

Protección del copyright de imágenes digitales.

Pedro Reverte Morales
ETIS

Antoni Martínez Ballesté

18/06/04

PROTECCIÓN DEL COPYRIGHT DE IMÁGENES DIGITALES

Pedro Reverte Morales

Estudiante Ingeniería Técnica en Informática de Sistemas

Universitat Oberta de Catalunya

Jun-2004

Resumen

La protección de los derechos de autor para materiales digitales es un importante problema todavía abierto en el que el uso de las marcas de agua digitales se perfila como una solución prometedora. El presente trabajo se estructura en dos partes. En la primera parte se ha realizado un estudio del estado del arte del uso de marcas de agua digitales para aplicaciones de protección del copyright de materiales multimedia, incidiendo principalmente en las imágenes digitales. ¿Qué son las marcas de agua digitales?, ¿cómo funcionan?, ¿para que aplicaciones pueden ser usadas?, ¿qué tipos de marcas existen y cuales son sus propiedades?, ¿qué ataques pueden sufrir? son algunas de las cuestiones que encuentran respuesta en este estudio. En la segunda parte se ha desarrollado una aplicación práctica de las marcas de agua digitales para protección de los derechos de autor por detección de copia conocida como fingerprinting, que posibilita identificar el origen de posibles copias ilegales de imágenes digitales. Para ello se ha diseñado una técnica de marcado sobre el dominio DCT (Discrete Cosine Transform) basada en una propuesta realizada por E. Koch & J. Zhao y se ha estudiado su robustez frente a diversas manipulaciones de la imagen. Utilizando esta técnica se ha diseñado una aplicación sencilla que posibilita la inserción y extracción de marcas de agua en las imágenes para diferentes clientes, gestionando una pequeña base de datos sobre las imágenes marcadas y usando un código corrector de errores dual de un Hamming(7,4) para implementar las marcas. Finalmente se estudia la robustez de las imágenes marcadas frente al ataque más típico a las aplicaciones de fingerprinting, conocido como colisión. Para ello se ha diseñado una aplicación que permite generar una imagen confabulada por colisión de la misma imagen marcada con dos marcas diferentes.

Índice de contenidos

Índice de figuras.....	v
Índice de tablas.....	vi
Introducción.....	1
Justificación contexto y objetivos del TFC.....	1
Método utilizado.....	2
Planificación del TFC.....	2
Productos Obtenidos.....	4
Capítulos siguientes.....	4
1. Estado del arte.....	5
1.1. El problema del copyright.....	5
1.2. Introducción a las marcas de agua digitales.....	6
1.2.1. Que son las marcas de agua digitales.....	6
1.2.2. Origen de las marcas de agua digitales.....	6
1.3. Esquema general del mercado digital.....	7
1.3.1. Generación de la marca de agua.....	7
1.3.2. Inserción de la marca.....	8
1.3.3. Detección o extracción de la marca.....	8
1.4. Clasificación y propiedades de las marcas de agua.....	9
1.4.1. Visibles o invisibles.....	9
1.4.2. Frágiles o robustas.....	10
1.4.3. Privadas o públicas.....	11
1.4.4. Técnicas en el dominio espacial o espectral.....	11
1.4.5. Clasificación general de las técnicas de marcado....	11
1.5. Aplicaciones de las marcas de agua.....	12
1.5.1. Marcas visibles.....	12
1.5.2. Marcas invisibles robustas.....	13
1.5.3. Marcas invisibles frágiles.....	13
1.6. Técnicas de marcado robusto de imágenes digitales.....	13
1.7. Problemática de las imágenes marcadas.....	14
1.8. Una aplicación concreta: fingerprinting de imágenes.....	16
1.8.1. Códigos anti-colisión.....	17
1.9. Conclusiones y futuro del mercado digital.....	18
Referencias.....	18
2. Memoria de diseño.....	20
2.1. Diseño de la técnica de marcado.....	20
2.1.1. Algoritmo de inserción de marcas.....	20
2.1.2. Algoritmo de recuperación de la marca.....	24
2.1.3. Diseño estático.....	26
2.1.4. Elección de los parámetros máximo y mínimo.....	29
2.1.5. Pruebas de robustez.....	31
2.1.6. Posibles mejoras.....	34
Referencias.....	34
2.2. Diseño de la aplicación de inserción de marcas: ApplMDCT...	34
2.2.1. Diagrama de casos de uso.....	35
2.2.2. Diagramas de colaboración.....	38
2.2.3. Diseño de la persistencia.....	40
2.2.4. Diseño de la interficie de usuario.....	40
2.2.5. Diagrama estático.....	41
2.2.6. Código corrector de errores.....	42
2.2.7. Implementación.....	44
2.2.8. Posibles mejoras.....	45
Referencias.....	45
2.3. Diseño de la aplicación de confabulaciones: ApplConfabula..	45
2.3.1. Diagrama de casos de uso.....	45
2.3.2. Diagramas de colaboración.....	47
2.3.3. Diseño de la interficie de usuario.....	48
2.3.4. Diagrama estático.....	49
2.3.5. Implementación.....	49

2.3.6.	Pruebas con imágenes confabuladas.....	49
2.3.7.	Mejoras de los resultados.....	51
3.	Conclusiones.....	52
	Glosario de términos y acrónimos.....	55
	Bibliografía.....	55
	Anexos.....	56
A.	Manuales de usuario.....	56
A.1.	Aplicación ApplMDCT.....	56
A.2.	Aplicación ApplConfabula.....	59
B.	Código fuente ApplMDCT.....	61
B.1.	Clase AbaoutBox.java.....	61
B.2.	Clase AlgoritmoMDCT.java.....	63
B.3.	Clase ApplMDCT.java.....	73
B.4.	Clase BMPFile.java.....	82
B.5.	Clase DCT.java.....	87
B.6.	Clase Dialogos.java.....	99
B.7.	Clase GestorDatos.java.....	101
B.8.	Clase GestorImagenesM.java.....	108
B.9.	Clase HammingDual.java.....	112
B.10.	Clase ImageFilter.java.....	115
B.11.	Clase ImageIO.java.....	116
B.12.	Clase ImagenM.java.....	121
B.13.	Clase iObserver.java.....	123
B.14.	Clase PanelDatos.java.....	124
B.15.	Clase PanelExtracción.java.....	125
B.16.	Clase PanelInserción.java.....	128
B.17.	Clase utils.java.....	131
C.	Código fuente ApplConfabula.....	141
C.1.	Clase AboutBox.java.....	141
C.2.	Clase ApplConfabula.java.....	143
C.3.	Clase BMPFile.java.....	148
C.4.	Clase Dialogos.java.....	148
C.5.	Clase GestorConfa.java.....	148
C.6.	Clase ImageFilter.java.....	154
C.7.	Clase ImageIO.java.....	154
C.8.	Clase iObserver.java.....	154
C.9.	Clase PanelConfabulación.java.....	154
C.10.	Clase utils.java.....	156

Índice de figuras

1. Planificación gráfico de Gantt.....	3
2. Proceso de inserción de marca.....	8
3. Proceso de verificación.....	8
4. Diseño algoritmo de inserción.....	21
5. Descomposición en suma de cosenos.....	22
6. Funciones base DCT 2-dimensional.....	22
7. Coeficientes adecuados para marcado.....	23
8. Tabla de cuantización luminancias Jpeg.....	23
9. Diseño algoritmo de recuperación de marcas.....	24
10. Diagrama estático técnica de marcado.....	26
11. Diagrama de flujo método inserción de marcas.....	27
12. Diagrama de flujo método extracción de marcas.....	28
13. Influencia parámetro mínimo en la imagen A.....	30
14. Influencia parámetro máximo en la imagen A.....	30
15. Influencia parámetro mínimo en la imagen B.....	31
16. Influencia parámetro máximo en la imagen B.....	31
17. Img1 lena.bmp 512x512 24-bits color.....	31
18. Img2 corredor.jpg 283x212 24-bits color.....	31
19. Img3 logo_uoc.bmp 150x38 24-bits color	32
20. Img4 steve.jpg 320x240 8-bits escala de grises.....	32
21. Esquema aplicación ApplMDCT.....	34
22. Diagrama de casos de uso ApplMDCT.....	35
23. Diagrama de colaboración. Marcar imagen.....	38
24. Diagrama de colaboración. Insertar marca.....	39
25. Diagrama de colaboración. Grabar marcada.....	39
26. Diagrama de colaboración. Consultar datos.....	40
27. Interficie de usuario. Panel de inserción.....	41
28. Interficie de usuario. Panel de extracción.....	41
29. Interficie de usuario. Panel de consultas.....	41
30. Diagrama estático ApplMDCT.....	42
31. Esquema general ApplConfabula.....	45
32. Diagrama de casos de uso ApplConfabula.....	46
33. Diagrama de colaboración. Generar confabulada.....	47
34. Diagrama de colaboración. Grabar confabulada.....	48
35. Interficie de usuario. Panel ApplConfabula.....	48
36. Diagrama estático ApplConfabula.....	49
37. ApplMDCT: Marcar Imagen.....	57
38. ApplMDCT: Extraer Marca	58
39. ApplMDCT: Consultar Datos.....	59
40. ApplConfabula: Generar confabulada.....	60

Índice de tablas

1. Descomposición de tareas.....	3
2. Estudio parámetros para la imagen A.....	29
3. Estudio parámetros para la imagen B.....	30
4. Pruebas de robustez.....	32
5. Resultados pruebas robustez.....	33
6. Código dual Hamming(7,4).....	43
7. Resultados pruebas confabulación.....	50
8. Resultados método combinado.....	51

INTRODUCCIÓN

Justificación, contexto y objetivos del TFC.

Los productos multimedia ofrecen grandes ventajas respecto a los productos tradicionales en cuanto a su duplicación y distribución lo que ha propiciado su enorme desarrollo en los últimos años, sin embargo estas mismas facilidades contribuyen a su mayor punto débil: la dificultad de la protección de los derechos de autor. Los métodos preventivos que intentan evitar que se realice una copia ilegal de un material multimedia no han resultado efectivos hasta el momento, en este contexto se han desarrollado nuevos métodos disuasorios basados en el uso de marcas de agua digitales que se perfilan como la solución más eficaz, no obstante la protección de los derechos de autor continua siendo un problema abierto.

El presente trabajo pretende por una parte ofrecer una visión general del estado del arte del uso de marcas de agua digitales para aplicaciones de protección del copyright de forma que el lector sin conocimientos previos del tema pueda formarse una idea clara de que son las marcas de agua digitales, como funcionan, en que aplicaciones encuentran utilidad y que problemas sufren, entre otras cuestiones. Se incidirá especialmente en el caso de las imágenes digitales.

Las imágenes digitales con el enorme crecimiento de las redes globales como Internet se han revalorizado como un vehículo comunicativo fundamental, esto, unido a su facilidad de manejo y su pequeño tamaño las convierten en uno de los productos multimedia más vulnerables a los infringimientos de los derechos de autor. En una segunda parte de este trabajo aplicaremos los conceptos vistos anteriormente para desarrollar una aplicación práctica de las marcas de agua para la protección de los derechos de autor de imágenes digitales y evaluar sus propiedades. Se trata de una aplicación de protección del copyright mediante detección de copia, conocida genéricamente con el nombre de fingerprinting: cada imagen es marcada con una marca única asociada al usuario autorizado a utilizar dicha imagen, si en un momento determinado se encuentra una imagen cuyo uso no ha sido autorizado la marca presente en la imagen indicará el origen de su distribución ilegal.

Para ello, en primer lugar se diseñará una técnica de marcado, sencilla y que cumpla con los requisitos del tipo de aplicación a la que se destina: invisibilidad y robustez. Se estudiará la robustez de la técnica de marcado sometiendo a las imágenes marcadas a una serie de transformaciones simples y comprobando si resulta posible extraer la marca.

En segundo lugar se diseñará la aplicación propiamente dicha. Esta permitirá insertar y extraer las marcas de agua de las imágenes. Para implementar las marcas deberá utilizarse un código especial, diseñado para resistir colisiones de varios usuarios, son los denominados códigos anti-colisión. Entendemos por colisión un tipo de ataque también llamado confabulación en el que el atacante crea una nueva imagen con una marca modificada a partir de la colisión de la misma imagen marcada con diferentes marcas. La aplicación, usando dicho código, insertará una marca única en cada imagen que se asociará al usuario autorizado a utilizarla. Deberá por tanto gestionar una pequeña base de datos con la información de las imágenes marcadas y sus correspondientes usuarios.

Por ultimo se evaluará la resistencia del código utilizado como marca a la confabulación de dos usuarios. Para ello se diseñará una aplicación sencilla que permita generar una imagen confabulada a partir de la misma imagen con dos marcas diferentes, y se realizarán las pruebas para comprobar la robustez del código y determinar como podrían mejorarse los resultados.

Método utilizado.

La mayoría de las disciplinas en el campo de la Informática evolucionan extraordinariamente rápido, si ha esto unimos el hecho de que la protección del copyright mediante marcas de agua digitales es un campo relativamente nuevo tenemos una explicación de porque no existe una literatura impresa demasiado extensa sobre el tema. En este contexto la búsqueda de información en Internet se convierte en una herramienta extremadamente útil.

Para la elaboración de la primera parte del trabajo se ha realizado una búsqueda amplia de información en Internet, consultando artículos especializados y principalmente informes de investigadores, frecuentemente vinculados con el mundo universitario. También se han consultado algunos de los pocos libros existentes sobre la materia. El estudio sobre el estado del arte obtenido es fruto de la lectura, asimilación y síntesis de toda esta información.

Igualmente para la elaboración de la segunda parte del trabajo, de carácter más práctico, han resultado fundamentales las búsquedas de información en Internet, especialmente para el diseño de la técnica de marcado. Se han consultado diversos informes de los numerosos existentes con propuestas de técnicas de marcado por parte de investigadores de la materia, hasta encontrar uno que resultaba adecuado a los propósitos de este trabajo. A partir de dicho informe se ha realizado una implementación libre de la técnica de marcado descrita.

Para el diseño de las aplicaciones se han utilizado los conceptos de diseño orientado a objeto adquiridos a lo largo de los presentes estudios de Ingeniería Técnica en Informática de Sistemas.

La implementación de los programas necesarios se ha realizado en lenguaje Java bajo entorno Windows. Se ha elegido Java por ser un lenguaje orientado a objeto potente y versátil, que ofrece grandes facilidades para crear interfaces gráficas de usuario y que dispone de gran cantidad de librerías de clases implementadas. A este respecto, ha resultado útil el empleo de clases implementadas para el manejo del formato gráfico BMP que no se haya incluido en las librerías estándar de Java, así como una clase para realizar cálculos de la transformada del coseno discreto (DCT) usada en la implementación de la técnica de marcado.

Planificación del TFC.

El presente trabajo debe desarrollarse dentro de unos límites temporales estrictos, por lo que resulta necesaria una planificación cuidadosa. En primer lugar se ha

descompuesto el trabajo en una serie de tareas elementales que presentamos a continuación.

Búsquedas de información	
1.	Búsqueda de información sobre el formato gráfico BMP.
2.	Búsqueda de información sobre protección del copyright mediante el uso de marcas de agua.
3.	Búsqueda de información sobre códigos anti-confabulación.
Diseño	
4.	Diseño del algoritmo de marcado.
5.	Diseño de la aplicación de inserción/extracción de marcas.
6.	Diseño de la aplicación de confabulaciones.
Implementación	
7.	Implementación del algoritmo de marcado.
8.	Implementación del programa de inserción/extracción de marcas.
9.	Implementación del programa de confabulaciones
Pruebas	
10.	Pruebas de robustez del método de marcado.
11.	Pruebas de resistencia del código anti-confabulación.
Redacción de la memoria	
12.	Elaboración de la memoria: estado del arte.
13.	Elaboración de la memoria: diseño de aplicaciones y algoritmos.
14.	Elaboración de la memoria: manual de usuario.
15.	Elaboración de la memoria: anexos.
16.	Elaboración de la presentación del proyecto.

Tabla 1. Descomposición de tareas

A partir de esta descomposición de tareas, se ha estimado la duración de cada una de ellas y se ha realizado el siguiente gráfico de Gantt que refleja la temporización prevista.

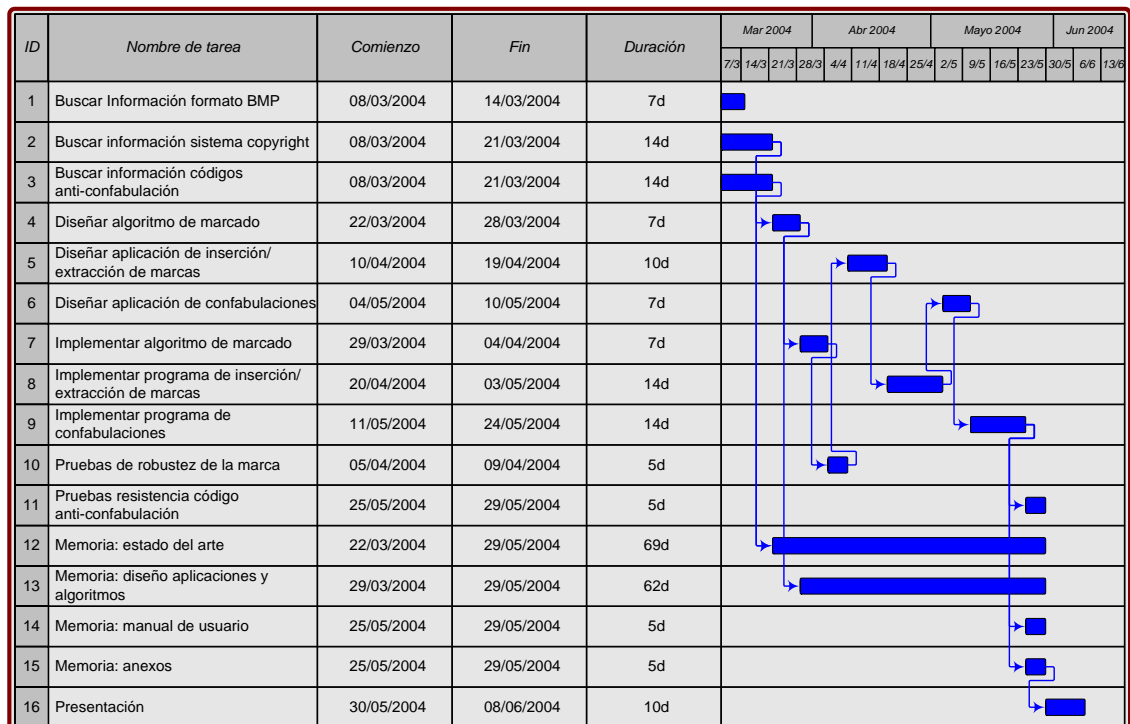


Ilustración 1. Planificación. Gráfico de Gantt

Productos obtenidos.

La labor realizada en el presente trabajo fructifica básicamente en tres productos:

- Un estudio del estado del arte. Informe teórico que nos ofrece una visión global del uso de las marcas de agua digitales para aplicaciones de protección de los derechos de autor, sus diferentes formas de usarlas y su problemática.
- Una aplicación de inserción/extracción de marcas. Que permite insertar y extraer marcas de las imágenes utilizando la técnica diseñada a ese objeto.
- Una aplicación de confabulaciones. Que permite generar imágenes confabuladas a partir de las copias de dos usuarios. Es independiente de la aplicación anterior y puede usarse sea cual sea la técnica de marcado empleada en las imágenes.

Capítulos siguientes.

El capítulo 1 presenta el estudio del estado del arte de las marcas de agua digitales, en el capítulo 2 se presenta la parte práctica de este trabajo, la sección 1 explica el diseño de la técnica de marcado y el resultado de las pruebas de robustez, la sección 2 el diseño de la aplicación de inserción y extracción de marcas, la sección 3 el diseño de la aplicación de confabulaciones y las pruebas de robustez ante las colisiones del código utilizado para implementar las marcas y finalmente en el capítulo 3 se presentan las conclusiones.

1. ESTADO DEL ARTE

1.1. El problema del copyright.

De forma simplificada el copyright es el derecho de un autor de ser dueño de su obra por un cierto periodo de tiempo. Ser dueño de su obra significa tener entre otros los derechos de hacer y distribuir copias de la obra, controlar su difusión, protegerla contra alteraciones y derecho a crear trabajos derivados. Estos derechos son exclusivos del autor, que por supuesto puede otorgar permisos a otros. Y son de una duración limitada que varía de país a país. El objetivo que se persigue es fomentar la creación de obras respaldando legalmente su explotación.

Los avances técnicos en el contexto del procesado, reproducción y distribución de las obras frecuentemente han tenido efectos contradictorios sobre los poseedores de un copyright. Por una parte mejoran la creación de obras y facilitan nuevas formas de explotación. Por otra, los mismos avances pueden ser usados contra los intereses de los dueños del copyright abriendo nuevos caminos para la violación de dicho copyright. Especialmente los desarrollos recientes en el manejo y la distribución de la información digital han tenido un tremendo impacto sobre los autores y sus derechos de autor.

En los últimos tiempos, el uso, distribución y comercialización de materiales multimedia digitales (imagen, video, música) ha experimentado un desarrollo notable debido al uso generalizado de ordenadores personales y al enorme crecimiento de Internet y ello ha traído asociado un aumento de los problemas relacionados con el copyright. Hasta el momento no existe ningún sistema perfecto para proteger estos materiales digitales frente a las copias ilegales. El motivo es claro: si hasta hace poco tiempo la realización de copias de obras creativas era un proceso bastante costoso, en la actualidad la revolución digital posibilita que los materiales multimedia digitales puedan ser fácilmente duplicados con una fidelidad idéntica a la del original, y lo que es más también posibilita que otras obras analógicas (como pinturas, dibujos o libros impresos) puedan ser digitalizadas y distribuidas en forma de perfectas copias digitales. En este panorama, la posibilidad de proteger los derechos de la propiedad intelectual de alguna forma se está convirtiendo en una autentica necesidad para los autores de estos trabajos.

Las soluciones técnicas para la protección del copyright, típicamente combinan métodos procedentes de varias áreas de investigación: junto a las medidas preventivas, que intentan evitar que se produzca el infringing del copyright, utilizando frecuentemente métodos criptográficos como los esquemas de encriptación, las funciones hash o las firmas digitales, y que por si solas no han conseguido mostrarse efectivas, aparece un nuevo enfoque, el de las medidas disuasorias, que si bien no ayudan a prevenir la violación del copyright, si lo hacen detectable, verificable y procesable, es el mundo de las marcas de agua digitales.

A pesar de la multitud y eficacia de estos mecanismos técnicos de protección, debe tenerse en cuenta que la protección de los derechos de la propiedad intelectual en la práctica no parece poder ser resuelta satisfactoriamente únicamente mediante estas medidas técnicas. Una protección satisfactoria de los derechos de la propiedad intelectual requeriría una adecuada y coordinada combinación de medidas legales, técnicas y económicas.

1.2. Introducción a las marcas de agua digitales.

1.2.1. Qué son las marcas de agua digitales.

Una marca de agua digital es una marca o etiqueta que se inserta en un archivo de datos digital de forma que la marca de agua puede ser posteriormente detectada o extraída para realizar una afirmación sobre el objeto. No se trata de una medida preventiva, no esta encaminada a evitar el uso indebido de los datos, sino que es mas bien una medida disuasoria, su utilidad radica en el hecho de que permite sacar conclusiones sobre el contenido de los datos y sobre el uso que se les ha dado (quien los creo y cuando, si han sido modificados, si tienen restricciones de uso, etc.).

Las marcas de agua pueden ser visibles o invisibles. Pese a que visible o invisible son términos visuales, las marcas de agua no se limitan a las imágenes, pueden ser también usadas para proteger otros tipos de objetos multimedia.

1.2.2. Origen de las marcas de agua digitales.

El uso de las marcas de agua tradicionales es casi tan viejo como la manufactura de papel. La técnica consiste en la impresión en el papel de una forma de imagen o texto derivada de un negativo en el molde donde las fibras de papel son prensadas y secadas. Las marcas de agua han sido ampliamente utilizadas desde la edad media y sus primeros usos estaban relacionados con el hecho de grabar la marca del fabricante en el producto de forma que su autenticidad pudiera ser claramente establecida sin degradar la estética y la utilidad del producto. En tiempos más recientes las marcas de agua se han utilizado por ejemplo para certificar la composición de un papel. En nuestros días la gran mayoría de países desarrollados utilizan marcas de agua en su papel moneda para dificultar su falsificación.

La digitalización de nuestro mundo ha expandido el concepto de las marcas de agua para incluir los materiales digitales para usos de protección de los intereses de los propietarios. No obstante, las marcas de agua digitales son semejantes a sus antecesoras sobre papel. Comunican algo acerca del documento o archivo en el que se hallan. Pretenden actuar como garantía de autenticidad, calidad, propiedad y procedencia. Al proceso de utilización de marcas de agua digitales sobre materiales multimedia se le conoce con el término inglés de *Digital Watermarking*, que podríamos traducir en lo sucesivo como marcado digital.

El marcado digital ha evolucionado a partir de la steganografía, con la que está estrechamente relacionado. La steganografía es una técnica antigua, cuyo nombre proviene del griego y viene a significar comunicación encubierta [1]. La idea principal de la steganografía es esconder un mensaje secreto ocultándolo o introduciéndolo en otros datos que no despiertan sospecha [2]. Se asume que la existencia del mensaje secreto es desconocida para terceras partes, de forma que la interceptación del mensaje no supondrá la revelación de la información oculta. Como consecuencia los métodos steganográficos no son robustos, si se manipulan los datos que contienen la información oculta, ésta puede perderse.

Por el contrario, el marcado digital posee la noción adicional de robustez ante los ataques. Podemos asumir que el contrario conoce la existencia de la información

oculta, y pese a ello le debe ser imposible eliminarla, degradarla selectivamente o reemplazarla por una fraudulenta. La información que se oculta mediante métodos steganográficos generalmente no esta relacionada con la cubierta (los datos donde se halla oculta) mientras que las marcas de agua pueden considerarse como un atributo de la señal donde son insertadas, proporcionando información adicional sobre ésta. Otra diferencia entre ambas técnicas, es que el marcado digital maneja habitualmente una cantidad de información mucho menor que la que se puede ocultar mediante la steganografía como consecuencia de los requerimientos de robustez del primero.

Los primeros textos que tratan sobre la idea del marcado digital datan de los inicios de la década de los 90 [3]. Parece ser que A. Tirkel fue el primero en acuñar la frase “water mark” que posteriormente se convertiría en watermark [4]. A partir de mediados de la década de los 90 el marcado digital recibiría una atención remarcable por parte de gran cantidad de investigadores, que comenzaron a crear una cantidad creciente de publicaciones sobre el tema, desde sus principios básicos hasta los más sofisticados algoritmos de marcado digital.

En lo que resta de documento cuando hablemos de marcas de agua nos estaremos refiriendo a marcas de agua digitales.

1.3. Esquema general del marcado digital.

En general, todo esquema de marcado digital consta de tres partes:

- Generación de la marca de agua.
- Inserción de la marca (algoritmo de inserción)
- Extracción o detección de la marca (algoritmo de verificación)

Cada propietario tiene una marca única o puede también poner diferentes marcas en diferentes objetos, el algoritmo de inserción incorpora la marca al objeto, el algoritmo de verificación autentica el objeto determinando tanto el propietario como la integridad del objeto.

1.3.1. Generación de la marca de agua.

La marca de agua es una información codificada que se inserta en un archivo de datos digital y que permite identificar la obra, pero sin afectarla. Por ejemplo puede identificar el nombre del autor, el del comprador, datos sobre su compra, uso, etc. Las marcas de agua pueden ser de cualquier naturaleza como un número, un texto o incluso una imagen. No obstante dependiendo de la técnica de inserción utilizada y de la aplicación a la que se destina, las marcas deberán cumplir ciertas condiciones. Por ejemplo para el caso de marcas frágiles o visibles estas condiciones son bastante flexibles y solo habrá restricciones en cuanto al tamaño de la marca o su formato (por ejemplo si la marca es un texto, que sólo contenga caracteres ASCII o si se trata de una imagen que este en formato BMP, etc.).

En las aplicaciones más exigentes, donde la robustez sea un factor importante, la marca de agua deberá tener un formato específico para que su uso resulte efectivo. En estos casos es común que el propietario del archivo a marcar proporcione una clave K a partir de la cual se generará una marca de agua apropiada a la técnica de marcado

utilizada. Matemáticamente podemos ver la marca de agua W como el resultado de cierta función F sobre una clave K ,

$$W = F(K). \quad (\text{Eq. 1.1})$$

Si además se tienen en cuenta las características particulares de los datos I sobre los que se insertará la marca, tendremos

$$W = F(K, I) \quad (\text{Eq. 1.2})$$

1.3.2. Inserción de la marca.

La figura 2 muestra el esquema general del proceso de inserción de una marca digital.

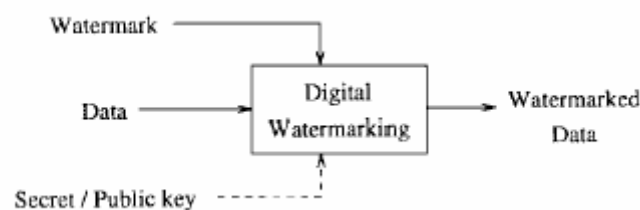


Ilustración 2. Proceso de inserción de marca

Las entradas son la marca digital W , los datos digitales I que contendrán la marca y una clave privada o pública K que es opcional y que en el caso de claves privadas se utilizará para reforzar la seguridad, por ejemplo haciendo que las posiciones donde se inserta la marca sean generadas a partir de dicha clave K .

En el proceso de inserción obtenemos los datos marcados I_w a partir de los datos originales I y la marca de agua W . Matemáticamente:

$$I_w = E(I, W) \quad (\text{Eq. 1.3})$$

La función de inserción E dependerá de la técnica de marcado concreta que se utilice. Como se verá posteriormente para el caso de las imágenes digitales, existen muy diversas técnicas de marcado.

1.3.3. Detección o extracción de la marca.

La figura 3 muestra el esquema general del proceso de detección/extracción de una marca digital.

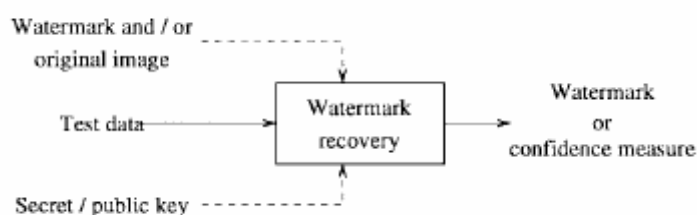


Ilustración 3. Proceso de verificación

Las marcas deben ser detectables o extraíbles para ser útiles. Dependiendo de la forma en que la marca es insertada y dependiendo de la naturaleza del algoritmo de marcado, el método usado para su detección/extracción puede implicar muy distintos acercamientos. En algunos esquemas de marcado, la marca puede ser extraída en su forma exacta. En el proceso de extracción una función de decodificación D toma unos datos I_w (que pueden estar marcados o no y presumiblemente corruptos) y extrae una marca W' . Matemáticamente:

$$W' = D(I_w) \quad (\text{Eq. 1.4})$$

En otros casos, podemos únicamente detectar cuando una marca específica está presente en la imagen. En este caso la función de decodificación D simplemente devuelve un valor de decisión Si/No acerca de la pregunta de si existe una marca determinada en los datos.

$$D(I_w, W, T) = \begin{cases} 1_{\text{si existe la marca } W \text{ en } I_w \text{ con fuerza} > T} \\ 0_{\text{en caso contrario}} \end{cases} \quad (\text{Eq. 1.5})$$

La marca puede estar debilitada debido a manipulaciones sobre los datos y muchas veces no puede decirse con absoluta certeza si se halla o no presente, el valor T se conoce como threshold, y es el valor mínimo de fuerza de la marca para el que diremos que se halla presente.

Debe resaltarse que la extracción de marca puede probar la propiedad mientras que la detección solo verifica la propiedad.

1.4. Clasificación y propiedades de las marcas de agua.

Tanto las marcas de agua como las técnicas de marcado pueden clasificarse en diversas categorías atendiendo a muy diversos criterios [5]. Cada una de esas categorías presentará unas propiedades determinadas que harán el marcado apto para ser empleado en un tipo de aplicaciones concreto. A continuación veremos las posibles clasificaciones de las marcas de agua y de las técnicas de marcado y sus propiedades básicas.

1.4.1. Visibles ó Invisibles.

Una primera clasificación fundamental de las marcas de agua es en lo que se refiere a si modifican los datos de forma perceptible para los usuarios o no.

Marcas visibles: suelen ser un logo o imagen traslucida sobrepuesta a los datos que se desea marcar. Se emplean normalmente para el marcado de imágenes o videos y su mayor propósito es indicar quien es el propietario de los datos. Las marcas visibles no son robustas puesto que su localización es obvia, lo que facilita los ataques sobre ella.

Las principales características deseables para este tipo de marcas son:

- Debe ser obvia tanto en imágenes en color como en imágenes monocromáticas.
- Debe abarcar una parte importante de la imagen de forma que no pueda ser eliminada recortando dicha imagen.

- Debe ser visible pero no interferir sobre los detalles de la imagen.
- Debe ser difícil de eliminar.
- Debe poder aplicarse automáticamente con poca intervención humana.

Marcas invisibles: la inserción de una marca en un archivo de datos implica necesariamente la modificación de dichos datos. Las marcas invisibles se realizan o bien en una parte del archivo de datos que no es directamente observable por el usuario, o bien se realizan de forma que resulten imperceptibles a los sentidos humanos. Esto último resulta posible debido a que los sentidos humanos (vista, oído) no son detectores perfectos y pueden aprovecharse sus limitaciones para conseguir que pequeñas modificaciones pasen desapercibidas. La mayor parte de las aplicaciones del mercado digital se basan en este tipo de marcas, puesto que uno de los objetivos que persiguen es que el archivo marcado no pierda su utilidad y el usuario final no pueda distinguirlo del original. Este tipo de marcas pueden ser utilizadas para todo tipo de materiales digitales. La principal característica que deben cumplir es que deben resultar invisibles sin degradar la calidad de los datos que la contienen.

1.4.2. Frágiles ó Robustas.

Las marcas de agua invisibles pueden clasificarse según su resistencia a las manipulaciones.

Marcas frágiles: Son aquellas que resultan alteradas o destruidas por ligeras manipulaciones de los datos en que se hallan. Se usan principalmente para verificar la integridad del archivo de datos sobre el que han sido insertadas. Si el usuario no puede extraer la marca del archivo de datos, o ésta no coincide con la marca original, tendrá constancia de que los datos han sido manipulados.

Las características deseables para este tipo de marcas son:

- Deben ser modificadas cuando se modifican ligeramente los datos que la contienen.
- La marca debe ser segura. Es decir, imposible de regenerar después de realizar modificaciones en los datos que la contienen incluso cuando se conoce el método de marcado o la estructura de la misma marca.

Marcas robustas: Son aquellas diseñadas para resistir diversos tipos de manipulaciones. Son utilizadas por buena parte de las aplicaciones del mercado digital que tienen unos requisitos de seguridad como es el caso de la protección de los derechos de autor. Lograr que la marca de agua sea robusta a todos los tipos de ataque es prácticamente imposible, por lo que se han desarrollado diferentes técnicas de marcado que se centran en lograr una robustez frente a determinados tipos de ataque. Dependerá de la aplicación a la que está destinada la elección de una técnica de marcado u otra.

Se pueden identificar unas pocas propiedades que comparten todos los sistemas de marcado robusto existentes:

- Imperceptibilidad. Las modificaciones causadas por la inserción de la marca digital deben permanecer por debajo de un valor límite de perceptibilidad. Son marcas invisibles.

- Robustez. Debe ser robusta a las distorsiones comunes de señal e incluso a ataques intencionados con el objeto de borrarla.
- Redundancia. Para asegurar la robustez, la marca se inserta de forma redundante a lo largo de toda la información que actúa como cubierta. De esta forma es posible extraer la marca incluso de una fracción pequeña de los datos marcados.
- Claves. En general estos sistemas suelen utilizar una o más claves criptográficamente seguras para obtener protección frente a la manipulación o borrado de las marcas.

1.4.3. Privadas ó Públicas.

Podemos establecer esta diferenciación de acuerdo a los requerimientos de seguridad de la clave utilizada en los procesos de inserción y recuperación de la marca. Normalmente se utiliza la misma clave en el proceso de inserción y en el de recuperación, si la clave es conocida, el tipo de marca se conoce como público, y si es secreta como privado. Las marcas públicas se pueden utilizar en aplicaciones que no tienen requerimientos especiales de seguridad.

1.4.4. Técnicas en el dominio espacial ó espectral.

Una forma de clasificar las técnicas de marcado digital (especialmente para el caso de imágenes digitales) es fijándonos sobre que dominio trabajan: el dominio espacial o el dominio espectral.

Técnicas sobre el dominio espacial: en este tipo de técnicas se manipulan directamente los bytes de la información que conforma la cubierta para almacenar la marca. Son los primeros tipos de técnicas de marcado que se implementaron y son en general esquemas relativamente simples que resultan poco resistentes a alteraciones comunes de la información que esconde la marca. Un ejemplo típico de este tipo de técnicas son para el caso de imágenes digitales las basadas en la modificación de los bits menos significativos de la imagen para ocultar la marca, que consiguen una perfecta invisibilidad al degradar muy poco la calidad de la imagen original.

Técnicas sobre el dominio espectral: Estas técnicas no trabajan directamente sobre los bytes de la información sino que la transforman previamente a otro dominio (utilizando transformadas de Fourier, del coseno discreto, wavelet etc.) para luego insertar la marca en dicho dominio. En general se considera que este tipo de técnicas son más robustas que las técnicas sobre el dominio espacial, sin embargo presentan un equilibrio más delicado entre invisibilidad y robustez puesto que la marca se aplica indiscriminadamente en el dominio espacial de la información.

1.4.5. Clasificación general de las técnicas de marcado.

De forma general podemos clasificar las técnicas de marcado en cuatro categorías atendiendo a la información que se utiliza durante el proceso de detección.

- Marcado Privado (nonblind watermarking). Requiere como mínimo el original de los datos marcados en el proceso de detección. Se puede subdividir en dos tipos:

- Tipo I. Detectan la marca con ayuda de la información original sin marcar:

$$D(I_w, I) = W' \quad (\text{Eq. 1.6})$$

- Tipo II. Requieren adicionalmente el uso de la marca original en el proceso de detección para responder a la pregunta de si una marca determinada se halla presente o no en la información.

$$D(I_w, I, W) = W' \quad (\text{Eq. 1.7})$$

$$Ct(W, W') = \begin{cases} 1 & \text{si } W = W' \text{ con fuerza } > t \\ 0 & \text{en caso contrario} \end{cases} \quad (\text{Eq. 1.8})$$

Este tipo de técnicas no debe confundirse con la definición dada anteriormente de marcas públicas y privadas.

- Marcado semiprivado (Semiblind watermarking). No usa el original de la información pero si la marca original.

$$D(I_w, W) = W' \quad (\text{Eq. 1.9})$$

$$Ct(W, W') = \begin{cases} 1 & \text{si } W = W' \text{ con fuerza } > t \\ 0 & \text{en caso contrario} \end{cases} \quad (\text{Eq. 1.10})$$

Muy usado en el caso de aplicaciones donde no es práctico el acceso a la información original.

- Marcado público (Blind watermarking). Sistemas en los que no se utiliza ni el original de la información ni el original de la marca durante el proceso de detección.

$$D(I_w) = W' \quad (\text{Eq. 1.11})$$

1.5. Aplicaciones de las marcas de agua.

Los diversos tipos de marcas de agua pueden ser utilizados en una amplia variedad de problemas [6], a continuación se exponen algunos de los más destacados.

1.5.1. Marcas visibles.

Las marcas visibles se utilizan principalmente para el marcado de imágenes, en este contexto podemos citar un par de aplicaciones usuales:

- Para protección de derechos de autor. El propietario de las imágenes añade una marca visible que permite el uso de la imagen para ciertos fines

(académicos, investigación, etc.) pero dificulta sus uso con fines comerciales sin pago de royalties, debido a que puede ser fácilmente identificada.

- Para indicar propiedad. El propietario de las imágenes añade una marca visible que lo identifica claramente de forma que al distribuir las imágenes quede bien clara la procedencia del material.

1.5.2. Marcas invisibles robustas.

Algunas aplicaciones típicas de este tipo de marcas son las siguientes.

- Para detección de copia y usos no autorizados. El vendedor incluye una marca invisible en el momento de la distribución de la imagen para indicar quien es el comprador. Si se encuentra una copia no autorizada se puede rastrear de donde procede en primera instancia recuperando la marca de la imagen. Esta aplicación se conoce también como fingerprinting o labelling.
- Como evidencia de propiedad. La marca insertada es un identificador del propietario. Si existiera una disputa sobre la propiedad del archivo, éste podrá demostrar que su marca se encuentra en el archivo y por tanto es de su propiedad.
- Control de uso. Este tipo de control se puede utilizar solo en el caso de que la reproducción o copiado del material multimedia requiera un hardware especial. Un ejemplo es el caso del video digital (DVD) en el que puede existir una marca en el disco para indicar si el video puede ser copiado o solo reproducido.

1.5.3. Marcas invisibles frágiles.

Las marcas frágiles encuentran su aplicación en los siguientes campos.

- Verificación de integridad de los datos. Se inserta una marca invisible con la propiedad de que cualquier alteración de los datos producirá una alteración de la marca, de esta forma chequeando la integridad de la marca presente podemos asegurarnos de la integridad de los datos.
- Marcado del contenido. En este caso simplemente la marca agrega una información adicional acerca de los datos del archivo. Puesto que no existen requerimientos de seguridad suelen emplearse marcas frágiles.

1.6. Técnicas de marcado robusto de imágenes digitales.

Una gran mayoría de las publicaciones sobre marcado digital están dirigidas al marcado de imágenes, posiblemente debido al hecho de que existe una gran demanda de productos de marcado a causa de la gran cantidad de imágenes disponibles a través de Internet que precisan mecanismos de protección. Ofrecer una visión completa de todas las técnicas resulta prácticamente imposible por lo que en este apartado sólo presentaremos algunas ideas generales. Para obtener una visión global de las técnicas de

marcado existentes consultar el documento [7]. A continuación enunciaremos las principales propiedades deseables para toda técnica de marcado robusto:

- Las diferencias entre la imagen original y la imagen marcada deben ser perceptualmente invisibles.
- Una marca invisible debe ser indetectable para cualquier usuario no autorizado.
- La marca debe ser resistente a las distorsiones comunes de las señales y resistente a distintos tipos de ataques intencionados. Por tanto no se podrá eliminar o alterar la marca hasta hacerla irreconocible sin degradar ostensiblemente la calidad de la imagen.
- La extracción de la marca debe ser un proceso simple que no requiera mucho tiempo de computación. En el caso de detección deberá ser precisa evitando los falsos positivos y los falsos negativos.
- Al igual que ocurre en las técnicas criptográficas, los detalles de los algoritmos de marcado digital deben ser públicos.
- La marca debe hallarse directamente codificada en la imagen.
- La marca debe estar insertada asimétricamente a lo largo de toda la imagen y no solo en una región de ella.
- Debe ser posible producir numerosas marcas, de otra forma solo podría marcarse un número limitado de imágenes.
- La marca extraída debe identificar sin ambigüedades al propietario.
- Debe ser usado un código corrector de errores que asegure que la marca sea resistente a modificaciones intencionadas.

Diseñar técnicas de marcado que cumplan todas estas propiedades y especialmente la resistencia a distintos tipos de ataques intencionados es muy difícil, por lo que continuamente surgen nuevas técnicas que satisfacen parte de estos requisitos encaminadas a ser usadas en aplicaciones concretas.

1.7. Problemática de las imágenes marcadas.

La robustez frente a los ataques es uno de los mayores requisitos del marcado digital para muchas de sus aplicaciones. Conseguir una robustez absoluta contra todos los posibles tipos de ataque y sus combinaciones es actualmente imposible, por lo que el objetivo práctico que se plantea a la hora de diseñar una técnica de marcado es que un ataque con éxito debe deteriorar la calidad de la imagen hasta el punto de devaluar significativamente su valor comercial antes de que la marca se deteriore lo suficiente como para que no pueda ser recuperada. Con un buen diseño puede conseguirse un método bastante robusto, sin embargo la robustez está frecuentemente reñida con la imperceptibilidad y la capacidad de almacenamiento de datos de la imagen por lo que

debe siempre buscarse un adecuado compromiso entre ambas partes dependiendo de la aplicación.

Podemos definir un ataque como cualquier manipulación de una imagen marcada con el objetivo de evitar el propósito de una técnica de marcado para una aplicación dada. Esto incluye las operaciones normales de procesado de imágenes como la compresión, conversión D/A y A/D, filtrados y otras operaciones que se pueden realizar habitualmente y que podrían destruir inintencionadamente la marca.

Una posible clasificación de los tipos de ataques propuesta por algunos autores [8] los divide en cuatro categorías:

- 1) Ataques simples. Son ataques que intentan dañar la marca insertada mediante manipulaciones de la imagen marcada sin intentar identificar y aislar la marca. Ejemplos de este tipo de ataques son el filtrado lineal y no lineal, la compresión con pérdida de información, la adición de ruido, la adición de un offset, los recortes, la cuantización de los píxeles, la conversión analógica y la corrección gamma.
- 2) Ataques de sincronización. Son ataques que intentan hacer imposible la recuperación de la marca sin llegar a dañarla pero haciendo que el detector pierda la sincronización. Se basan principalmente en distorsiones geométricas como los desplazamientos espaciales, las rotaciones, los recortes, la inserción o eliminación de píxeles o áreas, los zooms etc.
- 3) Ataques de ambigüedad. Son ataques que intentan confundir creando falsos originales o imágenes marcadas falsas. Un ejemplo sería el denominado ataque de inversión que se basa en introducir una o varias nuevas marcas en la imagen para intentar evitar que se conozca cual es la marca original y legítima.
- 4) Ataques de supresión. Son ataques que intentan analizar la imagen marcada para poder aislar la marca y eliminarla. Ejemplos de este tipo de ataques pueden ser los ataques por colisión, ciertas operaciones de filtrado no lineal, o los ataques dirigidos específicamente a un esquema de marcado determinado explotando puntos flacos conocidos de dicho esquema.

Esta clasificación no es estricta y algunos tipos de ataque pueden ser clasificados en varias de las categorías propuestas. Los diferentes tipos de ataque serán posibles en función de los conocimientos y las herramientas de que disponga el atacante, en general cuantos más conocimientos y herramientas mayor fuerza tendrán los ataques.

Otro problema es que la evaluación de la calidad de una técnica de marcado es un tema complicado que hasta el momento no tiene una solución plenamente satisfactoria y estandarizada. El problema engloba dos facetas: la calidad de la imagen marcada y la robustez de la técnica utilizada. Estos dos factores que determinan la calidad de una técnica de marcado, se hallan enfrentados y no pueden ser maximizados simultáneamente por lo que se deberá buscar el punto de equilibrio adecuado dependiendo de la aplicación específica y sus requerimientos.

1.8. Una aplicación concreta: fingerprinting de imágenes.

Con el término inglés de *fingerprinting* se conoce la técnica cuyo objetivo es poder identificar a los usuarios que intentan hacer un uso ilegal de un contenido multimedia, como por ejemplo su redistribución. El proceso consiste en insertar unas marcas especiales conocidas como *fingerprints* haciendo uso de técnicas de marcado digital robustas a una serie de ataques.

Esta técnica junto con el marcado digital como evidencia de propiedad constituyen quizás las dos principales aplicaciones de las marcas de agua digitales en la lucha por la protección del copyright de los objetos multimedia. La diferencia entre las dos técnicas radica en que en la segunda la marca insertada es la misma para todos los compradores de un objeto, típicamente un mensaje de copyright, mientras que en la primera cada objeto es marcado con una marca diferente, dependiente de la identidad del comprador y que lo identifica unívocamente.

Como sucede con otras técnicas criptográficas existen algunos problemas a ser resueltos. Uno de los más importantes cuando hablamos de *fingerprinting* es el problema de las colisiones. Si uno o varios atacantes disponen de diversas copias de una misma imagen con diferentes marcas pueden compararlas para identificar los bits que forman la marca y alterarlos creando una nueva imagen que no contiene ninguna de las marcas de las imágenes de que procede, en este momento los atacantes están en disposición de distribuir la imagen sin que puedan ser identificados. Se trata de un método efectivo y de bajo coste computacional para identificar y eliminar las marcas. Blakley et al. [9] fueron los primeros autores en presentar el problema de las colisiones. A continuación se describen dos formas simples de realizar este ataque:

- Ataque por promedios. Es un método simple que se basa en calcular la media de múltiples copias de una misma imagen con diferentes marcas. Sin embargo su inconveniente es que la efectividad del ataque aumenta con el número de copias utilizadas y para que sea realmente alta la efectividad el número de copias requerido es muy elevado y esto resulta difícil de conseguir para un atacante.
- Ataque del mosaico. En este tipo de ataque se crea una nueva imagen seleccionando diferentes bloques o píxeles a partir de diferentes copias marcadas de la misma imagen. Es un ataque efectivo incluso para un número bajo de copias.

La solución a estos ataques pasa por elegir cuidadosamente el formato de las marcas insertadas como *fingerprints* y utilizar códigos específicamente diseñados para resistir las colisiones de un número determinado de confabuladores.

En el llamado fingerprinting clásico (simétrico), es el vendedor el encargado de insertar la marca en las imágenes y posteriormente proporcionárselas al comprador. Si el vendedor detecta una imagen no autorizada podrá recuperar la marca para averiguar que comprador es el origen de la distribución ilegal, sin embargo este esquema presenta un grave inconveniente y es que la recuperación de la marca por parte del vendedor no constituye una prueba de que el comprador identificado sea el auténtico responsable de la distribución ilegal debido al hecho de que tanto comprador como vendedor conocen

la imagen marcada y el mismo vendedor podría generar una imagen marcada que acusase falsamente a un comprador determinado. Sucede lo mismo que por ejemplo con los códigos simétricos de autenticación de mensajes: ambas partes comparten la misma clave secreta y por tanto no proporcionan el servicio de no-repudiación.

Para solucionar este problema aparece el denominado fingerprinting asimétrico. La idea es conseguir un sistema similar a la firma digital, sólo la persona poseedora de una firma digital puede firmar mensajes con dicha firma, así el mensaje firmado no puede ser repudiado. En el fingerprinting asimétrico solamente el comprador conoce la imagen marcada, por lo que si el vendedor encuentra una imagen ilegal puede identificar al comprador y probar ante terceros que proviene del comprador identificado. En el proceso de marcado intervienen tanto el vendedor como el comprador. Pfitzmann y Schunter [10] fueron los introductores de este tipo de esquemas.

Un concepto interesante para el comercio electrónico es el de las transacciones anónimas, en las que el vendedor no conoce la identidad del comprador como sucede en el comercio tradicional cuando por ejemplo se paga en efectivo. Con la intención de proporcionar anonimato al comprador Pfitzmann y Waidner [11] introducen el denominado fingerprinting anónimo. La idea es añadir a los esquemas de fingerprinting asimétrico la característica del anonimato, en el que el vendedor no conoce la identidad del comprador sin que ello resulte en ningún impedimento para su posterior identificación en el caso de redistribución no autorizada. Este esquema se basa en las firmas públicas. De forma simplificada, el comprador elige un pseudónimo y obtiene un par de claves pública y privada con las que firma con su autentica identidad obteniendo un certificado de una entidad certificadora. La entidad certificadora será la encargada de revelar la identidad del comprador en caso de que el vendedor detecte un redistribución no autorizada proveniente de un pseudónimo determinado.

1.8.1. Códigos anti-colisión

La solución contra los ataques de colisión pasa por la utilización de códigos especiales diseñados para resistir este tipo de ataques. Una referencia importante a este respecto es el documento de Boneh y Shaw [12] presentado en 1995 en el que sientan las bases para la creación de códigos seguros a un número c determinado de confabulados, conocidos como códigos c -seguros.

En el citado documento se llega a la conclusión de que en el contexto del fingerprinting, los códigos totalmente seguros para un número de confabulados mayor o igual que dos sobre un número de usuarios mayor o igual que tres no existen. Sin embargo bajo la hipótesis conocida como “Marking Assumption” que asume que dada una confabulación por colisión solo se podrán detectar y alterar los bits de la marca que difieren entre las copias presentes, si pueden construirse códigos c -seguros con error ϵ , en los cuales dada una coalición de c miembros se puede capturar un miembro de la coalición con probabilidad $1-\epsilon$.

El mayor problema que se presenta hasta el momento es la longitud del código a utilizar. Boneh y Shaw proponen un método para la construcción de códigos para n usuarios de longitud:

$$O(\log^4 n \log^2 \frac{1}{\varepsilon}) \quad (\text{Eq. 1.12})$$

donde $\varepsilon < \frac{1}{n}$ es la probabilidad de error

La construcción de códigos cortos es un problema todavía abierto.

1.9. Conclusiones y futuro del mercado digital.

En el presente capítulo se ha pretendido dar una visión general del uso de las marcas de agua para protección de los derechos de autor. Se trata de un tema muy amplio por lo que si bien no se ha tratado en profundidad si se han expuesto los conceptos fundamentales necesarios para que el lector pueda tener una idea clara de las posibilidades que estas técnicas ofrecen. Hemos comenzado hablando de las marcas de agua usadas en los productos multimedia de forma general, se ha visto qué son las marcas de agua y como surgieron, se ha visto el esquema general del mercado digital, en que tipos podemos clasificar las marcas de agua, sus principales características y para que aplicaciones encuentran su utilidad. Posteriormente nos hemos centrado en el caso de las imágenes digitales, hemos visto que las aplicaciones de protección de los derechos de autor se basan en el uso de marcas robustas y se han enumerado sus principales características, seguidamente se han visto los puntos débiles de las marcas robustas y que se puede hacer para solventarlos. Se han expuesto los problemas existentes para evaluar la calidad de las técnicas de marcado. Finalmente se han expuesto las principales características de una de las aplicaciones de las marcas de agua robustas para la protección de los derechos de autor, la protección por detección de copia o fingerprinting, se ha descrito el principal problema que padecen: el ataque por colisión y hemos hablado de la solución a este tipo de ataque, los códigos anti-confabulación.

Pese a que el interés en el uso de este tipo de técnicas es creciente, tanto académico por parte de los investigadores como por parte de la industria donde cada vez existen mayor número de compañías desarrollando productos, lo cierto es que todavía no ha alcanzado un desarrollo plenamente satisfactorio y quedan muchos problemas por resolver. Al mismo tiempo que evoluciona la robustez de las técnicas también lo hacen los ataques y si bien los métodos actuales ofrecen robustez contra los usuarios no expertos pueden resultar vulnerables a los ataques de expertos. Existe un delicado equilibrio entre invisibilidad, robustez y capacidad de almacenamiento de información y un método de marcado invisible, robusto a todos los ataques y de gran capacidad es todavía una ilusión. Pese a que el uso como prueba de propiedad fue la motivación que dio origen a este tipo de técnicas todavía está lejano el día en que pueda llegar a ser aceptada como una prueba ante los tribunales, e incluso es posible que nunca llegue a suceder. Actualmente el uso de las marcas de agua para protección de los derechos de autor necesita ser combinado con otro tipo de mecanismos criptográficos para poder ofrecer una autentica protección.

Referencias

[1] M. Kobayashi, "Digital watermarking: Historical roots," IBM Research, Tokyo Res. Lab., Tech. Rep., Apr. 1997.

- [2] “Information Hiding Techniques for Steganography and Digital Watermarking “
Stefan Katzenbeisser, Fabien A.P. Petitcolas (Editores)
Artech House, Ene. 2000, ISBN: 1580530354
- [3] K. Tanaka, Y. Nakamura, and K. Matsui, “Embedding secret information into a dithered multilevel image,” in *Proc. 1990 IEEE Military Commun. Conf.*, Sept. 1990, pp. 216–220.
- [4] A. Tirkel, G. Rankin, R. van Schyndel, W. Ho, N. Mee, and C. Osborne, “Electronic water mark,” in *Proc. DICTA 1993*, Dec. 1993, pp. 666–672.
- [5] Saraju P. Mohanty , "Digital Watermarking: A Tutorial Review", URL:
<http://www.csee.usf.edu/~smohanty/research/Reports/WMSurvey1999Mohanty.pdf>
- [6] F.Mintzer, et al., "Effective and Ineffective Digital Watermarks", IEEE International Conference on Image Processing, ICIP-97, 1997, Vol.3, pp. 9-12.
- [7] F. Hartung and M. Kutter, "Multimedia watermarking techniques," *Proceedings of the IEEE*, vol. 87, no. 7, July 1999, pp. 1079-1107.
- [8] F. Hartung, J. K. Su, and B. Girod, “Spread spectrum watermarking: Malicious attacks and counterattacks,” in *Proc. SPIE Security and Watermarking of Multimedia Contents 99*, San Jose, CA, Jan. 1999.
- [9] Blakley, G. R., C. Meadows, and G. B. Purdy, "Fingerprinting Long Forgiving Messages," in *Advances in Cryptology, Proceedings of CRYPTO '85*, vol. 218 of *Lecture Notes in Computer Science*, Springer-Verlag, 1986, pp. 180–189.
- [10] Pfitzmann, B., and M. Schunter, "Asymmetric Fingerprinting," in *Advances in Cryptology, Proceedings of EUROCRYPT '96*, vol. 1070 of *Lecture Notes in Computer Science*, Springer-Verlag, 1996, pp. 84–95.
- [11] Pfitzmann, B., and M. Waidner, "Anonymous Fingerprinting," in *Advances in Cryptology, Proceedings of EUROCRYPT '97*, vol. 1233 of *Lecture Notes in Computer Science*, Springer-Verlag, 1997, pp. 88–102.
- [12] Boneh, D., and J. Shaw, "Collusion-secure Fingerprinting for Digital Data," in *Advances in Cryptology, Proceedings of CRYPTO '95*, vol. 963 of *Lecture Notes in Computer Science*, Springer-Verlag, 1995, pp. 452–465.

2. MEMORIA DE DISEÑO

En esta segunda parte del trabajo de carácter más práctico, se han desarrollado algunos de los conceptos teóricos vistos en el apartado anterior para crear una aplicación completamente operativa. Si bien el objetivo no es crear una aplicación con una utilidad práctica concreta si podría llegar a tenerla mejorando alguno de los conceptos que se han utilizado, tal y como se describirá posteriormente.

2.1. Diseño de la técnica de marcado.

El objetivo que nos hemos propuesto es el de diseñar un método sencillo para introducir una marca de agua invisible en una imagen digital de forma que presente cierta robustez, es decir una alteración ligera de la imagen marcada no ha de impedir recuperar la marca.

A tenor de lo visto en el estudio del estado del arte sabemos que los algoritmos propuestos para el marcado de imágenes pueden dividirse en dos grandes grupos según la forma en que modifican la imagen para introducir la marca:

- algoritmos en el dominio espacial, que insertan la marca de agua modificando directamente el valor de determinados píxeles de la imagen.
- algoritmos en otros dominios, que transforman la imagen al dominio elegido insertan la marca en la señal obtenida en ese dominio y finalmente aplican la transformación inversa para obtener la imagen marcada.

Se han estudiado diversas propuestas de técnicas de marcado de varios autores, con el objeto de encontrar una que se adaptase al requerimiento básico del presente trabajo: un correcto equilibrio entre sencillez y robustez. Tras alguna prueba inicial con resultados negativos, se ha encontrado una propuesta que prometía cumplir perfectamente con los objetivos propuestos, se trata de un algoritmo de marcado en el dominio DCT (Transformada del Coseno Discreto) propuesto por E. Koch & J. Zhao [1]. Es un hecho aceptado que en general los algoritmos de marcado sobre el dominio DCT ofrecen mayor robustez que los basados en el dominio espacial por lo que el requerimiento de robustez parece en principio asegurado. En cuanto al requerimiento de sencillez, el algoritmo en sí mismo es sencillo y la única dificultad radica en los cálculos necesarios de la transformada del coseno discreto, sin embargo dicha dificultad no lo es tanto si pensamos que debido a que estos cálculos se utilizan en el algoritmo de compresión de imágenes Jpeg es fácil encontrar implementaciones para ellos realizadas en diversos lenguajes de programación. La técnica diseñada que se presenta a continuación se basa en las ideas procedentes del citado informe de E. Koch & J. Zhao, sin embargo se ha simplificado y adaptado libremente a las necesidades del presente trabajo.

2.1.1. Algoritmo de inserción de marcas.

Un esquema general del algoritmo de inserción puede verse en la siguiente figura:

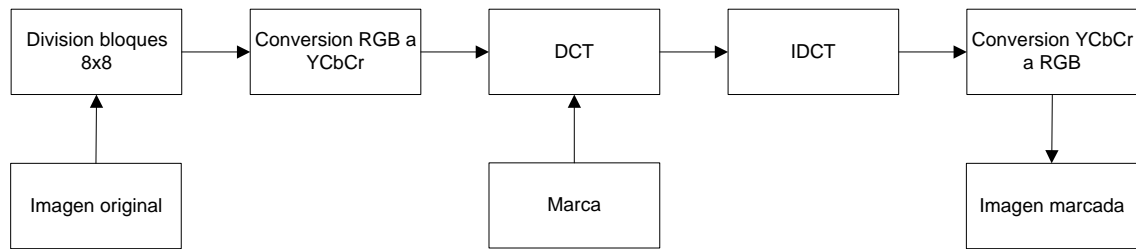


Ilustración 4. Diseño algoritmo de inserción

A continuación se describe el funcionamiento de cada uno de los bloques presentados en la figura anterior.

En un primer paso se divide la imagen a tratar en bloques de 8x8 píxeles, los bloques elegidos contendrán cada uno un bit de la marca. Este tipo de esquema presenta una capacidad de almacenamiento inferior a otros que introduzcan uno o varios bits en cada píxel o en cada bloque, no obstante puesto que nuestras necesidades de marcado son pequeñas (una marca estará formada por unos pocos bits), es más que suficiente para introducir la marca repetidamente a lo largo de la imagen. El hecho de que las dimensiones de la imagen no sean múltiplo de 8 tampoco supondrá ningún inconveniente, los píxeles sobrantes simplemente se despreciarán a la hora de introducir las marcas. Cada bloque de 64 valores de RGB se transforma al espacio de color conocido como YCbCr. Este modelo de espacio de color se caracteriza por dar información acerca de la Luminancia (Y) y la Crominancia (Cb y Cr). Para la conversión a este modo de color debemos aplicar las siguientes ecuaciones:

$$Y = 0.257 R + 0.504 G + 0.098 B + 16 \quad (\text{Eq. 2.1})$$

$$Cb = - 0.148 R - 0.291 G + 0.439 B + 128 \quad (\text{Eq. 2.2})$$

$$Cr = 0.439 R - 0.368 G - 0.071 B + 128 \quad (\text{Eq. 2.3})$$

En ellas podemos ver que el valor de la componente verde (G) es el que tiene mayor efecto sobre la luminancia, ello se debe a que el ojo humano es más sensible a dicha componente que a la roja y a la azul. Los valores de Cb y Cr nos indican como de azul y de rojo es el color. En nuestro algoritmo de marcado trabajaremos sobre los valores de la luminancia, estos son los que tienen una importancia perceptual mayor y por tanto son más respetados por los algoritmos de compresión como el Jpeg, proporcionando mayor robustez al marcado, no obstante son los que más fácilmente producen variaciones visibles en la imagen, por lo que se deberá llegar a un compromiso entre robustez e invisibilidad.

El siguiente paso es calcular la transformada del coseno discreto (DCT) del bloque de 8x8 valores de luminancia. La transformada DCT es una transformada semejante a la transformada rápida 2D de Fourier, ésta coge un conjunto de puntos de un dominio espacial y los transforma en una representación equivalente en el dominio de frecuencias. La ecuación de la Transformada Discreta del Coseno es la siguiente:

$$g(u, v) = \alpha_u \alpha_v \frac{2}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) \cos\left(\frac{(2x+1)u\pi}{2N}\right) \cos\left(\frac{(2y+1)v\pi}{2N}\right) \quad (\text{Eq. 2.4})$$

$$\alpha_u \alpha_v = \begin{cases} 1/2 & \text{cuando } u=v=0 \\ 1 & \text{en otro caso} \end{cases}$$

donde N es el tamaño del bloque cuadrado, en este caso como los bloques de la imagen son de 8x8 píxeles, N será igual a 8 y 'u' y 'v' son cada uno de los elementos del bloque que van desde 0,1,2,...N, en este caso, como los bloques tienen un tamaño de 8, 'u' y 'v' irán desde 0,1,2,...,7 y serán cada una de las componentes de una matriz de 8x8, donde la coordenada (0,0) será la esquina superior izquierda.

De forma breve, la transformada del coseno discreto es una técnica para convertir una señal en componentes de frecuencia elementales. Implica una descomposición de los datos de la imagen del bloque 8x8 en una serie de cosenos de diferentes frecuencias. Como ejemplo visual podemos ver la figura siguiente donde aparece una señal formada por la suma de dos cosenos de diferentes frecuencias, la onda roja de alta frecuencia y la azul de baja frecuencia.

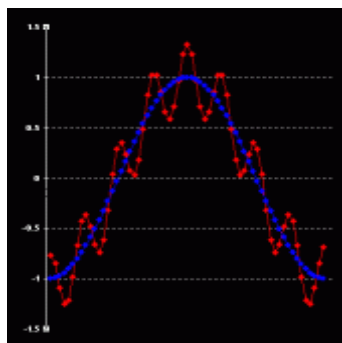


Ilustración 5. Descomposición en suma de cosenos

En la figura siguiente tenemos una matriz de 8x8 de las funciones base de la transformada del coseno discreto 2-dimensional.

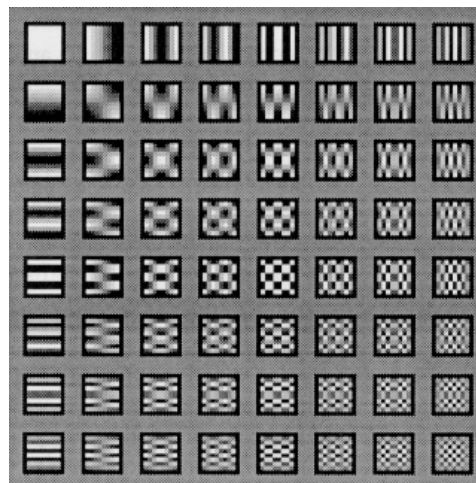


Ilustración 6. Funciones base DCT 2-dimensional

Podemos ver como aumentan la frecuencia horizontal de cada elemento de arriba hacia abajo y la frecuencia vertical de izquierda a derecha. El bloque de 8x8 coeficientes DCT describirá la proporción de cada una de estas funciones base que se hallan en la imagen. El elemento en la esquina superior izquierda se conoce como la

componente continua DC, siendo la media de los valores del bloque de entrada, se trata del coeficiente mayor en la DCT de imágenes naturales.

La marca, tal y como se observa en el esquema general del algoritmo, se introducirá en el bloque de 8x8 coeficientes DCT. Para ello se utiliza un método de marcado muy sencillo. Se elegirán dos coeficientes de la DCT, a los que podemos llamar B1 y B2.

Si $B1 > B2$ el bloque estará marcado con un 1.

Si $B1 < B2$ el bloque estará marcado con un 0.

En otro caso ($B1 = B2$) el bit de marca es indefinido.

A la hora de elegir los dos coeficientes B1 y B2 lo haremos en el rango de frecuencias medio-bajo. Los componentes de alta frecuencia tienen muy poco impacto visual sobre la imagen y son eliminados por los algoritmos de compresión como el Jpeg o por los procesos de filtrado de ruidos. Los elementos recomendados por E. Koch & J. Zhao en el informe [2] son los mostrados en las zonas sombreadas de la figura siguiente.

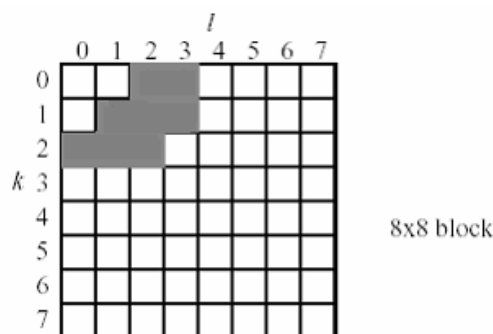


Ilustración 7. Coeficientes adecuados para marcado

Otro aspecto a tener en cuenta a la hora de la elección de coeficientes es la tabla de cuantización de valores de luminancia usada por el estándar Jpeg, que se muestra en la figura siguiente.

(u,v)	0	1	2	3	4	5	6	7
0	16	11	10	16	24	40	51	61
1	12	12	14	19	26	58	60	55
2	14	13	16	24	40	57	69	56
3	14	17	22	29	51	87	80	62
4	18	22	37	56	68	109	103	77
5	24	35	55	64	81	104	113	92
6	49	64	78	87	103	121	120	101
7	72	92	95	98	112	100	103	99

Ilustración 8. Tabla de cuantización luminancias Jpeg

Es conveniente que los coeficientes elegidos posean el mismo valor en la tabla de cuantización, de esta forma el proceso de compresión Jpeg respetará la proporción de ambos coeficientes a la hora de cuantificar y nuestro algoritmo será más robusto respecto a dicha compresión. Atendiendo a estos criterios las posiciones de los coeficientes elegidos para implementar el algoritmo son $B1 \rightarrow (2,0)$ y $B2 \rightarrow (1,2)$.

En orden a conseguir un equilibrio entre la robustez y la invisibilidad de la marca impondremos una serie de restricciones durante el proceso de marcado. El proceso será el siguiente:

Los bloques DCT de 8x8 los procesaremos de izquierda a derecha y de arriba abajo. Si los coeficientes elegidos son iguales o si su diferencia es superior a cierto valor límite máximo el bloque será descartado. De esta forma evitaremos que en caso de tener que invertir los coeficientes, los cambios sean notoriamente visibles en la imagen si la diferencia entre ellos es excesiva. También optaremos por incrementar la diferencia entre los dos coeficientes en caso de que ésta no llegue hasta un valor mínimo, incrementando así la robustez de la marca. Realizaremos un pequeño estudio para determinar los valores adecuados para ambos parámetros (valores máximo y mínimo).

Tal y como se ha descrito, la localización de los bits de la marca no será fija y será dependiente de la imagen. Una buena cantidad de bloques puede ser descartada, no obstante dadas las bajas necesidades en cuanto a cantidad de bits de la marca los bloques válidos serán suficientes para contener diversas veces la marca incluso en una imagen de pequeñas dimensiones.

2.1.2. Algoritmo de recuperación de la marca.

La figura siguiente presenta un esquema general del algoritmo de recuperación de marcas.

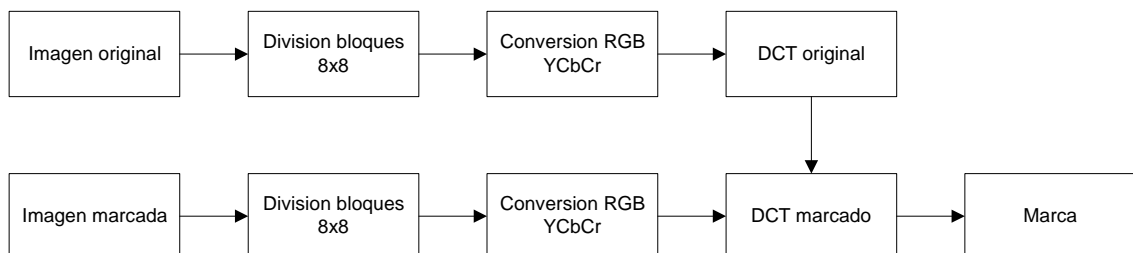


Ilustración 9. Diseño algoritmo de recuperación de marcas

El proceso seguido para extraer una marca de la imagen es el siguiente:

Dividimos la imagen original en bloques de 8x8 píxeles, los bloques se procesarán en el mismo orden que en el apartado anterior, de izquierda a derecha y de arriba abajo.

Para cada bloque convertimos de RGB a YCbCr y calculamos la transformada del coseno discreto (DCT) del bloque de 8x8 de luminancias (Y). Sobre el bloque de coeficientes DCT comprobamos si los dos coeficientes B1 y B2 anteriormente mencionados cumplen las restricciones necesarias para ser un bloque válido para marcado (ver apartado anterior). Si no es un bloque válido se descarta y se procede de igual manera con el siguiente bloque de la imagen.

Si el bloque es válido para marcado, obtenemos el mismo bloque de la imagen marcada, convertimos al espacio de color YCbCr y calculamos la DCT del bloque de luminancias Y. Finalmente extraemos el bit de la marca:

- Si $B1 > B2$ extraemos un bit 1.
- Si $B1 < B2$ extraemos un bit 0.
- Si $B1 = B2$ el bit de marca es indefinido.

El proceso se repite hasta procesar todos los bloques de la imagen. Puesto que la marca se halla insertada repetidamente en la imagen se utilizarán unos contadores para contabilizar los resultados de las extracciones de cada posición de la marca (0 ó 1). Los bits se encuentran almacenados de forma secuencial desde el primero de la marca hasta el último y vuelta a comenzar. Es posible que la última marca no se halle completa. Para cada posición de la marca el resultado final del bit extraído será el correspondiente al contador mayor (0 ó 1).

¿Por qué se utiliza la imagen original en el proceso de recuperación?

E. Koch & J. Zhao proponen en su informe realizar el marcado de la imagen tras efectuar los tres primeros pasos del modelo de compresión Jpeg: normalización, transformación DCT y cuantización. Tras el proceso de cuantización se produce una “pérdida de información” de la imagen que será tanto mayor cuanto menor sea el factor de calidad que se utilice en la compresión. En el esquema utilizado en el presente trabajo se ha optado por trabajar directamente sobre los coeficientes DCT por dos motivos:

Primero, evitar la pérdida de calidad de la imagen inherente al proceso de cuantización. Segundo, el trabajo con los coeficientes DCT es más sencillo que con los coeficientes cuantizados en el sentido de que pequeños cambios en los coeficientes DCT son menos perceptibles visualmente que los cambios sobre los coeficientes cuantizados, que se verán amplificados en el proceso de decuantización. No obstante el trabajo directo con los coeficientes DCT implica un problema, es fácil que por cuestiones de redondeo durante los cálculos de transformación de un dominio a otro (espacial-frecuencial) y de un espacio de color a otro (RGB-YCbCr) pueda producirse una pequeña variación en dichos coeficientes, según la precisión de las rutinas empleadas. Dicha variación, puede ser suficiente para que por ejemplo un mismo bloque sea considerado inválido para marcado en la imagen original, y válido en la imagen marcada, por lo que para determinar en que bloques se hallan los bits de la marca no podemos utilizar solamente la imagen marcada, sino que debemos utilizar la imagen original.

Dada la naturaleza de la aplicación final que se pretende desarrollar, el uso de la imagen original en el proceso de extracción de la marca no supondrá ningún inconveniente. Por el contrario supone un aumento de la seguridad, ya que incluso conociendo el algoritmo de marcado en muchos casos no resulta posible recuperar la marca correcta sin hacer uso de la imagen original.

Recuperación de la marca a partir de un recorte.

Con el método descrito anteriormente podremos recuperar los bits de la marca que contenga cualquier recorte de una imagen marcada. El único requisito será proporcionar las coordenadas de la esquina superior izquierda del recorte respecto de la imagen

original. Mediante el uso de un programa de retoque fotográfico y comparándola con la imagen original, resulta fácil obtener las coordenadas de dicha posición.

La forma de proceder será comprobar para cada bloque de la imagen original que se considera válido para marcado si queda dentro del área del recorte de la imagen marcada, si es así se extrae el bit de la marca, sino simplemente se incrementa la posición del bit de marca esperado. Para más detalles ver el diagrama de flujo del método de extracción de la marca.

Indicador de marca única.

Este indicador si esta activado indica que en el proceso de recuperación queremos recuperar una única marca (la primera marca que encuentre). Resulta útil en algunos casos como el ataque por colisiones (ver memoria de diseño del programa de confabulaciones).

2.1.3. Diseño Estático.

A continuación se presenta el diseño estático muy sencillo de las clases necesarias para implementar la técnica de marcado y los diagramas de flujo de los métodos más complicados, el método de inserción de marcas y el de extracción.

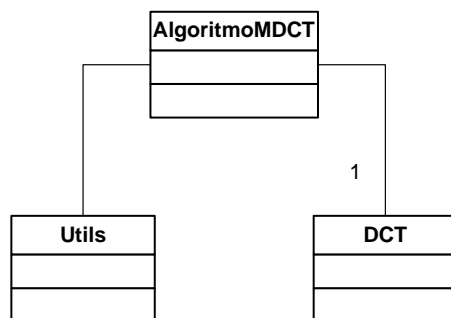


Ilustración 10. Diagrama estático técnica de marcado

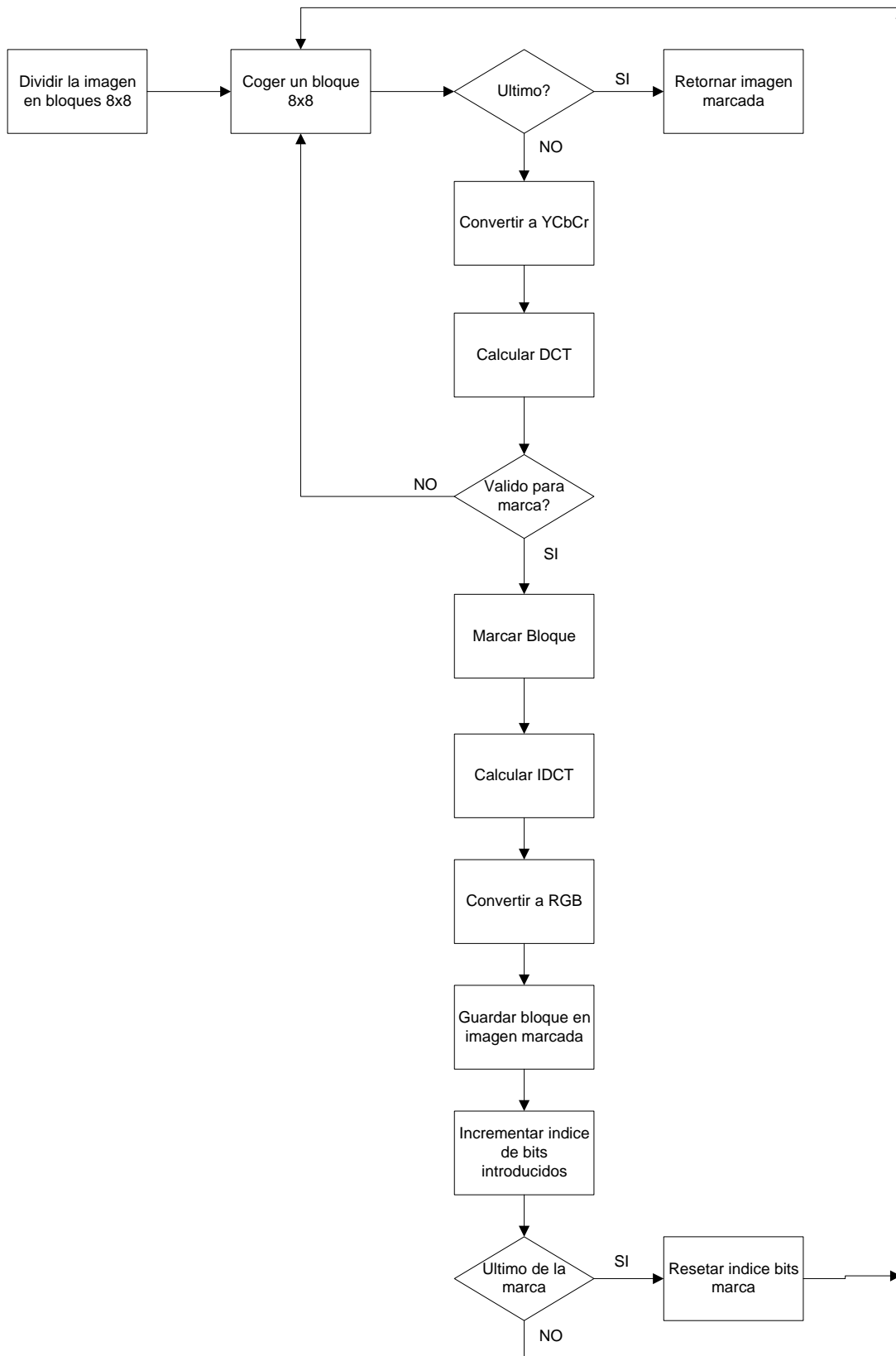
Diagrama de flujo método marcarImagen()**Ilustración 11. Diagrama de flujo método inserción de marcas**

Diagrama de flujo del método obtenerMarca()

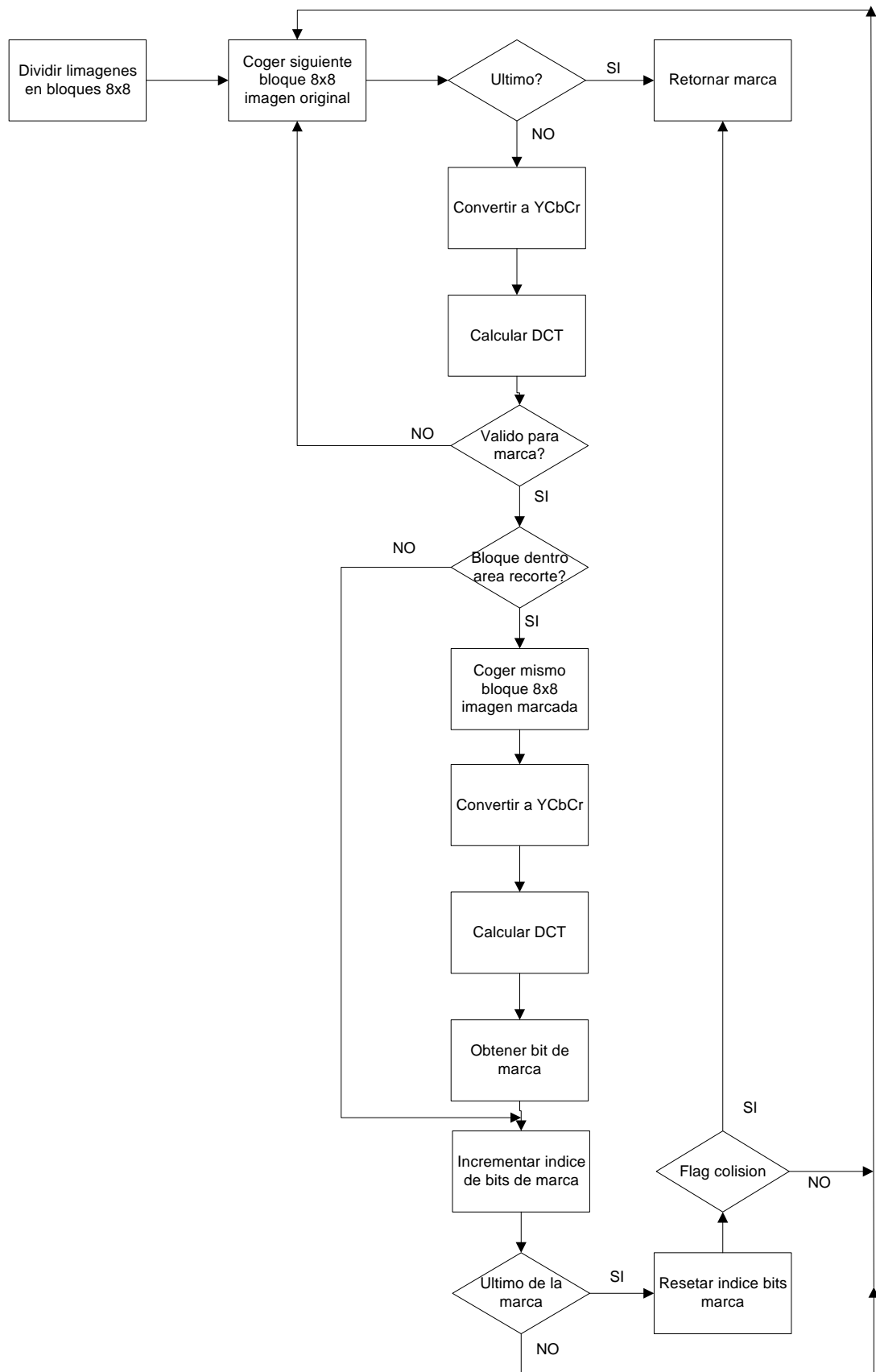


Ilustración 12. Diagrama de flujo método extracción de marcas

2.1.4. Elección de los parámetros máximo y mínimo.

La elección de los parámetros utilizados por el método de marcado se ha realizado de forma empírica. Para ello hemos hecho un estudio sobre dos imágenes con diversos valores de los parámetros y observado su efecto en cuanto a visibilidad del marcado y resistencia a la compresión Jpeg. Hemos elegido unos valores que no produzcan efectos visibles y presenten un buen resultado a la compresión, teniendo en cuenta que los resultados son dependientes de las características de las imágenes utilizadas.

Imagen A – lena color lowres, 128x128 píxeles formato bmp color 8 bits.

Tabla 2. Estudio parámetros para la imagen A

parámetros	Bloques válidos	Bloques inválidos	Efectos visibles	Jpeg 80%	Jpeg 70%	Jpeg 60%	Jpeg 50%	Jpeg 40%	Jpeg 30%
Max = 8,Min = 0	102	154	NO	SI	NO	-	-	-	-
Max = 10,Min = 0	112	144	NO	SI	NO	-	-	-	-
Max = 15,Min = 0	142	114	NO	SI	SI	NO	-	-	-
Max = 20,Min = 0	160	96	NO	SI	SI	SI	SI	NO	-
Max = 25,Min = 0	171	85	SI	SI	SI	SI	SI	NO	-
Max = 30,Min = 0	180	76	SI	SI	SI	SI	NO	-	-
Max = 8,Min = 4	102	154	NO	SI	SI	NO	-	-	-
Max = 10,Min = 4	112	144	NO	SI	SI	SI	NO	-	-
Max = 15,Min = 4	142	114	NO	SI	SI	SI	NO	-	-
Max = 20,Min = 4	160	96	NO	SI	SI	SI	SI	SI	NO
Max = 25,Min = 4	171	85	SI	SI	SI	SI	SI	NO	-
Max = 30,Min = 4	180	76	SI	SI	SI	SI	SI	SI	NO
Max = 8,Min = 6	102	154	NO	SI	SI	SI	NO	-	-
Max = 10,Min = 6	112	144	NO	SI	SI	SI	NO	-	-
Max = 15,Min = 6	142	114	NO	SI	SI	SI	NO	-	-
Max = 20,Min = 6	160	96	NO	SI	SI	SI	SI	SI	NO
Max = 25,Min = 6	171	85	SI	SI	SI	SI	NO	-	-
Max = 30,Min = 6	180	76	SI	SI	SI	SI	SI	NO	-
Max = 8,Min = 8	102	154	NO	SI	SI	SI	NO	-	-
Max = 10,Min = 8	112	144	NO	SI	SI	SI	NO	-	-
Max = 15,Min = 8	142	114	NO	SI	SI	SI	NO	-	-
Max = 20,Min = 8	160	96	NO	SI	SI	SI	SI	SI	NO
Max = 25,Min = 8	171	85	SI	SI	SI	SI	SI	NO	-
Max = 30,Min = 8	180	76	SI	SI	SI	SI	SI	SI	NO
Max = 8,Min = 10	102	154	NO	SI	SI	SI	NO	-	-
Max = 10,Min = 10	112	144	NO	SI	SI	SI	NO	-	-
Max = 15,Min = 10	142	114	NO	SI	SI	SI	NO	-	-
Max = 20,Min = 10	160	96	NO	SI	SI	SI	SI	SI	NO
Max = 25,Min = 10	171	85	SI	SI	SI	SI	SI	NO	-
Max = 30,Min = 10	180	76	SI	SI	SI	SI	SI	SI	NO
Max = 8,Min = 15	102	154	NO	SI	SI	SI	NO	-	-
Max = 10,Min = 15	112	144	NO	SI	SI	SI	NO	-	-
Max = 15,Min = 15	142	114	NO	SI	SI	SI	NO	-	-
Max = 20,Min = 15	160	96	NO	SI	SI	SI	SI	SI	NO
Max = 25,Min = 15	171	85	SI	SI	SI	SI	SI	SI	NO
Max = 30,Min = 15	180	76	SI	SI	SI	SI	SI	SI	NO

Imagen B – corredor, 283x212 píxeles formato jpeg.

Tabla 3. Estudio parámetros para la imagen B

parámetros	Bloques válidos	Bloques inválidos	Efectos visibles	Jpeg 80%	Jpeg 70%	Jpeg 60%	Jpeg 50%	Jpeg 40%	Jpeg 30%
Max = 8,Min = 0	377	533	NO	SI	SI	SI	NO	-	-
Max = 10,Min = 0	383	527	NO	SI	SI	SI	NO	-	-
Max = 15,Min = 0	460	450	NO	SI	SI	SI	SI	NO	-
Max = 20,Min = 0	495	415	NO	SI	SI	SI	NO	-	-
Max = 25,Min = 0	518	392	NO	SI	SI	SI	SI	NO	-
Max = 30,Min = 0	533	377	NO	SI	SI	SI	SI	NO	-
Max = 8,Min = 4	377	533	NO	SI	SI	SI	NO	-	-
Max = 10,Min = 4	383	527	NO	SI	SI	SI	NO	-	-
Max = 15,Min = 4	460	450	NO	SI	SI	SI	SI	NO	-
Max = 20,Min = 4	495	415	NO	SI	SI	SI	SI	NO	-
Max = 25,Min = 4	518	392	NO	SI	SI	SI	SI	NO	-
Max = 30,Min = 4	533	377	NO	SI	SI	SI	SI	NO	-
Max = 8,Min = 6	377	533	NO	SI	SI	SI	SI	NO	-
Max = 10,Min = 6	383	527	NO	SI	SI	SI	NO	NO	-
Max = 15,Min = 6	460	450	NO	SI	SI	SI	SI	NO	-
Max = 20,Min = 6	495	415	NO	SI	SI	SI	SI	NO	-
Max = 25,Min = 6	518	392	NO	SI	SI	SI	SI	NO	-
Max = 30,Min = 6	533	377	NO	SI	SI	SI	SI	NO	-
Max = 8,Min = 8	377	533	NO	SI	SI	SI	SI	NO	-
Max = 10,Min = 8	383	527	NO	SI	SI	SI	SI	NO	-
Max = 15,Min = 8	460	450	NO	SI	SI	SI	SI	NO	-
Max = 20,Min = 8	495	415	NO	SI	SI	SI	SI	NO	-
Max = 25,Min = 8	518	392	NO	SI	SI	SI	SI	NO	-
Max = 30,Min = 8	533	377	NO	SI	SI	SI	SI	NO	-
Max = 8,Min = 10	377	533	NO	SI	SI	SI	SI	NO	-
Max = 10,Min = 10	383	527	NO	SI	SI	SI	SI	NO	-
Max = 15,Min = 10	460	450	NO	SI	SI	SI	SI	NO	-
Max = 20,Min = 10	495	415	NO	SI	SI	SI	SI	NO	-
Max = 25,Min = 10	518	392	NO	SI	SI	SI	SI	NO	-
Max = 30,Min = 10	533	377	NO	SI	SI	SI	SI	NO	-
Max = 8,Min = 15	377	533	NO	SI	SI	SI	SI	NO	-
Max = 10,Min = 15	383	527	NO	SI	SI	SI	SI	NO	-
Max = 15,Min = 15	460	450	NO	SI	SI	SI	SI	SI	NO
Max = 20,Min = 15	495	415	NO	SI	SI	SI	SI	SI	NO
Max = 25,Min = 15	518	392	NO	SI	SI	SI	SI	SI	NO
Max = 30,Min = 15	533	377	NO	SI	SI	SI	SI	SI	NO

A partir de los datos de la ambas tablas podemos obtener los siguientes gráficos:

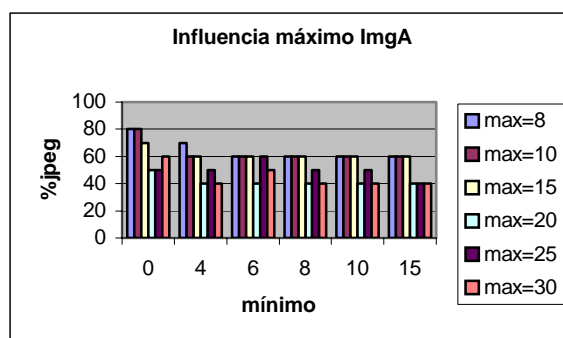
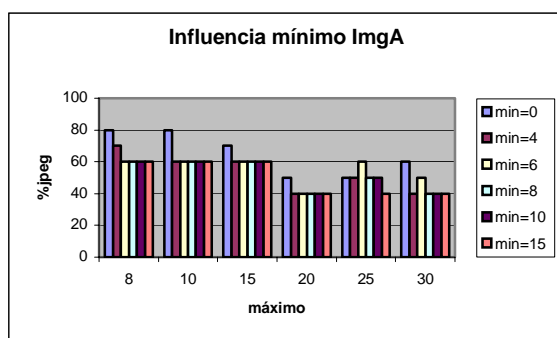


Ilustración 13. Influencia parámetro mínimo en la imagen A

Ilustración 14. Influencia parámetro máximo en la imagen A

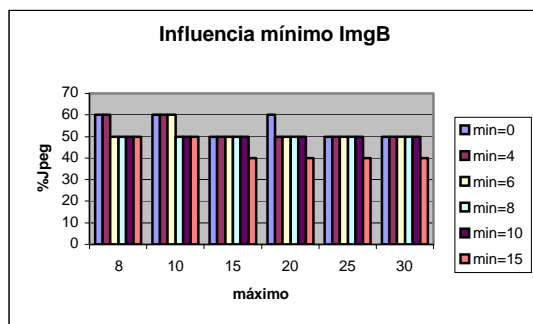


Ilustración 15. Influencia parámetro mínimo en la imagen B

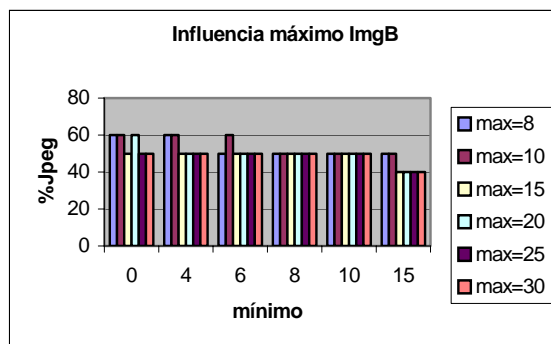


Ilustración 16. Influencia parámetro máximo en la imagen B

De los resultados de las pruebas observamos que la capacidad de almacenamiento de bits en la imagen marcada (payload) depende del parámetro máximo, cuanto mayor sea éste mayor es la capacidad. Así mismo los efectos visibles en la imagen son también mayores cuanto mayor es el parámetro máximo. Dichos efectos se han evaluado de forma subjetiva, por observación detallada de la imagen. Para valores del parámetro máximo superiores a 20 comienzan a observarse efectos visibles sobre la imagen A, con dificultad observables a simple vista debido al pequeño tamaño de la imagen, pero perceptibles si ampliamos la imagen. Puesto que un parámetro máximo de 20 presenta una buena capacidad de almacenamiento (más de la mitad de la imagen válida para marcado) y nuestras necesidades no son muy elevadas no pasaremos de ese valor.

Observando los gráficos vemos que para un valor de máximo=20, un buen valor para el parámetro mínimo es 8, que presenta unos buenos resultados a la compresión (40% imagenA, 50% imagenB) sin afectar visiblemente las imágenes. Valores superiores del mínimo=8 no aportan en principio ninguna mejora.

Así pues los valores utilizados serán máximo=20 y mínimo=8.

2.1.5. Pruebas de robustez.

Para comprobar la robustez de la técnica de marcado someteremos a varias imágenes marcadas a diversas transformaciones simples.

Las imágenes originales utilizadas serán:



Ilustración 17. Img1 lena.bmp 512x512 24-bits color



Ilustración 18. Img2 corredor.jpg 283x212 24-bits color



**Ilustración 19. Img3 logo_uoc.bmp 150x38
24-bits color**



**Ilustración 20. Img4 steve.jpg 320x240 8-bits
escala de grises**

Cada imagen ha sido marcada con varios códigos diferentes (marcas de 7 bits de longitud, no se ha usado ningún ECC) y sometida a las pruebas mostradas en la siguiente tabla:

Prueba	Descripción
JPEG90	Compresión Jpeg al 90%
JPEG60	Compresión Jpeg al 60%
JPEG40	Compresión Jpeg al 40%
REI5	Recorte a partir de la esquina superior izquierda a un tamaño del 5% del tamaño de la imagen original.
REI10	Recorte a tamaño del 10%
REI30	Recorte a tamaño del 30%
REI50	Recorte a tamaño del 50%
RE5	Recorte a partir de un punto cualquiera de tamaño 5% de la imagen
RE10	Recorte a partir de un punto cualquiera de tamaño 10% de la imagen
RE30	Recorte a partir de un punto cualquiera de tamaño 30% de la imagen
RE50	Recorte a partir de un punto cualquiera de tamaño 50% de la imagen
ESC25	Escalado de la imagen a un 25% de su tamaño original, grabar y volver a escalar a tamaño original
ESC50	Escalado a un 50%
ESC75	Escalado a un 75%
ESC125	Escalado a un 125%
ESC150	Escalado a un 150%
EF1	Eliminación de una fila de píxeles
EF5	Eliminación de un bloque de 5 filas de píxeles
EF10	Eliminación de un bloque de 10 filas de píxeles
EC1	Eliminación de una columna de píxeles
EC5	Eliminación de un bloque de 5 columnas de píxeles
EC10	Eliminación de un bloque de 10 columnas de píxeles
R30	Adición de ruido al 30%
R60	Adición de ruido al 60%
FPA	Filtro pasa alto radio 10 píxeles. Elimina detalles de baja frecuencia
CVG	Conversión a escala de grises (8 bits)
CV8	Conversión a 8 bits color
CVGIF	Conversión a formato gif
ROT5	Rotación de 5° y grabar, rotación inversa y recortar a tamaño original
ROT10	Rotación de 10° y grabar, rotación inversa y recortar a tamaño original
ROT30	Rotación de 30° y grabar, rotación inversa y recortar a tamaño original

Tabla 4. Pruebas de robustez

Los resultados obtenidos se presentan en la siguiente tabla

Prueba	Img1	Img2	Img3	Img4
JPEG90	SI	SI	SI	SI
JPEG60	SI	SI	NO	SI
JPEG40	SI	SI	NO	SI
REI5	SI	SI	NO	SI
REI10	SI	SI	NO	SI
REI30	SI	SI	NO	SI
REI50	SI	SI	NO	SI
RE5	SI	SI	NO	SI
RE10	SI	SI	NO	SI
RE30	SI	SI	NO	SI
RE50	SI	SI	NO	SI
ESC25	NO	NO	NO	NO
ESC50	SI	SI	SI	SI
ESC75	SI	SI	SI	SI
ESC125	SI	SI	SI	SI
ESC150	SI	SI	SI	SI
EF1	SI	SI	NO	SI
EF5	SI	SI	NO	SI
EF10	SI	SI	NO	SI
EC1	SI	SI	NO	SI
EC5	SI	SI	NO	SI
EC10	SI	SI	NO	NO
R30	SI	SI	NO	NO
R60	SI	NO	NO	NO
FPA	SI	SI	NO	SI
CVG	SI	SI	SI	-
CV8	SI	SI	NO	SI
CVGIF	SI	SI	NO	SI
ROT5	SI	SI	SI	SI
ROT10	SI	SI	SI	SI
ROT30	SI	SI	SI	SI

Tabla 5. Resultados pruebas de robustez

En general el método de marcado tiene un buen comportamiento ante los ataques utilizados salvo para el caso de la imagen 3 y ello es debido a que se trata de un logo de pequeño tamaño y que presenta un color muy uniforme. La resistencia a la compresión Jpeg es buena llegando en 3 de las 4 imágenes al 40%. El comportamiento ante los recortes también es muy bueno, salvo para la imagen 3. Ante los escalados también presenta un buen comportamiento superando todos los ataques menos el que reduce el tamaño de la imagen hasta un 25% de su tamaño original. Sorprenden positivamente los resultados a la eliminación de porciones de la imagen, encaminados a producir una pérdida de sincronía entre el algoritmo de extracción y la marca y que son superados en casi todos los casos sin necesidad de restaurar la imagen a su tamaño original. La robustez a la adición de ruido no es muy buena sólo una de las imágenes supera el ataque de 60% de ruido. En cambio los resultados ante los cambios de formato probados si son buenos. También son buenos los resultados ante las rotaciones, a partir del original podemos determinar con bastante precisión el ángulo de rotación necesario para invertir ésta y luego recortar si es necesario al tamaño original.

2.1.6. Posibles mejoras.

Una posible mejora del método de marcado radica en aumentar la capacidad de almacenar bits de la imagen marcada. Una forma de conseguirlo es aumentar el número de bloques válidos para marcado y ello puede lograrse utilizando más coeficientes para almacenar los bits de la marca. Podemos elegir tres o cuatro parejas de coeficientes y formar una lista ordenada con ellos de forma que si los dos primeros no son válidos para marcado pasaríamos a considerar los dos siguientes y así sucesivamente hasta agotar la lista. Otra posibilidad a estudiar sería la de incrementar el número de bits de la marca introducidos en cada bloque utilizando varias de las parejas propuestas en la mejora anterior de forma simultánea. Sería necesario estudiar sus efectos sobre la invisibilidad del marcado. También podría aumentarse la seguridad del método de marcado si se utilizara una clave para seleccionar los bloques que se utilizarán para albergar los bits de la marca.

Referencias

- [1] E. Koch and J. Zhao, "Toward robust and hidden image copyright labeling," in *Proc. Workshop Nonlinear Signal and Image Processing*, Marmaros, Greece, June 1995.
- [2] E. Koch and J. Zhao, "Embedding robust labels into images for copyright protection", *Proc. Of the International Congress on Intellectual Property Rights for Specialized Information, Knowledge and News Technologies*, Vienna Austria, August 21,25, 1995, 242-251

2.2. Diseño de la aplicación de inserción/extracción de marcas: ApplMDCT.

Se ha denominado con el nombre de ApplMDCT a la aplicación principal que utilizando el algoritmo de marcado diseñado en el apartado anterior permitirá insertar y extraer marcas de las imágenes digitales. Se trata de una aplicación sencilla enfocada a la protección del copyright mediante detección de copia por lo que mantendrá un pequeño archivo de datos para almacenar cierta información sobre cada imagen marcada: el nombre de la imagen original, el nombre del cliente para el que se marcó la imagen y el código o número de serie que se insertó como marca.

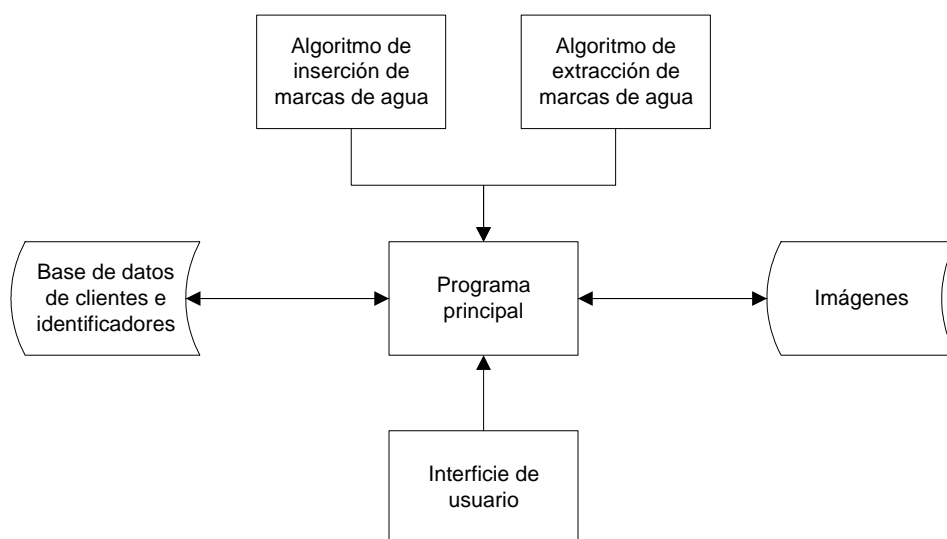


Ilustración 21. Esquema aplicación ApplMDCT

Hemos visto en el apartado dedicado al estudio del estado del arte que uno de los rasgos característicos de este tipo de aplicación es el uso de códigos especiales para la implementación de las marcas. Por su sencillez, se ha utilizado un código corto dual de un código hamming que se describirá en detalle posteriormente y que posibilitará la corrección de errores de 1 bit. El uso de códigos cortos supone una seria limitación a la utilidad de la aplicación puesto que limita la cantidad de marcas diferentes que se pueden aplicar a una imagen a un número muy bajo, no obstante el diseño de códigos más complejos queda fuera del alcance del presente trabajo.

2.2.1. Diagrama de casos de uso.

El diagrama de casos de uso nos permite ver la interacción de la aplicación con el usuario.

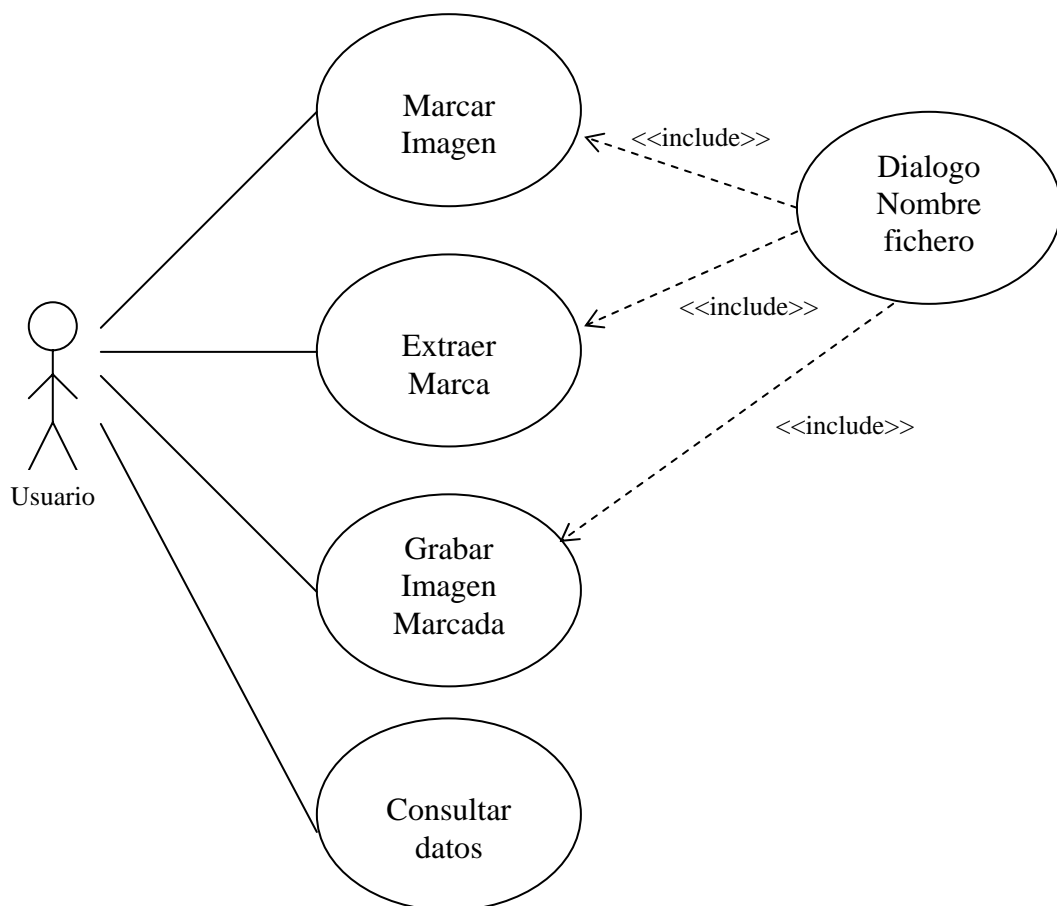


Ilustración 22. Diagrama de casos de uso ApplMDCT

Especificación textual.

Caso de uso: *Marcar Imagen*

Resumen de funcionalidad: Crear una imagen marcada con un código dado.

Papel dentro del trabajo del usuario: Básico para el usuario.

Actores: Usuario

Casos de uso relacionados: Grabar Imagen Marcada, Extraer Marca, Dialogo nombre fichero

Precondición: existe algún archivo de imagen.

Poscondición: se ha creado una imagen marcada residente en memoria.

Proceso normal principal:

1. El sistema pide al usuario que introduzca el nombre de la imagen a marcar.
2. El sistema pide al usuario que introduzca el nombre del cliente.
3. El sistema pide al usuario que introduzca el código o número de serie con el que marcar la imagen.
4. El sistema obtiene la marca física (bits) correspondientes al código a marcar.
5. El sistema genera una nueva imagen en memoria con la marca insertada.
6. El sistema presenta los resultados en pantalla.

Alternativas del proceso y excepciones:

- 1a. El usuario cancela la operación.
 - 1a1. El sistema termina.
- 2a. El usuario cancela la operación.
 - 2a1. El sistema termina.
- 3a. El usuario cancela la operación.
 - 3a1. El sistema termina.
- 3b. El usuario introduce un código incorrecto.
 - 3b1. El sistema presenta un mensaje de error y termina.

Caso de uso: *Extraer Marca*

Resumen de funcionalidad: Obtener el nombre del cliente de una imagen marcada.

Papel dentro del trabajo del usuario: Básico para el usuario.

Actores: Usuario

Casos de uso relacionados: Insertar Marca, Dialogo nombre fichero

Precondición: existe algún archivo de imagen marcada.

Poscondición: se ha obtenido el nombre del cliente para el que se marcó la imagen.

Proceso normal principal:

1. El sistema pide al usuario que introduzca el nombre de la imagen marcada a procesar.
2. El sistema pide al usuario que introduzca el nombre de la imagen original correspondiente a la imagen marcada.
3. El sistema extrae la marca física presente en la imagen.
4. El sistema obtiene a partir de la marca física el código correspondiente.
5. El sistema obtiene a partir del archivo de datos de imágenes marcadas el nombre del cliente correspondiente.
6. El sistema presenta los resultados en pantalla.

Alternativas del proceso y excepciones:

- 1a. El usuario cancela la operación.
 - 1a1. El sistema termina.
- 2a. El usuario cancela la operación.
 - 2a1. El sistema termina.
- 4a. El sistema no puede recuperar el código correspondiente a la marca.
 - 4a1. El sistema asigna un código desconocido.
- 5a. El sistema no puede recuperar el cliente.
 - 5a1. El sistema asigna un cliente desconocido.

Caso de uso: *Grabar Imagen Marcada*

Resumen de funcionalidad: Crear un archivo con una imagen marcada

Papel dentro del trabajo del usuario: Básico para el usuario.

Actores: Usuario

Casos de uso relacionados: Insertar Marca, Dialogo nombre fichero

Precondición: existe algún archivo de imagen marcada en memoria.

Poscondición: se ha creado un archivo físico con la imagen marcada.

Proceso normal principal:

1. El sistema comprueba que existe una imagen marcada en memoria.
2. El sistema pide al usuario que introduzca el nombre de la imagen a guardar.
3. El sistema crea el archivo físico con la imagen marcada.
4. El sistema añade la información de la imagen marcada al archivo de datos.
5. El sistema presenta un mensaje informativo.

Alternativas del proceso y excepciones:

- 1a. No existe una imagen marcada previamente en memoria.
 - 1a1. El sistema presenta mensaje de error y termina.
- 2a. El usuario cancela la operación.
 - 2a1. El sistema termina.

Caso de uso: *Consultar Datos*

Resumen de funcionalidad: Presenta en pantalla los datos correspondientes a las imágenes marcadas.

Papel dentro del trabajo del usuario: Ocasional para el usuario.

Actores: Usuario

Casos de uso relacionados: Grabar Imagen Marcada, Insertar Marca

Precondición: ninguna.

Poscondición: si el archivo de datos no existía se ha creado uno nuevo.

Proceso normal principal:

1. El sistema presenta en pantalla la información del archivo de datos de imágenes marcadas.

Alternativas del proceso y excepciones:

- 1a. El archivo de datos no existe.
 - 1a1. El sistema crea un nuevo archivo vacío.

Caso de uso: *Dialogo Nombre Fichero*

Resumen de funcionalidad: Obtiene del usuario un nombre de fichero

Papel dentro del trabajo del usuario: Básico para el usuario.

Actores: Usuario

Casos de uso relacionados: Grabar Imagen Marcada, Insertar Marca, Extraer Marca

Precondición: ninguna.

Poscondición: se ha obtenido el nombre de un archivo.

Proceso normal principal:

1. El sistema pide al usuario que introduzca o seleccione el nombre de un archivo.

Alternativas del proceso y excepciones:

- 1a. El usuario cancela la operación.
 - 1a1. El sistema devuelve un nombre nulo.

2.2.2. Diagramas de colaboración.

A continuación se han realizado diagramas de colaboración simplificada para los casos de uso más complejos. Mediante ellos se puede observar el comportamiento de los casos de uso en términos de intercambio de mensajes entre objetos, e identificar las principales clases de frontera, de gestión y de entidad que nos conducirán posteriormente al diagrama estático.

Caso de uso: Marcar Imagen

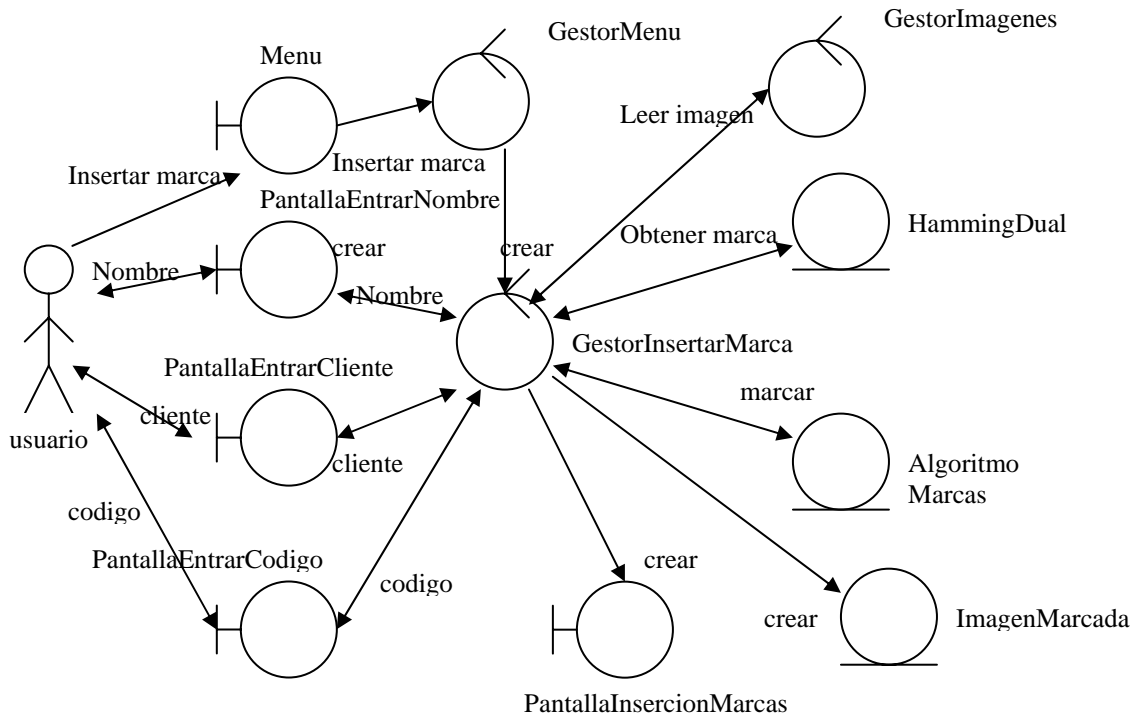


Ilustración 23. Diagrama de colaboración. Marcar Imagen

Caso de uso: *Extraer Marca*

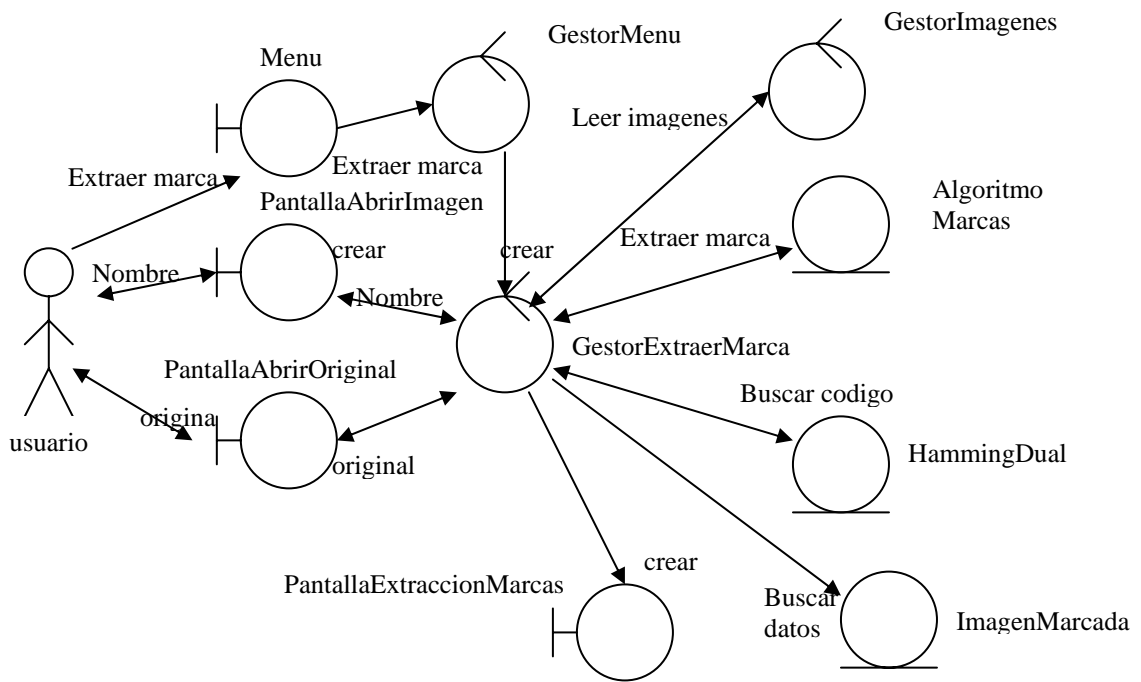


Ilustración 24. Diagrama de colaboración. Insertar Marca

Caso de uso: *Grabar Marcada*

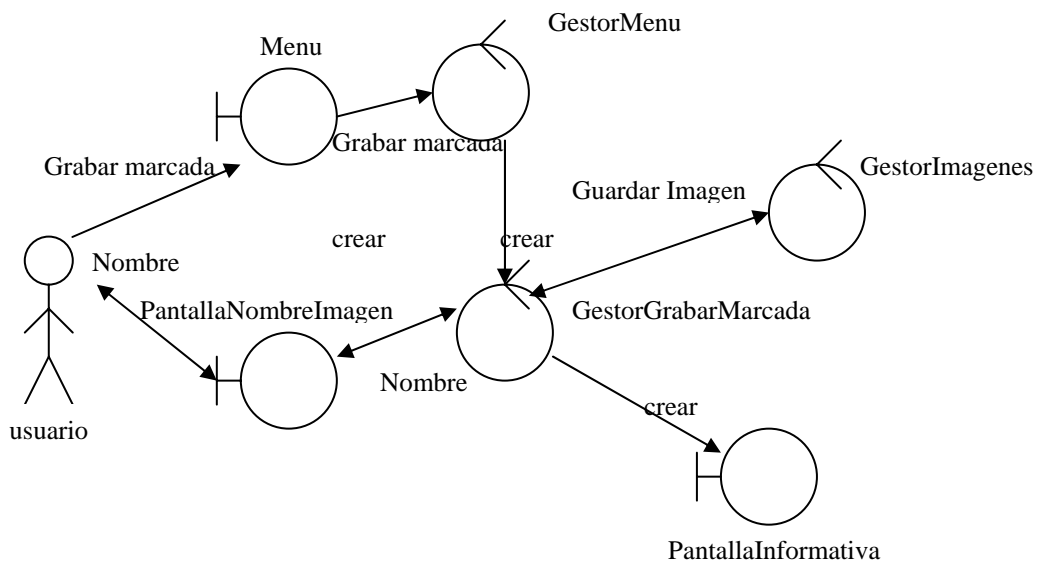
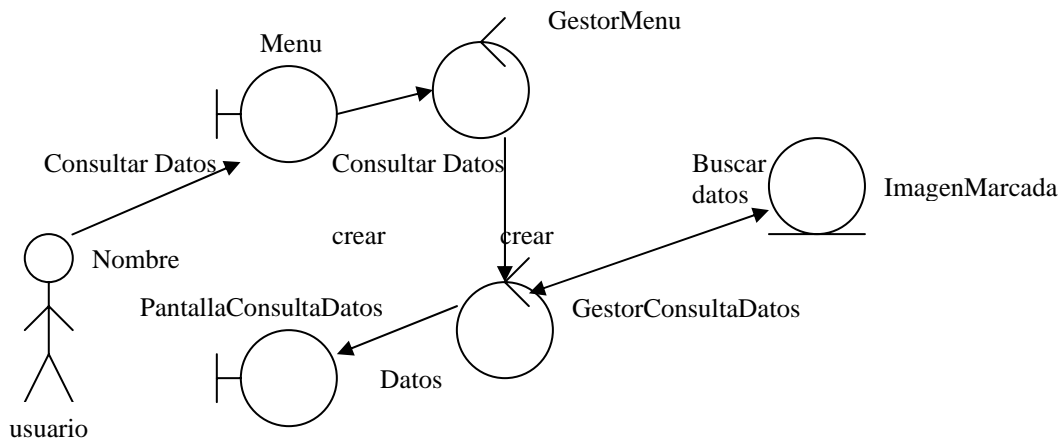


Ilustración 25. Diagrama de colaboración. Grabar Marcada

Caso de uso: Consultar Datos**Ilustración 26. Diagrama de colaboración. Consultar Datos****2.2.3. Diseño de la persistencia.**

Los únicos elementos que requieren persistencia son los concernientes a la información que debemos almacenar acerca de cada imagen marcada. Para ello se ha creado una clase que almacenará la información de cada imagen marcada. La información que guardaremos será el nombre de la imagen original, el nombre del cliente para el que se marcó la imagen, el código utilizado como número de serie y la tira de bits correspondientes a la marca física, aunque esta última no es estrictamente necesaria.

La información se almacenará en un fichero de texto simple, donde cada línea representará los datos correspondientes a una imagen marcada con el siguiente formato:

Nombre imagen original,código,nombre cliente,bits marca

Se ha utilizado una clase para gestionar las operaciones con el archivo de texto, leer el archivo y crear las instancias de imágenes marcadas, añadir nuevas instancias, formatear los datos para su presentación en pantalla, grabar las instancias en el archivo de texto etc. El archivo de texto lo creará la aplicación automáticamente en caso de no existir previamente.

2.2.4. Diseño de la interficie de usuario.

Las clases de frontera de los diagramas de colaboración conforman la interficie de usuario. Dicha interficie está formada por un marco con tres elementos básicos:

- Menú. Contiene las opciones correspondientes a los casos de uso más las opciones de salir y mostrar ayuda.
- Barra de opciones. Contiene botones para un acceso rápido a las funciones básicas (casos de uso).
- Paneles de datos. Presentan en pantalla la información resultante de las funciones básicas. Son tres paneles: panel de inserción, panel de extracción y panel de consulta.

A continuación se presenta el diseño de dichos paneles.

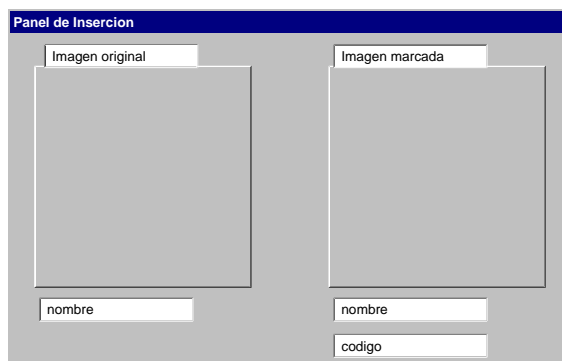


Ilustración 27. Interficie de usuario. Panel de inserción.

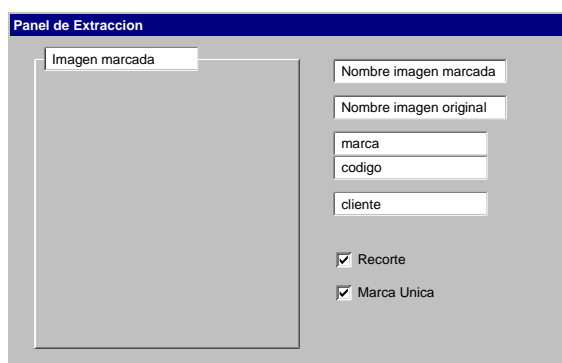


Ilustración 28. Interficie de usuario. Panel de extracción.



Ilustración 29. Interficie de usuario. Panel de consultas.

Adicionalmente existirán ventanas simples para mostrar mensajes informativos, de error y de petición de información, así como ventanas de dialogo para abrir y guardar los archivos de imágenes.

2.2.5. Diagrama estático.

A partir de los diseños anteriores se ha obtenido el siguiente diagrama estático con todas las clases que intervendrán en la aplicación.

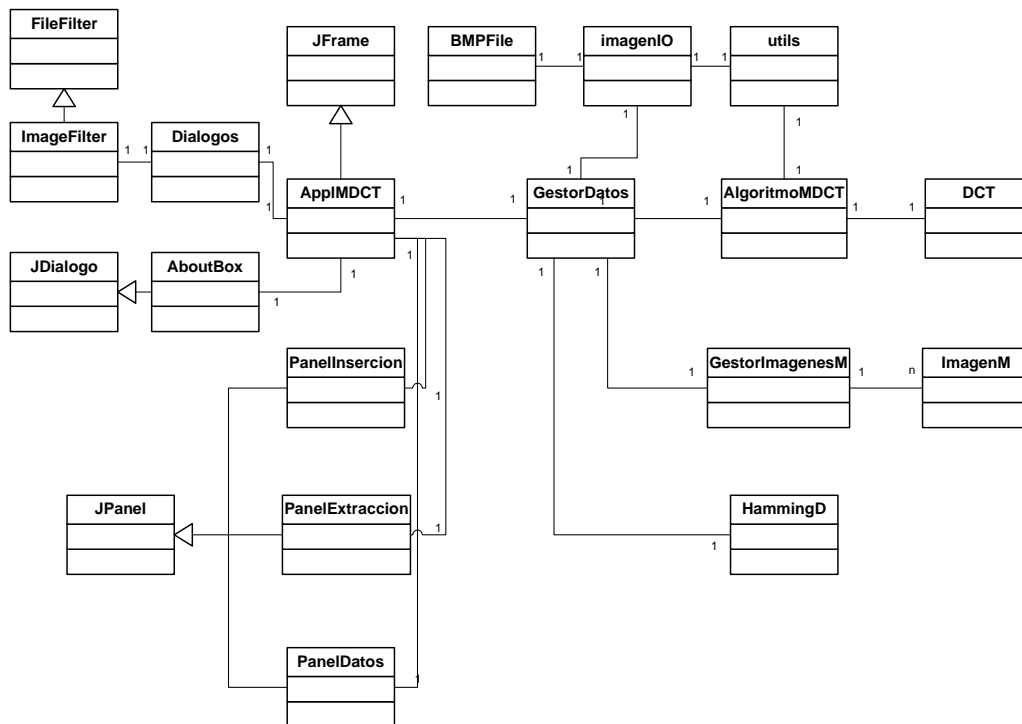


Ilustración 30. Diagrama estático ApplMDCT.

2.2.6. Código corrector de errores.

Para mejorar las prestaciones de la aplicación se utiliza un código corrector de errores para implementar las marcas físicas. Utilizaremos un código Hamming dual, que siendo un código corto presenta un buen funcionamiento como código anti-confabulaciones para el caso de confabulación de dos usuarios [1].

Un código Hamming dual es aquel cuya matriz generadora es la matriz de chequeo de paridad de un código Hamming. Una propiedad de estos códigos, importante para el caso que nos ocupa, es que la distancia entre dos palabras cualquiera del código es fija: para un código de longitud $N=2^n-1$, la distancia entre dos palabras es 2^{n-1} .

Este tipo de código permite corregir $2^{n-2}-1$ errores. La probabilidad de identificar a uno de los dos participantes en una 2-colisión es:

$$1-p \text{ donde } p \leq \left(\frac{1}{2}\right)^{2^{n-1}} 2^n \quad (\text{Eq. 2.5})$$

En una 2-colisión solamente las palabras con exactamente 2^{n-2} errores no pueden ser corregidas.

El código que se ha utilizado en el presente trabajo es el presentado a continuación. Se trata de un código Hamming dual de 7 bits, que permite usar 8 códigos diferentes para marcar cada imagen.

código	marca
0	0000000
1	0011011
2	0101101
3	0110110
4	1001110
5	1010101
6	1100011
7	1111000

Tabla 6. Código dual Hamming (7,4)

La distancia entre cada dos palabras del código es de: $2^{n-1} = 2^{3-1} = 4$ bits.

Permite corregir $2^{n-2} = 2^{3-2} = 1$ bit de error.

Dada una colisión de 2 participantes, la probabilidad de identificar a uno de ellos es:

$$\text{Probabilidad} = 1-p \text{ donde } p \leq \left(\frac{1}{2}\right)^{2n-1} \quad 2^n = \left(\frac{1}{2}\right)^{2*3-1} \quad 2^3 = 0.25 \quad (\text{Eq. 2.6})$$

luego la probabilidad es: $1-0.25=0.75$

Solamente las palabras con $2^{n-2} = 2^{3-2} = 2$ errores no pueden ser corregidas.

La matriz generadora del código, la matriz de chequeo de paridad y su transpuesta son las siguientes:

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix} \quad (\text{Eq. 2.7})$$

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (\text{Eq. 2.8})$$

$$H^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\text{Eq. 2.9})$$

Dada una palabra recibida r , se puede calcular el síndrome con la siguiente expresión:

$$s = rH^T \quad (\text{Eq. 2.10})$$

Una vez calculado el síndrome s , podemos obtener una serie de ecuaciones para calcular el vector de error e , a partir de la expresión $s = eH^T$. Para el caso de nuestro código tenemos:

$$e_0 + e_1 + e_2 + e_3 = s_0 \quad (\text{Eq. 2.11})$$

$$e_0 + e_1 + e_4 = s_1 \quad (\text{Eq. 2.12})$$

$$e_0 + e_2 + e_5 = s_2 \quad (\text{Eq. 2.13})$$

$$e_1 + e_2 + e_6 = s_3 \quad (\text{Eq. 2.14})$$

Para resolverlo, fijamos los valores de e_0, e_1, e_2 y e_3 y calculamos los valores de e_4, e_5 y e_6 , el vector de error que presente menos unos será el más probable. Si solo hay un bit de error sólo habrá un vector e con un bit 1 y podremos corregir el error. Los errores de 2 bits darán lugar a varios vectores de error e con dos bits a 1 y el error no podrá ser corregido.

2.2.7. Implementación.

La implementación en Java de la aplicación no ha presentado problemas destacados. Los cálculos de la transformada del coseno discreto se han resuelto mediante el uso de una clase externa implementada originalmente para la compresión de imágenes. Igualmente para el manejo de las imágenes se han utilizado una serie de clases externas que permiten leer los datos de las imágenes en formato bmp, jpg y gif y crear imágenes con formato bmp, dichas clases han sido modificadas para adaptarlas a las necesidades concretas del presente trabajo.

A la hora de implementar la corrección de errores se ha observado que debido a la simplicidad del código (un código muy corto) no resultaba necesario implementar todo el proceso formal descrito en el apartado anterior, sino que bastaba con una estrategia mucho más sencilla. En primer lugar se comprueba si la palabra recibida coincide directamente con una palabra del código si es así no hay errores y hemos terminado, sino es así entonces existen errores en la palabra recibida, en ese caso se altera cada vez un bit de la palabra recibida y se comprueba si coincide con alguna palabra código, si coincide con alguna se trata de un error de un bit que puede ser corregido y la palabra con la que coincide es la palabra correcta. Si no coincide con ninguna se trata de un error de más de un bit que no puede ser corregido por el código.

2.2.8. Posibles mejoras.

Para que la aplicación tenga utilidad práctica sería necesario aumentar el número de marcas diferentes que pueden insertarse para cada imagen (sólo 8 con el diseño actual). Ello pasa por la utilización de un código más largo que manteniendo unas buenas propiedades como código anti-confabulación disponga de un mayor número de marcas diferentes.

Referencias

[1] J. Domingo-Ferrer and J. Herrera-Joancomarti, “Simple collusion-secure fingerprinting schemes for images”, *Proc. Intl. Conf. Inf. Tech.: Coding & Computing 2000*, pp. 128-132, Mar. 2000.

2.3. Diseño de la aplicación de confabulaciones: ApplConfabula.

El objetivo es diseñar una aplicación sencilla que permita generar una imagen marcada a partir de la confabulación de dos versiones de la misma imagen con diferentes marcas. Este tipo de ataque a las imágenes marcadas se conoce como ataque por colisión, en este caso concreto de dos usuarios. La aplicación, que se ha denominado ApplConfabula, nos permitirá generar las imágenes confabuladas para posteriormente estudiar la robustez del código anti-confabulaciones que se ha implementado en la aplicación de marcado ApplMDCT.

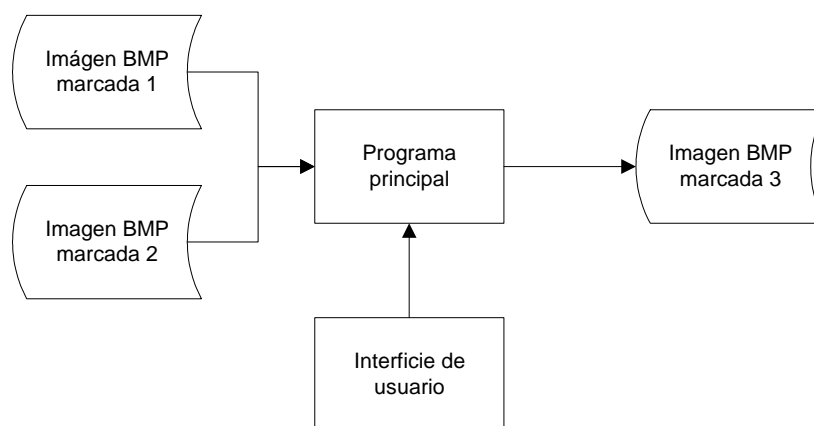


Ilustración 31. Esquema general ApplConfabula

ApplConfabula parte de la premisa que desconoce el método de marcado y la estructura de las marcas existentes en las imágenes, y se limitará a examinar dos imágenes para encontrar los píxeles que difieren en ambas y que lógicamente formarán parte de la marca de las imágenes. La imagen confabulada se generará eligiendo al azar píxeles de una imagen o de otra para todos aquellos puntos en que las imágenes difieren. La nueva imagen creada de esta forma respeta la hipótesis de *Marking Assumption* (ver apartado 1.8.1 Códigos anti-confabulación de la memoria de estado del arte). Existen otras formas de generar una imagen confabulada, por ejemplo combinando aleatoriamente porciones de las dos imágenes confabuladas, o realizando un promedio del valor de los píxeles de ambas imágenes, sin embargo el método elegido es presumiblemente el más “destrutivo” con respecto a las imágenes confabuladas.

2.3.1. Diagrama de casos de uso.

El diagrama de casos de uso que nos permite ver la interacción de la aplicación con el usuario es en este caso muy sencillo.

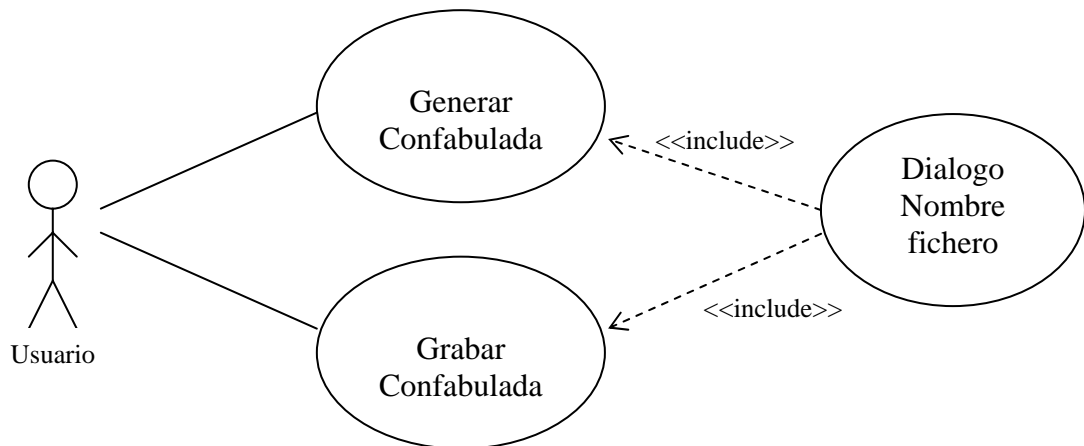


Ilustración 32. Diagrama de casos de uso ApplConfabula.

Especificación textual.

Caso de uso: *Generar Confabulada*

Resumen de funcionalidad: Crear una imagen confabulada a partir de dos versiones marcadas de la misma imagen.

Papel dentro del trabajo del usuario: Básico para el usuario.

Actores: Usuario

Casos de uso relacionados: Grabar Confabulada, Dialogo nombre fichero

Precondición: existe dos versiones de la misma imagen con diferentes marcas.

Poscondición: se ha creado una imagen confabulada a partir de las dos marcadas.

Proceso normal principal:

1. El sistema pide al usuario que introduzca el nombre de la primera imagen.
2. El sistema pide al usuario que introduzca el nombre de la segunda imagen.
3. El sistema genera una imagen mostrando las diferencias de ambas.
4. El sistema genera una nueva imagen en memoria mediante confabulación.
5. El sistema presenta los resultados en pantalla.

Alternativas del proceso y excepciones:

- 1a. El usuario cancela la operación.
 - 1a1. El sistema termina.
- 2a. El usuario cancela la operación.
 - 2a1. El sistema termina.
- 3a. El sistema no puede leer alguna de las imágenes
 - 3a1. El sistema presenta un mensaje de error y termina.
- 3b. Las imágenes no son del mismo tamaño
 - 3b1. El sistema presenta un mensaje de error y termina.

Caso de uso: *Grabar Confabulada*

Resumen de funcionalidad: Crear un archivo con una imagen confabulada

Papel dentro del trabajo del usuario: Básico para el usuario.

Actores: Usuario

Casos de uso relacionados: Generar Confabulada, Dialogo nombre fichero

Precondición: existe algún archivo de imagen confabulada en memoria.

Poscondición: se ha creado un archivo físico con la imagen confabulada.

Proceso normal principal:

1. El sistema comprueba que existe una imagen confabulada en memoria.
2. El sistema pide al usuario que introduzca el nombre de la imagen a guardar.
3. El sistema crea el archivo físico con la imagen marcada.
4. El sistema presenta un mensaje informativo.

Alternativas del proceso y excepciones:

- 1a. No existe una imagen confabulada previamente en memoria.
 - 1a1. El sistema presenta mensaje de error y termina.
- 2a. El usuario cancela la operación.
 - 2a1. El sistema termina.

2.3.2. Diagramas de colaboración.

A continuación se han realizado diagramas de colaboración simplificada para ambos casos de uso. Mediante ellos observaremos el comportamiento de los casos de uso en términos de intercambio de mensajes entre objetos, y nos permitirán identificar las principales clases de frontera, de gestión y de entidad que nos conducirán posteriormente al diagrama estático.

Caso de uso: Generar Confabulada.

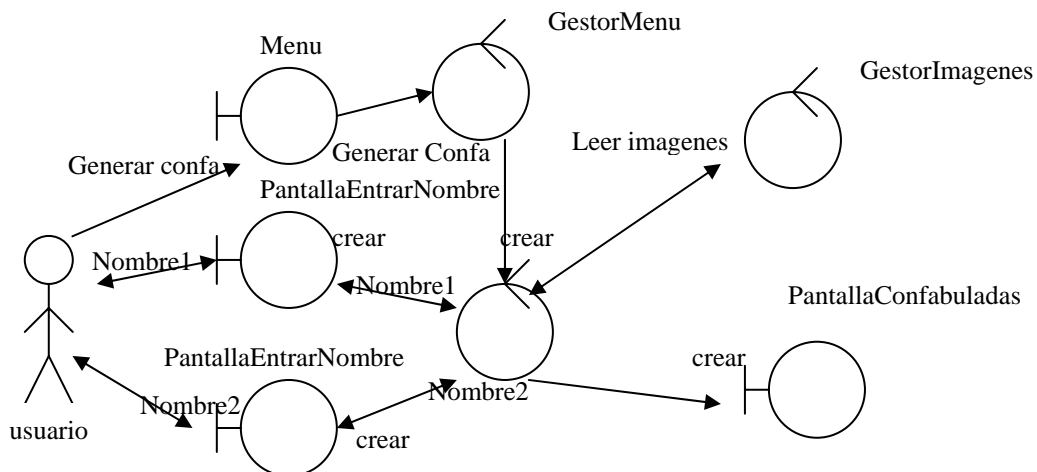
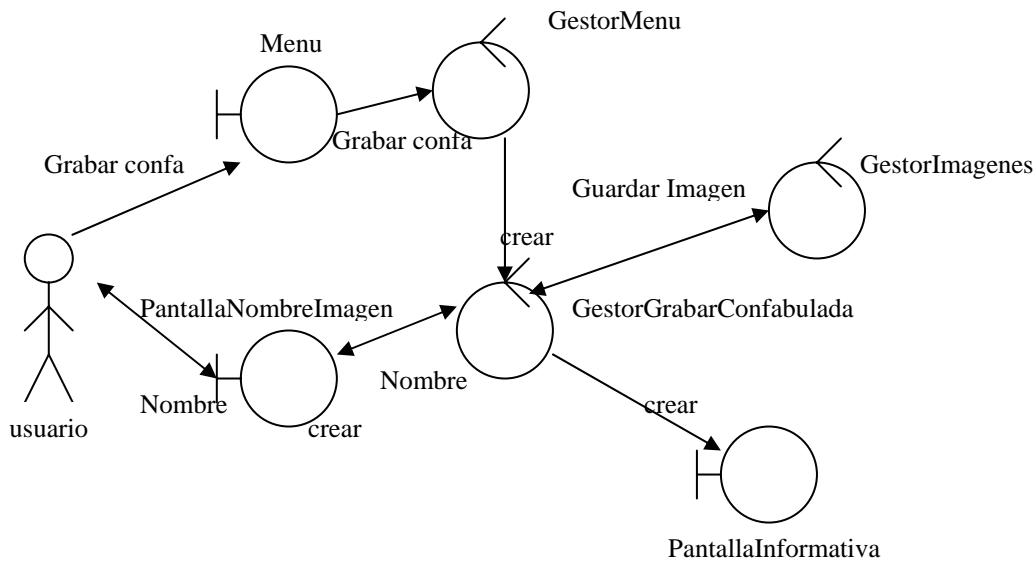
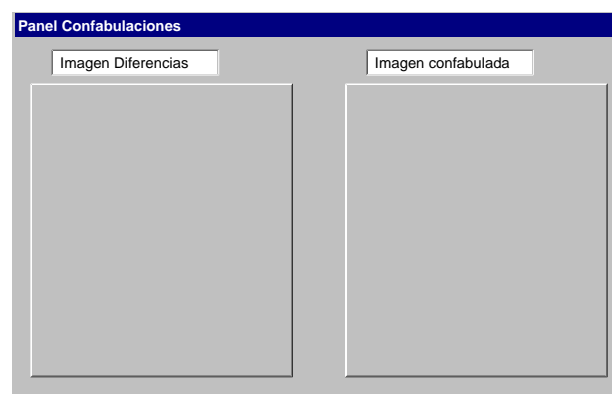


Ilustración 33. Diagrama de colaboración. Generar Confabulada.

Caso de uso: Grabar Confabulada.**Il·lustració 34. Diagrama de col·laboració. Grabar Confabulada.****2.3.3. Disseny de la interfície de usuari.**

La interfície de usuari és molt simple. A partir de les classes de frontera de los diagramas de col·laboració se ha dissenyat dicha interfície. Esta formada por un marco con tres elementos básicos:

- Menú. Contiene las opciones correspondientes a los casos de uso más las opciones de salir y mostrar ayuda.
- Barra de opciones. Contiene botones para un acceso rápido a las funciones básicas (casos de uso).
- Panel de datos. Presenta en pantalla la información resultante de las funciones básicas. A continuación se presenta el diseño de dicho panel.

**Il·lustració 35. Interfície de usuari. Panel ApplConfabula.**

Adicionalment existirán ventanes simples para mostrar mensajes informativos, de error y de petición de información, así como ventanes de diálogo para abrir y guardar los archivos de imágenes.

2.3.4. Diagrama estático.

A partir de los diagramas y especificaciones anteriores se ha obtenido el diagrama estático donde aparecen las clases que conformarán la aplicación.

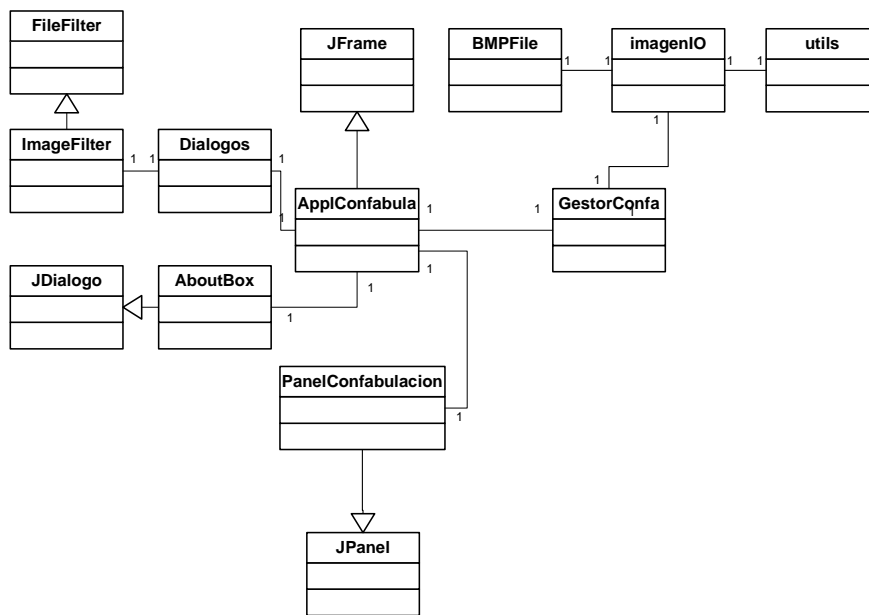


Ilustración 36. Diagrama estático ApplConfabula.

2.3.5. Implementación.

La implementación en Java de la aplicación no ha presentado problemas destacados. La estructura de la aplicación, similar a la anterior, ha posibilitado la reutilización de clases, lo que ha facilitado la implementación.

2.3.6. Pruebas con imágenes confabuladas.

Hemos utilizado la aplicación diseñada para generar varias imágenes confabuladas y posteriormente extraer la marca presente en las imágenes con la aplicación ApplMDCT, comprobando el nivel de supervivencia del código usado a un ataque por colisión de 2 usuarios. Las imágenes originales utilizadas son las mismas usadas en las pruebas de robustez del método de marcado y son las siguientes:

Img1	- lena.bmp	512x512 24-bits color
Img2	- corredor.jpg	283x212 24-bits color
Img3	- logo_uoc.bmp	150x38 24-bits color
Img4	- steve.jpg	320x240 8-bits escala de grises

Cada una de ellas se ha marcado con diferentes valores de un código corrector de errores (ver apartado 2.2.6 Código corrector de errores de la memoria de diseño de la aplicación de marcas), se han agrupado por parejas y se han generado 10 imágenes

confabuladas para cada pareja. Finalmente se ha probado de extraer la marca presente en cada imagen confabulada. Los resultados obtenidos se reflejan en la siguiente tabla.

Imagen	Código A	Código B	Nº códigos nulos extraídos	Nº códigos A extraídos	Nº códigos B extraídos
lena.bmp	1	5	2	4	4
	4	7	5	3	2
	1	4	7	2	1
corredor.jpg	2	6	3	6	1
	1	4	4	2	4
	3	7	4	4	2
Steve.jpg	2	3	4	4	2
	1	6	3	4	3
	4	5	3	3	4
Logo_uoc.bmp	2	7	6	1	3
Total			41	33	26
Porcentaje			Nulos: 41%	Válidos: 59%	

Tabla 7. Resultados pruebas confabulación.

Si recordamos lo que se dijo en la memoria de diseño de la aplicación de marcas (apartado 2.2.6) para implementar las marcas físicas utilizamos un código corrector de errores (ECC). Para dicho ECC y bajo la hipótesis conocida como *marking assumption* (en un ataque por confabulación de dos imágenes sólo podrán resultar alterados los bits de las marcas que difieran para ambas imágenes y quedarán intactos los que coincidan) la probabilidad de recuperar el código de uno de los atacantes es de un 75%. La aplicación de confabulaciones respeta la hipótesis de *marking assumption* no obstante los resultados obtenidos son algo inferiores a los resultados teóricos.

Conclusiones: Podemos concluir que el código corrector de errores utilizado (dual de un Hamming (7,4)), que a priori presenta un nivel de protección frente a confabulaciones de dos usuarios bueno, obtiene un resultado solo correcto en combinación con el método de marcado diseñado.

Como curiosidad podemos observar lo que sucede al intentar extraer la marca de una imagen confabulada. Confabulación de lena.bmp con los códigos 1 y 4:

Código 1	0011011	
Código 4	1001110	
Bits modificables	X0X1X1X	
Marca extraída	0001111	Código inválido (2 errores)

Detalles de la proporción obtenida para cada bit en una de las imágenes confabuladas:

Bit 0	valores 0: 167	valores 1: 156
Bit 1	valores 0: 435	valores 1: 0
Bit 2	valores 0: 162	valores 1: 150
Bit 3	valores 0: 0	valores 1: 435
Bit 4	valores 0: 159	valores 1: 161
Bit 5	valores 0: 0	valores 1: 434
Bit 6	valores 0: 158	valores 1: 164

Podemos ver que en la imagen se hallan 435 marcas, sin embargo para los bits que existe colisión el número de marcas recuperadas es inferior, entre 312 y 323, lo que significa que existe un número considerable de marcas para las que no se ha podido recuperar algún bit de la marca, como mínimo entre 112 y 132. Para que un bit sea indefinido es necesario que los dos coeficientes utilizados para almacenar el bit (ver diseño de la técnica de marcado) sean iguales. Cada bit de la marca se “almacena” en un bloque de 8x8 píxeles, modifica por tanto presumiblemente 64 píxeles. La generación de un nuevo bloque de 8x8 a partir de la elección aleatoria de píxeles de un bloque marcado con un cero y un bloque marcado con un uno (se utiliza una función pseudo aleatoria uniformemente distribuida) produce curiosamente con bastante frecuencia un bloque con los dos coeficientes mencionados iguales.

2.3.7. Mejoras de los resultados.

Un método que debería llegar al 100% de recuperaciones con éxito consistiría en ir recuperando una a una cada marca presente en la imagen hasta dar con una marca válida que casi con total seguridad siempre existirá en la imagen, todo ello siempre bajo la hipótesis conocida como *marking assumption*. El problema que presenta es que si no se respeta dicha hipótesis se podría fácilmente atribuir la culpabilidad a un código inocente.

En este mismo sentido, un experimento que resulta sencillo de implementar es el de extraer una única marca de cada imagen. Podemos combinar la extracción normal con éste último caso de forma que si falla el primero tengamos la opción de recurrir al segundo. Para ello se han modificado los diseños originales del método de marcado y de la aplicación ApplMDCT de forma que esta última incluya una casilla de verificación para indicar en el proceso de extracción que solo deseamos extraer una marca, la primera que encuentre. A continuación se han combinado el método normal y esta opción a la extracción de la marca para las mismas imágenes confabuladas usadas en el apartado anterior. Los resultados obtenidos son los siguientes:

Imagen	Código A	Código B	Nº códigos nulos extraídos	Nº códigos A extraídos	Nº códigos B extraídos
lena.bmp	1	5	2	4	4
	4	7	5	3	2
	1	4	7	2	1
Corredor.jpg	2	6	2	6	2
	1	4	4	2	4
	3	7	2	5	3
Steve.jpg	2	3	3	5	2
	1	6	3	4	3
	4	5	1	3	6
Logo_uoc.bmp	2	7	6	1	3
Total			35	35	30
Porcentaje			Nulos: 35%	Válidos: 65%	

Tabla 8. Resultados método combinado.

Como puede verse los resultados han experimentado sólo una pequeña mejora del 6%.

Evidentemente otra forma de mejorar los resultados sería la utilización de un código anti-confabulación con mejores prestaciones que el utilizado. A este respecto sería por ejemplo interesante comprobar los resultados de utilizar un código dual de un Hamming (15,11), sin embargo no se ha podido experimentar con el uso de estos códigos más complejos debido a la estricta temporización del trabajo.

3. CONCLUSIONES

En la primera parte de este trabajo se ha realizado un estudio del estado del arte del uso de marcas de agua digitales para aplicaciones de protección de los derechos de autor. Se ha visto que los sistemas de protección basados en métodos preventivos no han resultado eficaces hasta la fecha y si bien los métodos disuasorios como son las marcas de agua tampoco son actualmente 100% eficaces si constituyen un camino con un futuro prometedor.

Se han aclarado los principales conceptos acerca de las marcas de agua digitales, su tipología, sus aplicaciones y sus problemas de uso, incidiendo especialmente en el caso de las imágenes digitales uno de los elementos que sufre en mayor grado el problema de los derechos de autor debido principalmente a su extensa utilización a través de Internet. Hemos visto que para la protección de los derechos de autor de las imágenes digitales se utilizan marcas de agua invisibles y robustas y hemos constatado que existen muy diferentes técnicas de marcado y que se encuentran en constante evolución.

Ha quedado claro que la principal característica a tener en cuenta a la hora de utilizar las marcas de agua para la protección de los derechos de autor es la robustez de la técnica de marcado utilizada. Robustez tanto a las manipulaciones “naturales” que pueden suceder durante el uso de una imagen como a los diversos tipos de “ataques” intencionados que puede llegar a sufrir. La robustez está en conflicto directo con la invisibilidad y la capacidad de almacenamiento de información por lo que siempre resulta necesario hallar un punto de equilibrio entre ellas teniendo en cuenta los requisitos de la aplicación a la que se destina. Se ha visto que hasta el momento no existen técnicas robustas a todas las posibles manipulaciones y/o ataques por lo que nuevamente en función de la aplicación concreta a la que se destina deberá elegirse la que resulte más adecuada.

Finalmente hemos estudiado una aplicación concreta de las marcas de agua digitales: la protección del copyright mediante detección de copia, en cada imagen se introduce una marca única que la identifica de forma inequívoca (huella digital) y que permite establecer el origen de cualquier distribución no autorizada. Hemos visto que esta aplicación se apoya en dos pilares fundamentales: primero el uso de una técnica de marcado robusta y segundo el uso de marcas con una estructura física especialmente diseñada para resistir la mayor amenaza a este tipo de aplicación, el ataque por colisión. Y es en el diseño de estas marcas especiales o códigos anti-colisión donde radica la mayor dificultad. No existen códigos totalmente seguros contra dos o más confabulados, pero si pueden diseñarse bajo la hipótesis conocida como Marking Assumption, códigos capaces de identificar a uno de los confabulados con una probabilidad tan cercana al 100% como deseemos a costa de incrementar la longitud del código. El diseño de códigos de este tipo cortos es hasta el momento una utopía.

En la segunda parte del trabajo se han utilizado los conocimientos adquiridos previamente para diseñar e implementar una aplicación sencilla de protección de los derechos de autor mediante detección de copia.

Hemos comprobado que mediante el auxilio de una clase para realizar cálculos de la transformada del coseno discreto (DCT) resulta fácil diseñar e implementar una técnica de marcado sencilla basada en la relación entre dos coeficientes DCT de la luminancia de la imagen. Se ha experimentado con los límites de dicha relación para encontrar un adecuado equilibrio entre robustez e invisibilidad teniendo en cuenta que los resultados obtenidos son dependientes de las imágenes utilizadas en las pruebas. Hemos constatado que para un correcto funcionamiento de la extracción de la marca resulta necesario la utilización de la imagen original y se ha explicado por que. Finalmente se han sometido a varias imágenes a un conjunto de pruebas para comprobar la robustez de la técnica de marcado obteniendo unos buenos resultados ante la mayor parte de las manipulaciones, especialmente ante la compresión Jpeg llegando a soportar hasta el 40%. El principal inconveniente de la técnica desarrollada radica en su baja capacidad de almacenamiento de información, 1 bit por cada bloque de 64 píxeles y aproximadamente algo más de la mitad de los bloques de la imagen válidos para marcado, por lo que se han propuesto algunas alternativas para aumentar la capacidad, como disponer de una lista de varias parejas de coeficientes elegibles para el marcado lo que aumentaría el número de bloques válidos para marcado, o incluso utilizar varias de estas parejas simultáneamente con lo que se incrementaría el número de bits de la marca en cada bloque válido, no obstante debería realizarse previamente un estudio de los efectos sobre la invisibilidad de la marca de estas variaciones. También se ha constatado que las zonas de color uniforme no resultan aptas para el marcado, por lo que la técnica tiene dificultades con imágenes con colores uniformes como los logos, especialmente si son de pequeño tamaño.

Utilizando la técnica de marcado descrita se ha diseñado una aplicación sencilla que permite insertar y recuperar las marcas de las imágenes guardando la información relacionada con las imágenes marcadas. Se ha considerado importante dotar a la aplicación de una interficie de usuario gráfica que permita un funcionamiento intuitivo y agradable. Como ya hemos visto, este tipo de aplicaciones se fundamentan en dos pilares básicos, el uso de una técnica de marcado robusta y de una estructura física de las marcas adecuada. Con respecto a este segundo aspecto se ha utilizado para implementar las marcas un código dual de un Hamming(7,4), en esta elección a primado por encima de otras consideraciones la sencillez de implementación de un código corto como es el caso, considerando que el objetivo es más la experimentación con los conceptos del marcado digital que no la utilidad práctica real de la aplicación. El código utilizado permite únicamente 8 marcas diferentes por lo que tiene poca utilidad real. No obstante la aplicación con la única mejora de implementar un código que permita un mayor número de marcas diferentes es perfectamente válida para un uso práctico.

Finalmente se ha estudiado la robustez de la aplicación ante su mayor enemigo los ataques por colisión, en concreto ante la colisión de 2 confabuladores. Para ello ha sido necesario diseñar una pequeña aplicación que permita generar las imágenes confabuladas a partir de la colisión de dos versiones de la misma imagen con diferentes marcas. De entre los posibles métodos para generar una imagen confabulada se ha optado por implementar uno basado en la elección aleatoria de píxeles individuales procedentes de una imagen u otra por considerar que podía resultar de los más

“dañinos” para la marca con independencia de la técnica de marcado utilizada. A este respecto se ha experimentado con otro método consistente en formar la nueva imagen a partir de bloques de 5 líneas elegidos de forma aleatoria de las imágenes confabuladas constatándose que los resultados son más favorables que con el método anterior. Los resultados finales obtenidos son algo inferiores a los valores teóricos del código utilizado aunque dentro de lo correcto (aproximadamente un 59% de éxito) de lo que podemos inferir que la técnica de marcado utilizada tiene cierto efecto negativo sobre las propiedades del código empleado. Como posibles mejoras se propone un método que teóricamente permitiría un éxito cercano al 100%, a su vez no exento de problemas. Una alternativa sencilla que se ha implementado consiste en recuperar una única marca, la primera presente en la imagen y confiar en que sea una marca válida. Combinando el funcionamiento normal y esta última opción se ha incrementado el porcentaje de éxitos hasta el 65%. Como curiosidad podemos mencionar que la aplicación dispone también de interficie gráfica de usuario que permite visualizar una imagen con las diferencias entre las imágenes confabuladas, así en el caso que nos ocupa, podemos aun desconociendo la técnica de marcado utilizada, ver que ésta se basa en bloques de pequeño tamaño que se observan claramente en la imagen de diferencias.

GLOSARIO DE TÉRMINOS Y ACRÓNIMOS

<i>BMP</i>	- Bitmap, formato gráfico de Microsoft usado para almacenar imágenes en forma de fichero.
<i>Copyright</i>	- Derechos de autor.
<i>DCT</i>	- Discrete Cosine Transform. Transformada del Coseno Discreto. Usada por algoritmos de compresión como el Jpeg.
<i>ECC</i>	- Error Correction Code. Código corrector de errores.
<i>Fingerprinting</i>	- Marcado digital con marcas únicas (huellas dactilares) que identifican inequívocamente un producto.
<i>gif</i>	- Formato gráfico de imágenes.
<i>Jpeg</i>	- Joint Photographic Experts Group. Algoritmo de compresión de imágenes.
<i>Jpg</i>	- Formato gráfico de imágenes.
<i>LSB</i>	- Least Significant Bit. Bit menos significativo.
<i>Píxel</i>	- Picture element. Unidad básica de color programable en una pantalla de ordenador o en una imagen.
<i>RGB</i>	- Espacio de representación de los colores en tres ejes rojo, verde y azul.
<i>Watermarking</i>	- Proceso de utilización de las marcas de agua para el marcado de objetos multimedia.

BIBLIOGRAFIA

Libros consultados:

- *Information Hiding Techniques for Steganography and Digital Watermarking*
Stefan Katzenbeisser, Fabien A.P. Petitcolas (Editores)
Artech House, Ene. 2000, ISBN: 1580530354
- *Techniques and applications of digital watermarking and content protection*
Michael Arnold, Martin Schmucker, Stephen D. Wolthusen.
Artech House, 2003, ISBN: 1-58053-111-3

Recursos web:

- Manejo de los formatos gráficos bmp, jpeg, gif, en Java
<http://www.cs.queensu.ca/home/blostein/java.html>
- Cálculos DCT transformada del coseno discreto en Java
<http://eagle.uccb.ns.ca/steve/home.html>
- Formato gráfico Jpeg
<http://www.cs.sfu.ca/CourseCentral/365/li/material/notes/Chap4/Chap4.2/Chap4.2.html>
- Formatos gráficos
<http://www.dcs.ed.ac.uk/home/mxr/gfx/2d-hi.html>
- Watermarking
<http://www.jjtc.com/Steganography/>
- Informes sobre protección de la propiedad intelectual
<http://citeseer.ist.psu.edu/Security/IntellectualPropertyProtection/>

ANEXOS

A. Manuales de usuario.

A.1 Aplicación ApplMDCT.

Requerimientos.

Para ejecutar la aplicación se requiere tener instalado Java, a ser posible versión 1.4 o superior. Resulta recomendable una resolución de pantalla mínima de 1024x768.

Compilación y ejecución.

Para compilar la aplicación ejecutar la siguiente sentencia, en el directorio donde se hallen todas las clases:

```
>javac ApplMDCT.java
```

Para iniciar la aplicación ejecutar la siguiente sentencia:

```
>java ApplMDCT
```

Es necesario que todas las clases se hallan compilado previamente, también es necesario que se encuentren en el mismo directorio los siguientes archivos:

consultar.gif , extraer.gif, guardar.gif, marcar.gif

Funcionamiento.

Las funciones que permite realizar la aplicación pueden ser ejecutadas bien desde los menús de la aplicación o bien desde la barra de herramientas. La barra de herramientas dispone de cuatro botones para realizar las operaciones básicas de la aplicación, de izquierda a derecha: insertar una marca en una imagen, grabar una imagen marcada, extraer la marca de una imagen marcada y consultar los datos de imágenes marcadas. Los menús de la aplicación permiten además salir de la aplicación y mostrar la ventana de información.

Los resultados de las operaciones realizadas se presentan en tres pestañas diferentes: insertar marcas, extraer marcas y consultar datos. Al realizar cualquier operación se cambia el foco automáticamente a la pestaña sobre la que aparecerán los resultados de dicha operación.

La forma de realizar las funciones básicas es la siguiente:

- Insertar una marca en una imagen.
 1. La aplicación presenta una ventana de dialogo para elegir la imagen a marcar, por defecto aparecen únicamente las imágenes cuyo formato es soportado por la aplicación: bmp, jpg y gif. El formato preferido es el

bmp, si una imagen no puede ser abierta por la aplicación se recomienda convertirla previamente a dicho formato.

2. Una vez elegida la imagen debemos introducir el nombre que identificará al cliente para el que se marca la imagen.
3. A continuación debemos introducir el código con el que se marcará la imagen (códigos válidos: un número de 0 a 7).
4. Si el proceso ha sido correcto en la pestaña de insertar marcas se visualizarán la imagen original y la imagen marcada así como el nombre del cliente y el código de la marca para esta última. Si deseamos conservar la imagen marcada deberá utilizarse la opción de grabar una imagen marcada.

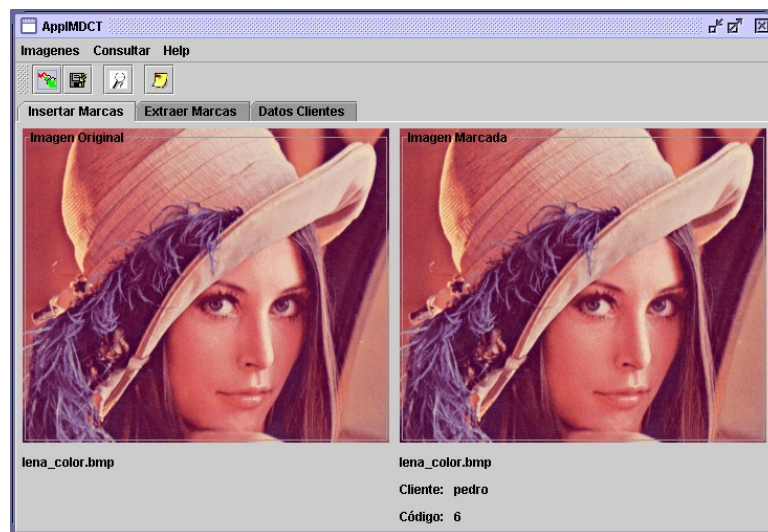


Ilustración 37. ApplMDCT: Marcar Imagen

- Grabar una imagen marcada.
 1. Esta opción debe utilizarse cuando existe una imagen marcada previamente y deseamos conservarla.
 2. Aparece un cuadro de dialogo para introducir el nombre para la imagen a grabar. El único formato soportado para grabar imágenes es el bmp. Al introducir el nombre, la aplicación añade la extensión .bmp si ésta no se introduce.
 3. Si todo ha funcionado correctamente aparecerá un mensaje para indicar que la imagen ha sido grabada y la imagen marcada desaparecerá de la pestaña de insertar marcas puesto que sólo puede existir una única imagen marcada con un código dado y ya ha sido grabada.
 4. En este momento los datos correspondientes a la imagen marcada se añaden a la base de datos de imágenes marcadas y pueden ser consultados en la pestaña correspondiente.
- Extraer la marca de una imagen marcada.
 1. En primer lugar aparecerá una ventana de dialogo para elegir la imagen marcada de la que queremos extraer la marca.

2. A continuació se mostra un missatge que nos indica que hem de introduir el nom de la imatge original corresponent a la imatge marcada.
3. Apareix novament una finestra de diàleg per introduir el nom de la imatge original. L'aplicació no realitza cap comprovació sobre la imatge original per lo que si es introdueix una imatge que no correspon el resultat de la marca obtinguda no correspondrà amb la marca real de la imatge marcada.
4. Si tot ha anat correctament a la pestanya d'extraer marques es visualitzarà la imatge marcada juntament amb la següent informació: nom de la imatge marcada i de la imatge original, bits de la marca recuperada, codi corresponent si existeix o -1 si la marca no correspon a un codi vàlid, nom del client pel qual existeixen dades a la base de dades d'imatges marcades amb aquesta imatge original i el codi recuperat, si no existeix es mostra la cadena "Desconocido".

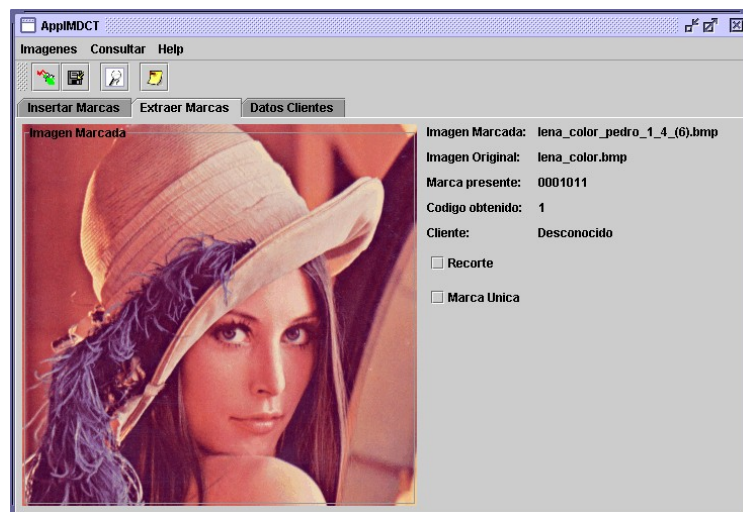


Ilustración 38. ApplMDCT: Extraer marca

Existen dos casos especiales que se comentan a continuación:

Extracción de la marca de un recorte de imagen marcada: Podemos intentar extraer la marca de un recorte de imagen, para ello primero, en la pestaña de extraer marcas debemos seleccionar la casilla de recorte, después procedemos normalmente con la opción de extraer marca. El proceso es el mismo que en el caso general con la única diferencia que tras seleccionar el nombre de la imagen marcada se nos pide que introduzcamos las coordenadas de la esquina superior izquierda del recorte con respecto a la imagen original. Estas pueden ser obtenidas fácilmente comprando el recorte con la imagen original en cualquier programa de manipulación de imágenes.

Extracción de una única marca. Si en la pestaña de extraer marcas seleccionamos la casilla Marca Unica, al elegir la opción de extraer marca, la aplicación extraerá únicamente la primera marca presente en la imagen, es decir, si la marca tiene una longitud de 7 bits, se extraerán los 7 primeros bits con independencia de que puedan ser indefinidos, en cuyo caso se mostrarían con un -1 en su lugar correspondiente. Esta es una opción experimental que puede resultar útil en algunos casos, por ejemplo si ha fallado la extracción de la marca de una imagen confabulada podemos intentar extraer la primera marca por si fuese correcta.

- Consultar datos de clientes.
 1. Si elegimos esta opción en la pestaña de Datos Clientes se mostrará la base de datos de las imágenes marcadas.

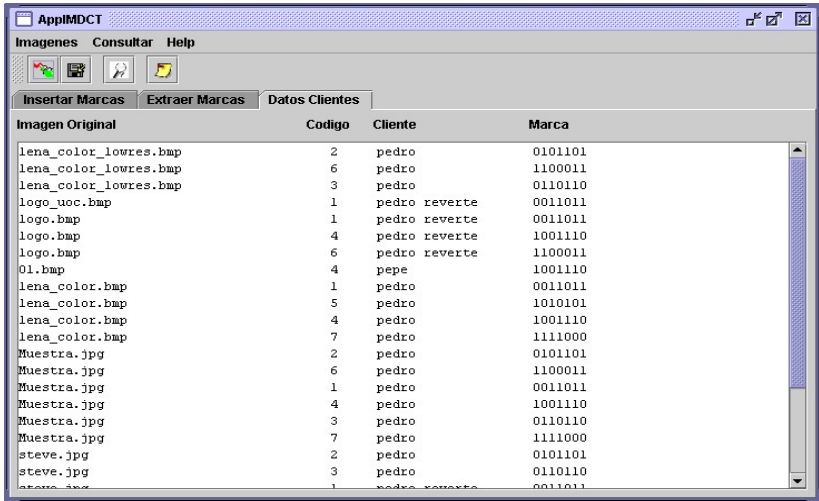


Imagen Original	Codigo	Cliente	Marca
lena_color_lowres.bmp	2	pedro	0101101
lena_color_lowres.bmp	6	pedro	1100011
lena_color_lowres.bmp	3	pedro	0110110
logo_uoc.bmp	1	pedro revertte	0011011
logo.bmp	1	pedro revertte	0011011
logo.bmp	4	pedro revertte	1001110
logo.bmp	6	pedro revertte	1100011
01.bmp	4	pepe	1001110
lena_color.bmp	1	pedro	0011011
lena_color.bmp	5	pedro	1010101
lena_color.bmp	4	pedro	1001110
lena_color.bmp	7	pedro	1111000
Muestra.jpg	2	pedro	0101101
Muestra.jpg	6	pedro	1100011
Muestra.jpg	1	pedro	0011011
Muestra.jpg	4	pedro	1001110
Muestra.jpg	3	pedro	0110110
Muestra.jpg	7	pedro	1111000
steve.jpg	2	pedro	0101101
steve.jpg	3	pedro	0110110
steve.jpg	1	pedro revertte	0011011

Ilustración 39. ApplMDCT: Consultar Datos

A.2 Aplicación ApplConfabula.

Requerimientos.

Para ejecutar la aplicación se requiere tener instalado Java, a ser posible versión 1.4 o superior. Resulta recomendable una resolución de pantalla mínima de 1024x768.

Compilación y ejecución.

Para compilar la aplicación ejecutar la siguiente sentencia, en el directorio donde se hallan todas las clases:

```
>javac ApplConfabula.java
```

Para iniciar la aplicación ejecutar la siguiente sentencia:

```
>java ApplConfabula
```

Es necesario que todas las clases se hallan compilado previamente, también es necesario que se encuentren en el mismo directorio los siguientes archivos:

guardar.gif, marcar.gif

Funcionamiento.

Las funciones que permite realizar la aplicación pueden ser ejecutadas bien desde los menús de la aplicación o bien desde la barra de herramientas. La barra de herramientas dispone de dos botones para realizar las operaciones básicas de la

aplicación, de izquierda a derecha: generar una imagen confabulada y grabar una imagen confabulada. Los menús de la aplicación permiten además salir de la aplicación y mostrar la ventana de información.

La forma de realizar las funciones básicas es la siguiente:

- Generar una imagen confabulada.
 1. La aplicación presenta una ventana para informar que debemos seleccionar los nombres de las dos imágenes a confabular.
 2. La aplicación presenta una ventana de dialogo para elegir el nombre de la primera imagen para la confabulación.
 3. A continuación aparece nuevamente la ventana de dialogo para elegir el nombre de la segunda imagen de la confabulación. La única comprobación que se realiza de que se trata de la misma imagen es comprobar su tamaño, si no coincide se presenta una mensaje de error.
 4. Si el proceso ha sido correcto se presenta por una parte la imagen resultado de la confabulación y por otra una imagen que muestra las diferencias entre las dos imágenes confabuladas. Si un píxel es igual en ambas imágenes se muestra de color negro, si es diferente se muestra de color blanco.

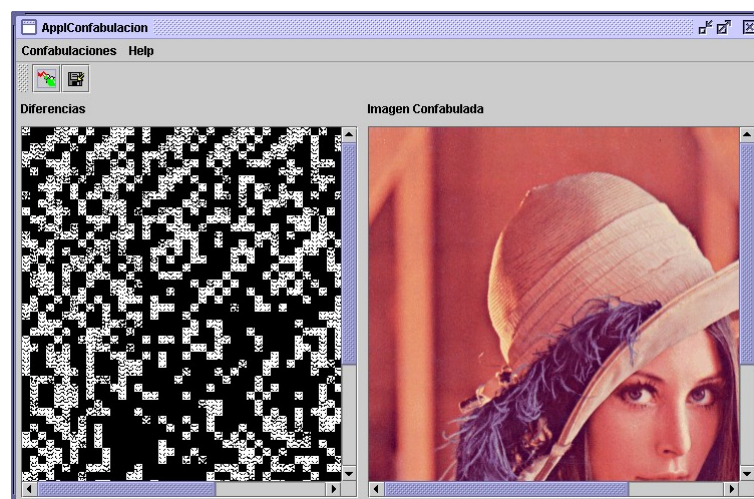


Ilustración 40. ApplConfabula: Generar confabulada

- Grabar una imagen confabulada.
 1. Esta opción debe utilizarse cuando existe una imagen confabulada creada previamente y deseemos conservarla.
 2. Aparece un cuadro de dialogo para introducir el nombre para la imagen a grabar. El único formato soportado para grabar imágenes es el bmp. Al introducir el nombre la aplicación añade la extensión .bmp si ésta no se introduce.
 3. Si todo ha funcionado correctamente aparecerá un mensaje para indicar que la imagen ha sido grabada y la imagen confabulada desaparecerá de la pantalla.

B. Código fuente ApplMDCT.

B.1 Clase AboutBox.java

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

/**
 * @author Pedro Reverte
 *
 * Implementa una ventana para presentar los créditos de la aplicación
 *
 * 2004
 */
public class AboutBox extends JDialog implements ActionListener {

    JPanel panel1 = new JPanel();
    JPanel panel2 = new JPanel();
    JPanel insetsPanel1 = new JPanel();
    JPanel insetsPanel2 = new JPanel();
    JPanel insetsPanel3 = new JPanel();
    JButton button1 = new JButton();
    JLabel imageLabel = new JLabel();
    JLabel label1 = new JLabel();
    JLabel label2 = new JLabel();
    JLabel label3 = new JLabel();
    JLabel label4 = new JLabel();
    JLabel label5 = new JLabel();
    JLabel label6 = new JLabel();
    JLabel label7 = new JLabel();
    JLabel label8 = new JLabel();
    BorderLayout borderLayout1 = new BorderLayout();
    BorderLayout borderLayout2 = new BorderLayout();
    FlowLayout flowLayout1 = new FlowLayout();
    GridLayout gridLayout1 = new GridLayout();
    String producto = "ApplMDCT - Marcas en DCT";
    String version = "Version 1.0";
    String copyright = "Pedro Reverte - 2004";
    String linea = "ApplMDCT ha sido realizada como parte de";
    String linea2 = " un TFC Seguretat Informàtica de ETIS de la";
    String linea3 = "Universitat Oberta de Catalunya";
    String linea4 = "(http://www.uoc.es)";

    /**
     * Constructor. Inicializa la ventana.
     * @param parent Frame padre para presentar la ventana.
     */
    public AboutBox(Frame parent) {
        super(parent);
        enableEvents(AWTEvent.WINDOW_EVENT_MASK);
        try {
            jbInit();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        pack();
    }

    /**
     * Method jbInit. Inicializa los datos de la ventana
     */
    private void jbInit() throws Exception {
        //
        imageLabel.setIcon(new
        ImageIcon(AboutBox.class.getResource("prevertem.jpg")));
        this.setTitle("About");
        setResizable(false);
    }
}

```

```

        panel1.setLayout(borderLayout1);
        panel2.setLayout(borderLayout2);
        insetsPanel1.setLayout(flowLayout1);
        insetsPanel2.setLayout(flowLayout1);
        insetsPanel2.setBorder(BorderFactory.createEmptyBorder(10, 10,
10, 10));
        GridLayout1.setRows(8);
        GridLayout1.setColumns(1);
        label1.setText(producto);
        label2.setText(version);
        label3.setText(copyright);
        label4.setText("");
        label5.setText(linea);
        label6.setText(linea2);
        label7.setText(linea3);
        label8.setText(linea4);
        insetsPanel3.setLayout(GridLayout1);
        insetsPanel3.setBorder(BorderFactory.createEmptyBorder(10, 60,
10, 10));
        button1.setText("Ok");
        button1.addActionListener(this);
        insetsPanel2.add(imageLabel, null);
        panel2.add(insetsPanel2, BorderLayout.WEST);
        this.getContentPane().add(panel1, null);
        insetsPanel3.add(label1, null);
        insetsPanel3.add(label2, null);
        insetsPanel3.add(label3, null);
        insetsPanel3.add(label4, null);
        insetsPanel3.add(label5, null);
        insetsPanel3.add(label6, null);
        insetsPanel3.add(label7, null);
        insetsPanel3.add(label8, null);
        panel2.add(insetsPanel3, BorderLayout.CENTER);
        insetsPanel1.add(button1, null);
        panel1.add(insetsPanel1, BorderLayout.SOUTH);
        panel1.add(panel2, BorderLayout.NORTH);
    }

    /**
     * Method processWindowEvent. Maneja el cierre de la ventana
     * @param e - evento producido
     */
    protected void processWindowEvent(WindowEvent e) {
        if (e.getID() == WindowEvent.WINDOW_CLOSING) {
            cancel();
        }
        super.processWindowEvent(e);
    }

    /**
     * Method cancel. Cierra la ventana
     */
    private void cancel() {
        dispose();
    }

    /**
     * Method actionPerformed. Cierra la ventana cuando sucede un
    evento de boton
     */
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == button1) {
            cancel();
        }
    }
}

```

B.2 Clase algoritmoMDCT.java

```

/**
 * @author Pedro Reverte
 *
 * Clase que implementa el algoritmo de inserción/extracción de marcas
 en los
 * coeficientes DCT de una imagen.
 *
 * 2004
 */

public class algoritmoMDCT {

    // etiquetas para los valores de los 4 pixel de RGB
    final static int RED      = 0;
    final static int GREEN   = 1;
    final static int BLUE    = 2;
    final static int OFFSET  = 3;

    // puntero a las imagenes a tratar
    private int [] [] [] imagen;
    private int [] [] [] imgEncriptada;
    // ancho de la imagen original
    private int widthImagen;
    // alto de la imagen original
    private int heightImagen;
    // ancho de la imagen marcada
    private int widthEncriptada;
    // alto de la imagen marcada
    private int heightEncriptada;

    // valores usados para establecer la fuerza de la marca
    private int maximo;
    private int minimo;
    // flag para recuperación de una única marca
    private boolean marcaUnica;

    /**
     * Constructor algoritmoMDCT.
     */
    public algoritmoMDCT () {

        // inicializar a imagenes vacias
        imagen = null;
        imgEncriptada = null;
        maximo=20;
        minimo=8;
        marcaUnica=false;
        // crea objeto de la clase DCT con calidad 100%
        DCT dctTrans = new DCT(0);
    }

    /**
     * setImagen. Establece el puntero de imagen original a la dada
 como parametro.
     * Usado tanto para recuperar una marca como para marcar una
 imagen.
     * @param image - Array de pixels RGB de la imagen.
     */
    public void setImagen(int [] [] [] image) {

        // establece la imagen
        this.imagen = image;
        // establece su ancho y alto
        this.heightImagen = image.length;
        this.widthImagen = image[0].length;
    }
}

```

```

    }

    /**
     * setImagenEncrip. Establece el puntero de imagen marcada a la
     dada como parametro.
     * Usado para recuperar la marca de una imagen.
     * @param image - Array de pixels RGB de la imagen.
     */
    public void setImagenEncrip(int [][][] image){

        // establece la imagen marcada
        this.imgEncriptada = image;
        // establece su ancho y alto
        this.heightEncriptada = image.length;
        this.widthEncriptada = image[0].length;

    }

    /**
     * setMaximo. Establece el parámetro máximo (diferencia máxima
     entre los
     * dos coeficientes dct para considerar el bloque válido para
     marcado)
     * @param max - Entero, valor máximo para la diferencia de
     coeficientes
     */
    public void setMaximo(int max){

        // establece el parámetro máximo
        this.maximo = max;

    }

    /**
     * setMinimo. Establece el parámetro minimo (diferencia minima
     entre los
     * dos coeficientes dct a partir de la cual se ampliará
     artificialmente)
     * @param min - Entero, valor minimo para la diferencia de
     coeficientes
     */
    public void setMinimo(int min){

        // establece el parámetro minimo
        this.minimo = min;

    }

    /**
     * setMarcaUnica. Establece el parámetro marcaUnica (usado para
     indicar que solo queremos
     * recuperar una única marca, util para algunos casos como el
     ataque de colision)
     * @param unica - Boolean, flag de una sola marca
     activado/desactivado
     */
    public void setMarcaUnica(boolean unica){

        this.marcaUnica = unica;

    }

    /**
     * bloqueToDCT. Calcula la DCT de un bloque de 8x8 con ayuda de la
     * clase DCT.
     * @param bloque - Bloque de 8x8 valores de una imagen.
     * @return bloque de 8x8 coeficientes DCT del bloque de entrada
     */
    private int[][] bloqueToDCT(int [][] bloque){

        int out[][] = new int [8][8];
    }

```

```

        char in[][] = new char [8][8];

        // convierte los valores de entrada al tipo char
        for(int i=0; i<8; i++){
            for(int j=0; j<8; j++){
                in[i][j] = (char)bloque[i][j];
            }
        }
        // calcula la transformada del coseno discreto del bloque
de entrada
        out = DCT.forwardDCT(in);
        return out;
    }

    /**
    * cuantizaBloqueDCT. Cuantiza un bloque 8x8 de coeficientes DCT
con ayuda de la
    * clase DCT
    * @param dct - Bloque de 8x8 de coeficientes DCT.
    * @return bloque de 8x8 coeficientes DCT cuantificados
    */
    private int[][] cuantizaBloqueDCT(int [][] dct){

        return DCT.quantitizeImage(dct, false);
    }

    /**
    * dequantizaBloqueDCT. Decuantiza un bloque de 8x8 coeficientes
DCT cuantificados
    * con ayuda de la clase DCT
    * @param dct - Bloque de 8x8 de coeficientes DCT cuantificados.
    * @return bloque de 8x8 coeficientes DCT originales
    */
    private int[][] dequantizaBloqueDCT(int [][] dct){

        int [][] temporal;

        temporal = DCT.dequantitizeImage(dct, false);
        return temporal;
    }

    /**
    * IDCTBloque. Calcula la inversa DCT de un bloque de 8x8
coeficientes DCT
    * con ayuda de la clase DCT
    * @param in - Bloque de 8x8 de coeficientes DCT.
    * @return bloque de 8x8 valores de la imagen.
    */
    private int[][] IDCTBloque(int [][] in){

        return DCT.inverseDCT(in);
    }

    /**
    * marcarBloqueDCT. Marca si es posible los coeficientes dct del
bloque
    * con el dato entregado (0 o 1) modificando la relacion de dos
coeficientes dct
    * preestablecidos.
    * @param dct - Puntero a bloque de 8x8 coeficientes DCT a marcar.
    * @param dato - Valor del bit a marcar (0 o 1).
    * @return cierto si se ha marcado el bloque, falso en caso
contrario
    */
    private boolean marcarBloqueDCT(int [][] dct, byte dato){

        boolean marcado = false;

```

```

        int temp1;
        int temp2;
        int temp3;
        temp1=dct [2] [0];
        temp2=dct [1] [2];
        // colocamos el valor del coeficiente mayor en temp1 y el
menor en temp2
        if(temp1<temp2){
            temp3=temp2;
            temp2=temp1;
            temp1=temp3;
        }
        // comprobamos que el bloque es apto para marcado
        if((temp1-temp2)<this.maximo && (temp1-temp2)!=0){
            // si la diferencia entre ambos es menor del minimo
la incrementamos para mayor robustez
            if((temp1-temp2)<this.minimo)
                temp1=temp1+3;
            // marcamos el bloque. (1) coeficiente (2,0)>(1,2)
            // (2,0)<(1,2) (0) coeficiente

            if(dato==1){
                dct [2] [0]=temp1;
                dct [1] [2]=temp2;
            }
            else{
                dct [2] [0]=temp2;
                dct [1] [2]=temp1;
            }
            marcado=true;
        }
        return marcado;
    }

/**
 * marcarImagen. Se encarga de marcar la imagen con los bits
entregados
 * como parámetro. Devuelve el array de valores de la imagen
marcada o null
 * si no se ha podido marcar.
 * @param datos - Valores de los bits a marcar (0 o 1).
 * @return array de pixels de la imagen marcada
 */
public int [] [] [] marcarImagen(byte [] datos){

    // si no esta establecida la imagen original devuelve null
    if(imagen == null){
        return null;
    }
    int xpos, ypos;
    // bloque de coeficientes dct
    int [] [] dct;
    // bloque de coeficientes dct marcados
    int [] [] bloque_marcado;
    // contadores
    int indice datos = 0;
    int contador marcados = 0;
    int contador fallos = 0;
    // flag de marcado
    boolean marcado = false;
    // array para los pixels de la imagen marcada
    int [] [] [] imagenMarcada = new
int [this.heightImagen] [this.widthImagen] [4];
    // array de 8x8 pixels de la imagen
    int [] [] [] bloque = new int [8] [8] [4];
    // array para cada uno de los canales RGB
    int arrayBlue [] [] = new int [8] [8];
    int arrayGreen [] [] = new int [8] [8];

```



```

    int arrayRed[] [] = new int [8][8];
    // arrays para luminancias y crominancias
    int arrayY[] [];
    int arrayCb[] [];
    int arrayCr[] [];
    // numero de bloques 8x8
    int bloquesX = (int)(this.widthImagen/8);
    int bloquesY = (int)(this.heightImagen/8);

    // realizamos una copia de la imagen original
    for(int i=0; i<this.heightImagen; i++)
        for(int j=0; j<this.widthImagen; j++)
            for(int h=0; h<4; h++)
                imagenMarcada[i][j][h] =
this.imagen[i][j][h];
    // bucle para procesar los bloques de la imagen
    for(int h=0; h<bloquesY; h++)
        for(int s=0; s<bloquesX; s++){
procesar
            //establece la posicion inicial del bloque a
            xpos = s*8;
            ypos = h*8;
            // capturamos los datos del bloque original
            for(int i=0; i<8; i++){
                for(int j=0; j<8; j++){
                    for(int k=0; k<4; k++){
                        bloque[i][j][k] =
imagen[ypos+i][xpos+j][k];
                    }
                }
            }

            // preparamos los 3 arrays de valores RGB
            for(int i=0; i<8; i++){
                for(int j=0; j<8; j++){
                    arrayBlue[i][j] =
bloque[i][j][BLUE];
                    arrayGreen[i][j] =
bloque[i][j][GREEN];
                    arrayRed[i][j] = bloque[i][j][RED];
                }
            }
            // convertimos a espacio YCrCb
            arrayY= utils.convertRGBtoY(arrayRed,
arrayGreen, arrayBlue, 8,8);
            arrayCb= utils.convertRGBtoCb(arrayRed,
arrayGreen, arrayBlue, 8,8);
            arrayCr= utils.convertRGBtoCr(arrayRed,
arrayGreen, arrayBlue, 8,8);
            // calculamos la transformada del coseno
discreto del array de luminancias Y
            dct = this.bloqueToDCT(arrayY);
            // procedemos a intentar marcar el bloque
            if(!marcarBloqueDCT(dct, datos[indice datos])){
                // sino se ha podido marcar debemos
buscar otro
                contador_fallos++;
                continue;
            }
            else{
                // el bloque se ha marcado
                // calculamos la inversa de DCT
                arrayY = this.IDCTBloque(dct);

                // convertimos a espacio RGB
                arrayRed = utils.convertYCbCrtoR(arrayY,
arrayCb, arrayCr,8,8);

```

```

arrayCb, arrayCr,8,8);
arrayBlue = utils.convertYCbCrtoB(arrayY,
arrayGreen =
utils.convertYCbCrtoG(arrayY, arrayCb, arrayCr,8,8);

// guardamos los valores en la imagen
final
for(int i=0; i<8; i++){
    for(int j=0; j<8; j++){
        bloque[i][j][BLUE]
        bloque[i][j][RED]
        bloque[i][j][GREEN]
    }
}

for(int i=0; i<8; i++){
    for(int j=0; j<8; j++){
        for(int k=0; k<4; k++){
            imagenMarcada [ypos+i] [xpos+j] [k] = bloque[i][j][k];
        }
    }
}

// incrementamos el contador de datos
procesados
indice_datos++;
contador_marcados++;
if(indice_datos == datos.length){
    // hemos procesado todos los datos
    indice_datos = 0;
    marcado = true;
}
}
}

// System.out.println("marcados: "+contador_marcados+" fallos:
"+contador fallos);
if(marcado)
    return imagenMarcada;
else
    return null;
}

/**
 * obtenerMarcaBloqueDCT. Recupera el valor 0 o 1 de la marca que
 * contiene el bloque de 8x8 coeficientes DCT.
 * @param dct - Valores del bloque 8x8 de coeficientes dct.
 * @return bit de la marca del bloque (0,1)
 */
private byte obtenerMarcaBloqueDCT(int[][] dct){
    byte resultado = -1;

    // si el coeficiente (2,0) es mayor que el (1,2) devuelve 1
    // si es menor devuelve 0, en otro caso devuelve -1
    if(dct[2][0]>dct[1][2])
        resultado=1;
    else
        if(dct[2][0]<dct[1][2])
            resultado=0;
    return resultado;
}

```

```

    /**
     * comprobarValidezBloque. Comprueba si un bloque 8x8 de
     coeficientes dct
     * es apto para su marcado.
     * @param bloqueDct - Valores del bloque 8x8 de coeficientes dct.
     * @return cierto si el bloque es apto, falso en caso contrario
     */
    private boolean comprobarValidezBloque(int[][] bloqueDct) {

        boolean valido = false;

        int temp1;
        int temp2;
        int temp3;
        temp1=bloqueDct[2][0];
        temp2=bloqueDct[1][2];
        // colocamos el coeficiente mayor en temp1 y el menor en
temp2
        if(temp1<temp2){
            temp3=temp2;
            temp2=temp1;
            temp1=temp3;
        }
        // si la relacion de coeficientes cumple las restricciones
es apto para marcado
        if((temp1-temp2)<maximo && (temp1-temp2)!=0){
            valido=true;
        }
        return valido;

    }

    /**
     * obtenerMarca. Recupera la marca de una array de pixels de una
     imagen marcada.
     * @param num datos - numero de bits que componen la marca.
     * @param izquierdaY - coordenada Y de la esquina superior
     izquierda respecto a
     *
     *
     * la imagen original. Util para
     recuperar marcas de recortes
     * @param izquierdaX - coordenada X de la esquina superior
     izquierda respecto a
     *
     *
     * la imagen original.
     * @return array de bits que componen la marca. Si alguno de los
     bits no se halla
     *
     * devuelve un valor de -1 para dicho bit
     */
    public byte[] obtenerMarca(int num_datos, int izquierdaY, int
     izquierdaX){

        int[][][] bloque = new int[8][8][4];
        int[][][] bloqueOrg = new int[8][8][4];
        int[][] dct;
        int xpos =0;
        int ypos=0;
        int posy;
        int posx;
        byte indice =0;
        byte marca;
        byte[] resultado = new byte[num_datos];
        int arrayBlue[][] = new int [8][8];
        int arrayGreen[][] = new int [8][8];
        int arrayRed[][] = new int [8][8];
        int arrayY[][];
        int arrayCb[][];
    }

```

```

    int arrayCr[] [];
    boolean salir=false;

    // inicializa array de contadores de los valores
    int[] [] valores = new int[num datos][2];
    for(int i=0; i<num datos; i++){
        valores[i][0]=0;
        valores[i][1]=0;
    }
    // si no hay establecidas las imagenes original y marcada
    devuelve null
    if(this.imagen == null || this.imgEncriptada==null)
        return null;
    // bucle para procesar los bloques de la imagen
    int bloquesX = (int)(this.widthImagen/8);
    int bloquesY = (int)(this.heightImagen/8);

    for(int h=0; h<bloquesY; h++){
        for(int s=0; s<bloquesX; s++){
            //establece la posicion inicial del bloque a
            procesar
                xpos = s*8;
                ypos = h*8;
                // capturamos los datos del bloque de la imagen
                original
                    for(int i=0; i<8; i++){
                        for(int j=0; j<8; j++){
                            for(int k=0; k<4; k++){
                                bloqueOrg[i][j][k] =
                                imagen[ypos+i][xpos+j][k];
                            }
                        }
                    }
                // convertimos el bloque a espacio YCrCb
                for(int i=0; i<8; i++){
                    for(int j=0; j<8; j++){
                        arrayBlue[i][j] =
                        bloqueOrg[i][j][BLUE];
                        arrayGreen[i][j] =
                        bloqueOrg[i][j][GREEN];
                        arrayRed[i][j] =
                        bloqueOrg[i][j][RED];
                    }
                }
                arrayY= utils.convertRGBtoY(arrayRed,
                arrayGreen, arrayBlue, 8,8);
                arrayCb= utils.convertRGBtoCb(arrayRed,
                arrayGreen, arrayBlue, 8,8);
                arrayCr= utils.convertRGBtoCr(arrayRed,
                arrayGreen, arrayBlue, 8,8);
                // calculamos la transformada del coseno
                discreto
                    dct = this.bloqueToDCT(arrayY);
                // comprobamos si el bloque es válido para
                marcar
                    if(!comprobarValidezBloque(dct)){
                        // si no es válido buscamos otro
                        continue;
                    }
                // bloque con marca válida
                // capturamos los datos del bloque marcado si
                se encuentra completo dentro del recorte
                    if(ypos>=izquierdaY &&
                    ypos<=(izquierdaY+heightEncriptada-8) && xpos>=izquierdaX &&
                    xpos<=(izquierdaX+widthEncriptada-8)){

```

```

// calculamos las posiciones iniciales
del bloque respecto al recorte
posy=ypos-izquierdaY;
posx=xpos-izquierdaX;
imagen marcada
// capturamos los datos del bloque de la
imgEncriptada [posy+i] [posx+j] [k];
for(int i=0; i<8; i++){
    for(int j=0; j<8; j++){
        for(int k=0; k<4; k++){
            bloque[i] [j] [k] =
        }
    }
}
// convertimos el bloque a espacio YCrCb
bloque[i] [j] [BLUE];
for(int i=0; i<8; i++){
    for(int j=0; j<8; j++){
        arrayBlue[i] [j] =
        arrayGreen[i] [j] =
        arrayRed[i] [j] =
    }
}
arrayY= utils.convertRGBtoY(arrayRed,
arrayGreen, arrayBlue, 8,8);
arrayCb= utils.convertRGBtoCb(arrayRed,
arrayGreen, arrayBlue, 8,8);
arrayCr= utils.convertRGBtoCr(arrayRed,
arrayGreen, arrayBlue, 8,8);
discreto
// calculamos la transformada del coseno
dct = this.bloqueToDCT(arrayY);
del bloque
// procedemos a intentar obtener la marca
marca = obtenerMarcaBloqueDCT(dct);
// si existe bit de marca incrementamos
el valor correspondiente
if(marca != -1){
    valores[indice] [marca]++;
}
// indice de bits extraidos
indice++;
// cuando se supera el numero de datos
reseteamos el indice
if(indice > num datos-1){
    indice = 0;
    // si esta activado el indicador de marca
unica, al obtener la primera marca
if(marcaUnica){
    // preparamos la salida del bucle
    salir = true;
    break;
}
}
if(salir)
    break;
}
for(int i=0; i<num_datos; i++){
// System.out.println("Valor "+i+" 0: "+valores[i] [0]+
Valor "+i+" 1: "+valores[i] [1]);
// si no hay marca de algun valor devuelve -1
if(valores[i] [0]==0 && valores[i] [1]==0){
    resultado[i] =-1;
}
}

```



```

/**
 * Constructor ApplMDCT
 */
ApplMDCT() {
    try {
        // inicializa componentes menu y barra de
herramientas
        jbInit();
    }
    catch (Exception e) {
        e.printStackTrace();
    }

    // crea instancias de GestorDatos y dialogo
    dialogo = new Dialogos(this);
    if (misDatos == null)
        misDatos = new GestorDatos(this.dialogo);
    // crea los tres paneles principales
    panelInsercion = new PanelInsercion(misDatos);
    panelExtraccion = new PanelExtraccion(misDatos);
    panelDatos = new PanelDatos(misDatos);
    // los añade al panel de pestañas
    principalTabbedPane.addTab("Insertar Marcas",
panelInsercion);
    principalTabbedPane.addTab("Extraer Marcas", panelExtraccion);
    principalTabbedPane.addTab("Datos Clientes", panelDatos);
    // selecciona inicialmente el panel de insercion
    principalTabbedPane.setSelectedIndex(0);
}

/**
 * jbInit. Inicializa los componentes principales de la
aplicación.
 */
private void jbInit() throws Exception {
    contentPane = (JPanel) this.getContentPane();
    contentPane.setLayout(borderLayout1);
    this.setSize(750, 510);
    this.setTitle("ApplMDCT");

    jMenuImagen.setText("Imágenes");
    jMenuImagenMarcar.setText("Marcar Imagen");
    jMenuImagenMarcar.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            jMenuImagenMarcar_actionPerformed(e);
        }
    });
    jMenuImagenGrabar.setText("Grabar Imagen Marcada");
    jMenuImagenGrabar.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            jMenuImagenGrabar_actionPerformed(e);
        }
    });

    jMenuImagenExtraer.setText("Extraer marca");
    jMenuImagenExtraer.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            jMenuImagenExtraer_actionPerformed(e);
        }
    });

    jMenuImagenSalir.setText("Salir");
    jMenuImagenSalir.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            jMenuImagenSalir_actionPerformed(e);
        }
    });
}

```

```

    });
    jMenuConsulta.setText("Consultar");
    jMenuConsultaDatos.setText("Consultar Datos Clientes");
    jMenuConsultaDatos.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            jMenuConsultaDatos_actionPerformed(e);
        }
    });

    jMenuHelp.setText("Help");
    jMenuHelpAbout.setText("About");
    jMenuHelpAbout.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            jMenuHelpAbout_actionPerformed(e);
        }
    });

    jButtonMarcarImagen.setIcon(new
    ImageIcon(App1MDCT.class.getResource("marcar.gif")));
    jButtonMarcarImagen.addActionListener(new
    java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
            jButtonMarcarImagen_actionPerformed(e);
        }
    });
    jButtonMarcarImagen.setToolTipText("Marcar Imagen");

    jButtonGrabarImagen.setIcon(new
    ImageIcon(App1MDCT.class.getResource("guardar.gif")));
    jButtonGrabarImagen.addActionListener(new
    java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
            jButtonGrabarImagen_actionPerformed(e);
        }
    });
    jButtonGrabarImagen.setToolTipText("Grabar marcada");

    jButtonExtraer.setIcon(new
    ImageIcon(App1MDCT.class.getResource("extraer.gif")));
    jButtonExtraer.addActionListener(new
    java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
            jButtonExtraer_actionPerformed(e);
        }
    });
    jButtonExtraer.setToolTipText("Extraer marca");

    jButtonConsultar.setIcon(new
    ImageIcon(App1MDCT.class.getResource("consultar.gif")));
    jButtonConsultar.addActionListener(new
    java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
            jButtonConsultar_actionPerformed(e);
        }
    });
    jButtonConsultar.setToolTipText("Consultar Datos");

    jToolBar.add(jButtonMarcarImagen);
    jToolBar.add(jButtonGrabarImagen);
    jToolBar.addSeparator();
    jToolBar.add(jButtonExtraer);
    jToolBar.addSeparator();
    jToolBar.add(jButtonConsultar);

    jMenuImagen.add(jMenuImagenMarcar);
    jMenuImagen.add(jMenuImagenGrabar);

```



```

        jMenuItemImagen.addSeparator();
jMenuItemImagen.add(jMenuItemImagenExtraer);
        jMenuItemImagen.addSeparator();
jMenuItemConsulta.add(jMenuItemConsultaDatos);
        jMenuItemImagen.addSeparator();
jMenuItemImagen.add(jMenuItemImagenSalir);
        jMenuItemHelp.add(jMenuItemHelpAbout);

        jMenuItemBar1.add(jMenuItemImagen);
jMenuItemBar1.add(jMenuItemConsulta);
        jMenuItemBar1.add(jMenuItemHelp);

        this.setJMenuBar(jMenuBar1);
        contentPane.add(jToolBar, BorderLayout.NORTH);
        contentPane.add(principalTabbedPane, BorderLayout.CENTER);
    }

    /**
     * Method resetPanels. Resetea el contenido de los 3 paneles de
     la aplicación
     */
    private void resetPanels() {
        principalTabbedPane.setComponentAt(0, panelInsercion);
        principalTabbedPane.setComponentAt(1, panelExtraccion);

        principalTabbedPane.setComponentAt(2, panelDatos);
    }

    /**
     * Method jMenuItemHelpAbout_actionPerformed. Implementa las
     acciones a realizar cuando se elige
     * la opción About del menu Help. Crea una ventana con la ayuda
     y la presenta en pantalla
     * @param e Evento que se ha producido
     */
    private void jMenuItemHelpAbout_actionPerformed(ActionEvent e) {
        AboutBox about = new AboutBox(this);
        Dimension aboutSize = about.getPreferredSize();
        Dimension frmSize = getSize();
        Point loc = getLocation();
        about.setLocation((frmSize.width - aboutSize.width) / 2 + loc.x,
(frmSize.height - aboutSize.height) / 2 + loc.y);
        about.setModal(true);
        about.show();
    }

    /**
     * Method processWindowEvent. Sobreescribe el método de JFrame
     para finalizar la aplicación cuando
     * se cierra la ventana de ésta.
     * @param e Evento que se ha producido
     */
    protected void processWindowEvent(WindowEvent e) {
        super.processWindowEvent(e);
        if (e.getID() == WindowEvent.WINDOW_CLOSING) {
            jMenuItemImagenSalir_actionPerformed(null);
        }
    }

    /**
     * Method jMenuItemImagenSalir_actionPerformed. Implementa las
     acciones a realizar cuando se elige
     * la opción Salir del menu Imagenes. Termina la aplicación
     * @param e Evento que se ha producido
     */
    private void jMenuItemImagenSalir_actionPerformed(ActionEvent e) {

```

```

        System.exit(0);
    }

    /**
     * Method jMenuItemImagenExtraer actionPerformed. Implementa las
     acciones a realizar cuando se elige
     * la opción Extraer Marca del menu Imagenes. Invoca el método
     del boton correspondiente a esa accion
     * @param e Evento que se ha producido
     */
    private void jMenuItemImagenExtraer actionPerformed(ActionEvent e) {
        jButtonExtraer_actionPerformed(e);
    }

    /**
     * Method jMenuItemImagenGrabar actionPerformed. Implementa las
     acciones a realizar cuando se elige
     * la opción Grabar marcada del menu Imagenes. Invoca el método
     del boton correspondiente a esa accion
     * @param e Evento que se ha producido
     */
    private void jMenuItemImagenGrabar actionPerformed(ActionEvent e) {
        jButtonGrabarImagen_actionPerformed(e);
    }

    /**
     * Method jMenuItemImagenMarcar_actionPerformed. Implementa las
     acciones a realizar cuando se elige
     * la opción Marcar Imagen del menu Imagenes. Invoca el método
     del boton correspondiente a esa accion
     * @param e Evento que se ha producido
     */
    private void jMenuItemImagenMarcar actionPerformed(ActionEvent e) {
        jButtonMarcarImagen_actionPerformed(e);
    }

    /**
     * Method jMenuItemConsultaDatos actionPerformed. Implementa las
     acciones a realizar cuando se elige
     * la opción Consultar Datos del menu Consultar. Invoca el
     método del boton correspondiente a esa accion
     * @param e Evento que se ha producido
     */
    private void jMenuItemConsultaDatos actionPerformed(ActionEvent e) {
        jButtonConsultar_actionPerformed(e);
    }

    /**
     * Method jButtonConsultar actionPerformed. Implementa las
     acciones a realizar cuando se pulsa el boton
     * Consultar Datos. Establece la pestaña activa al panel de
     consulta de datos.
     * @param e Evento que se ha producido
     */
    private void jButtonConsultar actionPerformed(ActionEvent e) {
        principalTabbedPane.setSelectedIndex(2);
    }

    /**
     * Method jButtonExtraer actionPerformed. Implementa las
     acciones a realizar cuando se pulsa el boton
     * Extraer Marca. Pide los datos necesarios para extraer la
     marca de una imagen, la extrae y presenta
     * los resultados en pantalla.
     * @param e Evento que se ha producido

```

```

*/
private void jButtonExtraer_actionPerformed(ActionEvent e) {

    int cx, cy;
    String recorte;

    // selecciona pestaña activa al panel de extraccion
    principalTabbedPane.setSelectedIndex(1);
    // pide el nombre de la imagen a procesar
    String name = dialogo.abrirFichero();
    if(name==null) // si se cancela termina
        return;
    // si esta activa la opcion de recorte
    if(panelExtraccion.getRecorte()){
        // pide las coordenadas de la esquina superior
        izquierda del recorte
        recorte = dialogo.preguntarInfo("Datos
Recorte", "Entrar coordenadas del recorte (x,y)");
        if(recorte==null) // si se cancela
            termina
                return;
        // separa las dos coordenadas y las convierte a
        enteros
        StringTokenizer st = new
StringTokenizer(recorte, ",");
        String[] token = new String[2];
        int indice = 0;
        while(st.hasMoreTokens()){
            token[indice] = st.nextToken();
            indice++;
            if(indice>1)
                break;

        }
        cx=Integer.parseInt(token[0]);
        cy=Integer.parseInt(token[1]);
    }
    else{
        cx=0; // si la opción recorte no esta activa las
        coordenadas son 0,0
        cy=0;
    }
    // activa/desactiva el flag de marca unica para el
    algoritmo de marcado según la opción marcaUnica
    if(panelExtraccion.getUnica())
        misDatos.setUnica(true);
    else
        misDatos.setUnica(false);
    // avisa que se introduzca el nombre de la imagen original
    original");
    this.dialogo.mostrarMensaje("Abrir", "Seleccione la imagen
original");
    String nameOriginal = dialogo.abrirFichero();
    if(nameOriginal==null) // si se cancela
        termina
            return;
    // invoca al Gestor de datos para que extraiga la marca
    dialogo);
    misDatos.recuperarMarca(name, nameOriginal, cx, cy,
    dialogo);
    // presenta los resultados en pantalla
    panelExtraccion.resetDatos(misDatos);
    principalTabbedPane.setComponentAt(1, panelExtraccion);

    principalTabbedPane.setSelectedIndex(1);
}

/**

```

```

    * Method jButtonGrabarImagen_actionPerformed. Implementa las
    acciones a realizar cuando se pulsa el boton
    * Grabar Imagen. Graba la ultima imagen marcada si existe.
    * @param e Evento que se ha producido
    */
    private void jButtonGrabarImagen_actionPerformed(ActionEvent e)
{
    // selecciona pestaña activa al panel de insercion
    principalTabbedPane.setSelectedIndex(0);
    // comprueba si existe imagen marcada en el gestor de datos
    if(misDatos.getCryp()==null){
imagen marcada");
        dialogo.mostrarMensaje("Grabar","No existe ninguna
        return;
    }
    // pide el nombre para guardar la imagen
    String name = dialogo.salvarFichero();
    if(name==null) //termina si se cancela
        return;
    // invoca la accion de grabar imagen marcada en el gestor
de datos
    if(misDatos.grabarMarcada(name, dialogo)){
        // si se ha marcado, presenta resultados en pantalla
        dialogo.mostrarMensaje("Grabar",name+" grabada");
        this.panelInsercion.resetDatos(misDatos);

        this.panelDatos.resetDatos(misDatos);
        principalTabbedPane.setComponentAt(0,
panelInsercion);
        principalTabbedPane.setComponentAt(2, panelDatos);

        principalTabbedPane.setSelectedIndex(0);
    }
}

/**
 * Method jButtonMarcarImagen_actionPerformed. Implementa las
 * acciones a realizar cuando se pulsa el boton
 * Marcar Imagen. Pide los datos necesarios para insertar una
 * marca en una imagen y la inserta
 * @param e Evento que se ha producido
 */
    private void jButtonMarcarImagen_actionPerformed(ActionEvent e)
{
    String nameOriginal;
    String cliente;
    String codigo;

    // establece pestaña activa al panel de insercion
    principalTabbedPane.setSelectedIndex(0);
    // pide el nombre de la imagen a marcar
    nameOriginal = dialogo.abrirFichero();
    if(nameOriginal==null) // si cancelamos termina
        return;
    // pide el nombre del cliente
    cliente = dialogo.preguntarInfo("Marcar Imagen","Entrar
nombre del cliente");
    if(cliente==null) // si cancelamos termina
        return;
    // pide el codigo a insertar
    codigo = dialogo.preguntarInfo("Marcar Imagen","Entrar
codigo para el cliente");
    if(codigo==null) // si cancelamos termina
        return;
    // si el codigo no es válido presenta mensaje de error

```

```

        if(Integer.parseInt(codigo)<0 ||
Integer.parseInt(codigo)>misDatos.getNumDatosHD()-1){
            dialogo.mostrarError("Error","codigo fuera de
rango");
            return;
        }
        // invoca al gestor de datos para marcar la imagen

        if(misDatos.marcarImagen(nameOriginal,cliente,Integer.parseInt(c
odigo),dialogo)){
            // si se ha marcado presenta los resultados en
pantalla
            this.panelInsercion.resetDatos(misDatos);
            principalTabbedPane.setComponentAt(0,
panelInsercion);
            principalTabbedPane.setSelectedIndex(0);
        }
    }

    /**
     * Method main. Genera una instancia de la aplicación y la
presenta en pantalla
     */
    public static void main(String[] args) {

        // Comprueba la version de Java
        String javaVersion = null;
        int major = 0;
        int minor = 0;

        try {
            javaVersion = System.getProperty("java.version");
            major = (new
Integer(javaVersion.substring(0,1)).intValue());
            minor = (new
Integer(javaVersion.substring(2,3)).intValue());
        }

        // si no detecta la version no pasa nada
        catch (Exception e) {
        }

        // si la version es 1.4 o superior acepta modo decorado
        if ((major > 1) || (major>0 && minor > 3)) {
            JFrame.setDefaultLookAndFeelDecorated(true);
            JDialog.setDefaultLookAndFeelDecorated(true);
        }

        ApplMDCT appli = new ApplMDCT();

        appli.pack();
        // centra la ventana y la presenta en pantalla
        Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = appli.getSize();
        if (frameSize.height > screenSize.height) {
            frameSize.height = screenSize.height;
        }
        if (frameSize.width > screenSize.width) {
            frameSize.width = screenSize.width;
        }
        appli.setLocation((screenSize.width - frameSize.width) / 2,
(screenSize.height - frameSize.height) / 2);
        appli.setVisible(true);
    }

```

```
}

```

B.4 Class BMPFile.java

```
// Rutinas de Dorothea Blostein
// from http://www.cs.queensu.ca/home/blostein/java.html
// CISC 124 Introduction to Computing Science II. Last taught Fall
// 2002.
// Here is Java image manipulation code used in this course.
// This code can be used as a starting point by anyone wishing to
// write
// image manipulations in Java.

import java.awt.*;
import java.io.*;
import java.awt.image.*;

public class BMPFile extends Component {

    //--- Private constants
    private final static int BITMAPFILEHEADER_SIZE = 14;
    private final static int BITMAPINFOHEADER_SIZE = 40;

    //--- Private variable declaration

    //--- Bitmap file header
    private byte bitmapFileHeader [] = new byte [14];
    private byte bfType [] = { (byte) 'B', (byte) 'M' };
    private int bfSize = 0;
    private int bfReserved1 = 0;
    private int bfReserved2 = 0;
    private int bfOffBits = BITMAPFILEHEADER_SIZE +
    BITMAPINFOHEADER_SIZE;

    //--- Bitmap info header
    private byte bitmapInfoHeader [] = new byte [40];
    private int biSize = BITMAPINFOHEADER_SIZE;
    private int biWidth = 0;
    private int biHeight = 0;
    private int biPlanes = 1;
    private int biBitCount = 24;
    private int biCompression = 0;
    private int biSizeImage = 0x030000;
    private int biXPelsPerMeter = 0x0;
    private int biYPelsPerMeter = 0x0;
    private int biClrUsed = 0;
    private int biClrImportant = 0;

    //--- Bitmap raw data
    private int bitmap [];

    //--- File section
    private FileOutputStream fo;

    //--- Default constructor
    public BMPFile() {

    }

    public void saveBitmap (String parFilename, int[] imagePix, int
        parWidth, int parHeight) {

        try {
            fo = new FileOutputStream (parFilename);
            save (imagePix, parWidth, parHeight);
            fo.close ();
        }
    }
}

```

```

    }
    catch (Exception saveEx) {
        saveEx.printStackTrace ();
    }
}

/*
 * The saveMethod is the main method of the process. This method
 * will call the convertImage method to convert the memory image
to
 * a byte array; method writeBitmapFileHeader creates and writes
 * the bitmap file header; writeBitmapInfoHeader creates the
 * information header; and writeBitmap writes the image.
 *
 */
private void save (int[] imagePix, int parWidth, int parHeight) {
    try {
        convertImage (imagePix, parWidth, parHeight);
        writeBitmapFileHeader ();
        writeBitmapInfoHeader ();
        writeBitmap ();
    }
    catch (Exception saveEx) {
        saveEx.printStackTrace ();
    }
}

/*
 * convertImage converts the memory image to the bitmap format
(BRG).
 * It also computes some information for the bitmap info header.
 *
 */
private boolean convertImage (int[] imagePix, int parWidth, int
parHeight) {
    int pad;
    bitmap = imagePix;

//    //          PixelGrabber pg = new PixelGrabber (parImage, 0, 0,
parWidth, parHeight,
//    //          bitmap, 0, parWidth);

//    try {
//        pg.grabPixels ();
//    }
//    catch (InterruptedException e) {
//        e.printStackTrace ();
//        return (false);
//    }

    pad = (4 - ((parWidth * 3) % 4)) * parHeight;
    biSizeImage = ((parWidth * parHeight) * 3) + pad;
    bfSize = biSizeImage + BITMAPFILEHEADER_SIZE +
        BITMAPINFOHEADER_SIZE;
    biWidth = parWidth;
    biHeight = parHeight;

    return (true);
}

/*
 * writeBitmap converts the image returned from the pixel grabber
to
 * the format required. Remember: scan lines are inverted in

```

```

* a bitmap file!
*
* Each scan line must be padded to an even 4-byte boundary.
*/
private void writeBitmap () {

    int size;
    int value;
    int j;
    int i;
    int rowCount;
    int rowIndex;
    int lastRowIndex;
    int pad;
    int padCount;
    byte rgb [] = new byte [3];

    size = (biWidth * biHeight) - 1;
    pad = 4 - ((biWidth * 3) % 4);

    //The following bug correction will cause the bitmap to be
unreadable by
    //GIMP. It must be there for the bitmap to be readable by
most other
    //graphics packages.
    if (pad == 4) { // <==== Bug correction
        pad = 0;
    } // <==== Bug correction

    rowCount = 1;
    padCount = 0;
    rowIndex = size - biWidth;
    lastRowIndex = rowIndex;

    try {
        // The following three lines of code are a correction
        // by Alin Arsu, Feb 2003. The original code set the top-
        // pixel in the image to black, and also shifted the
        // of the image by one pixel.
        // The original code was the following two lines:
        // for (j = 0; j < size; j++) {
        //     value = bitmap [rowIndex];
        // This is replaced by the three lines that appear next.
        for (j = 0; j < size+1; j++) {
            if (j<biWidth) { value = bitmap [rowIndex+1]; }
            else { value = bitmap [rowIndex]; }

            rgb [0] = (byte) (value & 0xFF);
            rgb [1] = (byte) ((value >> 8) & 0xFF);
            rgb [2] = (byte) ((value >> 16) & 0xFF);
            fo.write (rgb);
            if (rowCount == biWidth) {
                padCount += pad;
                for (i = 1; i <= pad; i++) {
                    fo.write (0x00);
                }
                rowCount = 1;
                rowIndex = lastRowIndex - biWidth;
                lastRowIndex = rowIndex;
            }
            else
                rowCount++;
            rowIndex++;
        }
    }
}

```



```

        //--- Update the size of the file
        bfSize += padCount - pad;
        biSizeImage += padCount - pad;
    }
    catch (Exception wb) {
        wb.printStackTrace ();
    }
}

/*
 * writeBitmapFileHeader writes the bitmap file header to the
file.
 *
 */
private void writeBitmapFileHeader () {
    try {
        fo.write (bfType);
        fo.write (intToDWord (bfSize));
        fo.write (intToWord (bfReserved1));
        fo.write (intToWord (bfReserved2));
        fo.write (intToDWord (bfOffBits));
    }
    catch (Exception wbfh) {
        wbfh.printStackTrace ();
    }
}

/*
 *
 * writeBitmapInfoHeader writes the bitmap information header
 * to the file.
 *
 */
private void writeBitmapInfoHeader () {
    try {
        fo.write (intToDWord (biSize));
        fo.write (intToDWord (biWidth));
        fo.write (intToDWord (biHeight));
        fo.write (intToWord (biPlanes));
        fo.write (intToWord (biBitCount));
        fo.write (intToDWord (biCompression));
        fo.write (intToDWord (biSizeImage));
        fo.write (intToDWord (biXPelsPerMeter));
        fo.write (intToDWord (biYPelsPerMeter));
        fo.write (intToDWord (biClrUsed));
        fo.write (intToDWord (biClrImportant));
    }
    catch (Exception wbih) {
        wbih.printStackTrace ();
    }
}

/*
 *
 * intToWord converts an int to a word, where the return
 * value is stored in a 2-byte array.
 *
 */
private byte [] intToWord (int parValue) {

```

```

    byte retValue [] = new byte [2];

    retValue [0] = (byte) (parValue & 0x00FF);
    retValue [1] = (byte) ((parValue >> 8) & 0x00FF);

    return (retValue);
}

/*
 *
 * intToDWord converts an int to a double word, where the return
 * value is stored in a 4-byte array.
 *
 */
private byte [] intToDWord (int parValue) {

    byte retValue [] = new byte [4];
    retValue [0] = (byte) (parValue & 0x00FF);
    retValue [1] = (byte) ((parValue >> 8) & 0x000000FF);
    retValue [2] = (byte) ((parValue >> 16) & 0x000000FF);
    retValue [3] = (byte) ((parValue >> 24) & 0x000000FF);

    return (retValue);
}
}

```

B.5 Class DCT.java

```

import java.lang.*;
import java.util.*;
import java.io.*;

/**
 * <h3><b>DCT - A Java implementation of the Discreet Cosine
 Transform</b></h3><br><br>
 * <hr>
 * The discreet cosine transform converts spatial information to
 "frequency" or
 * spectral information, with the X and Y axes representing
 frequencies of the
 * signal in different dimensions. This allows for "lossy" compression
 of image
 * data by determining which information can be thrown away without
 compromising
 * the image.<br><br>
 * The DCT is used in many compression and transmission codecs, such
 as JPEG, MPEG
 * and others. The pixels when transformed are arraged from the most
 significant pixel
 * to the least significant pixel. The DCT functions themselves are
 lossless.
 * Pixel loss occurs when the least significant pixels are quantitized
 to 0.
 * <br><br>
 * This is NOT a JPEG or JFIF compliant implementation however it
 could
 * be with very little extra work. (i.e. A huffman encoding stage
 needs
 * to be added.) I am making this source available in the hopes that
 * someone will add this functionality to the class, if you do, please
 * email me! As always, if you have any problems feel free to contact
 * me. (Or comments, or praise, etc..)
 * <br><br>

```

```

* <b>Keep in mind</b> that when compressing color images with this,
you will
* need to break the image up into it's R G B components and preform
* the calculations three times!!
* <br><br>
* <b>A general algorithm for DCT compression with this class:</b>
<br><br>
* 1) Create a DCT Object.<br>
* 2) Set up your program to read pixel information in 8*8 blocks. See
example.<br>
* 3) Run the forwardDCT() on all blocks.<br>
* 4) Run the quantitizeImage() on all blocks.<br>
* 5) If you want, send the information to the
imageCompressor().<br><br>
* <b>A general algorithm for DCT decompression with this class:</b>
<br><br>
* 1) Create a DCT Object. <br>
* 2) Set up the program to convert compressed data in 8*8 blocks. (if
compressed)<br>
* 3) Run the data through dequantitizeImage(). <br>
* 4) Run the data through inverseDCT().<br>
* <br><br>
* A complete implementation of an image compressor which compares
* the quality of the two images is also available. See the
* JEncode/Decode <a href="JEncode.java">source code.</a> The
* <a href="DCT.java">DCT.java source code</a> is also available.
* The implementation is also handy for seeing how to break the image
* down, read in 8x8 blocks, and reconstruct it. The <a
href="steve.jpg">
* sample graphic</a> is handy to have too. (Bad pic of me)
* The best way to get ahold of me is through my
* <a href="http://eagle.uccb.ns.ca/steve/home.html">homepage</a>.
There's
* lots of goodies there too.
* @version 1.0.1 August 22nd 1996
* @author <a href="http://eagle.uccb.ns.ca/steve/home.html">Stephen
Manley</a> - smanley@eagle.uccb.ns.ca
*/

// <pre>
public class DCT
{
    /**
     * DCT Block Size - default 8
     */
    public static int N          = 8;

    /**
     * Image Quality (0-25) - default 25 (worst image / best
compression)
     */
    public static int QUALITY    = 0;

    /**
     * Image width - must correspond to imageArray bounds - default
320
     */
    public static int ROWS      = 320;

    /**
     * Image height - must correspond to imageArray bounds - default
200
     */
    public static int COLS      = 240;

    /**
     * The ZigZag matrix.
     */
    public static int zigZag[] [] = new int[64][2];

```

```

/**
 * Cosine matrix. N * N.
 */
public static double c[] [] = new double[N] [N];

/**
 * Transformed cosine matrix, N*N.
 */
public static double cT[] [] = new double[N] [N];

/**
 * Quantitization Matrix.
 */
public static int quantum[] [] = new int[N] [N];

/**
 * DCT Result Matrix
 */
public static int resultDCT[] [] = new int[ROWS] [COLS];

/**
 * Constructs a new DCT object. Initializes the cosine transform
matrix
 * these are used when computing the DCT and it's inverse. This
also
 * initializes the run length counters and the ZigZag sequence.
Note that
 * the image quality can be worse than 25 however the image will
be
 * extemely pixelated, usually to a block size of N.
 *
 * @param QUALITY The quality of the image (0 best - 25 worst)
 *
 */
public DCT(int QUALITY)
{
    initZigZag();
    initMatrix(QUALITY);
}

/**
 * This method sets up the quantization matrix using the Quality
parameter
 * and then sets up the Cosine Transform Matrix and the Transposed
CT.
 * These are used by the forward and inverse DCT. The RLE encoding
 * variables are set up to track the number of consecutive zero
values
 * that have output or will be input.
 * @param quality The quality scaling factor
 */
private static void initMatrix(int quality)
{
    int i;
    int j;

    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
        {
            quantum[i] [j] = (1 + ((1 + i + j) * quality));
        }
    }

    for (j = 0; j < N; j++)
    {
        double nn = (double) (N);

```

```

        c[0][j] = 1.0 / Math.sqrt(nn);
        cT[j][0] = c[0][j];
    }

    for (i = 1; i < 8; i++)
    {
        for (j = 0; j < 8; j++)
        {
            double jj = (double)j;
            double ii = (double)i;
            c[i][j] = Math.sqrt(2.0/8.0) * Math.cos(((2.0 * jj +
1.0) * ii * Math.PI) / (2.0 * 8.0));
            cT[j][i] = c[i][j];
        }
    }
}

/**
 * Initializes the ZigZag matrix.
 */
private static void initZigZag()
{
    zigZag[0][0] = 0; // 0,0
    zigZag[0][1] = 0;
    zigZag[1][0] = 0; // 0,1
    zigZag[1][1] = 1;
    zigZag[2][0] = 1; // 1,0
    zigZag[2][1] = 0;
    zigZag[3][0] = 2; // 2,0
    zigZag[3][1] = 0;
    zigZag[4][0] = 1; // 1,1
    zigZag[4][1] = 1;
    zigZag[5][0] = 0; // 0,2
    zigZag[5][1] = 2;
    zigZag[6][0] = 0; // 0,3
    zigZag[6][1] = 3;
    zigZag[7][0] = 1; // 1,2
    zigZag[7][1] = 2;
    zigZag[8][0] = 2; // 2,1
    zigZag[8][1] = 1;
    zigZag[9][0] = 3; // 3,0
    zigZag[9][1] = 0;
    zigZag[10][0] = 4; // 4,0
    zigZag[10][1] = 0;
    zigZag[11][0] = 3; // 3,1
    zigZag[11][1] = 1;
    zigZag[12][0] = 2; // 2,2
    zigZag[12][1] = 2;
    zigZag[13][0] = 1; // 1,3
    zigZag[13][1] = 3;
    zigZag[14][0] = 0; // 0,4
    zigZag[14][1] = 4;
    zigZag[15][0] = 0; // 0,5
    zigZag[15][1] = 5;
    zigZag[16][0] = 1; // 1,4
    zigZag[16][1] = 4;
    zigZag[17][0] = 2; // 2,3
    zigZag[17][1] = 3;
    zigZag[18][0] = 3; // 3,2
    zigZag[18][1] = 2;
    zigZag[19][0] = 4; // 4,1
    zigZag[19][1] = 1;
    zigZag[20][0] = 5; // 5,0
    zigZag[20][1] = 0;
    zigZag[21][0] = 6; // 6,0
    zigZag[21][1] = 0;
    zigZag[22][0] = 5; // 5,1
    zigZag[22][1] = 1;
    zigZag[23][0] = 4; // 4,2

```

```
zigZag [23] [1] = 2;
zigZag [24] [0] = 3; // 3,3
zigZag [24] [1] = 3;
zigZag [25] [0] = 2; // 2,4
zigZag [25] [1] = 4;
zigZag [26] [0] = 1; // 1,5
zigZag [26] [1] = 5;
zigZag [27] [0] = 0; // 0,6
zigZag [27] [1] = 6;
zigZag [28] [0] = 0; // 0,7
zigZag [28] [1] = 7;
zigZag [29] [0] = 1; // 1,6
zigZag [29] [1] = 6;
zigZag [30] [0] = 2; // 2,5
zigZag [30] [1] = 5;
zigZag [31] [0] = 3; // 3,4
zigZag [31] [1] = 4;
zigZag [32] [0] = 4; // 4,3
zigZag [32] [1] = 3;
zigZag [33] [0] = 5; // 5,2
zigZag [33] [1] = 2;
zigZag [34] [0] = 6; // 6,1
zigZag [34] [1] = 1;
zigZag [35] [0] = 7; // 7,0
zigZag [35] [1] = 0;
zigZag [36] [0] = 7; // 7,1
zigZag [36] [1] = 1;
zigZag [37] [0] = 6; // 6,2
zigZag [37] [1] = 2;
zigZag [38] [0] = 5; // 5,3
zigZag [38] [1] = 3;
zigZag [39] [0] = 4; // 4,4
zigZag [39] [1] = 4;
zigZag [40] [0] = 3; // 3,5
zigZag [40] [1] = 5;
zigZag [41] [0] = 2; // 2,6
zigZag [41] [1] = 6;
zigZag [42] [0] = 1; // 1,7
zigZag [42] [1] = 7;
zigZag [43] [0] = 2; // 2,7
zigZag [43] [1] = 7;
zigZag [44] [0] = 3; // 3,6
zigZag [44] [1] = 6;
zigZag [45] [0] = 4; // 4,5
zigZag [45] [1] = 5;
zigZag [46] [0] = 5; // 5,4
zigZag [46] [1] = 4;
zigZag [47] [0] = 6; // 6,3
zigZag [47] [1] = 3;
zigZag [48] [0] = 7; // 7,2
zigZag [48] [1] = 2;
zigZag [49] [0] = 7; // 7,3
zigZag [49] [1] = 3;
zigZag [50] [0] = 6; // 6,4
zigZag [50] [1] = 4;
zigZag [51] [0] = 5; // 5,5
zigZag [51] [1] = 5;
zigZag [52] [0] = 4; // 4,6
zigZag [52] [1] = 6;
zigZag [53] [0] = 3; // 3,7
zigZag [53] [1] = 7;
zigZag [54] [0] = 4; // 4,7
zigZag [54] [1] = 7;
zigZag [55] [0] = 5; // 5,6
zigZag [55] [1] = 6;
zigZag [56] [0] = 6; // 6,5
zigZag [56] [1] = 5;
zigZag [57] [0] = 7; // 7,4
zigZag [57] [1] = 4;
zigZag [58] [0] = 7; // 7,5
```

```

        zigZag[58][1] = 5;
        zigZag[59][0] = 6; // 6,6
        zigZag[59][1] = 6;
        zigZag[60][0] = 5; // 5,7
        zigZag[60][1] = 7;
        zigZag[61][0] = 6; // 6,7
        zigZag[61][1] = 7;
        zigZag[62][0] = 7; // 7,6
        zigZag[62][1] = 6;
        zigZag[63][0] = 7; // 7,7
        zigZag[63][1] = 7;
    }

    /**
     * This method preforms a matrix multiplication of the input pixel
     data matrix
     * by the transposed cosine matrix and store the result in a
     temporary
     * N * N matrix. This N * N matrix is then multiplied by the
     cosine matrix
     * and the result is stored in the output matrix.
     *
     * @param input The Input Pixel Matrix
     * @returns output The DCT Result Matrix
     */
    public static int[][] forwardDCT(char input[][])
    {
        int output[][] = new int[N][N];
        double temp[][] = new double[N][N];
        double temp1;
        int i;
        int j;
        int k;

        for (i = 0; i < N; i++)
        {
            for (j = 0; j < N; j++)
            {
                temp[i][j] = 0.0;
                for (k = 0; k < N; k++)
                {
                    temp[i][j] += (((int) input[i][k]) - 128) *
cT[k][j]);
                }
            }
        }

        for (i = 0; i < N; i++)
        {
            for (j = 0; j < N; j++)
            {
                temp1 = 0.0;

                for (k = 0; k < N; k++)
                {
                    temp1 += (c[i][k] * temp[k][j]);
                }

                output[i][j] = (int) Math.round(temp1);
            }
        }

        return output;
    }

    /**
     * This method reads in DCT codes dequantizes them
     * and places them in the correct location. The codes are stored
     in the

```

```

    * zigzag format so they need to be redirected to a N * N block
through
    * simple table lookup. After dequantitization the data needs to
be
    * run through an inverse DCT.
    *
    * @param inputData 8x8 Array of quantitized image data
    * @param zigzag Boolean switch to enable/disable zigzag path.
    * @returns outputData A N * N array of de-quantitized data
    *
    */
public static int[] [] dequantitizeImage(int[] [] inputData, boolean
zigzag)
{
    int i = 0;
    int j = 0;
    int a = 0;
    int b = 0;
    int row;
    int col;
    int outputData[] [] = new int[N] [N];

    double result;

    if (zigzag)
    {
        for (i=0; i<(N*N); i++)
        {
            row = zigZag[i] [0];
            col = zigZag[i] [1];

            result = inputData[row] [col] * quantum[row] [col];
            outputData[row] [col] = (int) (Math.round(result));
        }
    }
    else
    {
        for (i=0; i<8; i++)
        {
            for (j=0; j<8; j++)
            {
                result = inputData[i] [j] * quantum[i] [j];
                outputData[i] [j] = (int) (Math.round(result));
            }
        }
    }

    return outputData;
}

/**
 * This method orders the DCT result matrix into a zigzag pattern
and then
 * quantitizes the data. The quantitized value is rounded to the
nearest integer.
 * Pixels which round or divide to zero are the loss associated
with
 * quantitizing the image. These pixels do not display in the AWT.
(null)
 * Long runs of zeros and the small ints produced through this
technique
 * are responsible for the small image sizes. For block sizes < or
> 8,
 * disable the zigzag optimization. If zigzag is disabled on
encode it
 * must be disabled on decode as well.
 *
 * @param inputData 8x8 array of DCT image data.

```



```

    * @param zigzag Boolean switch to enable/disable zigzag path.
    * @returns outputData The quantitized output data
    */
    public static int[][] quantitizeImage(int inputData[][], boolean
zigzag)
    {
        int outputData[][] = new int[N][N];
        int i = 0;
        int j = 0;
        int a = 0;
        int b = 0;
        int row;
        int col;

        double result;

        if (zigzag)
        {
            for (i = 0; i < (N*N); i++)
            {
                row = zigZag[i][0];
                col = zigZag[i][1];
                result = (inputData[row][col] / quantum[row][col]);
                outputData[row][col] = (int)(Math.round(result));
            }
        }
        else
        {
            for (i=0; i<N; i++)
            {
                for (j=0; j<N; j++)
                {
                    result = inputData[i][j] / quantum[i][j];
                    outputData[i][j] = (int)(Math.round(result));
                }
            }
        }

        return outputData;
    }

    /**
    * <br><br>
    * This does not huffman code,
    * it uses a minimal run-length encoding scheme. Huffman, Adaptive
Huffman
    * or arithmetic encoding will give much better preformance. The
information
    * accepted is quantitized DCT data. The output array should be
    * scanned to determine where the end is.
    *
    * @param image Quantitized image data.
    * @returns The string representation of the image. (Compressed)
    */
    public int[] compressImage(int[] QDCT, boolean log)
    {
        int i = 0;
        int j = 0;
        int k = 0;
        int temp = 0;
        int curPos = 0;
        int runCounter = 0;
        int imageLength = ROWS*COLS;

        int pixel[] = new int[ROWS*COLS];

        while((i<imageLength))

```

```

    {
        temp = QDCT[i];

        while((i < imageLength) && (temp == QDCT[i]))
        {
            runCounter++;
            i++;
        }

        if (runCounter > 4)
        {
            pixel[j] = 255;
            j++;
            pixel[j] = temp;
            j++;
            pixel[j] = runCounter;
            j++;
        }
        else
        {
            for (k=0; k<runCounter; k++)
            {
                pixel[j] = temp;
                j++;
            }
        }

        if (log)
        {
            System.out.print(".") + "\r";
        }

        runCounter = 0;
        //i++;
    }

    return pixel;
}

/**
 * This method determines the runs in the input data, decodes it
 * and then returnss the corrected matrix. It is used to decode
the data
 * from the compressImage method. Huffman encoding, Adaptive
Huffman or
 * Arithmetic will give much better compression.
 *
 * @param DCT Compressed DCT int array (Expands to whole image).
 * @returns The decompressed one dimensional array.
 */
public int[] decompressImage(int[] DCT, boolean log)
{
    int i = 0;
    int j = 0;
    int k = 0;
    int temp = 0;
    int imageLength = ROWS*COLS;
    int pixel[] = new int[ROWS*COLS];

    while (i < imageLength)
    {
        temp = DCT[i];

        if (k < imageLength)
        {
            if (temp == 255)
            {
                i++;
                int value = DCT[i];

```

```

        i++;
        int length = DCT[i];
        for(j=0; j<length; j++)
        {
            pixel[k] = value;
            k++;
        }
    }
    else
    {
        pixel[k] = temp;
        k++;
    }
}
if (log)
{
    System.out.print(".." + "\r");
}
i++;
}

for (int a = 0; a < 80; a++)
{
    System.out.print(pixel[a] + " ");
}
System.out.println();
for (int a = 0; a < 80; a++)
{
    System.out.print(DCT[a] + " ");
}

return pixel;
}

/**
 * This method is preformed using the reverse of the operations
 * preformed in
 * the DCT. This restores a N * N input block to the corresponding
 * output
 * block with values scaled to 0 to 255 and then stored in the
 * input block
 * of pixels.
 *
 * @param input N * N input block
 * @returns output The pixel array output
 */
public static int[] [] inverseDCT(int input[] [])
{
    int output[] [] = new int[N][N];
    double temp[] [] = new double[N][N];
    double temp1;
    int i;
    int j;
    int k;

    for (i=0; i<N; i++)
    {
        for (j=0; j<N; j++)
        {
            temp[i][j] = 0.0;

            for (k=0; k<N; k++)
            {
                temp[i][j] += input[i][k] * c[k][j];
            }
        }
    }
}

```

```

    }
    for (i=0; i<N; i++)
    {
        for (j=0; j<N; j++)
        {
            temp1 = 0.0;

            for (k=0; k<N; k++)
            {
                temp1 += cT[i][k] * temp[k][j];
            }

            temp1 += 128.0;

            if (temp1 < 0)
            {
                output[i][j] = 0;
            }
            else if (temp1 > 255)
            {
                output[i][j] = 255;
            }
            else
            {
                output[i][j] = (int)Math.round(temp1);
            }
        }
    }

    return output;
}

/**
 * This method uses bit shifting to convert an array of two bytes
 * to a integer (16 bits max). Byte 0 is the most significant byte
 * and Byte 1 is the least significant byte.
 * @param bitSet Two bytes to convert
 * @returns The constructed integer
 */
private static int bytetoInt(byte bitSet[])
{
    int returnInt = 0;

    byte MSB = bitSet[0];
    byte LSB = bitSet[1];

    returnInt = ((MSB+128) << 8) | (LSB+128);

    return returnInt;
}

/**
 * This method uses bit shifting to convert an integer to an array
 * of two bytes, byte 0, the most significant and byte 1, the
least
 * significant.
 * @param count The integer to convert. (16 bit max)
 * @returns The array of two bytes.
 */
private static byte[] inttoByte(int count)
{
    int LSB = 0;
    int MSB = 0;
    byte bitSet[] = new byte[2];

    if (!(count > 65535))
    {
        LSB = ((count & 0x000000ff));
    }

```

```

        MSB = ((count & 0x0000ff00) >> 8);
    }
    else
    {
        System.out.println("Integer > than 16 bit. Exiting..");
        System.exit(count);
    }

    bitSet[0] = (byte) (MSB-128);
    bitSet[1] = (byte) (LSB-128);

    return bitSet;
}
}
// </pre>

```

B.6 Clase Dialogos.java

```

import java.io.*;
import javax.swing.*;

/**
 * @author Pedro Reverte
 *
 * Clase utilizada para la presentacion en pantalla de mensajes y
 * ventanas de dialogo.
 *
 * 2004
 */

public class Dialogos {

    private JFrame main;
    // ultimo directorio usado en los dialogos de abrir/guardar
    private File lastDir;

    /**
     * Constructor Dialogos.
     * @param padre - Frame padre para mostrar los mensajes
     */
    Dialogos(JFrame padre) {
        this.main = padre;
    }

    /**
     * Method dialogoFichero. Presenta en pantalla una ventana de
     * dialogo de abrir/guardar
     * según el parámetro entregado
     * @param open - Booleano cierto para dialogo abrir, falso para
     * dialogo guardar
     * @return String - nombre del fichero elegido o nulo si se ha
     * cancelado
     */
    private String dialogoFichero(boolean open) {
        JFileChooser fc = new JFileChooser();
        fc.setMultiSelectionEnabled(false);
        fc.addChoosableFileFilter(new ImageFilter());
        // Si existe un directorio usado previamente repite
        if (this.lastDir != null) {
            fc.setCurrentDirectory(this.lastDir);
        }
        // sino usa el actual
        else {
            fc.setCurrentDirectory(new
            File(System.getProperty("user.dir")));
        }
    }
}

```

```

        // muestra la ventana de dialogo
        int returnVal =
(open)?fc.showOpenDialog(main):fc.showSaveDialog(main);
        // si no se ha cancelado devuelve el nombre elegido
        if (returnVal == JFileChooser.APPROVE_OPTION){
            File filename = fc.getSelectedFile();
            this.lastDir = filename.getParentFile();
            return filename.getAbsolutePath();
        }
        // si se ha cancelado devuelve nulo
        return null;
    }

    /**
     * Method abrirFichero. Genera un dialogo para abrir un fichero
     * @return String - nombre del fichero elegido o nulo si se ha
cancelado
     */
    public String abrirFichero(){
        return dialogoFichero(true);
    }

    /**
     * Method salvarFichero. Genera un dialogo para guardar un
fichero
     * @return String - nombre del fichero elegido o nulo si se ha
cancelado
     */
    public String salvarFichero(){
        return dialogoFichero(false);
    }

    /**
     * Method mostrarMensaje. Genera un mensaje en una ventana de
mensajes.
     * @param titol - String con el titulo de la ventana
     * @param msg - String con el texto del mensaje
     */
    public void mostrarMensaje(String titol, String msg){
JOptionPane.showMessageDialog(main,msg,titol,JOptionPane.INFORMATION_M
ESSAGE);
    }

    /**
     * Method preguntarInfo. Genera una ventana para peticion de
información.
     * @param titol - String con el titulo de la ventana
     * @param msg - String con el texto de la peticion
     * @return String con la respuesta a la peticion
     */
    public String preguntarInfo(String titol, String msg){
        return
JOptionPane.showInputDialog(main,msg,titol,JOptionPane.QUESTION_MESSAG
E);
    }

    /**
     * Method mostrarError. Genera un mensaje en una ventana de
error
     * @param titol - String con el titulo de la ventana
     * @param msg - String con el texto del error
     */
    public void mostrarError(String titol, String msg){
JOptionPane.showMessageDialog(main,msg,titol,JOptionPane.ERROR_MESSAGE
);
    }
}

```

}

B.7 Clase GestorDatos.java

```

import javax.swing.*;

/**
 * @author Pedro Reverte
 *
 * Clase principal de control que implementa las acciones a realizar
 * por la aplicación:
 * Marcado de imagenes, extracción de marcas y consulta de datos.
 *
 * 2004
 */
public class GestorDatos {

    // imagenes
    private ImageIcon imagenUncryp;
    private ImageIcon imagenCryp;
    private ImageIcon imagenExtraida;

    // datos de las ultimas imagenes usadas
    private ImagenM imagenUn;
    private ImagenM imagenCr;
    private ImagenM imagenEx;

    private String NameOriginal;
    private Dialogos dialogo;
    private GestorImagenesM gestorImg;
    private HammingDual hd;
    private algoritmoMDCT algoritmo;

    /**
     * Constructor GestorDatos.
     * @param dia - Instancia de Dialogos para presentar mensajes
     */
    GestorDatos(Dialogos dia) {

        this.dialogo = dia;
        this.NameOriginal="-";
        this.imagenUncryp = null;
        this.imagenCryp = null;
        this.imagenUn = null;
        this.imagenCr = null;
        this.imagenEx = null;
        this.gestorImg = new GestorImagenesM();
        this.hd = new HammingDual();
        this.algoritmo = new algoritmoMDCT();
    }

    /**
     * Method getUncryp. Accesor que devuelve la ultima imagen
     * original leida.
     * @return ImageIcon - imagen original.
     */
    public ImageIcon getUncryp() {

        return imagenUncryp;
    }

    /**
     * Method getCryp. Accesor que devuelve la ultima imagen
     * marcada.
     * @return ImageIcon - imagen marcada.
     */
    public ImageIcon getCryp() {

```

```

        return imagenCryp;
    }

    /**
     * Method getExtraida. Accesor que devuelve la ultima imagen
    marcada usada
     * en una extracción de marca
     * @return ImageIcon - imagen marcada
     */
    public ImageIcon getExtraida() {

        return imagenExtraida;
    }

    /**
     * Method getNameImagen. Devuelve el nombre de la ultima imagen
    original utilizada
     * @return String - nombre de la imagen original
     */
    public String getNameImagen() {
        return imagenUn.getNombre();
    }

    /**
     * Method getNameMarcada. Devuelve el nombre de la ultima imagen
    marcada
     * @return String - nombre de la imagen marcada
     */
    public String getNameMarcada() {
        return imagenCr.getNombre();
    }

    /**
     * Method getNameExtraida. Devuelve el nombre de la ultima
    imagen marcada utilizada en una
     * extracción
     * @return String - nombre de la imagen usada en extraccion.
     */
    public String getNameExtraida() {
        return imagenEx.getNombre();
    }

    /**
     * Method getNameOriginalExtraida. Devuelve el nombre de la
    ultima imagen original utilizada
     * en una extracción de marca
     * @return String - nombre de la imagen original
     */
    public String getNameOriginalExtraida() {
        return this.NameOriginal;
    }

    /**
     * Method getCodigoMarcada. Devuelve el codigo usado en la
    ultima imagen marcada.
     * @return entero - codigo de imagen marcada
     */
    public String getCodigoMarcada() {
        return Integer.toString(imagenCr.getCodigo());
    }

    /**
     * Method getClienteMarcada. Devuelve el nombre del cliente
    usado en la ultima imagen marcada.
     * @return String - nombre del cliente
     */
    public String getClienteMarcada() {
        return imagenCr.getCliente();
    }

```



```

    }

    /**
     * Method getCodigoExtraida. Devuelve el codigo obtenido en la
ultima extracción de marca.
     * @return entero - codigo de imagen marcada
     */
    public String getCodigoExtraida() {
        return Integer.toString(imagenEx.getCodigo());
    }

    /**
     * Method getClienteExtraida. Devuelve el nombre del cliente
obtenido en la ultima
     * extracción de marca.
     * @return String - nombre del cliente
     */
    public String getClienteExtraida() {
        return imagenEx.getCliente();
    }

    /**
     * Method getMarcaExtraida. Devuelve la marca obtenida en la
ultima extracción de marca.
     * @return String - marca obtenida.
     */
    public String getMarcaExtraida() {
        return imagenEx.getMarca();
    }

    /**
     * Method getNumDatos. Devuelve el numero de codigos validos de
la instancia de Hamming dual
     * @return entero - numero de codigos validos.
     */
    public int getNumDatosHD() {
        return hd.getNumWords();
    }

    /**
     * Method marcarImagen. Abre la imagen cuyo nombre se recibe
como parámetro y la marca
     * con el codigo recibido. Crea las instancias de ImagenM
correspondientes a la imagen
     * original y la marcada.
     * @param name - String nombre de la imagen original a marcar
     * @param cliente - String nombre del cliente
     * @param cod - entero codigo de la marca
     * @param dialogo - instancia de Dialogos para presentar mensajes
     * @return boolean - Cierto si se ha marcado la imagen, falso en
caso contrario
     */
    public boolean marcarImagen(String name, String cliente, int
cod, Dialogos dialogo) {

        int indiceNombre = name.lastIndexOf("\\");
        // obtiene el nombre del path si es necesario
        String soloNombre = name.substring(indiceNombre+1);
        // comprueba que la imagen no haya sido previamente marcada
con el mismo codigo
        if (gestorImg.buscar(soloNombre, cod) != null) {
            dialogo.mostrarError("Error", "Imagen ya marcada con ese
codigo");
            this.imagenUncryp=null;
            this.imagenCryp=null;
            return false;
        }
        // lee el archivo de la imagen original
        this.imagenUncryp = imageIO.loadImagen(name);
    }

```

```

        if(this.imagenUncryp == null){
            dialogo.mostrarError("Error", "No se ha podido leer el
archivo de imagen");
            this.imagenUncryp=null;
            this.imagenCryp=null;
            return false;
        }

// obtiene los bits de la marca correspondientes al codigo
byte[] datos = this.hd.getMarca(cod);
// obtiene los bits de la marca en forma de string
String marca = this.hd.getString(cod);
// obtiene el array de pixels de la imagen original

int[][][] imagePixels =
imageIO.getImagePixels(this.imagenUncryp);
// establece el array de pixels en la instancia de algoritmo
algoritmo.setImagen(imagePixels);
// crea el array de pixels correspondiente a la imagen marcada
int[][][] imageMarcadaPixels = algoritmo.marcarImagen(datos);
if(imageMarcadaPixels==null){
    dialogo.mostrarError("Error", "Error al marcar la imagen");
    this.imagenUncryp=null;
    this.imagenCryp=null;
    return false;
}
// crea la imagen marcada a partir del array de pixels
this.imagenCryp = imageIO.crearImagen(imageMarcadaPixels);
// crea las instancias de ImagenM correspondientes
this.imagenUn = new ImagenM(soloNombre);
this.imagenCr = new ImagenM(soloNombre, cliente, cod, marca);

return true;
}

/**
 * Method grabarMarcada. Graba si existe, la ultima imagen
 * marcada a un archivo con el
 * nombre recibido como parámetro
 * @param name - String Nombre del archivo a guardar
 * @param dialogo - Instancia de Dialogos para mostrar mensajes
 * @return boolean - Cierto si se ha grabado el archivo, falso
 * en caso contrario
 */
public boolean grabarMarcada(String name, Dialogos dialogo) {
    // crea el archivo con la imagen marcada
    imageIO.saveImagen(this.imagenCryp, name);
    // obtiene el nombre a partir del path si es necesario
    int indiceNombre = name.lastIndexOf("\\");
    String soloNombre = name.substring(indiceNombre+1);
    // comprueba que el nombre termine en bmp
    soloNombre = imageIO.bmpTack(soloNombre);
    // establece el nombre de la instancia de ImagenM de la
imagen marcada
    imagenCr.setNombre(soloNombre);
    // añade la imagen al fichero de texto que almacena los
datos de imagenes marcadas

    gestorImg.añadir(imagenUn.getNombre(), imagenCr.getCliente(), imag
enCr.getCodigo(), imagenCr.getMarca());
    // guarda el fichero
    gestorImg.save();
    // reseteamos las imagenes
    this.imagenUncryp=null;
    this.imagenCryp=null;

    return true;
}

```

```

    }

    /**
     * Method getTextoDatos. Metodo que devuelve el texto formateado
de los datos de
     * imagenes marcadas
     * @return String - Texto formateado
     */
    public String getTextoDatos() {
        return gestorImg.getTextoFormateado();
    }

    /**
     * Method recuperarMarca. Recupera la marca de la imagen
indicada y sus datos
     * asociados si existen
     * @param name - String nombre de la imagen marcada
     * @param nameOriginal - String nombre de la imagen original
correspondiente a la marcada
     * @param cx - entero coordenada x de la esquina superior
izquierda de la imagen respecto
     * a la original
     * @param cy - entero coordenada y de la esquina superior
izquierda de la imagen respecto
     * a la original
     * @param dialogo - Instancia de Dialogos para mostrar mensajes
     * @return boolean - Cierto si se ha recuperado la marca, falso
en caso contrario
     */
    public boolean recuperarMarca(String name, String nameOriginal,
int cx, int cy, Dialogos dialogo) {

        ImageIcon original;
        String cliente;
        int codigo;
        String cadenaMarca;
        String nombre;

        // lee el archivo de la imagen marcada
        this.imagenExtraida = imageIO.loadImagen(name);
        if(this.imagenExtraida == null){
            dialogo.mostrarError("Error", "No se ha podido leer el
archivo de imagen marcada");
            this.imagenExtraida = null;
            return false;
        }
        // lee el archivo de la imagen original
        original = imageIO.loadImagen(nameOriginal);
        if(original == null){
            dialogo.mostrarError("Error", "No se ha podido leer el
archivo de imagen original");
            this.imagenExtraida = null;
            return false;
        }
        // obtiene los arrays de pixels correspondientes
        int[][][] pixelsMarcada =
imageIO.getImagePixels(this.imagenExtraida);
        int[][][] pixelsOriginal = imageIO.getImagePixels(original);
        // establece las imagenes en la instancia de algoritmo
        algoritmo.setImagen(pixelsOriginal);
        algoritmo.setImagenEncrip(pixelsMarcada);
        // obtiene el numero de bits de la marca
        int numDatos = this.hd.getNumDatos();
        // obtiene los bits de la marca
        byte[] marca = algoritmo.obtenerMarca(numDatos, cy, cx);
        // convierte los bits a string
        cadenaMarca = hd.obtenerString(marca);
    }

```

```

        // obtiene el codigo correspondiente a la marca
        codigo = hd.getCodigo(marca);
        // obtiene los nombres a partir del path si es necesario
        int indiceNombre = name.lastIndexOf("\\");
        nombre = name.substring(indiceNombre+1);

        indiceNombre = nameOriginal.lastIndexOf("\\");
        this.NameOriginal = nameOriginal.substring(indiceNombre+1);

        if(codigo != -1){// si es -1 no se ha encontrado el codigo
de esa marca
            // busca el cliente correspondiente al codigo y la
imagen
            cliente=gestorImg.buscar(this.NameOriginal,codigo);
            if(cliente==null){
                cliente="Desconocido";
            }
        }
        else{
            cliente="Desconocido";
        }
        // crea una instancia de ImagenM con los datos obtenidos
        this.imagenEx = new ImagenM(nombre, cliente, codigo,
cadenaMarca);

        return true;
    }

    /**
     * Method setUnica. Establece el flag de una sola marca en el
    algoritmo de marcado/recuperación
     * @param b - Boolean Flag activado/desactivado
     */
    public void setUnica(boolean b) {
        algoritmo.setMarcaUnica(b);
    }
}

```

B.8 Clase GestorImagenesM.java

```

import java.io.*;
import java.util.*;

/**
 * @author Pedro Reverte
 *
 * Gestiona el almacenamiento de datos de imagenes marcadas. Los datos
se guardan en un fichero de
 * texto. Los datos de las imagenes se hallan representados mediante
lineas de texto con el siguiente
 * formato: "nombre_imagen_original,codigo_de_la_marca,cliente,marca"
 *
 * 2004
 */

public class GestorImagenesM {

    // tamaño de los campos para presentacion en pantalla
    final static int[] SIZES = {40,6,20,8};
    // fichero de texto con los datos
    final static String fichero="ImagenesM.txt";

    private ArrayList listado;

```

```

private String imagen;
private String cliente;
private String marca;
private int codigo;
private String textoFormateado;

/**
 * Constructor GestorImagenesM. Lee los datos del fichero de
 texto o lo crea si no existe,
 * crea una lista con los datos de las imagenes marcadas y
 prepara un string para presentarlos
 * en pantalla.
 */
GestorImagenesM() {

    String strLine;
    listado = new ArrayList();

    try{
        ImagenM tmpImg;
        // abre el fichero de datos
        File file = new File(fichero);
        // si no existe, lo crea
        if(!file.exists()){
            BufferedWriter buf = new BufferedWriter( new
FileWriter(fichero));
            buf.close( );
        }
        // lee los datos del fichero
        BufferedReader br = new BufferedReader(new
FileReader(fichero));

        while((strLine=br.readLine())!=null){
            // divide cada linea en los datos que
representa
            StringTokenizer st = new
StringTokenizer(strLine, ",");
            String token;

            int indice =0;
            while(st.hasMoreTokens()){
                token = st.nextToken();
                switch(indice){
                    case 0: this.imagen = token; break;
                    case 1: this.codigo =
Integer.parseInt(token); break;
                    case 2: this.cliente = token;
break;
                    case 3: this.marca = token; break;
                }
                indice++;
            }
            // crea una instancia de ImagenM que representa
los datos
            // de una imagen marcada
            tmpImg = new ImagenM(imagen, cliente, codigo,
marca);
            // la añade a la lista
            listado.add(tmpImg);
        }
        br.close();
    }
    catch( IOException ioe ) {
        System.out.println( "Error leyendo fichero" );
        System.out.println( ioe );
    }
}

```

```

// si existen datos los formatea para su presentacion en
pantalla
    if(listado != null)
        textoFormateado = formatearTexto();
    }

/**
 * Method formatearTexto. Prepara un String con los datos del
 fichero de texto para
 * su presentacion en pantalla alineandolos en cuatro columnas
 de tamaño especificado
 * por el atributo SIZES
 * @return String - Datos formateados
 */
private String formatearTexto() {

    StringBuffer texto = new StringBuffer();

    Iterator it = listado.iterator();
    ImagenM imgTemp;
    String linea;
    int i=0;
    // recorre la lista de objetos ImagenM
    while (it.hasNext()){
        imgTemp = (ImagenM) listado.get(i);
        // formatea el objeto para presentacion en pantalla
        linea = formateaLinea(imgTemp);
        i++;
        // lo añade al string
        texto.append(linea);
        it.next();
    }

    return texto.toString();
}

/**
 * Method formateaLinea. Formatea los datos de un objeto ImagenM
 en una línea para
 * su presentacion en pantalla.
 * @param img - objeto ImagenM con los datos de una imagen
 marcada
 * @return String - Datos de ImagenM formateados
 */
private String formateaLinea(ImagenM img){

    String linea = img.toString();
    StringTokenizer st = new StringTokenizer(linea, ",");
    String token;
    String tokenF;
    StringBuffer lineaF = new StringBuffer();
    // divide el string que contiene los datos de ImagenM en
 sus componentes
    int indice =0;
    while(st.hasMoreTokens()){
        token = st.nextToken();
        // formatea cada componente a su tamaño adecuado
        tokenF= pad(token, SIZES[indice]);
        indice++;
        if(indice>3)
            indice=0;
        // añade el componente formateado a la línea
        lineaF.append(tokenF);
    }
    lineaF.append("\n");
    return lineaF.toString();
}

/**

```

```

    * Method pad. Añade espacios en blanco a la derecha de un
string hasta alcanzar el
    * tamaño especificado como parámetro
    * @param input - String de entrada
    * @param requiredSize - Tamaño deseado para el string
    * @return String - String con el tamaño requerido
    */
    private String pad(String input, int requiredSize){

        int inputLength = input.length();
        if (inputLength >= requiredSize)
            return input;

        StringBuffer buffer = new StringBuffer(requiredSize);
        buffer.append(input);
        for (int i = requiredSize - inputLength; i > 0; --i) {
            buffer.append(" ");
        }
        return buffer.toString();
    }

    /**
    * Method añadir. Añade los datos de una nueva imagen marcada al
    fichero de datos.
    * @param imagen - String con el nombre de la imagen original
    * @param nombre - String con el nombre del cliente
    * @param codigo - entero con el codigo de la marca
    * @param marca - String con la marca insertada
    */
    public void añadir(String imagen, String nombre, int codigo,
String marca){
        // crea una nueva instancia de ImagenM
        ImagenM tmp = new ImagenM(imagen, nombre, codigo, marca);
        // la añade a la lista
        this.listado.add(tmp);
        // la añade al texto formateado
        this.textoFormateado = formatearTexto();
    }

    /**
    * Method getTextoFormateado. Accesor al atributo con el texto
    formateado
    * @return String - Datos formateados para presentacion en
    pantalla
    */
    public String getTextoFormateado(){

        return this.textoFormateado;
    }

    /**
    * Method save. Guarda los datos existentes en un archivo del
    sistema de ficheros
    */
    public void save(){

        try{
            BufferedWriter buf = new BufferedWriter( new
FileWriter(fichero));

            Iterator it = listado.iterator();
            ImagenM imgTemp;
            String linea;
            // recorrido por la lista de ImagenM
            int i=0;
            while (it.hasNext()){
                imgTemp = (ImagenM) listado.get(i);
                // obtiene los datos como string en el formato
adecuado

```

```

        linea = imgTemp.toString();
        // los guarda en el archivo
        buf.write(linea);
        buf.write("\n");
        i++;
        it.next();
    }
    buf.close( );
}
catch( IOException ioe ) {
    System.out.println( "Error en fichero" );
    System.out.println( ioe );
}
}

/**
 * Method buscar. Busca los datos de una imagen a partir del
 * nombre de la imagen y su
 * codigo de marcado y si existe devuelve el nombre del cliente.
 * @param imagen - String con el nombre de la imagen original
 * @param codigo - entero con el codigo de la marca
 * @return String - con el nombre del cliente si se ha
 * encontrado o null en caso contrario
 */
public String buscar(String imagen, int codigo){
    Iterator it = listado.iterator();
    ImagenM imgTemp;
    String cliente=null;
    // recorrido por la lista de imagenM
    int i=0;
    while (it.hasNext()){
        imgTemp = (ImagenM) listado.get(i);
        if(imgTemp.getNombre().equalsIgnoreCase(imagen) &&
imgTemp.getCodigo()==codigo) {
            // si existe la imagen buscada devuelve el
nombre del cliente
            cliente=imgTemp.getCliente();

            break;
        }
        i++;
        it.next();
    }
    return cliente;
}
}

```

B.9 Clase HammingDual.java

```

/**
 * @author Pedro Reverte
 *
 * Implementacion de un codigo hamming dual. Consta de 8 codigos
 * diferentes (0-7)
 *
 * 2004
 */
public class HammingDual {

    // codigo hamming usado
    private static final String codigo[] = {"0000000", "0011011",
"0101101",
    "0110110", "1001110", "1010101", "1100011", "1111000"};
}

```



```

    /**
     * Constructor HammingDual.
     */
    HammingDual() {
        ;
    }

    /**
     * Method getMarca. devuelve la marca correspondiente a un
    codigo.
     * @param cod - entero con el codigo
     * @return byte[] - devuelve un array con los bits de la marca
    correspondiente
     */
    public byte[] getMarca(int cod) {
        return obtenerBytes(codigo[cod]);
    }

    /**
     * Method getNumDatos. Devuelve el numero de bits de que consta
    una marca del
     * codigo hamming dual.
     * @return entero - numero de bits de una marca
     */
    public int getNumDatos() {
        return codigo[0].length();
    }

    /**
     * Method getNumWords. Devuelve el numero de palabras de que
    consta el
     * codigo hamming dual.
     * @return entero - numero de palabras del codigo hamming dual
     */
    public int getNumWords() {
        return codigo.length;
    }

    /**
     * Method getCodigo. Devuelve el codigo correspondiente a una
    marca dada. Corrige errores de
     * un bit en la marca. Devuelve -1 si la marca contiene errores
    que no puede corregir.
     * @param marca - array de bits con la marca
     * @return entero - codigo correspondiente a la marca (0-7) o -1
     */
    public int getCodigo(byte[] marca) {
        String antes=null;
        String despues=null;
        String bit;
        String newCadena;
        // convertimos la marca a un string
        String cadenaMarca = this.obtenerString(marca);
        // si es una marca existente devolvemos el codigo
        for(int i=0; i<codigo.length; i++){
            if(codigo[i].equalsIgnoreCase(cadenaMarca))
                return i;
        }
        // sino comprobamos si hay un error en un bit
        for(int j=0; j<cadenaMarca.length(); j++){
            // cambiamos un bit de una posicion de la marca cada
    vez

            // para ver si coincide con una marca existente
            antes=cadenaMarca.substring(0,j);

            despues=cadenaMarca.substring(j+1,cadenaMarca.length());

            if(cadenaMarca.substring(j,j+1).equalsIgnoreCase("0"))
                bit="1";

```

```

        else
            bit="0";
            newCadena=antes+bit+despues;
            for(int i=0; i<codigo.length; i++){
                // si coincide devolvemos el codigo
                if(codigo[i].equalsIgnoreCase(newCadena))
                    return i;
            }
        // si no coincide con ninguna marca devolvemos -1
        return -1;
    }

/**
 * Method obtenerBytes. Convierte un string de una marca en un
array de bits
 * @param datos - String con la marca
 * @return byte[] - array con los bits correspondientes a la
marca
 */
private byte[] obtenerBytes(String datos){

    byte[] nuevos = new byte[datos.length()];

    for(int i=0; i<datos.length(); i++)
        nuevos[i]= (byte)
Integer.parseInt(datos.substring(i,i+1));

    return nuevos;
}

/**
 * Method obtenerString. Convierte un array de bits de una marca
a un String.
 * @param datos - array con los bits correspondientes a la marca
 * @return String - con la marca correspondiente
 */
public String obtenerString(byte[] datos){

    String marca="";
    for(int i=0; i<datos.length; i++){
        marca=marca.concat(Byte.toString(datos[i]));
    }
    return marca;
}

/**
 * Method getString. Accesor que devuelve el String de la marca
correspondiente
 * a un codigo.
 * @param cod - entero con el codigo.
 * @return String - marca correspondiente.
 */
public String getString(int cod) {
    return codigo[cod];
}
}

```

B.10 Clase ImageFilter.java

```

import java.io.*;
import javax.swing.filechooser.FileFilter;

/**
 * @author Pedro Reverte
 */

```

```

* Filtro de ficheros para una instancia de JFileChooser.
* Implementada a partir de la documentacion de Sun
*
http://java.sun.com/docs/books/tutorial/uiswing/components/filechooser
.html

*/
public class ImageFilter extends FileFilter{

    /**
    * Accept all directories and all gif, jpg, or png files.
    * @return File's acceptance.
    */
    public boolean accept(File f) {
        if (f.isDirectory()) {
            return true;
        }

        String extension = this.getExtension(f);
        if (extension != null) {
            if (extension.equals("gif") ||
                extension.equals("jpeg") ||
                extension.equals("jpg") ||
                extension.equals("bmp")) {
                return true;
            } else {
                return false;
            }
        }

        return false;
    }

    /**
    * The description of this filter
    * @return The literal string "Images (png, gif, jpg)".
    */
    public String getDescription() {
        return "Imagenes (bmp, gif, jpg)";
    }

    /**
    * Get the extension of a file.
    * @return File extension.
    */
    private String getExtension(File f) {
        String ext = null;
        String s = f.getName();
        int i = s.lastIndexOf('.');

        if (i > 0 && i < s.length() - 1) {
            ext = s.substring(i+1).toLowerCase();
        }
        return ext;
    }
}

```

B.11 Clase imageIO.java

```

// imageIO.java
// This class provides routines for image input and output.
// Rutinas de Dorothea Blostein
// from http://www.cs.queensu.ca/home/blostein/java.html
// CISC 124 Introduction to Computing Science II. Last taught Fall
2002.
// Here is Java image manipulation code used in this course.

```

```

// This code can be used as a starting point by anyone wishing to
write
// image manipulations in Java.
// Adaptado a mis necesidades.
// Modificadas por Pedro Reverte en orden a cubrir las necesidades de
mi
// presente proyecto.

import java.util.*;
import java.awt.*;
import java.awt.image.*;
import java.awt.MediaTracker.*;
import java.io.*;
import javax.swing.ImageIcon;

public class imageIO {

/** ***** loadImagen
*****
* Lee el contenido de un fichero de imagen bmp, jpeg o gif y lo
devuelve en forma
* de ImageIcon. El nombre puede ser un nombre local (como "Camel.jpg")
o un path
* (como "/home/turing/alan.bmp").
*
* public static ImageIcon loadImage(String openName)
* Parametros:
*   String openName - Nombre de la imagen a abrir
*
* Precondicion:
*   ninguna
*
* Postcondicion:
* Devuelve un ImageIcon con la imagen leida
*/
    public static ImageIcon loadImagen(String openName) {
        Image img = (Image)null; // create a null Image

        try{
            // Check that the file name has a legal extension
            if (openName.endsWith(".gif") ||
openName.endsWith(".jpg")
||
openName.endsWith(".jpeg")) {
                img = Toolkit.getDefaultToolkit().getImage(openName);
            }
            else if (openName.endsWith(".bmp")) {
                img = utils.loadbitmap("./", openName);
            }
            else {
                img=(Image)null; // we can't read the file
            }

            if (img != null) {
                // Make sure the entire image is loaded before
continuing
                Button b = new Button(); // Create a button to
use as a parameter
// to the constructor
for MediaTracker.
                MediaTracker tracker = new MediaTracker(b);
                tracker.addImage(img,0);
                tracker.waitForID(0);

                // Create "observer", an object that allows us to

```

```

        // use getWidth and getHeight.
        iObserver observer = new iObserver();
        int width = img.getWidth(observer);
        int height = img.getHeight(observer);

        if(width== -1 || height== -1){
            // the image has not loaded.
            img = (Image)null;
        }
    } // if img != null

    // If the image did not load, print an explanatory
    // message to the user and ask him/her to try again.
    if (img == null) {
        return null;
    }
    // System.out.println("Could not read an image from
file "
        //
        // +openName);
    } // end of "try"

    // Catch InterruptedException for tracker.waitFor(), and catch
    // IOException for the console operations.
    catch(InterruptedException e) {
        System.out.println(e);
        System.exit(1);
    }
    return new ImageIcon(img);
} // end of method loadImage

/** ***** getImagePixels
*****
// El metodo getImagePixels convierte un objeto imageIcon en un array
3D
// representativo de sus pixels (rows, columns, pixel value (red,
green, blue, offset))
//
// private static int[][][] getImagePixels(ImageIcon image)
//
// Parametros:
// img - la imagen a convertir
//
// Precondicion:
// la imagen debe estar completamente cargada
//
// Postcondicion:
// se devuelve un array 3D representando los pixels de la imagen
*/
public static int[][][] getImagePixels(ImageIcon image) {

    Image img = image.getImage();
    // Get the raw pixel data
    iObserver observer = new iObserver();
    int width1 = img.getWidth(observer);
    int height1 = img.getHeight(observer);
    int[] rawPixels = utils.getPixels(img,width1,height1);

    // Each pixel is represented by 32 bits. Separate the tH32
bits into
    // four 8-bit values (red, green, blue, offset).
    int[][] rgbPixels = new int[rawPixels.length][4];
    for(int j=0; j<rawPixels.length; j++) {
        rgbPixels[j][0] = ((rawPixels[j]>>16)&0xff);
        rgbPixels[j][1] = ((rawPixels[j]>>8)&0xff);
        rgbPixels[j][2] = (rawPixels[j]&0xff);
    }
}

```

```

        rgbPixels[j][3] = ((rawPixels[j]>>24)&0xff);
    } // for j

    // Arrange the data by rows and columns
    int[][][] imagePixels = new int[height1][width1][4];
    int index=0;
    for(int row=0; row<imagePixels.length; row++) {
        for(int col=0; col<imagePixels[0].length; col++) {
            for(int rgbo=0; rgbo<4; rgbo++) {

imagePixels[row][col][rgbo]=rgbPixels[index][rgbo];
                } // for rgbo
                index++;
            } // for col
        } // for row
    }
    return imagePixels;
} // end of method getImagePixels

/** ***** saveImage
*****
// El metodo saveImage guarda un ImageIcon como un bitmap de windows
(.bmp).
//
// public static void saveImage(ImageIcon image)
//
// Parametros:
//     image - un ImageIcon con la imagen a grabar
//     saveName - String con el nombre del fichero a grabar
//
// Precondicion:
//     image != null
//
// Postcondicion:
//     la imagen se ha convertido a un bitmap de windows, y se ha
grabado
//     con el nombre proporcionado por el usuario. (se añade .bmp al
nombre del
//     fichero en caso de que no tenga esta extensión)
*/
public static void saveImagen(ImageIcon image, String saveName){

    int[][][] imagePixels;

    imagePixels = getImagePixels(image);
    int height = imagePixels.length;
    int width = imagePixels[0].length;
    int[][] flat = new int[width*height][4];

    // If saveName does not already end in .bmp, then add .bmp to
saveName.
    saveName=bmpTack(saveName);

    // Flatten the image into a 2D array.
    int index=0;
    for(int row=0; row<height; row++) {
        for(int col=0; col<width; col++) {
            for(int rgbo=0; rgbo<4; rgbo++) {
                flat[index][rgbo]=imagePixels[row][col][rgbo];
            }
            index++;
        } // for col
    } // for row

    // Combine the 8-bit red, green, blue, offset values into 32-
bit words.
    int[] outPixels = new int[flat.length];
    for(int j=0; j<flat.length; j++) {

```

```

        outPixels[j] = ((flat[j][0]&0xff)<<16) |
((flat[j][1]&0xff)<<8)
        | (flat[j][2]&0xff) |
((flat[j][3]&0xff)<<24);
    } // for j

    // Write the data out to file with the name given by string
saveName.
    BMPFile bmpf = new BMPFile();
    bmpf.saveBitmap(saveName, outPixels, width, height);
    // System.out.println("Saved " + saveName);
} // end of method saveImage

/** ***** crearImage
*****
// El metodo crearImage convierte un array de pixels RGB en un
ImageIcon.
//
// public static ImageIcon saveImage(int[][][] imagePixels)
//
// Parametros:
// imagePixels - array 3D con los datos de los pixels
//
// Precondicion:
// imagePixels != null
//
// Postcondicion:
// imagePixels ha sido convertido a un ImageIcon
*/
public static ImageIcon crearImagen(int[][][] imagePixels){

    int height = imagePixels.length;
    int width = imagePixels[0].length;
    int[][] flat = new int[width*height][4];

    // Flatten the image into a 2D array.
    int index=0;
    for(int row=0; row<height; row++) {
        for(int col=0; col<width; col++) {
            for(int rgbo=0; rgbo<4; rgbo++) {
                flat[index][rgbo]=imagePixels[row][col][rgbo];
            }
            index++;
        } // for col
    } // for row

    // Combine the 8-bit red, green, blue, offset values into 32-
bit words.
    int[] outPixels = new int[flat.length];
    for(int j=0; j<flat.length; j++) {
        outPixels[j] = ((flat[j][0]&0xff)<<16) |
((flat[j][1]&0xff)<<8)
        | (flat[j][2]&0xff) |
((flat[j][3]&0xff)<<24);
    } // for j

    BufferedImage bi = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
    bi.setRGB(0,0,width,height,outPixels,0,width);
    return new ImageIcon(bi);

}

// ***** bmpTack
*****
// The bmpTack method checks whether the name parameter ends in
".bmp".
// If it doesn't then ".bmp" is appended to the name.
//

```

```

// private static String bmpTack(String name)
// Parameters:
//   name - a string
//
// Precondition:
//   name != null
//
// Postcondition:
//   If name ended in .bmp before being passed into bmpTack, name is
returned.
//   If name did not end in .bmp before being passed into bmpTack,
name+.bmp
//   is returned
    public static String bmpTack(String name) {
        if (name.endsWith(".bmp"))
            return name;
        else
            return name+".bmp";
    } // end of method bmpTack
} // end of class imageIOm

```

B.12 Clase ImagenM.java

```

/**
 * @author Pedro Reverte
 *
 * Clase que representa los datos de una imagen marcada. Los datos que
almacena son los siguientes:
 * nombre de la imagen original correspondiente a la imagen marcada,
nombre del cliente para el que
 * se marco la imagen, codigo de la marca, String con la marca
insertada.
 *
 * 2004
 */
public class ImagenM {

    private String Nombre;
    private String Cliente;
    private intCodigo;
    private String Marca;

    /**
 * Constructor ImagenM. Crea una instancia de imagenM con los
datos entregados
 * @param nombre - String con el nombre de la imagen original.
 * @param cliente - String con el nombre del cliente
 * @param codigo - entero con el codigo de la marca
 * @param marca - String con la marca
 */
    ImagenM(String nombre, String cliente, int codigo, String
marca) {

        this.Nombre = nombre;
        this.Cliente = cliente;
        this.Codigo = codigo;
        this.Marca = marca;
    }

    /**
 * Constructor ImagenM. Crea una instancia de imagenM con un
solo dato, el nombre
 * de la imagen original. El resto de datos son nullos.
 * @param nombre - String con el nombre de la imagen original.
 */
    ImagenM(String nombre) {

```



```
        this.Nombre = nombre;
        this.Cliente = null;
        this.Codigo = -1;
        this.Marca = null;
    }

    /**
     * Method getNombre. Accesor que devuelve el String
     correspondiente al nombre original de una
     * imagen marcada.
     * @return String - nombre de la imagen original.
     */
    public String getNombre(){
        return this.Nombre;
    }

    /**
     * Method getCliente. Accesor que devuelve el String
     correspondiente al cliente de una
     * imagen marcada.
     * @return String - nombre del cliente
     */
    public String getCliente(){
        return this.Cliente;
    }

    /**
     * Method getCodigo. Accesor que devuelve el entero
     correspondiente al codigo de una
     * imagen marcada.
     * @return entero - codigo de la marca
     */
    public int getCodigo(){
        return this.Codigo;
    }

    /**
     * Method getMarca. Accesor que devuelve el String
     correspondiente a la marca de una
     * imagen marcada.
     * @return String - marca
     */
    public String getMarca(){
        return this.Marca;
    }

    /**
     * Method setNombre. Accesor que establece el nombre original de
     una imagen marcada
     * @param soloNombre - nombre original de la imagen
     */
    public void setNombre(String soloNombre) {
        this.Nombre = soloNombre;
    }

    /**
     * Method toString. Redefinicion del método original para
     devolver un string en el formato
     * requerido por la aplicacion.
     * @return String - datos de la imagen marcada
     */
    public String toString(){

        StringBuffer linea = new StringBuffer();

        linea.append(this.Nombre.trim());
        linea.append(",");
        linea.append( new Integer(this.Codigo).toString());
    }
}
```

```

        linea.append(",");
        linea.append(this.Cliente.trim());
        linea.append(",");
        linea.append(this.Marca.trim());

        return linea.toString();
    }

    /**
     * Method equals. Redefinición del método original. Dos imagenM
     son iguales si coinciden su nombre
     * de imagen original y su código de marca.
     * @param img - ImagenM para la comparación.
     * @return boolean - cierto si img es coincide con la actual,
     falso en caso contrario.
     */
    public boolean equals(ImagenM img){

        return (img.getNombre().equalsIgnoreCase(this.Nombre) &&
img.getCodigo() == this.Codigo);
    }

}

```

B.13 Clase iObserver.java

```

// Rutinas de Dorothea Blostein
// from http://www.cs.queensu.ca/home/blostein/java.html
// CISC 124 Introduction to Computing Science II. Last taught Fall
2002.
// Here is Java image manipulation code used in this course.
// This code can be used as a starting point by anyone wishing to
write
// image manipulations in Java.

import java.awt.image.ImageObserver;
import java.awt.*;

public class iObserver implements ImageObserver {

    public boolean imageUpdate (Image img, int infoflags,
        int x, int y, int width, int height) {
        return true;
    }

}

```

B.14 Clase PanelDatos.java

```

import java.awt.*;
import javax.swing.*;

/**
 * @author Pedro Reverte
 *
 * Implementa el panel utilizado para mostrar los datos de imagenes
marcadas. Los datos
 * mostrados son: nombre de la imagen original correspondiente a la
marcada, código
 * utilizado para la marca, cliente para el que se marco la imagen y
marca insertada.

```

```

*
* 2004
*/
public class PanelDatos extends JPanel {

    GridBagConstraints c = new GridBagConstraints();
    JLabel label1;
    JLabel label2;
    JLabel label3;
    JLabel label4;
    JTextArea textarea;
    JScrollPane scrollPane;

    /**
     * Constructor PanelDatos.
     * @param misDatos - datos necesarios para inicializar el panel
     */
    PanelDatos(GestorDatos misDatos){

        label1 = new JLabel("Imagen Original");
        label2 = new JLabel("Codigo");
        label3 = new JLabel("Cliente");
        label4 = new JLabel("Marca");
        label1.setHorizontalAlignment(JLabel.LEFT);
        label1.setPreferredSize(new Dimension(250,10));
        label2.setHorizontalAlignment(JLabel.LEFT);
        label2.setPreferredSize(new Dimension(50,10));
        label3.setHorizontalAlignment(JLabel.LEFT);
        label3.setPreferredSize(new Dimension(130,10));

        textarea = new JTextArea("", 10, 80);
        textarea.setFont(new Font("MonoSpaced", Font.PLAIN, 12));
        textarea.setEditable(false);
        scrollPane = new JScrollPane(textarea);
        this.setLayout(new GridBagLayout());
        // los componentes crecen en las dos dimensiones
        c.fill = GridBagConstraints.BOTH;
        c.insets = new Insets(5,5,5,5);
        // inserta los datos en el panel
        resetDatos(misDatos);

        c.gridx = 0; c.gridy = 0; c.gridwidth = 5; c.gridheight=1;
        c.weightx = c.weighty = 0.0;
        this.add(label1, c);

        c.gridx = 5; c.gridy = 0; c.gridwidth = 3; c.gridheight=1;
        c.weightx = c.weighty = 0.0;
        this.add(label2, c);

        c.gridx = 8; c.gridy = 0; c.gridwidth = 5; c.gridheight=1;
        c.weightx = c.weighty = 0.0;
        this.add(label3, c);

        c.gridx = 13; c.gridy = 0; c.gridwidth = 3; c.gridheight=1;
        c.weightx = c.weighty = 0.0;
        this.add(label4, c);

        c.gridx = 0; c.gridy = 2; c.gridwidth = 16;
        c.gridheight=10;
        c.weightx = c.weighty = 1.0;

        this.add(scrollPane, c);

    }

    /**
     * Method resetDatos. Establece los datos presentados en el
     panel de acuerdo con el
     * contenido del gestor de datos
    */
}

```

```

        * @param misDatos - Gestor de datos
        */
        public void resetDatos(GestorDatos misDatos) {
            textarea.setText(misDatos.getTextoDatos());
        }
    }
}

```

B.15 Clase PanelExtraccion.java

```

import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;

/**
 * @author Pedro Reverte
 *
 * Implementa el panel utilizado para la extraccion de marcas. El
 * panel esta formado por la imagen
 * marcada y diversas etiquetas representando los siguientes datos:
 * nombre de la imagen marcada
 * y original, marca presente, codigo correspondiente (si se halla),
 * cliente correspondiente
 * (si existe). Finalmente existe un checkbox utilizado para indicar
 * que la imagen es un recorte
 * y no una imagen completa, y otro para indicar que deseamos extraer
 * una única marca
 *
 * 2004
 */
public class PanelExtraccion extends JPanel{

    GridBagConstraints c = new GridBagConstraints();
    JLabel label1;
    JLabel label2;
    JLabel label3;
    JLabel label4;
    JLabel label5;
    JLabel label6;
    JLabel label7;
    JLabel label8;
    JLabel label9;
    JLabel label10;
    JLabel imagenMarcada;
    JCheckBox checkbox;
    JCheckBox checkbox2;

    /**
     * Constructor PanelExtraccion.
     * @param misDatos - datos necesarios para inicializar el panel
     */
    public PanelExtraccion(GestorDatos misDatos) {

        imagenMarcada = new JLabel();
        imagenMarcada.setHorizontalAlignment(JLabel.CENTER);
        imagenMarcada.setPreferredSize(new Dimension(350,300));

        TitledBorder titled = new TitledBorder("Imagen Marcada");
        imagenMarcada.setBorder(titled);
        label1 = new JLabel("Imagen Marcada:");
        label2 = new JLabel("Imagen Original:");
        label3 = new JLabel("Marca presente:");
        label4 = new JLabel("Codigo obtenido:");
        label5 = new JLabel("Cliente:");
        label1.setHorizontalAlignment(JLabel.LEFT);
        label1.setPreferredSize(new Dimension(100,15));
    }
}

```

```

label2.setHorizontalAlignment (JLabel.LEFT);
label2.setPreferredSize (new Dimension (100, 15));
label3.setHorizontalAlignment (JLabel.LEFT);
label3.setPreferredSize (new Dimension (100, 15));
label4.setHorizontalAlignment (JLabel.LEFT);
label4.setPreferredSize (new Dimension (100, 15));
label5.setHorizontalAlignment (JLabel.LEFT);
label5.setPreferredSize (new Dimension (100, 15));
label6 = new JLabel ("-");
label7 = new JLabel ("-");
label8 = new JLabel ("-");
label9 = new JLabel ("-");
label10 = new JLabel ("-");
label6.setHorizontalAlignment (JLabel.LEFT);
label6.setPreferredSize (new Dimension (200, 15));
label7.setHorizontalAlignment (JLabel.LEFT);
label7.setPreferredSize (new Dimension (200, 15));
label8.setHorizontalAlignment (JLabel.LEFT);
label8.setPreferredSize (new Dimension (200, 15));
label9.setHorizontalAlignment (JLabel.LEFT);
label9.setPreferredSize (new Dimension (200, 15));
label10.setHorizontalAlignment (JLabel.LEFT);
label10.setPreferredSize (new Dimension (200, 15));

checkbox = new JCheckBox ("Recorte");
checkbox2 = new JCheckBox ("Marca Unica");

this.setLayout (new GridBagLayout ());
// los componentes crecen en las dos dimensiones
c.fill = GridBagConstraints.BOTH;
c.insets = new Insets (5, 5, 5, 5);
// inserta los datos en el panel
resetDatos (misDatos);

c.gridx = 0; c.gridy = 0; c.gridwidth = 3; c.gridheight=16;
c.weightx = c.weighty = 1.0;
this.add (imagenMarcada, c);

c.gridx = 4; c.gridy = 1; c.gridwidth = 1; c.gridheight=1;
c.weightx = c.weighty = 0.0;
this.add (label1, c);

c.gridx = 5; c.gridy = 1; c.gridwidth = 1; c.gridheight=1;
c.weightx = c.weighty = 0.0;
this.add (label6, c);

c.gridx = 4; c.gridy = 2; c.gridwidth = 1; c.gridheight=2;
c.weightx = c.weighty = 0.0;
this.add (label2, c);

c.gridx = 5; c.gridy = 2; c.gridwidth = 1; c.gridheight=2;
c.weightx = c.weighty = 0.0;
this.add (label7, c);

c.gridx = 4; c.gridy = 4; c.gridwidth = 1; c.gridheight=1;
c.weightx = c.weighty = 0.0;
this.add (label3, c);

c.gridx = 5; c.gridy = 4; c.gridwidth = 1; c.gridheight=1;
c.weightx = c.weighty = 0.0;
this.add (label8, c);

c.gridx = 4; c.gridy = 5; c.gridwidth = 1; c.gridheight=2;
c.weightx = c.weighty = 0.0;
this.add (label4, c);

c.gridx = 5; c.gridy = 5; c.gridwidth = 1; c.gridheight=2;
c.weightx = c.weighty = 0.0;
this.add (label9, c);

```

```

        c.gridx = 4; c.gridy = 7; c.gridwidth = 1; c.gridheight=2;
        c.weightx = c.weighty = 0.0;
        this.add(label5, c);

        c.gridx = 5; c.gridy = 7; c.gridwidth = 1; c.gridheight=2;
        c.weightx = c.weighty = 0.0;
        this.add(label10, c);

        c.gridx = 4; c.gridy = 9; c.gridwidth = 1; c.gridheight=1;
        c.weightx = c.weighty = 0.0;
        this.add(checkbox, c);

        c.gridx = 4; c.gridy = 10; c.gridwidth = 1; c.gridheight=1;
        c.weightx = c.weighty = 0.0;
        this.add(checkbox2, c);
    }

    /**
     * Method resetDatos. Establece los datos presentados en el
panel de acuerdo con el
     * contenido del gestor de datos
     * @param misDatos - Gestor de datos
     */
    public void resetDatos(GestorDatos misDatos) {
nulos        // si no existe una imagen seleccionada presenta datos

        if(misDatos.getExtraida() == null) {
            imagenMarcada.setText("Imagen no seleccionada");
            imagenMarcada.setIcon(null);
            label6.setText("Sin nombre");
            label7.setText("-");
            label8.setText("-");
            label9.setText("-");
            label10.setText("-");
        }
        // si existe imagen presenta sus datos correspondientes
        else{
            imagenMarcada.setIcon(misDatos.getExtraida());
            imagenMarcada.setText("");
            label6.setText(misDatos.getNameExtraida());
            label7.setText(misDatos.getNameOriginalExtraida());

            label8.setText(misDatos.getMarcaExtraida());
            label9.setText(misDatos.getCodigoExtraida());
            label10.setText(misDatos.getClienteExtraida());
        }
    }

    /**
     * Method getRecorte. Comprueba si esta seleccionado el checkbox
de recorte.
     * @return cierto si el recorte ha sido seleccionado, falso en
caso contrario
     */
    public boolean getRecorte() {
        if(checkbox.isSelected())
            return true;
        else
            return false;
    }

    /**
     * Method getUnica. Comprueba si esta seleccionado el checkbox
de marca unica.
    */

```

```

    * @return cierto si el recorte ha sido seleccionado, falso en
    caso contrario
    */
    public boolean getUnica() {
        if (checkbox2.isSelected())
            return true;
        else
            return false;
    }
}

```

B.16 Clase PanelInsercion.java

```

import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;

/**
 * @author Pedro Reverte
 * Implementa el panel utilizado para la insercion de marcas. El panel
 * esta formado por las imagenes
 * marcada y original y diversas etiquetas representando los
 * siguientes datos: nombre de la imagen marcada
 * y original, cliente y codigo de la marca.
 *
 * 2004
 */
public class PanelInsercion extends JPanel {

    GridBagConstraints c = new GridBagConstraints();

    JLabel imagen = new JLabel();
    JLabel imagenMarcada = new JLabel();

    JLabel label1 = new JLabel("Sin nombre");
    JLabel label2 = new JLabel("Sin nombre");
    JLabel label3 = new JLabel("Cliente:");
    JLabel label4 = new JLabel("-");
    JLabel label5 = new JLabel("Código:");
    JLabel label6 = new JLabel("-");

    /**
     * Constructor PanelInsercion.
     * @param misDatos - Datos necesarios para generar el panel
     */
    PanelInsercion(GestorDatos misDatos) {

        this.setLayout(new GridBagLayout());
        // los componentes crecen en las dos dimensiones
        c.fill = GridBagConstraints.BOTH;
        c.insets = new Insets(5,5,5,5);

        imagen.setHorizontalAlignment(JLabel.CENTER);
        TitledBorder titled = new TitledBorder("Imagen Original");
        imagen.setBorder(titled);
        imagen.setPreferredSize(new Dimension(350,300));
        imagenMarcada.setHorizontalAlignment(JLabel.CENTER);
        TitledBorder titled2 = new TitledBorder("Imagen Marcada");
        imagenMarcada.setBorder(titled2);
        imagenMarcada.setPreferredSize(new Dimension(350,300));
        // inserta los datos en el panel
        resetDatos(misDatos);

        c.gridx = 0; c.gridy = 0; c.gridwidth = 3; c.gridheight=7;
        c.weightx = c.weighty = 1.0;
        this.add(imagen, c);

        c.gridx = 5; c.gridy = 0; c.gridwidth = 3; c.gridheight=7;

```

```

        c.weightx = c.weighty = 1.0;
        this.add(imagenMarcada, c);

        c.gridx = 0; c.gridy = 8; c.gridwidth = 3; c.gridheight=1;
        c.weightx = c.weighty = 0.0;
        this.add(label1, c);

        c.gridx = 5; c.gridy = 8; c.gridwidth = 3; c.gridheight=1;
        c.weightx = c.weighty = 0.0;
        this.add(label2, c);

        c.gridx = 5; c.gridy = 9; c.gridwidth = 1; c.gridheight=1;
        c.weightx = c.weighty = 0.0;
        this.add(label3, c);

        c.gridx = 6; c.gridy = 9; c.gridwidth = 2; c.gridheight=1;
        c.weightx = c.weighty = 0.0;
        this.add(label4, c);

        c.gridx = 5; c.gridy = 10; c.gridwidth = 1; c.gridheight=1;
        c.weightx = c.weighty = 0.0;
        this.add(label5, c);

        c.gridx = 6; c.gridy = 10; c.gridwidth = 2; c.gridheight=1;
        c.weightx = c.weighty = 0.0;
        this.add(label6, c);
    }

    /**
     * Method resetDatos. Establece los datos presentados en el
panel de acuerdo con el
     * contenido del gestor de datos
     * @param misDatos - Gestor de datos
     */
    public void resetDatos(GestorDatos misDatos){

        // si no existe imagen original, datos nulos
        if(misDatos.getUncryp() == null){
            imagen.setText("Imagen no seleccionada");
            imagen.setIcon(null);
            label1.setText("Sin nombre");
        }
        // si existe se establecen los datos del GestorDatos
        else{
            imagen.setIcon(misDatos.getUncryp());
            imagen.setText("");
            label1.setText(misDatos.getNameImagen());
        }
        // lo mismo para la imagen marcada
        if(misDatos.getCryp() == null){
            imagenMarcada.setText("Imagen no seleccionada");
            imagenMarcada.setIcon(null);
            label2.setText("Sin nombre");
            label4.setText("-");
            label6.setText("-");
        }
        else{
            imagenMarcada.setIcon(misDatos.getCryp());
            imagenMarcada.setText("");
            label2.setText(misDatos.getNameMarcada());
            label4.setText(misDatos.getClienteMarcada());

            label6.setText(misDatos.getCodigoMarcada());
        }
    }
}

```


B.17 Clase utils.java

```
// Rutinas de Dorothea Blostein
// from http://www.cs.queensu.ca/home/blostein/java.html
// CISC 124 Introduction to Computing Science II. Last taught Fall
2002.
// Here is Java image manipulation code used in this course.
// This code can be used as a starting point by anyone wishing to
write
// image manipulations in Java.

import java.awt.*;
import java.awt.image.*;
import java.io.*;
import java.util.*;
import java.util.Vector.*;

public class utils{

    /*loadbitmap originally written by Jeff West, and published at
    *http://www.javaworld.com/javaworld/javatips/jw-javatip43.html
    *modified slightly by Jeb Thorley.
    */

    public static Image loadbitmap (String sdir, String sfile)
    {
        Image image;
        // System.out.println("loading:"+sdir+sfile);
        try
        {
            FileInputStream fs;
            if(sdir.equals("./")){//a bit of a hack
                fs=new FileInputStream(sfile);
            }
            else{
                fs=new FileInputStream(sdir+sfile);
            }

            int bflen=14; // 14 byte BITMAPFILEHEADER
            byte bf []=new byte[bflen];
            fs.read(bf,0,bflen);
            int bilen=40; // 40-byte BITMAPINFOHEADER
            byte bi []=new byte[bilen];
            fs.read(bi,0,bilen);

            // Interpret data.
            int nsize = (((int)bf[5]&0xff)<<24)
                | (((int)bf[4]&0xff)<<16)
                | (((int)bf[3]&0xff)<<8)
                | (int)bf[2]&0xff;
            //System.out.println("File type is
: "+(char)bf[0]+(char)bf[1]);
            //System.out.println("Size of file is :"+nsize);

            int nbisize = (((int)bi[3]&0xff)<<24)
                | (((int)bi[2]&0xff)<<16)
                | (((int)bi[1]&0xff)<<8)
                | (int)bi[0]&0xff;
            //System.out.println("Size of bitmapinfoheader is
: "+nbisize);

            int nwidth = (((int)bi[7]&0xff)<<24)
                | (((int)bi[6]&0xff)<<16)
                | (((int)bi[5]&0xff)<<8)
                | (int)bi[4]&0xff;
```

```

//System.out.println("Width is :"+nwidth);

int nheight = (((int)bi[11]&0xff)<<24)
| (((int)bi[10]&0xff)<<16)
| (((int)bi[9]&0xff)<<8)
| (int)bi[8]&0xff;
//System.out.println("Height is :"+nheight);

int nplanes = (((int)bi[13]&0xff)<<8) |
(int)bi[12]&0xff;
//System.out.println("Planes is :"+nplanes);

int nbitcount = (((int)bi[15]&0xff)<<8) |
(int)bi[14]&0xff;
//System.out.println("BitCount is :"+nbitcount);

// Look for non-zero values to indicate compression
int ncompression = (((int)bi[19])<<24)
| (((int)bi[18])<<16)
| (((int)bi[17])<<8)
| (int)bi[16];
//System.out.println("Compression is :"+ncompression);

int nsizeimage = (((int)bi[23]&0xff)<<24)
| (((int)bi[22]&0xff)<<16)
| (((int)bi[21]&0xff)<<8)
| (int)bi[20]&0xff;
//System.out.println("SizeImage is :"+nsizeimage);

int nxpm = (((int)bi[27]&0xff)<<24)
| (((int)bi[26]&0xff)<<16)
| (((int)bi[25]&0xff)<<8)
| (int)bi[24]&0xff;
//System.out.println("X-Pixels per meter is :"+nxpm);

int nypm = (((int)bi[31]&0xff)<<24)
| (((int)bi[30]&0xff)<<16)
| (((int)bi[29]&0xff)<<8)
| (int)bi[28]&0xff;
//System.out.println("Y-Pixels per meter is :"+nypm);

int nclrused = (((int)bi[35]&0xff)<<24)
| (((int)bi[34]&0xff)<<16)
| (((int)bi[33]&0xff)<<8)
| (int)bi[32]&0xff;
//System.out.println("Colors used are :"+nclrused);

int nclrimp = (((int)bi[39]&0xff)<<24)
| (((int)bi[38]&0xff)<<16)
| (((int)bi[37]&0xff)<<8)
| (int)bi[36]&0xff;
//System.out.println("Colors important are
:"+nclrimp);

if (nbitcount==24)
{
// No Palatte data for 24-bit format but scan
lines are
// padded out to even 4-byte boundaries.
int npad = (nsizeimage / nheight) - nwidth *
3;

//added for Bug correction
if(npad == 4){
npad=0;
}
int ndata[] = new int [nheight * nwidth];
byte brgb[] = new byte [( nwidth + npad) * 3 *
nheight];

```

```

nheight);
        fs.read (brgb, 0, (nwidth + npad) * 3 *
int nindex = 0;
for (int j = 0; j < nheight; j++)
    {
        for (int i = 0; i < nwidth; i++)
            {
                ndata [nwidth * (nheight - j -
1) + i] =
                    (255&0xff)<<24
                    |
                    ((int)brgb [nindex+2]&0xff)<<16)
                    |
                    ((int)brgb [nindex+1]&0xff)<<8)
                    | (int)brgb [nindex]&0xff;
                nindex += 3;
            }
        nindex += npad;
    }

    image
=Toolkit.getDefaultToolkit().createImage( new MemoryImageSource
(nwidth, nheight, ndata, 0, nwidth));
    }
    else if (nbitcount == 8)
    {
        // Have to determine the number of colors, the
        // parameter is dominant if it is greater than
        // zero, calculate colors based on
        // zero, calculate colors based on
        // zero, calculate colors based on
        int nNumColors = 0;
        if (nclrused > 0)
            {
                nNumColors = nclrused;
            }
        else
            {
                nNumColors = (1&0xff)<<nbitcount;
            }
        //System.out.println("The number of Colors
is"+nNumColors);

        // Some bitmaps do not have the sizeimage
        // Ferret out these cases and fix 'em.
        if (nsizeimage == 0)
            {
                nsizeimage = (((nwidth*nbitcount)+31)
& ~31 ) >> 3);
                nsizeimage *= nheight;
                //System.out.println("nsizeimage
(backup) is"+nsizeimage);
            }

        // Read the palatte colors.
        int npalette[] = new int [nNumColors];
        byte bpalette[] = new byte [nNumColors*4];
        fs.read (bpalette, 0, nNumColors*4);
        int nindex8 = 0;
        for (int n = 0; n < nNumColors; n++)
            {
                npalette [n] = (255&0xff)<<24
                |
                ((int)bpalette [nindex8+2]&0xff)<<16)
                |
                ((int)bpalette [nindex8+1]&0xff)<<8)
                | (int)bpalette [nindex8]&0xff;
            }
    }
}

```

```

        nindex8 += 4;
    }
    // Read the image data (actually indices into
the palette)
    // Scan lines are still padded out to even 4-
byte
    // boundaries.
    int npad8 = (nsizeimage / nheight) - nwidth;
    //System.out.println("nPad is:"+npad8);

    int ndata8[] = new int [nwidth*nheight];
    byte bdata[] = new byte
[(nwidth+npad8)*nheight];
    fs.read (bdata, 0, (nwidth+npad8)*nheight);
    nindex8 = 0;
    for (int j8 = 0; j8 < nheight; j8++)
        {
            for (int i8 = 0; i8 < nwidth; i8++)
                {
                    ndata8 [nwidth*(nheight-j8-
1)+i8] =
                        npalette
                        [((int)bdata [nindex8]&0xff)];
                    nindex8++;
                }
            nindex8 += npad8;
        }

    image =
Toolkit.getDefaultToolkit().createImage( new MemoryImageSource
(nwidth, nheight,ndata8, 0, nwidth));
    }
    else
    {
        //
        System.out.println ("Not a 24-bit or 8-bit
Windows Bitmap, aborting...");
        image = (Image)null;
    }

    fs.close();
    return image;
}
catch (Exception e)
{
    //
    System.out.println("Caught exception in
loadbitmap!");
}
return (Image)null;
}

public static int[] getPixels(Image parImage, int parWidth, int
parHeight) {
    int[] bitmap = new int [parWidth * parHeight];
    PixelGrabber pg = new PixelGrabber (parImage, 0, 0, parWidth,
parHeight,
                                bitmap, 0, parWidth);

    try {
        pg.grabPixels ();
    }
    catch (InterruptedException e) {
        e.printStackTrace ();
    }
    return bitmap;
}

/**

```

```

    * Las siguientes rutinas han sido obtenidas del paquete
    GraphixTools de Stephen Manley, y modificadas
    * ligeramente para adaptarlas a las necesidades de este
    proyecto.
    * <b>GraphixTools - A collection of handy graphics
functions</b>
    * <br><br>
    * This is a collection of graphics mundanes that I had to cut
and
    * paste one too many times. While laziness on my part I think
that
    * I was doing this stuff enough to warrant packaging up these
goodies
    * into their own proper class. Hopefully this will evolve into
something
    * that's worth releasing.
    * <br><br>
    * <b>Feature list:</b><br>
    * Array returns (R,G,B).<br>
    * E-Z Image Producing.<br>
    * RGB to YUV colorspace conversion.<br>
    * YUV to RGB colorspace conversion.<br>
    * Filesize and other information<br>
    * Free & Used VM Machine Memory Info<br>
    * <br><br>
    * You can get the <a href="GrafixTools.java">source code</a>
here.
    * @author <a
href="http://eagle.uccb.ns.ca/steve/home.html">Stephen Manley</a>
(smanley@eagle.uccb.ns.ca)
    * @version 0.9.7 August 20th 1996
    */

/**
 * This method converts an RGB colorspace to a YUV colorspace.
(YUV is more
 * technically correct when referred to a YCbCr colorspace.) YCbCr
was developed
 * according to recommondation ITU-R BT.601 (Formerly: CCIR 601)
to develop
 * a world-wide digital video standard. It is commonly used in
many streaming
 * video and compression routines, such as <a
href="http://www.mpeg.org">MPEG</a>
 * and <a href="http://www.telenor.nl">H.263.</a>
 * <br><br>
 * This routine handles dealing with 0..255 values per pixel, the
more common
 * PC format. The passed arrays should all have the same bounds,
corresponding
 * to the image dimensions.
 * @param redArray An array of 0..255 red pixel values.
 * @param greenArray An array of 0..255 green pixel values.
 * @param blueArray An array of 0..255 blue pixel values.
 * @param bloqueWidth entero
 * @param bloqueHeight entero
 * @returns Y The Y component
 */
public static int[][] convertRGBtoY(int[][] redArray, int[][]
greenArray, int[][] blueArray, int bloqueWidth, int bloqueHeight)
{
    int i;
    int j;
    int Y[][] = new int[bloqueWidth][bloqueHeight];

    for (i=0; i<bloqueWidth; i++)
    {
        for (j=0; j<bloqueHeight; j++)

```

```

        {
            Y[i][j] = (int)((0.257 * (float)redArray[i][j]) + (0.504
* (float)greenArray[i][j]) + (0.098 * (float)blueArray[i][j]) + 16.0);
        }
    }

    return Y;
}

/**
 * This method converts an RGB colorspace to a YUV colorspace.
 * (YUV is more
 * technically correct when referred to a YCbCr colorspace.) YCbCr
 * was developed
 * according to recommondation ITU-R BT.601 (Formerly: CCIR 601)
 * to develop
 * a world-wide digital video standard. It is commonly used in
 * many streaming
 * video and compression routines, such as <a
 * href="http://www.mpeg.org">MPEG</a>
 * and <a href="http://www.telenor.nl">H.263.</a>
 * <br><br>
 * This routine handles dealing with 0..255 values per pixel, the
 * more common
 * PC format. The passed arrays should all have the same bounds,
 * corresponding
 * to the image dimensions.
 * @param redArray An array of 0..255 red pixel values.
 * @param greenArray An array of 0..255 green pixel values.
 * @param blueArray An array of 0..255 blue pixel values.
 * @returns Cb The Cb component.
 */
public static int[][] convertRGBtoCb(int[][] redArray, int[][]
greenArray, int[][] blueArray, int bloqueWidth, int bloqueHeight)
{
    int i;
    int j;
    int Cb[][] = new int[bloqueWidth][bloqueHeight];

    for (i=0; i<bloqueWidth; i++)
    {
        for (j=0; j<bloqueHeight; j++)
        {
            Cb[i][j] = (int)((-0.148 * (float)redArray[i][j]) -
(0.291 * (float)greenArray[i][j]) + (0.439 * (float)blueArray[i][j]) +
128.0);
        }
    }

    return Cb;
}

/**
 * This method converts an RGB colorspace to a YUV colorspace.
 * (YUV is more
 * technically correct when referred to a YCbCr colorspace.) YCbCr
 * was developed
 * according to recommondation ITU-R BT.601 (Formerly: CCIR 601)
 * to develop
 * a world-wide digital video standard. It is commonly used in
 * many streaming
 * video and compression routines, such as <a
 * href="http://www.mpeg.org">MPEG</a>
 * and <a href="http://www.telenor.nl">H.263.</a>
 * <br><br>
 * This routine handles dealing with 0..255 values per pixel, the
 * more common
 * PC format. The passed arrays should all have the same bounds,
 * corresponding

```

```

    * to the image dimensions.
    * @param redArray An array of 0..255 red pixel values.
    * @param greenArray An array of 0..255 green pixel values.
    * @param blueArray An array of 0..255 blue pixel values.
    * @returns Cr The Cr component.
    */
    public static int[][] convertRGBtoCr(int[][] redArray, int[][]
greenArray, int[][] blueArray, int bloqueWidth, int bloqueHeight)
    {
        int i;
        int j;
        int Cr[][] = new int[bloqueWidth][bloqueHeight];

        for (i=0; i<bloqueWidth; i++)
        {
            for (j=0; j<bloqueHeight; j++)
            {
                Cr[i][j] = (int)((0.439 * (float)redArray[i][j]) -
(0.368 * (float)greenArray[i][j]) - (0.071 * (float)blueArray[i][j]) +
128.0);
            }
        }

        return Cr;
    }

    /**
     * This method converts a YUV (aka YCbCr) colorspace to a RGB
     colorspace.
     * This is handy when trying to reconstruct an image in Java from
     YCbCr transmitted
     * data. This routine expects the data to fall in the standard PC
     0..255 range
     * per pixel, with the array dimensions corresponding to the
     imageWidth and imageHeight.
     * These variables are either set manually in the case of a null
     constructor,
     * or they are automatically calculated from the image parameter
     constructor.
     * @param Y The Y component set.
     * @param Cb The Cb component set.
     * @param Cr The Cr component set.
     * @returns R The R component.
     */
    public static int[][] convertYCbCrtoR(int[][] Y, int[][] Cb,
int[][] Cr, int bloqueWidth, int bloqueHeight)
    {
        int i;
        int j;
        int R[][] = new int[bloqueWidth][bloqueHeight];

        for (i=0; i<bloqueWidth; i++)
        {
            for (j=0; j<bloqueHeight; j++)
            {
                R[i][j] = (int)((1.164*((float)Y[i][j]-16.0)) +
(1.596*((float)Cb[i][j] - 128.0)));
                // comprobamos que los valores no se salgan del rango
                (0-255)
                if(R[i][j] < 0)
                    R[i][j] = 0;
                if(R[i][j] > 255)
                    R[i][j] = 255;
            }
        }

        return R;
    }
}

```

```

    /**
     * This method converts a YUV (aka YCbCr) colorspace to a RGB
     colorspace.
     * This is handy when trying to reconstruct an image in Java from
     YCbCr transmitted
     * data. This routine expects the data to fall in the standard PC
     0..255 range
     * per pixel, with the array dimensions corresponding to the
     imageWidth and imageHeight.
     * These variables are either set manually in the case of a null
     constructor,
     * or they are automatically calculated from the image parameter
     constructor.
     * @param Y The Y component set.
     * @param Cb The Cb component set.
     * @param Cr The Cr component set.
     * @returns G The G component.
     */
    public static int[][] convertYCbCrtoG(int[][] Y, int[][] Cb,
    int[][] Cr, int bloqueWidth, int bloqueHeight)
    {
        int i;
        int j;
        int G[][] = new int[bloqueWidth][bloqueHeight];

        for (i=0; i<bloqueWidth; i++)
        {
            for (j=0; j<bloqueHeight; j++)
            {
                G[i][j] = (int)((1.164*((float)Y[i][j]-16.0)) -
    (0.813*((float)Cr[i][j] - 128.0)) - (0.392*((float)Cb[i][j] -
    128.0)));
                // comprobamos que los valores no se salgan del rango
    (0-255)
                if(G[i][j] < 0)
                    G[i][j] =0;
                if(G[i][j] > 255)
                    G[i][j] = 255;
            }
        }
        return G;
    }

    /**
     * This method converts a YUV (aka YCbCr) colorspace to a RGB
     colorspace.
     * This is handy when trying to reconstruct an image in Java from
     YCbCr transmitted
     * data. This routine expects the data to fall in the standard PC
     0..255 range
     * per pixel, with the array dimensions corresponding to the
     imageWidth and imageHeight.
     * These variables are either set manually in the case of a null
     constructor,
     * or they are automatically calculated from the image parameter
     constructor.
     * @param Y The Y component set.
     * @param Cb The Cb component set.
     * @param Cr The Cr component set.
     * @returns B The B component.
     */
    public static int[][] convertYCbCrtoB(int[][] Y, int[][] Cb,
    int[][] Cr, int bloqueWidth, int bloqueHeight)
    {
        int i;
        int j;
        int B[][] = new int[bloqueWidth][bloqueHeight];

```



```
        for (i=0; i<bloqueWidth; i++)
        {
            for (j=0; j<bloqueHeight; j++)
            {
                B[i][j] = (int) (((1.164*((float)Y[i][j]-16.0))) +
(2.017*((float)Cb[i][j] - 128.0)));
                // comprobamos que los valores no se salgan del rango
(0-255)
                if(B[i][j] < 0)
                    B[i][j] =0;
                if(B[i][j] > 255)
                    B[i][j] = 255;
            }
        }
        return B;
    }
}
```

C. Código fuente ApplConfabula.

C.1 Clase AboutBox.java

```
import java.awt.*;
```

```

import java.awt.event.*;
import javax.swing.*;

/**
 * @author Pedro Reverte
 *
 * Implementa una ventana para presentar los créditos de la aplicación
 *
 * 2004
 */
public class AboutBox extends JDialog implements ActionListener {

    JPanel panel1 = new JPanel();
    JPanel panel2 = new JPanel();
    JPanel insetsPanel1 = new JPanel();
    JPanel insetsPanel2 = new JPanel();
    JPanel insetsPanel3 = new JPanel();
    JButton button1 = new JButton();
    JLabel imageLabel = new JLabel();
    JLabel label1 = new JLabel();
    JLabel label2 = new JLabel();
    JLabel label3 = new JLabel();
    JLabel label4 = new JLabel();
    JLabel label5 = new JLabel();
    JLabel label6 = new JLabel();
    JLabel label7 = new JLabel();
    JLabel label8 = new JLabel();
    BorderLayout borderLayout1 = new BorderLayout();
    BorderLayout borderLayout2 = new BorderLayout();
    FlowLayout flowLayout1 = new FlowLayout();
    GridLayout gridLayout1 = new GridLayout();
    String producto = "ApplConfabula - Ataques por confabulacion";
    String version = "Version 1.0";
    String copyright = "Pedro Reverte - 2004";
    String linea = "ApplConfabula ha sido realizada como parte de";
    String linea2 = " un TFC Seguretat Informàtica de ETIS para la";
    String linea3 = "Universitat Oberta de Catalunya";
    String linea4 = "(http://www.uoc.es)";

    /**
     * Constructor. Inicializa la ventana.
     * @param parent Frame padre para presentar la ventana.
     */
    public AboutBox(Frame parent) {
        super(parent);
        enableEvents(AWTEvent.WINDOW_EVENT_MASK);
        try {
            jbInit();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        pack();
    }

    /**
     * Method jbInit. Inicializa los datos de la ventana
     */
    private void jbInit() throws Exception {
        // imageLabel.setIcon(new
        ImageIcon(AboutBox.class.getResource("prevertem.jpg"));
        this.setTitle("About");
        setResizable(false);
        panel1.setLayout(borderLayout1);
        panel2.setLayout(borderLayout2);
        insetsPanel1.setLayout(flowLayout1);
        insetsPanel2.setLayout(flowLayout1);
        insetsPanel2.setBorder(BorderFactory.createEmptyBorder(10, 10,
10, 10));

```

```

        gridLayout1.setRows(8);
        gridLayout1.setColumns(1);
        label1.setText(producto);
        label2.setText(version);
        label3.setText(copyright);
        label4.setText("");
        label5.setText(linea);
        label6.setText(linea2);
        label7.setText(linea3);
        label8.setText(linea4);
        insetsPanel3.setLayout(gridLayout1);
        insetsPanel3.setBorder(BorderFactory.createEmptyBorder(10, 60,
10, 10));
        button1.setText("Ok");
        button1.addActionListener(this);
        insetsPanel2.add(imageLabel, null);
        panel2.add(insetsPanel2, BorderLayout.WEST);
        this.getContentPane().add(panel1, null);
        insetsPanel3.add(label1, null);
        insetsPanel3.add(label2, null);
        insetsPanel3.add(label3, null);
        insetsPanel3.add(label4, null);
        insetsPanel3.add(label5, null);
        insetsPanel3.add(label6, null);
        insetsPanel3.add(label7, null);
        insetsPanel3.add(label8, null);
        panel2.add(insetsPanel3, BorderLayout.CENTER);
        insetsPanel1.add(button1, null);
        panel1.add(insetsPanel1, BorderLayout.SOUTH);
        panel1.add(panel2, BorderLayout.NORTH);
    }

    /**
     * Method processWindowEvent. Maneja el cierre de la ventana
     * @param e - evento producido
     */
    protected void processWindowEvent(WindowEvent e) {
        if (e.getID() == WindowEvent.WINDOW_CLOSING) {
            cancel();
        }
        super.processWindowEvent(e);
    }

    /**
     * Method cancel. Cierra la ventana
     */
    private void cancel() {
        dispose();
    }

    /**
     * Method actionPerformed. Cierra la ventana cuando sucede un
    evento de boton
     */
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == button1) {
            cancel();
        }
    }
}

```

C.2 Clase ApplConfabula.java

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

```

```

import java.awt.image.renderable.*;
import java.io.*;
import java.util.*;

/**
 * @author Pedro Reverte
 *
 * Aplicación que permite generar una imagen confabulada a partir de
 la misma imagen marcada con dos codigos diferentes
 *
 * 2004
 */
public class ApplConfabula extends JFrame {

    JPanel contentPane;

    JMenuBar jMenuBar1 = new JMenuBar();

    JMenu jMenuConfabulacion = new JMenu();
    JMenuItem jMenuConfaGenerar = new JMenuItem();
    JMenuItem jMenuConfaGrabar = new JMenuItem();
    JMenuItem jMenuConfaSalir = new JMenuItem();

    JMenu jMenuHelp = new JMenu();
    JMenuItem jMenuHelpAbout = new JMenuItem();

    JToolBar jToolBar = new JToolBar();

    JButton jButtonGenerarConfa = new JButton();
    JButton jButtonGrabarConfa = new JButton();

    BorderLayout borderLayout1 = new BorderLayout();

    PanelConfabulacion panelConfabulacion;

    GestorConfa misDatos;
    Dialogos dialogo;

    /**
     * Constructor ApplConfabula
     */
    ApplConfabula() {

        // crea instancias de GestorDatos y dialogo
        dialogo = new Dialogos(this);
        if (misDatos == null)
            misDatos = new GestorConfa(this.dialogo);
        // crea el panel principal
        panelConfabulacion = new PanelConfabulacion(misDatos);
        try {
            // inicializa componentes
            jbInit();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * jbInit. Inicializa los componentes principales de la
 aplicación.
     */
    private void jbInit() throws Exception {

        contentPane = (JPanel) this.getContentPane();
        contentPane.setLayout(borderLayout1);
        this.setSize(750, 510);
    }
}

```

```

this.setTitle("ApplConfabulacion");

jMenuConfabulacion.setText("Confabulaciones");
jMenuConfaGenerar.setText("Generar confabulada");
jMenuConfaGenerar.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jMenuConfaGenerar_actionPerformed(e);
    }
});
jMenuConfaGrabar.setText("Grabar Imagen confabulada");
jMenuConfaGrabar.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jMenuConfaGrabar_actionPerformed(e);
    }
});

jMenuConfaSalir.setText("Salir");
jMenuConfaSalir.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jMenuConfaSalir_actionPerformed(e);
    }
});

jMenuHelp.setText("Help");
jMenuHelpAbout.setText("About");
jMenuHelpAbout.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jMenuHelpAbout_actionPerformed(e);
    }
});

jButtonGenerarConfa.setIcon(new ImageIcon("marcar.gif"));
jButtonGenerarConfa.addActionListener(new
java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jButtonGenerarConfa_actionPerformed(e);
    }
});
jButtonGenerarConfa.setToolTipText("Generar confabulada");

jButtonGrabarConfa.setIcon(new
ImageIcon(ApplConfabula.class.getResource("guardar.gif")));
jButtonGrabarConfa.addActionListener(new
java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jButtonGrabarConfa_actionPerformed(e);
    }
});
jButtonGrabarConfa.setToolTipText("Grabar confabulada");

jToolBar.add(jButtonGenerarConfa);
jToolBar.add(jButtonGrabarConfa);

jMenuConfabulacion.add(jMenuConfaGenerar);
jMenuConfabulacion.add(jMenuConfaGrabar);
jMenuConfabulacion.addSeparator();
jMenuConfabulacion.add(jMenuConfaSalir);

jMenuHelp.add(jMenuHelpAbout);

jMenuBar1.add(jMenuConfabulacion);
jMenuBar1.add(jMenuHelp);

this.setJMenuBar(jMenuBar1);
contentPane.add(jToolBar, BorderLayout.NORTH);
contentPane.add(panelConfabulacion, BorderLayout.CENTER);
}

```

```

    /**
     * Method jButtonGrabarConfa_actionPerformed. Implementa las
     acciones a realizar cuando
     * se pulsa el boton Grabar Confabulada. Muestra los resultados
     en pantalla
     * @param e Evento que se ha producido
     */
    private void jButtonGrabarConfa_actionPerformed(ActionEvent e) {

        // comprueba si existe imagen confabulada en el gestor de
datos
        if(misDatos.getImgConfa()==null){
            dialogo.mostrarMensaje("Grabar","No existe ninguna
imagen confabulada");
            return;
        }
        // pide el nombre para guardar la imagen
String name = dialogo.salvarFichero();
        if(name==null) //termina si se cancela
            return;
        // invoca la accion de grabar imagen confabulada en el
gestor de datos
        if(misDatos.grabarConfabulada(name, dialogo)){
            // si se ha grabado, presenta resultados en pantalla
            dialogo.mostrarMensaje("Grabar",name+" grabada");
            // resetea los datos del panel
            this.panelConfabulacion.resetDatos(misDatos);
        }
    }

    /**
     * Method jButtonGenerarConfa_actionPerformed. Implementa las
     acciones a realizar cuando
     * se pulsa el boton Generar Confabulada. Muestra los resultados
     en pantalla
     * @param e Evento que se ha producido
     */
    private void jButtonGenerarConfa_actionPerformed(ActionEvent e)
    {

        dialogo.mostrarMensaje("Generar Confabulada","Introduzca
los nombres de las dos imagenes");
        // pide el nombre de la primera imagen
String name = dialogo.abrirFichero();
        if(name==null){ // si se cancela termina
            // resetea las imagenes del gestor de datos
            misDatos.setImgDif((ImageIcon) null);
            misDatos.setImgConfa((ImageIcon) null);
            this.panelConfabulacion.resetDatos(misDatos);

            return;
        }
        // pide el nombre de la segunda imagen
String name2 = dialogo.abrirFichero();
        if(name2==null){ // si se cancela termina
            // resetea las imagenes del gestor de datos
            misDatos.setImgDif((ImageIcon) null);
            misDatos.setImgConfa((ImageIcon) null);

            this.panelConfabulacion.resetDatos(misDatos);

            return;
        }
        // invoca al Gestor de datos para que realice la
confabulacion
        misDatos.confabular(name, name2, dialogo);
        // resetea los datos del panel

```

```

        this.panelConfabulacion.resetDatos(misDatos);
    }

    /**
     * Method jMenuItemConfaGrabar actionPerformed. Implementa las
     acciones a realizar cuando se elige
     * la opción Grabar confabulada del menu Confabulaciones. Invoca
     el método del boton
     * correspondiente a esa accion
     * @param e Evento que se ha producido
     */
    private void jMenuItemConfaGrabar actionPerformed(ActionEvent e) {
        jMenuItemConfaGrabar_actionPerformed(e);
    }

    /**
     * Method jMenuItemConfaGenerar actionPerformed. Implementa las
     acciones a realizar cuando se elige
     * la opción Generar confabulada del menu Confabulaciones.
     Invoca el método del boton
     * correspondiente a esa accion
     * @param e Evento que se ha producido
     */
    private void jMenuItemConfaGenerar actionPerformed(ActionEvent e) {
        jMenuItemConfaGenerar_actionPerformed(e);
    }

    /**
     * Method jMenuItemHelpAbout actionPerformed. Implementa las
     acciones a realizar cuando se elige
     * la opción About del menu Help. Crea una ventana con la ayuda
     y la presenta en pantalla
     * @param e Evento que se ha producido
     */
    private void jMenuItemHelpAbout actionPerformed(ActionEvent e) {
        AboutBox about = new AboutBox(this);
        Dimension aboutSize = about.getPreferredSize();
        Dimension frmSize = getSize();
        Point loc = getLocation();
        about.setLocation((frmSize.width - aboutSize.width) / 2 + loc.x,
        (frmSize.height - aboutSize.height) / 2 + loc.y);
        about.setModal(true);
        about.show();
    }

    /**
     * Method processWindowEvent. Sobreescribe el método de JFrame
     para finalizar la aplicación cuando
     * se cierra la ventana de ésta.
     * @param e Evento que se ha producido
     */
    protected void processWindowEvent(WindowEvent e) {
        super.processWindowEvent(e);
        if (e.getID() == WindowEvent.WINDOW_CLOSING) {
            jMenuItemConfaSalir_actionPerformed(null);
        }
    }

    /**
     * Method jMenuItemConfaSalir actionPerformed. Implementa las
     acciones a realizar cuando se elige
     * la opción Salir del menu Confabulaciones. Termina la
     aplicación
     * @param e Evento que se ha producido
     */

```

```

private void jMenuItemConfaSalir_actionPerformed(ActionEvent e) {
    System.exit(0);
}

/**
 * Method main. Genera una instancia de la aplicación y la
 * presenta en pantalla
 */
public static void main(String[] args) {
    // Comprueba la version de Java
    String javaVersion = null;
    int major = 0;
    int minor = 0;

    try {
        javaVersion = System.getProperty("java.version");
        major = (new
Integer(javaVersion.substring(0,1)).intValue());
        minor = (new
Integer(javaVersion.substring(2,3)).intValue());
    }

    // si no detecta la version no pasa nada
    catch (Exception e) {
    }

    // si la version es 1.4 o superior acepta modo decorado
    if ((major > 1) || (major>0 && minor > 3)) {
        JFrame.setDefaultLookAndFeelDecorated(true);
        JDialog.setDefaultLookAndFeelDecorated(true);
    }

    ApplConfabula appli = new ApplConfabula();

    appli.pack();
    // centra la ventana y la presenta en pantalla
    Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = appli.getSize();
    if (frameSize.height > screenSize.height) {
        frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
        frameSize.width = screenSize.width;
    }
    appli.setLocation((screenSize.width - frameSize.width) / 2,
(screenSize.height - frameSize.height) / 2);
    appli.setVisible(true);
}
}

```

C.3 Clase BMPFile.java

Ver código fuente del apartado B.4

C.4 Clase Dialogos.java

Ver código fuente del apartado B.6

C.5 Clase GestorConfa.java


```

import java.util.*;
import javax.swing.*;

/**
 * @author Pedro Reverte
 *
 * Clase principal de control que implementa las acciones a realizar
 por la aplicación:
 * confabular imagenes y grabar la imagen confabulada
 *
 * 2004
 */
public class GestorConfa {

    // imagenes
    private ImageIcon imagen1;
    private ImageIcon imagen2;
    private ImageIcon imagenDif;
    private ImageIcon imagenConfa;

    private Dialogos dialogo;

    /**
     * Constructor GestorDatos.
     * @param dia - Instancia de Dialogos para presentar mensajes
     */
    GestorConfa(Dialogos dia) {

        this.dialogo = dia;
        this.imagenDif = null;
        this.imagenConfa = null;
    }

    /**
     * Method getImgDif. Accesor que devuelve la imagen de
diferencias de las confabuladas.
     * @return ImageIcon - imagen diferencial.
     */
    public ImageIcon getImgDif() {

        return imagenDif;
    }

    /**
     * Method getImgConfa. Accesor que devuelve la imagen creada por
confabulacion
     * @return ImageIcon - imagen confabulada.
     */
    public ImageIcon getImgConfa() {

        return imagenConfa;
    }

    /**
     * Method setImgDif. Accesor para establecer la imagen de
diferencias de las confabuladas
     * @param ImageIcon - imagen diferencial.
     */
    public void setImgDif(ImageIcon imageIcon) {
        this.imagenDif=imageIcon;
    }

    /**
     * Method setImgConfa. Accesor para establecer la imagen creada
por confabulacion
     * @param ImageIcon - imagen confabulada.
     */
    public void setImgConfa(ImageIcon imageIcon) {

```

```

        this.imagenConfa=imageIcon;
    }

    /**
     * Method confabular. Crea una imagen mostrando las diferencias
     existentes entre dos
     * versiones de la misma imagen marcadas con distinto código.
     Los pixeles diferentes se muestran
     * en color blanco y los que son iguales en negro. Crea una
     nueva imagen confabulada a
     * partir de las dos marcadas eligiendo de forma aleatoria los
     pixeles que difieren.
     * @param name - String Nombre de la primera imagen
     * @param name2 - String Nombre de la segunda imagen
     * @param dialogo - instancia de Dialogos para presentar mensajes
     * @return boolean - Cierto si se ha creado la imagen
     confabulada, falso en caso contrario
     */
    public boolean confabular(String name, String name2, Dialogos
    dialogo) {

        // lee el archivo de la primera imagen
        this.imagen1 = imageIO.loadImagen(name);
        if(this.imagen1 == null){
            dialogo.mostrarError("Error","No se ha podido leer el
            archivo de la primera imagen");
            this.imagenDif=null;
            this.imagenConfa=null;
            return false;
        }

        // lee el archivo de la segunda imagen
        this.imagen2 = imageIO.loadImagen(name2);
        if(this.imagen2 == null){
            dialogo.mostrarError("Error","No se ha podido leer el
            archivo de la segunda imagen");
            this.imagenDif=null;
            this.imagenConfa=null;
            return false;
        }

        // obtiene los arrays de pixels correspondientes
        int [][] [] pixelsImagen1 =
        imageIO.getImagePixels(this.imagen1);
        int [][] [] pixelsImagen2 =
        imageIO.getImagePixels(this.imagen2);
        // comprobamos que el tamaño de las imagenes coincida
        int height1 = pixelsImagen1.length;
        int width1 = pixelsImagen1[0].length;
        int height2 = pixelsImagen2.length;
        int width2 = pixelsImagen2[0].length;
        if((height1 != height2) || (width1 != width2)){
            dialogo.mostrarError("Error","Las imagenes tienen diferente
            tamaño");
            this.imagenDif=null;
            this.imagenConfa=null;
            return false;
        }
        // preparamos un generador de valores alaeatorios
        Random generador = new Random();
        // creamos los arrays para almacenar la diferencia y la
        imagen confabulada
        int [][] [] diferencias = new int[height1][width1][4];
        int [][] [] confabulada = new int[height1][width1][4];
        // los llenamos
        for(int i=0; i<height1; i++){
            for(int j=0; j<width1; j++){
                // si el pixel difiere en ambas imagenes

```

```

        if(pixelsImagen1[i][j][1] !=
pixelsImagen2[i][j][1]){
        // guardamos uno u otro de forma
        aleatoria
        boolean guardar=generador.nextBoolean();
        for(int h=0; h<4; h++){
        // pixels de diferencias a blanco
        diferencias[i][j][h]=255;
        // guardamos uno u otro de forma
        aleatoria en la confabulada
        if(guardar)
        confabulada[i][j][h] =
pixelsImagen1[i][j][h];
        else
        confabulada[i][j][h] =
pixelsImagen2[i][j][h];
        }
        }
        // si ambos pixeles son iguales
        else{
        for(int h=0; h<4; h++){
        // establecemos el pixel de
        diferencias[i][j][h]=0;
        // en la confabulada guardamos uno
        cualquiera
        confabulada[i][j][h] =
pixelsImagen1[i][j][h];
        }
        }
        }
        // crea una imagen a partir del array de diferencias
        this.imagenDif = imageIO.crearImagen(diferencias);
        // crea una imagen a partir del array de confabulada
        this.imagenConfa = imageIO.crearImagen(confabulada);

        return true;
    }

    /**
     * Method confabular2. Crea una imagen mostrando las diferencias
     existentes entre dos
     * versiones de la misma imagen marcadas con distinto código.
     Los pixeles diferentes se muestran
     * en color blanco y los que son iguales en negro. Crea una
     nueva imagen confabulada a
     * partir de las dos marcadas eligiendo de forma aleatoria 5
     lineas consecutivas de cada imagen
     * @param name - String Nombre de la primera imagen
     * @param name2 - String Nombre de la segunda imagen
     * @param dialogo - instancia de Dialogos para presentar mensajes
     * @return boolean - Cierto si se ha creado la imagen
     confabulada, falso en caso contrario
     */
    public boolean confabular2(String name, String name2, Dialogos
    dialogo) {
        // lee el archivo de la primera imagen
        this.imagen1 = imageIO.loadImagen(name);
        if(this.imagen1 == null){
        dialogo.mostrarError("Error", "No se ha podido leer el
        archivo de la primera imagen");
        this.imagenDif=null;
        this.imagenConfa=null;
        return false;
        }
    }

```

```

        // lee el archivo de la segunda imagen
        this.imagen2 = imageIO.loadImagen(name2);
        if(this.imagen2 == null){
            dialogo.mostrarError("Error", "No se ha podido leer el
archivo de la segunda imagen");
            this.imagenDif=null;
            this.imagenConfa=null;
            return false;
        }
        // obtiene los arrays de pixels correspondientes
        int[][][] pixelsImagen1 =
imageIO.getImagePixels(this.imagen1);
        int[][][] pixelsImagen2 =
imageIO.getImagePixels(this.imagen2);
        // comprobamos que el tamaño de las imagenes coincida
        int height1 = pixelsImagen1.length;
        int width1 = pixelsImagen1[0].length;
        int height2 = pixelsImagen2.length;
        int width2 = pixelsImagen2[0].length;
        if((height1 != height2) || (width1 != width2)){
            dialogo.mostrarError("Error", "Las imagenes tienen diferente
tamaño");
            this.imagenDif=null;
            this.imagenConfa=null;
            return false;
        }
        // preparamos un generador de valores alaeatorios
        Random generador = new Random();
        // creamos los arrays para almacenar la diferencia y la
imagen confabulada
        int[][][] diferencias = new int[height1][width1][4];
        int[][][] confabulada = new int[height1][width1][4];
        // los llenamos
        // contador de lineas
        int contador=0;
        // generador aleatorio
        boolean guardar=generador.nextBoolean();
        for(int i=0; i<height1; i++){
            for(int j=0; j<width1; j++){
                // si el contador es mayor 4 lo reseteamos y
preparamos un nuevo valor aleatorio
                if(contador>4){
                    guardar=generador.nextBoolean();
                    contador=0;
                }
                // si el pixel difiere en ambas imagenes
                if(pixelsImagen1[i][j][1] !=
pixelsImagen2[i][j][1]){
                    for(int h=0; h<4; h++)
                        // pixels de diferencias a blanco
                        diferencias[i][j][h]=255;
                }
                else{
                    // si los pixeles son iguales
                    for(int h=0; h<4; h++)
                        // establecemos el pixel de
diferencias a negro
                        diferencias[i][j][h]=0;
                }
                // guardamos una linea u otra de forma
aleatoria en la confabulada
                for(int h=0; h<4; h++){
                    if(guardar)
                        confabulada[i][j][h] =
pixelsImagen1[i][j][h];
                    else

```

```

                                confabulada[i][j][h] =
pixelsImagen2[i][j][h];
                                }
                                }
                                contador++;
                                }
                                // crea una imagen a partir del array de diferencias
                                this.imagenDif = imageIO.crearImagen(diferencias);
                                // crea una imagen a partir del array de confabulada
                                this.imagenConfa = imageIO.crearImagen(confabulada);

                                return true;
                                }
                                /**
                                * Method grabarConfabulada. Graba si existe, la ultima imagen
                                confabulada a un archivo con el
                                * nombre recibido como parámetro
                                * @param name - String Nombre del archivo a guardar
                                * @param dialogo - Instancia de Dialogos para mostrar mensajes
                                * @return boolean - Cierto si se ha grabado el archivo, falso
                                en caso contrario
                                */
                                public boolean grabarConfabulada(String name, Dialogos dialogo)
                                {
                                // crea el archivo con la imagen marcada
                                imageIO.saveImagen(this.imagenConfa, name);
                                // reseteamos las imagenes
                                this.imagenDif=null;
                                this.imagenConfa=null;

                                return true;
                                }
                                }

```

C.6 Clase ImageFilter.java

Ver código fuente del apartado B.10

C.7 Clase imageIO.java

Ver código fuente del apartado B.11

C.8 Clase iObserver.java

Ver código fuente del apartado B.13

C.9 Clase PanelConfabulacion.java

```

import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;

/**
 * @author Pedro Reverte
 * Implementa el panel utilizado para la confabulacion de imagenes. El
 * panel esta formado por la imagen
 * que representa las diferencias entre las dos confabuladas y la
 * imagen resultado de la confabulacion
 *
 * 2004
 */
public class PanelConfabulacion extends JPanel{

```

```

GridBagConstraints c = new GridBagConstraints();

JLabel diferencias = new JLabel();
JLabel confabulada = new JLabel();
JScrollPane scrollPaneDif = new JScrollPane(diferencias);
JScrollPane scrollPaneConfa = new JScrollPane(confabulada);

JLabel label1 = new JLabel("Diferencias");
JLabel label2 = new JLabel("Imagen Confabulada");

/**
 * Constructor PanelConfabulacion.
 * @param misDatos - Datos necesarios para generar el panel
 */
PanelConfabulacion(GestorConfa misDatos) {

    this.setLayout(new GridBagLayout());
    // los componentes crecen en las dos dimensiones
    c.fill = GridBagConstraints.BOTH;
    c.insets = new Insets(5,5,5,5);

    diferencias.setHorizontalAlignment(JLabel.CENTER);
    confabulada.setHorizontalAlignment(JLabel.CENTER);

    // inserta los datos en el panel
    resetDatos(misDatos);

    c.gridx = 0; c.gridy = 0; c.gridwidth = 3; c.gridheight=1;
    c.weightx = c.weighty = 0.0;
    this.add(label1, c);

    c.gridx = 5; c.gridy = 0; c.gridwidth = 3; c.gridheight=1;
    c.weightx = c.weighty = 0.0;
    this.add(label2, c);

    c.gridx = 0; c.gridy = 1; c.gridwidth = 3; c.gridheight=7;
    c.weightx = c.weighty = 1.0;
    this.add(scrollPaneDif, c);

    c.gridx = 5; c.gridy = 1; c.gridwidth = 3; c.gridheight=7;
    c.weightx = c.weighty = 1.0;
    this.add(scrollPaneConfa, c);

}

/**
 * Method resetDatos. Establece los datos presentados en el
panel de acuerdo con el
 * contenido del gestor de datos
 * @param misDatos - Gestor de datos
 */
public void resetDatos(GestorConfa misDatos) {

    // si no existe imagen de diferencias, datos nulos
    if(misDatos.getImgDif() == null) {
        diferencias.setText("Imágenes no seleccionadas");
        diferencias.setIcon(null);
        diferencias.setPreferredSize(new Dimension(350,300));
    }
    // si existe se establecen los datos del GestorDatos
    else {
        diferencias.setIcon(misDatos.getImgDif());
        diferencias.setPreferredSize(new
Dimension(misDatos.getImgDif().getIconWidth(), misDatos.getImgDif().get
IconHeight()));
        diferencias.setText("");
    }
}

```

```
// lo mismo para la imagen confabulada
if (misDatos.getImgConfa() == null) {
    confabulada.setText("No creada");
    confabulada.setIcon(null);
    confabulada.setPreferredSize(new Dimension(350, 300));
}
else {
    confabulada.setIcon(misDatos.getImgConfa());
    confabulada.setPreferredSize(new
Dimension(misDatos.getImgConfa().getIconWidth(), misDatos.getImgConfa()
.getIconHeight()));
    confabulada.setText("");
}
}
}
```

C.10 Clase utils.java

Ver código fuente del apartado B.17