinado espacio dentro del fichero; para saber dónde se encuentra una *stream* concreta hay que acceder a la tabla de asignación y obtener los sectores (o incluso "mini-sectores") en los que se encuentra.

Como ya se comenta en los problemas encontrados (sección 5.1), la documentación oficial de Microsoft acerca de este formato resulta un tanto confusa, mientras que la ofrecida por Open Office es más aclaratoria.

Para poder extraer metainformación de este tipo de archivos, hay que buscar y analizar las *streams* *DocSummaryInfo* y *SummaryInfo*. Esta categoría define métodos que construyen las distintas tablas de asignación en memoria y pasan a analizar el nodo raíz de directorios. A partir de él se puede obtener el identificador de *stream* de las buscadas, para acceder a ellas después gracias a las tablas previamente obtenidas.

Puesto que los datos pueden estar contenidos en diversos sectores, que pueden no ser consecutivos, se hizo indispensable implementar un método de lectura que tuviera en cuenta si al leer del fichero era necesario cambiar de posición el puntero por acabarse un sector y comenzar otro. También se le añadió funcionalidad para poder leer de "mini-sectores". De esta forma, se puede realizar una lectura del fichero de forma transparente.

Para cada una de las *streams* de interés se ha creado un método específico para extraer la información. También existen métodos adicionales de apoyo que calculan *offsets*, fechas, tiempo invertido y que traducen los identificadores de clave a cadenas localizadas.

Documentación oficial de este formato en [15]. La documentación facilitada por Open Office puede encontrase en [16].

A.12. Categoría Office zip

A partir de la versión 2003 de Microsft Office, los archivos de este paquete de ofimática utilizan el formato *zip*, del mismo modo que se hace en los archivos de iWork (sección A.6).

El método implementado descomprime el archivo y busca dentro de la carpeta "docProps" los archivos "core.xml" y "app.xml", que posteriormente envía a un método específico de la categoría XML (sección A.18) para su análisis.

Documentación de Microsoft sobre este formato en [17].

A.13. Categoría ogg

Los archivos ogg utilizan una estructura de páginas para almacenar la información, estando cada una dividida en segmentos. Al comienzo de ella, tras los 4 bytes con "OggS" y un byte a 0, se indica el tipo de la página. Algo después aparece el número de *lacing values*, que indica la cantidad de segmentos que aparecen en la página. Los tamaños de éstos aparecen a continuación, todos juntos, de la siguiente manera: se leen bytes que se van sumando hasta que aparece un valor inferior a 0xFF (255), que también se suma, que indica el final del tamaño de ese segmento; a continuación se lee el tamaño del siguiente segmento. Esto se repite tantas veces como *lacing values*.

Una vez se han calculado los tamaños de los segmentos, se itera sobre ellos, saltando los que no sean comentarios. Cuando se encuentra uno de este tipo, se llama a un método que extrae la información y actualiza el diccionario con los valores obtenidos.

Documentación sobre el formato puede encontrarse en [18].

A.14. Categoría pdf

El formato pdf se estructura en objetos y su análisis comienza por el final del fichero, que debería estar marcado con la cadena "%%EOF". Los ficheros pdf hacen uso, según la documentación, de los saltos de línea para separar las cosas. Sin embargo, como se comentará más adelante, esto no siempre se cumple.

La línea anterior al final del fichero indica el *offset* de la tabla de referencias, que contiene los *offset* de los distintos objetos en el fichero.

Siguiendo la lectura hacia atrás, se encuentra el trailer, que detalla algunos aspectos del archivo: dónde se encuentra la anterior tabla de referencias (que, si aparece, se habrá de tener en cuenta en lugar de la anteriormente leída), el número total de objetos, cuál es el objeto raíz y cuál es el objeto de información.

Con la tabla de referencias construida y los offset calculados, gracias a otro método, es posible acceder a cualquier objeto dentro del fichero. En concreto se accede a los dos últimos objetos mencionados, si están presentes, que son analizados en busca de metainformación por métodos específicos. El objeto raíz no contiene metadatos en sí mismo, pero puede tener una referencia a un objeto "metadata" que encapsula código XML, que es procesado por la categoría asociada (sección A.18).

Pdf es un formato muy utilizado y numerosos programas ofrecen la posibilidad de exportar documentos a este formato. Lamentablemente, como ya se ha comentado en los problemas encontrados (sección 5.1), en ocasiones la especificación no se respeta completamente y ha sido común encontrar ficheros que no incluían saltos de línea o espacios cuando debían estar presentes o que tenían estructuras ligeramente distintas. Por ello, los métodos se han hecho algo menos estrictos, permitiendo que soporten también las variaciones encontradas.

Documentación sobre el formato puede encontrarse en [19].

A.15. Categoría RIFF

En el formato RIFF (wav y avi, por ejemplo, utilizan este formato) se vuelve a utilizar el sistema de "chunks" o átomos visto anteriormente. En este caso se busca el identificado como "LIST", así que se saltan "chunks" hasta que se encuentra uno de este tipo. Una vez en él, se obtiene de que tipo es, ya que únicamente interesa el de tipo "INFO".

Una vez se ha encontrado el punto donde se almacenan los datos, se itera dentro del contenedor, obteniendo la clave del dato, la longitud del valor y el valor en sí mismo, que son añadidos al diccionario.

Documentación sobre el formato puede encontrase en [20].

A.16. Categoría rtf

Los ficheros rtf, a diferencia de la gran mayoría de los formatos explicados hasta ahora, son de tipo texto. Incorporan una cabecera que puede contener

información.

En concreto se busca la estructura {\info[{\CLAVE VALOR}]+}; es decir, dentro de las llaves que delimitan "info" pueden aparecer uno o más elementos, también encerrados entre llaves, donde primero aparece el nombre de la clave, precedido por '\', y a continuación el valor del dato.

Un método adicional se encarga de descomponer los elementos en clave y valor y de decidir si se trata de un dato relevante para que a continuación sea incorporado al diccionario.

Documentación sobre este formato puede encontrarse en [21].

A.17. Categoría tiff

Los archivos tiff comienzan con 2 bytes que indican el *endian* que se está utilizando: 0x4D4D para *big endian* y 0x4949 para *little endian*. Los 2 siguientes bytes tienen el valor 0x002A para terminar de identificar el fichero. A continuación, aparecen 4 bytes indicando el *offset* en el que comienza la cabecera y al que hay que ir para analizar el fichero.

Una vez en la cabecera se lee el número de valores que aparecen y, para cada valor, se itera obteniendo su clave, la longitud del valor y el valor en sí o un offset al mismo. En el caso de la aplicación, todos los metadatos que se buscan almacenan en ese campo el offset. Con él, se mueve el puntero del lector del fichero y se leen tantos bytes como indica el tamaño. La clave es localizada y los datos son seguidamente incorporados al diccionario

Documentación sobre este formato puede encontrarse en [22].

A.18. Categoría XML

La categoría de XML añade soporte para la extracción de metadatos de archivos XML. La forma de escribir un archivo de este tipo varía según la herramienta, por lo que se han implementado 3 métodos específicos: uno para archivos de iWork, otro para ficheros de Office y un tercero para el XML de documentos pdf.

El XML de los últimos es en realidad XMP (sección A.19) y se llama al método de la categoría correspondiente para la extracción de información.

En cuanto a los otros dos, el análisis en muy similar y consiste en analizar tag a tag (entendiendo por tag el texto comprendido entre '<' y '>') buscando alguno que contenga metadatos. Puesto que hace falta analizar todo el fichero, estos métodos suelen requerir más tiempo de ejecución. Estos dos tipos de documentos también tienen un método adicional, cada uno, para la localización de las claves de los metadatos.

A.19. Categoría XMP

El aspecto de los archivos XMP es muy similar al de los XML, pero en esta categoría se ha implementado un método concreto para lidiar con el análisis de este formato.

En este caso, la categoría, además de realizar un análisis similar del XML, permite obtener metadatos contenidos en tags que se autocierran.

Puesto que no se trabaja con ficheros XMP, sino con cadenas de XMP incrustadas en otros ficheros, los métodos desarrollados trabajan con cadenas de texto.

A.20. Categoría *zip*

Los archivos *zip* se utilizan para comprimir carpetas u otros archivos. El contenido de los mismos, por tanto, depende de lo que se haya comprimido antes y puede no contener nada interesante. No obstante, puesto que la clase MetaAnalyzer comprueba extensiones y firmas de ficheros, puede ocurrir que se intente analizar en archivo con firma *zip* cuya extensión no coincide con un fichero de de los paquetes de ofimática iWork u Office.

En estos casos, se llama a esta categoría, que descomprime los contenidos en un directorio temporal y busca archivos que podrían ser propios de iWork u Office. Si se encuentra alguno, se llama al método de la categoría correspondiente. En caso contrario, no se hace nada.

Apéndice B

Manual de usuario

Una vez que la aplicación ha sido descargada e instalada, su aspecto en el dispositivo es el que aparece en la figura B.1. Presionar el icono la lanzará.

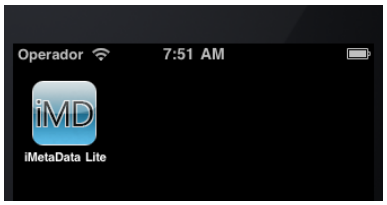

Figura B.1: Icono de la aplicación en el dispositivo

Al iniciar la aplicación se presenta al usuario, como puede verse en la figura B.2, un menú con dos botones, que le permiten comenzar a explorar el sistema de archivos o mostrar información sobre la herramienta.

La segunda opción (fig. B.3) contiene una descripción de la aplicación junto a un hipervínculo en forma de imagen a la página web de Informática64, con quien se ha desarrollado esta aplicación. Para volver al menú principal se ha de presionar el botón "Atrás".

Si desde el menú se presiona el primer botón, se muestra una nueva vista desde la que explorar el sistema de ficheros (fig. B.4). La barra superior contiene 2 botones, usados, de izquierda a derecha, para subir un directorio o para salir al menú principal. En el centro de ésta aparece una etiqueta de texto indicando el directorio en el que nos encontramos actualmente.

Inmediatamente debajo de la barra aparece una tabla con los contenidos del directorio actual. Los subdirectorios están marcados con un pequeño accesorio en la parte derecha y si se presiona uno, ya sea la celda que contiene

Figura B.2: Menú al iniciar la aplicación

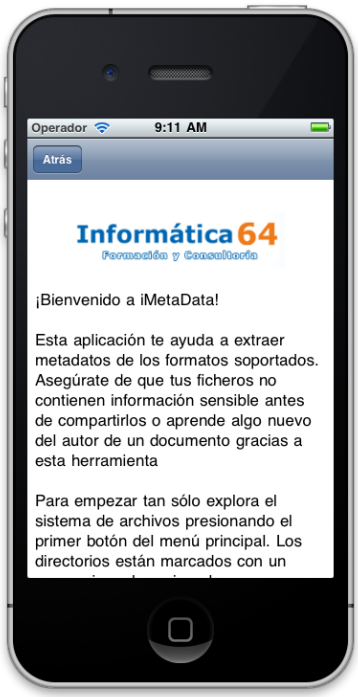

Figura B.3: Acerca de iMetadata

Figura B.4: Explorando el sistema de ficheros

el nombre del directorio o su accesorio, se actualiza la tabla con los contenidos del nuevo directorio así como la etiqueta superior, para que contenga el nombre del nuevo directorio.

Si se produce un error al cambiar de directorio, se mantiene el actual y se notifica al usuario con una alerta (fig. B.5).

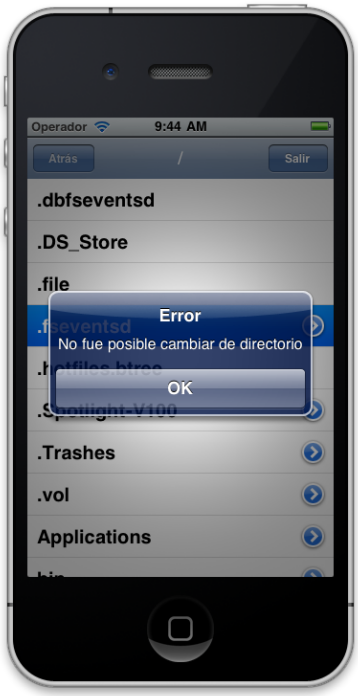

Figura B.5: Se alerta al usuario si no se puede cambiar de directorio

Mientras se muestra el directorio raíz no es posible subir un directorio, por lo que el botón "Atrás" aparece desactivado (fig. B.6).

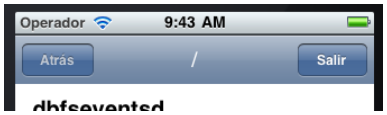

Figura B.6: No hay directorios por encima de la raíz

Al presionar un archivo, la aplicación comprueba de qué tipo es y comienza su exploración en busca de metadatos. Si el tipo de fichero no estuviera soportado o el formato del mismo fuera incorrecto, se mostraría una alerta con el aviso (figuras B.7 y B.8, respectivamente). La aplicación también mostraría una alerta si surgiesen problemas al abrir o descomprimir un archivo, si la versión del formato no estuviera soportada (fig B.9) o en el caso de que ocurriese un error inesperado.

Figura B.7: Alerta si el tipo de fichero no está soportado

Si el análisis del fichero resulta exitoso y el fichero contenía metainformación, la aplicación mostrará una nueva vista con los resultados (fig. B.10). En cada sección el título indica el tipo del metadato y a continuación aparecen tantas filas como metadatos de ese tipo estuvieran almacenados en el archivo. En la figura B.11 aparece un ejemplo en el que un archivo contiene varias entradas para un mismo tipo.

Podría ocurrir, no obstante, que pese a que el análisis de un fichero haya sido posible, éste no contuviera metadatos de interés. En ese caso la aplicación mostrará una alerta avisando de ello al usuario (fig. B.12).

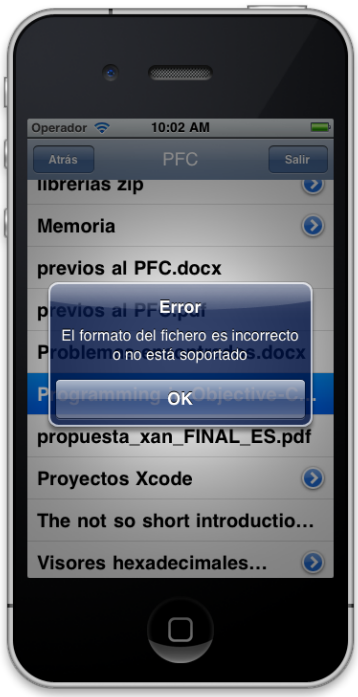

Figura B.8: Alerta si el formato del fichero no está soportado

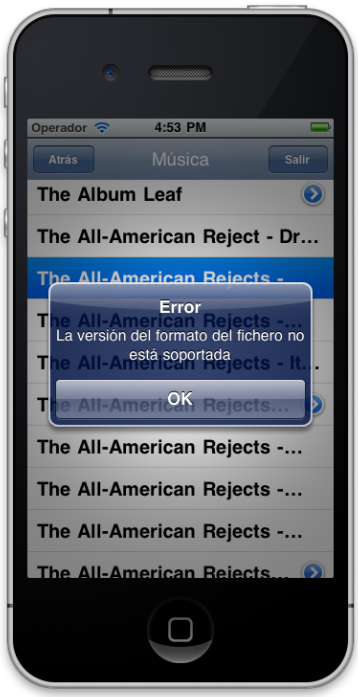

Figura B.9: Alerta si la versión del formato no está soportada

Figura B.10: El fichero contenía metainformación

Figura B.11: Ejemplo de fichero con varias entradas para el mismo tipo de metadato

Figura B.12: El fichero no contenía metadatos

La aplicación ajusta su idioma automáticamente al seleccionado en el dispositivo[1], siempre y cuando esté disponible la traducción. Si se selecciona inglés o un idioma aún no soportado, la aplicación se mostrará en inglés. En la siguiente figura se puede ver el aspecto de la aplicación en este caso.

Figura B.13: La aplicación ajusta su idioma al seleccionado en el dispositivo

Actualmente la aplicación se encuentra disponible en alemán, español, inglés y ruso. Además, gracias a la internacionalización realizada en ella, es posible traducirla de forma sencilla a otros lenguajes: tan sólo hay que añadir un fichero *strings* con las cadenas en el idioma correspondiente, siguiendo el mismo esquema de los ya disponibles.

[1]La configuración de idioma del dispositivo se realiza en Ajustes / General / Internacional / Idioma

Apéndice C

Pruebas

Durante el desarrollo del proyecto ha habido dos fases de pruebas diferentes: la primera ha abarcado el desarrollo de la clase principal MetaAnalyzer, probándose archivos del tipo apropiado cada vez que se realizaba una nueva categoría, para comprobar su correcto funcionamiento.

Después, una vez que la interfaz gráfica fue diseñada, llegó la segunda fase de pruebas, en la que se generaron nuevos archivos, para comprobar que la información se obtenía correctamente, y se obtuvieron ficheros de distintos lugares para comprobar que no se producían errores durante la ejecución.

Por ejemplo, con *Photoshop* se crea un archivo en el que se introduce metainformación como se ve en la figura C.1.

Seguidamente se comprueba con un editor hexadecimal que los datos están allí (fig. C.2. En este caso, los datos están en el segmento APP13 (ver A.7) del fichero jpg, que efectivamente ha de contener estos datos según la especificación [11].

Cuando se lanza la aplicación y se analiza el fichero en cuestión, los metadatos son recuperados correctamente (fig. C.3).

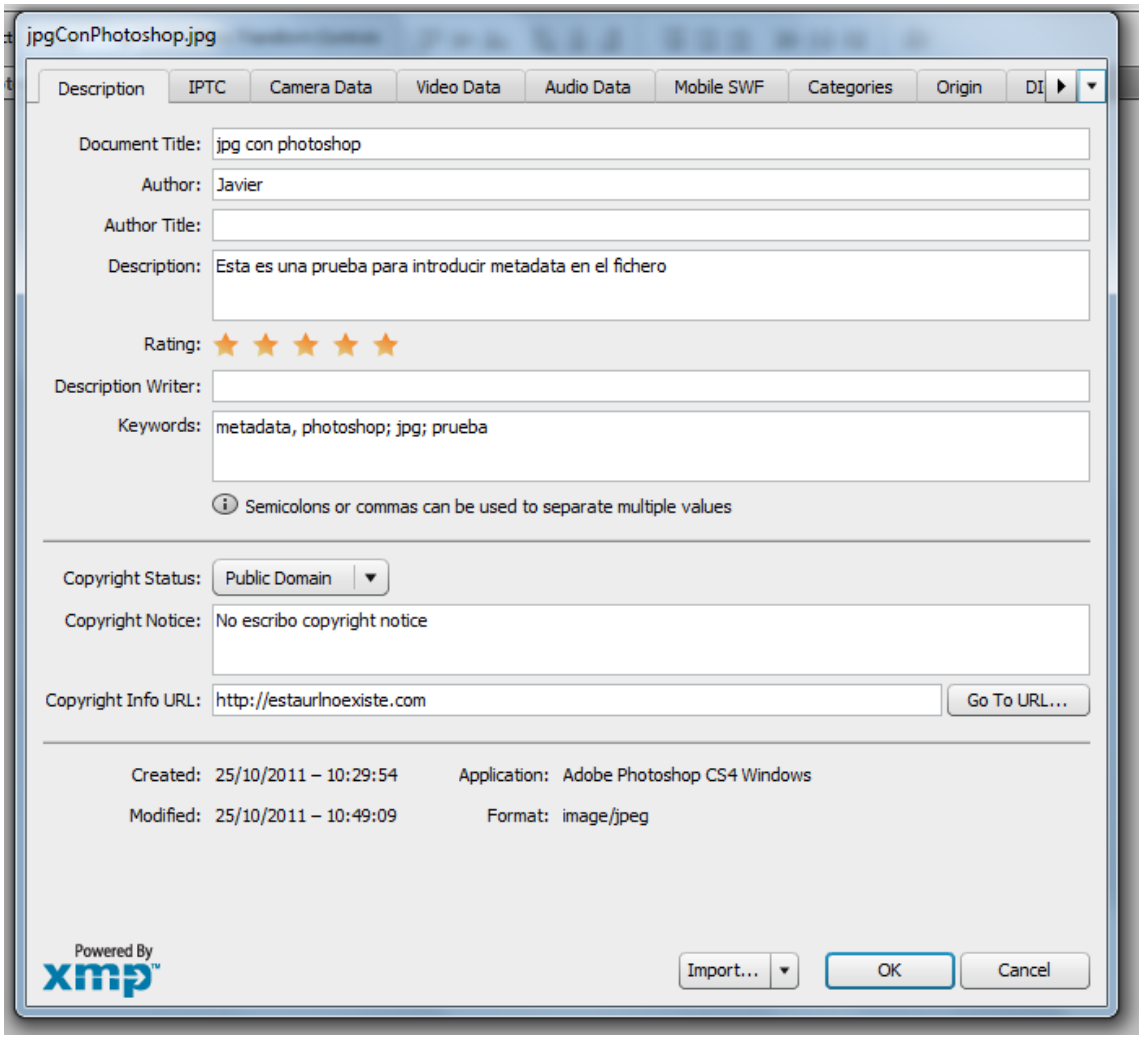

Figura C.1: Creando un fichero con metadatos

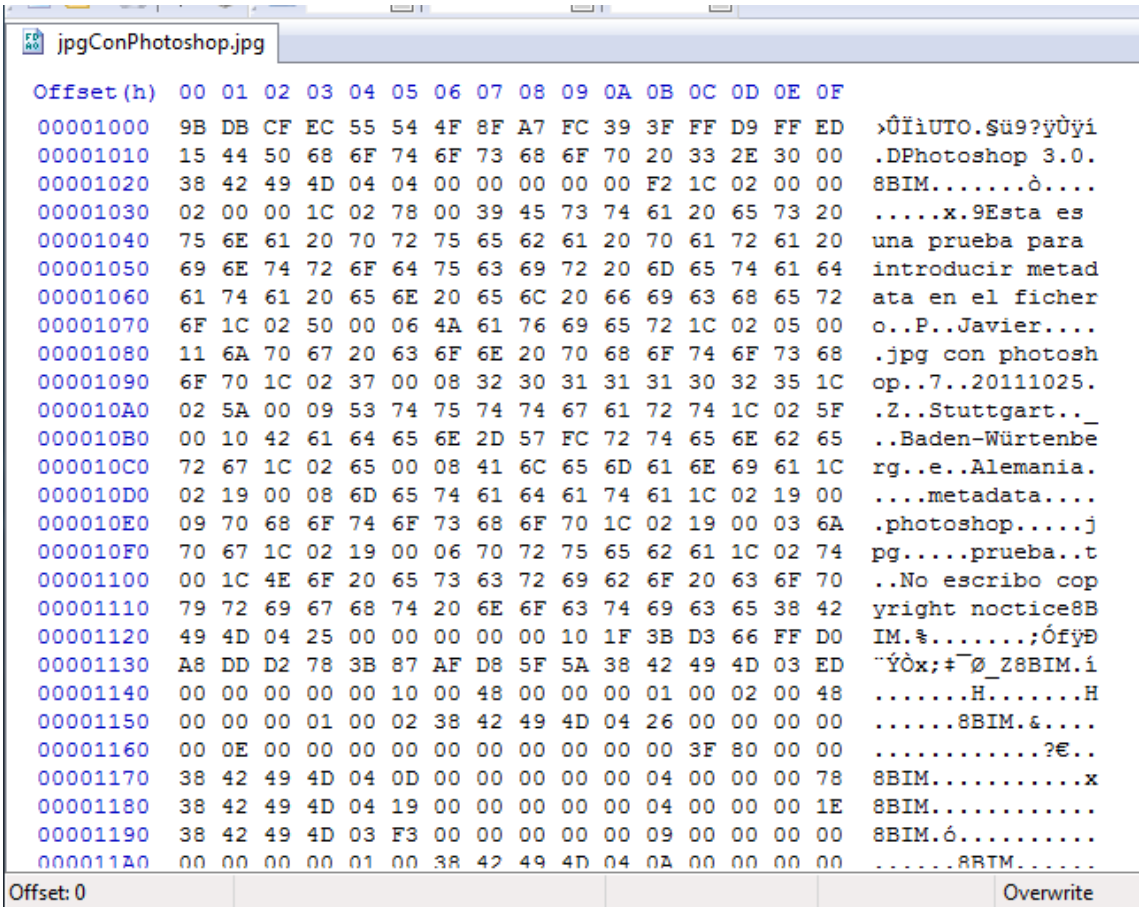

Figura C.2: Con el editor hexadecimal se comprueba que los datos están donde deben

Figura C.3: Los metadatos introducidos son extraídos correctamente

En otra prueba, se crea un archivo mov desde un ordenador de Apple con las características mostradas en la figura C.4.

Figura C.4: Datos que se van a almacenar al crear un archivo mov

De nuevo, se comprueba con el editor hexadecimal que los datos están presentes y aparecen en el átomo “meta” (fig. C.5).

Finalmente, se utiliza la aplicación para comprobar que los resultados son los esperados (fig. C.6).

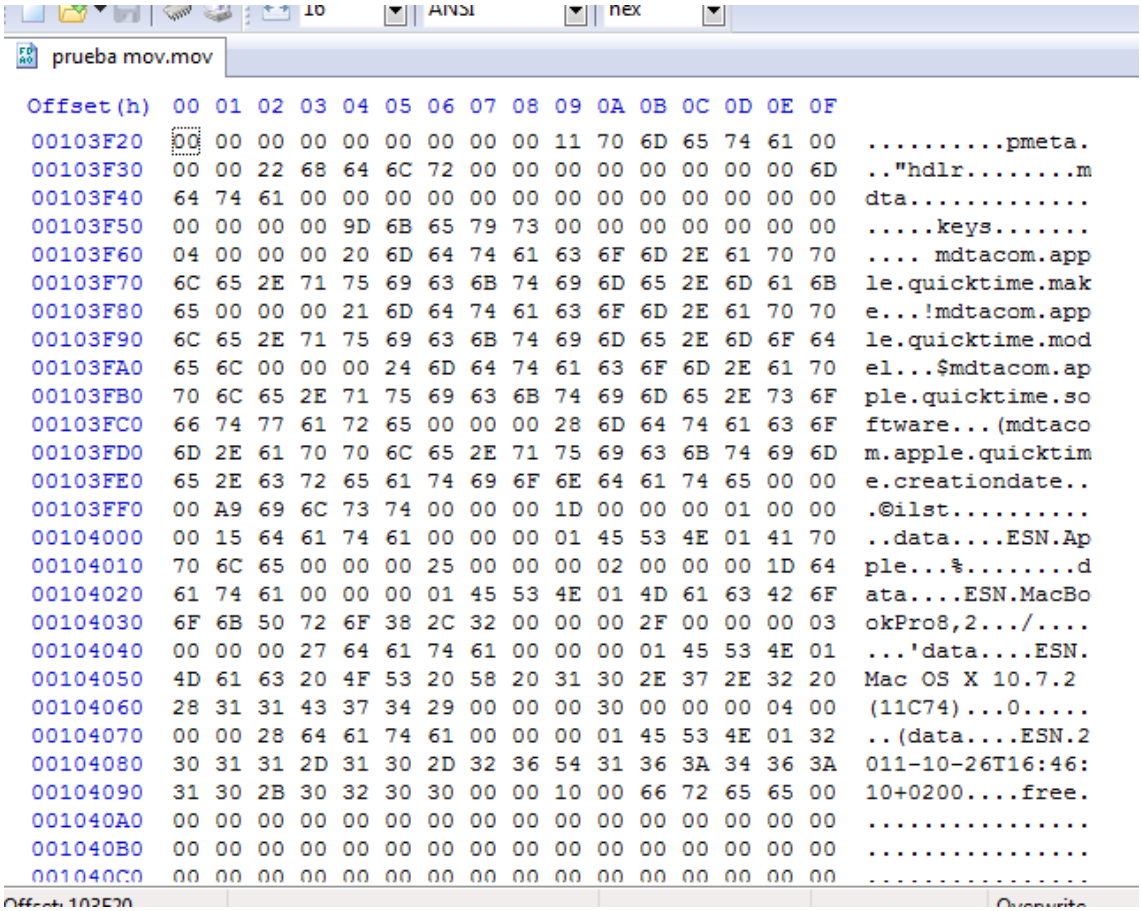

Figura C.5: Se comprueba con el editor hexadecimal que los datos están presentes en el archivo

Figura C.6: Se extraen los datos con la aplicación

Para incorporar variedad a las pruebas, se han buscado y descargado archivos de los diferentes formatos que posteriormente se han analizado con la aplicación. En la figura C.7 pueden verse los resultados de buscar documentos docx (de Micrososft Office con formato *zip*).

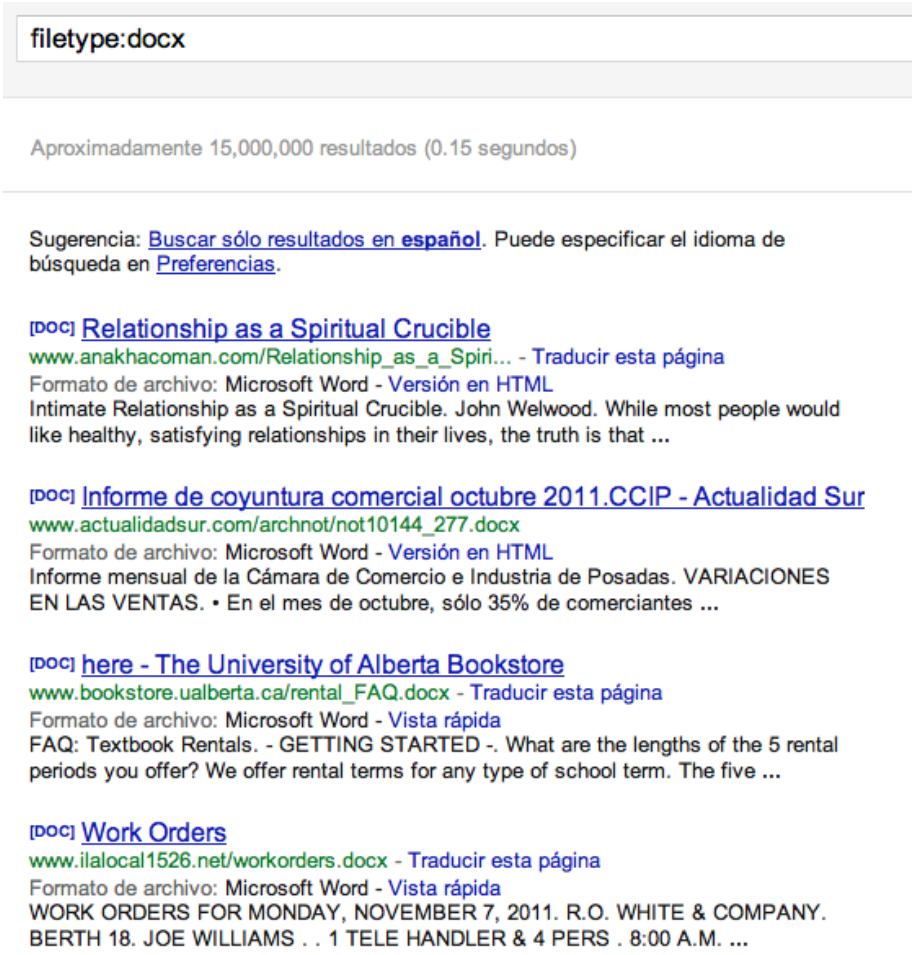

Figura C.7: Se han utilizado motores de búsqueda para encontrar archivos de prueba

Algunos de los documentos encontrado utilizan en sus metadatos alfabetos distintos del latino. En la figura C.8 pueden verse los contenidos de un archivo core.xml dentro de un docx encontrado en internet con esta característica.

En este caso, la aplicación soporta la codificación y extrae y muestra los datos de forma adecuada, como puede verse en la figura C.9.

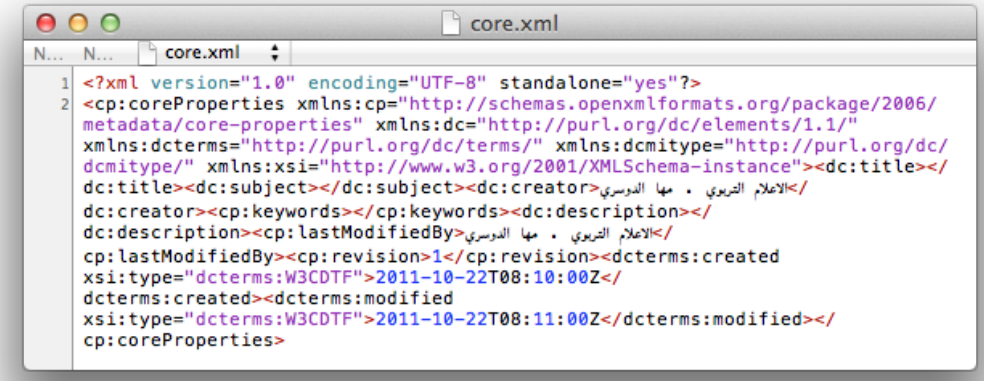

Figura C.8: Pueden aparecer metadatos con otros alfabetos

Figura C.9: La aplicación puede mostrar texto en otros alfabetos

Bibliografía

[1] Stephen G. Kochan: *Programming in Objective C 2.0*, Addison-Wesley Professional, ISBN-10: 0321566157.

[2] Documentación office de Apple sobre la internacionalización: `http://developer.apple.com/library/ios/#documentation/MacOSX/Conceptual/BPInternational/BPInternational.html`.

[3] Página del proyecto ZipArchive: `http://code.google.com/p/ziparchive/`.

[4] Página del proyecto LineReader: `https://github.com/johnjohndoe/LineReader`.

[5] Documentación del formato 3gp: `http://www.3gpp.org/ftp/Specs/archive/26_series/26.244/26244-930.zip`, archivo *zip*.

[6] Documentación del formato AIFF: `http://bellatrix.ece.mcgill.ca/Documents/AudioFormats/AIFF/AIFF.html`.

[7] Documentación del formato ASF: `http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=14995`.

[8] Documentación del formato gif: `http://www.martinreddy.net/gfx/2d/GIF89a.txt`.

[9] Algoritmo en python de Sean M. Burke: `http://interglacial.com/~sburke/pub/list_gif_comments.pl`.

[10] Documentación del formato HTML: `http://www.comptechdoc.org/independent/web/html/guide/htmlhead.html`.

[11] Documentación del formato jpg: `http://www.metadataworkinggroup.org/pdf/mwg_guidance.pdf`.

[12] Documentación del formato mov: `http://developer.apple.com/library/mac/#documentation/QuickTime/QTFF/QTFFPreface/qtffPreface.html`.

[13] Documentación del formato mp3: http://www.id3.org/id3v2.3.0.

[14] ISO del formato mp4: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=38538.

[15] Documentación oficial del formato binario de Office: http://msdn.microsoft.com/en-us/library/dd942265(v=prot.10).aspx.

[16] Documentación de Open Office sobre el formato binario de Office: http://sc.openoffice.org/compdocfileformat.pdf.

[17] Documentación del formato zip de Office: http://msdn.microsoft.com/en-us/library/aa338205(v=office.12).aspx.

[18] Documentación del formato ogg: http://www.xiph.org/ogg/doc/.

[19] Documentación del formato pdf: http://partners.adobe.com/public/developer/en/pdf/PDFReference.pdf.

[20] Documentación del formato RIFF: http://www-mmsp.ece.mcgill.ca/documents/audioformats/wave/Docs/riffmci.pdf.

[21] Documentación del formato rtf: http://www.biblioscape.com/rtf15_spec.htm#Heading25.

[22] Documentación del formato tiff: http://partners.adobe.com/public/developer/en/tiff/TIFF6.pdf.

Índice de figuras

4.1. Menú principal de la aplicación 17
4.2. Explorando el sistema de ficheros 18
4.3. Los datos se muestran en una nueva pantalla 19

5.1. Esquema del reparto de horas 24

B.1. Icono de la aplicación en el dispositivo 37
B.2. Menú al iniciar la aplicación 38
B.3. Acerca de iMetadata 39
B.4. Explorando el sistema de ficheros 40
B.5. Se alerta al usuario si no se puede cambiar de directorio . . . 41
B.6. No hay directorios por encima de la raíz 41
B.7. Alerta si el tipo de fichero no está soportado 42
B.8. Alerta si el formato del fichero no está soportado 43
B.9. Alerta si la versión del formato no está soportada 44
B.10. El fichero contenía metainformación 45
B.11. Ejemplo de fichero con varias entradas para el mismo tipo de metadato 46
B.12. El fichero no contenía metadatos 47
B.13. La aplicación ajusta su idioma al seleccionado en el dispositivo 48

C.1. Creando un fichero con metadatos 50
C.2. Con el editor hexadecimal se comprueba que los datos están donde deben 51
C.3. Los metadatos introducidos son extraídos correctamente . . . 52
C.4. Datos que se van a almacenar al crear un archivo mov 53

C.5. Se comprueba con el editor hexadecimal que los datos están presentes en el archivo 54
C.6. Se extraen los datos con la aplicación 55
C.7. Se han utilizado motores de búsqueda para encontrar archivos de prueba 56
C.8. Pueden aparecer metadatos con otros alfabetos 57
C.9. La aplicación puede mostrar texto en otros alfabetos 58

Figura 4.1: Menú principal de la aplicación

Si el fichero no contuviera metadatos o se produjera algún error, se mostraría un aviso por pantalla con detalles de lo ocurrido

En el apéndice B, que contiene el manual de usuario, puede encontrase un recorrido en profundidad y con más imágenes de la interfaz gráfica así como del funcionamiento de la aplicación en general.

Figura 4.2: Explorando el sistema de ficheros

Figura 4.3: Los datos se muestran en una nueva pantalla

Figura B.2: Menú al iniciar la aplicación

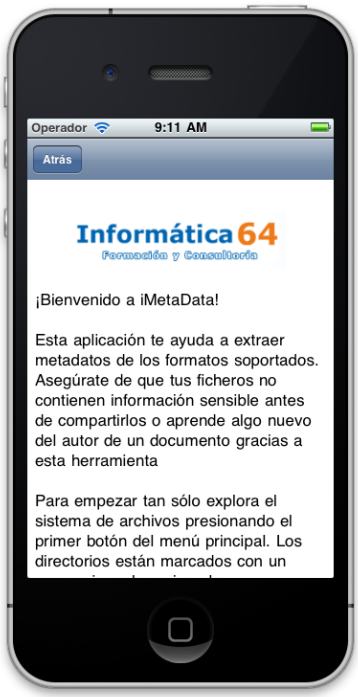

Figura B.3: Acerca de iMetadata

Figura B.4: Explorando el sistema de ficheros

el nombre del directorio o su accesorio, se actualiza la tabla con los contenidos del nuevo directorio así como la etiqueta superior, para que contenga el nombre del nuevo directorio.

Si se produce un error al cambiar de directorio, se mantiene el actual y se notifica al usuario con una alerta (fig. B.5).

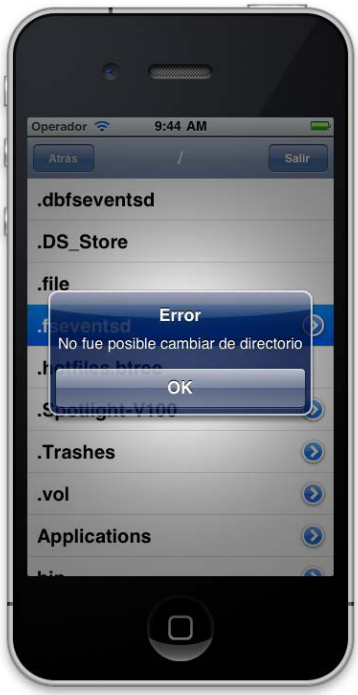

Figura B.5: Se alerta al usuario si no se puede cambiar de directorio

Mientras se muestra el directorio raíz no es posible subir un directorio, por lo que el botón "Atrás" aparece desactivado (fig. B.6).

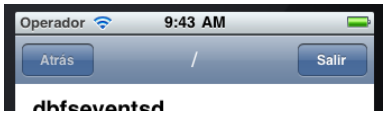

Figura B.6: No hay directorios por encima de la raíz

Al presionar un archivo, la aplicación comprueba de qué tipo es y comienza su exploración en busca de metadatos. Si el tipo de fichero no estuviera soportado o el formato del mismo fuera incorrecto, se mostraría una alerta con el aviso (figuras B.7 y B.8, respectivamente). La aplicación también mostraría una alerta si surgiesen problemas al abrir o descomprimir un archivo, si la versión del formato no estuviera soportada (fig B.9) o en el caso de que ocurriese un error inesperado.

Figura B.7: Alerta si el tipo de fichero no está soportado

Si el análisis del fichero resulta exitoso y el fichero contenía metainformación, la aplicación mostrará una nueva vista con los resultados (fig. B.10). En cada sección el título indica el tipo del metadato y a continuación aparecen tantas filas como metadatos de ese tipo estuvieran almacenados en el archivo. En la figura B.11 aparece un ejemplo en el que un archivo contiene varias entradas para un mismo tipo.

Podría ocurrir, no obstante, que pese a que el análisis de un fichero haya sido posible, éste no contuviera metadatos de interés. En ese caso la aplicación mostrará una alerta avisando de ello al usuario (fig. B.12).

Figura B.8: Alerta si el formato del fichero no está soportado

Figura B.9: Alerta si la versión del formato no está soportada

Figura B.10: El fichero contenía metainformación

Figura B.11: Ejemplo de fichero con varias entradas para el mismo tipo de metadato

Figura B.12: El fichero no contenía metadatos

La aplicación ajusta su idioma automáticamente al seleccionado en el dispositivo[1], siempre y cuando esté disponible la traducción. Si se selecciona inglés o un idioma aún no soportado, la aplicación se mostrará en inglés. En la siguiente figura se puede ver el aspecto de la aplicación en este caso.

Figura B.13: La aplicación ajusta su idioma al seleccionado en el dispositivo

Actualmente la aplicación se encuentra disponible en alemán, español, inglés y ruso. Además, gracias a la internacionalización realizada en ella, es posible traducirla de forma sencilla a otros lenguajes: tan sólo hay que añadir un fichero *strings* con las cadenas en el idioma correspondiente, siguiendo el mismo esquema de los ya disponibles.

[1]La configuración de idioma del dispositivo se realiza en Ajustes / General / Internacional / Idioma

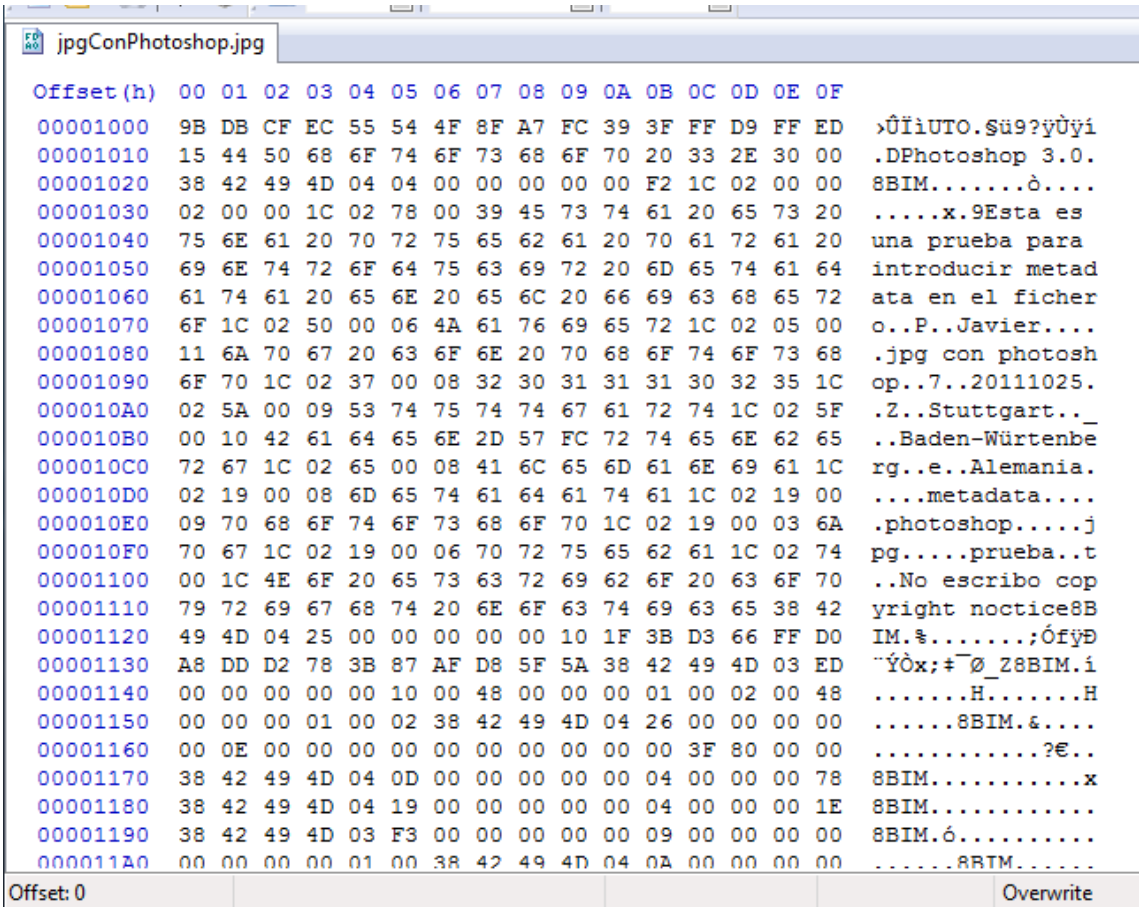

Figura C.2: Con el editor hexadecimal se comprueba que los datos están donde deben

Figura C.3: Los metadatos introducidos son extraídos correctamente

En otra prueba, se crea un archivo mov desde un ordenador de Apple con las características mostradas en la figura C.4.

Figura C.4: Datos que se van a almacenar al crear un archivo mov

De nuevo, se comprueba con el editor hexadecimal que los datos están presentes y aparecen en el átomo “meta” (fig. C.5).

Finalmente, se utiliza la aplicación para comprobar que los resultados son los esperados (fig. C.6).

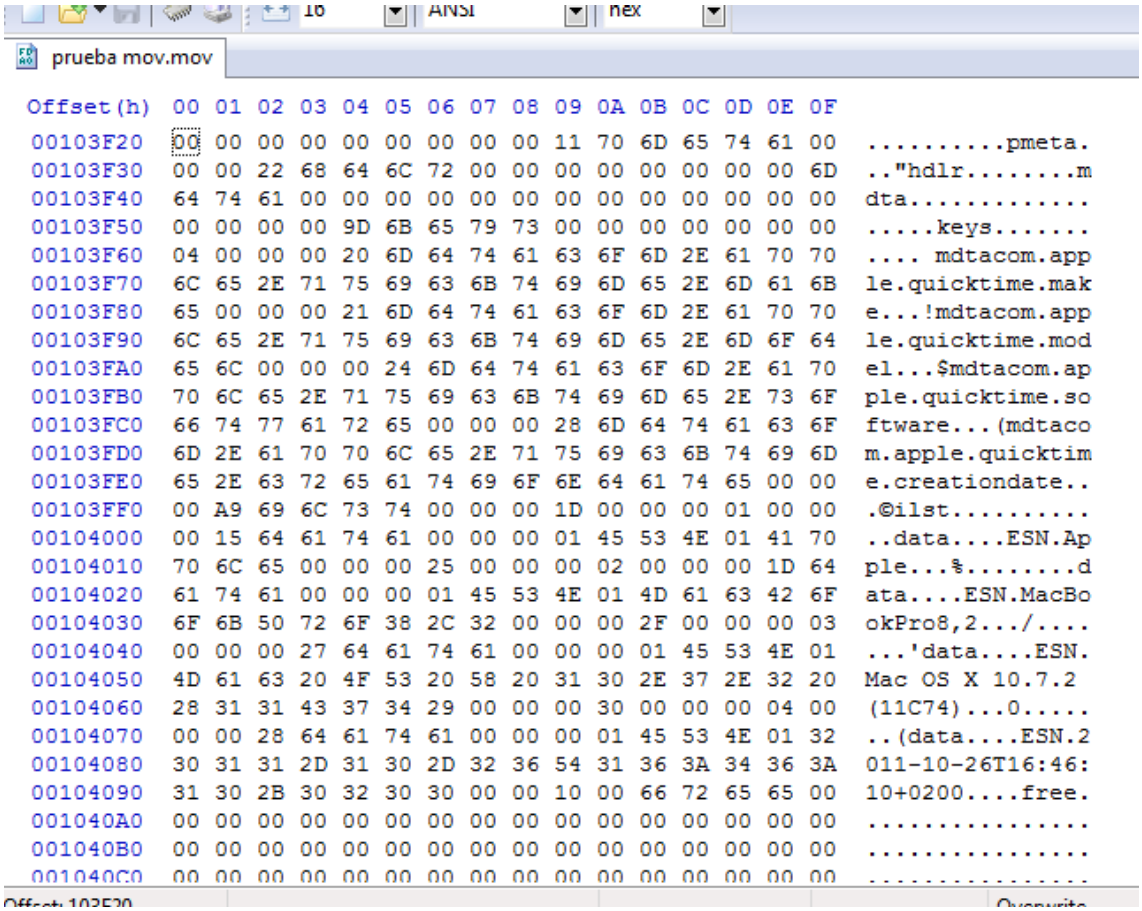

Figura C.5: Se comprueba con el editor hexadecimal que los datos están presentes en el archivo

Figura C.6: Se extraen los datos con la aplicación

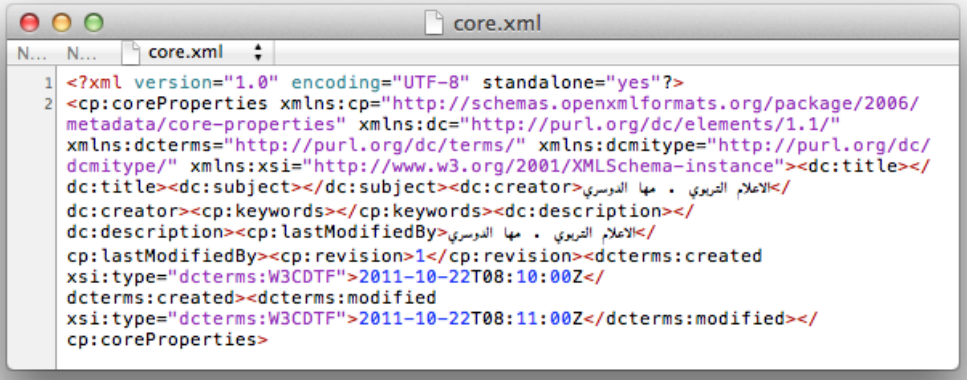

Figura C.8: Pueden aparecer metadatos con otros alfabetos

Figura C.9: La aplicación puede mostrar texto en otros alfabetos