Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
(Mención en Tecnologías de la Información)

Minería de Datos en Imágenes

Autor:
D. Diego Vázquez Blanco

Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
(Mención en Tecnologías de la Información)

Minería de Datos en Imágenes

Autor:
D. Diego Vázquez Blanco

Tutor:
D. Quiliano Isaac Moro Sancho

Glosario

- **ComboBox**: Elemento GUI que permite al usuario seleccionar una opción de una lista existente de opciones.
- **MessageBox**: Mensaje en un cuadro de diálogo emergente, espera a que el usuario haga clic en un botón y devuelve un entero que indica el botón utilizado.
- **Label**: Etiqueta de texto que suele usarse como título activo delante de un cuadro de texto u otro elemento de control. Su texto puede cambiarse en tiempo de ejecución. También pueden servir como contenedores de una imagen.
- **Código Fuente**: Conjunto de líneas de texto escrito en un lenguaje de programación, que son las directrices que debe seguir la computadora para realizar dicho programa. Debe ser traducido a otro lenguaje máquina que el ordenador pueda ejecutar. Para esta traducción se emplean los llamados compiladores, ensambladores o intérpretes.
- **Compilador**: Programa informático que traduce un programa que ha sido escrito en un lenguaje de programación a un lenguaje diferente para que pueda ser ejecutado, usualmente lenguaje de máquina, aunque también puede ser traducido a un código intermedio (bytecode).
- **Máquina Virtual Java**: Es una máquina virtual de proceso nativo, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (el bytecode Java), el cual es generado por el compilador del lenguaje Java.
- **IDE**: Un entorno de desarrollo integrado, en inglés Integrated Development Environment, es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.
- **InputBox**: Mensaje en un cuadro de diálogo emergente, espera a que el usuario escriba un texto o haga clic en un botón y devuelve una cadena con el contenido del cuadro de texto.
- **ScrollBar**: Objeto de la interfaz gráfica de usuario mediante el cual una página de internet, una imagen, un texto, etc, pueden ser deslizados hacia abajo o arriba. También hay barras deslizantes horizontales, cuando el contenido es demasiado ancho para la pantalla.
- **ToolTip**: Herramienta de ayuda visual, que funciona al situar el cursor sobre algún elemento gráfico, mostrando una ayuda adicional para informar al usuario de la finalidad del elemento sobre el que se encuentra.
- **GUI**: Interfaz gráfica de usuario, del inglés Graphical User Interface, es un elemento informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz.

Índice

1. Introducción

A lo largo de los últimos años, el avance de la tecnología para dispositivos móviles ha sido tal, que hoy en día prácticamente todos los *smartphones* disponen de cámaras de fotos con una calidad de imagen más que suficiente para satisfacer al usuario medio. Esto, incentiva a que el usuario se anime a hacer uso de la cámara de forma cotidiana capturando un gran número de fotografías.

Por otra parte, las mejores características de conexión a Internet y la popularidad de las aplicaciones de mensajería instantánea, han elevado el tráfico de imágenes por la red incrementándose el número de imágenes descargadas de Internet y recibidas por la *app* de mensajería.

Es indudable que se han producido grandes progresos en tecnología, incluyendo los concernientes a imágenes y fotografías, pero detrás de cada avance hay la posibilidad de que surja otra necesidad.

1.1. Motivación Original

El desarrollo de una aplicación de escritorio para establecer una comparativa de imágenes viene motivado principalmente por la necesidad de los usuarios de cámaras fotográficas de poder comparar de manera práctica las diferentes fotos tomadas. Así mismo, también puede cubrir la necesidad de limpieza de fotos descargadas de internet o que han sido recibidas mediante mensajería instantánea y que en ocasiones pueden dar lugar a la existencia duplicados mermando innecesariamente la capacidad de almacenamiento del dispositivo que las contiene.

1.2. Objetivos del proyecto

Como principal objetivo de este proyecto se ha fijado desarrollar una herramienta que, dado un conjunto de imágenes digitalizadas, determine cuáles de ellas son similares o iguales; para ello debiendo de extraer de forma automática características de cada una de las fotografías y mediante algún algoritmo permitir establecer un ranking con las más similares.

Otro de los principales objetivos de un producto de estas características es la forma de llegar al usuario, por lo que una interfaz amigable y la facilidad de uso se hacen indispensables en este proyecto. La robustez es otro de los objetivos de cualquier sistema software, a través de la cual se mide, en su mayoría, la calidad del producto.

El último de los objetivos a destacar es el reto que supone llevar a cabo un proyecto software desde cero, planificando todas sus etapas, desde el estudio y análisis, hasta la codificación y las pruebas necesarias, así como la puesta en práctica de las capacidades y conocimientos adquiridos a lo largo de toda la carrera.

1.3. Planificación y Costes

Para cumplir con el objetivo final, se deben tener en cuenta una serie de dependencias y restricciones que será necesario cumplir.

Como restricción fundamental se tiene el tiempo del que se dispone que es limitado, se trata de una asignatura de 12 ECTS del Grado en Ingeniería Informática que en teoría se debería realizar en unas 300 horas a lo largo de un cuatrimestre, por lo que se tendrá que realizar el proyecto teniendo en cuenta la asignación del tiempo a las fases de análisis y desarrollo de la aplicación, además de tener en cuenta el tiempo a invertir en el documento de la memoria y los manuales de usuario e instalación.

También se tiene la restricción de no poder disponer de más personal y, por tanto, la única persona que va a poder realizar las diferentes tareas para completar el proyecto, es el autor de éste.

Las dependencias y requisitos estrictamente referentes al software se verán en las secciones de análisis y diseño.

En cuanto a las herramientas a emplear, tenemos de dos tipos:

1. Desarrollo: Las herramientas desarrollo que se van a usar son NetBeans como plataforma, JVM como máquina virtual para la ejecución del programa en java y GIMP para la obtención visual de histogramas de imágenes con el fin de estudiar su naturaleza.
2. Documentación: A fin de recopilar la información sobre el proyecto se empleará la herramienta online *Sharelatex*, la cual permite realizar documentos en formato PDF de forma personalizada, dispone de autoguardado, mantiene un historial de cambios y permanece alojado en la nube.

Dado que emplearemos un patrón MVC para desarrollar el código de la aplicación, usaremos la tecnología con la que se ha aplicado dicho patrón en las clases del grado al que pertenece este trabajo. En cuanto a la persistencia de la información, no se requerirá ninguna tecnología porque no se guarda información alguna.

Los lenguajes de programación, tecnologías y elementos a utilizar serán:

- Java *Swing* para las interfaces de usuario.
- *Bash/Shell Scripts* para ejecutar la aplicación de forma automática por el sistema cuando se requiera de un reinicio de la aplicación y de esta forma se faciliten al usuario las tareas posibles.

En este apartado se da una visión general de las fases seguidas a lo largo de la realización del proyecto, que son las correspondientes a la mayoría de un desarrollo software.

- **Estudio**: Investigación sobre las posibles tecnologías a utilizar, posible estudio de mercado y de viabilidad. Indagación y pruebas sobre aplicaciones relacionadas, con el objetivo de extraer lo que se suele utilizar en proyectos similares. Investigación sobre la naturaleza de los datos que se van a tratar. [3] [1]
- **Análisis**: Definición de requisitos con el objetivo de dilucidar exactamente qué es lo que se desea construir.
- **Diseño**: Tras finalizar las fases de estudio y análisis, y con el listado de requisitos bien definido, se procede a diseñar la arquitectura de la aplicación. Esto incluye diversos diagramas que definirán el diseño de la interfaz, estructuración del código y robustez.
- **Implementación**: Programación de la aplicación de escritorio en Java.
- **Pruebas**: Una vez finalizadas todas las fases anteriores, se hace necesario la realización de una fase de pruebas generales para comprobar la calidad y efectividad del software desarrollado.
- **Documentación**: Según una guía sobre desarrollo software [4], la documentación es algo indispensable para el correcto mantenimiento de cualquier producto software. Tanto a nivel de código como de documento, con el fin de dejar constancia por escrito de las fases llevadas a cabo en el ciclo de vida del producto software.

1.3.1. Costes

Se trata de un trabajo de fin de grado sin financiación por ninguna persona, entidad u organización; por ello los costes están limitados a los recursos mínimos de los que puede disponer el autor para la exitosa realización tanto del producto software como de la documentación.

Los recursos mínimos mencionados son los siguientes:

1. Ordenador de sobremesa o portátil de potencia media (500-800 €)

2. Licencia del sistema operativo si no lo incluye el ordenador (100 €)

3. Pendrive para copias de seguridad (10 €)

4. Dos CDs (0,40 €)

Los demás recursos utilizados son de uso gratuito o de software libre por lo que no influyen en los costes. Los gastos de electricidad e Internet se han obviado dado que son gastos generales de vivienda y el cálculo de su utilización para el desarrollo del proyecto es muy impreciso.

1.4. A quién va dirigida

Se pueden decir dos grupos de personas a los que principalmente va dirigida esta aplicación. Uno de los grupos es bastante específico destinada a personas bastante concretas. El otro es más genérico pero también especifica qué usuarios se incluyen. Estos grupos son:

1. Usuarios generales: Este es el grupo que designa a los usuarios que harían un uso menos frecuente pero que ocasionalmente les puede ayudar a ordenar fotografías y ahorrar espacio quitando las imágenes repetidas. A estas personas les resulta útil la aplicación porque normalmente no es de su preocupación todas las imágenes que se guardan en sus *smartphones* debido a la mensajería instantánea y tampoco revisar las fotografías de la cámara por si hay fotos muy parecidas a otras que resulten innecesarias. Por lo tanto no necesitan añadir nada al código o modificar la aplicación.

2. Usuarios para orientación de Minería de Datos: Son los que usarían el programa para establecer otros criterios de ordenación, como una búsqueda de patrones, o reconocimiento de objetos. En todo caso, son usuarios que realizarían modificaciones en el programa, por ejemplo cambiando bibliotecas.

1.5. Limitaciones

En el apartado de planificación se han tratado limitaciones referentes a la realización general del TFG. Pero hay otras limitaciones más concretas en las que se puede profundizar, y éstas son las referentes a las capacidades de la aplicación para realizar acciones, ya que las funcionalidades que ofrece no son infinitas.

Las limitaciones de funcionamiento de la aplicación que más se pueden destacar para su ámbito de uso son las siguientes:

- No está diseñado para funcionar en plataformas móviles como Android o iOS.
- Está limitado a funcionar con determinados tipos de imágenes. En general admite JPEG, JPG, PNG, BMP y GIF; pero al aplicar el filtro de escala de grises sólo admite JPEG, JPG y BMP.
- El máximo número de imágenes que puede cargar en memoria depende de la cantidad de memoria libre durante la ejecución.

1.6. Requisitos Hardware y Software

Existen una serie de requisitos concretos que son necesarios como base para llevar a cabo el desarrollo de la aplicación y la memoria, y también la ejecución del programa. Éstos están divididos en dos tipos, hardware y software.

1.6.1. Creación del TFG

Los requisitos hardware para hacer posible el desarrollo del proyecto, tanto el código como la documentación, serían:

1. Ordenador de potencia media.
2. Conexión a Internet.
3. Conexión a la red eléctrica.
4. Un pendrive o disco duro para guardar copias de seguridad.
5. Dos CDs.

Por otra parte, los requisitos software para el mismo objetivo serían:

1. Licencia del sistema operativo.
2. Versión más actualizada del sistema que permita ejecutar las herramientas de desarrollo y creación de documentos.
3. Entorno de Desarrollo Integrado (IDE) que permita un buen desarrollo del código fuente.

4. Compilador apropiado para construir el ejecutable de la aplicación.
5. Herramienta de creación de documentos PDF con múltiples servicios y compatible con Latex.
6. Navegador web que permita ejecutar las herramientas online de creación de documentos.

1.6.2. Ejecución de la Aplicación

Los requisitos hardware en este caso estarían compuestos por un subconjuntos de los nombrados en la creación del TFG:

1. Ordenador de potencia media.
2. Conexión a la red eléctrica.

Entre los requisitos software de ejecución se encontrarían otros diferentes al caso de la anterior subsección además de alguno igual:

1. Licencia del sistema operativo.
2. Versión más actualizada del sistema que permita ejecutar la máquina virtual necesaria y/o la aplicación.
3. Máquina virtual actualizada a la última versión que permita ejecutar de manera correcta la aplicación.

1.7. Estudio de Histogramas de una imagen

Lo siguiente, es una investigación realizada para conocer la naturaleza de un posible recurso clave a utilizar tanto en la comparación de imágenes como en el reconocimiento de formas y cosas similares.

1.7.1. Qué es un histograma de una imagen

El histograma de una imagen nos muestra el número de píxeles que tiene un mismo nivel de un color. En una gráfica de barras, en las abscisas se representan los distintos colores de la imagen (niveles del color, desde el más oscuro al más claro) y en las ordenadas la frecuencia relativa de aparición de cada nivel del color.

Hay un histograma independiente por cada color RGB (Rojo, Verde, Azul) y un histograma general e escala de grises.

Cada **canal** del histograma representa el conjunto de **niveles** de un color y sus frecuencias relativas.

Los **niveles** son los valores de luminosidad, los diferentes matices de brillo de un color luz que aparecen en una imagen.

1.7.2. Ejemplo de imagen y su histograma

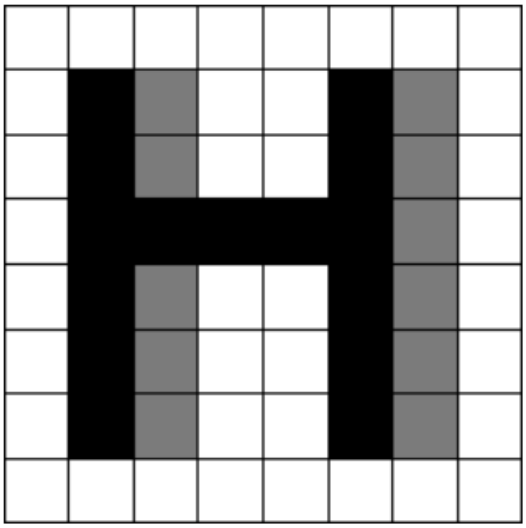

Figura 1: Imagen de 8 x 8 píxeles.

Nivel de gris	Brillo
0	Negro
1	Gris oscuro
2	Gris claro
3	Blanco

Tabla 1: 2 bits para codificar el brillo.

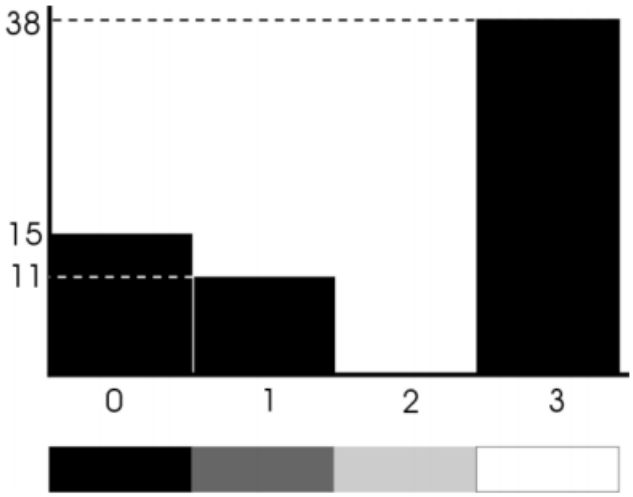

Figura 2: Imagen de 8 x 8 píxeles.

1.7.3. Reconocimiento de objetos

El histograma es útil en tareas de análisis automático de imágenes, por ejemplo una aplicación de inspección automática que trata de determinar los píxeles que corresponden a un objeto que se sabe que se encuentra sobre un **fondo uniforme**.

El método es encontrar un umbral en el histograma tal que quede como se muestra en la siguiente imagen:

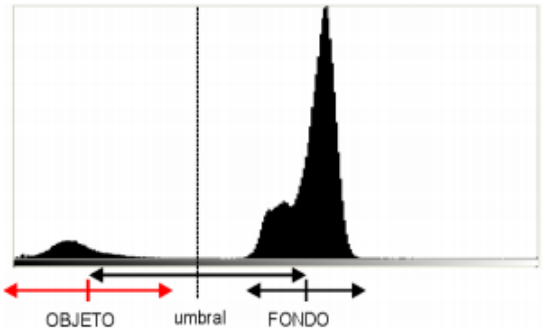

Figura 3: Histograma Bimodal de una imagen de un objeto sobre fondo claro uniforme.

2. Análisis

En esta parte se presenta la fase de análisis, la inicial de todo proyecto software, donde se definen los requisitos y se muestra una visión global de la arquitectura pensada para el sistema. En el siguiente capítulo se tendrá en cuenta este análisis de requisitos como base para el diseño de todos los aspectos de la aplicación. Es por ello por lo que la fase de análisis es de suma importancia para el devenir de todo producto software. Aquí es donde se deben asentar las bases, a modo de cimientos, del proyecto, y, a partir de las cuales se construirá todo lo demás.

El proceso a seguir se basa en primer lugar en una buena definición de requisitos y en elegir una metodología de desarrollo acorde con el proyecto.

Éste capítulo es fundamental para entender el proyecto. A continuación se desglosan la funcionalidad y las características a modo de análisis de requisitos, teniendo en cuenta tanto requisitos funcionales como no funcionales.

2.1. Análisis de Requisitos

Este apartado define la especificación del comportamiento que se espera de cualquier proyecto software. Estudiando los requisitos de otras aplicaciones del mismo tipo, se ha predefinido una serie de requisitos que se consideran indispensables para el proyecto. A continuación, se muestra una enumeración y breve descripción de los requisitos establecidos para el diseño y desarrollo de la aplicación.

2.1.1. Requisitos Funcionales

Los requisitos funcionales describen todas las interacciones que tendrán los usuarios con el software.

RF1: Selección de carpeta de iImágenes

1. Se le abrirá al usuario un cuadro de diálogo en el que podrá navegar por los directorios del dispositivo y seleccionar la carpeta de imágenes deseada.

2. El programa validará la selección del usuario.

3. En caso de validación positiva, se cargarán las imágenes y serán mostradas en el panel principal del mismo modo en el que lo hacen las carpetas.

RF2: Selección de imágenes

1. Se podrán seleccionar las imágenes del panel principal y del ranking haciendo clic en ellas y deseleccionar haciendo clic otra vez en las que están seleccionadas.

2. Para seleccionar o deseleccionar todas las imágenes (sólo del panel principal), se pulsarán los botones correspondientes en la vista principal.

RF3: Cambio de algoritmos

1. Se seleccionará el algoritmo deseado del *combobox* correspondiente.

2. En el caso del algoritmo de cálculo del ranking funcionará tras pulsar el botón Comparar.

3. En el caso del algoritmo del filtro a aplicar se pulsará el botón de cambiar filtro para que se produzca el cambio de biblioteca.

RF4: Aplicar un filtro a una o varias imágenes

1. Se seleccionarán las imágenes deseadas en la vista principal.

2. Se pulsará el botón de aplicar filtro.

RF5: Cálculo del ranking

1. Se seleccionarán las imágenes a ser comparadas entre sí en la vista principal.

2. Se pulsará el botón de comparar.

RF6: Eliminación de imágenes

1. Se seleccionarán las imágenes a ser eliminadas en la vista del ranking.

2. Se pulsará el botón de eliminar.

RF7: Regresar a la vista principal

1. Se debe encontrar en la vista del ranking.

2. En la vista del ranking se pulsará el botón de volver.

RF8: Vista principal

1. La vista principal constará de 6 botones:

   a) **Seleccionar Carpeta**: Se mostrará un cuadro de diálogo para elegir carpeta.

   b) **Información**: Se mostrará una ventana de mensaje con información de uso.

   c) **Seleccionar Todas**: Se seleccionarán todas las imágenes de la carpeta seleccionada.

   d) **Deseleccionar Todas**: Se deseleccionarán todas las imágenes de la carpeta seleccionada.

   e) **Aplicar Filtro**: Se aplicará un filtro que cambiará el aspecto de las imágenes seleccionadas.

   f) **Comparar**: Se procederá al cálculo de las imágenes más parecidas entre las seleccionadas.

2. La vista principal constará de 1 *ComboBox*:

   a) **Algoritmo de Comparación**: Contendrá la lista de algoritmos a elegir para realizar la comparación entre imágenes.

3. La vista principal constará de 1 ScrollPanel:

   a) **ScrollPanel Principal**: Mostrará las imágenes de la carpeta seleccionada y permitirá hacer *scroll*. Las imágenes serán seleccionables.

RF9: Vista del ranking

1. La vista del ranking constará de 2 botones:

   a) **Volver**: Cerrará la ventana del ranking y volverá a la ventana principal sin imágenes cargadas.

   b) **Eliminar**: Eliminará del dispositivo las imágenes seleccionadas del ranking.

2. La vista del ranking constará de 2 ScrollPanel:

   a) **ScrollPanel Seleccionadas**: Mostrará las imágenes seleccionadas de la vista principal.

   b) **ScrollPanel ranking**: Mostrará las imágenes más parecidas a cada seleccionada. Se mostrarán de más parecida a menos parecida en orden de izquierda a derecha respectivamente. Las imágenes serán seleccionables.

RF10: Notificaciones

1. El sistema debe lanzar notificaciones en los siguientes casos:

   a) Tras cargar las imágenes de una carpeta, informará de cuántas se han cargado.

   b) Si no hay imágenes admitidas en la carpeta seleccionada.

   c) Si se selecciona un fichero o archivo durante el proceso de selección de directorio.

   d) Al presionar un botón de información.

   e) Al cambiar el algoritmo del filtro a aplicar, se informará que se reiniciará la aplicación antes de producirse.

   f) Si se selecciona para aplicar un filtro a un tipo de imagen al que no es posible aplicarlo.

   g) Si se hace clic en el botón de comparar sin haber seleccionado ninguna imagen.

   h) Si no se introduce correctamente el número máximo de imágenes que tendrá el ranking cuando lo pida el programa después de pulsar al botón de comparar.

   i) Al comenzar a procesar algo que pueden llevar algo de tiempo.

   j) Al cancelar la operación de comparación de las imágenes seleccionadas.

   k) Si se pulsa el botón eliminar sin haber seleccionado ninguna imagen.

2.1.2. Requisitos No Funcionales

Requisitos complementarios o atributos de calidad. Especifican criterios que juzgan operaciones del sistema en lugar de su comportamiento.

RNF1: Documentación

1. Manual de instalación de la aplicación.

2. Manual de usuario de la aplicación.

3. La codificación del sistema deberá ser clara y estar documentada de manera que algún programador pueda agregar funcionalidad posteriormente, procurando seguir los estándares de programación Java.

RNF2: Mantenibilidad y portabilidad

1. Disponibilidad para los dispositivos de escritorio Windows o Linux.

2. La aplicación estará desarrollada en Java siguiendo un patrón MVC.

3. Será necesario disponer de una máquina virtual de java instalada en el equipo.

RNF3: Interfaz y usabilidad

1. La aplicación debe constar de una interfaz sencilla, atractiva e intuitiva. De tal forma que su uso no suponga un impedimento o esfuerzo al usuario a la hora hacer uso de la aplicación. Para ello se utilizará el framework *Swing* de Java.

2. La introducción de datos debe estar estructurada procurando evitar errores.

RNF4: Rendimiento

1. Se esperan tiempos de respuesta asumibles por el usuario en relación a la cantidad de datos que se procesen.

2. Se debe informar al usuario de los procesos que se llevan a cabo si se estima que pueden tardar demasiado.

3. Diseño

En la presente sección se describe todo el proceso de diseño de la aplicación. Se ha realizado un diseño que abarca todos los requisitos descritos en el apartado 2.1 Análisis de requisitos. El diseño proporciona una idea completa del software desarrollado en el proyecto. Además, se justifican las decisiones tomadas para el posterior desarrollo.

3.1. Arquitectura

El proyecto accedería de forma local al directorio especificado, por lo que la separación tan popular entre cliente y servidor no tendría sentido en este caso. En lugar de ello, se optaría por una separación interna que permitiese un posterior mantenimiento más eficiente y una mayor capacidad de escalado para el trabajo futuro.

Las ventajas que supone dicha separación son notables, ventajas de tipo organizativo debidas a la eficiencia de la gestión de la información y separación de responsabilidades, lo que clarifica y facilita el sistema. La escalabilidad es otra de las ventajas más destacables de esa separación local, permitiendo aumentar funcionalidades o recursos fácilmente. El patrón que encaja con dicha filosofía y que se puede seguir para desarrollar el código de la aplicación es el basado en capas.

3.1.1. Patrón de capas

Cuando planteamos el diseño de una aplicación, lo primero es conseguir separar las tareas que el sistema debe desempeñar entre distintas capas lógicas y en base a la naturaleza de dichas tareas. La tendencia más aceptada es la aplicación de algún patrón de diseño que divida la responsabilidad en diferentes capas que interaccionen unas con otras a través de sus interfaces. Uno de los patrones comunes, y el que vamos a utilizar, es el patrón de las tres capas:

- **Presentación:** Se limita a la navegabilidad y a gestionar todos aquellos aspectos relacionados con la lógica de presentación de la aplicación.
- **Negocio:** El resultado del análisis funcional de la aplicación viene a ser la identificación del conjunto de reglas de negocio que abstraen el problema real a tratar. Son las que realmente suponen el motor del sistema.
- **Persistencia/Acceso a Datos:** Es la encargada de persistir las entidades que se manejan en la capa negocio, al acceso a datos, etc. Aunque puede ofrecer servicios más complejos.

3.2. Tecnologías utilizadas

En este pequeño apartado se enumeran las diferentes herramientas y tecnologías utilizadas durante el desarrollo del proyecto.

3.2.1. MockFlow

Es un constructor gráfico de maquetas. Permite al diseñador organizar y diseñar las maquetas utilizando su potente herramienta *drag & drop*, con la que puede arrastrar elementos y moverlos a su antojo dentro de la maqueta. Dentro de los numerosos editores de maquetas existentes, MockFlow permitía la creación de un proyecto para un ordenador de Escritorio, pudiendo hacer uso de la mayoría de elementos característicos de este tipo de sistemas operativos (por ejemplo Windows, el cual es un SO destinado principalmente a ordenadores de sobremesa y portátiles). [10]

3.2.2. astah professional

Es una herramienta de diseño de sistemas que soporta UML, Diagrama de Relación de Entidades, diagramas de flujo, CRUD, Diagrama de flujo de datos, Tabla de Requisiciones y Mapas Mentales. En esta ocasión lo usaremos por su utilidad para la creación de los diagramas UML básicos para la fase de diseño. Es un software que se suele ser la opción a utilizar por los centros universitarios de informática por la sencillez de su manejo. [8]

3.2.3. NetBeans

Es un entorno de desarrollo integrado libre y gratuito sin restricciones de uso, hecho principalmente para el lenguaje de programación Java, y además existe un número importante de plugins para extenderlo.

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándo nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

3.2.4. Java

Es un lenguaje de programación, con una estructura y una sintaxis similar a C, aunque con un concepto más simple y eliminando las herramientas de bajo nivel. Se encuentra ampliamente extendido tanto en el mundo educativo como en el profesional por varias razones que se explican a continuación:

- **Es orientado a objetos**: Si bien existen detractores de esta modalidad, la programación orientada a objetos resulta muy conveniente para la mayoría de las aplicaciones, y es esencial para los videojuegos. Entre las ventajas más evidentes que ofrece se encuentra un gran control sobre el código y una mejor organización, dado que basta con escribir una vez los métodos y las propiedades de un objeto, independientemente de la cantidad de veces que se utilicen.

- **Es muy flexible**: Java es un lenguaje especialmente preparado para la reutilización del código; permite a sus usuarios tomar un programa que hayan desarrollado tiempo atrás y actualizarlo con mucha facilidad, sea que necesiten agregar funciones o adaptarlo a un nuevo entorno.

- **Funciona en cualquier plataforma**: A diferencia de los programas que requieren de versiones específicas para cada sistema operativo (tales como Windows o Mac), las aplicaciones desarrolladas en Java funcionan en cualquier entorno, dado que no es el sistema quien las ejecuta, sino la máquina virtual (conocida como Java Virtual Machine o JVM).

- **Su uso no acarrea inversiones económicas**: Programar en Java es absolutamente gratis; no es necesario adquirir ninguna licencia, sino simplemente descargar el kit de desarrollo (Java Development Kit o JDK) y dar riendas sueltas a la imaginación.

- **Es de fuente abierta**: Java ofrece el código de casi todas sus librerías nativas para que los desarrolladores puedan conocerlas y estudiarlas en profundidad, o bien ampliar su funcionalidad, beneficiándose a ellos mismos y a los demás.

- **Es un lenguaje expandible**: Continuando con el punto anterior, cada programador tiene la libertad de revisar y mejorar el código nativo de Java, y su trabajo puede convertirse en la solución a los problemas de muchas personas en todo el mundo. Infinidad de desarrolladores han aprovechado esta virtud del lenguaje y continúan haciéndolo.

3.2.5. Swing

Con Swing le daremos vida a nuestro sistema, ya que se crearán las vistas de la aplicación, por medio de las cuales el usuario interactuará con el sistema. Además de que se tiene una gran cantidad de posibilidades para estructurar nuestros desarrollos, se pueden manejar los eventos de cada componente dependiendo de nuestras necesidades, así como utilizar *look & feel* para modificar el aspecto visual de nuestras interfaces. [13]

3.2.6. Beans Binding

En muchas ocasiones, necesitamos sincronizar datos en varios componentes, de modo que, al cambiar los datos en un componente, se esto se refleje en los demás componentes. Para esto, normalmente se escriben oyentes de eventos, los cuales se agregan a los componentes para que reciban una notificación al realizarse un cambio. La cuestión es que a veces es mejor olvidarse de estos detalles para centrarse en la funcionalidad propia de la aplicación y permitir que se automatice la creación y adaptación de ese código.

Afortunadamente, gracias a la biblioteca Beans Binding y a que se integra bien con NetBeans, esto es posible. Así, nos liberamos de crear oyentes e implementar sus métodos para enlazar componentes, y el soporte de NetBeans para esta biblioteca hace esto por nosotros sin escribir código. [11]

3.2.7. Metadata Xtractor

Es una biblioteca sencilla de Java para leer metadatos de archivos de imagen. [9]

3.2.8. XMP

Extensible Metadata Platform (XMP) es un tipo de lenguaje especificado extensible de marcado usado en los archivos PDF y fotografía. XMP define un esquema particular de propiedades básicas útiles para registrar la historia de un recurso a través de múltiples pasos en su procesamiento, como tomar una fotografía o editar una imagen. XMP permite a programas o dispositivos agregar su propia información al recurso digital, con lo que puede quedar incorporada en el archivo final. Y dado que Adobe tiene una marca registrada sobre XMP y retiene el control sobre la especificación, necesitamos incorporar también la biblioteca "xmpcore-5.1.3.jar"que ofrecen, para que la extracción de metadatos funcione correctamente. [12]

3.3. Casos de uso

El diagrama de casos de uso representa la forma en como un Cliente (Actor) opera con el sistema en desarrollo, además de la forma, tipo y orden en como los elementos interactúan (operaciones o casos de uso).

A continuación, se muestra un diagrama con los casos de uso u operaciones que puede realizar el usuario en esta aplicación.

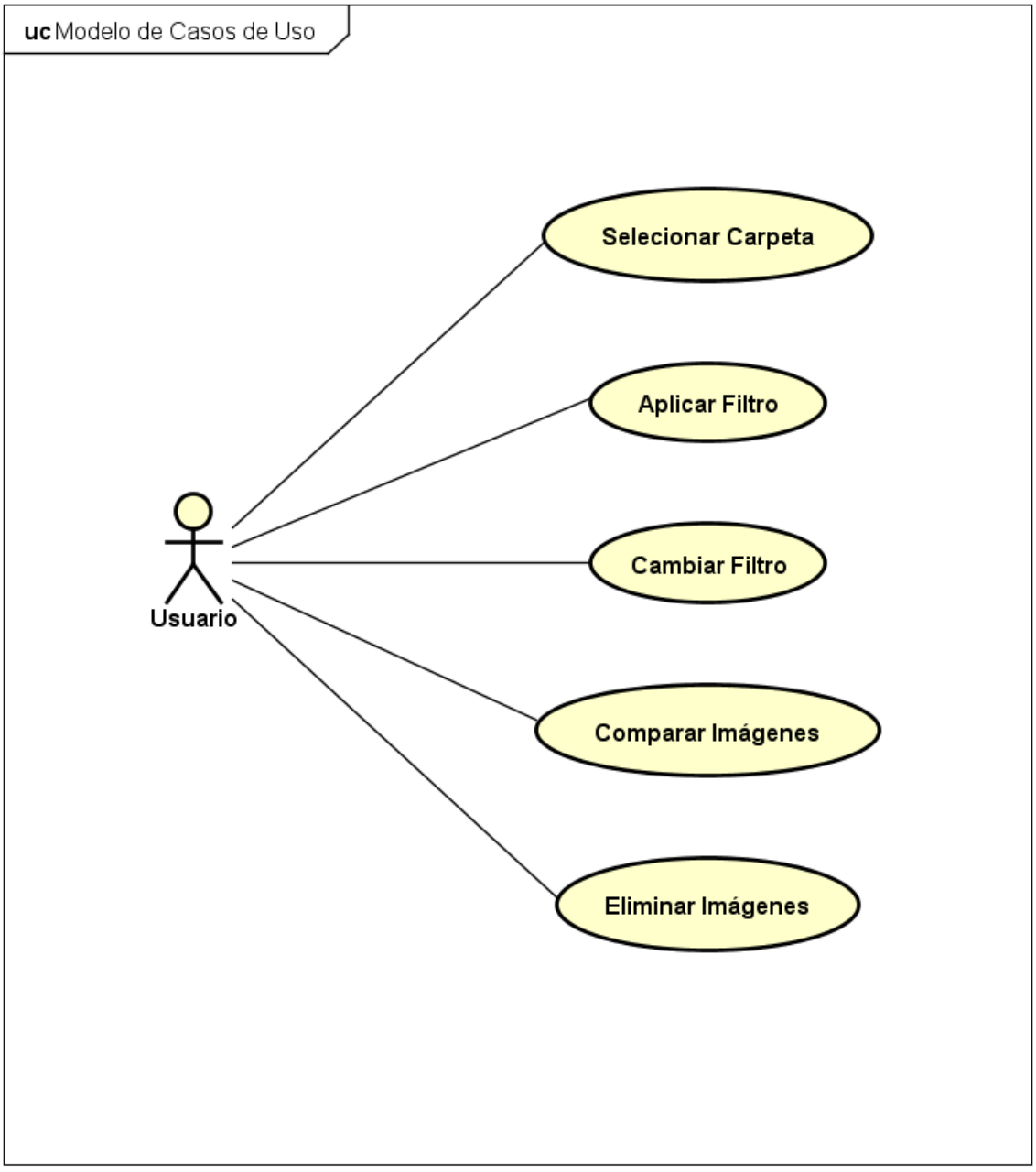

Figura 4: Diagrama de Casos de Uso.

3.4. Modelo de dominio

Un Modelo de Dominio es un artefacto construido con las reglas de UML durante la fase de concepción, en la tarea construcción del modelo de dominio, presentado como uno o más diagramas de clases. El modelo de dominio puede utilizarse para capturar y expresar el entendimiento ganado en un área bajo análisis como paso previo al diseño de un sistema, ya sea de software o de otro tipo. Similares a los mapas mentales utilizados en el aprendizaje, el modelo de dominio es utilizado como un medio para comprender el negocio al cual el sistema va a servir.

En este caso se muestra el siguiente modelo de dominio como una concepción de las clases que estarán presentes en la aplicación y sus interacciones entre ellas.

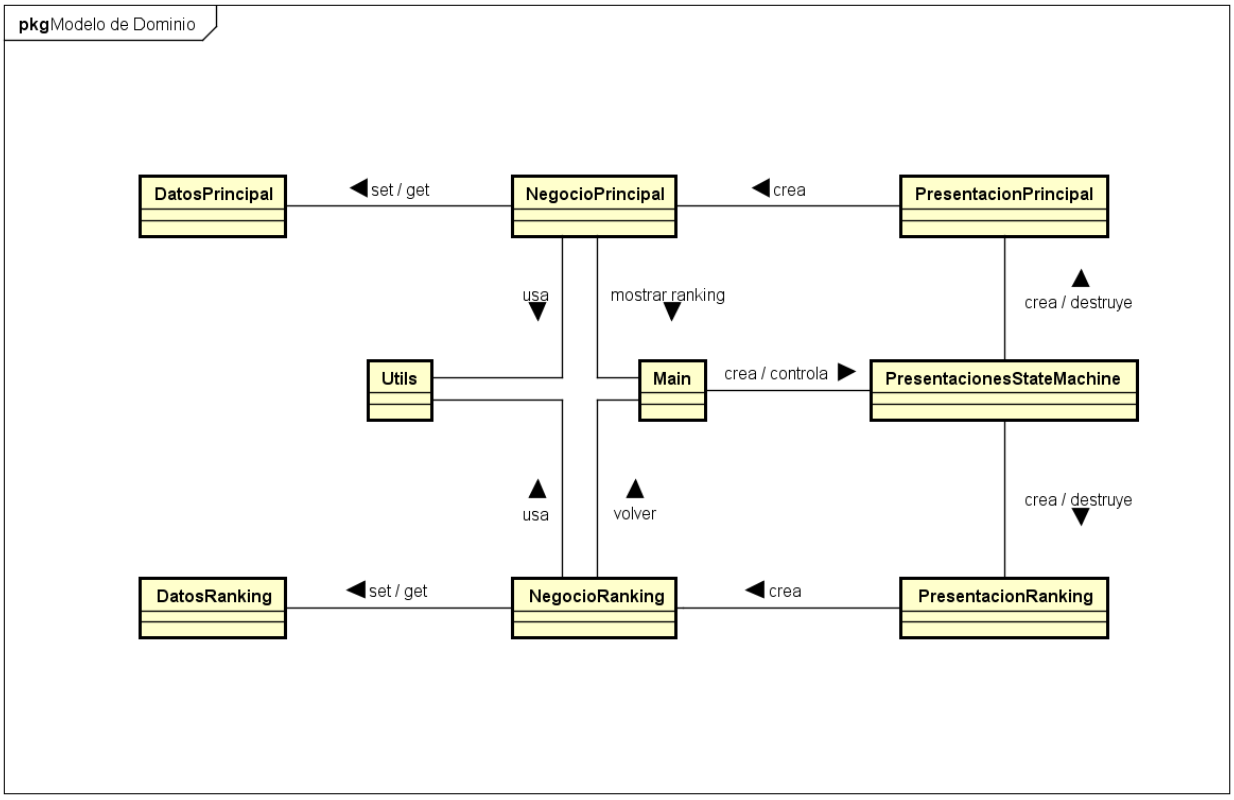

Figura 5: Diagrama del Modelo de Dominio.

3.5. Diagramas de secuencia

Estos diagramas muestran una interacción, que representa la secuencia de mensajes entre instancias de clases, componentes, subsistemas o actores. El tiempo fluye por el diagrama y muestra el flujo de control de un participante a otro.

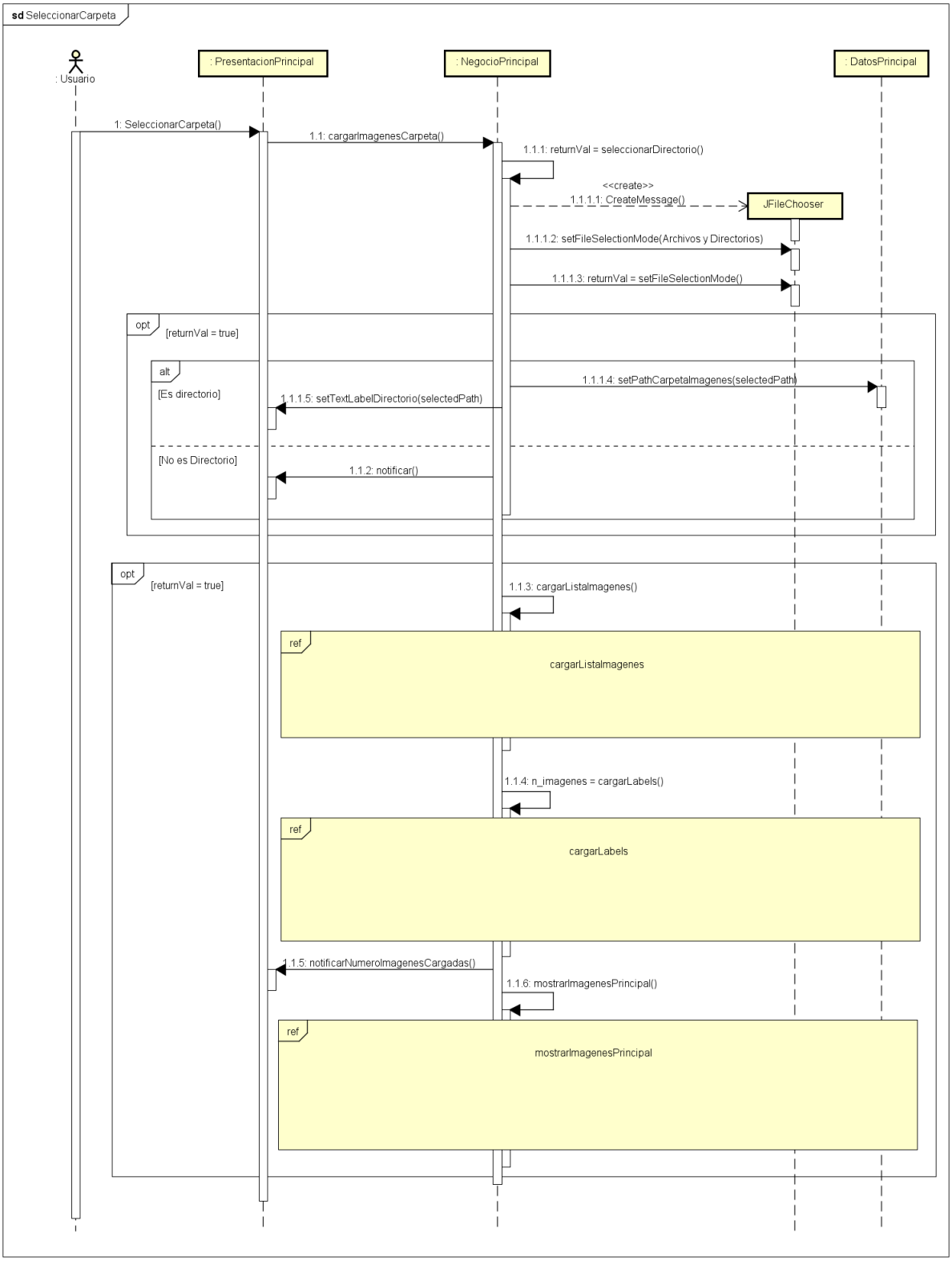

Figura 6: Diagrama de Secuencia del caso de uso SeleccionarCarpeta.

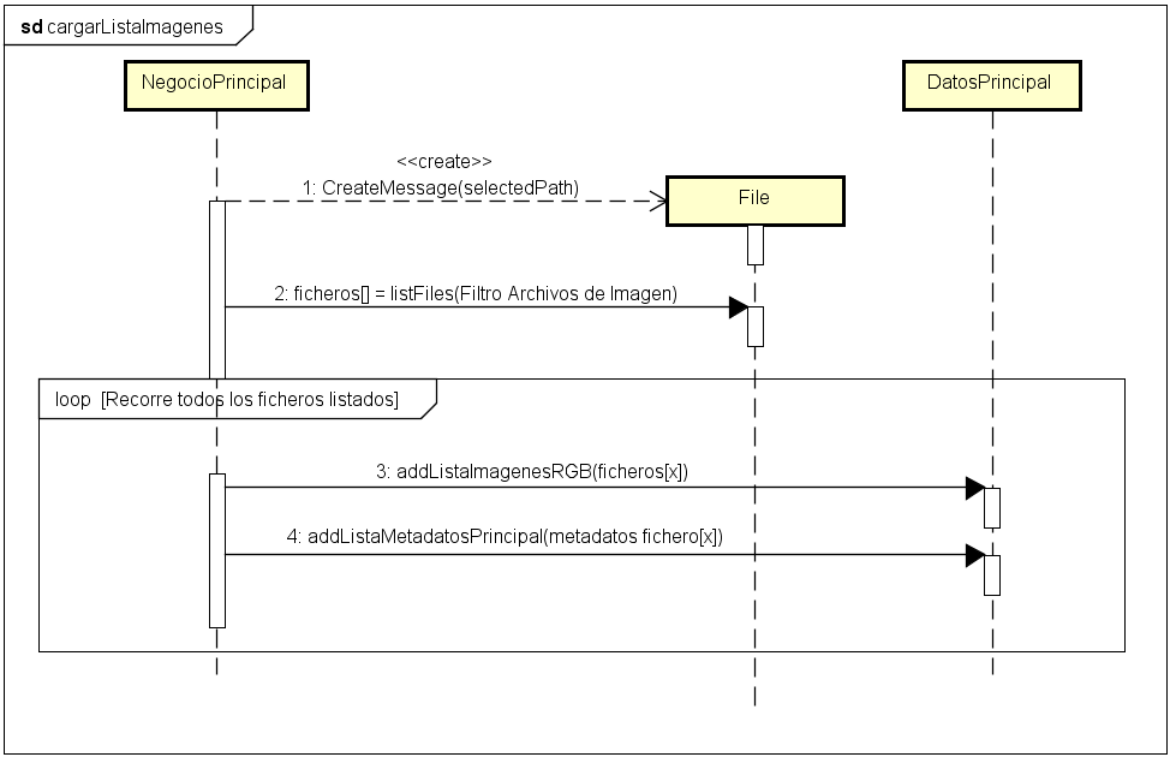

Figura 7: Diagrama de Secuencia de la referencia cargarListaImagenes.

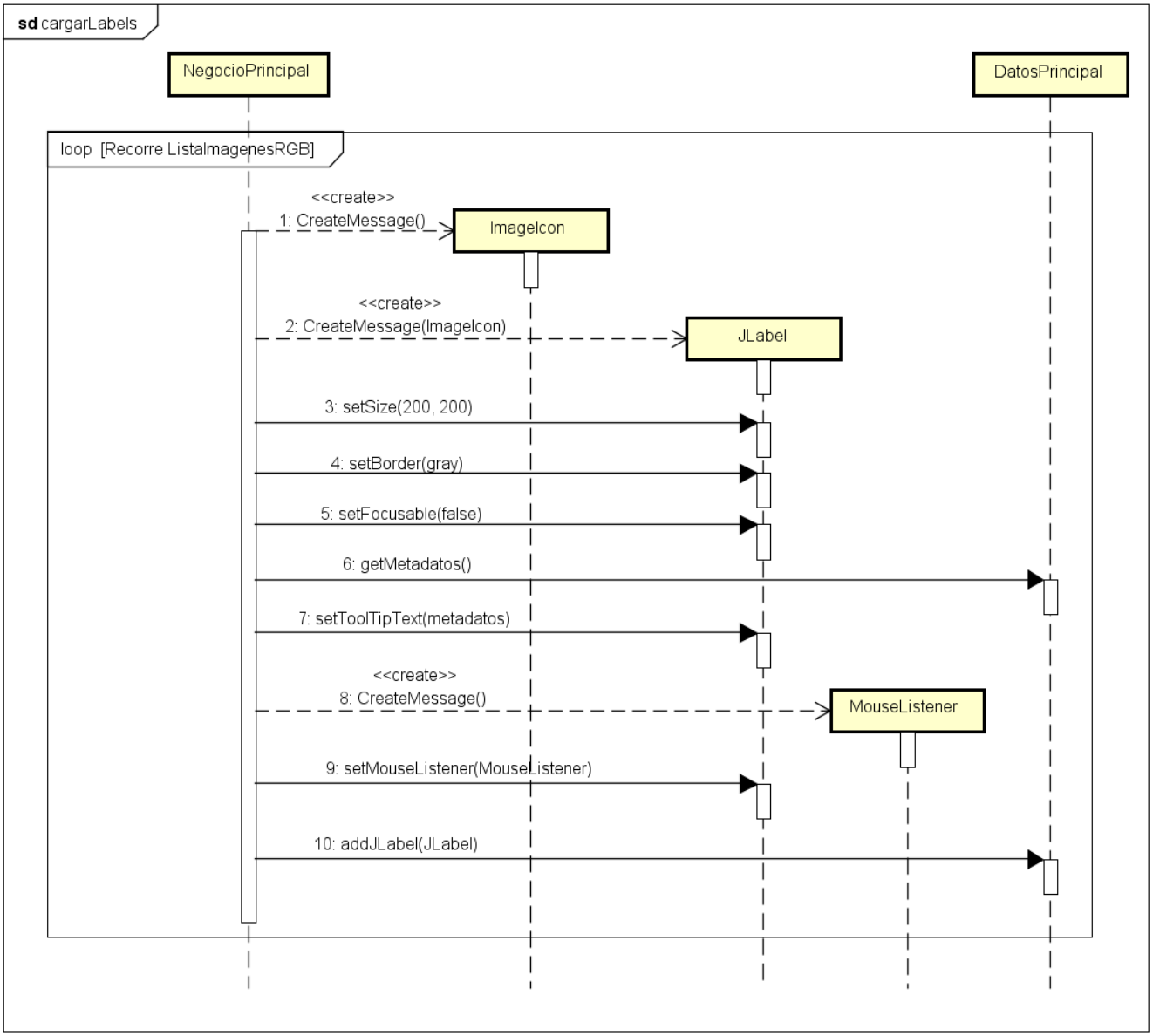

Figura 8: Diagrama de Secuencia de la referenciacargarLabels.

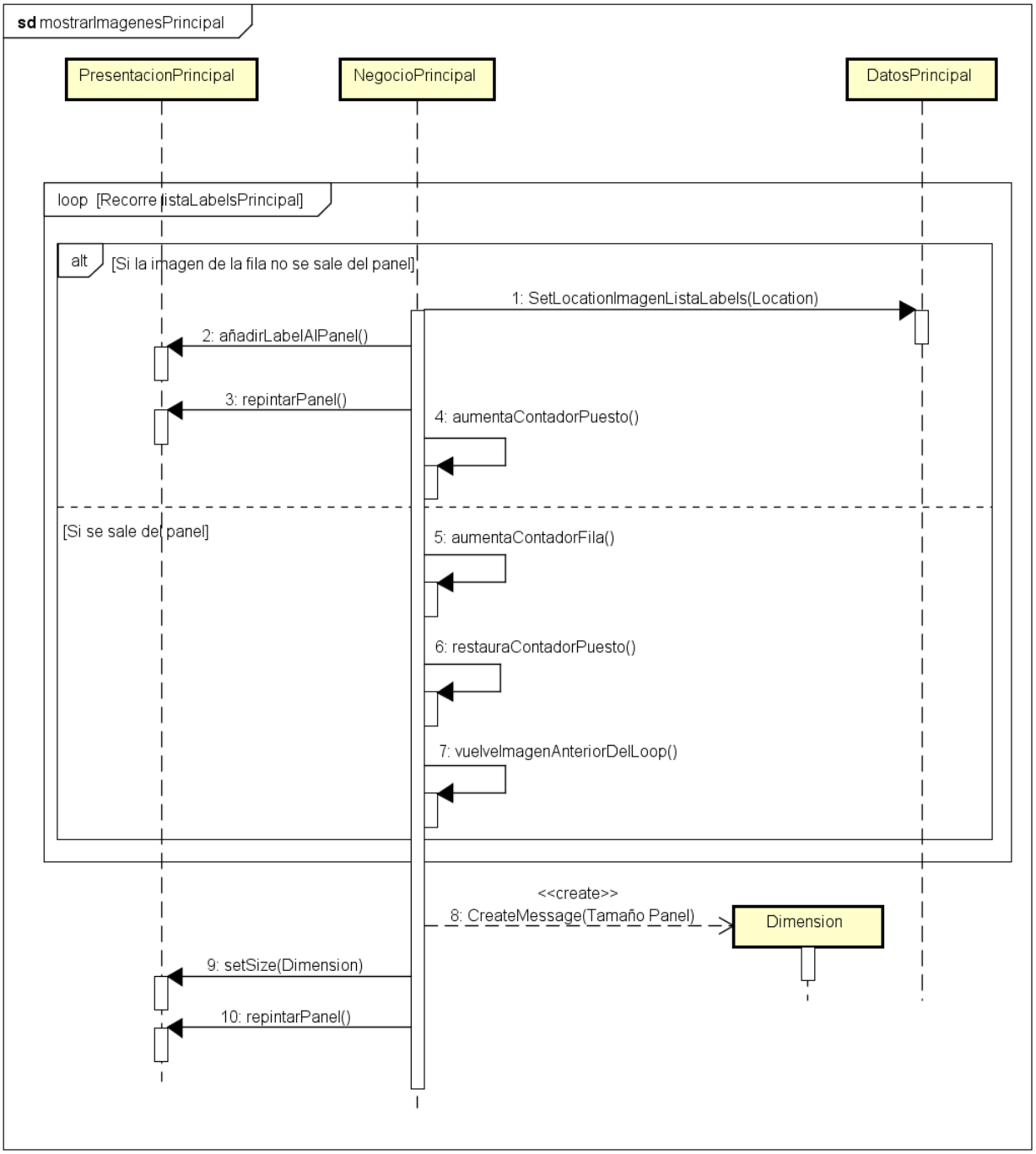

Figura 9: Diagrama de Secuencia de la referencia mostrarImagenesPrincipal.

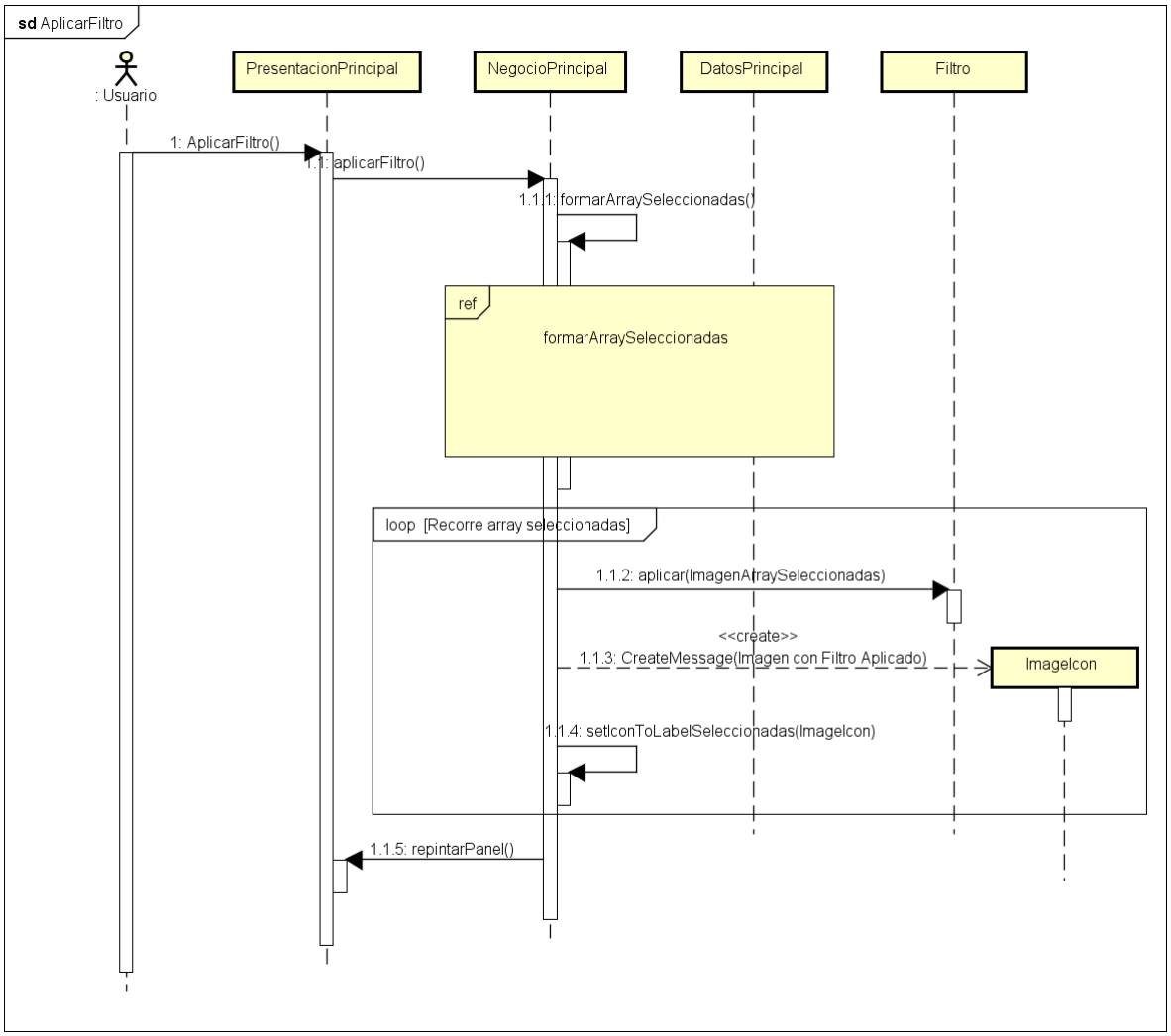

Figura 10: Diagrama de Secuencia del caso de uso AplicarFiltro.

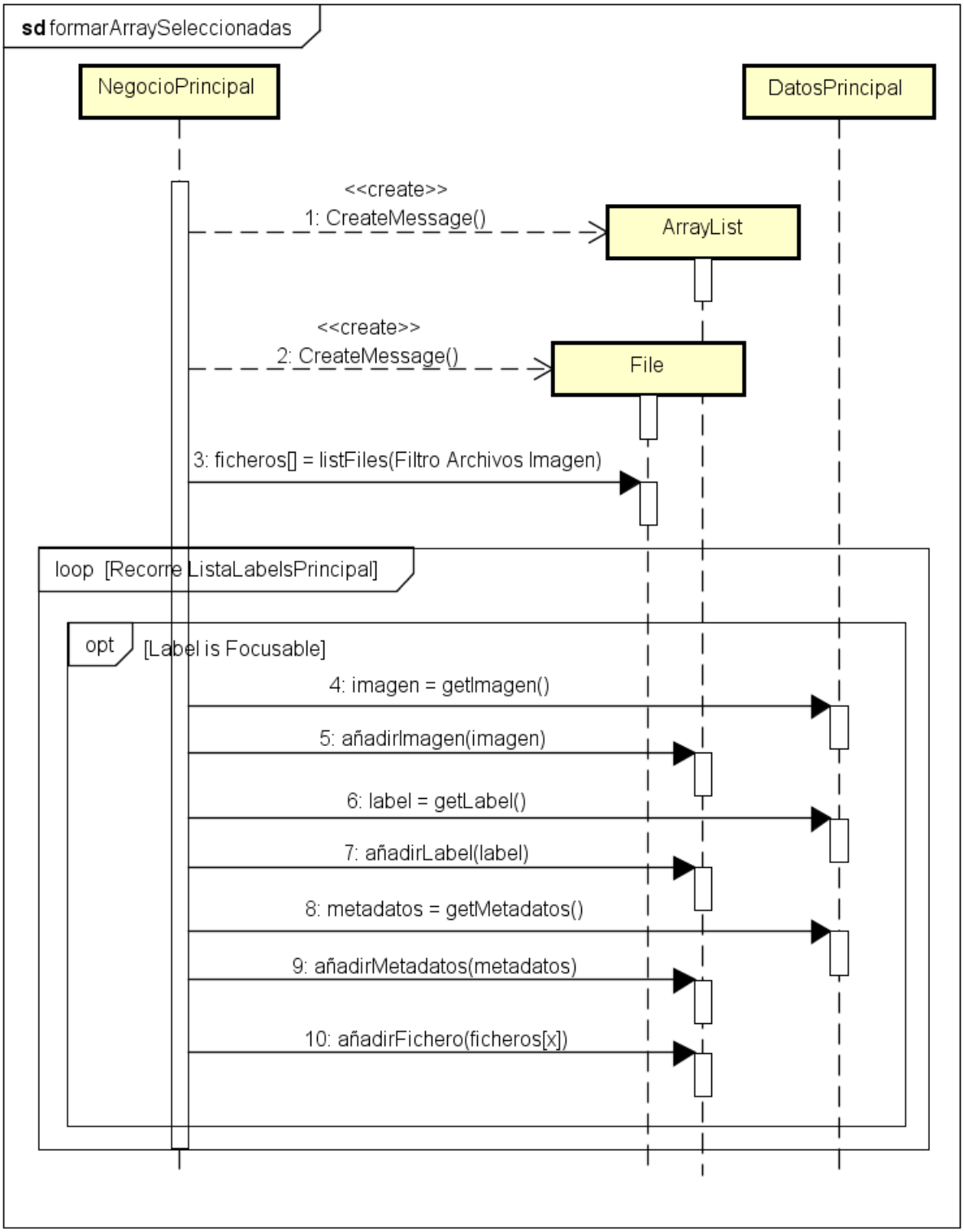

Figura 11: Diagrama de Secuencia de la referencia formarArraySeleccionadas.

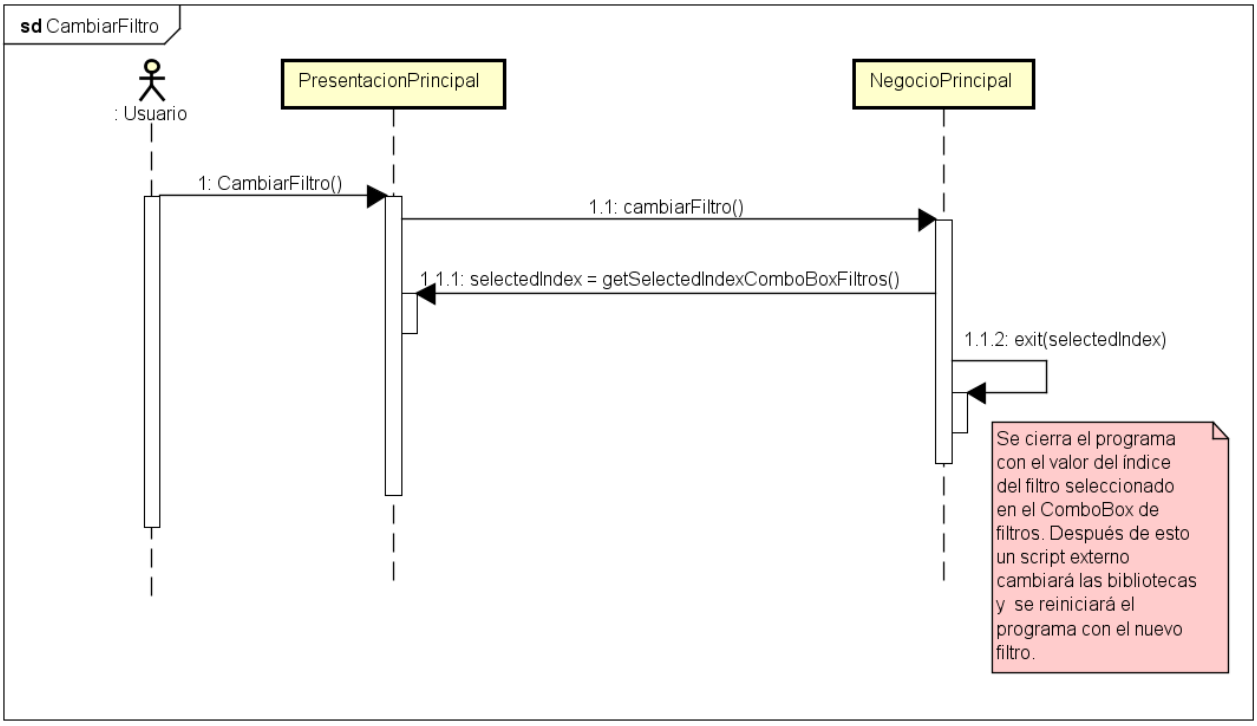

Figura 12: Diagrama de Secuencia del caso de uso CambiarFiltro.

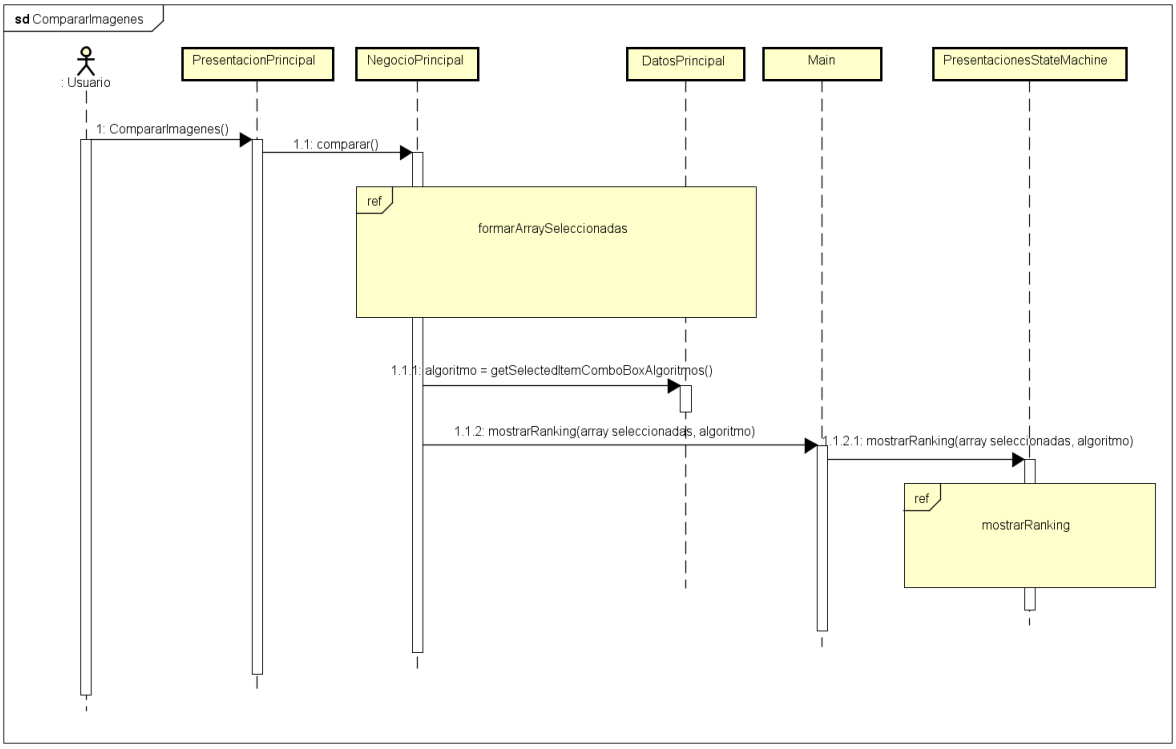

Figura 13: Diagrama de Secuencia del caso de uso CompararImagenes.

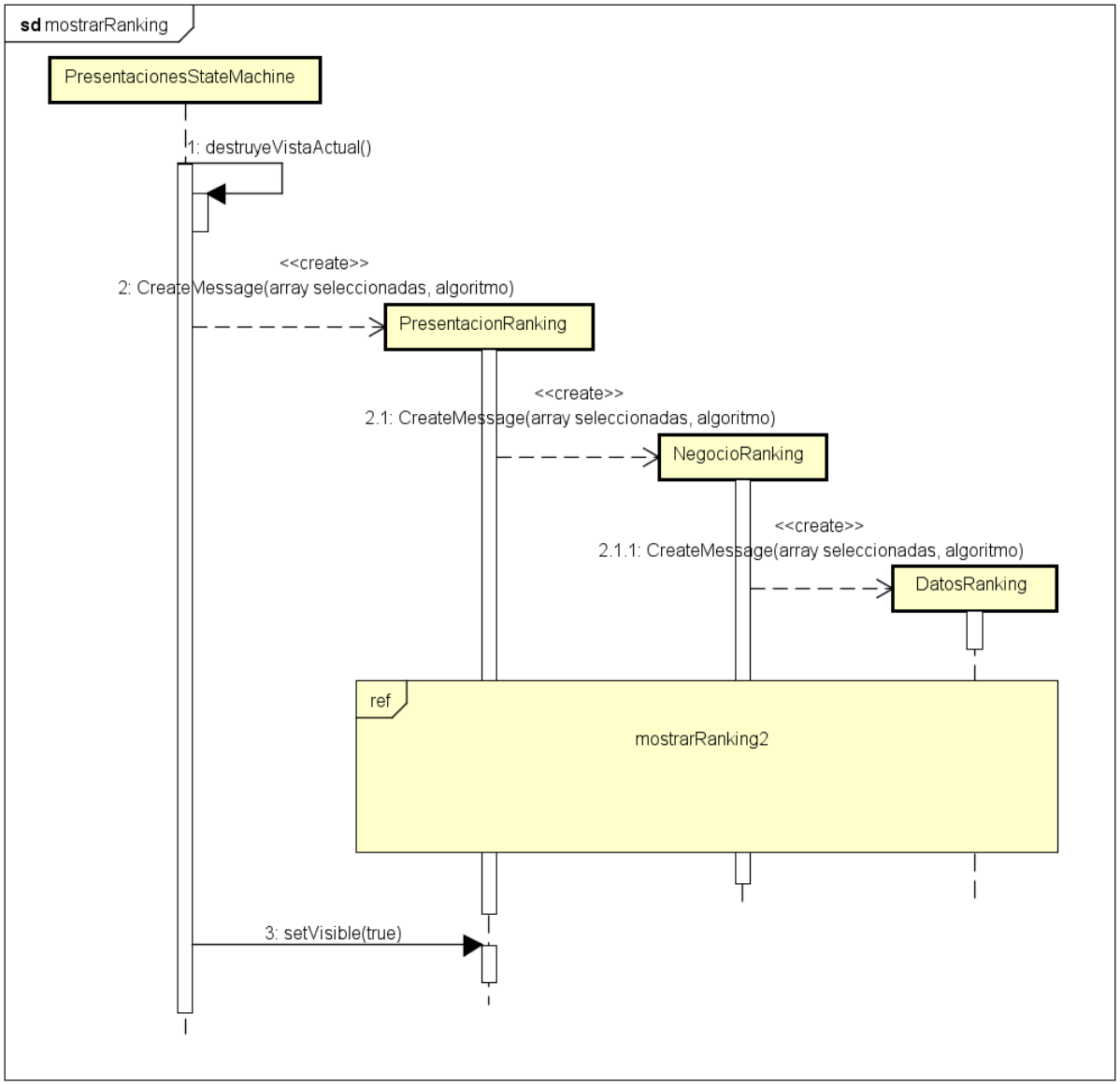

Figura 14: Diagrama de Secuencia de la referencia mostrarRanking.

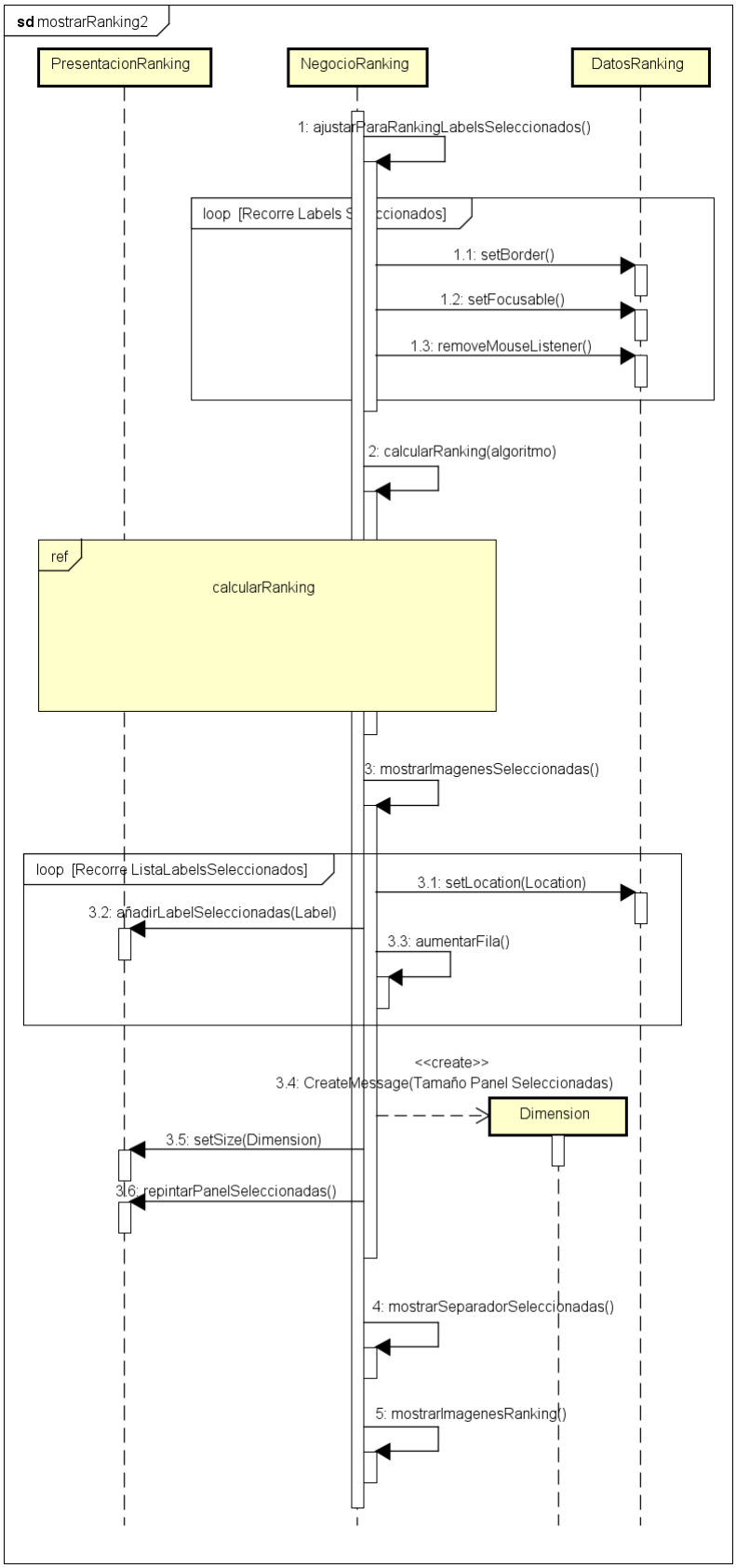

Figura 15: Diagrama de Secuencia de la referencia mostrarRanking2.

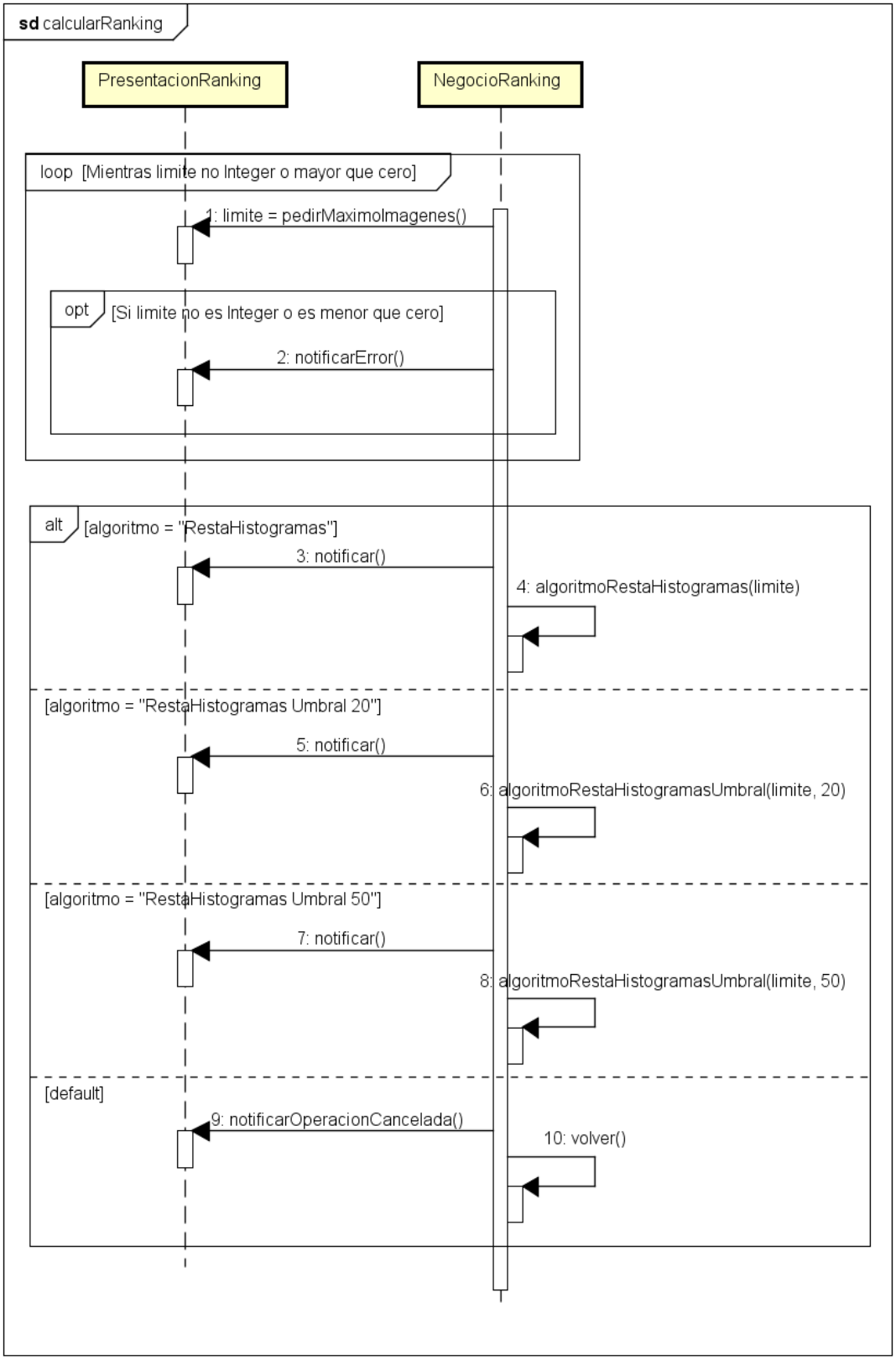

Figura 16: Diagrama de Secuencia de la referencia calcularRanking.

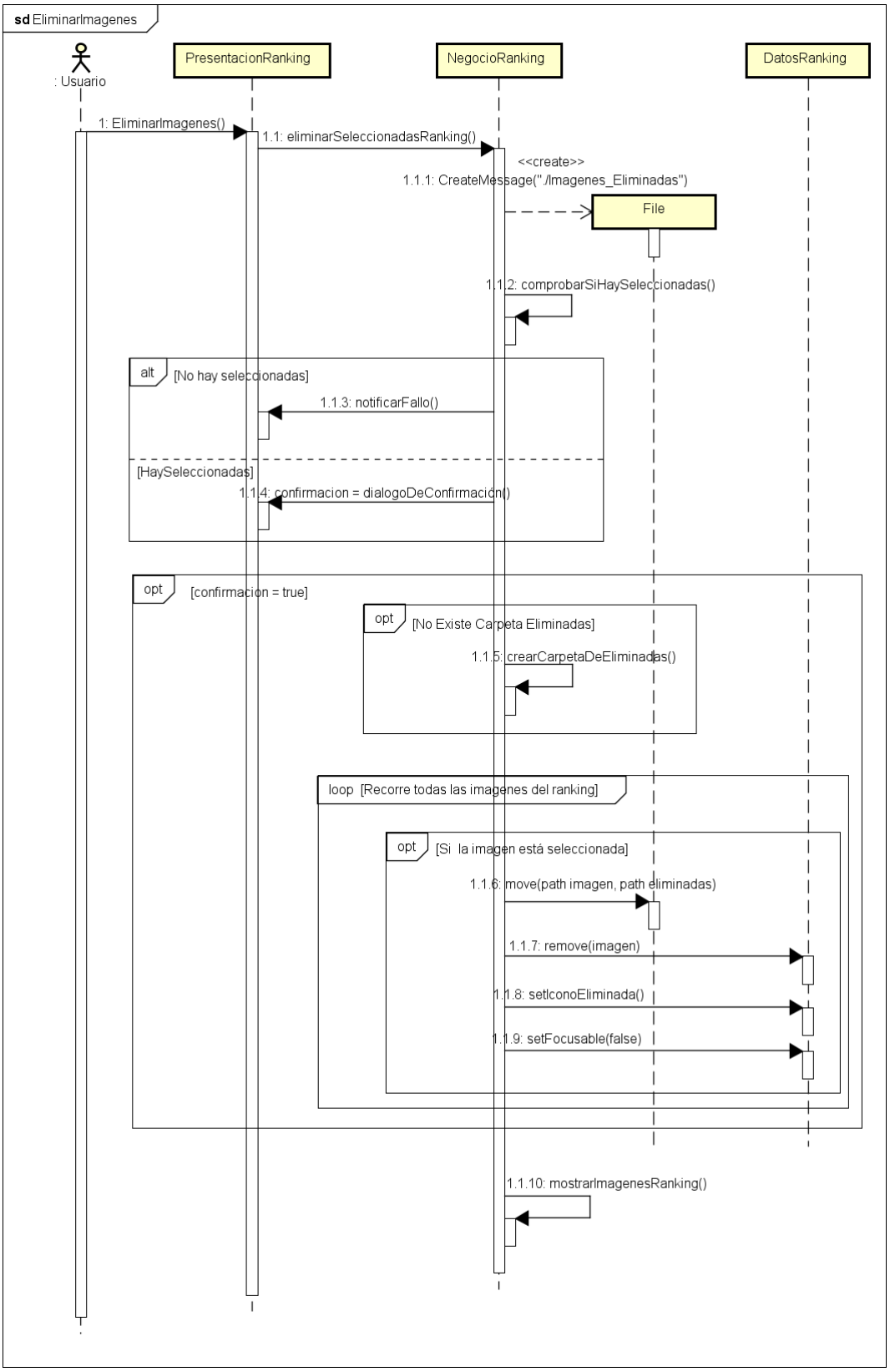

Figura 17: Diagrama de Secuencia del caso de uso EliminarImagenes.

3.6. Diagrama de clases

Un Diagrama de Clases de Diseño muestra la especificación para las clases software de una aplicación. Se podría decir que completa el Modelo de Dominio.

Incluye la siguiente información:

- Clases, asociaciones y atributos
- Interfaces, con sus operaciones y constantes
- Métodos
- Navegabilidad
- Dependencias

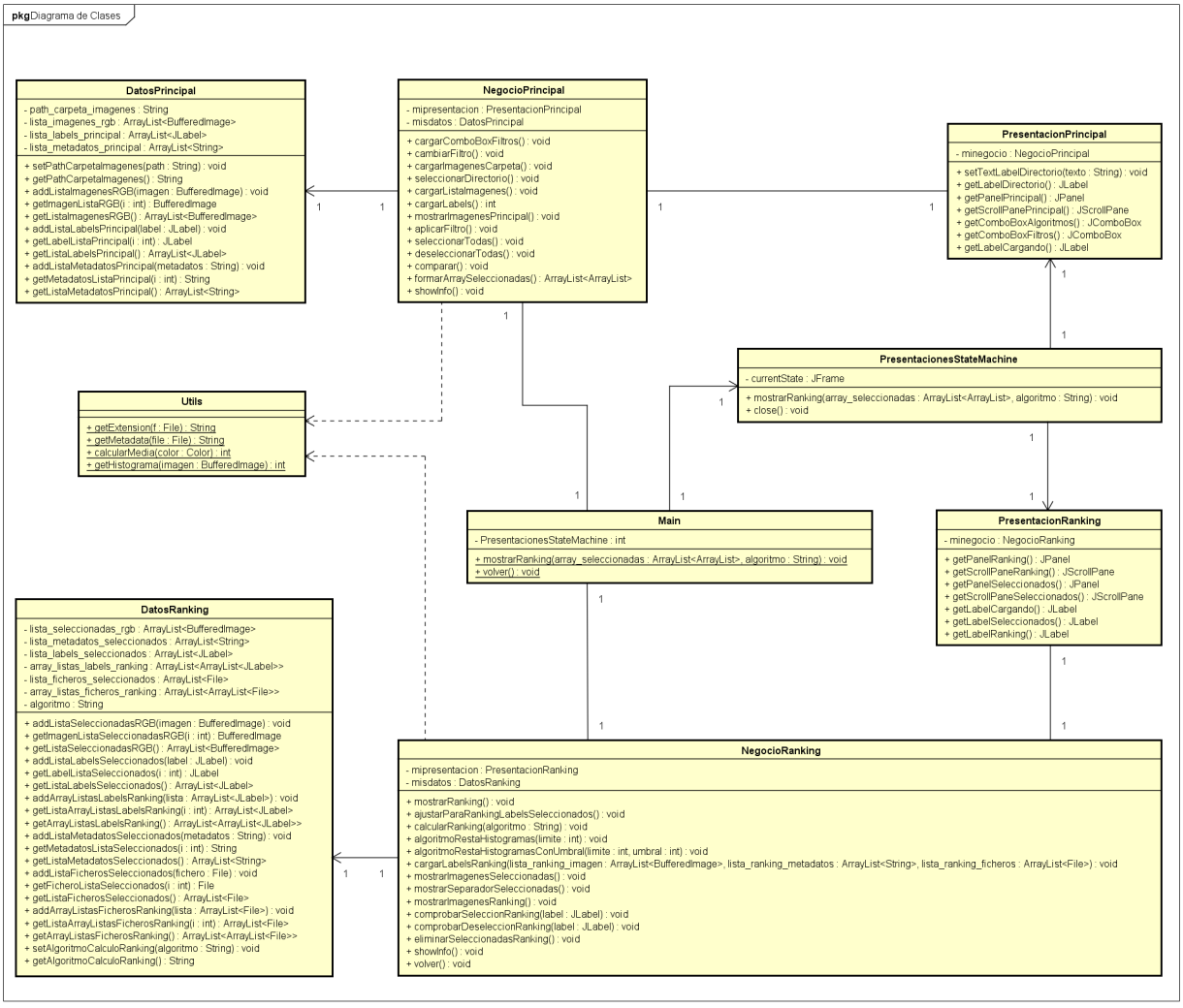

Figura 18: Diagrama de Clases de Diseño.

3.7. Diseño de las vistas

En esta sección se muestran los diseños gráficos realizados de las dos vistas de la aplicación (la principal y la del ranking). Se ha realizado mediante un software online especializado en este tipo de diseño.

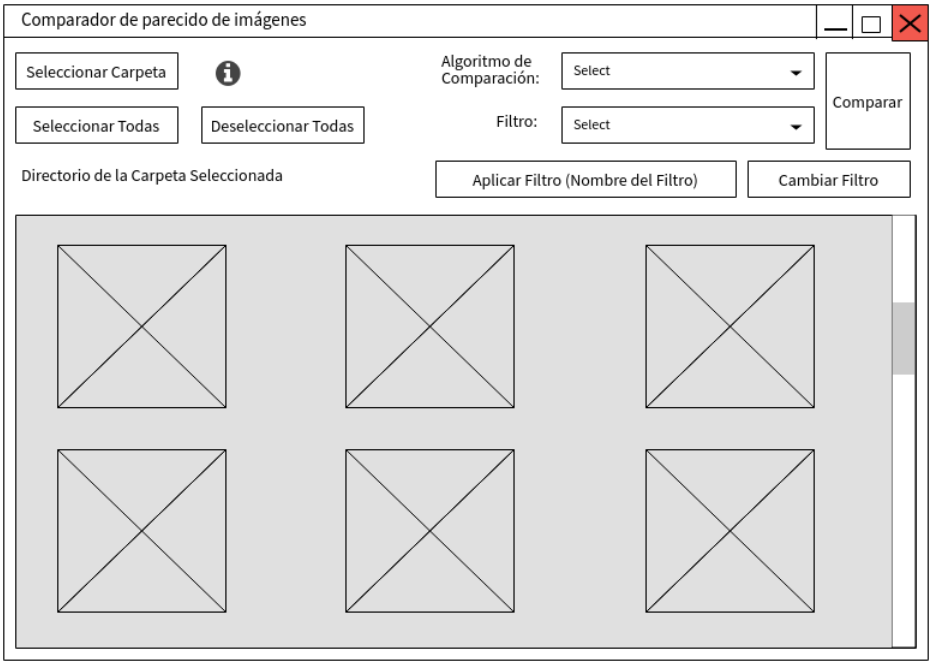

Figura 19: Boceto de la vista principal del programa.

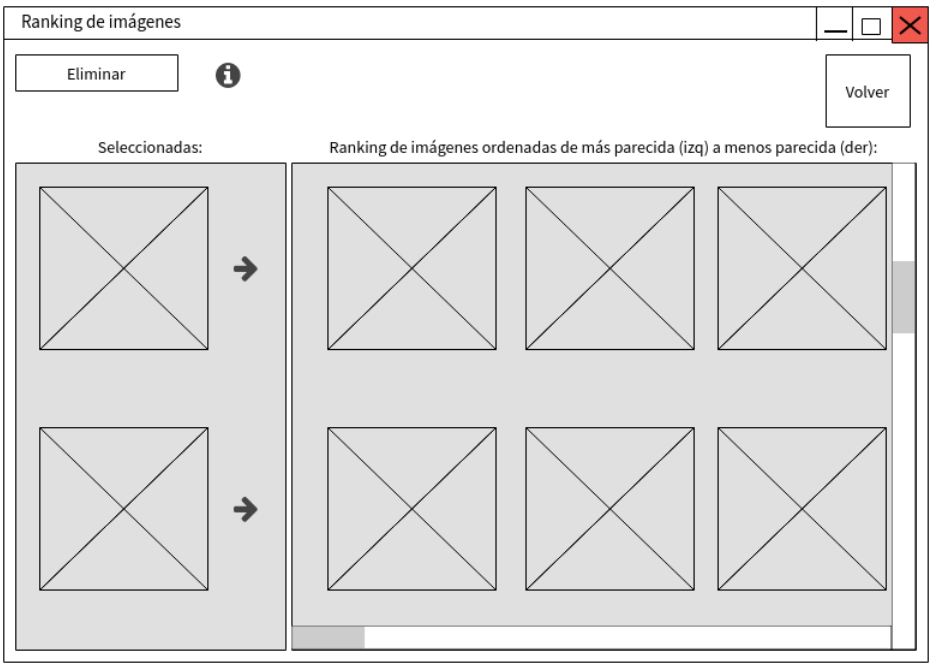

Figura 20: Boceto de la vista del ranking del programa.

4. Implementación

En esta sección se aborda la fase de implementación del proyecto. Se explicarán algunos detalles relevantes así como las diferentes dificultades que se encontraron en el proceso.

En esta fase se implementa el diseño realizado en el capítulo anterior de diseño, basado en el catálogo de requisitos de la anterior sección (Análisis). Como se ha especificado en el capítulo anterior, se utilizará NetBeans, junto a la biblioteca de interfaz gráfica Swing, entre otras tecnologías para la codificación de la aplicación.

A lo largo de la sección, durante la explicación de las partes más características del desarrollo, se tratarán los siguientes aspectos:

- Elementos clave de la funcionalidad
- Integración de las Bibliotecas utilizadas
- Interfaz de la aplicación

4.1. Arranque del programa

El programa se inicia con la clase **Main.java** y ésta construye la máquina de estados de las vistas que es definida por la clase **PresentacionesStateMachine.java** que es la que va a controlar el estado de las vistas e inicialmente deja creada y visible la pantalla principal.

```
public static void main(String args[]) {

    // Le da un estilo predefinido a la interfaz
   try {
        for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
            if ("Windows".equals(info.getName())) {
                javax.swing.UIManager.setLookAndFeel(info.getClassName());
                break;
            }
        }
    } catch (ClassNotFoundException ex) {
        java.util.logging.Logger.getLogger(PresentacionPrincipal.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
    } catch (InstantiationException ex) {
        java.util.logging.Logger.getLogger(PresentacionPrincipal.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
    } catch (IllegalAccessException ex) {
        java.util.logging.Logger.getLogger(PresentacionPrincipal.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
    } catch (javax.swing.UnsupportedLookAndFeelException ex) {
        java.util.logging.Logger.getLogger(PresentacionPrincipal.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
    }

    // Crea la maquina de estado de las vistas
    PresentacionesStateMachine = new PresentacionesStateMachine();
}
```

Figura 21: Gestión del tema de la interfaz y creación de la máquina de estados.

En el método de inicio del programa (main), se gestiona el tema que va a presentar la interfaz

de manera que se encuentre en armonía con el tema del sistema operativo en el que se está ejecutando. Por ejemplo, si se ejecuta en Windows, la interfaz presentará los mismos colores y formas de los elementos visuales característicos de Windows, y si por otra parte se ejecuta en alguna distribución de Linux, tomará otra temática con colores y formas diferentes.

```
public PresentacionesStateMachine() {
    java.awt.EventQueue.invokeLater(
    new Runnable() {
        public void run() {
            // se crea vistaPrincipal y se hace visible
            currentState = new PresentacionPrincipal();
            currentState.setVisible(true);
        }
    });
}
```

Figura 22: Construcción de la máquina de estados y creación de la interfaz principal.

En el constructor de la máquina de estados (PresentacionesStateMachine) vemos que la creación de la vista principal, se encuentra dentro de un *Runnable*. Esto es para que los escuchadores de eventos (*Listeners*) funcionen de manera óptima. El objeto de la vista principal se guarda en un atributo que marca la vista actualmente en curso (el estado de la máquina de estados).

```
public PresentacionPrincipal() {
    initComponents();
    minegocio = new NegocioPrincipal(this);
}
```

Figura 23: Contructor de la capa de presentación principal.

```
public NegocioPrincipal(PresentacionPrincipal vista){
    this.mipresentacion = vista;
    this.misdatos = new DatosPrincipal();
    cargarComboBoxFiltros();
}
```

Figura 24: Contructor de la capa de negocio principal.

```
public DatosPrincipal(){
    path_carpeta_imagenes = "";
    lista_imagenes_rgb = new ArrayList<>();
    lista_labels_principal = new ArrayList<>();
    lista_metadatos_principal = new ArrayList<>();
}
```

Figura 25: Contructor de la capa de datos principal.

Después la capa de presentación principal crea a la de negocio y ésta a la de datos. La vista

se pasa a si misma como argumento de la capa de negocio para que sea controlada por dicha capa.

4.2. Selección de la carpeta de imágenes

De la selección de la carpeta donde se encuentran las imágenes que queremos comparar, se encargan las clases de las 3 capas de la pantalla principal: **PresentacionPrincipal.java**, **NegocioPrincipal.java** y **DatosPrincipal.java**.

Tras pulsar el botón Seleccionar Carpeta, el primer método que se ejecuta es **cargarImagenesCarpeta**. Éste recopila una serie de métodos con otras funciones más concretas que en conjunto completan la operación de seleccionar el directorio y mostrar las imágenes que contiene. Después de cargar los Label, notifica el número de imágenes que se han cargado y que se mostrarán en el panel. Durante la ejecución de éste método, se hace invisible el panel para dejar mostrada una imagen indicando que se está procesando algo.

```
public void cargarImagenesCarpeta(){
    int returnVal;
    int n_imagenes;

    // volvemos invisible el scrollpane para que se vea el label con la imagen de cargando
        mipresentacion.getScrollPanePrincipal().setVisible(false);

    // Seleccion del directorio y guardado del path en el modelo
    returnVal = seleccionarDirectorio();

    // Si no se selecciona directorio (si se cancela), no se ejecutaran los siguientes metodos
    if(returnVal == JFileChooser.APPROVE_OPTION) {

        // Carga de las imagenes del directorio seleccionado en la lista de imagenes del modelo
        cargarListaImagenes();

        // carga las imagenes en la lista de labels del modelo
        n_imagenes = cargarLabels();

        JOptionPane.showMessageDialog(mipresentacion, "Se han cargado " + n_imagenes + " imagenes. "
                                                    + "A continuación se mostrarán en el panel.");

        // Volvemos a hacer visible el scrollpane antes de cargar las imagenes en el panel
        // para que no se bloque el programa en buble infinito
        mipresentacion.getScrollPanePrincipal().setVisible(true);
        mipresentacion.getComponent(0).validate();

        // coloca las imagenes en el panel principal
        mostrarImagenesPrincipal();
    }

    // Esto es por si acaso se cancela la seleccion de directorio
        mipresentacion.getScrollPanePrincipal().setVisible(true);
        mipresentacion.getComponent(0).validate();
}
```

Figura 26: Código del método cargarImagenesCarpeta.

Viendo ahora cada uno de los métodos más concretos, el de **seleccionarDirectorio** crea un objeto de selector de ficheros al que le añade un modo para poder seleccionar directorios. Al elegir un archivo o directorio, comprueba que sea directorio, en el caso de que sea archivo lo notifica diciendo que debe elegir una carpeta. Una vez verificado que es directorio, se graba el path de la carpeta en la capa de datos y se modifica el Label de la capa de presentación que indica el directorio actualmente seleccionado.

```
public int seleccionarDirectorio(){
    JFileChooser chooser = new JFileChooser();
    chooser.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
    int returnVal = chooser.showOpenDialog(mipresentacion);
    if(returnVal == JFileChooser.APPROVE_OPTION) {
        if (chooser.getSelectedFile().isDirectory()){
            misdatos.setPathCarpetaImagenes(chooser.getSelectedFile().toString());
            mipresentacion.setTextLabelDirectorio(misdatos.getPathCarpetaImagenes());
        }
        else{
            returnVal = JFileChooser.CANCEL_OPTION;
            JOptionPane.showMessageDialog(chooser, "Ha elegido para abrir un Fichero/Archivo. "
                                                + "Debe elegir Abrir un Directorio/Carpeta.");
            mipresentacion.setTextLabelDirectorio("Presione [Seleccionar Carpeta] para elegir "
                                                + "el directorio de las imágenes.");
        }
        return returnVal;
    }
    else {
        return returnVal;
    }
}
```

Figura 27: Código del método seleccionarDirectorio.

El siguiente, **cargarListaImagenes**, crea un objeto de manejo de ficheros al que le añade un filtro para que sólo liste algunos tipos de imagen comunes. Si se listan los ficheros, se recorren añadiéndolos como tipo imagen a un array de imágenes que se sitúa en la capa de datos. A su vez, también se van obteniendo los metadatos de cada imagen e incorporándolos a otro array de forma paralela. El tipo de lenguaje en el que se escriben los metadatos es HTML para que posteriormente se muestren de forma correcta en el ToolTip.

```
public void cargarListaImagenes(){

    File f = new File(misdatos.getPathCarpetaImagenes());

    misdatos.getListaImagenesRGB().clear();
    misdatos.getListaMetadatosPrincipal().clear();

    // Recopila las imagenes del directorio descartando los demas tipos de archivo
    File[] ficheros = f.listFiles(
    new FileFilter() {
        @Override
        public boolean accept(File f) {
            String extension = Utils.getExtension(f);
            if (extension != null) {
                if (extension.equals("jpeg") ||
                    extension.equals("png") ||
                    extension.equals("gif") ||
                    extension.equals("bmp") ||
                    extension.equals("jpg")) {
                        return true;
                } else {
                    return false;
                }
            }
            return false;
        }
    }
    );

    if (ficheros.length > 0){

        // carga imagenes en un array BufferedImage para poder tratar los valores RGB y obtiene los metadatos
        for (int x = 0; x < ficheros.length; x++){
            try{
                misdatos.addListaImagenesRGB(ImageIO.read(ficheros[x]));
                misdatos.addListaMetadatosPrincipal(
                        "<html>" + Utils.getMetadata(ficheros[x])
                        + "Height: " + misdatos.getImagenListaRGB(x).getHeight() + "px<br>"
                        + "Width: " + misdatos.getImagenListaRGB(x).getWidth() + "px<br>"
                        + "Type: " + Utils.getExtension(ficheros[x]).toUpperCase() + "<br>"
                        + "Location: " + ficheros[x].getParent() + "<br>"
                        + "</html>");
            }catch(IOException e){
                System.err.println("Alguna imagen no se pudo leer." + "\n\n" + "Exception message: " + e);
            }
        }
    }
    else {
        JOptionPane.showMessageDialog(mipresentacion, "Ha seleccionado un directorio que no contiene imágenes "
                                                    + "admitidas (JPEG, PNG, GIF, BMP, JPG)");
        mipresentacion.setTextLabelDirectorio(misdatos.getPathCarpetaImagenes() + " [Sin Imágenes]");
    }
}
```

Figura 28: Código del método cargarListaImagenes.

El de **cargarLabels** se recorren todas las imágenes que se han cargado antes. Se crea un Label que se va a configurar para mostrar una imagen a escala. En el Label se incluyen los metadatos de la imagen en forma de ToolTip y un Listener de clic de ratón para poder seleccionar la imagen. La imagen que se incluye en el Label se reescala utilizando unos cálculos matemáticos y un método especial de java (getScaledInstance). Finalmente se añade al array de Labels en la capa de datos el Label configurado.

```
public int cargarLabels(){
    int ancho;      // pixeles de anchura de la imagen reducida
    int alto;       // pixeles de altura de la imagen reducida
    JLabel jlabel;
    MouseListener mouseListener;
    misdatos.getListaLabelsPrincipal().clear();
    for (int x = 0; x < misdatos.getListaImagenesRGB().size(); x++){
        // Calcula el alto y ancho de la imagen reducida de forma que mantenga la relacion de aspecto
        if (misdatos.getImagenListaRGB(x).getHeight() > misdatos.getImagenListaRGB(x).getWidth()){
            alto = 200;
            ancho = misdatos.getImagenListaRGB(x).getWidth() * alto / misdatos.getImagenListaRGB(x).getHeight();
        }
        else {
            ancho = 200;
            alto = misdatos.getImagenListaRGB(x).getHeight() * ancho / misdatos.getImagenListaRGB(x).getWidth();
        }
        // Prepara la imagen que va a ser insertada en un label
        Icon icono;
        icono = new ImageIcon(misdatos.getImagenListaRGB(x).getScaledInstance(ancho, alto, Image.SCALE_DEFAULT));
        // Creacion de un label, configuracion y añadido al array de labels
        jlabel = new JLabel(icono);
        jlabel.setSize(200, 200);
        jlabel.setBorder(BorderFactory.createLineBorder(Color.gray));
        jlabel.setFocusable(false);
        jlabel.setToolTipText(misdatos.getMetadatosListaPrincipal(x)); // Incluye metadatos de la imagen al label
        mouseListener = new MouseListener() {
            @Override
            public void mouseClicked(MouseEvent e) {
                JLabel label = (JLabel) e.getComponent();
                if (label.isFocusable()){
                    label.setFocusable(false);
                    label.setBorder(BorderFactory.createLineBorder(Color.gray));
                }
                else {
                    label.setFocusable(true);
                    label.setBorder(BorderFactory.createLineBorder(Color.blue, 5));
                }
            }
            @Override
            public void mousePressed(MouseEvent e) {
            }
            @Override
            public void mouseReleased(MouseEvent e) {
            }
            @Override
            public void mouseEntered(MouseEvent e) {
            }
            @Override
            public void mouseExited(MouseEvent e) {
            }
        };
        jlabel.addMouseListener(mouseListener);
        misdatos.getListaLabelsPrincipal().add(jlabel);
    }
    return misdatos.getListaLabelsPrincipal().size();
}
```

Figura 29: Código del método cargarLabels.

Por último, **mostrarImagenesPrincipal** calcula la localización donde deben colocarse los Label con las imágenes y configura las dimensiones del panel donde se van a añadir todas. El algoritmo de posicionamiento de imágenes se verá más detalladamente en otra sección más adelante al final de esta de implementación.

```
public void mostrarImagenesPrincipal(){
    int fila = 0;   // contador de filas al ir añadiendo imagenes al panel
    int cont = 0;   // lleva el contador para no pasar del numero de columnas que caben

    mipresentacion.getPanelPrincipal().removeAll();

    // Coloca cada imagen donde debe (Contenedor cada imagen: 200x200, Margen: 10)
    for (int x = 0; x < misdatos.getListaLabelsPrincipal().size(); x++){

        if (mipresentacion.getScrollPanePrincipal().getWidth()-20 > ((cont+1)*210 + 10)){
            misdatos.getLabelListaPrincipal(x).setLocation(10 + 210*cont, 10 + 210*fila);
            mipresentacion.getPanelPrincipal().add(misdatos.getLabelListaPrincipal(x));
            mipresentacion.getPanelPrincipal().repaint();
            cont++;
        }
        else{
            fila++;
            cont=0;
            x--; // vuelve a la imagen que no pudo poner porque se salía del panel
        }
    }

    // Dimensiona el Panel Principal para contener todas las imagenes del directorio
    // quitamos 23px para que no aparezca la scrollbar horizontal
    Dimension d = new Dimension(mipresentacion.getScrollPanePrincipal().getWidth() - 23,
                                10 + 210 * (fila+1)/* le sumamos 1 para que cuente la
                                                      ultima fila para el scroll */);
    mipresentacion.getPanelPrincipal().setPreferredSize(d);
    mipresentacion.getPanelPrincipal().setSize(d);
    mipresentacion.getPanelPrincipal().repaint();
}
```

Figura 30: Código del método mostrarImagenesPrincipal.

4.3. Código del método

El método base para aplicar el filtro seleccionado a las imagenes seleccionadas es **aplicarFiltro**. Lo que hace es parecido al de **cargarLabels** pero este más bien modifica los que ya están cargados y mostrados. El modo de proceder es en primer lugar formar dos array con las imágenes seleccionadas, uno con los Label y otro con las imágenes originales, luego recorre el array de los Label y reescala las imágenes originales, después aplica el filtro y a continuación se incluye en los Label en sustitución de la anterior imagen.

```
public void aplicarFiltro(){
    int ancho;      // pixeles de anchura de la imagen reducida
    int alto;       // pixeles de altura de la imagen reducida
    ArrayList<BufferedImage> array_seleccionadas_rgb = formarArraySeleccionadas().get(0);
    ArrayList<JLabel> array_seleccionadas_labels = formarArraySeleccionadas().get(1);
    boolean mensaje = false;

    mipresentacion.getScrollPanePrincipal().setVisible(false);

    for (int x = 0; x < array_seleccionadas_labels.size(); x++){

        // Calcula el alto y ancho de la imagen reducida de forma que mantenga la relacion de aspecto
        if (array_seleccionadas_rgb.get(x).getHeight() > array_seleccionadas_rgb.get(x).getWidth()){
            alto = 200;
            ancho = array_seleccionadas_rgb.get(x).getWidth() * alto / array_seleccionadas_rgb.get(x).getHeight();
        }
        else {
            ancho = 200;
            alto = array_seleccionadas_rgb.get(x).getHeight() * ancho / array_seleccionadas_rgb.get(x).getWidth();
        }

        // Prepara la imagen que va a ser insertada en un label
        Icon icono;
        if ((array_seleccionadas_rgb.get(x).getType() == 6 || array_seleccionadas_rgb.get(x).getType() == 13)
                && "Escala de Grises".equals(Filtro.nombre_filtro)){
            icono = new ImageIcon(array_seleccionadas_rgb.get(x).getScaledInstance(ancho, alto, Image.SCALE_DEFAULT));
            mensaje = true;
        }
        else{
            icono = new ImageIcon(Filtro.aplicar(array_seleccionadas_rgb.get(x)).getScaledInstance(ancho, alto, Image.SCALE_DEFAULT));
        }

        // Creacion de un label, configuracion y añadido al array de labels
        array_seleccionadas_labels.get(x).setIcon(icono);
    }

    mipresentacion.getScrollPanePrincipal().setVisible(true);
    mipresentacion.getComponent(0).validate();

    if (mensaje)
            JOptionPane.showMessageDialog(mipresentacion, "Ha seleccionado para aplicar un filtro a un tipo de imagen "
                    + "a la que no es posible aplicarlo (Ej: PNG y GIF para la Escala de Grises).\n\nA las imágenes seleccionadas "
                    + "de diferente tipo al descrito se les habrá aplicado el filtro correctamente.");
}
```

Figura 31: Código del método aplicarFiltro.

El siguiente, **formarArraySeleccionadas**, es un método de tipo auxiliar, que al pricipio lista los archivos de imágenes que contiene la carpeta seleccionada, y luego recorre todas las imágenes del panel haciendo paradas sólo en las que se encuentran seleccionadas, para añadir todos sus datos (incluidos los archivos correspondientes de los listados al principio) en unos arrays que van a ser devueltos de retorno por el método.

```
public ArrayList<ArrayList> formarArraySeleccionadas() {

    ArrayList<BufferedImage> array_seleccionadas_rgb = new ArrayList<>();
    ArrayList<JLabel> array_seleccionadas_labels = new ArrayList<>();
    ArrayList<String> array_seleccionadas_metadatos = new ArrayList<>();
    ArrayList<File> array_seleccionadas_ficheros = new ArrayList<>();
    ArrayList<ArrayList> array_seleccionadas = new ArrayList<>();

    File f = new File(misdatos.getPathCarpetaImagenes());

    // Recopila las imagenes del directorio descartando los demas tipos de archivo
    File[] ficheros = f.listFiles(
    new FileFilter() {
        @Override
        public boolean accept(File f) {
            String extension = Utils.getExtension(f);
            if (extension != null) {
                if (extension.equals("jpeg") ||
                    extension.equals("png") ||
                    extension.equals("gif") ||
                    extension.equals("bmp") ||
                    extension.equals("jpg")) {
                        return true;
                } else {
                    return false;
                }
            }
            return false;
        }
    }
    );

    for (int x = 0; x < misdatos.getListaLabelsPrincipal().size(); x++){
        if (misdatos.getLabelListaPrincipal(x).isFocusable()){
            array_seleccionadas_rgb.add(misdatos.getImagenListaRGB(x));
            array_seleccionadas_labels.add(misdatos.getLabelListaPrincipal(x));
            array_seleccionadas_metadatos.add(misdatos.getMetadatosListaPrincipal(x));
            array_seleccionadas_ficheros.add(ficheros[x]);
        }
    }

    array_seleccionadas.add(array_seleccionadas_rgb);
    array_seleccionadas.add(array_seleccionadas_labels);
    array_seleccionadas.add(array_seleccionadas_metadatos);
    array_seleccionadas.add(array_seleccionadas_ficheros);

    return array_seleccionadas;
}
```

Figura 32: Código del método formarArraySeleccionadas.

4.4. Cambiar el filtro a aplicar

El método del código la aplicación en Java, no es en este caso el responsable de la mayoría de la funcionalidad. Para empezar **cambiarFiltro**, tras la pulsación del botón Cambiar Filtro sólo notifica que se va a cerrar la aplicación y que si se ha ejecutado mediante alguno de los script, se reiniciará con el nuevo filtro. Seguido de la notificación, cierra el programa con un número de código de salida, ese código es el número de la posición del filtro en la lista del ComboBox de filtros.

```
public void cambiarFiltro(){
    JOptionPane.showMessageDialog(mipresentacion, "Se cerrará la aplicación. "
                                + "Si la aplicación se ha ejecutado mediante\n"
                                + "el \"Launcher\", se volverá a iniciar con el "
                                + "filtro seleccionado.");
    System.exit(mipresentacion.getComboBoxFiltros().getSelectedIndex()+1);
}
```

Figura 33: Código del método cambiarFiltro.

Los siguientes scripts son los que poseen la mayor parte de la funcionalidad en el cambio de filtros. Su funcionamiento consiste en ejecutar la aplicación Java y esperar un código de respuesta. Una vez sabe el código, si es cero sale del programa definitivamente, pero si es otro número, indica el filtro seleccionado.

Lo que hace para saber qué filtro es el seleccionado, es listar todos los filtros de la carpeta donde están almacenados, y va recorriéndolos aumentando un contador que se parará cuando coincida con el código devuelto por el programa Java y se obtiene el nombre del filtro donde se ha parado.

Una vez que se sabe el nombre de la biblioteca, se elimina la presente en la carpeta de bibliotecas del programa y se copia en su lugar la nueva seleccionada renombrándolo con el nombre genérico “filtro.jar”para que funcione correctamente. Finalmente itera el bucle y vuelve a ejecutar la aplicación Java pero esta vez con el nuevo filtro.

```
@echo off
:bucle
set cont=1
set filtro=""
java -jar RankingImagenes.jar
set exitcode=%ERRORLEVEL%
if %exitcode% == 0 (exit)
dir Filtros /b > filtros.txt
FOR /F "tokens=*" %%A IN (filtros.txt) DO (
set filtro=%%A
if %exitcode% == %cont% (goto :break)
set /a cont=%cont%+1
)
:break
DEL /Q filtros.txt
COPY /Y ".\Filtros\%filtro%" ".\lib\filtro.jar"
goto :bucle
```

Figura 34: Script en Batch (Windows) para realizar el cambio de bibliotecas de filtros.

```
#!/bin/bash
typeset -i cont
while true;
do
    cont=1
    filtro=""
    java -jar RankingImagenes.jar
    exitcode=$?
    if [ $exitcode == 0 ]; then
        exit
    fi
    ls Filtros > filtros.txt
    for var in $(cat filtros.txt);
    do
        echo $var
        filtro=$var
        if [ $exitcode == $cont ]; then
            break
        fi
        cont=$cont+1
    done
    rm -f filtros.txt
    cp -f ./Filtros/$filtro ./lib/filtro.jar
done
```

Figura 35: Script en Shell (Linux) para realizar el cambio de bibliotecas de filtros.

Para que no haya errores con las bibliotecas de filtros y sean compatibles con la aplicación, se debe cumplir:

1. La biblioteca que contiene el filtro a ejecutarse debe tener siempre el nombre «filtro.jar» para que pueda ejecutarse la aplicación.

2. Para cambiar el filtro copiar en la carpeta «lib» otra biblioteca Jar con el filtro deseado renombrándola como «filtro.jar» y reemplazándola por la anterior.

3. Las bibliotecas de los filtros deben tener una estructura y nombres de paquete, clase, atributo, método específicos para que guarden compatibilidad con la aplicación.

```java
package com.rankingimagenes;

import java.awt.Color;
import java.awt.image.BufferedImage;

/**
 *
 * @author Autor
 */
public class Filtro {

    public static String nombre_filtro = "Nombre del Filtro";

    /**
     * Aplica el filtro a una imagen.
     * @param imagen Imagen que va a la que va a ser aplicada el filtro.
     * @return Devuelve la imagen convertida.
     */
    public static BufferedImage aplicar(BufferedImage imagen){
        //Variables que almacenarán los píxeles
        ...

        //Recorremos la imagen píxel a píxel
        for( int i = 0; i < imagen.getWidth(); i++ ){
                    for( int j = 0; j < imagen.getHeight(); j++ ){
                        ...
                    }
        }
            // Retornamos la imagen
            return imagen;
        }
}
```

Figura 36: Código ejemplo de la estructura a tener la biblioteca del filtro.

4.5. Comparar las imágenes y construir el ranking

Esta es la funcionalidad principal de la aplicación y por la que fue ideada su desarrollo, por ello en esta sección se explicarán bastantes métodos y se incluirán algunos constructores debido al cambio de vista.

La operación empieza tras la pulsación del botón Comparar, ejecutándose el método **comparar**. Este método hace uso de **formarArraySeleccionadas** visto anteriormente, luego comprueba si hay imágenes seleccionadas y en caso negativo salta una notificación sobre ello y se para la operación. Por otro lado, si hay imágenes seleccionadad, se llama al método **mostrarRanking** de la clase **Main** pasando como parámetros el array de seleccionadas con todos los datos de las imágenes, y el algoritmo que se encontraba seleccionado para realizar la comparación.

```
public void comparar(){

    ArrayList<ArrayList> array_seleccionadas;
    array_seleccionadas = formarArraySeleccionadas();

    // Si no se seleccionan imágenes no puede comparar
    if (array_seleccionadas.get(0).isEmpty())
        JOptionPane.showMessageDialog(mipresentacion,
                "No se ha seleccionado ninguna imagen para comparar.");
    else{
        Main.mostrarRanking(array_seleccionadas,
            mipresentacion.getComboBoxAlgoritmos().getSelectedItem().toString());
    }
}
```

Figura 37: Código del método comparar.

El método **mostrarRanking** de la clase **Main** es muy sencillo y es de transición por protocolo. Llama al método **mostrarRanking** de la clase **PresentacionesStateMachine** pasando los mismos parámetros.

```
public static void mostrarRanking(ArrayList<ArrayList> array_seleccionadas, String algoritmo){
    PresentacionesStateMachine.mostrarRanking(array_seleccionadas, algoritmo);
}
```

Figura 38: Código del método mostrarRanking de la clase Main.

El método **mostrarRanking** de la clase **PresentacionesStateMachine** es muy parecido a lo que realiza el constructor de dicha clase, la diferencia es que en este caso se destruye la vista del estado actual de la máquina de estados (la vista principal), y se crea la nueva vista

dentro de un *Runnable* para que funcionen los Eventos de la interfaz como ya se ha comentado anteriormente.

```
public void mostrarRanking(ArrayList<ArrayList> array_seleccionadas, String algoritmo) {
    currentState.dispose(); // se destruye vistaPrincipal

    java.awt.EventQueue.invokeLater(
    new Runnable() {
        public void run() {
            // se crea vistaRanking y se hace visible
            currentState = new PresentacionRanking(array_seleccionadas, algoritmo);
            currentState.setVisible(true);
        }
    });
}
```

Figura 39: Código del método mostrarRanking de la clase PresentacionesStateMachine.

A continuación se muestran los constructores de las 3 capas implicados en la nueva vista del ranking. Los parámetros que se estaban pasando (array de seleccionadas y algoritmo), siguen hasta que son almacenadas en la capa de **Datos**.

```
public PresentacionRanking(ArrayList<ArrayList> array_seleccionadas, String algoritmo) {
    initComponents();
    minegocio = new NegocioRanking(this, array_seleccionadas, algoritmo);
}
```

Figura 40: Constructor de la clase PresentacionRanking.

```
public NegocioRanking(PresentacionRanking vista, ArrayList<ArrayList> array_seleccionadas, String algoritmo){
    this.mipresentacion = vista;
    this.misdatos = new DatosRanking(array_seleccionadas, algoritmo);
}
```

Figura 41: Constructor de la clase NegocioRanking.

```
public DatosRanking(ArrayList<ArrayList> array_seleccionadas, String algoritmo){
    lista_seleccionadas_rgb = array_seleccionadas.get(0);
    lista_labels_seleccionados = array_seleccionadas.get(1);
    lista_metadatos_seleccionados = array_seleccionadas.get(2);
    lista_ficheros_seleccionados = array_seleccionadas.get(3);
    array_listas_labels_ranking = new ArrayList<>();
    array_listas_ficheros_ranking = new ArrayList<>();
    this.algoritmo = algoritmo;
}
```

Figura 42: Constructor de la clase DatosRanking.

El siguiente es un método que recopila una serie de métodos con otras funciones más con-

cretas que en conjunto completan la operación, la de calcular el ranking y mostrarlo. También interactúa con la capa de Presentación para hacer invisible los paneles mientras se realizan los cálculos y se muestra una imagen que indica que la operación está en proceso.

```
public final void mostrarRanking(){
    mipresentacion.getLabelSeleccionados().setVisible(false);
    mipresentacion.getLabelRanking().setVisible(false);
    mipresentacion.getScrollPaneSeleccionados().setVisible(false);
    mipresentacion.getScrollPaneRanking().setVisible(false);
    mipresentacion.getLabelCargando().setVisible(true);
    mipresentacion.getComponent(0).validate();

    ajustarParaRankingLabelsSeleccionados();
    calcularRanking(misdatos.getAlgoritmoCalculoRanking());
    mostrarImagenesSeleccionadas();
    mostrarSeparadorSeleccionadas();
    mostrarImagenesRanking();

    mipresentacion.getLabelSeleccionados().setVisible(true);
    mipresentacion.getLabelRanking().setVisible(true);
    mipresentacion.getScrollPaneSeleccionados().setVisible(true);
    mipresentacion.getScrollPaneRanking().setVisible(true);
    mipresentacion.getLabelCargando().setVisible(false);
    mipresentacion.getComponent(0).validate();
}
```

Figura 43: Código del método mostrarRanking de la clase NegocioRanking.

El método **ajustarParaRankingLabelsSeleccionados** recorre los Label seleccionados (del array de seleccionados que se pasó por parámetro) y les pone borde negro de grosor 2, les quita el atributo *Focusable* (por el cual se sabe si están seleccionadas), y les quita los Listeners porque a las seleccionadas no les hace falta en la interfaz del ranking.

```
public void ajustarParaRankingLabelsSeleccionados(){

    for (int x = 0; x < misdatos.getListaLabelsSeleccionados().size(); x++){
        misdatos.getLabelListaSeleccionados(x).setBorder(BorderFactory.createLineBorder(Color.black, 2));
        misdatos.getLabelListaSeleccionados(x).setFocusable(false);
        misdatos.getLabelListaSeleccionados(x).removeMouseListener(misdatos.getLabelListaSeleccionados(x).getMouseListeners()[1]
    }
}
```

Figura 44: Código del método ajustarParaRankingLabelsSeleccionados.

El método **calcularRanking** comienza pidiendo introducir el límite de imágenes a mostrar en el panel de las imágenes del ranking, y si se cancela se regresa a la vista principal con todo restablecido de inicio. Si se introduce correctamente, se llama al método del algoritmo elegido mediante un *switch-case* y se le pasa como parámetro el límite establecido, y en su caso algun otro necesario para el algoritmo.

Los métodos que ejecutan los algoritmos se verán más adelante en una sección de algoritmos.

```
public void calcularRanking(String algoritmo){
    int limite = 0;
    String aux = "";

    while(limite <= 0){
        try{
            aux=JOptionPane.showInputDialog("Escriba el numero máximo de imágenes más parecidas que\n"
                                            + "tendrá el ranking por cada imagen seleccionada. (Ej: 5)");
            if (aux == null){ // Si se presiona boton cancelar se para
                algoritmo = "";
                break;
            }
            else
                limite = Integer.parseInt(aux);
        }catch(HeadlessException | NumberFormatException e){
            JOptionPane.showMessageDialog(mipresentacion, "No ha introducido el número correctamente.\n"
                                                + "Por favor, asegúrese de no utilizar letras.");
        }
    }

    switch (algoritmo) {
        case "Resta Histogramas":
            JOptionPane.showMessageDialog(mipresentacion, "Se procederá a calcular el ranking de imágenes "
                    + "más parecidas según\nel algoritmo seleccionado. Esto podría tardar un tiempo.");
            algoritmoRestaHistogramas(limite);
            break;
        case "Resta Histogramas Umbral 20":
            JOptionPane.showMessageDialog(mipresentacion, "Se procederá a calcular el ranking de imágenes "
                    + "más parecidas según\nel algoritmo seleccionado. Esto podría tardar un tiempo.");
            algoritmoRestaHistogramasConUmbral(limite, 20);
            break;
        case "Resta Histogramas Umbral 50":
            JOptionPane.showMessageDialog(mipresentacion, "Se procederá a calcular el ranking de imágenes "
                    + "más parecidas según\nel algoritmo seleccionado. Esto podría tardar un tiempo.");
            algoritmoRestaHistogramasConUmbral(limite, 50);
            break;
        default:
            // Se ha cancelado la creacion del ranking y vuelve a la vistaPrincipal
            JOptionPane.showMessageDialog(mipresentacion, "Operación de comparación cancelada.");
            volver();
            break;
    }
}
```

Figura 45: Código del método calcularRanking.

El método mostrado a continuación, añade una localización a los Label de las imágenes seleccionadas recibidas de la vista Principal y los añade al panel de seleccionadas.

```
public void mostrarImagenesSeleccionadas(){
    int fila = 0;   // contador de filas al ir añadiendo imagenes al panel

    // Coloca cada imagen donde debe (Contenedor cada imagen: 200x200, Margen Horizontal: 10, Margen Vertical: 50)
    for (int x = 0; x < misdatos.getListaLabelsSeleccionados().size(); x++){
        misdatos.getLabelListaSeleccionados(x).setLocation(10, 50 + 250*fila);
        mipresentacion.getPanelSeleccionados().add(misdatos.getLabelListaSeleccionados(x));
        fila++;
    }

    // Dimensiona el Panel Principal para contener todas las imagenes del directorio
    Dimension d = new Dimension(10 + 200 + 10, // 10 margen + 200 imagen + 10 margen
                                50 + (200+50) * misdatos.getListaLabelsSeleccionados().size()); // 50 de margen vertical
    mipresentacion.getPanelSeleccionados().setPreferredSize(d);
    mipresentacion.getPanelSeleccionados().setSize(d);

    mipresentacion.getPanelSeleccionados().repaint();
}
```

Figura 46: Código del método mostrarImagenesSeleccionadas.

Este método tiene más que una funcionalidad visual. Coloca unas flechas a la derecha de cada Label del panel de seleccionadas y también un separador horizontal para separar cada uno.

```
public void mostrarSeparadorSeleccionadas(){
    int fila = 0;   // contador de filas al ir añadiendo labels al panel
    JLabel label;
    JLabel label2;

    // Coloca cada "-->" donde debe (Contenedor cada label: 20x10, Margen Horizontal: 10, Margen Vertical: 145)
    for (int x = 0; x < misdatos.getListaLabelsSeleccionados().size(); x++){
        label = new JLabel();
        label.setSize(20, 10);
        label.setText("-->");
        label.setLocation(10 + 200 + 10, 50 + 95 + (10+95+50+95)*fila);
        mipresentacion.getPanelSeleccionados().add(label);
        fila++;
    }

    fila = 0;
    // Coloca cada separador donde debe (Contenedor cada label: 260x2, Margen Horizontal: 0)
    for (int x = 0; x < misdatos.getListaLabelsSeleccionados().size() - 1; x++){
        label2 = new JLabel();
        label2.setSize(260, 2);
        label2.setText("-----------------------------------------------------------------------------------------------------");
        label2.setLocation(0, 50 + 200 + 24 + (26+200+24)*fila);
        mipresentacion.getPanelSeleccionados().add(label2);
        fila++;
    }

    // Dimensiona el Panel Principal para contener los separadores
    Dimension d = new Dimension(10 + 200 + 10 + 20, // 10 margen + 200 imagen + 10 margen + 20 (-->)
                                50 + (200+50) * misdatos.getListaLabelsSeleccionados().size()); // 30 de margen vertical
    mipresentacion.getPanelSeleccionados().setPreferredSize(d);
    mipresentacion.getPanelSeleccionados().setSize(d);

    mipresentacion.getPanelSeleccionados().repaint();
}
```

Figura 47: Código del método mostrarSeparadorSeleccionadas.

El siguiente hace lo mismo que el que encontramos en la clase **NegocioPrincipal** con la diferencia de que en el ranking se muestran siempre un una misma fila sin importar que se salgan del panel por la derecha activando así un ScrollBar horizontal. También pone un separador de filas.

```
public void mostrarImagenesRanking(){
    int cont = 0;    // lleva el contador para no pasar del numero de columnas que caben
    int max = 0;     // maximo tamano de entre todas las listas de labels ranking
    JLabel label;

    // Coloca cada imagen donde debe (ContenedorCadaImagen: 200x200, Margen: 10)
    for (int y = 0; y < misdatos.getArrayListasLabelsRanking().size(); y++){
        for (int x = 0; x < misdatos.getListaArrayListasLabelsRanking(y).size(); x++){

            // Mete la imagen
            misdatos.getListaArrayListasLabelsRanking(y).get(x).setLocation(10 + 210*cont, 50 + 250*y);
            mipresentacion.getPanelRanking().add(misdatos.getListaArrayListasLabelsRanking(y).get(x));

            // Mete el separador debajo de la imagen si no es la ultima fila
            if (y < misdatos.getArrayListasLabelsRanking().size() - 1)
            {
                label = new JLabel();
                label.setSize(225, 2);
                label.setText("------------------------------------------------------------------------");
                label.setLocation(210*cont, 50 + 200 + 24 + (26+200+24)*y);
                mipresentacion.getPanelRanking().add(label);
            }

            cont++;
        }
        cont=0;
        if (misdatos.getListaArrayListasLabelsRanking(y).size() > max)
            max = misdatos.getListaArrayListasLabelsRanking(y).size();
    }
    // Dimensiona el Panel Principal para contener todas las imagenes del directorio
        Dimension d = new Dimension(10 + 210 * max,
                                    50 + 250 * misdatos.getListaLabelsSeleccionados().size());
        mipresentacion.getPanelRanking().setPreferredSize(d);
        mipresentacion.getPanelRanking().setSize(d);

        mipresentacion.getPanelRanking().repaint();
}
```

Figura 48: Código del método mostrarImagenesRanking.

4.6. Eliminar las imágenes seleccionadas en el ranking

Para esta funcionalidad sólo hay un método que empieza comprobando si hay imágenes seleccionadas, y una vez comprueba que las hay, pide una confirmación para no realizar eliminaciones no deseadas. Una vez confirmado, si no existe la "papelera" local (carpeta Imagenes_Eliminadas) la crea, y en caso contrario comienza a recorrer las imágenes del ranking moviendo a la "papelera" local las que estaban seleccionadas. A la vez que las mueve, también cambia la imagen del Label afectado por el mensaje "Imagen Eliminada". Durante toda la ejecución de este método, se hacen invisibles los paneles para que se muestre una imagen notificando que se está procesando la operación.

```
public void eliminarSeleccionadasRanking(){
    boolean hay_seleccionadas = false;
    int opcion = JOptionPane.NO_OPTION;
    File carpetaEliminadas = new File("./Imagenes_Eliminadas");
    mipresentacion.getLabelSeleccionados().setVisible(false);
    mipresentacion.getLabelRanking().setVisible(false);
    mipresentacion.getScrollPaneSeleccionados().setVisible(false);
    mipresentacion.getScrollPaneRanking().setVisible(false);
    mipresentacion.getLabelCargando().setVisible(true);
    mipresentacion.getComponent(0).validate();
    for (int x = 0; x < misdatos.getArrayListasLabelsRanking().size(); x++){
        for (int y = 0; y < misdatos.getListaArrayListasLabelsRanking(x).size(); y++){
            if (misdatos.getListaArrayListasLabelsRanking(x).get(y).isFocusable()){
                hay_seleccionadas = true;
                break;
            }
        }
    }
    if (!hay_seleccionadas)
        JOptionPane.showMessageDialog(mipresentacion, "No se ha seleccionado ninguna imagen para eliminar.");
    else
        opcion = JOptionPane.showConfirmDialog(mipresentacion, "¿Seguro que desea eliminar "
                + "las imágenes seleccionadas?", "Confirmación", JOptionPane.YES_NO_OPTION);
    if(opcion == JOptionPane.YES_OPTION){
        if (!carpetaEliminadas.exists())
            carpetaEliminadas.mkdir();
        for (int x = 0; x < misdatos.getArrayListasLabelsRanking().size(); x++){
            for (int y = 0; y < misdatos.getListaArrayListasLabelsRanking(x).size(); y++){
                if (misdatos.getListaArrayListasLabelsRanking(x).get(y).isFocusable()){
                    try {
                        Files.move(Paths.get(misdatos.getListaArrayListasFicherosRanking(x).get(y).getAbsolutePath()),
                                Paths.get(new File(carpetaEliminadas,
                                    misdatos.getListaArrayListasFicherosRanking(x)
                                            .get(y).getName()).getAbsolutePath()),
                                    StandardCopyOption.REPLACE_EXISTING);
                    } catch (Exception e) {}
                    misdatos.getListaArrayListasFicherosRanking(x)
                        .remove(y);
                    misdatos.getListaArrayListasFicherosRanking(x)
                        .add(y, new File("/rankingimagenes/Imagen-Eliminada.png"));
                    misdatos.getListaArrayListasLabelsRanking(x).get(y)
                        .setIcon(new ImageIcon(getClass().getResource("/rankingimagenes/Imagen-Eliminada.png")));
                    misdatos.getListaArrayListasLabelsRanking(x).get(y)
                        .setFocusable(false);
                }
            }
        }
    }
    mostrarImagenesRanking();
    mipresentacion.getLabelSeleccionados().setVisible(true);
    mipresentacion.getLabelRanking().setVisible(true);
    mipresentacion.getScrollPaneSeleccionados().setVisible(true);
    mipresentacion.getScrollPaneRanking().setVisible(true);
    mipresentacion.getLabelCargando().setVisible(false);
    mipresentacion.getComponent(0).validate();
}
```

Figura 49: Código del método eliminarSeleccionadasRanking.

4.7. Algoritmos utilizados para calcular el ranking

Los dos métodos con un algoritmo de comparación de imágenes que se van a mostrar, son muy similares entre ellos porque sólo difieren en que el segundo tiene un umbral que limita los datos a coger de los Histogramas para el cálculo, pero los resultados tras la comparación pueden ser bastante diferentes según las imágenes seleccionadas.

El algoritmo se basa en la obtención de los datos de los histogramas de las imágenes, y una

vez obtenidos, a restarlos entre sí. Se van restando los histogramas de las imágenes a comparar con el de la imagen de referencia. La imagen de referencia es una de las seleccionadas de la vista principal, y las imágenes a comparar son el resto de seleccionadas. La imagen de referencia va rotando entre todas las seleccionadas hasta que la han sido todas. La resta se produce como la producida entre dos vectores, componente a componente (cada componente sería una tonalidad de color de las 256 que existen en RGB) y el resultado se da en valor absoluto. Una vez restadas se calcula la media de los resultados de las restas y lo que se compara es esta media.

Entonces, algoritmo con umbral haría lo mismo pero en vez de disponer de los 256 tonos, tendría ese número reducido en el número que especifique el umbral. Esto hace que no se consideren colores muy extremos como el blanco de la nieve o el negro de la noche.

```
public void algoritmoRestaHistogramas(int limite){
    ArrayList<int[][]> histogramas = new ArrayList<>();
    ArrayList<BufferedImage> lista_ranking_imagen = new ArrayList<>();
    ArrayList<String> lista_ranking_metadatos = new ArrayList<>();
    ArrayList<File> lista_ranking_ficheros = new ArrayList<>();
    int resta_histogramas[] = new int[256];
    int media[] = new int[misdatos.getListaSeleccionadasRGB().size()];
    ArrayList<Integer> lista_media = new ArrayList<>();
    int aux;

    for(int i = 0; i < misdatos.getListaSeleccionadasRGB().size(); i++){
        histogramas.add(Utils.getHistograma(misdatos.getImagenListaSeleccionadasRGB(i)));
    }
    for(int i = 0; i < misdatos.getListaSeleccionadasRGB().size(); i++){
        for(int j = 0; j < misdatos.getListaSeleccionadasRGB().size(); j++){
            media[j] = 0;
            for(int y = 0; y < 256; y++){
                resta_histogramas[y] = Math.abs(histogramas.get(i)[4][y] - histogramas.get(j)[4][y]);
                media[j] += resta_histogramas[y]/3;
            }
            lista_media.add(media[j]);
        }
        while(!lista_media.isEmpty()){
            aux = 2147483647;
            for(int j = 0; j < lista_media.size(); j++){
                if (lista_media.get(j) < aux){
                    aux = lista_media.get(j);
                }
            }
            for(int j = 0; j < lista_media.size(); j++){
                if (lista_media.get(j) == aux){
                    lista_media.remove(j);
                    break;
                }
            }
            for(int j = 0; j < misdatos.getListaSeleccionadasRGB().size(); j++){
                if (media[j] == aux){
                    lista_ranking_imagen.add(misdatos.getImagenListaSeleccionadasRGB(j));
                    lista_ranking_metadatos.add(misdatos.getMetadatosListaSeleccionados(j));
                    lista_ranking_ficheros.add(misdatos.getFicheroListaSeleccionados(j));
                    break;
                }
            }
            if (lista_ranking_imagen.size() == (limite+1))
                break;
        }
    // Eliminamos la primera porque la mas parecida siempre es la misma imagen que se compara consigo misma
        lista_ranking_imagen.remove(0);
        lista_ranking_metadatos.remove(0);
        lista_ranking_ficheros.remove(0);
        cargarLabelsRanking(lista_ranking_imagen, lista_ranking_metadatos, lista_ranking_ficheros);
        lista_media.clear();
        lista_ranking_imagen.clear();
        lista_ranking_metadatos.clear();
        lista_ranking_ficheros.clear();
    }
```

Figura 50: Código del método algoritmoRestaHistogramas.

```
public void algoritmoRestaHistogramasConUmbral(int limite, int umbral){
    ArrayList<int[][]> histogramas = new ArrayList<>();
    ArrayList<BufferedImage> lista_ranking_imagen = new ArrayList<>();
    ArrayList<String> lista_ranking_metadatos = new ArrayList<>();
    ArrayList<File> lista_ranking_ficheros = new ArrayList<>();
    int resta_histogramas[] = new int[256];
    int media[] = new int[misdatos.getListaSeleccionadasRGB().size()];
    ArrayList<Integer> lista_media = new ArrayList<>();
    int aux;
    for(int i = 0; i < misdatos.getListaSeleccionadasRGB().size(); i++){
        histogramas.add(Utils.getHistograma(misdatos.getImagenListaSeleccionadasRGB(i)));
    }
    for(int i = 0; i < misdatos.getListaSeleccionadasRGB().size(); i++){
        for(int j = 0; j < misdatos.getListaSeleccionadasRGB().size(); j++){
            media[j] = 0;
            for(int y = (0+umbral); y < (256-umbral); y++){
                resta_histogramas[y] = Math.abs(histogramas.get(i)[4][y] - histogramas.get(j)[4][y]);
                media[j] += resta_histogramas[y]/3;
            }
            lista_media.add(media[j]);
        }
        while(!lista_media.isEmpty()){
            aux = 2147483647;
            for(int j = 0; j < lista_media.size(); j++){
                if (lista_media.get(j) < aux){
                    aux = lista_media.get(j);
                }
            }
            for(int j = 0; j < lista_media.size(); j++){
                if (lista_media.get(j) == aux){
                    lista_media.remove(j);
                    break;
                }
            }
            for(int j = 0; j < misdatos.getListaSeleccionadasRGB().size(); j++){
                if (media[j] == aux){
                    lista_ranking_imagen.add(misdatos.getImagenListaSeleccionadasRGB(j));
                    lista_ranking_metadatos.add(misdatos.getMetadatosListaSeleccionados(j));
                    lista_ranking_ficheros.add(misdatos.getFicheroListaSeleccionados(j));
                    break;
                }
            }
            if (lista_ranking_imagen.size() == (limite+1))
                break;
        }
    // Eliminamos la primera porque la mas parecida siempre es la misma imagen que se compara consigo misma
        lista_ranking_imagen.remove(0);
        lista_ranking_metadatos.remove(0);
        lista_ranking_ficheros.remove(0);
        cargarLabelsRanking(lista_ranking_imagen, lista_ranking_metadatos, lista_ranking_ficheros);
        lista_media.clear();
        lista_ranking_imagen.clear();
        lista_ranking_metadatos.clear();
        lista_ranking_ficheros.clear();
    }
}
```

Figura 51: Código del método algoritmoRestaHistogramasUmbral.

4.8. Algoritmos utilizados para las bibliotecas de filtros

Las siguientes clases forman parte de lo que sería una biblioteca completa de un filtro para la aplicación que se está desarrollando.

La primera es la que incorpora el algoritmo para convertir una imagen a escala de grises,

para ello se calcula la media de las componentes de color RGB de todos los píxeles de la imagen y se desplazan los bits de color para cambiar a formato sRGB. [2]

```java
package com.rankingimagenes;

import java.awt.Color;
import java.awt.image.BufferedImage;

/**
 *
 * @author Diego Vázquez Blanco
 */
public class Filtro {

    public static String nombre_filtro = "Escala de Grises";

    /**
     * Calculamos la media de una variable Color
     * @param color Color del cual se quiere obtener la media
     * @return entero con el valor de la media
     */
    private static int calcularMedia(Color color){
        int mediaColor;
        mediaColor=(int)((color.getRed()+color.getGreen()+color.getBlue())/3);
        return mediaColor;
    }

    /**
     * Pasa los colores de una imagen (solo JPG o BMP) a sus respectivos de la escala de grises.
     * @param imagen Imagen que va a ser convertida (solo JPG o BMP).
     * @return Devuelve la imagen convertida en escala de grises.
     */
    public static BufferedImage aplicar(BufferedImage imagen){
        //Variables que almacenarán los píxeles
        int mediaPixel,colorSRGB;
        Color colorAux;

        //Recorremos la imagen píxel a píxel
        for( int i = 0; i < imagen.getWidth(); i++ ){
            for( int j = 0; j < imagen.getHeight(); j++ ){
                //Almacenamos el color del píxel
                colorAux=new Color(imagen.getRGB(i, j));
                //Calculamos la media de los tres canales (rojo, verde, azul)
                mediaPixel= calcularMedia(colorAux);
                //Cambiamos a formato sRGB para JPG y BMP
                colorSRGB=(mediaPixel << 16) | (mediaPixel << 8) | mediaPixel;
                //Asignamos el nuevo valor al BufferedImage
                imagen.setRGB(i, j,colorSRGB);
            }
        }
        // Retornamos la imagen en escala de grises
        return imagen;
    }
}
```

Figura 52: Código de la biblioteca del filtro Escala de Grises.

La segunda es la que incorpora el algoritmo para eliminar el color (lo pone en negro) de los píxeles de una imagen cuyo valor no supere el Umbral (32 en este caso). [2]

```java
package com.rankingimagenes;

import java.awt.image.BufferedImage;

/**
 *
 * @author Diego Vázquez Blanco
 */
public class Filtro {

    public static String nombre_filtro = "Umbral de 32";

    /**
     * Disminuye el valor del color de cada pixel de la imagen en 150.
     * @param imagen Imagen que va a la que va a ser aplicada el filtro.
     * @return Devuelve la imagen con el filtro aplicado.
     */
    public static BufferedImage aplicar(BufferedImage imagen){

        int color;
        int umbral = 32;

        //Recorremos la imagen pixel a pixel
        for( int i = 0; i < imagen.getWidth(); i++ ){
            for( int j = 0; j < imagen.getHeight(); j++ ){
                //Almacenamos el color del pixel
                color = imagen.getRGB(i, j);
                //Eliminamos el color que se encuentra en el umbral
                if(color <= umbral)
                    imagen.setRGB(i, j, 0);
                else
                    imagen.setRGB(i, j, color);
            }
        }
        // Retornamos la imagen
        return imagen;
    }
}
```

Figura 53: Código de la biblioteca del filtro Disminución de Color .

4.9. La clase Utils

Es una clase incorporada al proyecto a modo de clase auxiliar en la que recopilar los métodos de funcionalidad más generalista y que podrían usarse en otro proyecto diferente sin ningún cambio. Esta clase no dispone de constructor ya que no es necesario al no disponer de atributos y los métodos ser estáticos usándose tan sólo por su utilidad.

```java
public static 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;
}
```

Figura 54: Código del método getExtension.

```
public static String getMetadata(File file){
    String cadena = "";
    String extension = Utils.getExtension(file);
    Metadata metadata;

    try {
        switch (extension) {
            case "jpeg":
                metadata = JpegMetadataReader.readMetadata(file);
                break;
            case "png":
                metadata = PngMetadataReader.readMetadata(file);
                break;
            case "gif":
                metadata = GifMetadataReader.readMetadata(file);
                break;
            case "bmp":
                metadata = BmpMetadataReader.readMetadata(file);
                break;
            default:
                metadata = ImageMetadataReader.readMetadata(file);
                break;
        }

        // Itera sobre los datos
        // Un objeto Metadata contiene multiples objetos Directory
        for (Directory directory : metadata.getDirectories()) {

            cadena = "<p>"; // Encerramos entre <p> y </p> para que
                            // solo salga Nombre, Tamano, FechMod
            // Cada Directory guarda valores en los objetos Tag
            for (Tag tag : directory.getTags()) {

                if (!extension.equals("bmp")){
                    try{cadena += tag.getTagName().substring(5) + ": ";
                    }catch(IndexOutOfBoundsException e){}
                }
                else{
                    cadena += tag.getTagName() + ": ";
                }
                cadena += tag.getDescription() + "<br>";
            }

            cadena += "</p>";

            // Un Directory también puede contener mensajes de error
            if (directory.hasErrors()) {
                for (String error : directory.getErrors()) {
                    System.err.println("ERROR en getMetadata("+ file +"): " + error);
                }
            }
        }
    } catch (ImageProcessingException | IOException e) {}
    return cadena;
}
```

Figura 55: Código del método getMetadata.

```
public static int[][] getHistograma(BufferedImage imagen){
    Color colorAuxiliar;
    /* Creamos la variable que contendra el histograma
    El primer campo [0], almacenara el histograma Rojo
    [1]=verde [2]=azul [3]=alfa [4]=escala grises */
    int histogramaReturn[][]=new int[5][256];
    // Recorremos la imagen
    for( int i = 0; i < imagen.getWidth(); i++ ){
        for( int j = 0; j < imagen.getHeight(); j++ ){
            // Obtenemos color del píxel actual
            colorAuxiliar=new Color(imagen.getRGB(i, j));
            // Sumamos una unidad en la fila roja [0],
            // en la columna del color rojo obtenido
            histogramaReturn[0][colorAuxiliar.getRed()]+=1;
            histogramaReturn[1][colorAuxiliar.getGreen()]+=1;
            histogramaReturn[2][colorAuxiliar.getBlue()]+=1;
            histogramaReturn[3][colorAuxiliar.getAlpha()]+=1;
            histogramaReturn[4][calcularMedia(colorAuxiliar)]+=1;
        }
    }
    return histogramaReturn;
}
```

Figura 56: Código del método getHistograma.

4.10. Funcionalidades destacables de la interfaz

Una de las más destacables es la recolocación dinámica de las imágenes en el panel principal durante cada redimensionado de la ventana. Para ello recorre todos los Label que guardamos anteriormente en la capa de datos, va colocando cada Label a la derecha del anterior en fila y cuando llega al final del panel, cuando parte de la imagen quedaría fuera del cuadro, se pasa a la fila siguiente, se vuelve a empezar en la posición de la izquierda y se vuelve al anterior Label que no se pudo colocar por quedar fuera de la zona.

```
// Coloca cada imagen donde debe (Contenedor cada imagen: 200x200, Margen: 10)
for (int x = 0; x < misdatos.getListaLabelsPrincipal().size(); x++){

    if (mipresentacion.getScrollPanePrincipal().getWidth()-20 > ((cont+1)*210 + 10)){
        misdatos.getLabelListaPrincipal(x).setLocation(10 + 210*cont, 10 + 210*fila);
        mipresentacion.getPanelPrincipal().add(misdatos.getLabelListaPrincipal(x));
        mipresentacion.getPanelPrincipal().repaint();
        cont++;
    }
    else{
        fila++;
        cont=0;
        x--; // vuelve a la imagen que no pudo poner porque se salía del panel
    }
}
```

Figura 57: Código del método redimensionadoVentana.

La forma de poder seleccionar de forma intuitiva las imágenes del panel mediante un clic de ratón es también un hecho destacable. Para que se pueda producir, hay un escuchador de eventos (Listener) que se activa tras cada clic de ratón. El Listener se ocupa de marcar el label como seleccionado o desmarcarlo según sea el caso.

```
mouseListener = new MouseListener() {
    @Override
    public void mouseClicked(MouseEvent e) {
        JLabel label = (JLabel) e.getComponent();
        if (label.isFocusable()){
            label.setFocusable(false);
            label.setBorder(BorderFactory.createLineBorder(Color.gray));
        }
        else {
            label.setFocusable(true);
            label.setBorder(BorderFactory.createLineBorder(Color.blue, 5));
        }
    }
```

Figura 58: Código del método mouseListenerPrincipal.

Para el escuchador de eventos de ratón en la vista del ranking es un poco diferente. Además de que en esta vista se marcan las seleccionadas en rojo, se deben marcar todas las imágenes del panel que son la misma foto, porque pueden aparecer más de una vez ya que se pueden parecer a más de una imagen. Para ello se dispone a mayores de dos métodos, uno para comprobar la selección, y otro para la deselección.

```
mouseListener = new MouseListener() {
    @Override
    public void mouseClicked(MouseEvent e) {
        JLabel label = (JLabel) e.getComponent();
        if (label.isFocusable()){
            comprobarDeseleccionRanking(label);
        }
        else {
            comprobarSeleccionRanking(label);
        }
    }
```

Figura 59: Código del método mouseListenerRanking.

```
public void comprobarSeleccionRanking(JLabel label){
    for (int x = 0; x < misdatos.getArrayListasLabelsRanking().size(); x++){
        for (int y = 0; y < misdatos.getListaArrayListasLabelsRanking(x).size(); y++){
            if (misdatos.getListaArrayListasLabelsRanking(x).get(y).getToolTipText().equals(label.getToolTipText())
                    && !misdatos.getListaArrayListasFicherosRanking(x).get(y).getName().equals("Imagen-Eliminada.png")){
                misdatos.getListaArrayListasLabelsRanking(x).get(y).setFocusable(true);
                misdatos.getListaArrayListasLabelsRanking(x).get(y).setBorder(BorderFactory.createLineBorder(Color.red, 5));
            }
        }
    }
}
public void comprobarDeseleccionRanking(JLabel label){
    for (int x = 0; x < misdatos.getArrayListasLabelsRanking().size(); x++){
        for (int y = 0; y < misdatos.getListaArrayListasLabelsRanking(x).size(); y++){
            if (misdatos.getListaArrayListasLabelsRanking(x).get(y).getToolTipText().equals(label.getToolTipText())){
                misdatos.getListaArrayListasLabelsRanking(x).get(y).setFocusable(false);
                misdatos.getListaArrayListasLabelsRanking(x).get(y).setBorder(BorderFactory.createLineBorder(Color.gray));
            }
        }
    }
}
```

Figura 60: Código del método comprobarSeleccion.

Hay operaciones que si se han seleccionado una gran cantidad de imágenes o si se ha seleccionado un directorio con muchas imágenes a cargar, pueden tardar un tiempo que puede ser perceptible para el usuario y que parezca que la aplicación se haya bloqueado y no responda. Para ello existe en la aplicación la siguiente imagen mostrada, la cual se muestra en dichas situaciones con el fin de notificar al usuario.

Figura 61: Imagen que notifica cuando hay algo procesando.

5. Pruebas

Las pruebas realizadas por el autor se han realizado con un equipo dotado de las siguientes características:

- Java 8 Update 131 (64-bit)
- Windows 10 Pro 64 bits
- Xubuntu 16.04 Linux kernel 4.4 64 bits
- Portátil PackardBell i7-2630QM, 6GB RAM 1333MHz, SSD Samsung 850 evo 240GB.

Las pruebas han sido sencillas y se pueden dividir en dos partes, las pruebas personales realizadas por el autor, y las pruebas realizadas por el cliente que este caso es el tutor del TFG. Tras terminar la codificación de cada funcionalidad se ha realizado una serie de pruebas de forma personal por el autor con el objetivo de intentar detectar posibles errores y evitar su acumulación. Los tipos de pruebas por parte del autor son tanto de caja blanca como de caja negra [7].

En todas las fases de pruebas, el procedimiento de pruebas ha sido el que sigue. En primer lugar, se han realizado pruebas de caja negra, escogiendo métodos que podían dar problemas debido a su complejidad. Se han probado paso a paso intentando dar solución a los problemas encontrados, con el objetivo de evitar errores futuros mayores.

En segundo lugar se realizan pruebas de caja blanca, con conocimiento del código, y examinando las partes o módulos aparentemente más problemáticos, que normalmente coincidían con los módulos marcados como erróneos en las pruebas de caja negra.

Por último, y una vez solucionados los errores de cada fase, se llevan a cabo pruebas con el cliente, dando como resultado algún cambio en la interfaz, por no ser del todo intuitiva o informativa, y algunas ampliaciones extra necesarias para completar la funcionalidad global mínima.

A continuación se muestra una tabla que recoge todas las pruebas de las funcionalidades del programa:

Entrada	Salida esperada
Seleccionar carpeta válida	Se selecciona el directorio y se cargan las imágenes admitidas
Seleccionar carpeta vacía	Notificación de directorio sin imágenes admitidas
Seleccionar archivo en vez de carpeta	Notificación de selección errónea
Intento de comparación sin imágenes seleccionadas	Notificación de que no se han seleccionado imágenes
Pulsar en icono de información	Notificación con instrucciones para algunas operaciones
Seleccionar o Deseleccionar una imagen del panel principal	La imagen deseada se selecciona en borde azul o se deselecciona
Seleccionar o Deseleccionar todas las imágenes del panel principal	Todas las imágenes se seleccionan en borde azul o se deseleccionan
Cambiar el algoritmo de comparación	El item seleccionado del ComboBox de algoritmos de comparación pasa a ser el seleccionado en el desplegable
Cambiar el algoritmo de filtro	Se reinicia la aplicación. La biblioteca con el filtro seleccionado se carga en el programa y aparece el nombre entre paréntesis en el botón de Aplicar filtro
Cambiar de tamaño la ventana	Se desplazan los grupos de botones correspondientes al lado del que se aumenta permaneciendo junto al borde. En el panel las imágenes se recolocan de modo que nunca aparezca un ScrollBar Horizontal
Aplicar filtro a imágenes seleccionadas	Las imágenes mostradas se convierten según el filtro aplicado
Cuando pida el número máximo de imágenes, introducir letras o dejarlo en blanco	Notificación de que se ha introducido mal el dato y algún consejo de como introducirlo.
Cuando pida el número máximo de imágenes, cancelar la operación	Se notifica la cancelación y se regresa a la pantalla principal reseteada.
Ver los metadatos de alguna imagen	Tras un segundo aparece un cuadro con los metadatos en la posición del ratón
Seleccionar o Deseleccionar una imagen del panel de ranking	La imagen deseada se selecciona en borde rojo o se deselecciona
Intento de eliminación sin imágenes seleccionadas	Notificación de que no se han seleccionado imágenes

Eliminar imágenes	Se mueven los archivos de las imágenes seleccionadas a una carpeta local Ïmagenes Eliminadasz se cambian las imagenes eliminadas del panel de ranking por un mensaje informativo
No confirmar la eliminación de imágenes	Se queda en el mismo estado que antes de pulsar el botón de Eliminar
Volver a la vista principal desde la vista del ranking	Se regresa a la pantalla principal reseteada

Tabla 2: Tabla de pruebas de las funcionalidades del programa

6. Conclusiones

Esta sección se dedica a conclusiones finales y personales del proyecto, extraídas a lo largo de todo el proceso de desarrollo del mismo.

Se han cumplido todos los objetivos y requisitos mencionados en las respectivas secciones. El principal de ellos era crear una aplicación de escritorio para Windows y Linux capaz de ofrecer los servicios de comparación de imágenes a través de la construcción de un ranking de parecido.

La aplicación desarrollada ofrece también la posibilidad de aplicar un filtro a ciertas imágenes, el filtro es personalizable pudiendo compilar una biblioteca con un algoritmo propio y cargarlo en la aplicación, se puede establecer el número de imágenes que formarán las listas de fotos más parecidas, el algoritmo de cálculo de parecido se puede elegir al gusto de entre varios implementados y las imágenes no deseadas se pueden eliminar enviándolas a una "papelera"local.

Atendiendo al desarrollo técnico del proyecto, al comienzo fue algo difícil comprender todas las ideas y requisitos que había sobre la mesa. Se hizo un estudio previo sobre la naturaleza de los datos con los que se iba a tratar para conocer la mejor manera de sobrellevarlo y conocer la mejor manera de implementarlo en una aplicación. Sobretodo se realizó una investigación sobre histogramas de imágenes, que es el medio por el cual se han obtenido los datos de los parámetros por los que se componen imágenes. [3] [1]

Una vez realizadas las fases de análisis y diseño, las fases posteriores fueron encauzadas rápidamente. Por lo que las primeras fases de investigación, análisis y diseño resultaron ser de suma importancia para tener las ideas más claras.

Además de haber aprendido a desarrollar un proyecto de principio a fin, pasando por todas sus fases, he adquirido nuevos conocimientos, o ampliado muchos de ellos sobre Imágenes, Java, Swing, Modelo Vista Controlador, UML, NetBeans, Bibliotecas, Scripting, y otros conocimientos necesarios que se han requerido durante todas las fases.

7. Trabajo Futuro

Dentro del trabajo futuro en este proyecto pueden destacarse varias vertientes.

La primera de ellas es la mejora de la interfaz gráfica, con ayuda de una persona especialista puede hacerse que la aplicación tenga un acabado visual más profesional, ya que se ha usado una gama de colores y un tema genéricos porque este proyecto estaba más enfocado en la funcionalidad que en el atractivo visual.

Otro de los aspectos a mejorar es la variedad de funcionalidades, en este proyecto solo se ha tenido como objetivo principal el establecer un ranking determinado tan sólo por el parecido general de las imágenes, pero se podría pensar en implementar otras opciones más específicas como la de listar todas las imágenes que contengan un determinado objeto, o que sean de una familia concreta como imágenes que son todas paisajes o todas platos de comida.

También puede resultar interesante el ampliar la compatibilidad del programa realizando una versión para Android o iOS y así que se pueda ejecutar en dispositivos móviles, de esa manera sería más cómodo para el caso de fotografías tomadas con el móvil o de imágenes recibidas mediante mensajería instantánea, así no se tendrían que molestar en conectar el *smartphone* al ordenador pudiendo hacerse directamente desde el móvil.

Algo que igualmente puede aumentar la funcionalidad en gran medida es el aumento de los tipos de archivos admitidos por el programa. Se podría considerar admitir el formato RAW que se está volviendo muy popular en el mundo de la fotografía últimamente.

Un último trabajo futuro a mencionar sería la inclusión de una funcionalidad que permita exportar los datos del ranking. Los tipos de archivo a los que se podría exportar son múltiples (texto, CSV, Excel, Imagen). Esto le añadiría la característica de persistencia a la aplicación.

Referencias

[1] "riunet.upv.es/bitstream/handle/10251/12711/el histograma una imagen digital.pdf," (último acceso 15/04/2017).

[2] "dis.um.es/%7Eginesgm/files/doc/pav/tema2.ppt," (último acceso 15/04/2017).

[3] "www.ite.educacion.es/formacion/materiales/86/cd/m12/histograma.html," (último acceso 15/04/2017).

[4] "http://di002.edv.uniovi.es/%7Ecueva/asignaturas/doctorado/2001/complejidad.pdf," (último acceso 5/07/2017).

[5] "www.java.com/es/download/help/sysreq.xml," (último acceso 5/07/2017).

[6] "www.oracle.com/technetwork/java/javase/certconfig-2095354.html," (último acceso 5/07/2017).

[7] "http://www.lab.dit.upm.es/%7Elprg/material/apuntes/pruebas/testing.htm," (último acceso 6/07/2017).

[8] "http://astah.net/editions/professional," (último acceso 8/07/2017).

[9] "https://drewnoakes.com/code/exif/," (último acceso 8/07/2017).

[10] "https://mockflow.com/," (último acceso 8/07/2017).

[11] "https://netbeans.org/kb/docs/java/gui-binding.html," (último acceso 8/07/2017).

[12] "http://www.adobe.com/products/xmp.html," (último acceso 8/07/2017).

[13] "http://dalila.sip.ucm.es/%7Emanuel/JSW1/Slides/Swing.pdf," (último acceso 8/07/2017).

Anexo A. Manual de Instalación

Objetivo

El objetivo del manual de instalación es ofrecer una guía para el usuario en la que se explique los pasos necesarios para realizar la correcta instalación de la aplicación y sus requerimientos.

Requisitos Mínimos

El requisito mínimo para que se pueda ejecutar la aplicación de manera correcta es que se encuentre instalada en el equipo la máquina virtual de Java para la versión en la que está compilada la aplicación, que en este caso es Java 8 Update 131 (64-bit).

Y para poder hacer funcionar la citada máquina virtual de Java, se requieren de los siguientes requisitos de sistema [5] [6]:

Windows

- Windows 10 (8u51 y superiores)
- RAM: 128 MB
- Espacio en disco: 124 MB para JRE; 2 MB para Java Update
- Procesador: Mínimo Pentium 2 a 266 MHz
- Exploradores: Internet Explorer 9 y superior, Firefox

Linux

- Oracle Linux 7.x (64 bits)2 (8u20 y superiores)
- Red Hat Enterprise Linux 7.x (64 bits)2 (8u20 y superiores)
- Suse Linux Enterprise Server 12.x (64 bits)2 (8u31 y superiores)
- Ubuntu Linux 15.10 (8u65 y superiores)
- Exploradores: Firefox

Instalación Máquina Virtual de Java

Como se ha comentado anteriormente, la versión de Java a instalar es la 8u131, y se explica cómo instalar aunque el proceso es similar a las otras versiones. Java 8 Update 131 (64-bit), es necesario para la ejecución de la aplicación, ya que ésta se encuentra implementada en Java y compilada para funcionar en dicha versión, o si resulta compatible, una superior.

Para su instalación en Windows, es tan simple como ir a la web de descargas de Oracle, descargar el Java 8 Update 131 (64-bit) y luego ejecutar el instalador descargado. A continuación se encontrará el enlace a dicha página de descargas:

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

A continuación se muestran los comandos para instalarlo en Ubuntu 16 a modo de ejemplo para realizar una instalación en un sistema Linux:

```
> sudo add-apt-repository ppa:webupd8team/java
> sudo apt-get update
> sudo apt-get install oracle-java8-installer
```

Y en caso de tener más versiones de Java, para dejar la 8 como opción por defecto:

```
> sudo apt-get install oracle-java8-set-default
```

Instalación de la aplicación

En el CD se encontrará una carpeta con el ejecutable de la aplicación. Hay que copiar la carpeta del ejecutable, entera con todas las carpetas y archivos que contiene, a un directorio del sistema local en el que se dispongan de permisos de lectura y escritura (puede ser cualquier directorio mientras se tengan permisos).

Para que la aplicación sea correctamente ejecutable tan sólo deben permanecer en el mismo directorio los archivos y carpetas del programa tal como se muestra en la siguiente imagen:

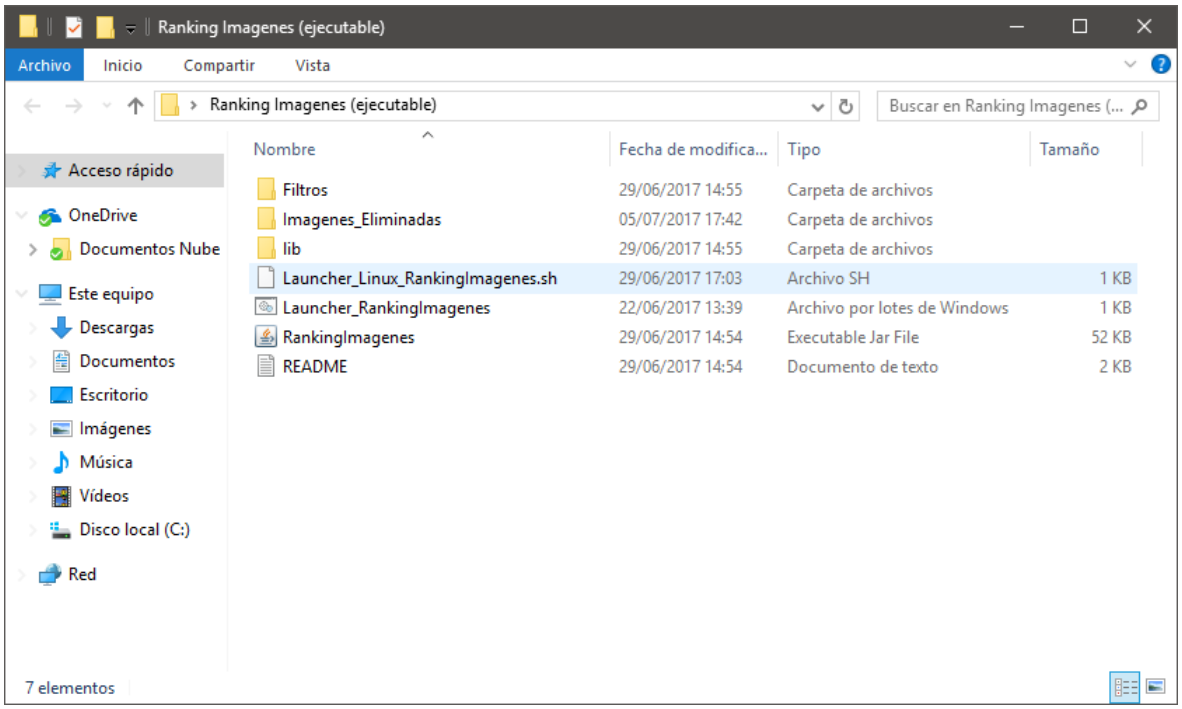

Figura 62: Disposición de los archivos y directorios de la aplicación.

Anexo B. Manual de Usuario

Objetivo

El objetivo de este manual es mostrar al usuario el uso de la aplicación de comparación de imágenes y todas las funcionalidades a las que podrá tener acceso. También se tratará de mostrar el significado de las posibles notificaciones que puedan aparecer durante la ejecución del programa.

Uso de la aplicación

Mediante la aplicación de creación de un ranking de imágenes, un usuario podrá comparar de diferentes formas las imágenes contenidas en un directorio. Además podrá aplicar diversos filtros personalizables a las imágenes que desee y eliminar las que se crea necesario una vez comparadas en el ranking.

Es necesario ejecutar la aplicación mediante el script (archivo .bat en Windows o .sh en Linux) para que la funcionalidad de cambio de filtro se pueda producir.

Funcionalidades

El usuario al iniciar la aplicación podrá ver un panel principal donde se mostrarán las imágenes pero que inicialmente estará vacío hasta que se seleccione una carpeta con imágenes. En la parte superior, encima del panel, encontrará diversos elementos de interacción con todas las opciones que puede realizar desde la vista principal. Estos pueden ser botones o *ComboBox*. Para cada imagen, puede ver dejando el ratón encima, un cuadro emergente con los metadatos (Nombre del archivo, Tamaño, Altura, Anchura, tipo, Localización, etc).

También podrá seleccionar, justo antes de que se empiece a calcular el ranking, el número máximo de imágenes a mostrar en el ranking por cada foto seleccionada. Después se muestran las imágenes seleccionadas anteriormente en un panel del lado izquierdo y el ranking en otro del lado derecho. En la vista del ranking hay en la parte superior elementos de interacción para eliminar imágenes y otra para volver a la pantalla principal y restablecer el programa.

Las funcionalidades que el usuario puede ver en la vista principal son: **Seleccionar Carpeta de Imágenes**, **Información**, **Seleccionar Todas**, **Deseleccionar Todas**, **Cambiar Algoritmo de Comparación** (*ComboBox*), **Cambiar Algoritmo de Aplicar Filtro** (*ComboBox* y botón), **Aplicar Filtro** y **Comparar**.

En la vista del ranking el usuario puede ver las opciones de: **Eliminar** y **Volver**

Además otra función que no se puede ver pero que es intuitiva y se encuentra en ambas vistas es: **Seleccionar Imagen** (haciendo clic en ella).

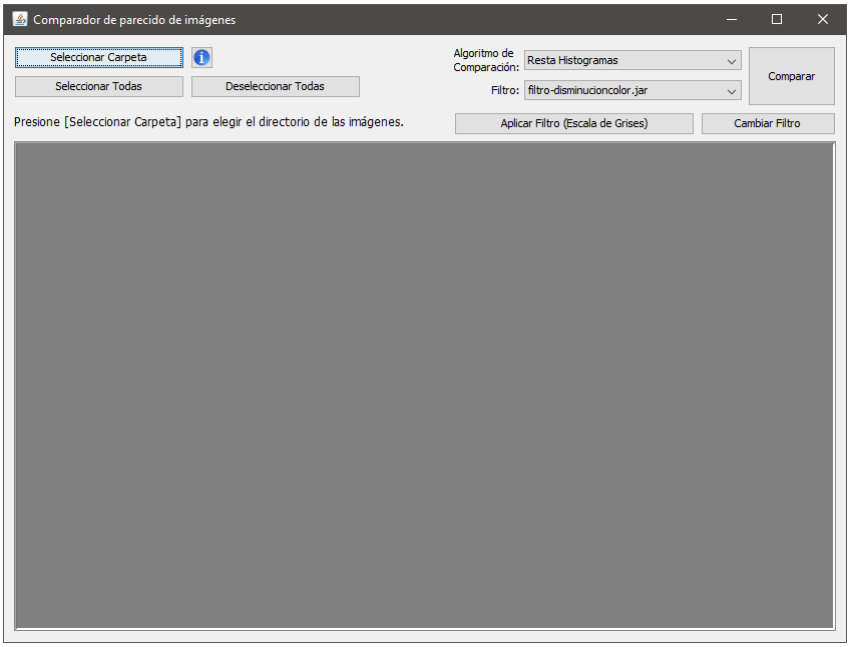

Figura 63: Vista Principal de la aplicación nada más iniciarla.

Seleccionar carpeta de imágenes

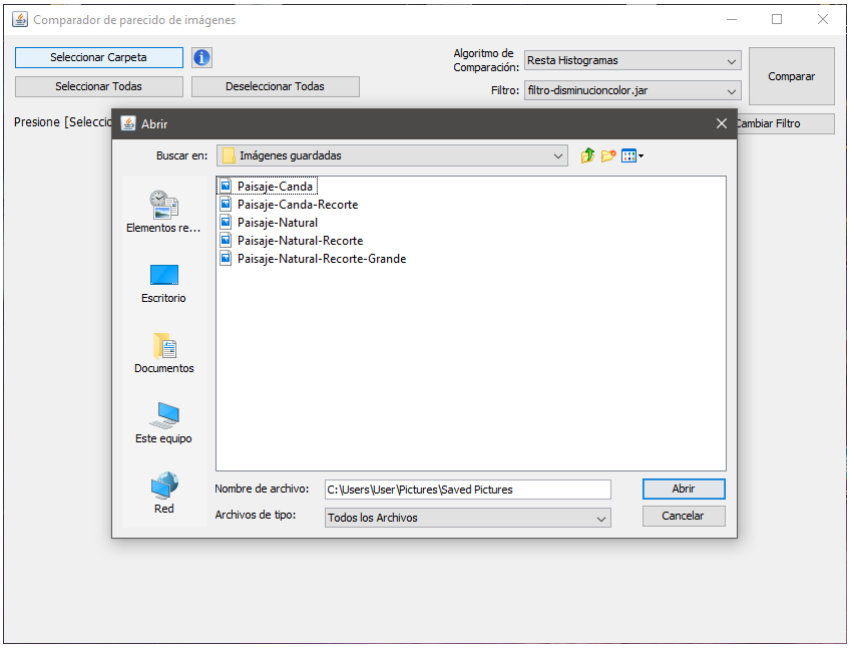

Figura 64: Cuadro de Diálogo tras presionar Seleccionar Carpeta.

Para poder realizar otras operaciones en la aplicación es necesario que haya imágenes cargadas, por lo tanto, si se selecciona un directorio que no contiene imágenes, se mostrará un *MessageBox* informando sobre ello.

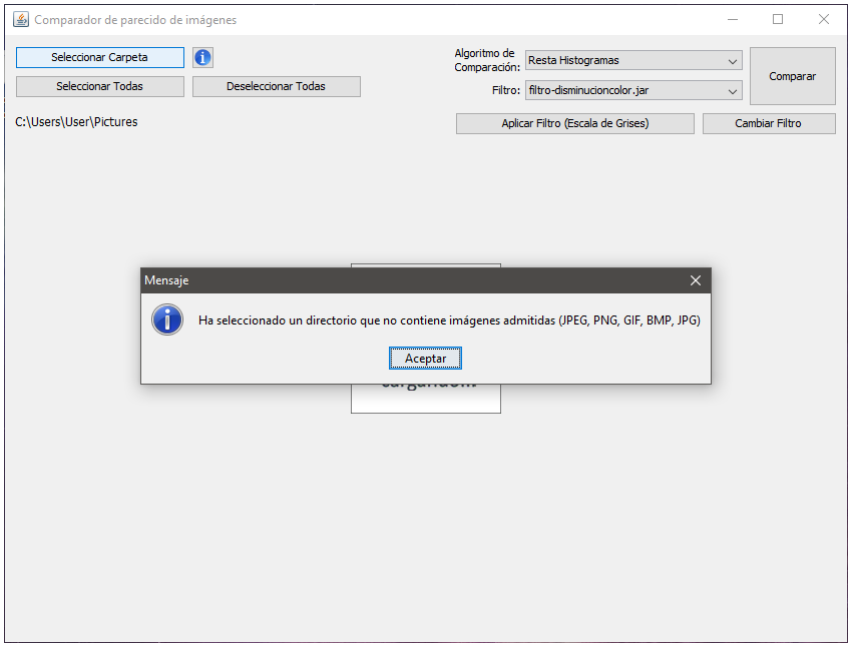

Figura 65: MessageBox informando de que la carpeta seleccionada no contiene imágenes.

Lo mismo ocurre si en vez de un directorio sin imágenes se selecciona un fichero o archivo. En ambos casos la aplicación vuelve al estado inicial sin tener ninguna imagen cargada y sin poder realizar otra operación diferente.

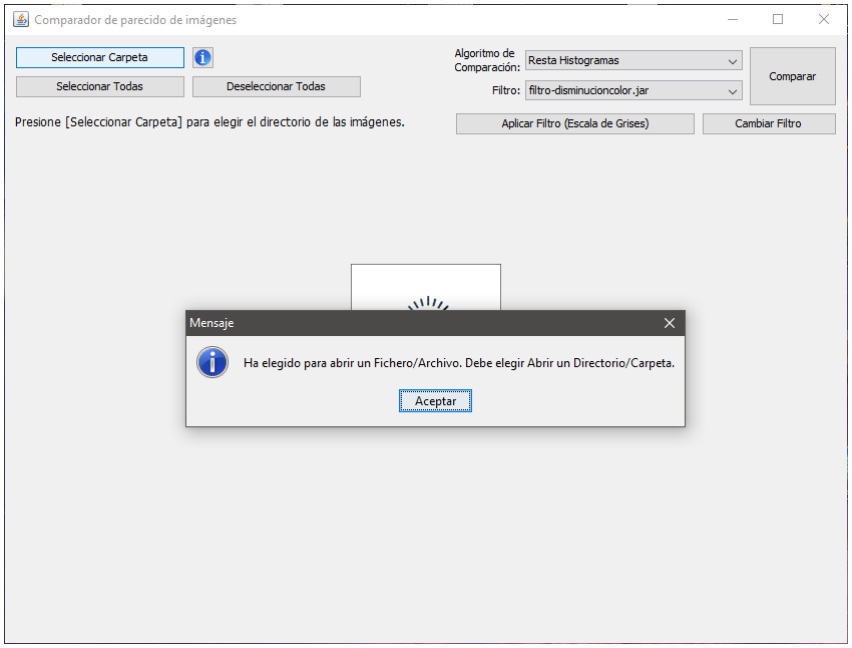

Figura 66: MessageBox informando de que se ha seleccionado un fichero/archivo.

Una vez seleccionado un directorio válido (que contenga imágenes admitidas), se notifica el número de imágenes válidas que contenía la carpeta y que se han cargado en la aplicación.

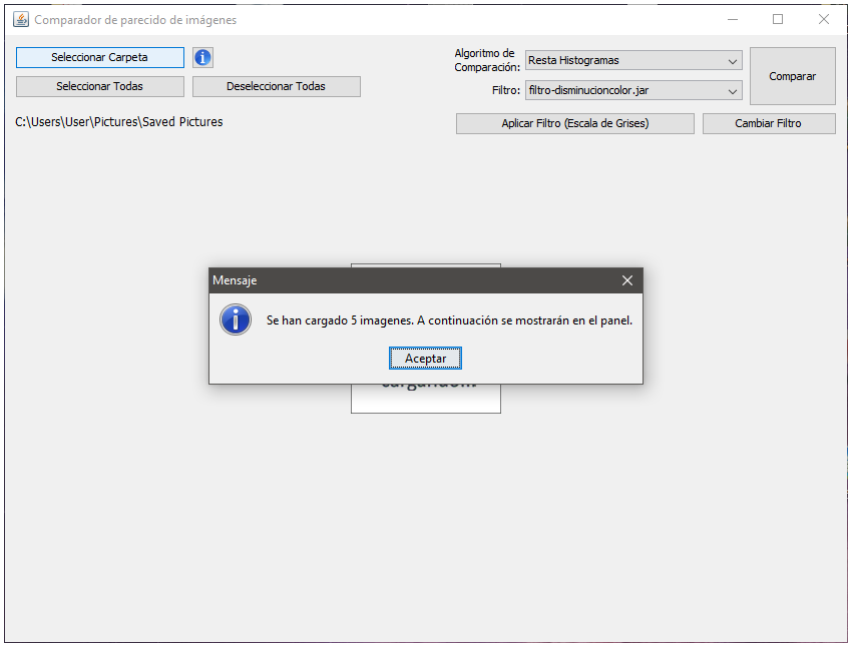

Figura 67: MessageBox de confirmación indicando el número de imágenes cargadas.

Información en la vista principal

Si se necesita información sobre cómo realizar algunas de las operaciones menos intuitivas, se puede pulsar el botón con el icono de información.

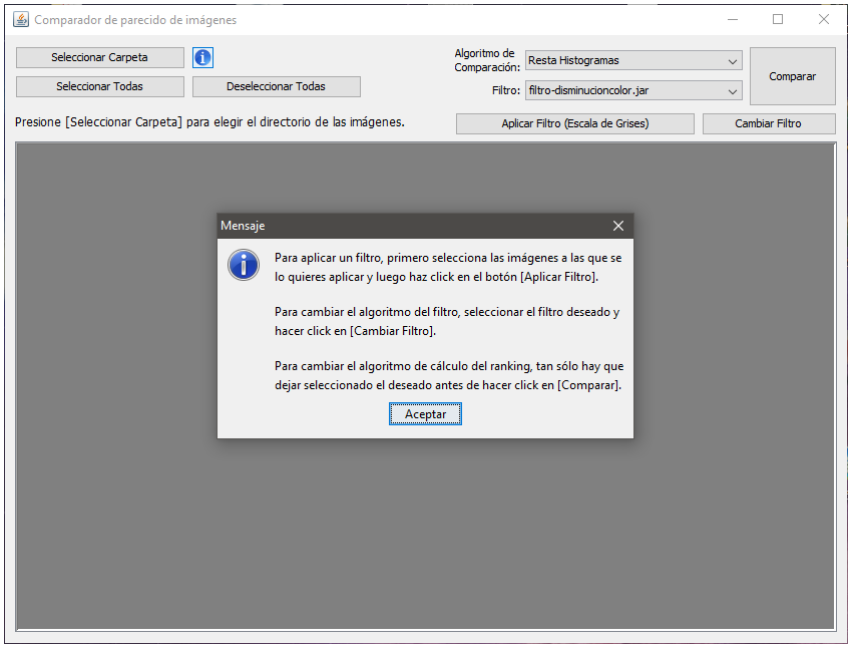

Figura 68: MessageBox con información sobre algunas operaciones.

Metadatos en la vista principal

Dejando el ratón sobre una imagen un corto periodo de tiempo, se mostrarán sus metadatos.

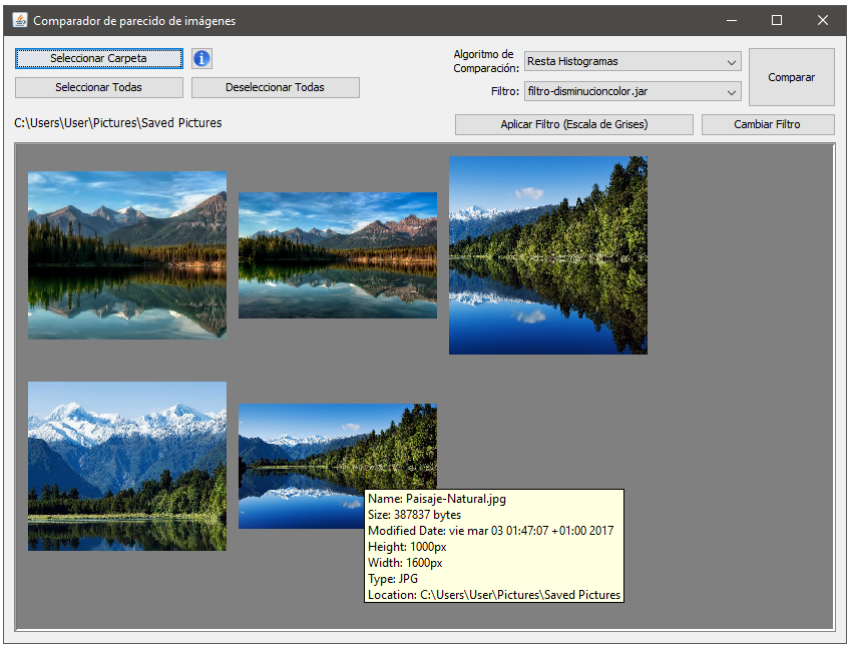

Figura 69: Cuadro emergente que contiene los metadatos de la imagen.

Seleccionar imágenes en la vista principal

Haciendo clic con el ratón en cada imagen se seleccionan de manera individual.

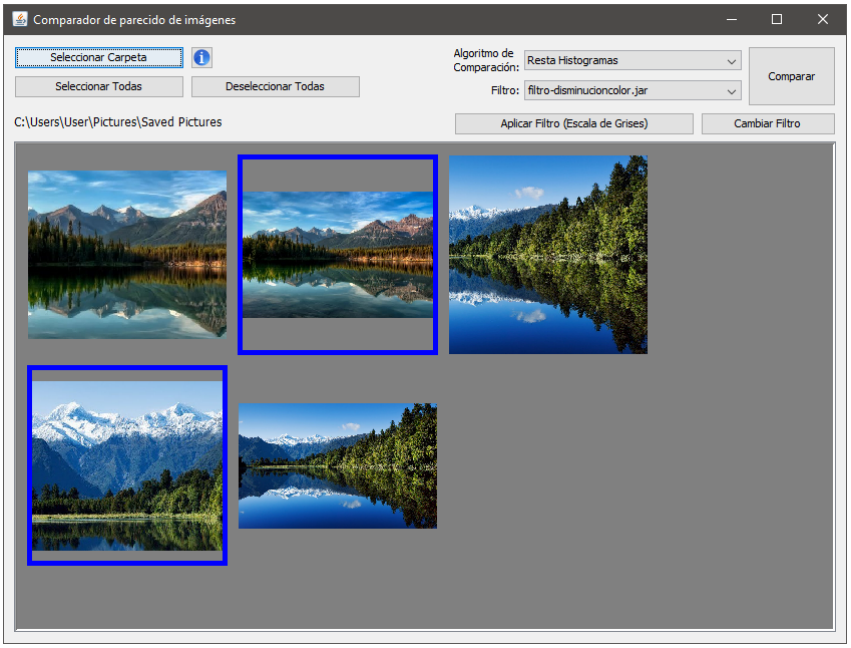

Figura 70: Imágenes seleccionadas individualmente mediante clic de ratón.

También existe la opción de seleccionar todas las imágenes a la vez o bien deseleccionarlas.

Cambiar algoritmo de aplicar filtro

Hay que seleccionar el filtro del ComboBox que contiene la lista de bibliotecas de filtros que pueden cargarse en la aplicación. Sólo puede estar cargado un filtro a la vez.

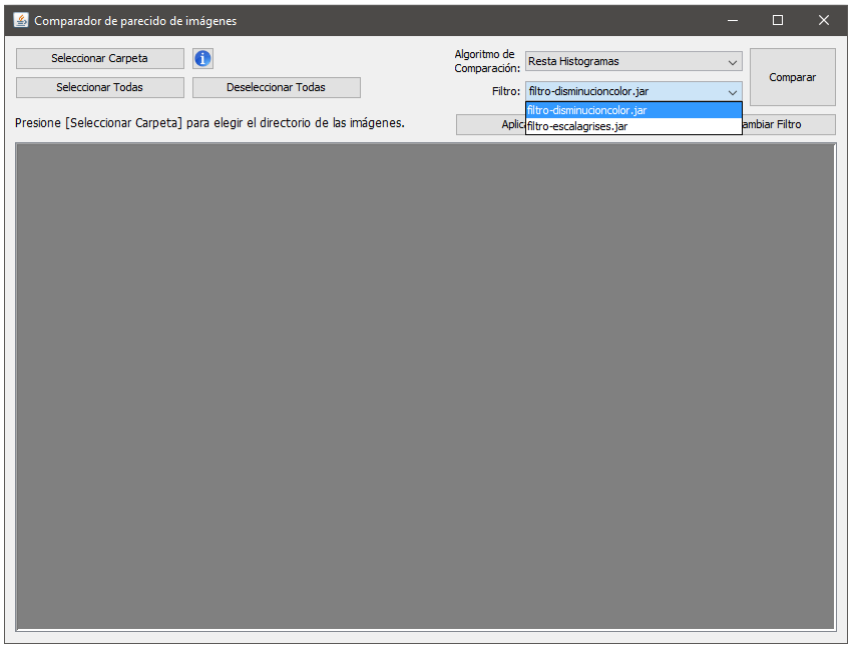

Figura 71: ComboBox desplegado que contiene la lista de bibliotecas de filtros.

Una vez seleccionado el filtro deseado, hay que presionar el botón de Cambiar Filtro y esperar a que se reinicie la aplicación con el nuevo algoritmo.

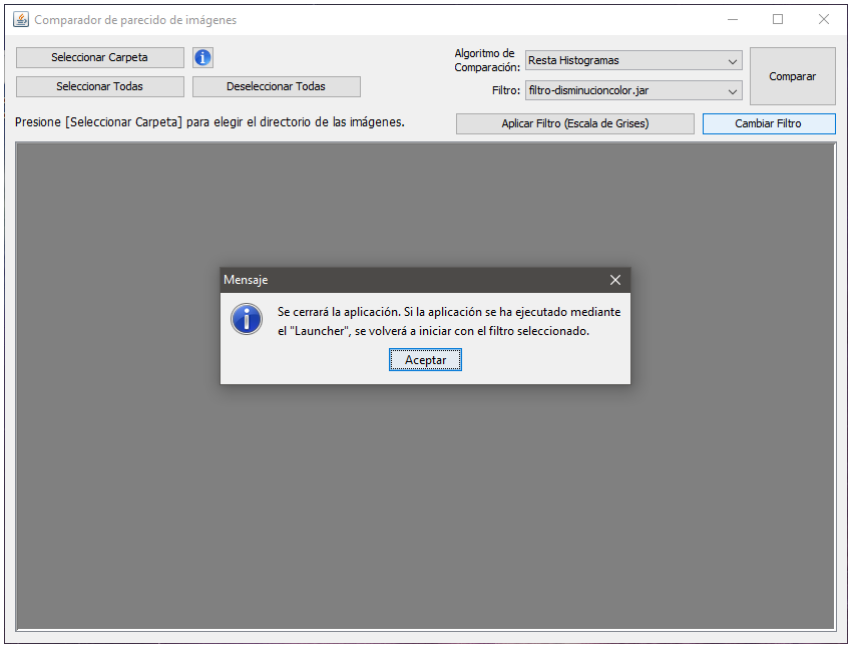

Figura 72: MessageBox informando del reinicio necesario para cambiar la biblioteca del filtro.

Aplicar filtro

Se seleccionan las imágenes a las que aplicar el filtro y se presiona Aplicar Filtro.

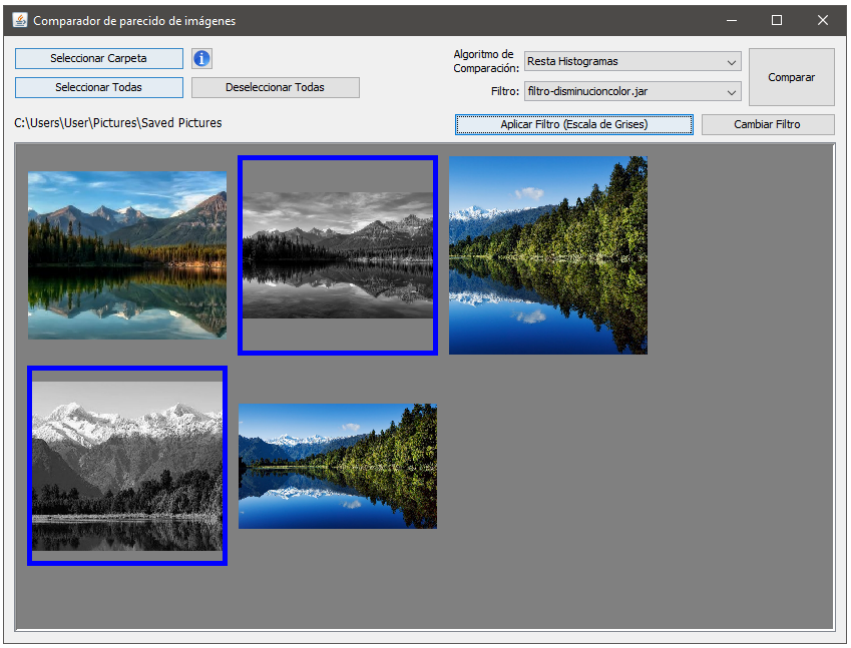

Figura 73: Filtro aplicado a las imágenes seleccionadas.

Cambiar algoritmo de comparación

Seleccionar el algoritmo deseado en el *ComboBox* designado por el correspondiente *Label*.

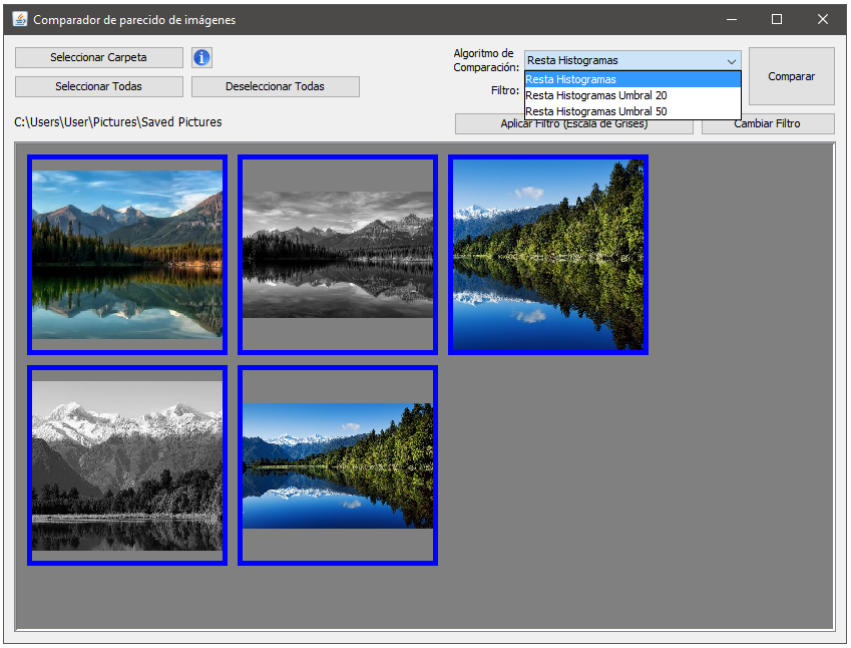

Figura 74: ComboBox desplegado que contiene la lista de algoritmos de Comparación.

Comparar

Para empezar a calcular el ranking, se pulsa el botón de Comparar. Lo primero antes de comenzar a realizar los cálculos, pedirá que se introduzca el número de imágenes parecidas a su referente a mostrar por cada imagen seleccionada (referente).

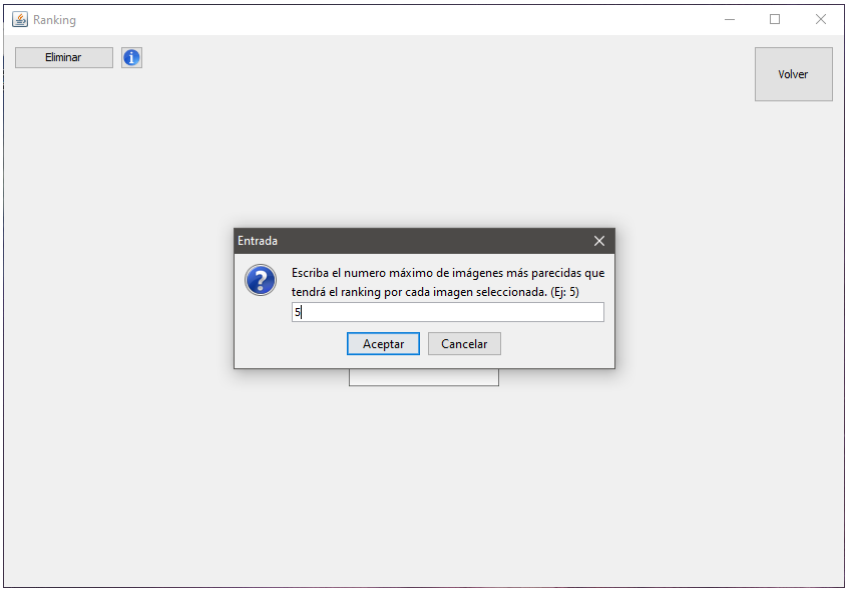

Figura 75: InputBox que pide introducir el número máximo de imágenes en el ranking.

Si en vez de introducir un número, se pulsa en cancelar, se abortará la operación de cálculo del ranking y se regresará a la vista principal sin carpeta seleccionada ni imágenes cargadas.

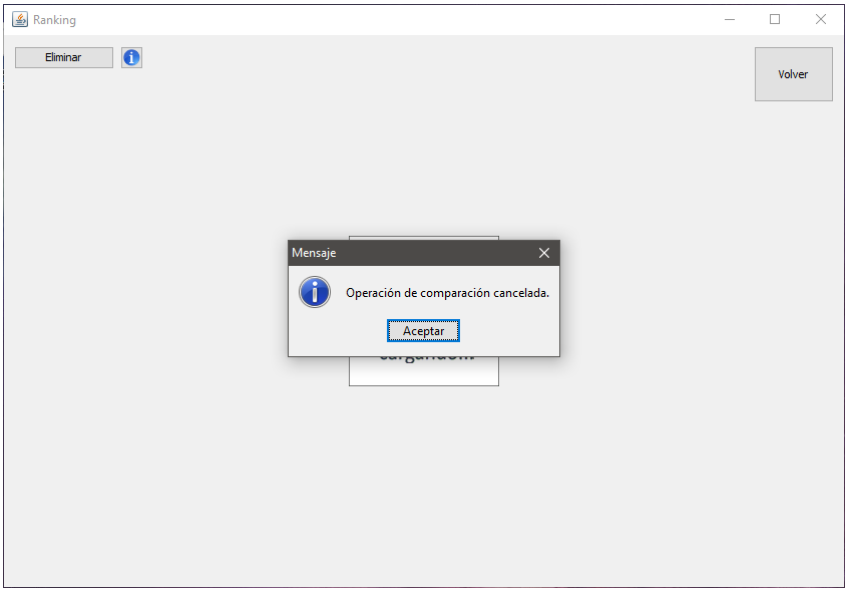

Figura 76: MessageBox informando de que la operación de Comparar ha sido cancelada.

Si la introducción del número no es correcta, por ejemplo si se escribe con letras o se deja en blanco, saltará un *MessageBox* que nos pedirá volver a introducirlo y alguna indicación.

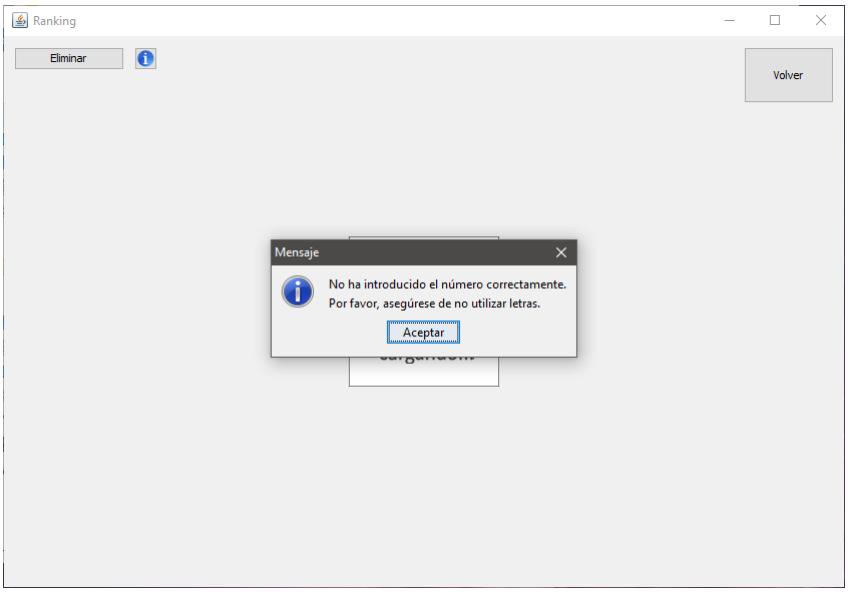

Figura 77: MessageBox de error por no introducir bien el número.

En cambio, si todo va bien, se confirmará que se va a empezar con el cálculo del ranking según qué algoritmo de comparación estuviese seleccionado y nos avisa de que puede llevar un rato.

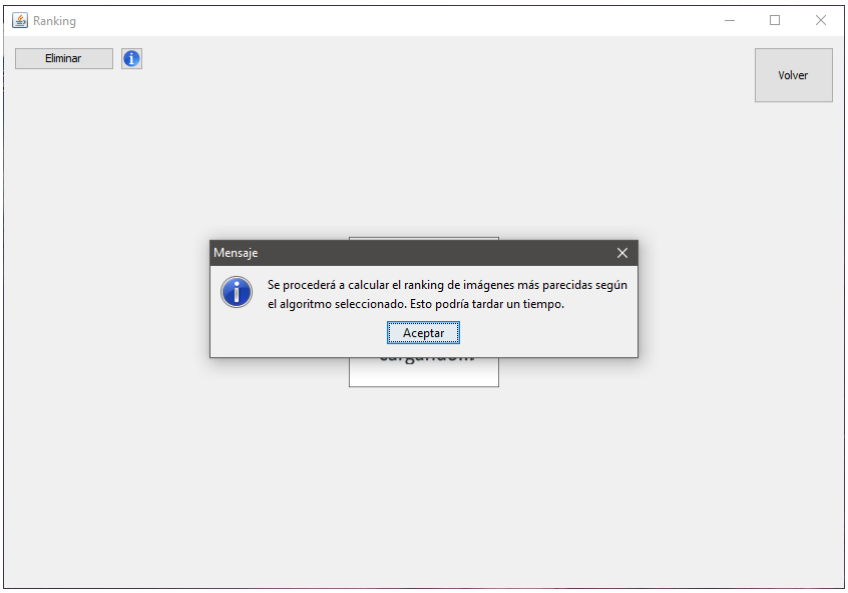

Figura 78: MessageBox de confirmación que dice que se ha comenzado a realizar los cálculos.

Información en la vista del ranking

Si se necesita información sobre cómo realizar algunas de las operaciones menos intuitivas, se puede pulsar el botón con el icono de información.

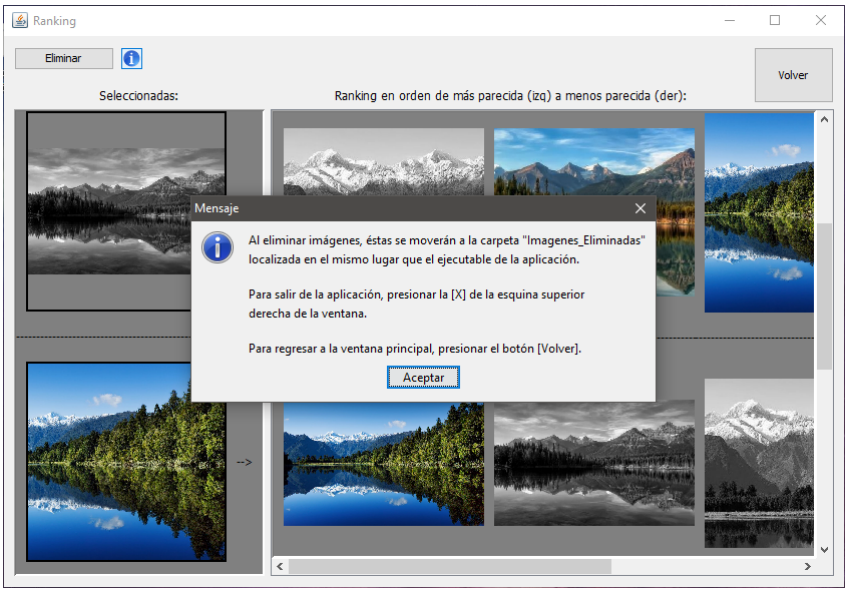

Figura 79: MessageBox con información sobre algunas operaciones.

Metadatos en la vista del ranking

Dejando el ratón sobre una imagen un corto periodo de tiempo, se mostrarán sus metadatos. Los filtros aplicados a la imagen no afectan a los metadatos de ésta.

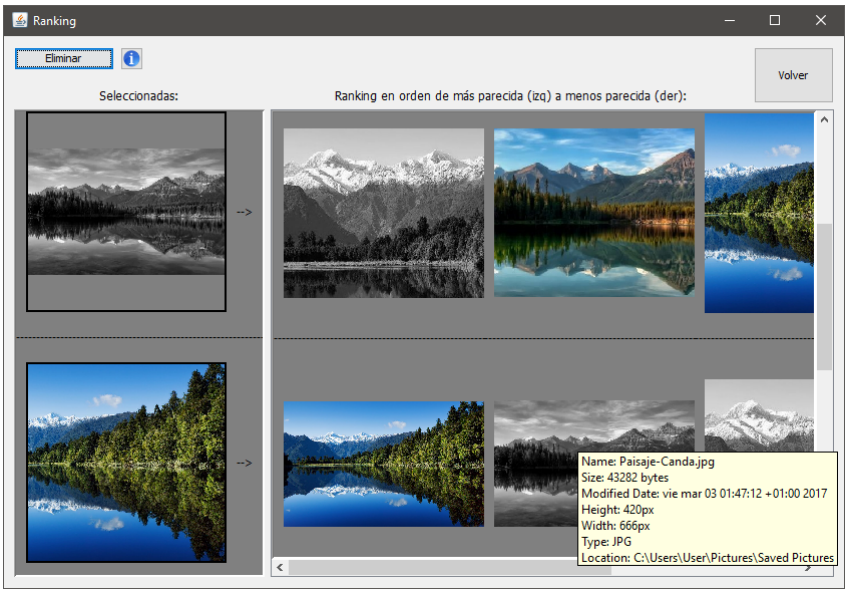

Figura 80: Cuadro emergente que contiene los metadatos de la imagen.

Seleccionar imágenes en la vista del ranking

Haciendo clic con el ratón en cada imagen se seleccionan de manera individual. En este caso se marca con el borde en rojo para resaltar que la operación a realizar en ellas es la de eliminar y que hay que llevarla a cabo con precaución para evitar pérdidas indeseadas.

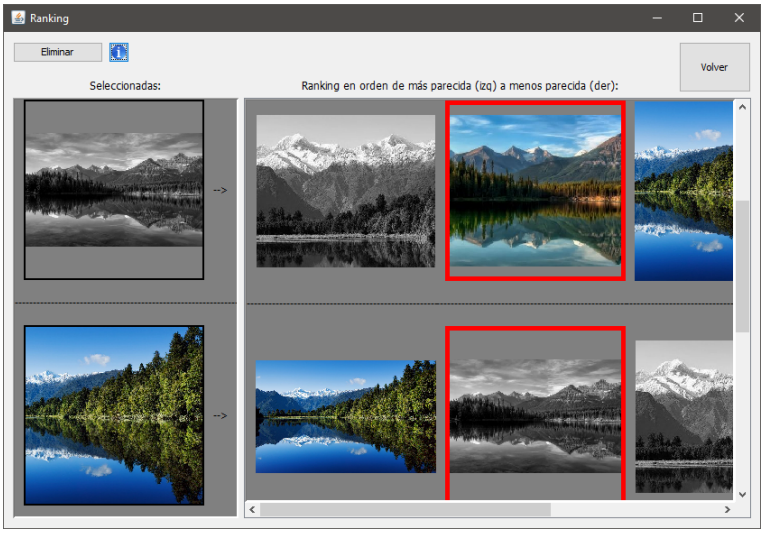

Figura 81: Imágenes seleccionadas individualmente mediante clic de ratón.

Eliminar imágenes

Se pulsa el botón de Eliminar y tras aceptar la confirmación se procederá a eliminar las imágenes previamente seleccionadas moviéndolas a la carpeta local Imágenes_Eliminadas.

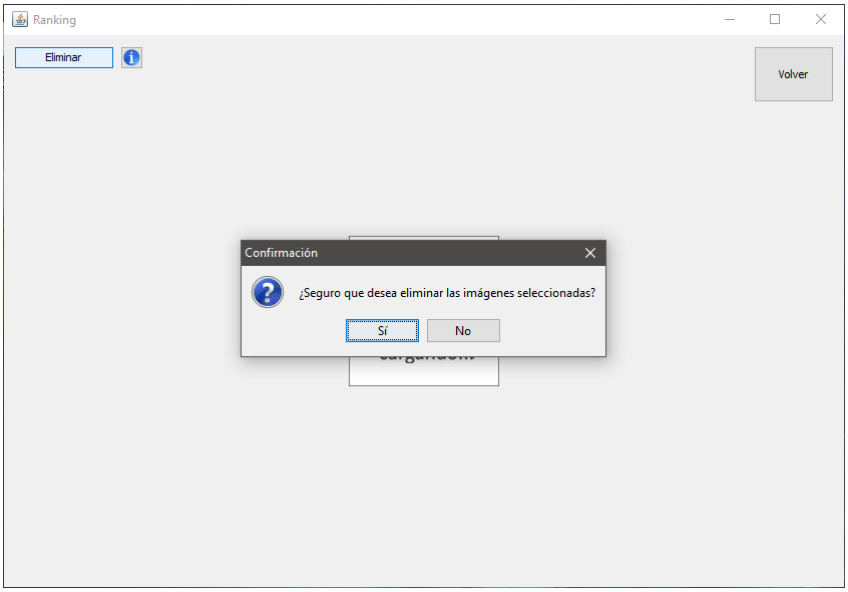

Figura 82: Diálogo de confirmación antes de eliminar una imagen.

En el panel, las imágenes eliminadas se sustituyen por mensajes informativos en el mismo puesto del ránking en el que se encontraba la original.

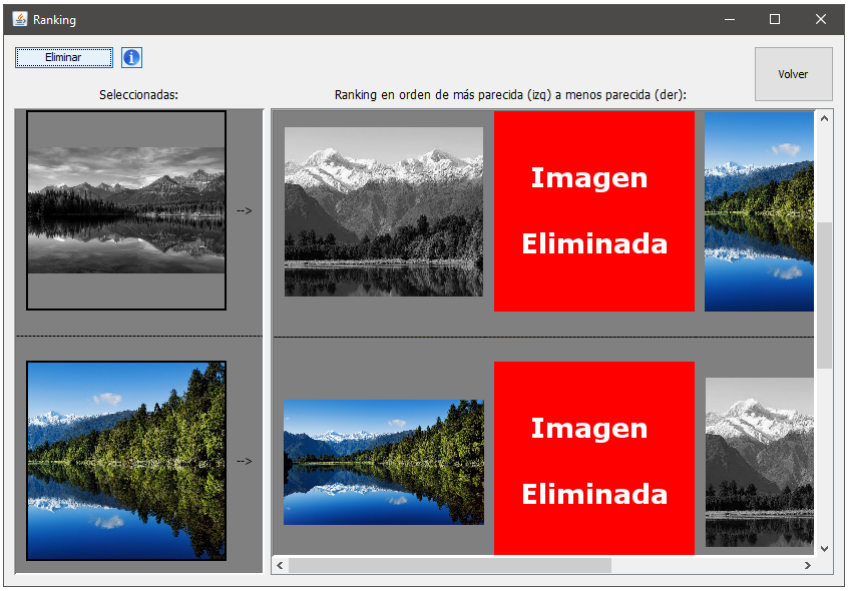

Figura 83: Panel del ránking después de eliminar imágenes.

Volver a la vista principal

Al pulsar en Volver, se restablece la aplicación en la vista principal sin carpeta seleccionada para comenzar de nuevo.

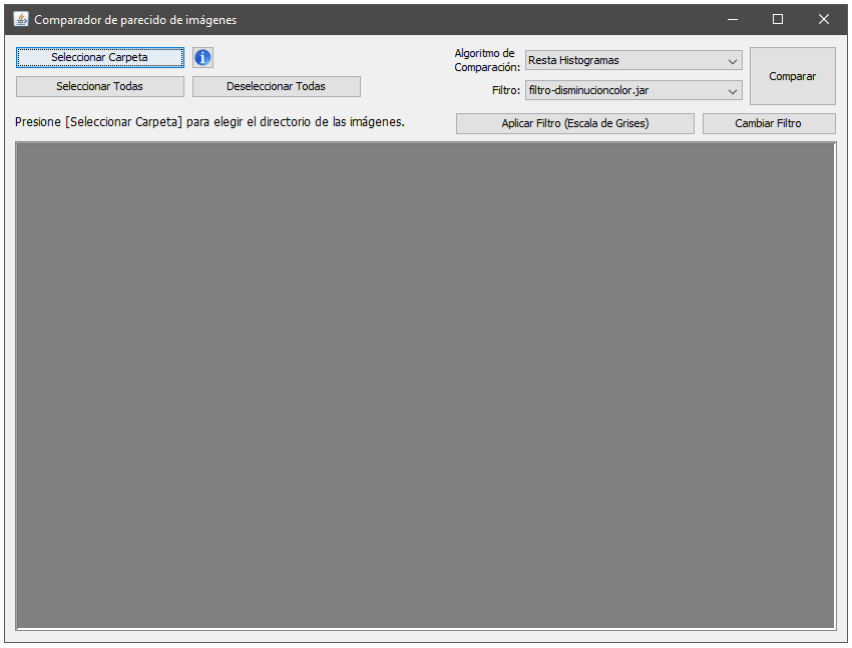

Figura 84: Vista principal sin directorio seleccionado.