ULPGC
Universidad de Las Palmas de Gran Canaria
Escuela de Ingeniería Informática

Aplicación para la gestión eficiente de datos satelitales

Universidad de las Palmas de Gran Canaria

Escuela de Ingeniería Informática

Estudiante

Ismael Álvarez Candelario

Tutores:

Agustín Trujillo Pino

Luis Gómez Déniz

Fecha

19 de noviembre de 2020

Las Palmas de Gran Canaria

Agradecimientos

Primero, agradecer a mis tutores, tanto a Luis Gómez como a Agustín Trujillo, cuya paciencia conmigo ha sido extremadamente elevada. Gracias por mantener la calma ante los numerosos problemas que acontecieron durante el desarrollo de la aplicación. Sin ellos, este proyecto no habría visto a la luz del día.

A mi familia y a amigos, que durante los últimos meses no han dejado de animarme desde la distancia. ¡Os he sentido muy cerca siempre!

Y, por último, pero no menos importante, a mi compadre Alexis, el cual me ha acompañado durante cada día de este proyecto, por escucharme, animarme e insistirme siempre en seguir hacia delante, independientemente de lo que pasara.

A todos, ¡muchas gracias!

Índice

1 Resumen

SatInf Manager es una aplicación para almacenar y gestionar información sobre imágenes satelitales, siendo capaz de buscar, organizar, descargar y realizar un procesado previo a dichas imágenes. Se ha utilizado la plataforma Copernicus Open Access Hub (repositorio online de imágenes satelitales) de la Agencia Espacial Europea para la obtención de las imágenes. El principal motivo para desarrollar esta aplicación es la dificultad para encontrar y gestionar imágenes satelitales de un modo rápido y sencillo. Existen aplicaciones que permiten realizar estas tareas, pero su tiempo de aprendizaje es a veces demasiado elevado como para justificarlo. Con esta aplicación intentamos solucionarlo, creando un software intuitivo y fácil de usar, para que el usuario/a se centre en lo importante: la gestión de imágenes.

2 Abstract

SatInfManager is an application for storage and manage satellite image information, and be able to search, organize, download, and preprocess those images. For the search of the satellite images, we have used the Copernicus Open Access Hub (online repository for satellite images). The first reason to create this application was the difficulty to find and manage satellite images in a fast and simple way. There are a lot of apps able to do the same operations as SatInfManager, but the time require to learn how to use it, it's sometimes too much. This application tries to solve this problem, with an easy-to-use and intuitive software, so that the user can focus on the key aspect, the managing of the images.

3 Introducción

A medida que nuestra capacidad para monitorizar y gestionar el estado actual de nuestro planeta avanza a pasos agigantados, gracias a los numerosos instrumentos que orbitan la Tierra en este mismo momento, y los que hay planeados para los próximos años y décadas, observamos que dichos recursos, o, mejor dicho, datos, se quedan al final en las manos de unos pocos. Su explotación está restringida, o en su defecto, para acceder a ellas, se requiere, por lo general, pasar un minucioso escrutinio para obtener el permiso para usarlas. Resulta evidente el pequeño avance que se ha hecho para ofrecer dichos recursos a toda la comunidad científica.

Aunque es verdad que cada vez existen más puntos de acceso hacia estos recursos, los que hay actualmente son un poco "antiguos", difíciles de entender y para nada intuitivos, no adecuados a la época en la que vivimos.

Hoy en día se pueden encontrar cada vez más proyectos que requieren de estos datos, con el auge de nuevos algoritmos que emplean técnicas de Deep Learning, y específicamente, de redes neuronales, que pueden ser utilizadas e implementadas por un gran número de personas. Y sin menospreciar a otros métodos de Machine Learning, que siguen siendo los más utilizados en este campo.

Pero estas herramientas no se adecuan a la velocidad de datos que demanda este campo científico, y es ahí donde se encuentra el cuello de botella. De los pocos servicios que hay, se ha de invertir mucho tiempo para aprender a usarlos con soltura.

En este proyecto se ha intentado reducir y mejorar los tiempos que necesita una persona para obtener imágenes satelitales, centrando de nuevo el foco en su gestión.

4 Objetivos y motivaciones

4.1 Objetivos

Como se ha comentado anteriormente, el objetivo principal es ofrecer al cliente final una mayor facilidad de uso sobre el ciclo de obtención de imágenes satelitales. Poder personalizar aquellas imágenes que quiere trabajar en específico, y rapidez para tener información sobre las imágenes de interés accesibles en una misma plataforma. Haciendo uso de servicios en la nube, la persona interesada podrá acceder a su colección de imágenes en cualquier ordenador que tenga instalado la aplicación, para su posterior descargado y procesado.

Está centrada básicamente en reducir el tiempo de búsqueda, clasificación, etc. de las imágenes para su posterior uso en proyectos de investigación, además de tener en la misma aplicación la capacidad de realizar un procesado previo para poder visualizarlas.

Se ha utilizado una metodología de desarrollo ágil, específicamente utilizando el modelo incremental y evolutivo, generando “trozos” de código, y por lo tanto de funcionalidad, en una serie de iteraciones llamadas *sprints*.

Además, se ha priorizado la comunicación con los tutores para obtener el *feedback* necesario para asegurar que la aplicación cumple con los objetivos y las expectativas planteadas al principio del proyecto, realizando una serie de reuniones al final de cada sprint.

Aun así, si había alguna duda sobre alguna funcionalidad o aspecto referido al proyecto, se han hecho tutorías para poder resolverlas.

4.2 Motivaciones

A la hora de realizar cualquier algoritmo de Deep Learning en este campo de la ciencia, encontramos los problemas descritos al inicio de este documento. Pocos servicios gratuitos y mucha desinformación. Si se quiere obtener un conjunto de datos para poder ejecutar tus algoritmos, tienes que recurrir a terceros para obtener dichas imágenes, y que, además, para estos tipos de algoritmos, es crítico contar con un conjunto de datos suficientemente amplio. Si esto lo unimos con el escaso número de repositorios de imágenes satelitales, comprobamos que efectivamente hay una necesitad que no está siendo satisfecha en estos momentos.

Utilizando el punto de acceso del programa Copérnico [1] para obtener imágenes SAR y ópticas recientes, me di cuenta de que para su correcto uso es necesario hacer un procesado previo a dichas imágenes, y así, poder visualizarlas y obtener los datos de las mismas. Para los científicos y profesionales del campo esto no es nada nuevo, de hecho, es algo normal y rutinario, pero que para la gente fuera de este ámbito es totalmente ajeno.

Este último caso es el mío concretamente, puesto que cuando empecé mis prácticas externas de la carrera en el CTIM (Centro de Tecnologías de la Imagen), me dieron la posibilidad de trabajar con estos recursos. Ahí fue, junto a mis tutores, donde nos dimos cuenta de que existía un pequeño cuello de botella con estos recursos. Sí, los podías obtener, pero su formato no era válido para los requisitos de nuestras aplicaciones, o para ejecutar algún algoritmo de Deep Learning o Machine Learning sobre ellas.

Ése fue mi punto de partida, creando pequeños programas en MatLab [2] para “revelar” dichas imágenes. Y a partir de ahí, fueron surgiendo ideas hasta la propuesta final para desarrollar una aplicación que uniera todas estas operaciones en una misma plataforma, fácil y rápida de usar.

Pero con una premisa clara, esta aplicación no es una sustitución del SNAP [3], programa oficial de la ESA para procesar estas imágenes, sino un complemento de esta. Si se quiere un nivel de procesado con un nivel de detalle mayor, es recomendable que utilice el propio SNAP.

Esta aplicación está dirigida a aquellas personas que quieran obtener imágenes con un nivel de procesado medio, centradas en obtener imágenes SAR u ópticas. También es ideal para aquellas personas que prefieran una aplicación que sea más fácil de usar, a diferencia del SNAP, aunque no disponga de todas las funcionalidades para un procesado de mayor nivel.

5 Aportaciones al entorno social

Esta aplicación permite ayudar a explotar y expandir el campo de las imágenes satelitales a un número mayor de personas, eliminando o reduciendo tiempos necesarios para aprender a utilizar herramientas con una cantidad abrumadora de opciones. A poder hacer más accesibles estos recursos que, en el caso del programa Copérnico, ya lo son, pero eliminando las trabas para su utilización.

En el ámbito de la educación, esta aplicación ayudaría muchísimo a estudiantes que empiezan a desarrollar sus propios algoritmos dentro del campo de la teledetección, pero que no tienen un conocimiento avanzado sobre el mismo, sobre un conjunto de datos actualizados, y una serie de características concretas.

En el ámbito científico, también podría aportar a aquellos profesionales que no cuenten con el tiempo para utilizar aplicaciones con una alta curva de aprendizaje, y que su único fin es obtener imágenes “puras”, es decir, sin ningún tipo de procesado ejecutado sobre ellas (a excepción de las operaciones necesarias para “revelarlas”, corrección de terreno, aplicado de un archivo de órbitas, etc.).

Como se ha comentado anteriormente, esta aplicación intenta ampliar el número de usuarios que puede acceder y utilizar imágenes satelitales para su uso personal.

6 Justificación de competencias

IS01: Capacidad para desarrollar, mantener y evaluar servicios y sistemas software que satisfagan todos los requisitos del usuario y se comporten de forma fiable y eficiente, sean asequibles de desarrollar y mantener y cumplan normas de calidad, aplicando las teorías, principios, métodos y prácticas de la ingeniería del software.

Para el desarrollo de esta aplicación ha sido necesario establecer e identificar una serie de requisitos que satisfagan los objetivos finales del proyecto, además implementarlas de forma que permitan la rápida interpretación del mismo, y que ayuden a su posible modificación y mantenimiento.

IS02: Capacidad para valorar las necesidades del cliente y especificar los requisitos software para satisfacer estas necesidades, reconciliando objetivos en conflicto mediante la búsqueda de compromisos aceptables dentro de las limitaciones derivadas del coste, del tiempo, de la existencia de sistemas ya desarrollados y de las propias organizaciones.

Este proyecto nace de las necesidades expresadas por mis tutores sobre su dificultad para encontrar y procesar imágenes satelitales. Entre los dos hemos valorados los diferentes requisitos que necesita cumplir la aplicación, y cuáles de ellos no eran posible implementarlos.

CP01: Capacidad para tener un conocimiento profundo de los principios fundamentales y modelos de la computación y saberlos aplicar para interpretar, seleccionar, valorar, modelar, y crear nuevos conceptos, teorías, usos y desarrollos tecnológicos relacionados con la informática.

Relacionado con el procesado de imágenes satelitales, con los diferentes algoritmos que se deben aplicar sobre ellas para obtener una imagen procesada, cumpliendo con las necesidades de los usuarios.

7 Estado del arte

En la actualidad existe una gran diversidad respecto a los instrumentos de observación que obtienen datos alrededor del mundo, siendo de vital importancia aquellos que se relacionan con la monitorización de nuestro planeta, y que se dedican, por ejemplo, a la observación de los casquetes polares, subida del nivel del mar, cambios de temperatura, desastres naturales, etc.

El campo que se dedica a realizar estas observaciones se denomina Teledetección, o detección remota (Remote Sensing), y como se ha comentado anteriormente, su objetivo es la monitorización física de nuestro planeta, estudiando una serie de señales de radiación que emite y refleja la superficie terrestre.

Algunos ejemplos de dispositivos que realizan este tipo de observaciones serían los satélites y aviones que obtienen imágenes sobre grandes áreas de terreno que no se pueden de conseguir desde la superficie; sistemas sónicos en barcos para crear un mapeado del fondo oceánico, etc.

En este proyecto se centra en los satélites, y en los diferentes tipos de imágenes que podemos obtener de ellos. Estos dispositivos los podemos separar en dos grandes grupos según la transmisión de sus señales: aquellos que **transmiten y reciben** (satélites activos), y otros que solo **reciben** (satélites pasivos).

Un ejemplo de satélites pasivos serían los ópticos, que reciben las ondas del espectro electromagnético reflejadas por nuestro planeta. Este tipo de satélites es el más común, y, de hecho, el que utilizamos ya casi a diario en nuestra vida cotidiana. Google Maps utiliza este tipo de satélites para obtener una imagen natural de todo nuestro planeta, con un grado de resolución increíblemente alto. Pero ¿cómo funcionan en realidad? Para poder contestar a esta pregunta, primero tenemos que conocer algunos términos. El primero de ellos es **imagen multiespectral**.

7.1 Imágenes multiespectrales

Este tipo de imagen trabaja sobre un rango amplio del espectro electromagnético de la luz, desde el infrarrojo hasta el ultravioleta, aportando nuevas formas de ver el mundo que es invisible a nuestros ojos, puesto que solo trabajan en la zona visible del espectro (ver Ilustración 1).

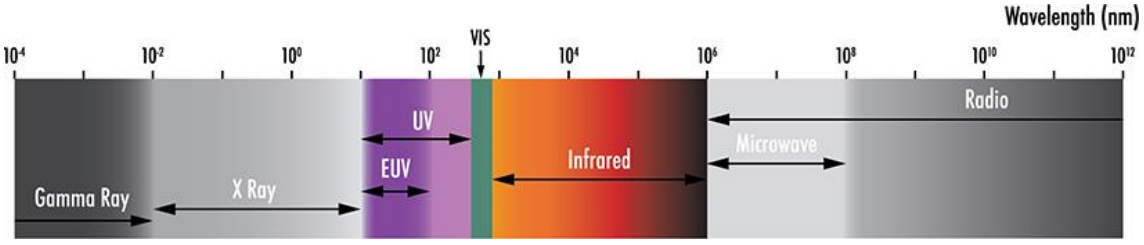


Ilustración 1 - Espectro electromagnética. Imagen obtenida de https://www.edmundoptics.es/

Este tipo de imágenes están formadas por un conjunto de bandas, cada una correspondiendo a un rango específico del espectro (ver ilustración 2). Cada de una de estas bandas tiene una numeración. Por ejemplo, la banda B4 se correspondería con la zona del espectro que denominamos “rojo”. B3 sería el verde, y el B2 el azul. Normalmente, este tipo de imágenes cuenta entre 1 y 12 bandas, dependiendo del tipo de instrumento.

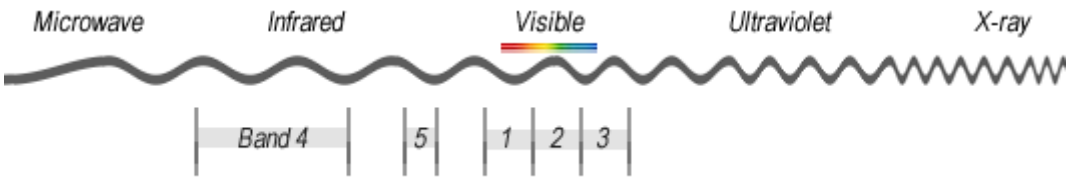


Ilustración 2 - Relación entre las bandas y el espectro electromagnético. Imagen obtenida de https://gisgeography.com/

Conociendo cuáles son las relaciones entre las bandas y su correspondencia dentro del rango del espectro óptico, podemos crear imágenes que destaquen con ciertas características u otras. Si queremos obtener una imagen dentro del rango visible, utilizamos las componentes **B4, B3** y **B2**, que se corresponden con el rojo, verde y azul, respectivamente (ver Ilustración 3).

Ilustración 3 - Imagen óptica al natural de La Palma

Si reorganizamos las bandas en otras combinaciones, podemos obtener de una forma visual la vegetación de una zona, puesto que esta refleja la luz a un rango del espectro definido. Algunos de estos perfiles pueden ser, por ejemplo, el de “Vegetación saludable”, utilizando bandas cerca del infrarrojo (ver Ilustración 4).

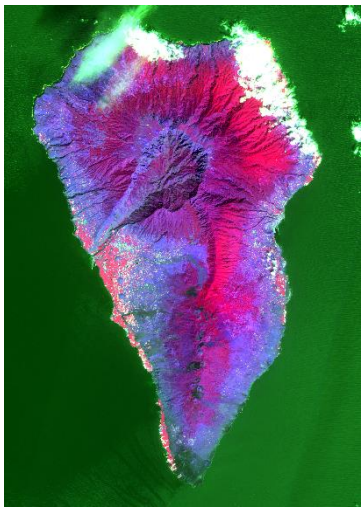


Ilustración 4 - Imagen de vegetación saludable de La Palma, compuesta por las bandas B8 (asociada a un rango cerca del infrarrojo), B2 (dentro de la zona visible y asociada al azul) y B11 (rango de onda corta del infrarrojo)

Sin embargo, hoy en día este tipo de imágenes está siendo sustituida por **imágenes hiperespectrales**.

7.2 Imágenes Hiperesprectrales

Su funcionamiento es muy parecido, pero con una diferencia que la hace destacar, sobre todo entre la comunidad científica, y es el rango del espectro que es asignada a cada banda. En este tipo de imágenes, este rango es muy pequeño, en torno a 10-20 nanómetros.

Esto hace que donde la imagen multiespectral tuviera una banda, una **hiperespectral** pueda tener cientos, ofreciendo un potencial espectacular para el estudio en este campo del procesado de imágenes (ver Ilustración 5) [4].

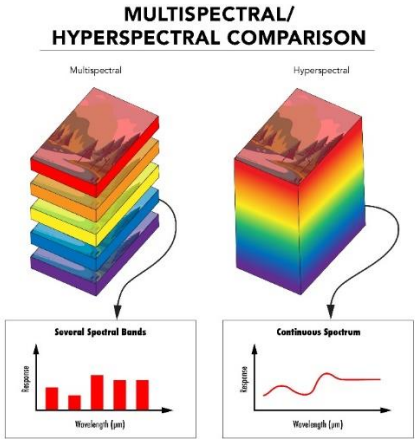


Ilustración 5 - Comparación entre imágenes multiespectrales e hiperespectrales. Imagen obtenida de https://www.celestis.com/

El inconveniente es que este tipo de imagen es menos común, y, además, más costosas a nivel computacional. Cuantas más bandas tenga la imagen, más características será capaz de identificar. En comparación con las imágenes multiespectrales, una banda que determinaba la vegetación en una imagen, en una hiperespectral puede identificar qué tipo de vegetación hay en una zona determinada, asociando cada tipo a una banda en concreto.

Sin embargo, ambos tipos de imágenes tienen un problema común, son susceptibles a la luz, tanto de día como de noche. Esto es algo evidente, puesto que estas imágenes se basan en el reflejo de la luz sobre la superficie del planeta, siendo recibida por un satélite o un avión. Cada tipo de material u objeto absorbe una determinada parte del espectro, y el resultado es reflejada al espacio, donde el satélite las recibe (ver ilustración 6).

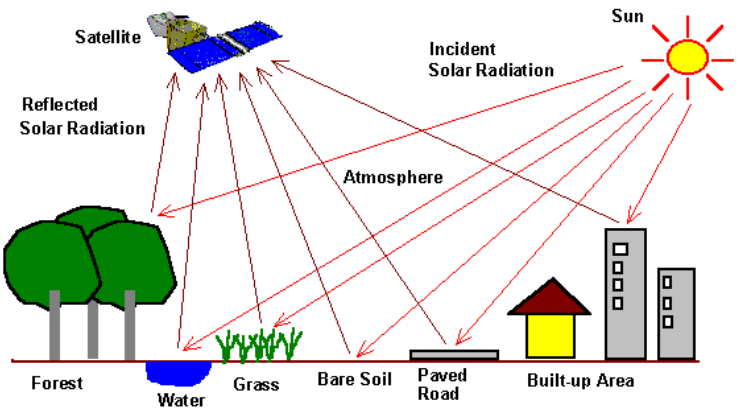


Ilustración 6 - Obtención de imágenes ópticas. Imagen obtenida de https://crisp.nus.edu.sg/

Y, además, también dependen de las condiciones meteorológicas. Si está nublado, se obtendrá una imagen cubierta completamente por ellas, perdiendo la vista de la superficie (ver ilustración 7).

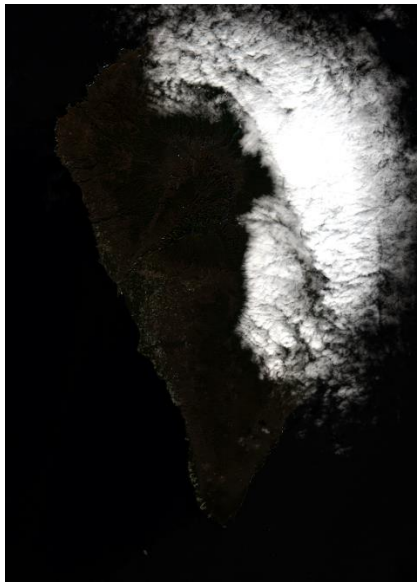


Ilustración 7 - Imagen óptica afectada por la luz y las condiciones meteorológicas

7.3 Imágenes RADAR o SAR

Sin embargo, se pueden obtener imágenes satelitales independientemente de las condiciones anteriormente descritas, y es utilizando otro tipo diferente de sensor. Las imágenes radar o **SAR** (***Synthetic Aperture Radar*)** permiten obtener imágenes a cualquier hora del día, y no es afectada por las condiciones meteorológicas. Esto es gracias a que su obtención no se basa la recepción del espectro óptico de la superficie, sino en una serie de transmisiones y recepciones

de señales (que podríamos decir que iluminan) hacia la superficie terrestre, creando una imagen con los detalles de dicha superficie (ver ilustración 8).

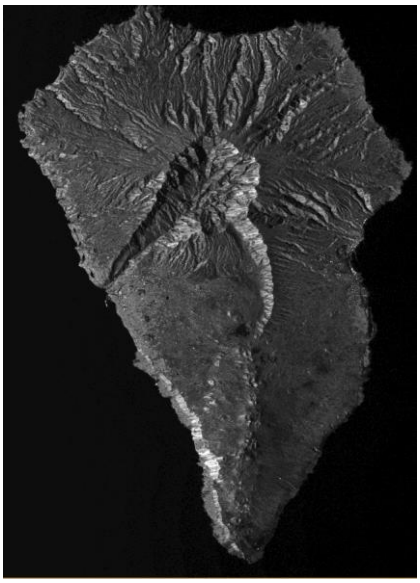


Ilustración 8 - Imagen SAR de La Palma

Un instrumento de estas características cuenta con un conjunto de antenas que permiten un rápido escaneado en elevación, ángulo que representa la elevación de un objeto respecto al horizonte, y en azimut, que representa el ángulo entre un objeto celeste y el norte (si está justo en el norte tendrá un azimut de 0º, en el sur de 180º) (ver ilustración 9).

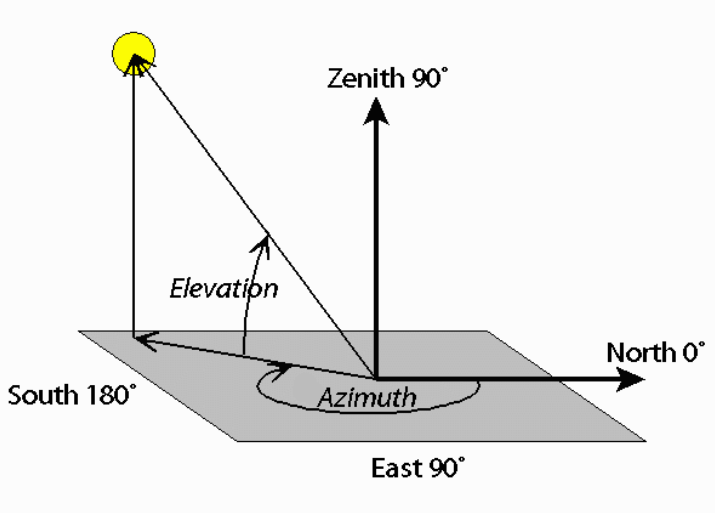


Ilustración 9 - Coordenadas en elevación y azimuth. Imagen obtenida de https://www.celestis.com/

Para obtener dichas coordenadas, primero el satélite transmite una señal con una determinada energía, que es reflejada por la superficie. Después, vuelve al espacio donde retorna al sensor del mismo satélite, pero con un nivel de energía diferente.

A diferencia de las imágenes ópticas, este tipo requiere un sistema mucho más complejo para poder pre-procesar dichas imágenes. Hay que tener en cuenta las probables modificaciones que pueden aparecer en las señales que emite el satélite cuando llega a la superficie terrestre (ver

Ilustración 10). Esta puede reflejarse sobre una superficie con una topografía muy cambiante, produciendo que la señal se refleje en varias direcciones, reduciendo la que puede llegar al satélite. Otra es la del doble rebote, donde la señal se refleja en varias superficies hasta llegar de nuevo al satélite.

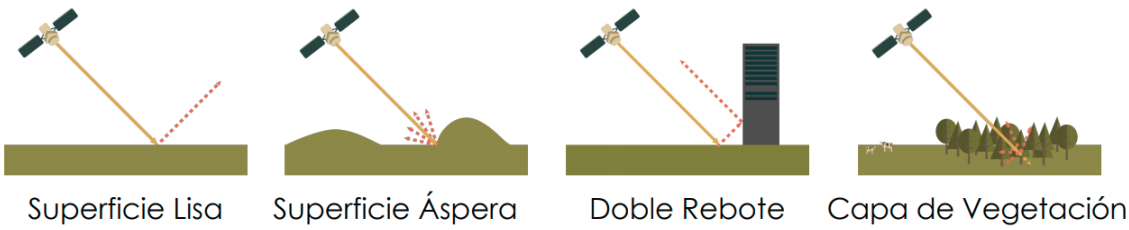


Ilustración 10 - Escenarios durante la transmisión y recepción de la señal. Imagen obtenida de https://aguaysig.com/

Para la trasmisión y recepción de la señal, estos instrumentos utilizan una serie de polarizaciones. Existen dos variantes, la vertical y horizontal. Un instrumento SAR transmite la señal en una determinada polarización, y es recibida utilizando la misma u otra diferente (ver ilustración 11).

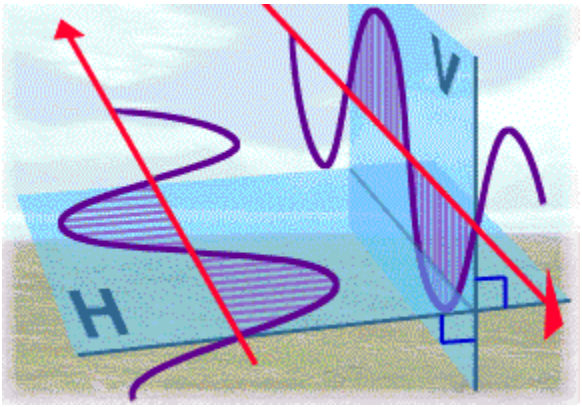


Ilustración 11 - Polarización de una señal. Imagen obtenida de http://envisat.esa.int/

Este tipo de polarizaciones permite resaltar ciertas características de la superficie, como vegetación o zonas urbanas, fondo oceánico, etc. Aquellas imágenes que solo transmiten y reciben con una determinada polarización se denominan "**Single Polametric SAR**", SAR de Polarización Única, y son los productos más básicos.

Sin embargo, en la actualidad contamos con otros que permiten utilizar varias polarizaciones a la vez, y son las "**Fully Polametric SAR**", SAR de Polarizaciones Completas. La clave de estos tipos de imagen son las diferentes combinaciones de polarizaciones que permite, danto lugar a pares como HH o VH. Cada pareja significa una transmisión y una recepción. En el caso de HH, significa que se ha transmitido en polarización horizontal y se ha recibido en horizontal también. VH se ha transmitido en vertical, y recibido en horizontal. Sin embargo, al aumentar el número de polarizaciones que permite, también incrementa su complejidad.

Este tipo de imagen es altamente utilizada en algoritmos de Machine Learning, primero debido a que no le afecta las condiciones meteorológicas ni que sea de día y de noche, y segundo, por la cantidad de información que se puede obtener de ellas. Se pueden utilizar para identificar zonas, materiales, variaciones en la altura de superficie (interferometría) tras un terremoto, o en un rango de tiempo, etc. (ver ilustración 12 y 13)

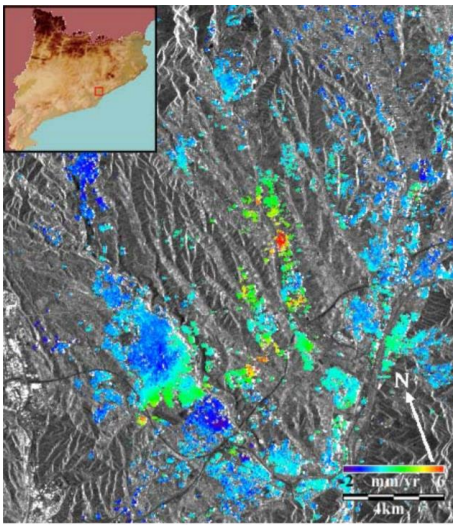


Ilustración 12 - Mapa de interferometría. Imagen obtenida de [5]

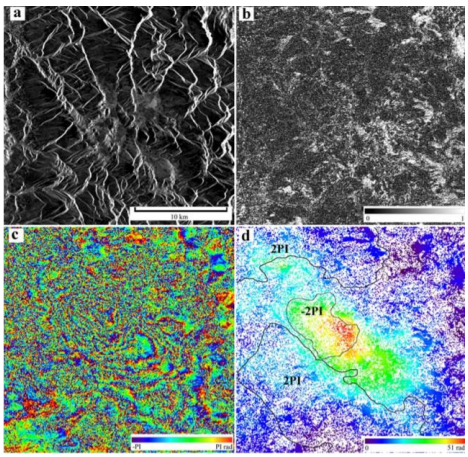


Ilustración 13 - Variación de la altura del suelo de una zona. Imagen obtenida de [5]

Actualmente, existen varios satélites que se dedican a la obtención de uno de los tipos de imagen anteriormente descritos en concreto. En imágenes ópticas se encuentran los Landsat, que llevan monitorizando el planeta desde la década de los 70, y que actualmente cuenta con dos satélites, el **Landsat 7 y 8** [6], a los que pronto se le unirá el 9, la próxima generación, que contará con una mejor gama de instrumentos para obtener respuestas ante eventos como la deforestación global, la expansión urbana, etc.; el Spot, con sus satélites **Spot-6** y **Spot-7**, que es una costelación de observación con fines comerciales; la familia de la **NOAA** (*National Oceanic and Atphmospheric Administration*), que cuenta con varios satélites destinados a la observación de los cambios de temperatura en el planeta y en los océanos, monitorización del tiempo, etc.; el **Sentinel-2**, de la Agencia Espacial Europea, que trabaja especialmente en el campo de la agricultura.

Respecto a las imágenes SAR, destacan satélites como el Radarsat-2, enfocada a la monitorización marítima, desastres naturales, control del hielo e icebergs, etc.; el Comos-Skymed y el Sentinel 1, ambos dedicados a la monitorización tanto marítima y de superficie, tanto para fines comerciales como militares (ver ilustración 14).

Ilustración 14 - Ejemplo de algunos de los satélites que existen en la actualidad. Imagen obtenida de https://www.intelligence-airbusds.com/

7.4 Programa Copérnico

Uno de los primeros puntos de acceso a recursos sobre imágenes satelitales y que fueran libres para usar fue el **programa Copérnico**.

Este programa está desarrollado por la Agencia Espacial Europea, y su objetivo es proporcionar una red de satélites que se dediquen a la monitorización del planeta, abarcando el mayor rango de aplicaciones posibles. Así, se puede encontrar satélites ya anteriormente nombrados, como el Sentinel 1 y el Sentinel 2, obteniendo imágenes Radar y ópticas, respectivamente. También se encuentra el Sentinel 3, que permiten tomar medidas sobre la superficie de los océanos, monitorización de la temperatura en tierra y océanos, obtención de imágenes ópticas; y el Sentinel 5, que está enfocado a la monitorización atmosférica, comprobando y controlando los diferentes elementos químicos que se encuentran en ella, como, por ejemplo, el ozono.

Además, se sigue expandiendo, con el lanzamiento de un nuevo instrumento, el Sentinel-6, dedicado exclusivamente a la medición de los cambios sobre el planeta causados por el cambio climático, concretamente, en el aumento de la altura de los océanos.

Como se puede apreciar, este conjunto de satélites englobados dentro de un mismo programa es perfecto para la comunidad científica, teniendo acceso a una gran cantidad de recursos de manera inmediata, sin trabas y de fácil acceso.

De hecho, esto es uno de los puntos clave sobre los que se basa el programa Copérnico, acceso gratuito a toda la comunidad, no solo a científicos de todo el mundo, sino también a personas fuera de este ámbito que tengan un interés particular en imágenes satelitales.

Dentro de la amplia gama de instrumentos que ofrece el Copérnico, se ha centrado en dos tipos principalmente, las imágenes RADAR y ópticas, y, por lo tanto, en los satélites Sentinel 1 y Sentinel 2, respectivamente.

7.5 Sentinel 1

Este satélite cuenta con un instrumento SAR que soporta operación tanto con polarización única (HH, HV, VV, o VH) como polarización dual (VV-VH o HH-HV) (ver ilustración 15). Además, cuenta con un rango de subproductos, enfocados a un tipo de aplicación en concreto. Por último, también permite modificar el modo de obtención de las imágenes, eligiendo entre los distintos modos de adquisición que ofrece (IW, EW, SM, etc.).

Ilustración 15 - Renderizado del Sentinel 1. Imagen obtenida de https://sentinel.esa.int/

Los subproductos que ofrece el Sentinel 1 están divididos en tres niveles, atendiendo al grado de procesado inicial que poseen:

- Nivel 0 – RAW: Este tipo de producto no se le ha aplicado ningún tipo de procesado, y es el dato obtenido por el instrumento SAR al realizar la toma de una imagen. Es el producto menos usado, ya que no es fácil su utilización, y la ESA no da soporte para poder procesarlas.
- Nivel 1
    - **Single Look Complex (SLC):** Productos que cuentan con datos SAR enfocados y georreferenciados usando datos de la órbita y de altitud. Además, y lo más destacado de este tipo de producto, es que conserva la información de fase.
    - **Ground Range Detected (GRD)**: Son productos SLC que han sido procesados usando un modelo elipsoide de la Tierra. Sin embargo, se pierde resolución especial y no cuentan con información de fase.
- Nivel 2: Productos oceánicos
    - Productos relacionados con el viento oceánico, y que se encargan de su monitorización.

Modos de adquisición:

- **Stripmap Mode**: Obtiene imágenes con una resolución espacial de 5m x 5m (es decir, que cada píxel representa 5 metros), y trabaja sobre una localización con un área pequeña, sobre unos 80km,
- **Interferometric Wide Swath Mode**: Este modo combina una resolución moderada (en torno a 5 metros y 20 metros) sobre una localización con un área de dimensiones entorno a unos 250km. Este es el modo por defecto en observaciones sobre superficie terrestre.

- **ExtraWide Swath Mode**: Modo orientado para servicios marítimos, donde son necesarios una gran zona de trabajo y un tiempo de revisitado corto. Su funcionamiento es parecido, pero tiene una menor resolución, entre 20 y 40 metros
- **Wave Mode**: Este modo permite, junto a los modelos de olas globales de los océanos, determinar la dirección, longitud y tamaño de olas en el oceánico abierto.

En la ilustración 16, se observa un modelo visual de los diferentes modos de adquisición del Sentinel 1.

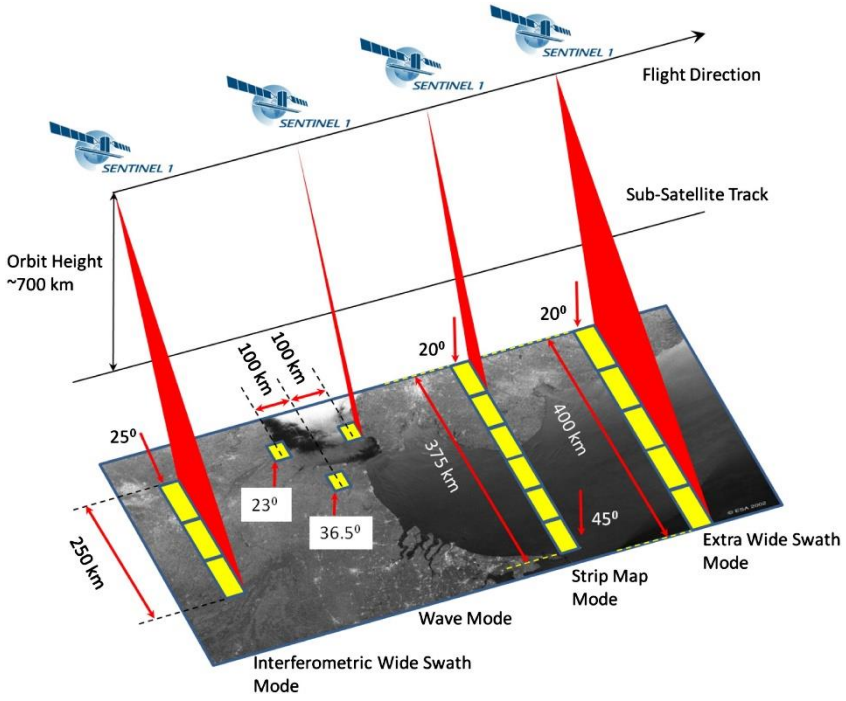


Ilustración 16 - Modos de adquisición del Sentinel 1. Imagen obtenida de https://sentinel.esa.int/

Otro aspecto que destacar es que los tipos de productos varían según el modo de adquisición. Por ejemplo, en el modo de OCN, no tenemos productos de tipo 1 ni 0, únicamente de nivel 2, que son los relacionados con el océano. En la ilustración 17 se muestra cómo se clasifican según el modo de adquisición y el nivel del producto.

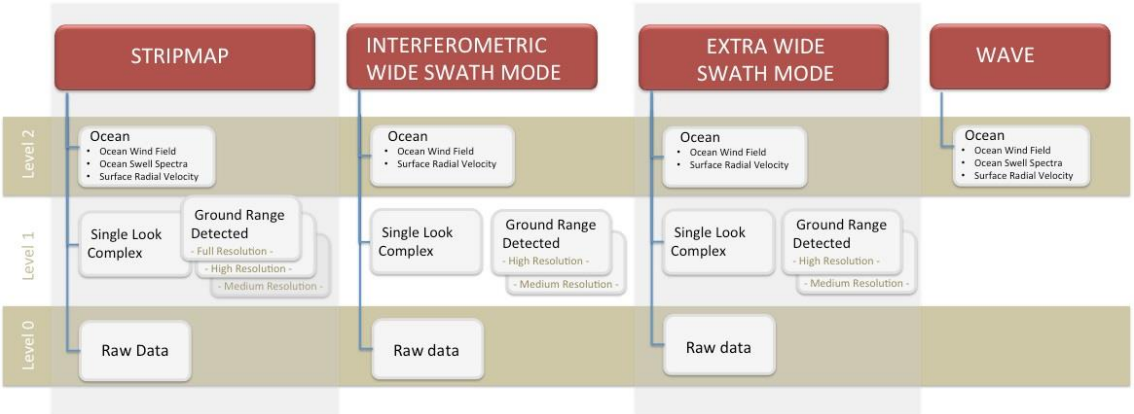


Ilustración 17 - Modos de adquisición y niveles de productos. Imagen obtenida de https://sentinel.esa.int/

La ESA publica cada año el tiempo de cobertura media entre observaciones en una localización en concreto. El último mapa oficial muestra que, para Europa, el tiempo de revisita es de unos 6 días, mientras que para el resto del mundo es de casi dos semanas, a excepción de los polos, que tienen el menor tiempo de revisita entre 1 y 4 días (ver ilustración 18).

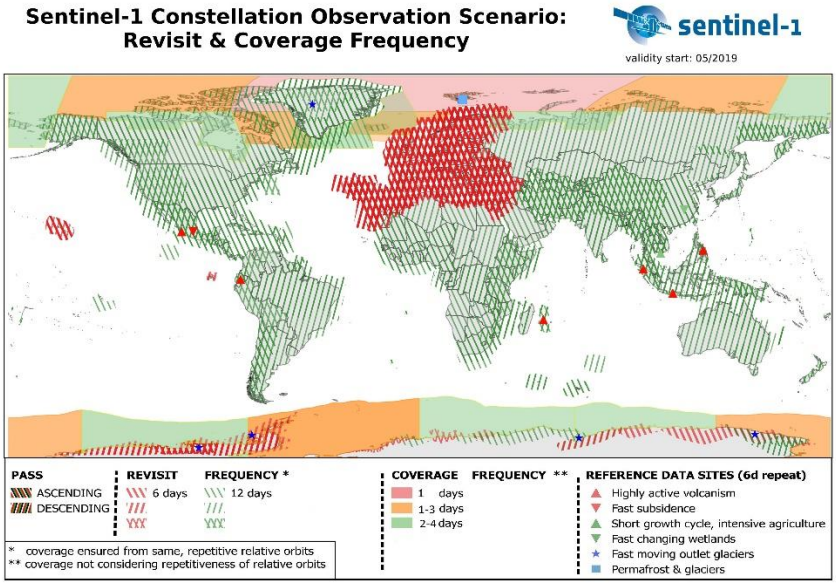


Ilustración 18 - Tiempo de revisitado del Sentinel 1. Imagen obtenida de https://sentinel.esa.int/

Los productos más comunes son los de nivel 1, junto al modo de adquisición IW. Esto se debe algunos de estos modos y productos están reservados para una zona del planeta en concreto. La ESA también ofrece un mapa para conocer cómo se distribuyen. También indica las polarizaciones utilizadas, reservando las polarizaciones HH o HH-HV para los polos, y para resto de la superficie terrestre las polarizaciones VV o VV-VH (ver ilustración 19).

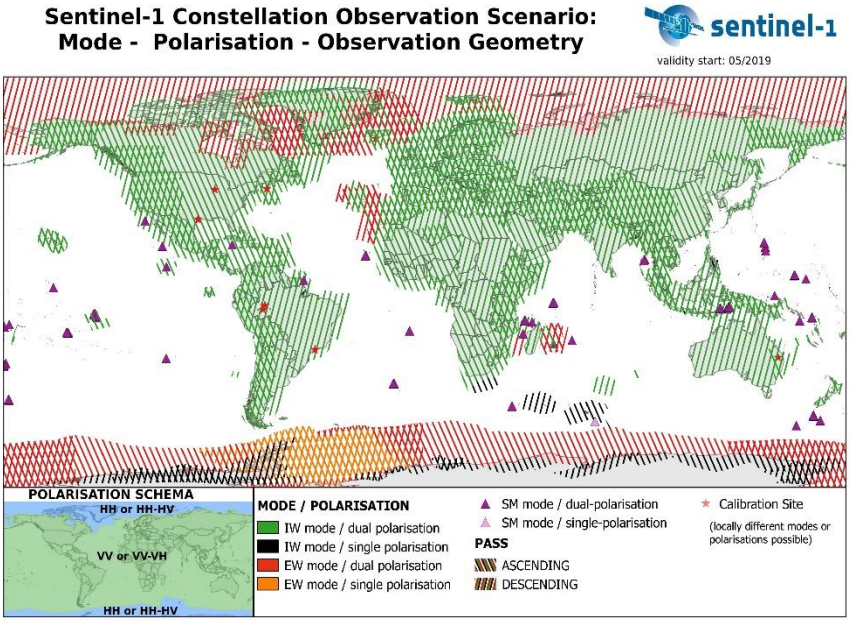


Ilustración 19 - Escenario de observación con la disponibilidad de las polarizaciones por zonas. Imagen obtenida de https://sentinel.esa.int/

7.6 Sentinel 2

El Sentinel 2 es una constelación de dos satélites que contienen un único instrumento multiespectral para la obtención de imágenes ópticas [7]. Su objetivo es monitorizar las condiciones de la superficie terrestre en grandes áreas y alto nivel de revisita (unos 10 días en localizaciones cerca del ecuador), aunque esto depende de las condiciones meteorológicas de la zona.

El instrumento multiespectral permite obtener un número total de 13 bandas: 4 de ellas con una resolución de 10 metros, 6 con una resolución de 20 metros, y 3 con 60 metros (ver ilustración 20).

Spatial Resolution (m)	Band Number	S2A		S2B	
		Central Wavelength (nm)	Bandwidth (nm)	Central Wavelength (nm)	Bandwidth (nm)
10	2	492.4	66	492.1	66
	3	559.8	36	559.0	36
	4	664.6	31	664.9	31
	8	832.8	106	832.9	106
20	5	704.1	15	703.8	16
	6	740.5	15	739.1	15
	7	782.8	20	779.7	20
	8a	864.7	21	864.0	22
	11	1613.7	91	1610.4	94
	12	2202.4	175	2185.7	185
60	1	442.7	21	442.2	21
	9	945.1	20	943.2	21
	10	1373.5	31	1376.9	30

Ilustración 20 - Bandas espectrales obtenidas por el Sentinel 2. Imagen obtenida de https://sentinel.esa.int/

Al igual que el Sentinel 1, el Sentinel 2 produce dos tipos de productos: 1 y 2.

- El nivel 1 proporciona imágenes que no han sido corregidas atmosféricamente, es decir, que no se ha eliminado aquella radiación que no ha sido emitida por la superficie, sino por la atmosfera.
- El nivel 2 son productos que han sido corregidos utilizando una ratio de reflectancia denominado "**Top of Atmosphere Reflectance**", y que determina la ratio de la radiación reflejada por el sol en una superficie determinada.

7.7 Portal de acceso

Todos estos satélites que realizan una tarea concreta cada uno, proporciona al programa Copérnico de un excelente servicio para un rango de aplicaciones muy amplio. Y como se ha descrito anteriormente, todos estos recursos están disponibles para toda la comunidad de manera gratuita.

Para proporcionar estos recursos, la Agencia Espacial Europea desarrolló un portal web, llamado **SciHub** (ver ilustración 21), con el fin de unificar los diferentes productos bajo un mismo punto de acceso. El único requisito para empezar a utilizarlas es registrarte en el portal. Las

credenciales que se utilicen aquí se migrarán a las diferentes APIs para que los usuarios no tengan que realizar ninguna gestión adicional.

A partir de la página principal, se encuentran varios recursos a los que se puede acceder. Primero tenemos un **Open Hub**, que es una interfaz gráfica para realizar búsquedas productos del mismo, a través de un navegador web. La siguiente es la **API Hub**, que es el recurso que se ha utilizado el proyecto. Luego tenemos otro punto de acceso a productos del Sentinel 5 y relacionados.

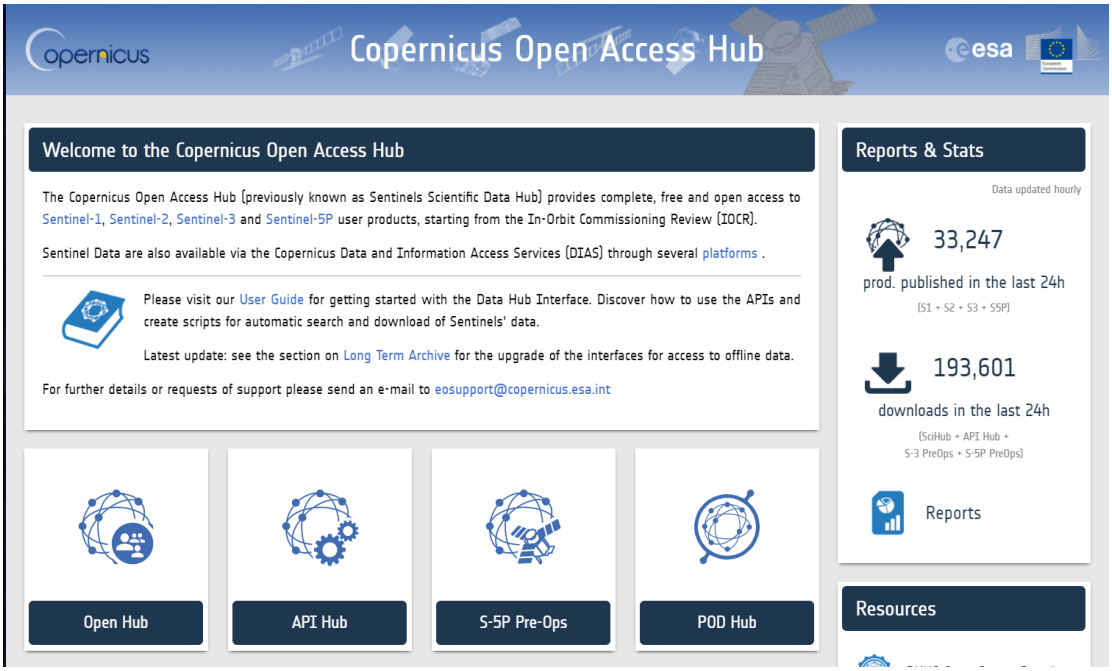


Ilustración 21 - Página principal del portal web del programa Copérnico. Imagen obtenida de https://scihub.copernicus.eu/

Esta API cuenta con varios recursos implementados bajo diferentes protocolos para ofrecer los productos satelitales a los usuarios (ver ilustración 22). Actualmente son dos, una de ellas implementada bajo el protocolo **OpenSearch**, y otra con el protocolo **OData**.

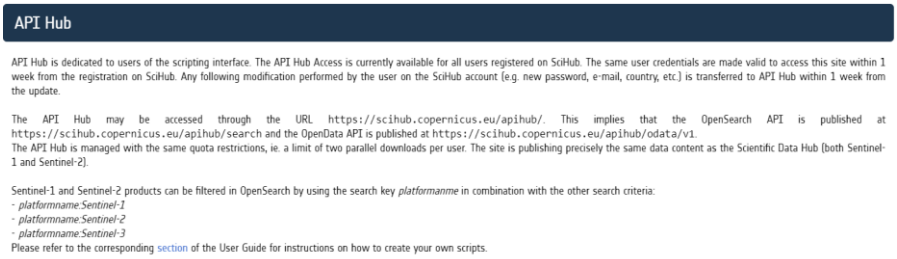


Ilustración 22 - Página principal de la API del Copérnico. Imagen obtenida de https://scihub.copernicus.eu/

7.7.1 OData

El protocolo **OData** se define como una interfaz diseñada para protocolos como **HTTP** y **REST** con el fin de ofrecer a un gran número de personas una herramienta de fácil acceso (utilizando un navegador web) para proveer servicios no solo para la búsqueda de elementos, sino también de descargas, a través de un único sistema de identificación basado en **URI** (*Uniform Resource Identifiers*).

Explicando un poco mejor cómo están repartidos los recursos bajo el protocolo **OData**, se observa que están divididos en una serie de entidades, por ejemplo, “Products”, y que dentro de esta colección se encuentran todos los productos accesibles por el servicio. Otra entidad que ofrece el Copérnico son las de los productos eliminados, que como su propio nombre dice, son aquellos que ya han sido eliminados de la API (se eliminan por antigüedad).

Por lo tanto, para acceder a un producto tenemos que acceder a la entidad de “Products”. Aquí hay varias opciones. La primera es que permite buscar un producto individualmente, utilizando un identificador único entre todos los productos que tiene el servidor, estén eliminados o no. Y la otra es un conjunto de filtros para obtener aquellos que cumplan con los parámetros especificados. Además, proporciona información sobre si un producto está disponible para descargar, debido al sistema de mantenimiento de productos del Copérnico (un producto está disponible durante un año después de haber sido registrado en la API).

7.7.2 OpenSearch

El recurso de búsqueda está implementado utilizando el protocolo **OpenSearch**. Desarrollado originalmente por Amazon, este protocolo contiene y utiliza un conjunto de tecnologías que permiten ofrecer datos de una forma generalizada y estandarizada, siendo accesible para todas las personas que lo requieran.

Dicho protocolo cuenta con una serie de parámetros fijos, que son inmutables indiferentemente del servicio que lo proporciona. Estos son, por ejemplo, aquellos parámetros responsables de la paginación de los datos que se muestran en la petición: número de elementos a mostrar, índice de la página actual, número de elementos totales dentro del servicio, etc.

Y por supuesto, cuenta con una forma común para pasar parámetros para realizar un filtrado para obtener aquellos elementos que coinciden con las características especificadas por los usuarios.

7.8 Procesado - SNAP

El **SNAP** es un programa de código abierto desarrollado por la Agencia Espacial Europea para el pre, y post procesado de imágenes satelitales, y que permite el procesado de todas las imágenes dentro del programa Copérnico.

Esta aplicación dispone de diferentes herramientas y operaciones para ejecutar sobre las imágenes, pudiendo ejecutar la gran mayoría de operaciones que los usuarios finales (personas dentro del campo científico) requieran.

Sin embargo, esta aplicación para personas ajenas al ámbito de la teledetección puede ser algo abrumadora, debido al gran número de opciones, botones, funcionalidades, etc. que permite.

Y un ejemplo para mostrar esto, es que a través de esta aplicación también es posible buscar productos, pero dichas opciones no están a simple vista, y tienes que dedicar bastante tiempo para su comprensión y posterior uso. (ver ilustración 23) [8].

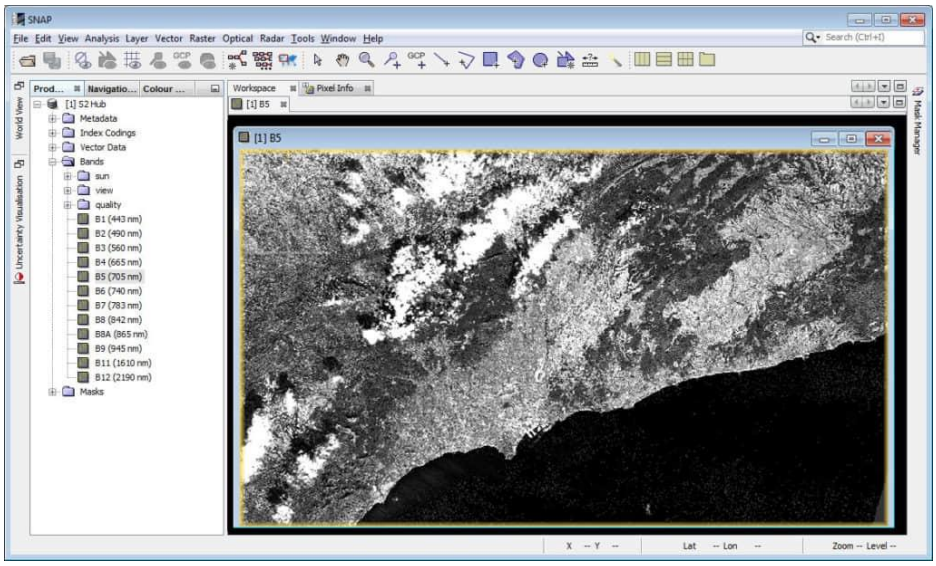


Ilustración 23 - Programa SNAP. Imagen obtenida de https://geoinnova.org/

Además, ofrece capacidades para realizar tipos de procesado más complejo, como por ejemplo interferometría de varios productos distribuidos a lo largo de un rango de tiempo. Sin embargo, para realizar estas operaciones se necesita de una capacidad computacional considerable, pues el volumen de datos, y también el área de trabajo, es a veces demasiado elevado.

Pero, aun así, este programa lo tiene todo para obtener el máximo partido a las imágenes satelitales del Copérnico, pero ofreciendo a cambio un tiempo para poder aprender a utilizarlo

8 Recursos utilizados

8.1 Intellij Communite

Es un entorno de desarrollo de gratuito de aplicaciones software en varios lenguajes de programación. Da soporte a desarrollos en diferentes plataformas, además de una gran cantidad de frameworks externos como Maven [9].

8.2 StarUML

Programa de modelaje de clases para desarrollos software. Se ha utilizado para organizar de manera gráfica las diferentes partes de la aplicación, así como su comportamiento mediante diagramas de actividad, etc.

8.3 Google Drive

Repositorio remoto para almacenamiento en la nube. Se ha utilizado para almacenar los diferentes documentos con los requisitos, casos de uso, bibliografías, etc.

8.4 Java

Lenguaje de programación orientado a objetos, uno de los líderes en su sector, y uno de los más conocidos a nivel mundial. Se le conoce por ser fuertemente tipado, pero robusto y aportando un gran rendimiento. Es el lenguaje seleccionado para desarrollar la aplicación. [10]

8.5 Maven

Repositorio online de librerías de Java. Permite importar librerías sin la necesidad de incluirlas manualmente en el proyecto. Además, ofrece operaciones de limpieza, ejecución de pruebas, inicio de la aplicación, empaquetado, etc.

Maven se ha convertido en una herramienta indispensable para los desarrolladores de Java hoy en día, gracias a su facilidad de uso, y su alto rendimiento

8.6 Git

Herramienta que permite un control de versiones de un proyecto. Mediante una serie de comandos podremos saber qué archivos se han modificado desde la última vez que se actualizó el código, pudiendo realizar operaciones sobre ellas.

Permite separar las versiones en una serie de “ramas”, dando la posibilidad de desarrollar en paralelo con varios programadores a la vez. También ofrece opciones de resoluciones de conflictos, puesto que es posible que varios programadores estén trabajando simultáneamente en la misma zona de código. Git permite además una serie de operaciones llamadas merge para solucionar dichos problemas.

8.7 MongoDB Cloud

Gestor de bases de datos NoSQL en la nube, gestionada por MongoDB. Es un servicio web totalmente gratuito si no sobrepasas el límite de almacenamiento. Ofrece la capacidad de hacer accesible toda la información de la base de datos a todos los usuarios, siempre y cuando tengan permisos para realizar operaciones de lectura y escritura.

8.8 JavaFX y FXML

JavaFX es un framework para el desarrollo de interfaces gráficas en Java [11]. Su funcionamiento es parecido al de Swing, donde se cuenta con una serie de elementos gráficos, como por ejemplo áreas de texto, listas, paneles, menús, etc.

JavaFX además viene con un lenguaje basado en etiquetas llamado FXML, donde a partir de un esquema XML, podemos crear una interfaz sin necesidad de utilizar código Java. Esto permite desacoplar las vistas de los controladores, y poder utilizar varias vistas sin necesidad de tener que tocar al código del controlador.

También ofrece un cargador para archivos FXML, donde se puede cargar una vista y asignarla a un controlador en concreto. Otra posibilidad es asignar en el FXML el controlador del mismo, y, entonces, el cargador obtendría la vista, y también el controlador, con todos los elementos inyectados en la clase, gracias a una serie de anotaciones.

8.9 Scene Builder

Aplicación desarrollada en Java y especializada en las creaciones de interfaces desarrolladas en JavaFX. Utilizando el sistema de etiquetas FXML, es capaz de crear interfaces de JavaFX, dando la capacidad de modificar su apariencia en la misma aplicación. Además, da soporte el cargado automático de controladores, relacionando la vista con un controlador en concreto.

8.10 Launch4j

Para la creación del ejecutable en formato EXE a partir del fichero JAR generado por Maven, se ha utilizado la aplicación Launch4j, el cual permite la creación y la configuración de dicha conversión. Algunos parámetros de esta configuración permiten establecer la versión mínima y máxima del JRE de Java, el nombre de la aplicación, versión, establecer icono, etc.

8.11 Inno Setup Compiler

Aplicación para la creación y la personalización de un instalador para un fichero ejecutable. Permite además poder añadir también las librerías requeridas por la aplicación para su correcto funcionamiento, además de instalar una serie de archivos en el directorio personal establecido por el Sistema Operativo.

8.12 Start UML

Aplicación para la creación de diseños de un desarrollo software. Permite de una forma fácil e intuitiva diseñar la aplicación software con una gama de diagramas, como, por ejemplo, diagrama de clases, de secuencia, de actividades, de casos de uso, etc.

9 Librerías utilizadas

9.1 GeoTools

Librería para la geolocalización de áreas en un mapa y obtención de coordenadas reales (latitud y longitud) [12].

9.2 SNAP

Librería para el procesado de imágenes satelitales del programa Copérnico [13].

9.3 Morphia

Librería para obtener los datos almacenados en la base de datos MongoDB, además de proporcionar operaciones de guardado, eliminado y filtrado de los mismos [14].

9.4 Jackson

Librería para "pasar" datos en formato JSON a objetos Java (y viceversa) mediante una serie de anotaciones [15].

9.5 ControlsFX

Librería que añade nuevos componentes gráficos a JavaFX, como, por ejemplo, nuevas barras de progresos, nuevos tipos de botones, etc. [16].

9.6 JMetro

Librería para mejorar la estética de la interfaz de JavaFX, ofreciendo nuevos estilos para aplicar a los diferentes componentes de JavaFX [17].

9.7 JPhoenix

Librería para JavaFX que ofrece nuevos componentes, como paneles, botones, etc., mejorando la estética de la interfaz [18].

9.8 Iconos FontAwesome y MaterialDesign

Librería de iconos FontAwesome y también de MaterialDesign compatible con interfaces JavaFX [19].

10 Análisis

Como se ha explicado anteriormente, esta aplicación es un software implementado en Java y utilizando el framework de **JavaFX** para la creación de un prototipo de interfaz. El objetivo es crear una interfaz que sea lo más sencilla e intuitiva posible, sin demasiadas opciones, solo las más básicas e importantes. Es clave que la aplicación sea fácil de utilizar para aquellos usuarios que no dominan el campo de imágenes satelitales.

Su funcionamiento se basa en una aplicación que permita buscar productos satelitales utilizando la API del Copérnico, ya sean de tipo Sentinel 1 o Sentinel 2, y poder organizarlos bajo listas o colecciones, donde varios de estos productos comparten alguna característica de interés por el usuario/a.

Una vez los productos estén organizados en listas, se podrá descargar dichos productos, ya sea la lista completa o una serie de productos en específico. Y terminados de descargar estos productos, se puede tener dos variantes respecto a su procesado. Uno de ellos es el procesado por defecto, donde no se tiene que introducir ni especificar ningún parámetro en las diferentes operaciones que se realizan. El otro es un nivel un más avanzado que el anterior, donde se puede modificar dichos parámetros para obtener los resultados tal y como desee.

Sin embargo, durante el análisis, no se conocían todas las características que iba a tener la aplicación desde el principio, así que, aprovechando esta separación en módulos, se realizó un desarrollo basado en prototipos, donde, en cada iteración del desarrollo, se añaden las funcionalidades a éste, primero, para probar dicha funcionalidad, y segundo, para identificar nuevas.

10.1 Características principales

Las características más importantes de este proyecto son:

- Aplicación todo en uno.
- Obtención de imágenes.
- Organización de imágenes.
- Descarga de imágenes.
- Procesado de imágenes.
- Ejecución de algoritmos sobre imágenes procesadas.

10.2 Requisitos

Tal y como se ha comentado anteriormente, a continuación, se establecen los requisitos iniciales que debe cumplir la aplicación:

10.2.1 Aspectos generales:

1. El sistema será capaz de conectarse a la API del Copernicus Open Access Hub mediante el nombre de usuario/a y contraseña registrada en este servicio.
2. El sistema será capaz de obtener información sobre productos satelitales a partir de la API del Copernicus Open Access Hub.
3. El sistema será capaz de descargar los productos satelitales a partir de la API del Copernicus Open Access Hub.
4. El sistema utilizará como credenciales de la aplicación las utilizadas para acceder a la API del Copernicus Open Access Hub.

10.2.2 Buscador:

1. El sistema deberá poder dejar al usuario/a buscar productos satelitales por nombre para obtener aquellos cuyo nombre sea el mismo que el especificado por el usuario/a.
2. El sistema deberá poder dejar al usuario/a buscar productos satelitales por fecha de creación para obtener aquellos cuya fecha de creación del producto sea la misma que la especificada por el usuario/a.
3. El sistema deberá poder dejar al usuario/a buscar productos satelitales en un rango fecha de creación para obtener aquellos productos cuya fecha de creación este comprendida entre el rango especificado por el usuario/a.
4. El sistema deberá poder dejar al usuario/a buscar productos satelitales por tipo satélite para obtener aquellos cuyo tipo de satélite sea el mismo que el especificado por el usuario/a.
5. El sistema deberá poder dejar al usuario/a buscar productos de tipo SAR por tipo de polarización para obtener aquellos cuya polarización sea la misma que la especificada por el usuario/a.
6. El sistema deberá poder dejar al usuario/a buscar productos satelitales mediante coordenadas terrestres para obtener aquellos productos cuyas coordenadas estén dentro o sean parte de las especificadas por el usuario/a.
7. Los filtros serán: buscar por nombre, fecha, rango de fechas, instrumento, polarización y satélite
8. El sistema deberá poder dejar al usuario/a buscar productos satelitales mediante filtros para obtener aquellos cuyo que cumplan con los requisitos especificados por el usuario/a.
9. El sistema deberá poder mostrar y listar los resultados de las búsquedas.
10. El sistema deberá poder dejar al usuario/a acceder a la información de un producto que está en los resultados de una búsqueda.

10.2.3 Listas de productos

1. El sistema deberá poder almacenar la información de un producto (nombre, ID, tamaño en bytes, satélite, instrumento, fecha de creación y fecha inicial y final del tiempo de adquisición.
2. El sistema deberá poder almacenar la información de un producto de tipo SAR (nombre, ID, tamaño en bytes, satélite, instrumento, fecha de creación y fecha inicial y final del tiempo de adquisición.
3. Una lista de productos tiene como campos un nombre y una descripción.
4. El sistema deberá poder dejar al usuario/a crear una lista de productos para organizar los productos que tengan una o varias características en común según el interés del usuario/a.
5. El sistema deberá poder dejar al usuario/a poder eliminar una lista de productos.
6. El sistema deberá poder dejar al usuario/a modificar el nombre y la descripción de una lista de productos.
7. El sistema deberá poder dejar al usuario/a añadir restricciones de tipo de plataforma y producto al crear una lista de productos
8. El sistema deberá poder dejar al usuario/a eliminar restricciones de tipo de plataforma y producto al crear una lista de productos
9. El sistema deberá mostrar un listado de todas las listas del usuario/a al iniciar sesión
10. El sistema deberá poder dejar al usuario/a añadir un producto a una lista de productos para organizar los productos que tengan una o varias características en común según el interés del usuario/a.
11. El sistema deberá poder dejar al usuario/a eliminar un producto a una lista de productos.
12. El sistema deberá poder mostrar y listar los productos de una lista de productos para poder conocer los productos asociados a esa lista.
13. El sistema deberá poder dejar al usuario/a acceder a la información de un producto que está dentro de una lista de productos.
14. El sistema deberá mostrar las áreas de los productos contenidos en una lista en un mapa
15. El sistema deberá poder dejar al usuario/a eliminar un área de producto
16. El sistema deberá poder dejar al usuario/a añadir una imagen de referencia
17. El sistema deberá poder dejar al usuario/a eliminar una imagen de referencia

10.2.4 Descargas

1. El sistema deberá poder dejar al usuario/a descargar un producto para almacenarlo de forma local en su computadora.
2. El sistema deberá poder dejar al usuario/a descargar una lista de productos para almacenarlos de forma local en su computadora.
3. El sistema deberá poder dejar al usuario/a elegir la carpeta donde se almacenarán todos los productos descargados.
4. El sistema deberá poder dejar al usuario/a elegir la forma de descarga paralela (hasta un máximo de 2 productos a la vez) para agilizar el proceso de descarga.
5. El sistema deberá poder dejar al usuario/a elegir la forma de descarga secuencial (un producto a la vez) para agilizar el proceso de descarga.
6. El sistema deberá poder descargar los productos en segundo plano para que el usuario/a pueda seguir utilizando la aplicación.
7. El sistema deberá poder mostrar al usuario/a el progreso de la descarga para saber el tiempo en el que se tardará en descargar los productos o lista de productos seleccionados por el usuario/a.

8. El sistema deberá poder mostrar al usuario/a si una lista o un producto ya está descargado en su ordenador.

10.2.5 Procesado

1. El sistema deberá poder dejar al usuario/a procesar un producto de tipo SAR para poder visualizarlo.
2. El sistema deberá poder dejar al usuario/a procesar un producto de tipo Óptico para poder visualizarlo en formato TIFF.
3. El sistema deberá poder dejar al usuario/a procesar una lista de productos de tipo ópticos para poder visualizarlo en formado TIFF.
4. El sistema deberá poder dejar al usuario/a procesar una lista de productos de tipo SAR para poder visualizarlo.
5. El sistema deberá poder procesar y empaquetar todos los productos de una lista y almacenarlos en el computador del usuario/a.
6. El sistema deberá procesar los productos en segundo plano para que el usuario/a pueda seguir utilizando la aplicación.

11 Diseño

11.1 Casos de uso

11.1.1 Buscador

Siguiendo con esta estructuración del proyecto, se dividió los casos de uso entre los diferentes módulos que componen la aplicación.

En el primer módulo se encuentra el buscador, que permite al usuario/a utilizar la API del Copérnico y buscar los productos por una serie de parámetros. Dicho buscador tiene que dar las opciones al usuario/a para cumplir con los requisitos previamente especificados. Entre los casos de uso más importantes se encuentran los de buscar por filtros complejos, y los de añadir productos a colecciones (ver ilustración 25).

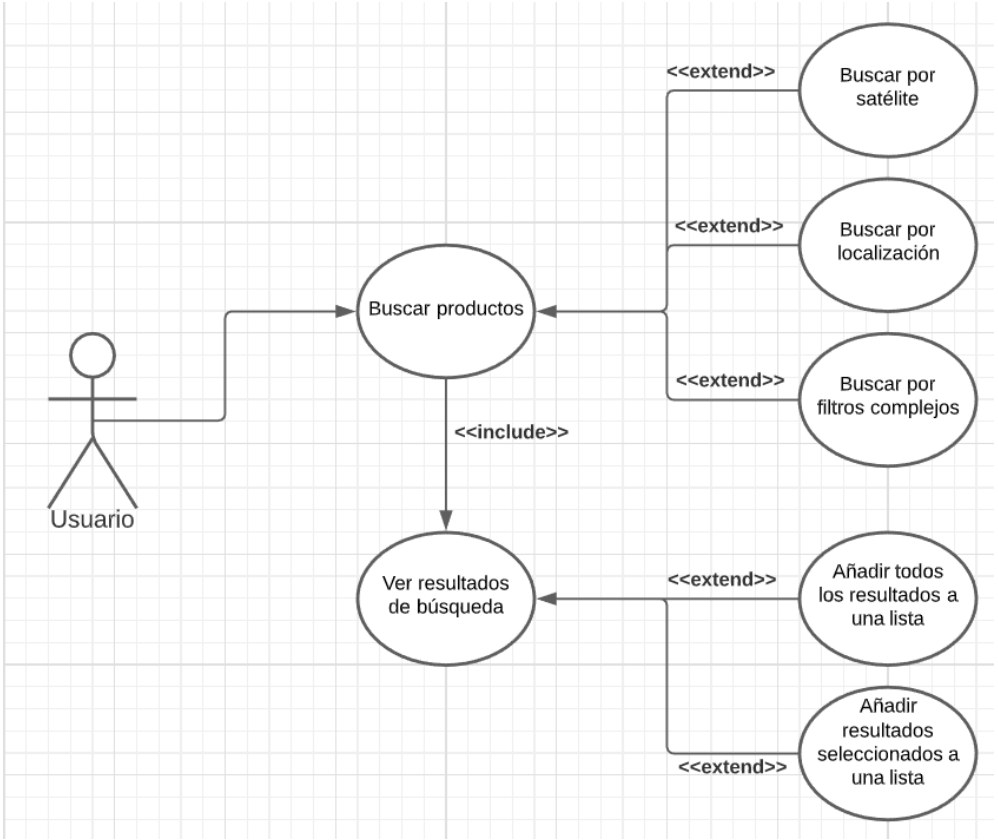


Ilustración 24 - Caso de uso para buscar productos

11.1.2 Listas de productos

El siguiente módulo es el de listas de productos. Aquí reside una parte importante de la aplicación. Este módulo tiene que ser capaz de almacenar y gestionar todos los productos que el usuario/a vaya especificando en él, además de dar soporte a una serie de operaciones, como por ejemplo la de añadir áreas de trabajo o imágenes de referencia (cuando el usuario/a lo requiera). Este diagrama ha sido objeto de varias modificaciones, puesto que al principio solo estaba planeado como un contenedor propiamente dicho, sin ninguna operación a realizar.

Pero una vez se empezó a desarrollar esta parte, se comprendió que era necesario una serie de operaciones precisamente para permitir al usuario/a un correcto uso de los productos. Las áreas de trabajo, que se explicarán con más detalles en la parte de Desarrollo, son unas zonas de interés marcadas por el usuario/a dentro de uno o varios productos satelitales. Su objetivo es reducir el área de trabajo de una imagen, puesto que sus dimensiones pueden ser inabarcables para el módulo de procesado. En cuanto a las imágenes de referencia, son productos Sentinel 2 que se utilizan para obtener una imagen contraste en aquellas listas que tengan productos RADAR.

Otro aspecto que se añadió durante el desarrollo de este apartado fueron las restricciones a las listas, pudiendo añadir una restricción por tipo de satélite, o por tipo de producto. Así, cuando el usuario/a vaya a añadir un producto, la colección verificará dichas restricciones. Si el producto las cumple, se añadirá a la colección.

Así pues, en la siguiente figura se representan todos estos aspectos de las listas, o colecciones, de productos. Además, cuenta con operaciones básicas como crear o eliminar una colección de listas (ver ilustración 25).

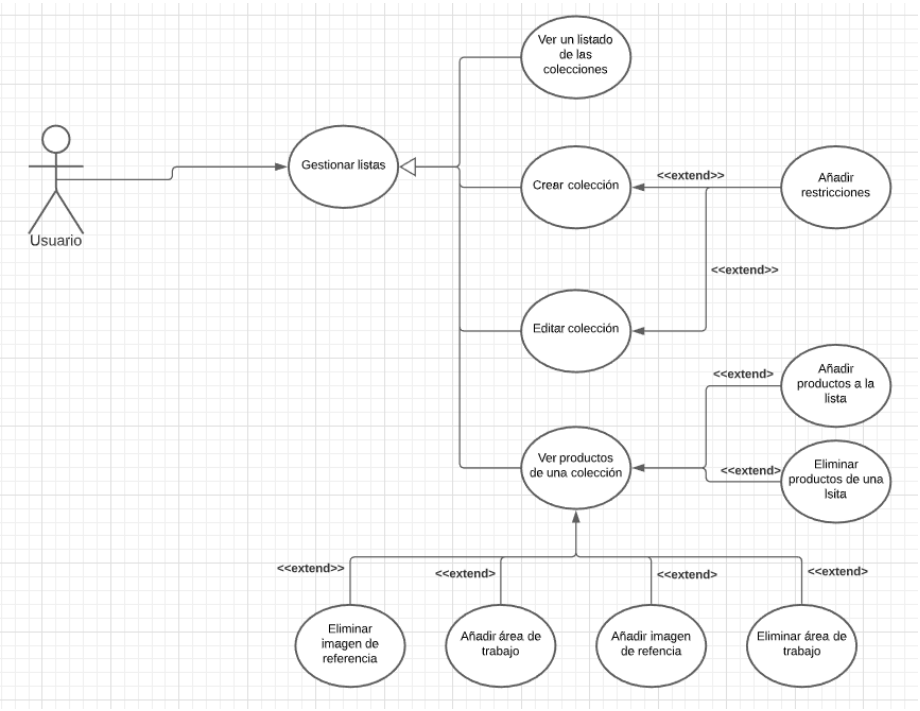


Ilustración 25 - Casos de uso para la organización de los productos en listas

11.1.3 Descargas

Este es el módulo más pequeño a priori de todos los del proyecto, pues se basa únicamente en obtener los productos de una lista, realizar una serie de comprobaciones y finalmente obtener el recurso en la API del Copérnico. Sin embargo, durante su desarrollo se encontraron numerosos problemas a la hora de implementarlos, principalmente por el hecho de tener un máximo número de productos descargándose concurrentemente. Actualmente ese límite es 2 para un usuario/a único en la aplicación.

Además, se han añadido casos de uso para poder manejar estas descargas, dando opciones como, por ejemplo, pausar una descarga, o reanudar una descarga, cancelar todas las descargas, o solo la de un producto en específico, vaciar cola, etc.

También se ha implementado la opción de modificar el directorio donde se van a almacenar los productos descargados. Con todo esto, el diagrama del módulo de descargas es el mostrado en la ilustración 26:

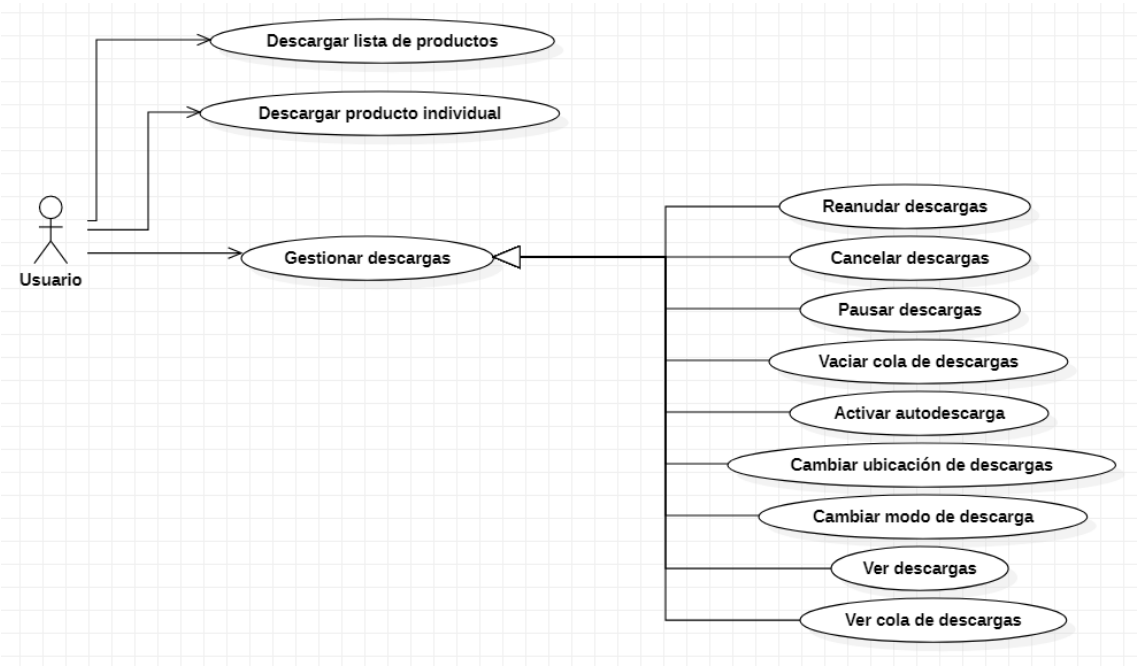


Ilustración 26 - Casos de uso para descargar listas y productos

11.1.4 Procesado

Y como último módulo principal, el procesado de imágenes. Posiblemente el más importante de todos. Su objetivo es obtener los productos descargados, procesarlos y guardar los archivos resultantes en otro directorio.

En un principio se quería implementar este procesado por métodos propios, pero durante el proceso de análisis del proyecto se comprobó que era inviable diseñar los algoritmos necesarios para "revelar" un producto de un tipo en concreto (que hay varios dentro de la familia de Sentinel 1 y Sentinel 2). Así pues, la opción fue aprovechar los recursos que existían en el momento, encontrando que el programa SNAP ofrecía su código fuente de manera gratuita. Así pues, se utilizó dicha librería para el procesado de las imágenes, reduciendo el número de operaciones que teníamos que realizar. El único problema de dicha librería, y se detallará en el capítulo del procesado, es que existen muy pocos ejemplos de uso de la misma. Así que ha requerido un tiempo de investigación bastante elevado, no solo ya por entender las diferentes operaciones que se le puede aplicar a un producto, sino también sobre cómo aplicarlas y manejarlas, y qué librerías se han de usar para obtener una imagen procesada.

Finalmente, el diagrama de casos de uso del procesado es el mostrado en la ilustración 27:

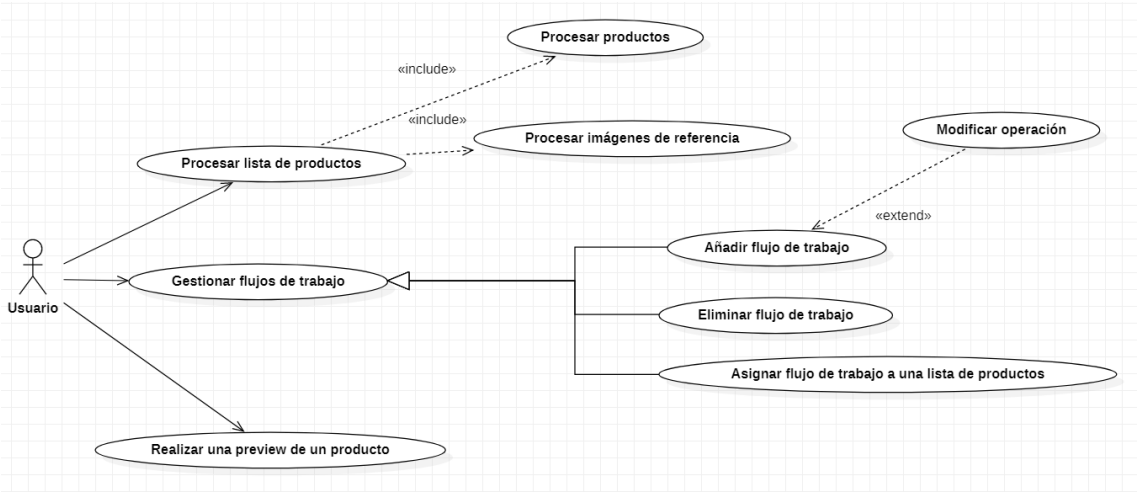


Ilustración 27 - Casos de uso para procesar productos

11.2 Prototipo de la interfaz

Desde el principio se identificó una serie de componentes que eran necesarios añadir a la interfaz. El primero es una barra de menú, con acceso a todas las operaciones que permite la aplicación. Luego una barra de herramientas, con las opciones claves de la interfaz, permitiendo un acceso rápido a las mismas. Un listado de las colecciones de productos del usuario/a, mostrando las listas y sus productos. Una consola para visualizar información sobre la aplicación, un panel de descargas y de procesado; y finalmente, un panel de pestañas donde se mostraría todas las vistas de la aplicación, buscador, detalles de productos, listas y productos, etc. (ver ilustración 28)

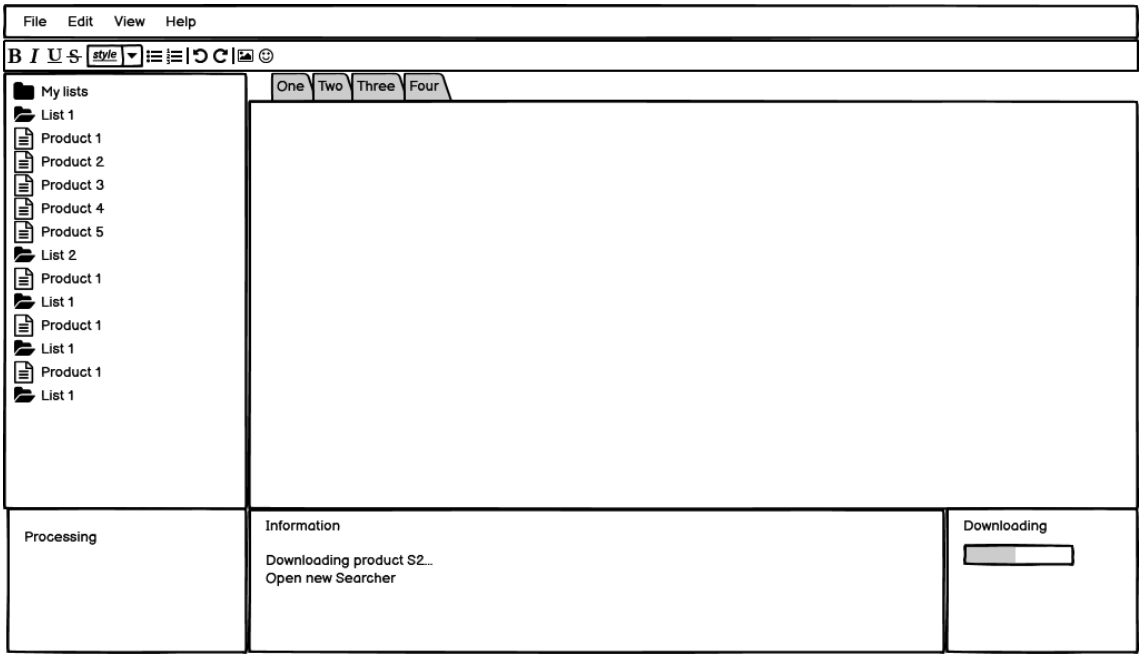


Ilustración 28 - Mockup de la aplicación

Su ubicación es la siguiente: en la parte superior se encontraría la barra del menú y la barra de herramientas; a la izquierda el listado de listas y en la parte inferior el panel de procesado. En el medio tendríamos el panel de pestañas, y en la parte superior, de izquierda a derecha, la consola y el panel de descargas.

11.3 Desarrollo basado en prototipos

Para el desarrollo de esta aplicación se ha tomado un enfoque basado en prototipos. Por una parte, debido a que no estaban claros todos los requisitos del sistema al principio del desarrollo, y otro que, aprovechando que se había identificado cuatro módulos clave en la aplicación, dividir cada uno de ellos en un prototipo para identificar sus requisitos. Por lo tanto, se ha construido un prototipo evolutivo con cada uno de ellos.

Este enfoque da la posibilidad de identificar los requisitos al comienzo de cada iteración, y también durante el desarrollo del mismo. Además, la comunicación aquí con los usuarios finales (en este caso, con los tutores) es clave para poder identificar dichos requisitos y obtener información sobre las funcionales implementadas a la hora de desarrollar este enfoque de manera satisfactoria.

En un prototipo evolutivo, se va añadiendo en cada iteración nuevas funcionalidades al prototipo inicial hasta que es aprobado por el usuario/a. En este caso, además, se ha hecho que dicho prototipo al final sea la aplicación ya desarrollada.

Existen otros tipos de desarrollo por prototipos, como, por ejemplo, el prototipo incremental, que se basa en crear varios prototipos de la aplicación, y luego unirlos todos en uno al final del desarrollo. Este tipo de prototipado no se ajusta al proyecto debido a que en cada iteración tendríamos que comenzar con un nuevo prototipo desde cero, lo cual suponía un esfuerzo superior a los que ofrece el prototipado evolutivo.

11.4 Modelo-Vista-Controlador

Mediante las diferentes iteraciones del desarrollo, se ha utilizado este patrón de diseño para organizar las funcionalidades de la aplicación. Por un lado, tenemos el modelo, que son las clases que representan los datos de la aplicación (información de las imágenes, datos del usuario, etc.). Las vistas son aquellas que permiten mostrar dicha información al usuario/a, y el controlador hace de intermediario para permitir la interacción con los datos a partir de la vista [20]. Sobre estas divisiones del proyecto se hablará más adelante, en el capítulo de desarrollo.

12 Desarrollo

12.1 Metodología

Para el desarrollo de esta aplicación no se ha utilizado ninguna metodología en concreto, sino más bien hemos adecuado los conceptos de las metodologías ágiles a nuestro propio desarrollo.

Utilizando el manifiesto ágil, algunas de las características clave que se han implementado durante el desarrollo son, como, por ejemplo, la comunicación, entregas constantes, y aceptar cambios en todas las etapas del desarrollo.

"*Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente*" -Manifiesto Ágil [21].

Al final de cada entrega, se ha realizado una reunión con los tutores, que han actuado como clientes del proyecto. Se ha promovido la comunicación constante con los tutores en todo momento, ya sea para mostrar funcionalidades, o incluso para transmitir dudas y formas sobre el trabajo que se estaba realizando en ese momento.

También se creó una serie de *sprints* para elaborar las funcionalidades previamente descritas, mediante historias de usuario. En total se ha necesitado de 6 *sprints* para el desarrollo de esta aplicación.

12.2 Planificación

12.2.1 Planificación inicial

En la Ilustración 29 se muestra la planificación inicial especificada en el TFT01 del proyecto. La planificación final se ha ajustado en cierta manera a la planificación inicial, pero se ha necesitado de más tiempo para investigar la API del Copérnico, y, sobre todo, algo que no estaba contemplado al principio, la investigación de las librerías de procesado de imágenes satelitales.

En cuanto a la parte de diseño y desarrollo, se ha ajusta correctamente a la planificación final, ya que se ha implementado todas las tareas especificadas con una duración estimada muy parecida a la duración final de la misma.

Fases	***Duración Estimada (horas)***	***Tareas*** ***(nombre y descripción, obligatorio al menos una por fase)***
Estudio previo / Análisis	25	Tarea 1.1: Estudio de la plataforma Copernicus Open Access Hub
		Tarea 1.2: Estudio del servicio web que ofrece la plataforma Copernicus Open Access Hub mediante el uso de pequeños prototipos.
		Tarea 1.3: Estudio y análisis de las necesidades del usuario.
Diseño / Desarrollo / Implementación	225	Tarea 2.1: Diseño y desarrollo del cliente que consuma el servicio web
		Tarea 2.2: Diseño de la base de datos para almacenar información sobre las imágenes satelitales
		Tarea 2.4: Desarrollo de la interfaz de usuario
		Tarea 2.5: Desarrollo de operaciones a ejecutar sobre las imágenes (búsqueda, filtrado, listas personalizadas, comparaciones, etc).
		Tarea 2.6: (Si queda tiempo). Implementación de procesado de imágenes utilizando Deep Learning
Evaluación / Validación / Prueba	25	Tarea 3.1: Análisis y verificación que todas las funcionalidades y partes de la aplicación funcionan adecuadamente.
		Tarea 3.2: Validación con posibles usuarios finales que la aplicación cumple con los requisitos establecidos
Documentación / Presentación	25	Tarea 4.1: Elaboración de la memoria
		Tarea 4.2: Preparación y desarrollo de la presentación

Ilustración 29 - Planificación inicial

A continuación, se muestra la planificación del desarrollo de la aplicación, divida en *sprints* e indicando las tareas de cada una de ellas. Se ha dedicado un *sprint* por cada módulo de la aplicación, y otros dos más al principio y al final para tareas de investigación y corrección de errores.

12.2.2 Planificación final:

Sprint	**Tareas iniciales**
Sprint 0	-Investigación de la API -Creación de una interfaz prototipo -Pequeños scripts para obtener los primeros datos
Sprint 1	-Creación del módulo de obtención de datos de la API -Modelado de los productos de la API -Obtención de características de interés -Creación de la vista del buscador
Sprint 2	-Creación de la base de datos -Repositorios en el proyecto
Sprint 3	-Desarrollo del módulo de descargas
Sprint 4	-Ejemplos de procesado -Elección de operaciones para cada producto -Implementación de las librerías de procesado -Crear vistas
Sprint 5	-Corrección de errores y retoques finales

12.3 Desarrollo de la aplicación

12.3.1 Sprint 0

Este sprint fue dedicado al estudio de los aspectos descritos en el apartado de "Estado del arte" de este documento, y, además, dando como prioridad la investigación de los recursos que ofrece el programa Copérnico de la Agencia Espacial Europea. Como primer apartado, una vez decidido los productos con los que se iba a trabajar (Sentinel 1 y el Sentinel 2), se realizó un estudio para identificar cuál era la API que se iba a utilizar para el buscador de la aplicación. Ambas APIs, OData y OpenSearch, devolvían el contenido de la petición en formato tanto en XML como en JSON.

El formato XML es un documento basado en etiquetas, donde cada valor se almacena utilizando este sistema.

<name>John</name>

El formato JSON son datos almacenados utilizando pares clave-valor, donde cada dato está relacionado con una clave.

{ "name" : "John" }

En comparación con XML, la clave de JSON sería el nombre de la etiqueta. El formato elegido para la aplicación fue en JSON, debido a la gran cantidad de librerías que existen para que su control en Java resulte sencillo, con un nivel de abstracción elevado.

12.3.1.1 Probando las API

Un ejemplo para acceder a un producto individual utilizando la API de OData sería el siguiente, especificando el ID del producto dentro de la colección de "Products".

ODataAPI/Products('2b17b57d-fff4-4645-b539-91f305c27c69')

Otro ejemplo, pero utilizando filtros sería el siguientes:

ODataAPI/Products?$filter=substringof("SLC",Name)

Todas las búsquedas se realizan dentro de la entidad Products, tal y como se ha explicado anteriormente. Para especificar los filtros, primero se identifica dicho parámetro con un símbolo $ al principio, seguido de todos los filtros que quiera el usuario/a. En el ejemplo se busca aquellos productos cuyo nombre contenga una porción de la cadena pasada por parámetro, en este caso "SLC".

Existen una gran cantidad de filtros dependiendo de los campos que el usuario/a quiera restringir. Para campos de fecha existen filtros de comparación de fecha, y operadores de mayor, menor o igual.

En el siguiente ejemplo, se devuelve una colección de productos que se han publicado en la API entre las 00:00 de la noche y las 12:00.

ODataAPI/Products?$filter=hour(IngestionDate) lt 12

Además, permite la construcción de filtro más complejos, anidando los diferentes parámetros en la ruta de la petición. Y también, existían librerías que mejoraban el uso de dicho protocolo en Java.

12.3.1.2 OpenSearch

A diferencia de la API del OData, aquí se utiliza un modelo de parámetros agrupadas en pares clave valor, separados por operadores AND y OR. Un ejemplo sería el siguiente, donde especificamos que la plataforma sea “Sentinel-1” y el tipo de producto “GRD”.

(plataformname : Sentinel-1 AND producttype : GRD)

Utilizando los parámetros anteriores, un ejemplo de petición a la API sería el siguiente, especificando los parámetros de paginación:

OpenSearchAPI/search?start=0&rows=100&q=(platformname:Sentinel-1%20AND%20producttype:GRD)

El contenido de la respuesta es un conjunto de entradas, es decir, los productos, junto a una cabecera que indica los parámetros utilizados para la paginación. En el siguiente capítulo se detallará más en profundidad el contenido de la respuesta, así como los diferentes elementos que la forman.

12.3.1.3 Elección de la API

Primero, se elaboró unos pequeños prototipos que simulaban la obtención de los datos a través de la API de OData. Sin embargo, dicha API presentaba un problema, más bien, una funcionalidad no implementada, que era la de buscar productos por geolocalización. Dicha funcionalidad sólo estaba presente en la API de OpenSearch, así que esta fue la “elección” final.

El inconveniente de esto es que este no cuenta con los servicios para descargar productos, así que al final utilizaremos ambos para el desarrollo de la aplicación. Una vez terminado con la investigación, y con la elección de la API y los productos a trabajar, se pasó a la siguiente fase del desarrollo, el buscador.

12.3.2 Sprint 1 - Buscador

La primera parte a desarrollar fue la creación de un módulo que sea capaz de obtener la respuesta de la API del *OpenSearch*, y pasarlas a objetos Java. Dentro de esta respuesta hay información referente al número de elementos que coinciden con los criterios de búsqueda, enlaces a otras peticiones para obtener los datos de los productos de la próxima página (*rows* y *page*). Después se encuentran los datos de los productos, denominados "*entry*".

La respuesta tiene un formado dividido en varias partes. Primero encontramos la raíz, que en este caso es "*feed*", que contiene todos los datos. Este objeto raíz cuenta con varios nodos hijo, correspondientes a información de la petición, como, por ejemplo, fecha en la que se realizó la petición, resumen, enlaces relacionados con la paginación de los productos, número de productos totales en la API que han coincidido con los parámetros de búsqueda, etc.

Después de estos se encuentran los nodos "*entry*", que cada uno se corresponde a un producto. Dentro de cada uno de ellos se encuentra la información referente a uno de los productos que ofrece el Copérnico: productos SAR (Sentinel-1 y sus diferentes tipos, RAW, GRD, SLC…), productos del Sentinel 2, y por último productos oceánicos (Sentinel-3), que no trabajaremos en este proyecto.

Existen una serie de atributos que se repiten en todos los productos, independientemente de la plataforma o el subtipo, y son, por ejemplo, el tipo de plataforma (Sentinel 1, 2...), tipo de producto (GRD, SLC, S2MSI1C…), fecha de adquisición, tamaño en disco, etc. Todos estos atributos se identificaron como claves para almacenar.

Ahora bien, también existen parámetros exclusivos para cada uno de los productos o subproductos. Por ejemplo, el Sentinel 1 cuenta con parámetros que indican el modo de polarización y el modo del sensor; para el Sentinel 2, parámetros relacionados con el porcentaje de nubosidad de la imagen, de agua, etc. Todos estos datos se almacenarán en la aplicación

Realizar esta traducción de JSON a Java sin ningún framework es un trabajo excesivo y poco eficiente. Así pues, se eligió la librería Jackson [22], que ofrecía esta capacidad de una manera rápida y fluida. Sin embargo, en la respuesta de la petición, los datos están agrupados de una manera peculiar, utilizando el tipo de dato para dividirlos. Esto por un lado es positivo, puesto que da a conocer de qué tipo es el dato que está devolviendo la API, pero, a la hora de obtenerlo y pasarlo a Java no es una operación inmediata, puesto que dichos valores, Jackson, los almacena en contenedores. Así, se encuentran contenedores de tipo String, Date, Double, Int, etc. Por ejemplo, el atributo de título del producto estaría en el contenedor de String.

Esto supuso la creación de una serie de *deseralizadores* (objetos para pasar de JSON a objetos Java) para obtener los datos almacenados dentro de estas colecciones, seleccionar aquellos necesarios según el tipo de producto, y finalmente, obtener un objeto Java (ver ilustración 30).

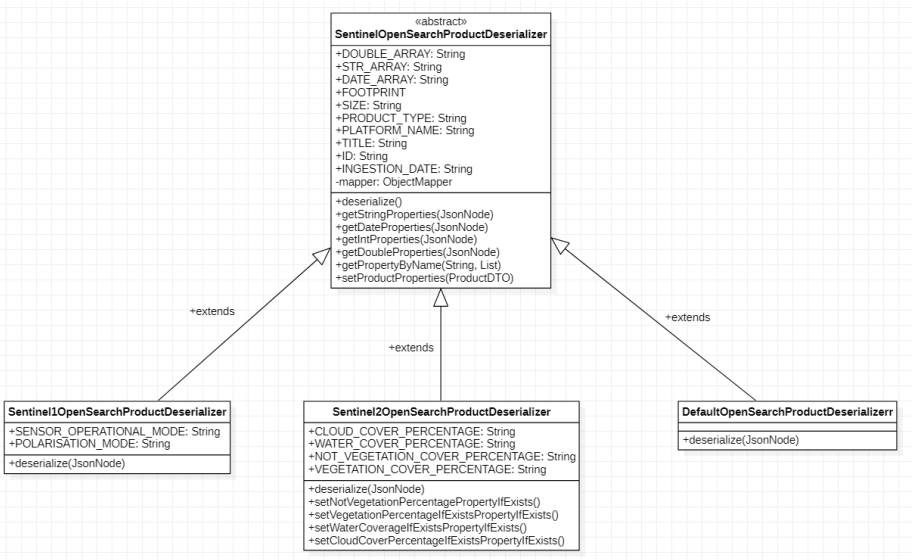


Ilustración 30 - Deserializadores de la API de OpenSearch

En primer lugar, está la clase *SentinelOpenSearchProductDeserializer*, que es una clase abstracta, y que cuenta con métodos para obtener los diferentes contenedores agrupados por tipo, como se ha comentado anteriormente, y, además, un método para almacenar las características comunes a todos los productos.

Las subclases que la extienden implementan el método deserializer(), que obtiene los datos únicos de un tipo de producto, y los almacena en la clase respondiente. Finalmente, se llama de nuevo a este *SentinelOpenSearchProductDeserializer* para obtener los datos comunes a todos los productos (título, identificador del producto, fecha de adquisición, tamaño en disco, etc.).

Todos estos *deserializadores* son gestionados por una clase superior, llamada *SentinelOpenSearchDeserializerManager*, que contiene un mapa con las diferentes implementaciones para cada tipo de producto. Esta clase es llamada por la librería Jackson para realizar el deserializado, y esta, según el tipo de producto, delega dicha responsabilidad a su clase correspondiente.

A su vez, para cada tipo de producto se ha creado su correspondiente en Java. Existe una clase global, llamada *ProductDTO*, que cuenta con los campos generales de todos los productos. Y luego tenemos una clase por cada tipo de producto, *Sentinel1ProductDTO* y *Sentinel2ProductDTO*. También se cuenta con una clase *SentinelProductDTO*, que engloba a todo el sistema Sentinel (ver ilustración 31). En todas estas clases se utilizan las clases *Property* de JavaFX, que permiten envolver un valor, ya sea string, entero o de otro tipo, en una clase para que, si se modifica, se extienda por toda la aplicación. Este funcionamiento se explicará con más detenimiento en los próximos capítulos

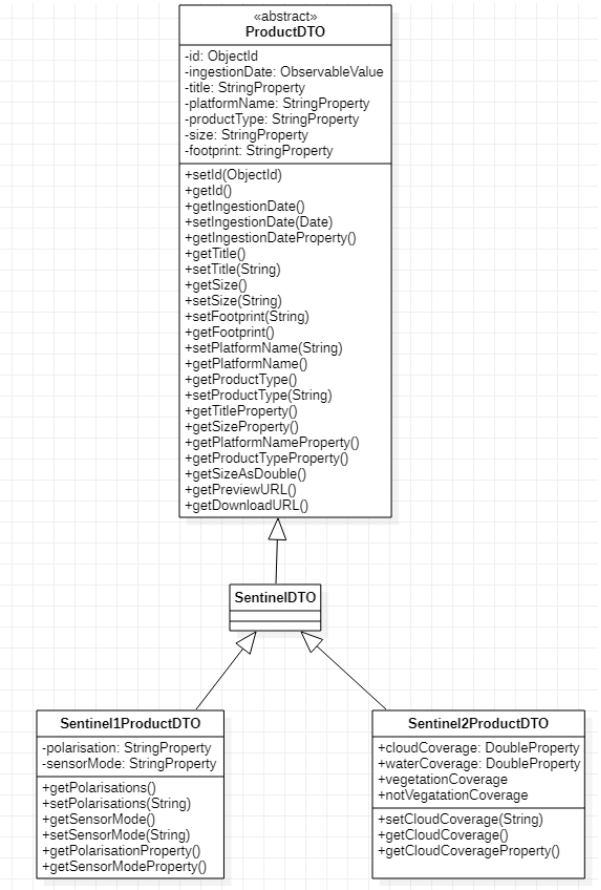


Ilustración 31 - Modelado de los productos

Cuando todos los productos se han deserializado, se devuelve un objeto llamado *OpenSearchResponse*, que contiene una colección con todos los productos y los parámetros para la paginación, número de elementos totales en la API, número de productos por página, etc.

Una vez que se comprueba que el *parseador* funciona correctamente (con la ejecución de una batería de test), el siguiente paso es la autentificación del usuario/a en la API. Para eso se utiliza la clase *HTTPAuthManager*, que cuenta con los métodos para que, a partir de un nombre de usuario/a y una contraseña, cree un *AuthentificationToken*, asignándole las credenciales del usuario/a, y que acompañará a las llamadas HTTP que se realicen a la API. Esta clase también responsable controlar los diferentes códigos de respuesta que devuelve la API, siendo de especial interés aquellos asociado al correcto inicio de sesión del usuario/a. Si las credenciales son incorrectas, lanzará una excepción, siendo tratada por la vista que llama a este servicio (ver ilustración 32)

```
private void isConnectionResponseOK(HttpsURLConnection connection) throws IOException, AuthenticationException {
    int connectionCode = connection.getResponseCode();

    if (connectionCode == HttpURLConnection.HTTP_UNAUTHORIZED) {

        logger.atWarn().log( message: "URL respond with {} code, login error?",connectionCode);
        closeConnection();
        throw new AuthenticationException("Incorrect username or password");

    } else if (connectionCode != HttpURLConnection.HTTP_OK) {

        logger.atWarn().log( message: "URL respond with {} code {}",connectionCode,errors.getOrDefault(connectionCode,
                defaultValue: "Resource not available"));

        if (connection.getInputStream() != null)
            connection.getInputStream().close();

        closeConnection();
        throw new HttpResponseException(connection.getResponseCode(),errors.getOrDefault(connectionCode,
                defaultValue: "Resource not available"));
    }
}
```

Ilustración 32 - Comprobación del código respuesta de la petición

También incluye los métodos para realizar peticiones a una URL y obtener los datos en un objeto *InputStream*, utilizando el método *getContentFromURL*, y con el uso de la librería *HttpsUrlConnection* para realizar las conexiones y utilizar el token descrito previamente.

Para utilizar una misma interfaz para acceder a los servicios HTTP, se desarrolló un servicio llamado *CopernicusService*, el cual cuenta con todos los métodos para realizar peticiones al *HTTPAuthManager*. Y, sobre todo, proporciona el manejo de las credenciales del usuario/a en toda la aplicación. Para conseguir esto, este servicio se convirtió en una clase *Singleton*, es decir, solo puede ser instanciada una vez en toda la aplicación. Cuando se llama, si no está instanciada, muestra un diálogo para obtener las credenciales del usuario/a. Luego, llama al *HTTPAuthManager* para comprobar si dichas credenciales son correctas (ver ilustración 33). Si lo son, se instancia el servicio, y las credenciales son almacenadas en ella para cualquier petición futura. De lo contrario, no se instancia el servicio, lanzado una excepción que indica que las credenciales eran erróneas.

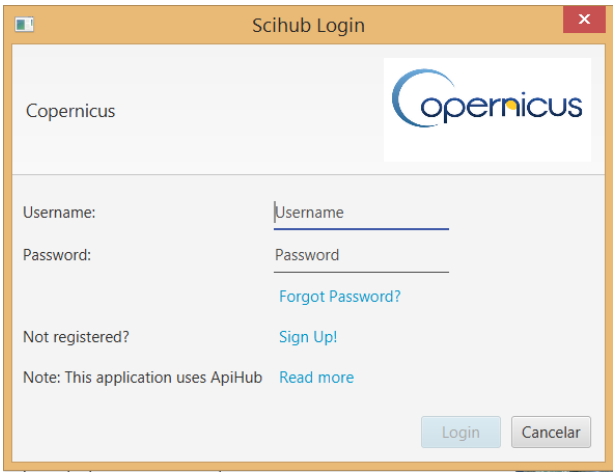


Ilustración 33 - Diálogo para iniciar sesión en la API

12.3.2.1 OpenSearch

Una vez desarrollado el "traductor" de JSON a Java, el módulo para hacer llamadas HTTP, y el servicio para controlar las credenciales, el siguiente paso fue implementar el servicio *OpenSearch*, el cual controla la construcción de la petición a la API. Este servicio cuenta con métodos para añadir parámetros, establecer los datos de paginación, y obtener un mensaje de respuesta (*OpenSearchResponse*).

Esta clase es la responsable de llamar al *CopernicusService* para obtener las credenciales y poder validarse en la API. Si estas son correctas, se realiza la petición. La respuesta se le pasa al *OpenSearchProductDeserializer*, que llama a los deserializadores correspondientes para obtener todos los productos, devolviendo finalmente un objeto *OpenSearchResponse* (ver ilustración 34).

```
@Override
public OpenSearchResponse search() throws IOException, AuthenticationException, NotAuthenticatedException {
    if (service == null) {
        logger.atWarn().log("Not authenticated in Copernicus OpenSearch");
        throw new NotAuthenticatedException("Not authenticated in OpenSearch");
    }
    long start = currentTimeMillis();
    InputStream contentFromURL = service.getContentFromURL(getURL());
    OpenSearchResponse response =
            (OpenSearchResponse) ProductDeserializerFactory.get("OpenSearch").deserialize(contentFromURL);
    contentFromURL.close();
    long finish = currentTimeMillis() - start;
    logger.atInfo().log( message: "{} products loaded in {} seconds",
            response.getProducts().size(), p1: finish/1000.0);

    return response;
}
```

Ilustración 34 - Petición de buscar del servicio de OpenSearch

12.3.2.2 Controladores y vistas

Tras desarrollar estos elementos, se comenzó a implementar la clase que se encarga de ofrecer dichos servicios al usuario/a, por medio de un controlar de vista de JavaFX, *CopernicusOpenSearchController*, con anotaciones y campos vinculados a una vista FXML (por ejemplo, a campos de texto, botones, componentes para el listado de ítems, etc.) (ver ilustración 35).

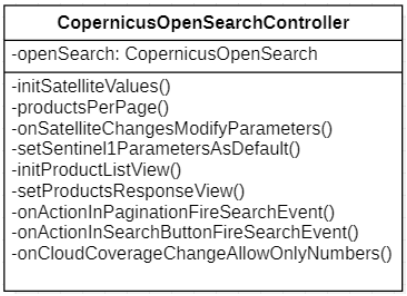


Ilustración 35 - Principales métodos del CopernicusOpenSearchController

Este componente se utiliza para controlar los parámetros de búsqueda (añadir, eliminar, modificar, etc.), manejar eventos de los mismos, realizar la búsqueda, etc. Además, cuenta con otro controlador añadido, llamado *GTMapController*, que gestiona el mapa de búsqueda de áreas.

Esto se ha decidido así para evitar que un solo controlador contenga todo el control de la vista. De esta manera, con el controlador del mapa, se evita tener en la clase principal eventos de Scroll o dragg (moverse por el mapa), que no tienen que ver con las búsquedas en sí.

Este controlador se ocupa de algo más importante, obtener las coordenadas reales que el usuario/a selecciona en el mapa. Para obtener dicha conversión se ha utilizado una librería de georreferencias, llamada *GeoTools*, la cual permite hacer una traducción entre las coordenadas de la escena, y su correspondencia real (ver ilustración 36).

```
public double[] transformSceneToWorldCoordinate(double x, double y) {
    DirectPosition2D sceneCoordinates = new DirectPosition2D(x, y);
    DirectPosition2D worldCoordinates = new DirectPosition2D();
    mapContent.getViewport().getScreenToWorld().transform(sceneCoordinates, worldCoordinates);
    return worldCoordinates.getCoordinate();
}
```

Ilustración 36 - Método para convertir coordenadas de una escena de JavaFX a coordenadas reales

Además, da la posibilidad de customizar el mapa en una serie de capas, donde en cada una de ellas se puede añadir una serie de características, con una forma geométrica concreta, y posteriormente mostrarlos en el mapa. Así, por ejemplo, es posible "pintar" un recuadro encima del mapa, y pasarlo al mismo con una serie de características reales, como sus coordenadas. Para realizar esto, se utiliza el formato **WKT**, *Well Know Text*, una serie de etiquetas para representar formas geométricas y polígonos con coordenadas del mundo real.

Su formato es el siguiente, primero se identifica el tipo de polígono, puede ser un punto, una línea recta, un rectángulo o una serie de vértices para un polígono de mayor complejidad. Después viene seguido por un conjunto de puntos, identificando pares de latitud y longitud.

POLYGON((-15.471987642042164 28.167268689575415,-15.402979768507008 28.167268689575415,-15.402979768507008 28.097330439717194,-15.471987642042164 28.097330439717194,-15.471987642042164 28.167268689575415))

El *GTMapController* trabaja sobre un canvas, que realmente realiza las conversiones, la gestión de las capas, etc. El controlador facilita su uso, con una serie de métodos que permiten añadir y eliminar polígonos, capas, etc., así como otros para mostrar las características asociadas a una capa, seleccionando una de ellas en concreto, etc. (ver ilustración 37).

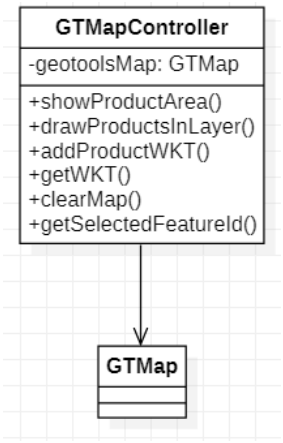


Ilustración 37 - Diagrama de clases del controlador del mapa, GTMapController

Respecto a la vista, se optó por dar protagonismo al mapa, donde el usuario/a introduce el área de la búsqueda y también donde se muestran las localizaciones de los productos resultantes. En

la parte izquierda contamos con el formulario para buscar los productos, mostrando por defecto las características del Sentinel 1. A la derecha, encontramos el mapa, con dos controles en él, uno para borrar la selección actual, y otra restablecer el mapa a su posición inicial (ver ilustración 38).

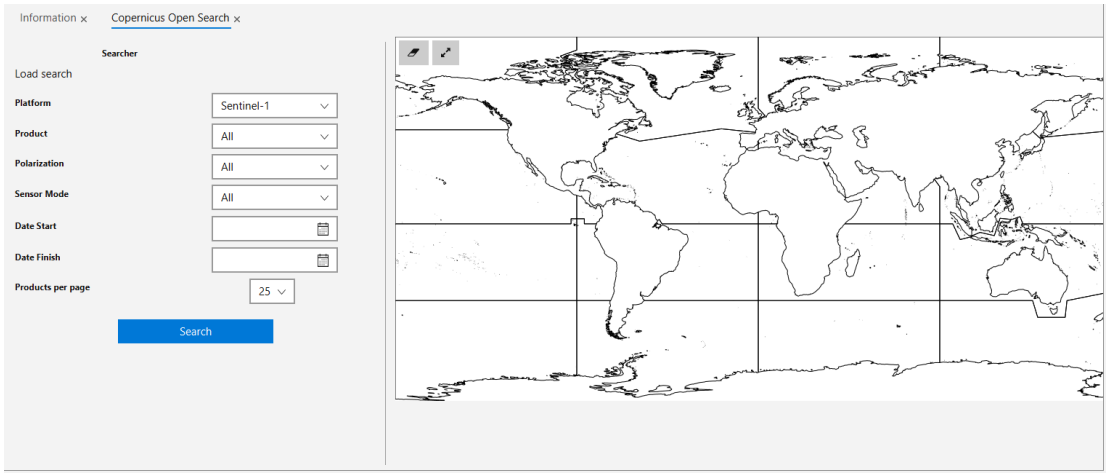


Ilustración 38 - Vista de la búsqueda de productos

En el buscador encontramos opciones para filtrar por plataforma, tipo de producto, polarización y rango de fecha. Además, contamos con un parámetro para limitar el número de resultados máximos que devuelve la API, siendo el máximo 100, y el mínimo 25 (ver ilustración 39)

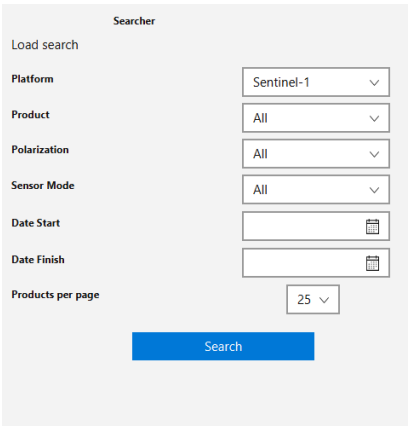


Ilustración 39 - Búsqueda de un producto Sentinel 1

Cuando cambiamos de plataforma, la vista se actualiza para mostrar los parámetros propios de ese producto en completo. Para el Sentinel 2, tenemos parámetros para buscar por porcentaje de nubes en las imágenes, junto con el rango de fechas (ver ilustración 40)

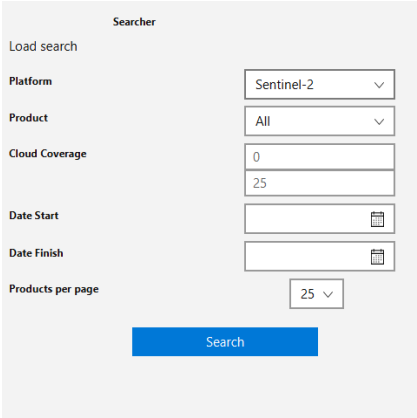


Ilustración 40 - Búsqueda de un producto Sentinel 2

Una vez se establen los criterios de búsqueda, y un área específica (aunque no es obligatoria), hacemos clic en “Buscar”. Dado que la petición tarda en procesarse en los servidores de la ESA, se muestra un mensaje al usuario/a para advertirle que la búsqueda puede tardar unos minutos.

Tras obtener los resultados, se visualizan en la vista. A la izquierda, se muestran todos los productos utilizando el componente ListView de JavaFX, para mostrarlos en forma de lista, junto a su información relativa respecto a su tipo de plataforma, instrumento y tamaño. Para esto, se ha diseñado otra vista (y su respectivo controlador), llamada *ProductSearchListCellView*, y así obtener una vista personalizada del producto dentro del componente de ListView. Con esto mejora la parte visual, y permite añadir nuevas funcionalidades asociadas a cada producto, como, por ejemplo, mostrar los detalles de un producto.

En el mapa aparecen las localizaciones de los productos de la búsqueda, pudiendo interactuar con ellas y seleccionarlas para conocer a qué producto corresponde (ver ilustración 41).

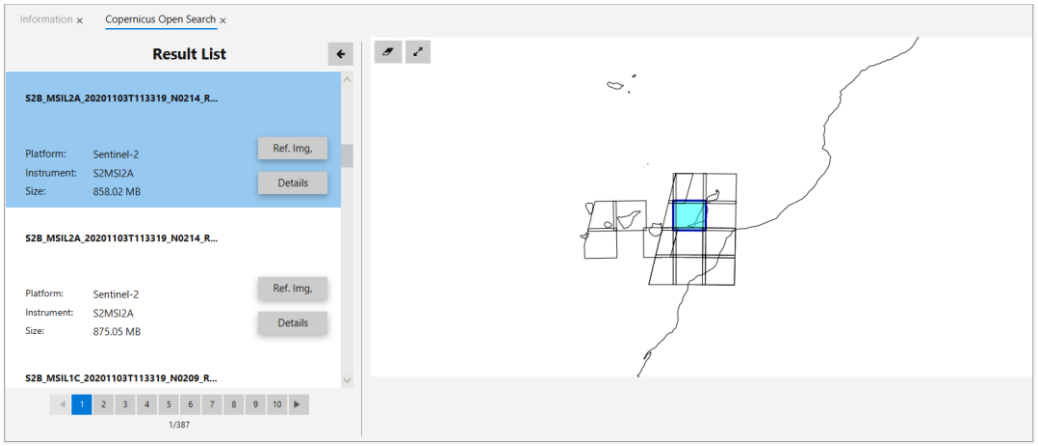


Ilustración 41 - Vista de los resultados de una búsqueda

Para realizar dicha funcionalidad, se utiliza el framework de JavaFX, el cual ofrece una serie de eventos que se llaman cuando algún componte ha cambiado de estado. Esto puede ser, por ejemplo, si un elemento se ha añadido o eliminado de un ListView. También se usan eventos de modificación, especialmente adecuados para los campos de textos. Existen otros tipos para conocer qué elementos de una lista se han seleccionado, obteniendo el nuevo elemento

seleccionado y el anterior. Un ejemplo de estos manejadores sería el que se encarga de que, cuando se selecciona un producto en la lista, se realce su localización en el mapa.

Este método permite la selección ya sea individual o múltiple de varios productos. También se realizan comprobaciones para verificar que hay productos seleccionados en el mapa, y si ya lo están, deseleccionarlos.

Finalmente, el diagrama de secuencias para buscar productos, utilizando toda la estructura de clases descrita anteriormente, sería como la mostrada en la ilustración 42:

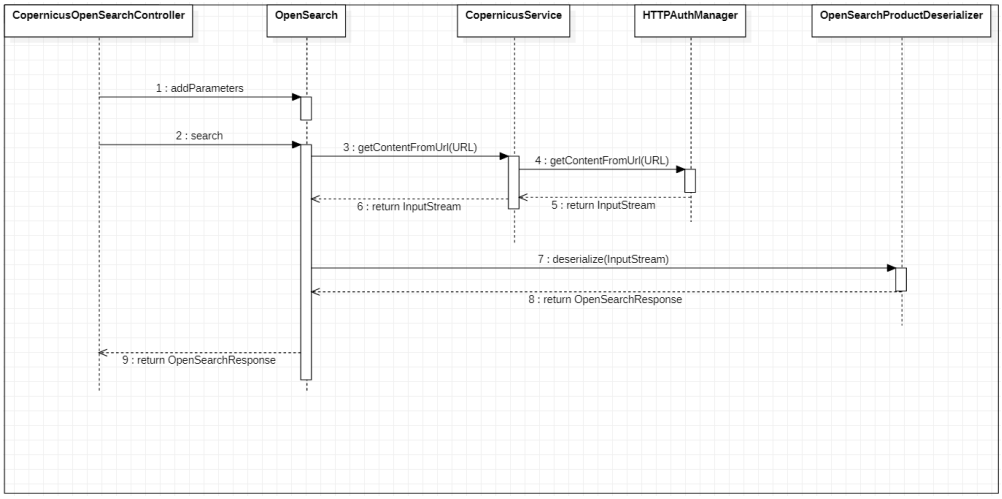


Ilustración 42 - Diagrama de secuencias para buscar productos

La siguiente vista a desarrollar fue la de los detalles de un producto, y su controlador correspondiente. Su objetivo es simplemente mostrar toda la información de un producto, junto a una imagen denominada "*Quicklook*", donde se observa su localización. Para obtener este "*Quicklook"* se ha realizado una llamada a la API para acceder a dicho recurso y mostrarlo (ver ilustración 43). Al no contar con ninguna interacción con el usuario/a, esta vista es solo de información.

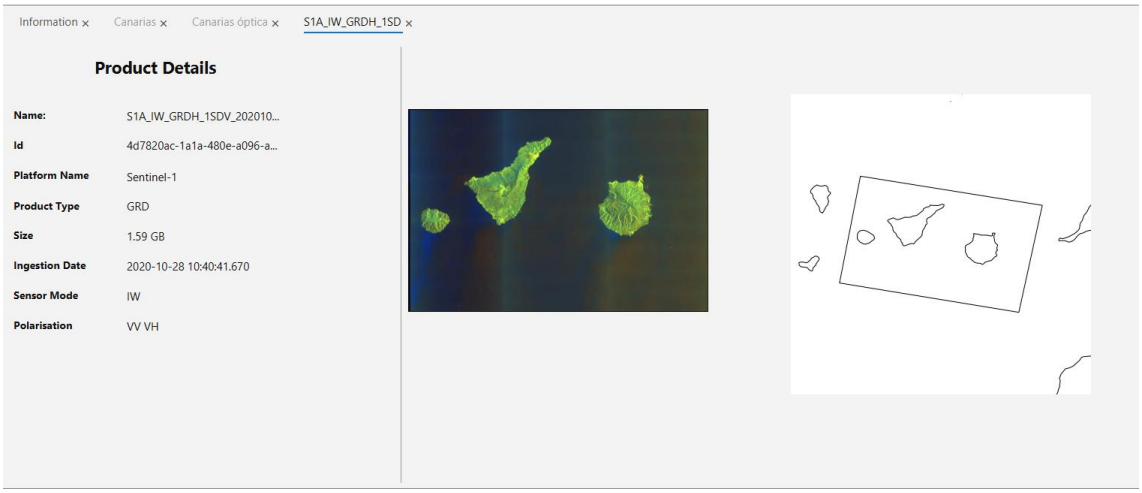


Ilustración 43 - Vista de los detalles de un producto

Sin embargo, estas dos vistas están contenidas en el componente de pestañas, denominado *TabPaneComponent*, descrito brevemente en el diseño de la aplicación. Esta clase se encarga de

gestionar todas las vistas que se muestran en dicho panel, ofreciendo métodos para cargar una nueva pestaña, seleccionar, obtener el controlador de la vista, etc.

Un problema que se observó es que, durante la inicialización de una pestaña, si esta realizaba alguna operación que necesitara de un periodo de tiempo considerable, la interfaz se congelaba. Para evitar este comportamiento se desarrolló un cargador de pestañas, utilizando las ventajas que proporciona FXML. Esto es, cargando la vista primero y luego, en un hilo aparte, el inicio de los servicios ociosos que pudiera tener. Esto se ha realizado mediante las clases Task de JavaFX, que permiten envolver una serie de operaciones y ejecutarlos en otro hilo distinto. Además, ofrecen operaciones para que, si el hilo se ha ejecutado satisfactoria, llamar a una serie de operaciones que se requieran. Existe el mismo funcionamiento si el hilo ha terminado con fallos o ha lanzado alguna excepción.

Un ejemplo claro que utiliza especialmente esta estructura es el buscador del Copérnico. Como se ha explicado con anterioridad, es necesario identificarse para utilizar el servicio y hacer una llamada a la API, comprobando que las credenciales son correctas. Si no utilizáramos un hilo secundario, la interfaz gráfica se congelaría pues esta verificación se estaría haciendo el hilo principal.

Para solucionar esto se necesitó de una interfaz común a todas las pestañas que se iniciaran en la aplicación, llamada *TabItem*. Cuenta con métodos para cargar el FXML de una pestaña, e iniciar los componentes o servicios que puede requerir dicha pestaña.

Todos los controladores de pestañas implementarán esta interfaz, logrando que, independientemente de lo que se cargue, el manejo siempre sea el mismo. Además, aprovechando esto, se desarrolló varios tipos de *TabItem*, como, por ejemplo, el *SearchTabItem*, el cual define que se trata de una pestaña buscador (ver ilustración 44). Por lo tanto, *CopernicusOpenSearchController* implementa esta interfaz.

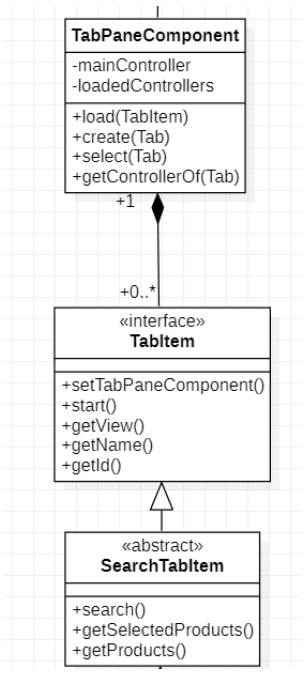


Ilustración 44 - Diagrama de clases del componente TabPaneComponent

El *TabPaneComponent* se encarga de almacenar tanto las vistas como los controladores, por si el usuario/a cierra sus pestañas no sea necesario volver a realizar la carga del componente, sino simplemente mostrarlo de nuevo en la interfaz.

Como último detalle, se han utilizado librerías para estilización de JavaFX, llamadas JMetro [17] y JPhoenix [18]. Éstas realizan un cambio de estilo, sustituyendo la capa base de JavaFX por otra más actual y moderna.

12.3.3 Sprint 2 – Colecciones de productos

Tras concluir el desarrollo del buscador, se comenzó con la búsqueda de una base de datos que alojara la información de los productos y las colecciones. Además, sería necesario del registro del usuario/a para asociar la cuenta a las colecciones.

Dicha búsqueda se basó en decidir qué tipo de base de datos usar, si una tradicional, SQL, o una NoSQL. Sus diferencias residen en el modo de almacenar la información. Mientras una base de datos SQL está estructura en una serie de tablas y filas de datos, y sus correspondientes relaciones (modelo relacional), en una NoSQL tenemos colecciones de datos y documentos no estructurados (modelo no relacional, basado en objetos formato JSON).

Una tabla se corresponde con una colección en NoSQL, y una fila de datos con un documento. Mientras la tabla tiene una serie de campos definidos, en NoSQL carecemos de dicha estructura. (ver ilustración 45). Es decir, no está limitada por ninguna restricción. Se puede almacenar cualquier tipo de dato en un mismo documento, y estos pueden ser diferente unos de otros bajo una misma colección. Esto permite una mayor facilidad a la hora de realizar cambios en la misma, puesto que no hay que modificar nada en la base de datos. Sin embargo, todo tiene un coste, y es que los datos no están sujetos bajo ninguna limitación, y manejarlos cuando hay una gran cantidad de información puede ser bastante complicado.

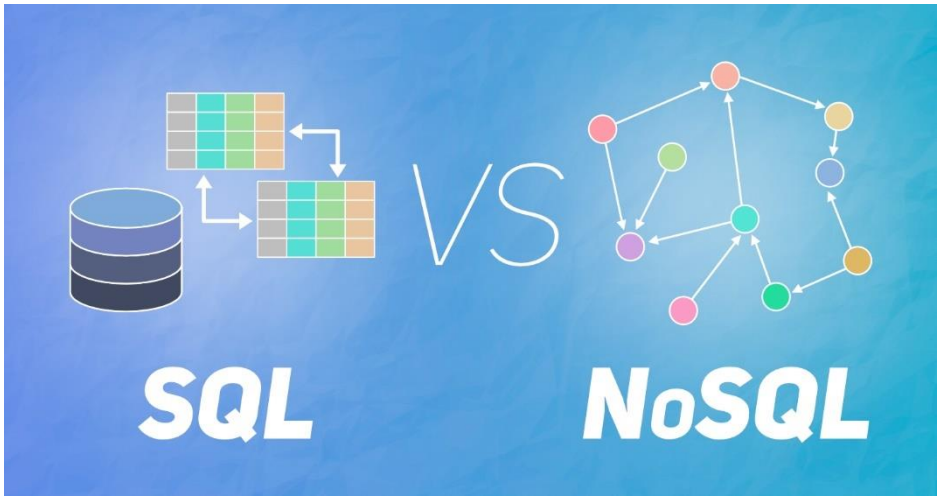


Ilustración 45 - Diferencia entre SQL y NoSQL. Imagen obtenida de https://databaseandtech.wordpress.com/

Para esta aplicación se ha decidido utilizar una base de datos NoSQL, principalmente por los servicios en nube que ofrece, como, por ejemplo, MongoDB. Una de las premisas de la aplicación es que la información sobre imágenes satelitales sea accesible desde cualquier dispositivo donde esté instalado la aplicación, y este tipo de base de datos ofrece esa capacidad, y de una forma rápida y efectiva. Además, permite una mayor flexibilidad a la hora de los datos a almacenar, no teniendo un esquema fijo con el trabajar. Otra de las razones por las que se me ha decantado por este tipo de base de datos es la facilidad para trabajar con otro tipo de aplicaciones con la misma base de datos.

Como framework para trabajar con MongoDB se va a utilizar Morphia. Esta librería permite realizar traducciones de colecciones y documentos de MongoDB a objetos Java, y viceversa. Su funcionamiento se basa en, por medio de unas clases entidades, relacionar un objeto y sus atributos, con una colección determinada. Para esto, utiliza una serie de anotaciones para indicar a qué colección corresponde cada objeto, indicar qué atributos son identificadores, o decidir cuáles se almacenan en la base de datos.

Por lo tanto, para realizar operaciones sobre la base de datos, empleamos unas clases llamadas entidades, que es un modelado de las colecciones, tanto para lectura de datos como para escritura.

Siguiendo el paradigma de las bases de datos NoSQL, las diferentes clases de productos (Sentinel1, Sentinel2...), en vez de estar cada una en tablas distintas, están todas almacenadas en una misma colección, Productos. Esto permite que toda la información de los productos pueda estar en un único documento, sin necesidad de hacer JOIN entre tablas, aumentando la efectividad y rapidez para obtener los datos.

Con el aumento de las diferentes funcionalidades permitidas ya en la aplicación, se dividió el proyecto siguiendo el paradigma del Modelo-Vista-Controlador. Tenemos una serie de directorios, separados por estos principios: el modelo, que son clases que representan la información de la aplicación; controlador, que gestiona la interacción del usuario con el modelo; y vistas, que es la "interfaz" donde se muestra la información del modelo, y que en esta aplicación son los archivos FXML, dentro de la carpeta de recursos.

Dentro del modelo tenemos clases DTO (Data Transfer Objects) para encapsular la información y mostrarlas en las vistas. Para los accesos a la base de datos se ha desarrollado un paquete de servicios, (donde se localiza también el servicio de búsquedas de la aplicación, *OpenSearch*, etc.) dentro de módulos DAO (Data Access Object).

Para separar el modelo de las capas de servicio con las de la vista se desarrollaron los modelos DTO. No tiene sentido pasarle a la base de datos un modelo con métodos que no tienen nada que ver con su almacenamiento en la base de datos. Así que, para separar estas capas, se desarrolló un conjunto de entidades, explicadas anteriormente, y que se corresponden a una clase DTO y a una colección de la base de datos. Por ejemplo, tenemos la entidad *Product*, y el modelo *ProductDTO*.

Para cada una de las colecciones de la base de datos tendremos un servicio DAO (Objeto para la obtención de datos) diferente. Todos los servicios DAO son comunes puesto que implementan una interfaz llamada DAO que especifica las operaciones básicas de este tipo de servicios: getCollection, find, save, remove, etc. Un ejemplo de este servicio es UserDAO, para obtener y registrar a los usuarios dentro de la aplicación (ver ilustración 46).

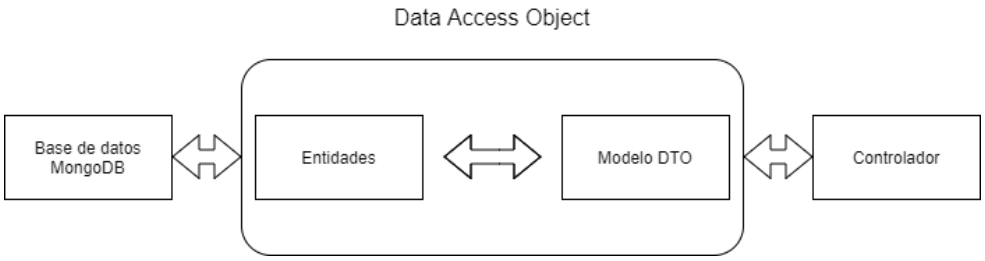


Ilustración 46 - Relación entre Entidades, Modelos DTO y el DAO

Una vez desarrollado estos servicios se implementó el inicio de sesión y el registro del usuario/a en la aplicación. Se desarrolló las vistas y controladores para cada uno de estos componentes. A estos se le pasa un modelo *UserDTO* para almacenar los datos del usuario/a. En el inicio de sesión, obtiene los datos introducidos por el usuario y comprueba si dicho usuario está registrado en la base de datos. Si lo está, inyecta el modelo posteriormente al controlador

principal, para que sea accesible por todos los componentes de la aplicación. Este controlador se encarga de obtener las credenciales del usuario/a, y luego, llamar al módulo de base de datos de Usuarios para comprobar que dichas credenciales se encuentran registradas en el sistema.

Por otra parte, el controlador principal es el que contiene y carga los componentes gráficos de la aplicación, y proporciona métodos para el acceso a cada uno de ellos. Además, es el que proporciona el acceso a los daos del usuario/a que ha iniciado sesión en la aplicación (ver ilustración 47).

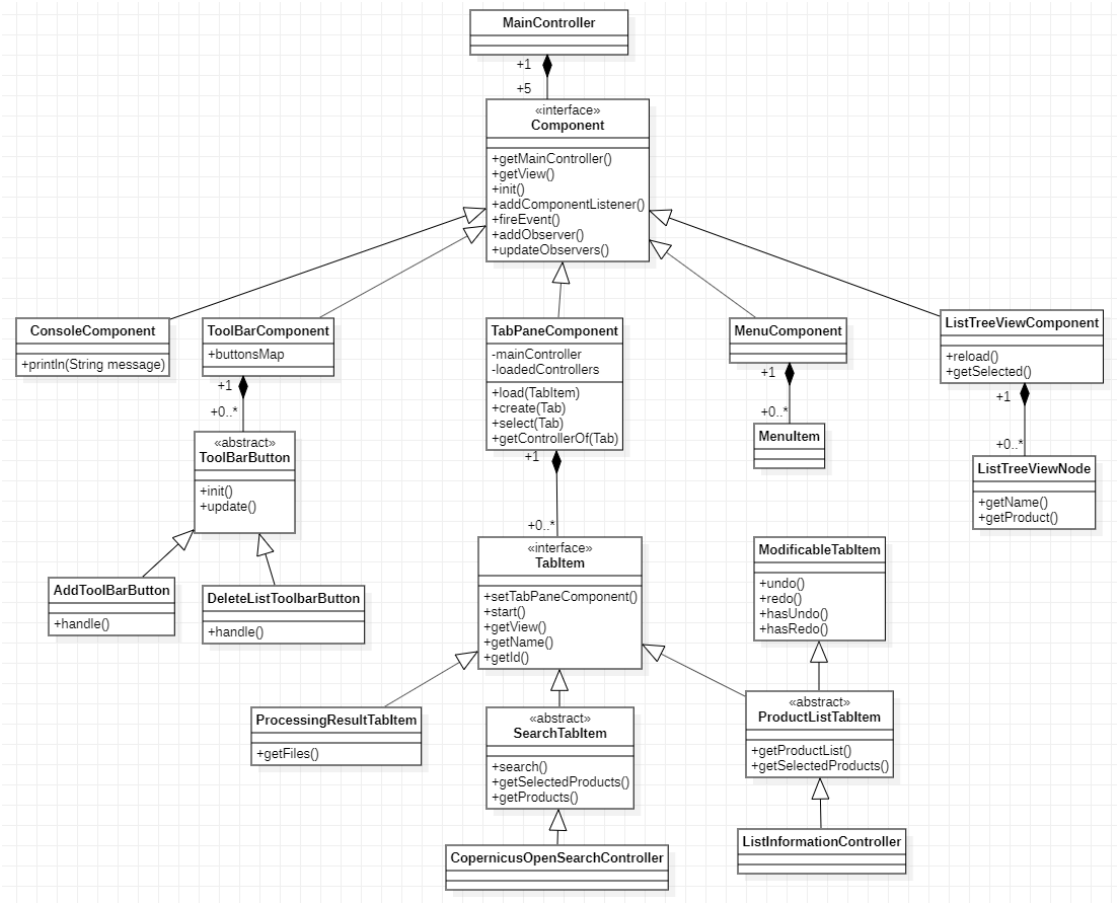


Ilustración 47 - Diagrama de clases de la interfaz

Tras contar con los servicios de la base de datos, y el módulo de identificación del usuario/a terminado, se dispuso a desarrollar el modelado de las listas o colecciones de productos. Dichas listas de productos se definen como un conjunto de productos que tienen alguna característica en común y de interés para el usuario/a. Además, cuenta con un nombre y una descripción para su identificación, además de métodos para obtener el número de productos, y el tamaño de almacenamiento, obteniendo la suma total en gigabytes.

La primera operación que se implementó sobre este modelo fue la creación de una lista. Se utiliza un botón ubicado en la barra de herramientas (*ToolBarComponent*), y que, al accionarlo, muestra un dialogo para especificar las características de dicha lista. Esto es, nombre y una descripción para dar una explicación un poco más detallada sobre el propósito de la lista. Y, por último, una serie de restricciones (ver ilustración 48).

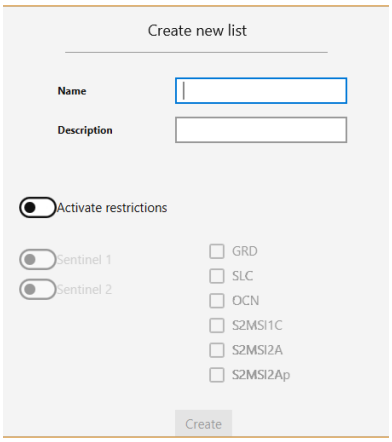


Ilustración 48 - Vista de crear lista

El objetivo de las restricciones es evitar confusiones a la hora de añadir una serie de productos a una lista. Estas restricciones son, de momento, de plataforma (Sentinel 1, 2, o ambos) y tipo de productos (GRD, SLC, OCN…) El usuario/a puede tanto desactivar estas restricciones o seleccionar cualquiera de ellas, según sus intereses.

Una vez ya se pueden crear nuevas listas en la aplicación, se desarrolló el controlador llamado *ListInformationController* y su correspondiente vista para visualizar todos los productos de una lista. Al igual que el *CopernicusOpenSearchController*, también cuenta con atributos y métodos dedicados a componentes de JavaFX y FXML. Dichos métodos se encargan de controlar el correcto funcionamiento de los diferentes eventos que se ejecutan tras la interacción del usuario con la interfaz.

Se ha basado en la vista de resultados de una búsqueda, ubicando en la parte izquierda los productos de la lista, y a la derecha un mapa donde se muestran las localizaciones de los productos (ver ilustración 49).

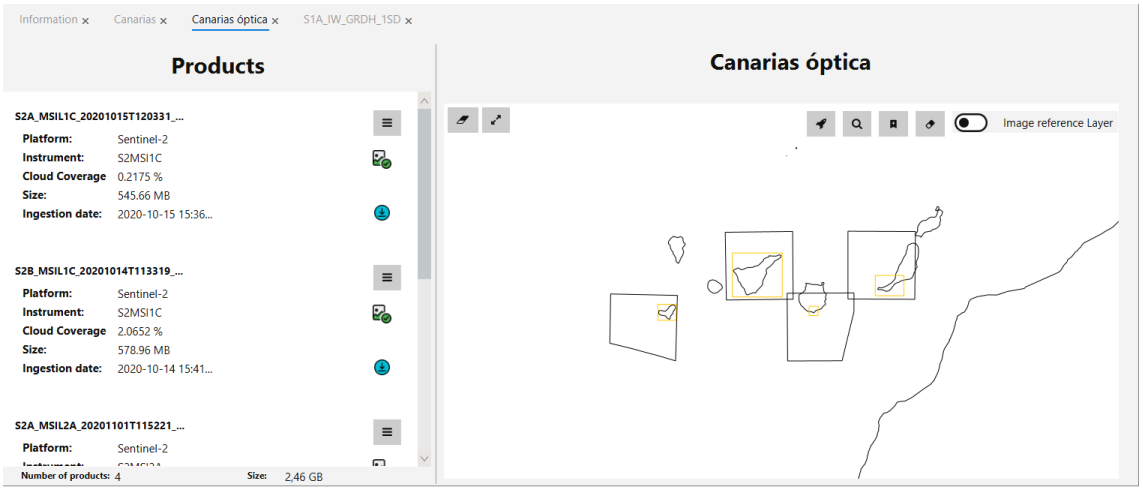


Ilustración 49 – Vista de una lista de productos

Con esto, se desarrolló una clase abstracta, homóloga a la utilizada para el buscador, para permitir identificar esta pestaña como una pestaña de tipo *ProductListTabItem*. Esta define métodos como, por ejemplo, obtener la lista de productos asociada al controlador, obtener los productos seleccionados, etc. Todas estas operaciones están dirigidas para los menús y la barra

de herramientas, proporcionando una interfaz única para cuando se realicen operaciones sobre dicha lista.

12.3.3.1 Añadir productos seleccionados a la lista

Para añadir productos a una lista se tiene que pulsar el botón ubicado en el parte superior llamado “Add selected produtct to list”, o en el menú de listas. Esta acción permite añadir los elementos seleccionados en la pestaña del buscador (de momento, la del Copérnico) y añadirla a una lista que seleccionamos posteriormente.

Estas operaciones entre la barra de herramientas y el menú son compartidas, y utilizando la nomenclatura ya utilizada por JavaFX, se han denominado “Eventos”. Existe un evento por cada tipo de operación que se puede realizar en la aplicación, e implementan la interfaz *EventHandler*, empleada por JavaFX para ejecutarlos. Por ejemplo, en el caso anterior, se llama al *AddSelectedProductsToListEvent*, donde, inyectándole el controlador principal, es capaz de llamar al componente *TabPaneComponent*, obtener el controlador de la pestaña activa, verificar que se trata de una pestaña buscador, y añadir los productos seleccionados a la lista deseada.

Utilizando la interfaz común para los buscadores (*SearchTabItem*), se consigue desacoplar estos módulos, así que independientemente del buscador utilizado, se puede obtener los productos seleccionados. Para agilizar esta operación, si el usuario/a no cuenta con ninguna lista, y añade los productos seleccionados, automáticamente se creará una lista llamada “Default” y se agregaran dichos productos a ella. Si en el caso de que el usuario/a solo cuente con una sola lista, al ejecutar la acción se agregarán dichos productos a esta lista (sin necesidad de tener que seleccionar una lista). El último caso es si el usuario/a tiene varias listas, se mostrará un diálogo con un listado de lista que el usuario/a puede seleccionar, incluso varias a la vez (ver ilustración 50).

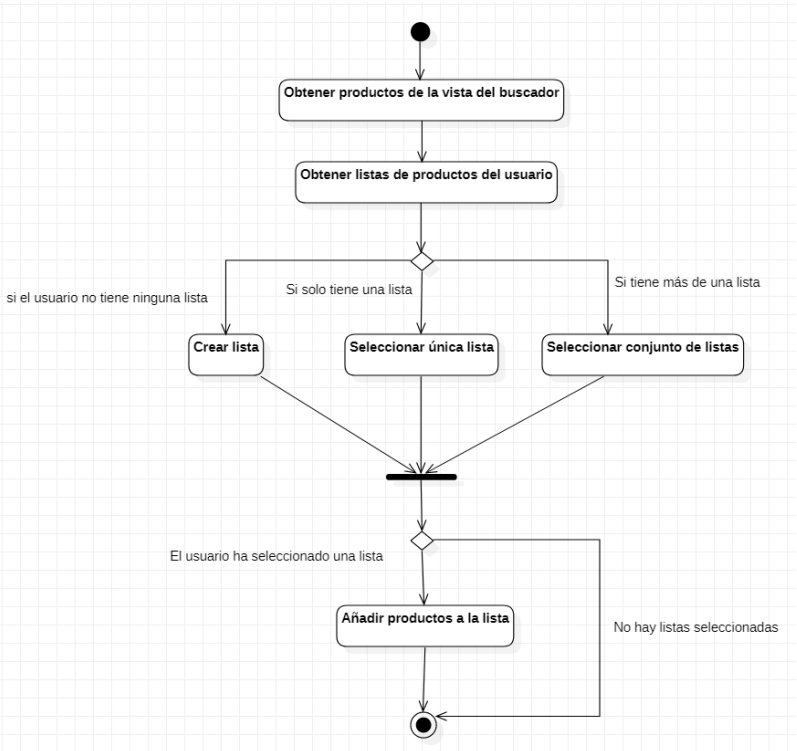


Ilustración 50 - Diagrama de actividades para añadir productos a una lista

Cuando se añaden estos productos a una lista, si esta cuenta con restricciones se verifica que todos los productos cumplen con dichas restricciones. En el caso contrario, no los añade (ver ilustración 51).

```
public boolean addProduct(List<ProductDTO> products) {
    products.forEach(p->{
        if (!this.products.contains(p) && validate(p))
            this.products.add(p);
    });
    //Reload sizes property
    reloadProperties();
    return true;
}
```

Ilustración 51 - Método para añadir una serie de productos a una lista

Se está haciendo uso de las colecciones que ofrece JavaFX, llamadas *Observable*. Estas colecciones implementan el patrón Observer por defecto. Este patrón de diseño permite, a través de una serie de interfaces, llamar a una serie de objetos observadores cuando se ha modificado el estado del objeto observado. El objetivo es que si varios comparten un dato (una cadena, un entero, etc.), y se modifica en una parte del código, dicho dato se actualice también en el resto de la aplicación.

Esto es especialmente útil en la parte de la presentación, debido que si se muestran unos datos que posteriormente han sido modificados (ya sea por el usuario/a, o por alguna ejecución de la aplicación), automáticamente se actualicen en el resto de los componentes. Además, ofrece la posibilidad de añadir eventos dentro de estas colecciones, de tal forma de que, si ha alterado, ya sea añadiendo, eliminando o modificando, se accione. Con esta funcionalidad, podemos hacer que, si se ha modificado un dato, se accione el evento para guardar dichos datos en la base de datos.

Un ejemplo de este comportamiento es el siguiente: si el usuario/a está en la vista de búsqueda, si añade nuevos productos a una lista que ya está cargada en el panel de pestañas, utilizando este tipo de contenedores, la vista se actualizaría automáticamente, sin tener que realizar ninguna operación de refresco.

12.3.3.2 Añadir todos los productos a la lista

Esta opción es para añadir todos los productos dentro de los resultados de una búsqueda y agregarlos a una lista. Esta operación utiliza la misma mecánica que la de añadir productos seleccionados: llama al *TabPaneComponent* para obtener controlador *SearchTabItem* y obtener todos los productos de la búsqueda, y añadirlos a una o varias listas.

Con el aumento de operaciones que se pueden realizar en la aplicación, se decidió separar los botones del componente de la barra de herramientas. Se creó una clase abstracta llamada *ToolBarButton*, que extiende de *Button* (clase de JavaFX) e implementa un *EventHandler* (manejador de eventos). De esta manera, todos los botones de la barra de herramienta extenderán esta clase, teniendo que implementar métodos para almacenar una referencia a la barra de herramientas (y proporcionar comunicación entre los otros componentes), iniciar los datos del botón y desarrollar el evento que se ejecutará cuando se accione. Esta capa de

abstracción hace que simplemente se vayan cargando los botones (Almacenados en un mapa), sin necesidad que la barra de herramientas se responsabilice del funcionamiento de los mismos. Algunos de estos botones son los descritos anteriormente, como, por ejemplo, los de crear lista, añadir productos seleccionados en el buscador, etc. (ver ilustración 52).

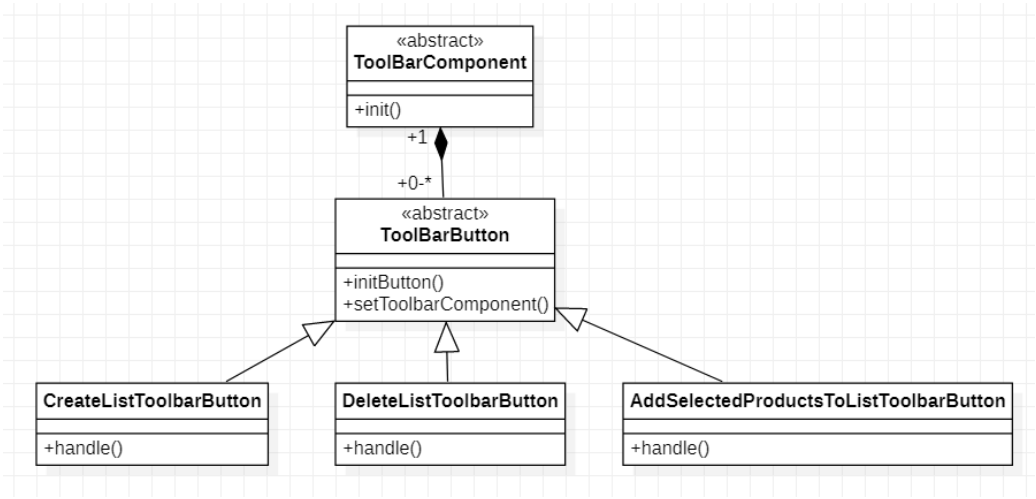


Ilustración 52 - Diagrama de clases de ToolBarComponent

12.3.3.3 Eliminar productos seleccionados de una lista

Se ha añadido un botón para eliminar los productos seleccionados de una lista. Se actualizan automáticamente tanto los valores de número de productos en una lista, como el total de ocupación de los productos en el sistema de almacenamiento del usuario/a (sin comprimir).

También se ha implementado un botón que se muestra dentro de lista de productos que sirve para eliminar a un producto en específico de la misma.

12.3.3.4 Aumentado la capacidad de las listas

La siguiente funcionalidad a desarrollar fue la de seleccionar un área de trabajo a los productos de una lista, ya que para el procesado es necesario reducir el tamaño del producto (cuando más grande es la dimensión de la imagen, más amplio será el tiempo dedicado a su procesado). Para esto, se deben considerar varios aspectos.

El escenario óptimo sería que todos los productos de una lista “contuvieran” el área de trabajo, sin embargo, esto no siempre debe ser así. Así pues, las listas almacenarán una serie de ubicaciones para las diferentes localizaciones de los productos.

En una primera aproximación para abordar este problema, se desarrolló una localización por defecto, donde la aplicación comprobaba si la localización del producto contenía dicha área por defecto. Si es así, se mostraba la imagen verificada (mostrando un icono de verificación) y pasaba a la siguiente imagen; de lo contrario, la aplicación trataba de encontrar una localización específica para dicho producto, utilizando su identificador único. Si encontraba una ubicación, al igual que en el anterior caso, se verificaba y pasaba al siguiente producto. Si de lo contrario no se encontraba, aparecía no verificado (en rojo), teniendo el usuario/a que resolver dicho problema seleccionando un área para dicho producto.

El problema de esta aproximación es la redundancia de datos, ya que, si teníamos varios productos dentro del misma área y no estaban dentro del área por defecto, sino en otra de

interés del usuario/a, para cada producto se guardaría la misma zona. Si esto lo expandimos podemos tener un serio problema para manejar dichos datos, ya sea en operaciones de lectura y escritura, sobre todo, puesto que, si eliminamos un área, habría que comprobar las áreas asociadas a un producto y realizar las operaciones oportunas.

Por lo tanto, para evitar esta redundancia, se implementó simplemente como un contenedor, que almacena todas las áreas de interés. Así pues, se necesitó que el modelo de la lista de productos tuviera operaciones para obtener las áreas de trabajo que pueden estar contenidas dentro de un producto.

Una cosa positiva sobre estas áreas es que hemos permitido asignar más de un área de trabajo sobre un mismo producto, explotando al máximo las características que puede ofrecer. En la siguiente figura se puede observar un ejemplo de lo expuesto anteriormente, donde en un solo producto puede contener varias áreas de trabajo (ver ilustración 53).

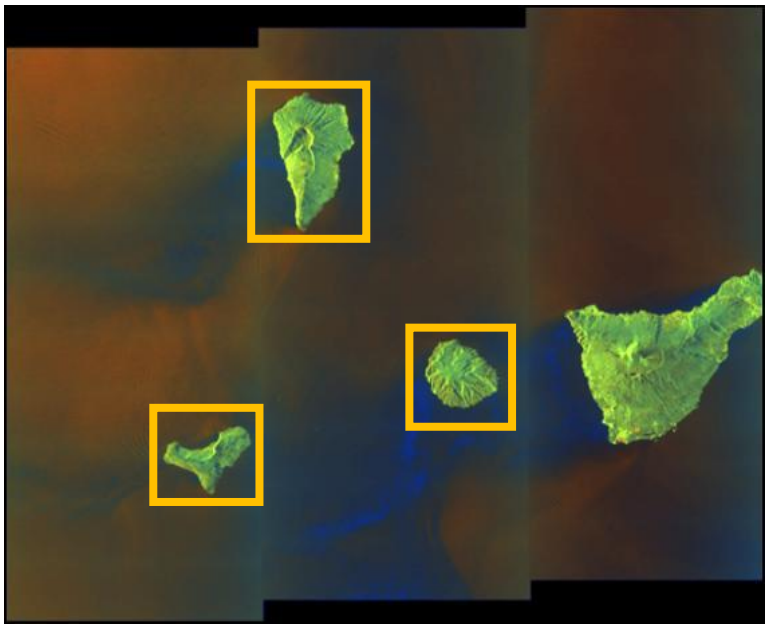


Ilustración 53 -Sub imagen de las Islas Canarias. Imagen obtenida de la API del Copérnico

Su implementación en el mapa fue bastante sencilla, debido a las operaciones ya desarrolladas durante la búsqueda. Lo único que fue necesario hacer fue expandirlo. Tener varias capas al mismo tiempo, representar varios productos en diferentes, poder seleccionar una capa en concreto, etc.

El controlador contiene los métodos para añadir una nueva área de trabajo, o eliminarla. Para esto primero obtiene el área que el usuario/a ha seleccionado, y lo inserta en la lista de productos (ver ilustración 54).

```java
private void onActionInAddAreaOfWorkCreateAreaOfWork() {
    addAreaOfProduct.setOnAction(event -> {
        //If a area is marked
        if (!mapController.getWKT().isEmpty()) {
            //Add area to productList
            productListDTO.addAreaOfWork(mapController.getWKT());
            drawInMapTheAreasOfWork();
        }
    });
}
```

Ilustración 54 - Añadir área de trabajo a una lista

Automáticamente, se refresca el componente de lista de JavaFX que se utiliza para mostrar los productos cuando se añade o se elimina un área de trabajo, con el fin de mostrar visualmente si un producto cumple con la nueva área de trabajo.

Tras tener estas operaciones definas, se desarrolló el DAO de las listas para su correspondiente almacenamiento en la base de datos. Aquí se puede observar con más claridad por qué separar el modelado de los controladores y las vistas de la base de datos, puesto que en el DTO tenemos operaciones como obtener las áreas de producto contenidas por un producto, u obtener el tamaño total de la lista. Este tipo de operaciones en la parte de servicios no tienen ninguna utilidad, puesto que en esa parte lo único necesario son los datos para almacenarlos en la base de datos y viceversa, indicando en sus campos las relaciones y restricciones para cumplir el modelo de base de datos.

Para el desarrollo del *ProductListDBDAO*, es necesario desarrollar el DAO correspondiente a los productos, para almacenar su información (ver ilustración 55). La peculiaridad de éstos son que existen diversas clases que extienden a *ProductDTO*. Para controlador el correcto paso de datos entre el modelo de entidades y DTOs, se creó un traductor para mediar entre los dos.

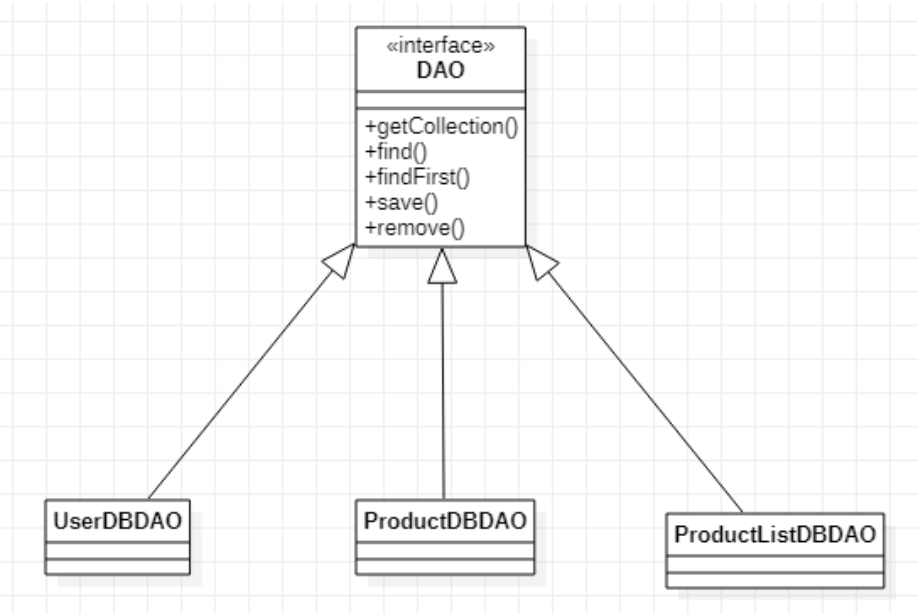


Ilustración 55 - Diagrama de clases para conectar con la base de datos

A la hora de implementar esta colección, hubo un par de problemas iniciales, pero que, gracias a las características que ofrece NoSQL, lo reducen de manera considerable. Estos problemas son las características de la lista, que cuenta con un conjunto de contenedores, uno de Strings, que almacenan las áreas de interés de los productos, y otro de restricciones, dependiendo de lo que haya especificado el usuario/a (que puede ser ninguno). Para MongoDB, y especialmente para *Morphia*, el framework utilizado para utilizado para manejar la base de datos (y de realizar los mapeos de las colecciones a objetos Java y viceversa), necesitaba que dichas restricciones se almacenarán en una colección diferente, para así poder serializarlas de forma correcta. Sin embargo, el mismo framework ofrece hacer esta clase "*Embedded*", que hace que, en vez almacenarse en una colección, se almacene junto a las listas, en forma de objeto (con las características de las restricciones). Esto facilita tanto la lectura como la escritura de las listas, puesto que se almacenan en un mismo sitio, y con una simple operación, obtenemos todas las listas, sus áreas y sus restricciones. Sin embargo, tal como se puede ver en la ilustración 55, *Morphia* necesita de conocer la clase a la que pertenece ese objeto, por lo que almacena su nombre en el documento (ver ilustración 56).

Ilustración 56 - Colección de Listas de Productos en MongoDB

En cuanto a los productos, no hemos utilizado este comportamiento. Se ha seguido un enfoque relacional, creando una colección de las mismas. El motivo es evitar la excesiva redundancia que podría haber de los productos, puesto que el mismo producto puede estar en varias listas, almacenando un producto tantas veces como este el producto en una lista. Si el producto está en 50 listas, habrá 50 copias del mismo producto.

12.3.4 Sprint 3 – Descargas

Para desarrollar este módulo se ha necesitado tres clases concretas, la primera, el servicio de descargas, que se encarga de controlar todas las descargas de la aplicación. Otra clase que llamada *DownloadItem*, que se define como un elemento a descargar, compuesto por un producto y una ubicación. Además, proporciona métodos para comunicarse con la vista, mediante propiedades de JavaFX (para el progreso de la descarga). Y por último el *DownloadThread*, que se encarga de descargar el producto en un hilo diferente al de JavaFX. La particularidad de este módulo es la restricción que existe en la API del Copérnico para el número de descargas concurrentes, que es 2 cómo máximo. Así pues, se ha implementado cumpliendo dicha restricción.

Cuando el usuario/a acciona el botón que determina que hay que descargar una serie de productos, se ejecuta un evento, que los añade a una pila almacenada en *Downloader*. Si éste verifica que no hay ninguna descarga actualmente activa, o si las hay, son menores a dos, entonces obtiene un *DownloadItem* de la pila y lo inyecta a un *DownloadThread*. Este se encarga de verificar que el producto no está descargado, y que está disponible en la API del Copérnico. Tras eso, comienza la descarga en un hilo aparte. Si la descarga se ha realizado de forma satisfactoria, entonces, se elimina el producto de la lista de descargas activas. Si quedan más productos que procesar, se irán descargando hasta que la cola se vacíe. Si, por el contrario, la cola de descargas activas está llena, entonces el *Downloader* espera a que estas descargas se terminen para comenzar la descarga del siguiente producto (ver ilustración 57).

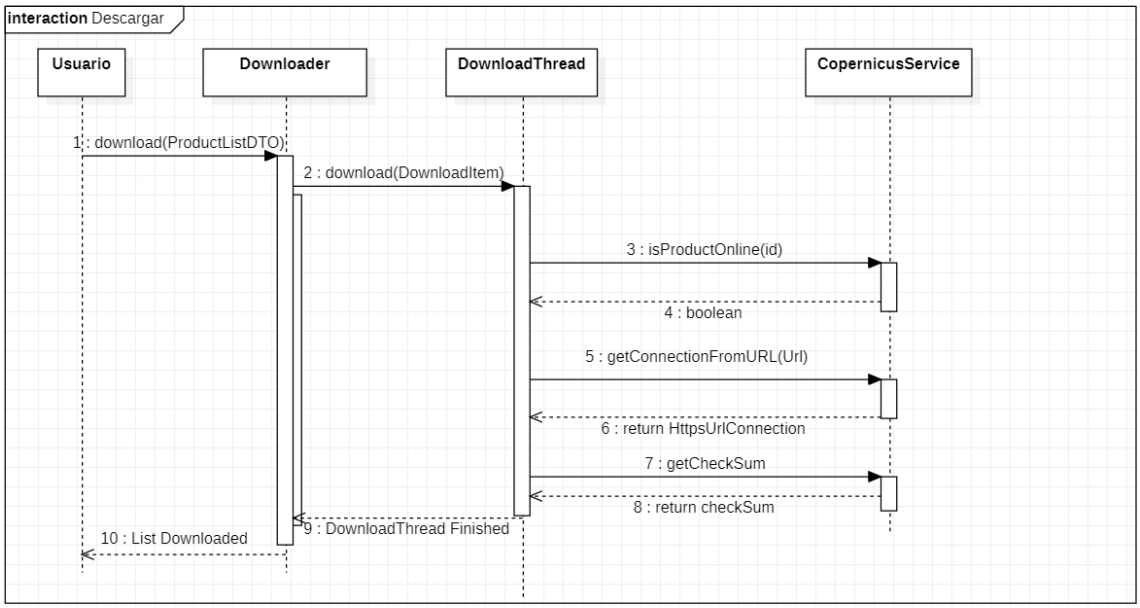


Ilustración 57 - Diagrama de secuencia de descarga de una lista de productos

La descarga del producto se hace mediante una llamada al *CopernicusService*, el cual da la posibilidad de obtener el contenido de una URL en forma de *InputStream* (al igual que se usa para el sistema de búsqueda). Sin embargo, para implementar de forma correcta esta parte, se ha implementado otro método, que en vez de retornar el *InputStream*, devuelve la conexión (ver ilustración 58). Esto es así ya que se necesita acceder a las cabeceras de la respuesta HTTP del servidor para conocer el tamaño del producto en bytes, con el fin de implementar un sistema de progreso. También se verifica antes de comenzar si un producto está disponible para descargar. Para esto necesitamos hacer una llamada a la API para comprobar dicho estado. Se ha intentado realizar esta operación durante la obtención de productos en el buscador, pero el

tiempo requerido para realizarlo es muy amplio (entre 0.5 y 2 segundos), ya que dicho estado se obtiene a través de otro recurso de la API. Por lo tanto, se decidió dejar esta comprobación durante la descarga, ya que aquí ese tiempo no importa tanto (en el buscador, dependiendo del número de productos obtenidos, el tiempo se multiplicó por 3, de 20 segundos a 60).

```
@Override
public HttpURLConnection getConnectionFromURL(URL url) throws IOException,
        AuthenticationException, NotAuthenticatedException {

    if (httpManager == null)
        throw new NotAuthenticatedException("Not authenticated");
    return CopernicusHTTPAuthManager.getNewHttpManager(pair.getKey(),pair.getValue()).getConnectionFromURL(url);
}
```

Ilustración 58 - Método en el servicio Copérnico para obtener la conexión a una petición HTTP

Este sistema lee el *InputStream* de forma tradicional, es decir, leyendo a través de un buffer una cantidad de datos, y escribiéndolos en un *FileOputWriter*. Mediante este sistema, se calcula la cantidad total de bytes, y el número de bytes leídos, y así poder implementar el progreso de la descarga. Al final de esta, se comprueba el *checksum* del archivo resultante, y verifica que se ha descargado correctamente. Para esto se utiliza de nuevo la API, que nos da el *checksum* del archivo en su servidor. Si estos no coindicen, se borra el archivo resultante.

Otro de las funcionalidades que se le permite al usuario/a es la de parar, reanudar, o cancelar todas las descargas (o una en específico). Para esto se ha usado una enumeración llamada *Command*, al que se les pasa a los hilos (que se almacenan en el mánager en una lista convencional). Estos comandos son los de parar, pausar o reanudar. Cuando se ejecuta alguno de estos comandos, es leído por el hilo. Por ejemplo, si se ejecuta el comando de pausar, para la descarga del producto hasta que se vuelva a reanudar. Si es el de cancelar, entonces el hilo cancela la descarga, elimina el contenido descargado si lo hubiera, e indica al *DownloadManager* que ha terminado.

Si el producto ya existe, o no está disponible, se quita de la cola, la aplicación muestra un mensaje de aviso, y continua con el siguiente producto a descargar (ver ilustración 59).

```
if (FileUtils.fileExists(item.getLocation()+"\\"+item.getProductDTO().getTitle()+".zip"))
    throw new FileAlreadyExistsException("Product already exits");

if (!ServiceFactory.getService(item.getProductDTO()).isProductOnline(item.getProductDTO().getId())) {
    throw new ProductNotAvailableException("Product not available!");
}
```

Ilustración 59 - Comprobaciones en DownloadThread antes de comenzar a descargar un producto

Finalmente, el diagrama de actividad de las descargas sería como el mostrado en la ilustración 60:

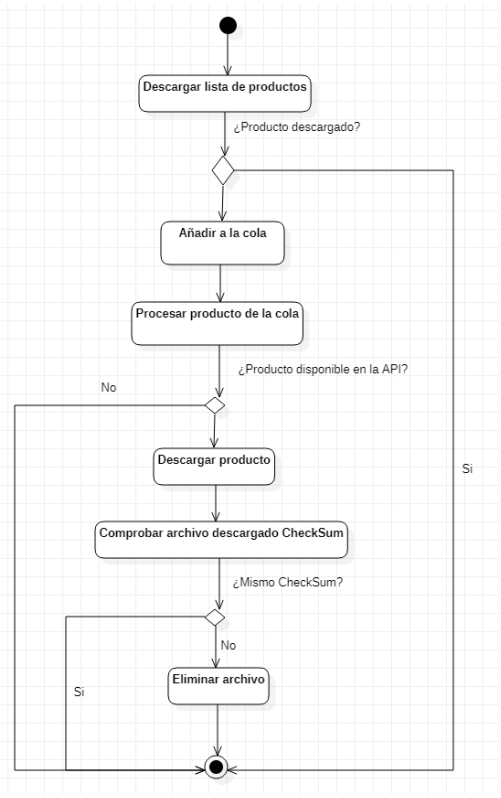


Ilustración 60 - Diagrama de actividades de las descargas

Un problema que se ha encontrado durante el desarrollo de esta versión de las descargas es el número máximo de descargas que permite la API concurrentemente, 2. En algunas ocasiones, parecía que la cancelación o la terminación de una descarga, y el inicio de otra podía dar lugar a un error de esta concurrencia. Esto es debido a que es necesario cerrar todos los recursos que están conectados a la petición HTTP. Esto son los *Inputstream*, conexiones, etc. A veces, aun cerrándolos, es posible que Java tarde en ejecutar el cierre de éstos. Para evitar que las peticiones se cancelen sin parar, se ha dado un número de intentos para intentar descargar un producto, 5. Si tras esos intentos sigue sin poderse sin descargar un producto, el número máximo de productos que se podrán descargar a la vez se reducirá a uno.

Normalmente este caso aparecía cuando había dos descargas ejecutándose y una se cancelaba. De ahí que solo se permita una. Cuando esta acaba, se reinicia el contador. Si tras esto sigue sin poderse descargar un producto, se vacía la cola y se manda un mensaje de error general al usuario/a, indicando que realice la descarga más tarde (ver ilustración 61). Si, por el contrario, se permite la descarga, vuelve a pasar a poder descargarse dos descargas concurrentemente.

```
private void onFailed(DownloadItemThread thread, WorkerStateEvent e) {
    thread.setCommand(DownloadEnum.DownloadCommand.STOP);
    thread.cancel();
    try {
        thread.closeStreams();
    } catch (IOException ioException) {
        logger.atError().log( message: "Error while closing download item thread {}",e);
        ioException.printStackTrace();
    }

    handleError(thread.getDownloadItem(), e);

    removedDownloadItemFromQueues(thread.getDownloadItem());

    if (e.getSource().getException() instanceof HttpResponseException) {
        if (downloadAttempts>0) {
            attempt();
            logger.atError().log( message: "Attempt to download! {} left...",downloadAttempts);
            queue.add(thread.getDownloadItem());
            Platform.runLater(()->historical.add(thread.getDownloadItem()));
        } else {
            logger.atError().log("5 attempts to download products, all with error. Try later!");
        }
    }
    removeDownloadFile();
```

Ilustración 61 - Método para controlar el fallo de una descarga

La vista desarrollada para las descargas es bastante simple, y reconocible. Se trata de dos listas, una mostrando los productos que se están descargando en este momento, y otra con los productos en la cola de descargas. A la derecha encontramos una serie de botones para controlar las descargas: pausar, reanudar, limpiar cola y cancelar todas las descargas (ver ilustración 62).

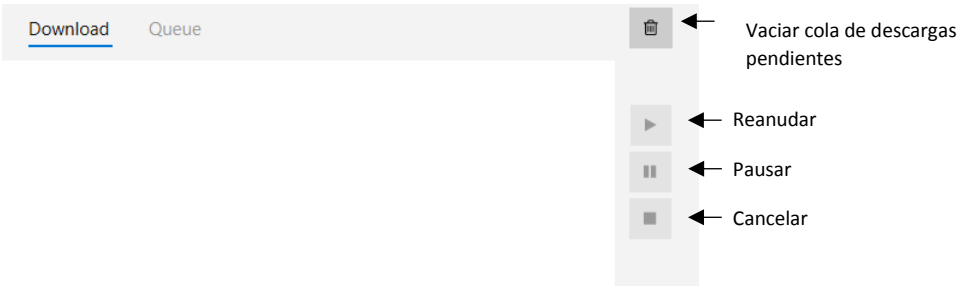


Ilustración 62 - Vista de las descargas

En el listado de los productos que se están descargando actualmente aparece una barra de progreso para cada uno de ellos (ver ilustración 63). Esto lo conseguimos obteniendo el tamaño del archivo, almacenado en las cabeceras de la petición HTTP. Si el contenido es muy grande, esta cabecera no podrá almacenar el tamaño del producto, así que se utiliza el tamaño obtenido de la API. Dicho tamaño no es del todo fiable, puesto que ese tamaño del producto representa el tamaño del archivo sin comprimir, por lo que el tamaño final siempre será inferior.

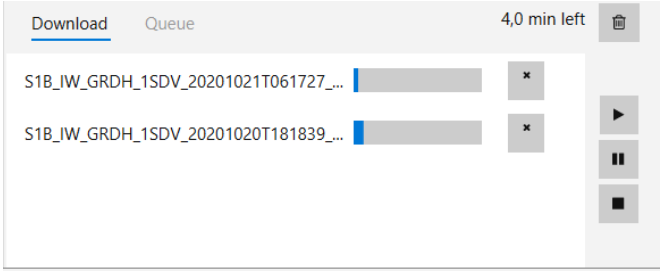


Ilustración 63 - Descargas activas

Además de la barra de progreso, a su derecha, existe un botón para cancelar dicha descarga. Para esto están los comandos, los cuales son enviados al hilo correspondiente, parando la descarga, eliminando el archivo temporal, y terminado su ejecución. La vista se actualiza automáticamente utilizando las colecciones observables de *JavaFX*.

Por último, para indicar al usuario/a final que un producto de la lista ha sido descargado, se utiliza un icono para visualizar de manera clara éste hecho. Consecuentemente, aquellos que no cuenten con él, significa que no han sido descargados todavía.

Para dotar de una mayor flexibilidad al usuario/a con la gestión de las descargas, se ha creado una vista de preferencias, o configuración, de las mismas. Para implementar esto, se ha utilizado las preferencias de Java, que proporcionan de una interfaz para almacenar información de forma local y permanente. Utilizando una serie de nodos, que son utilizados para identificar una preferencia (que puede tener varios campos).

Dentro de esta configuración están los directorios destino de los productos y las listas (donde se almacenarán las imágenes procesadas), pudiéndose modificar en cualquier momento. Si existen descargas activas en ese momento, dichas descargas se terminarán descargando en el directorio antiguo. Todas las nuevas descargas que empiecen después de modificar el directorio final se almacenarán en él.

Otra configuración es el modo de descarga, secuencial o en paralelo. Si se activa el modo secuencial mientras hay varias descargas activas, se espera a que estas terminen, pero no se deja entrar a ningún producto nuevo hasta que la cola que se estén descargando actualmente no se vacíe.

Por último, otra configuración interesante es el la de autodescargado. Si el usuario/a activa esta opción, los productos que se añadan a una lista se descargaran automáticamente.

12.3.4.1 Imagen de referencia.

Una nueva funcionalidad que se ha implementado es la de añadir imágenes ópticas como imágenes de referencia, siempre asignadas a zonas de trabajo. Esta asignación es totalmente opcional por parte del usuario/a. Si finalmente las añade, la aplicación descargará estas imágenes juntos al resto de productos, almacenándolos en el mismo directorio.

Se ha modificado el modelado de las listas de productos, permitiendo tanto añadir como eliminar imágenes de referencia. Éstas se guardan a en un contenedor diferente a los productos de la colección.

La primera versión para implementar la funcionalidad anteriormente descrita se basa en una serie de botones y un *ToggleSwitch*. Este último está empotrado en el mapa, y su objetivo es que el usuario/a pueda elegir si quiere seleccionar áreas de trabajo, o imágenes de referencia.

Tras buscar las imágenes, y seleccionar la mejor de los resultados, tendrá que accionar un botón en la barra de herramientas llamado “Add selected product as reference image”. Si el producto no contiene ninguna zona de trabajo, no se podrá añadir, mandando un mensaje de error al usuario/a.

Hay varios colores diferentes para representar a cada elemento de la lista. Para los productos se utiliza el color negro, ya que es el que no se modifica, a menos que se añada un nuevo producto o se elimine uno. El color amarillo es para las áreas de trabajo, y las áreas verdes indican las

imágenes de referencia. Y, por último, el color azul para los productos que están seleccionados en el componente de listas de JavaFX (ver ilustración 64).

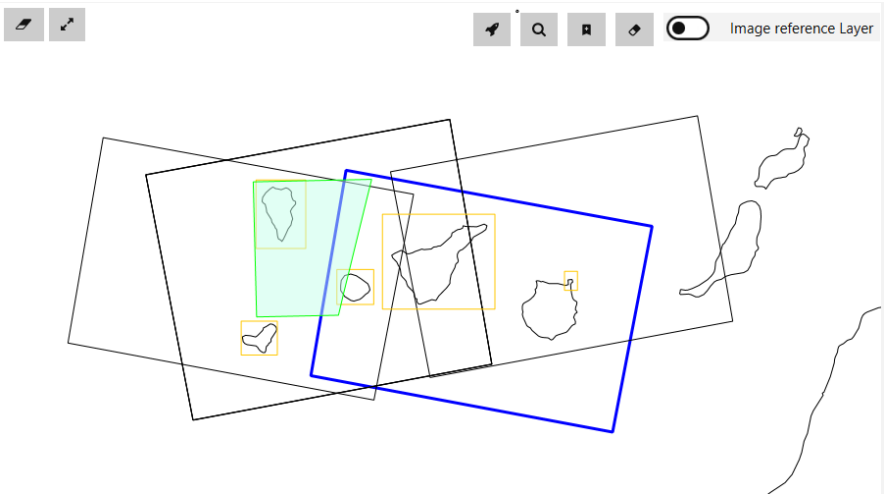


Ilustración 64 - Lista de productos con imágenes de referencia

12.3.5 Sprint 4 - Procesado

Para esta última parte del desarrollo de la aplicación, se han empleado las librerías fuente del programa descrito en el estado del arte, el SNAP [13]. Con esta librería, se mejora el trabajo de procesado, puesto que cuenta con métodos desarrollados propiamente para los productos del Copérnico. Sin embargo, el nivel de complejidad para saber usarlos en su plenitud es muy elevado, por lo que, siguiendo el objetivo del proyecto, su implementación tiene que ser lo más sencilla para el usuario/a, pero que a la vez permita una capa de configuración básica para aquellos que la necesiten.

En el caso del Copérnico, el SNAP ha desarrollado una serie de operaciones que se pueden aplicar a un producto, como por ejemplo aplicar un corrector de terreno, calibrado, aplicar archivo de órbitas, etc. [23] Y cada una de esas operaciones requiere de una serie de parámetros (ve Anexo I). Por este motivo, se han realizado una serie de pruebas probando las distintas combinaciones que se pueden hacer durante el procesado de la imagen. Como resultado, se obtiene un archivo *GeoTIFF*, georreferenciado y con un conjunto de imágenes, una por cada polarización y banda.

12.3.5.1 GRD

Para productos tipo GRD [24], se ha realizado pruebas con las siguientes operaciones [25]:

- **Apply Orbit File**: Utilizado para obtener los datos de órbita del satélite en el momento de adquisición. Su objetivo es georreferenciar el producto.
- **ThermalNoiseRemoval:** Reducción del ruido térmico sobre la imagen.
- **Calibration:** Calibración de la imagen
- **TerrainCorrection:** Corrección de terreno con la utilización de mapas de elevación
- **Write:** Formato de salida, normalmente en TIFF.

Un ejemplo de una imagen SAR tras realizarle un procesado sería la ilustración 65. El tiempo empleado para este procesado ha sido en torno a un rango de 3-5 minutos, dependiendo de las polarizaciones con las que cuente las imágenes.

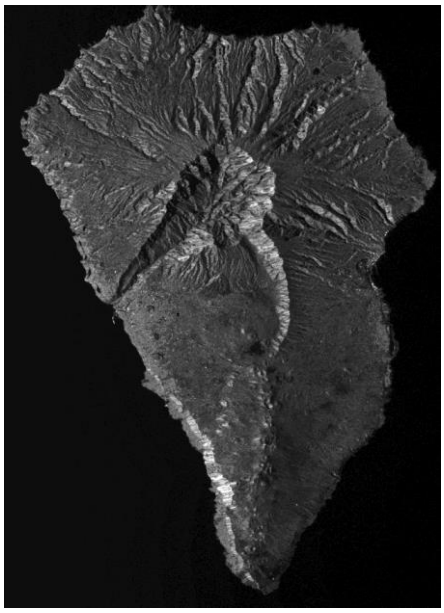


Ilustración 65- Imagen SAR procesada

Durante las primeras pruebas se encontraron una serie de errores en las imágenes procesadas, en zonas con una topografía elevada (ver ilustración 66).

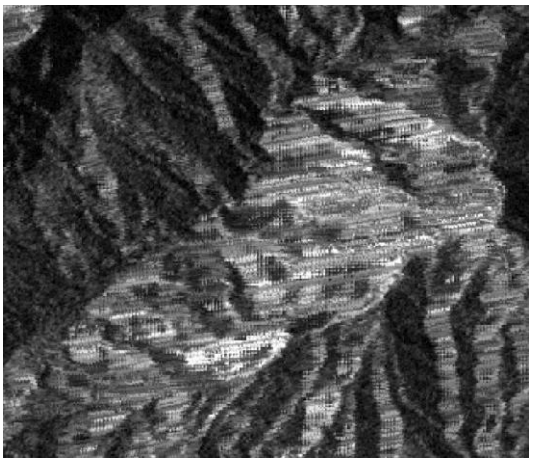


Ilustración 66 – Error en el procesado de la imagen

Dichos errores podrían ser ocasionadas por el ángulo de adquisición del satélite. Para comprobar esto, se eliminaron una serie de operaciones para comprobar el resultado del procesado. Al eliminar la corrección del terreno, se verificó que esos errores se deben a dicho ángulo, junto a una topografía elevada. Como se ve en la ilustración 67, estas zonas se generan cuando la corrección de terreno intenta reconstruir la imagen como si hubiera sido obtenido justamente en vertical. (Ver la ilustración 68 para observar una comparación más cercana).

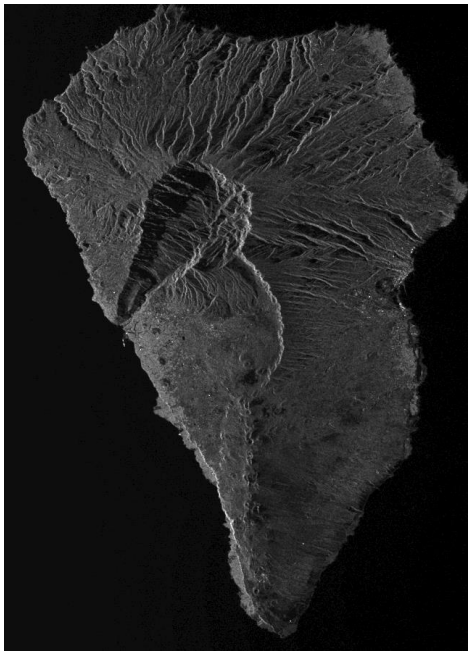


Ilustración 67 – Imagen sin corrección de terreno

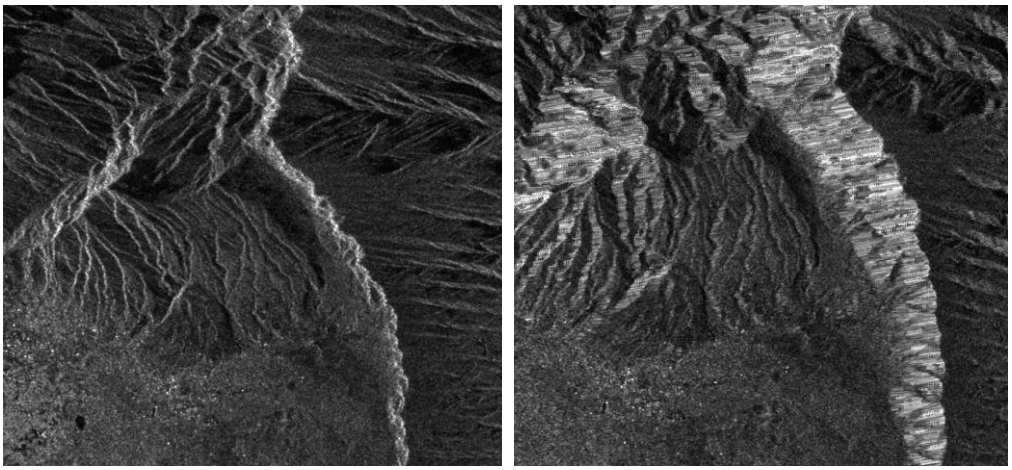


Ilustración 68 – Izquierda, sin corrección de terreno. Derecha, aplicando la corrección de terreno

Para arreglar este tipo de problema, el SNAP cuenta con una operación llamada TerrainFlattening, que intenta reducir estos errores aplicando un allanador sobre la imagen (ver ilustración 69). Sin embargo, para realizar este proceso, se requiere de mucho más tiempo de procesado (en torno a 3 veces más), y, además, perdiendo calidad en la imagen (ver ilustración 70).

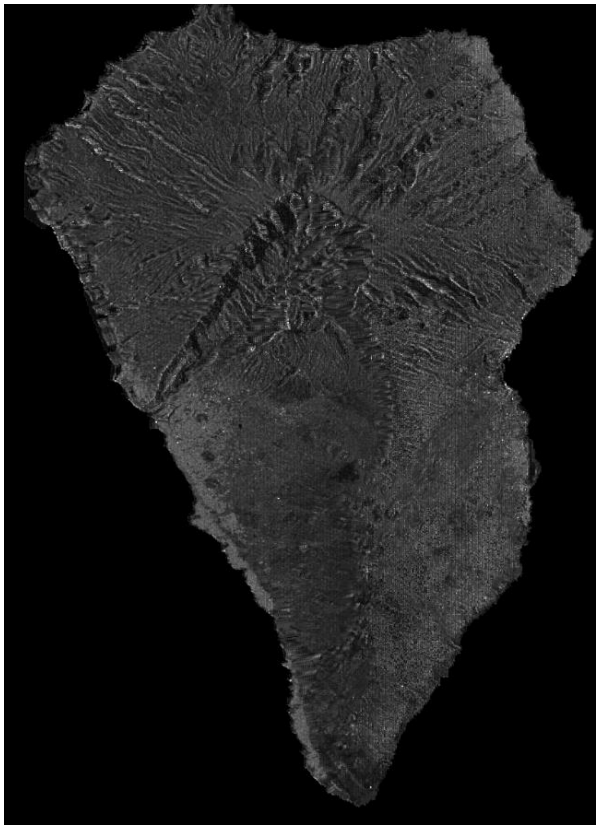


Ilustración 69 – Resultado de la imagen tras aplicar un TerrainFlattening

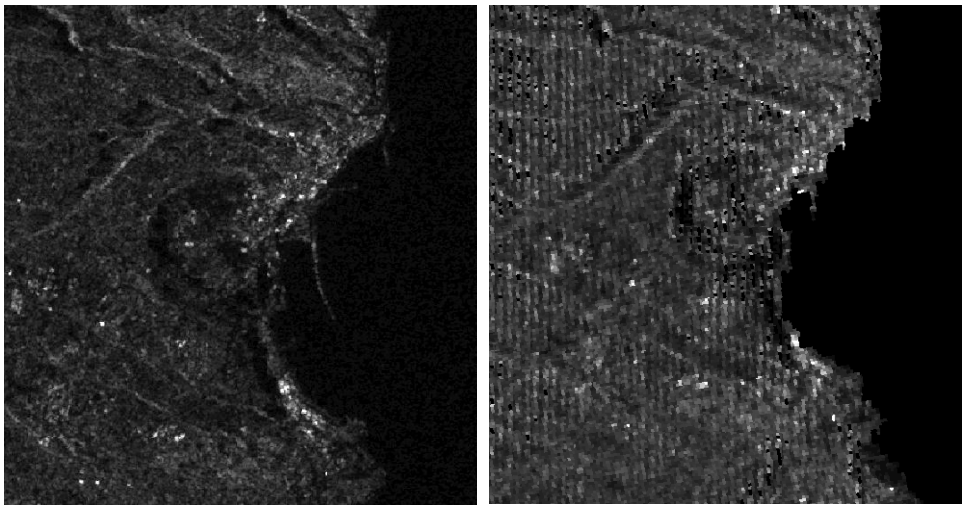


Ilustración 70 – Puerto de S/C de la Palma. Izquierda, sin aplicar TerrainFlattening. Derecha, aplicándolo

12.3.5.2 SLC

Este tipo de imagen cuenta con varias bandas, ya que contiene información sobre la fase (información compleja), y, además, cada una de ellas se divide en tres swaths (áreas de adquisición) [26] (ver ilustración 71).

Se han realizado las siguientes operaciones:

- **TopSARSplit**: Elección del Swath a trabajar. Cada imagen SLC tiene 3 Swaths.
- **OrbitFile**: Aplicado de archivo de órbita
- **Calibration**: Calibración de la imagen
- **TopSarDeburst**: Se utiliza para eliminar las divisiones entre los 3 Swath que cuenta la imagen.
- **TerrainCorrection**: Corrección de terrero
- **Write:** Formato de salida, normalmente en TIFF.

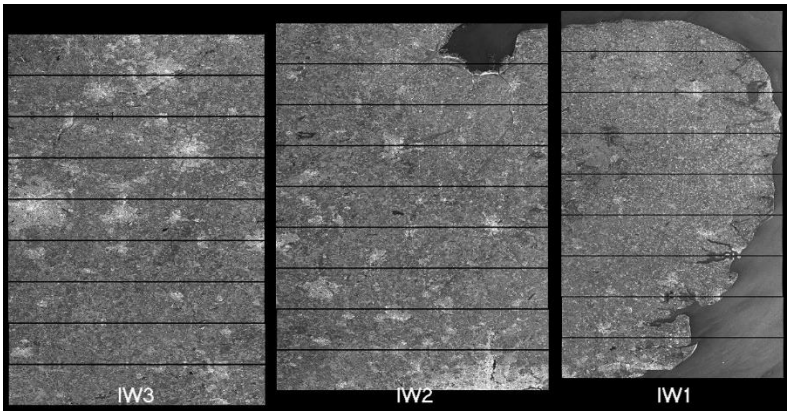


Ilustración 71 – Imagen SLC dividida en los tres swaths. Imagen obtenida de https://sentinel.esa.int/web/sentinel/user-guides/sentinel-1-sar/acquisition-modes/interferometric-wide-swath

12.3.5.3 Sentinel 2:

El procesado de estos productos es mucho más sencillo [27], puesto que solo necesitan de una operación de "**remuestro**", cuyo objetivo es obtener la misma resolución espacial para todas las bandas del producto (imagen multiespectral) (ver ilustración 72 y 73).

Ilustración 72 - Imagen RGB con colores naturales

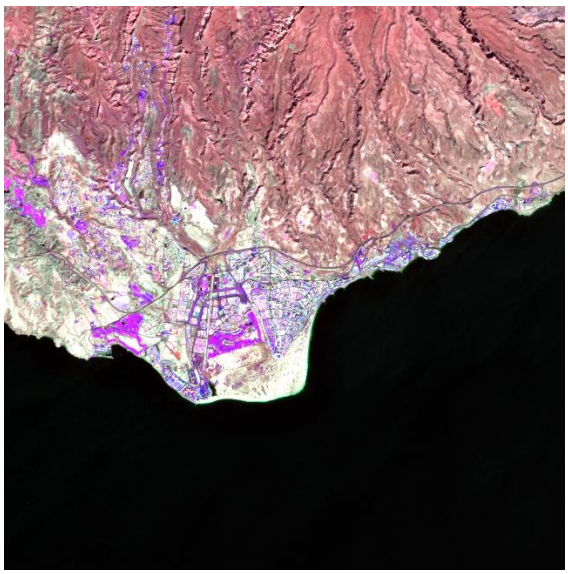


Ilustración 73 - Análisis de vegetación - RGB -> B11, B4, B8

12.3.5.4 Implementacion en la aplicación: flujos de trabajo

Tras comprobar el correcto procesado de estos tres tipos de productos, se comenzó con su desarrollo en la aplicación. Lo primero fue la creación de una clase *Workflow*, que permitiera almacenar una serie de operaciones y aplicarlas sobre un producto. También se creó la clase *Operation*, que se encarga de guardar los parámetros para cada operación.

Así pues, el *Workflow* se define como una serie de operaciones, cada uno con sus parámetros correspondientes, y que se ejecutan en un procesador, llamado *SentinelProcessing*. Para cada uno de los subtipos de productos habrá un *Workflow* diferente, puesto que un producto SLC no basta solo con las operaciones para un GRD, como se ha explicado anteriormente.

Además, se permite que el usuario/a almacene sus propios *Workflow*, y los asigne a una lista de productos, según el tipo. Estos se almacenarán en la base de datos en su propia colección, debido a que debe de haber una referencia del flujo tanto en la colección de usuarios como en la de las listas de productos.

Por otro lado, para los usuarios básicos que no quieran entrar en los detalles sobre el procesado, se da la opción de, si no asigna ningún *Workflow* a una lista, se utilice una serie de flujos de trabajo por defecto, y que están asociados a un tipo de producto en concreto.

Para identificar cada *Workflow* con cada producto, se está utilizando el parámetro "*productType*". Con esto basta para reconocer el tipo de procesado que queremos realizar sobre un producto. Ya sea óptico o SAR, dentro de un tipo "da igual" el tipo de procesado que realizamos, es decir, al nivel con el que se está desarrollando esta aplicación. Para cumplir con este requisito, se ha desarrollado una enumeración llamada *WorkflowType* para los tipos de productos SLC, GRD y los ópticos (ver ilustración 74).

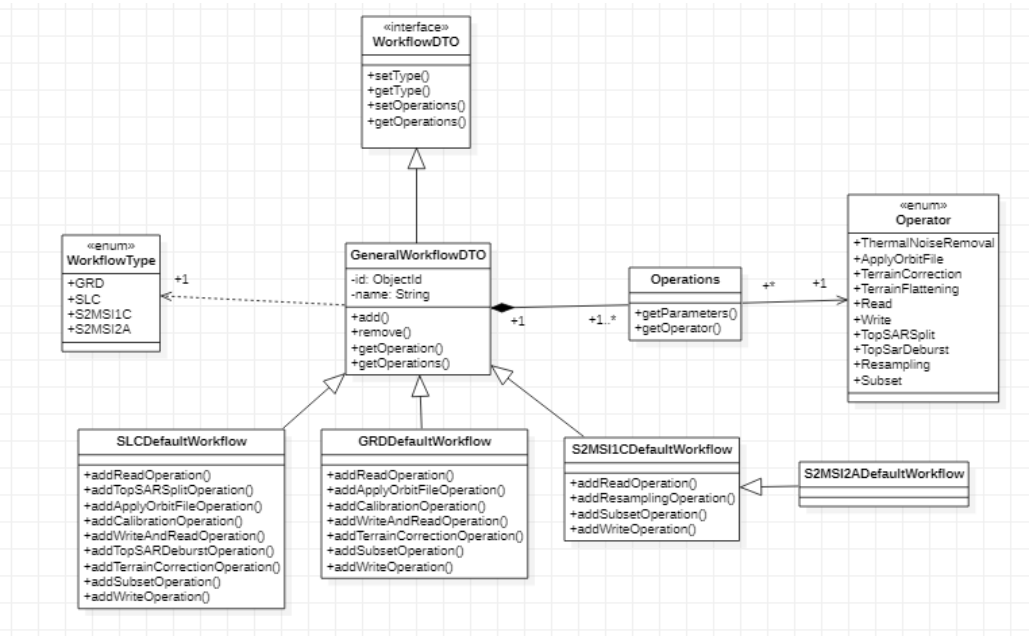


Ilustración 74 - Diagrama de clases de los flujos de trabajo

Respecto a las vistas que muestran dichas operaciones, se han creado una serie de controladores para cada una de ellas. Estos son manejados y administrados por controladores superiores, que son los que definen el *Workflow* de un producto, como, por ejemplo, el *Sentinel1GRDWorkflowController,* para productos GRD.

Cuando a estos controladores se le inyecta un *Workflow*, itera entre sus operaciones, cargando los parámetros en los subcontroladores (ver ilustración 75). En dichos controladores, el usuario/a interactúa modificando los parámetros.

```
@Override
public void setWorkflow(WorkflowDTO workflow) {
    this.workflow = workflow;
    if (workflow.getOperations().isEmpty()) {
        addGRDOperations(this.workflow);
    }
    loadOperationsIntoControllers(this.workflow);
}

private void loadOperationsIntoControllers(WorkflowDTO workflow) {
    List<Operation> operations = workflow.getOperations();
    for (Operation operation : operations) {
        if (operationsMap.containsKey(operation.getName())) {
            OperationController op = operationsMap.get(operation.getName());
            op.setParameters(operation.getParameters());
            if (operation.getName() == Operator.TERRAIN_FLATTENING &&
                    !accordion.getPanes().contains(flatteningPane))

                accordion.getPanes().add( index: 5, flatteningPane);
        }
    }
}
```

Ilustración 75 - Controlador Sentinel1GRDWorkflowController

También se desarrolló la interfaz para visualizar los flujos de trabajo vinculados a un usuario/a, y mostrar las operaciones y parámetros de éstos. Esta vista también tiene la capacidad de añadir, eliminar y modificar los *Workflow*. Si se quiere modificar uno, se hace clic encima de él y se cargará los parámetros de las operaciones asociados a dicho flujo. También hay que destacar que aquí se encuentra la opción de asignar un flujo o varios a una lista de productos.

Se ha utilizado la misma vista para visualizar los flujos asignados a una lista. Si esta no tiene ninguna, dicha interfaz se mostrará vacía. Sin embargo, si se crea un nuevo *Workflow*, se almacenará directamente en la base de datos, y se añadirá la referencia a la cuenta del usuario/a que se encuentra asignada en la aplicación, además de insertar otra referencia en la lista de productos especificada. Para su almacenamiento en la base de datos, se crea el DAO, la entidad y el traductor de Entidad a DTO, y viceversa.

Para ejecutar dichos flujos sobre los productos se ha creado *ProcessorManager*, el cual gestiona su procesado. Mediante una factoría de procesadores, se carga el procesador correspondiente para cada tipo de producto. Esto se hizo así para que, si en un futuro se añaden productos que no sean del Copérnico, la aplicación sea capaz de seguir funcionando sin realizar cambios en el código (ver ilustración 76). Para su identificación tenemos una enumeración de los diferentes tipos de productos, y la clase *ProcessingConfiguration,* que proporciona un mapa asociando cada tipo de producto con su procesador.

```java
private BufferedImage processProduct(ProductDTO product, List<String> areasOfWork, WorkflowDTO workflow,
                                     String path, boolean bufferedImage) throws Exception {
    Runtime.getRuntime().gc();
    return getProcessor(product).process(product, areasOfWork, workflow, path, bufferedImage);
}
```

Ilustración 76 - Método para llamar a un procesador según el producto

Para procesar los productos del Copérnico, se ha desarrollado la clase *SentinelProcessor*, ya nombrada anteriormente, y que cuenta con los métodos de las librerías del SNAP. El *ProcessorManager*, tras cargar el procesador adecuado, le inyecta el producto, junto a un flujo y unas áreas de trabajo (ambos obtenidas a partir de la lista de productos).

Antes del procesado se comprueba que, efectivamente, el producto está descargado y que existe un flujo de trabajo asignado. Si por un casual no lo estuviera, entonces el procesador obtendría el tipo de producto y cargaría su flujo de trabajo por defecto (ver ilustración 77).

```java
public BufferedImage process(ProductDTO product, List<String> areasOfWork, WorkflowDTO workflow, String path,
                             boolean generateBufferedImage) throws Exception {

    if (!FileUtils.productExists(product.getTitle())) {
        logger.atError().log( message: "File {}.zip doesn't exists",product.getTitle());
        return null;
    }

    if (hasNotAreaOfWorkReturn(areasOfWork)) {
        logger.atError().log( message: "No area of work assigned to product {}", product.getTitle());
        return null;
    }

    if (hasNotWorkflowLoadDefault(workflow)) {
        if (noDefaultWorkflow(product))
            throw new NoWorkflowFoundException("No workflow found for product type "+product.getProductType());

        logger.atInfo().log( message: "Loaded default Workflow for {} products",product.getProductType());
        workflow = getWorkflow(WorkflowType.valueOf(product.getProductType()));
    }

    logger.atInfo().log( message: "====== Processing product {}",product.getTitle());
    BufferedImage bufferedImage = startProcess(product, areasOfWork, workflow, path, generateBufferedImage);
    logger.atInfo().log("====== Processing ended! =========");
    return bufferedImage;
}
```

Ilustración 77 - Comprobaciones antes de comenzar con el procesado de un producto

Tras comprobar que todos los datos son correctos, empieza el procesado. Por cada operación del flujo de trabajo, se va creando un resultado utilizando la librería de SNAP, asignando los parámetros de cada operación. Es necesario ir pasándole el resultado de una operación a la siguiente.

El diagrama de secuencia de estas interacciones sería el mostrado en la ilustración 78:

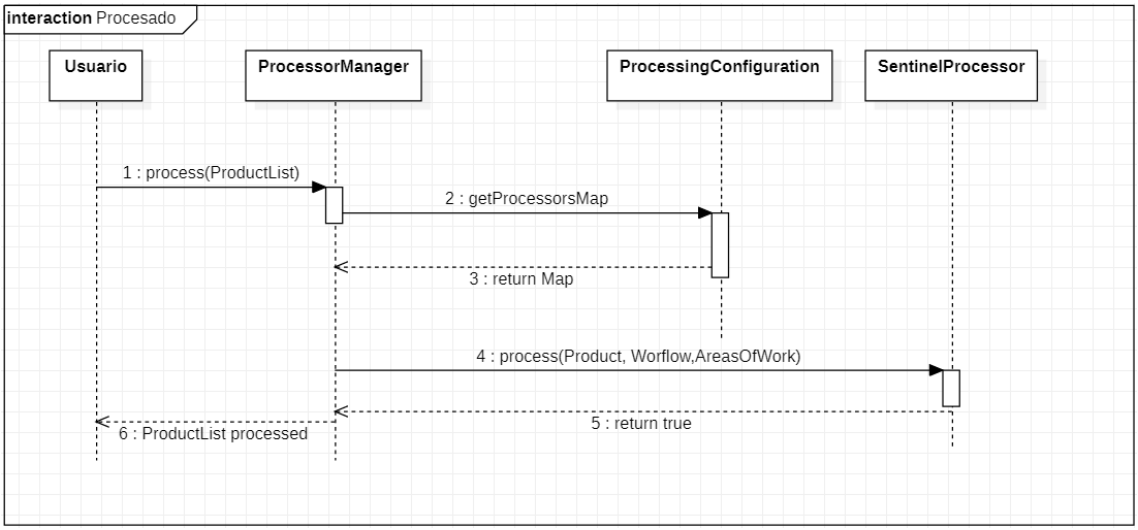


Ilustración 78 – Diagrama de secuencia del procesado de una lista

Para tener una forma visual del progreso del procesado, se ha desarrollado una vista que incluye tres barras de progreso. Una nos indica el número de productos procesados de una lista; el segundo las operaciones que se han ejecutado durante el procesado de un producto (por ejemplo, en productos de tipo GRD, son un total de 7 operaciones); y la última el progreso de la operación que se está ejecutando en ese momento. Estas imágenes resultantes se almacenan en la carpeta de listas, que por defecto está ubicada en la carpeta de "Documentos" del usuario/a.

También a destacar que, a la vez que se procesan los productos de una lista, también lo hacen las imágenes de referencias. Estas se almacenarán dentro de una carpeta llamada "Reference Images", dentro del directorio que corresponde a una lista en concreto.

Con esto concluido, tras poder procesar un conjunto de productos, se enfocó al desarrollo de la funcionalidad de *Preview*. Siguiendo los consejos de los tutores, se decidió crear esta vista para realizar una prueba visual de sobre cómo quedaría una imagen tras el procesado, con el fin de no tener que esperar al final del procesado de todos los productos para comprobar el resultado.

Para conseguir esto, se ha utilizado una opción que brinda las librerías del SNAP para obtener una imagen durante el procesado de un producto. Sin embargo, el único problema es que estas librerías están pensadas para funcionar con interfaces Swing, así que dicha imagen la devuelve como imagen de Swing. Para evitar esto, se hace uso de otra librería que ofrece la propia JavaFX para obtener una imagen de JavaFX (y poder guardarla) de una imagen de Swing.

Su funcionamiento es básico, dada una lista de productos, el usuario/a selecciona uno de ellos para realizar el procesado previo. Además, tiene que seleccionar un área de trabajo que esté contenida dentro de dicha imagen. Tras esto, se abrirá una nueva pestaña donde se mostrarán dos elementos: un mapa y un visualizador de imagen. El mapa contiene una "rejilla" roja, que el usuario/a podrá mover para seleccionar dentro del área de trabajo una pequeña zona donde realizar la muestra (ver ilustración 79). Y, para disminuir todavía más el tiempo de procesado, se ha reducido el número de bandas de salida a solo una, también elegida por el usuario/a. utilizando una nueva utilidad llamada *ProductBands*. Esta clase obtiene las bandas de salida del procesado a partir de un producto y su flujo de trabajo. Se ha implementado de tal forma que cada tipo de producto obtenga sus bandas de salida, mediante de uso de clases específicas para

cada uno de ellos. Así aseguramos que, si en un futuro se añade un nuevo tipo de producto, pueda ser utilizado por esta clase.

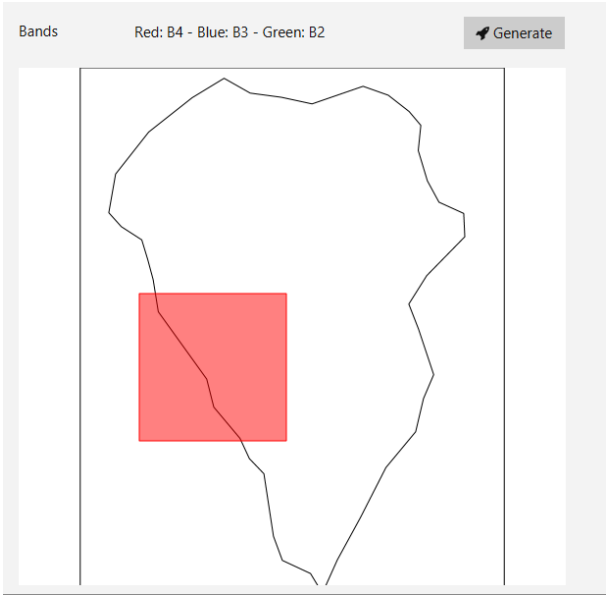


Ilustración 79 - Seleccionar zona dentro de un área de trabajo para realizar una preview del procesado

Finalmente, la última parte del proyecto es el manejo de las imágenes resultantes del procesado. Cuando comenzamos un procesado, se muestra una pestaña de "Resultados del procesado", donde irán apareciendo los diferentes archivos (TIFF o PNG).

Se puede ver el archivo resultante, siempre que sea PNG, puesto que el TIF no contiene una imagen como tal, sino un conjunto de imágenes. También se dado la posibilidad de eliminar archivos desde la anterior vista.

Si es durante el procesado, para actualizar los ficheros que se van procesando, se ha utilizado una librería de Java, llamada *WatchService*, que permite monitorizar un directorio, y mandar un evento cuando ha ocurrido algún cambio dentro de él. Este tipo de comportamiento es perfecto para cumplir los requisitos anteriormente descritos: que las imágenes se vayan mostrando según se van procesando.

Finalmente, para este sprint también se han añadido las funcionalidades de editar un perfil de usuario/a (cambiar contraseña), así como la posibilidad de eliminar la cuenta de la aplicación.

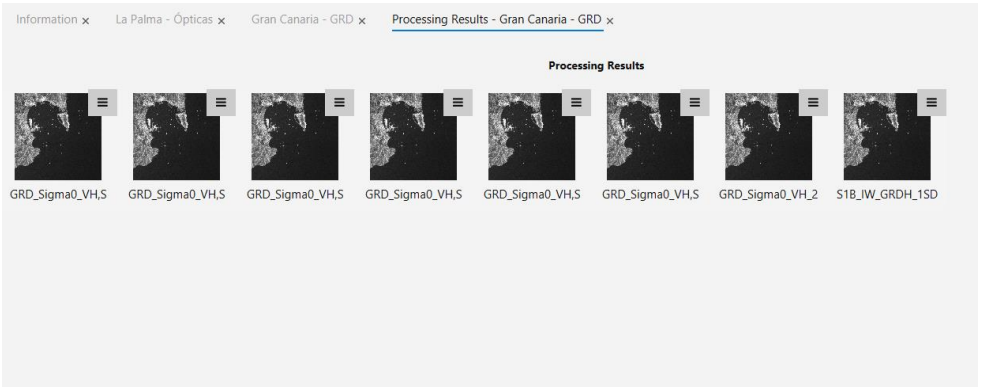


Ilustración 80 - Vista de las imágenes procesadas de una colección de productos

12.3.6 Sprint 5 - Corrección de errores y retoques finales

Se ha implementado el patrón Observer para los ítems del menú, que requieren que haya un tipo de pestaña determinada activa. Así pues, cada vez que se cambia una pestaña, se llama a los observadores, actualizando su estado.

Por ejemplo, para los menús de añadir productos a una lista, se necesita que haya una pestaña de búsqueda activa. Por lo tanto, si no lo hay, estas opciones deberían estar desactivas. Con el patrón Observer conseguimos dicho comportamiento. Si no es una pestaña de búsqueda, se desactiva (ver ilustración 81).

```
public void update(Object args) {
    Platform.runLater(()->{
        deselect.setDisable(!(args instanceof ProductListTabItem));
        workflows.setDisable(!(args instanceof ProductListTabItem));
        addSel.setDisable(!(args instanceof SearchController<?>));
        addAll.setDisable(!(args instanceof SearchController<?>));
    });

}
```

Ilustración 81 - Clase ListMenuItem, que implementa la interfaz Observer, y que es llamada por el componente que se ha modificado

Otro aspecto que se ha implementado es la opción de rehacer y deshacer, por medio de la interfaz *ModificableTabItem*, que tiene métodos para realizar esas operaciones, y que están vinculados con el menú “Edit” (ver ilustración 82).

```
public interface ModifiableTabItem {
    void undo();
    void redo();
    String getRedo();
    String getUndo();
}
```

Ilustración 82 - Interfaz ModificableTabItem

Por ejemplo, *ListController* implementa esta interfaz, permitiendo estas operaciones a la hora de crear áreas de trabajo, borrar, añadir y borrar productos e imágenes de referencia.

Para actualizar la interfaz tras la modificación, creación o eliminado de algún elemento en la aplicación, se ha diseñado una serie de eventos. Estos eventos, denominados *ComponentEvent*, son llamados por un componente (o hijo) tras haber realizado alguna acción, indicando sobre qué objeto se ha hecho dicha modificación (por ejemplo, sobre Listas). Esto hace que, dependiendo de sobre qué objeto que se ha modificado, se actualice un componente u otro.

Un ejemplo es, si se ha creado una lista, se ha de llamar al componente *ListTreeViewComponent*, para que se actualice y muestre la nueva que se ha creado. Para eso, lanza un evento mediante un método definido en la interfaz *Component*, llamado *fireEvent*().

Respecto a las pruebas, se han creado una batería para los controladores utilizando la librería TestFX, que permite que se ejecute un hilo de JavaFX para poder mostrar las vistas, y realizar pruebas sobre ellas (ver ilustración 83).

```java
@Test
public void get_parameters() {
    GRDDefaultWorkflowDTO workflow = new GRDDefaultWorkflowDTO();

    interact(() -> {
        controller.setParameters(workflow.getOperation(Operator.TERRAIN_CORRECTION).getParameters());
    });

    clickOn( query: "#demName");
    type(KeyCode.DOWN);
    type(KeyCode.ENTER);

    Map<String, Object> parameters = controller.getParameters();
    assertThat(parameters.get("demName")).isEqualTo("SRTM 1Sec HGT");
    assertThat(parameters.get("demResamplingMethod")).isEqualTo("NEAREST_NEIGHBOUR");
    assertThat(parameters.get("imgResamplingMethod")).isEqualTo("NEAREST_NEIGHBOUR");
}
```

Ilustración 83 - Pruebas utilizando la librería TestFX

También se ha comprobado que la aplicación cumple con los casos de uso, tanto en su flujo normal como los de error. También se ha validado entregando la aplicación a los tutores para su uso y testeo.

A destacar además la clase *UserManager*, que nos proporciona algunos métodos para facilitar el acceso a la base de datos para algunas operaciones, como, por ejemplo, añadir o eliminar flujos de trabajo. Finalmente, el diagrama de las clases más importantes del modelado y de la interfaz se encuentran en el Anexo II y III.

Como último paso, se convirtió el JAR resultante de la aplicación a un ejecutable, utilizando la aplicación descrita en las tecnologías usadas. Después de esto, se creó un instalador personalizado con el programa Inno Setup.

13 Normativa y legislación vigente

Para cualquier proyecto software hoy en día es necesario comprobar y verificar que nuestra aplicación con las leyes hoy en día vigentes, tanto en el ámbito nacional como en el de la Unión Europea.

En el ámbito nacional, la ley que establecía los reglamentos respecto a la protección de datos ha sido sustituida por el Reglamento General de Protección de Datos. Dicho reglamento, el más importante ahora mismo a nivel europeo, fue aprobada por la Unión Europea en 2016, y es aplicada en todos los territorios que la componen. Dicho reglamento entró en vigor en España en 2018, y enuncia una serie de normas que buscan otorgar de nuevos derechos a la sociedad respecto a la gestión y el control de sus datos personales.

Algunos de estos principios son los siguientes enumerados:

- **Transparencia de la información, comunicación y modalidades de ejercicio de los derechos del interesado:** Ofrecer al ciudadano/a una redacción clara y precisa sobre el uso que se le van a dar a sus datos y con qué fin.
- **Derecho al olvido:** Permite al ciudadano/a poder eliminar sus datos personales de una red social, aplicación o buscador web.
- **Portabilidad de los datos:** Permite al ciudadano/a poder trasladar sus datos personales de un proveedor de internet a otro.

Esta aplicación, al no tratar con datos personales del usuario/a, no necesita implementar estas series de reglamentos. Sin embargo, se ha añadido la opción de permitir borrar los datos de un usuario/a de la aplicación.

Respecto al uso de credenciales de terceros, como, por ejemplo, las de la API del Copérnico, esta aplicación no almacena ni gestiona dichas credenciales. Su único uso es la correcta identificación del usuario/a en las API.

14 Conclusiones y trabajos futuros

14.1 Resultados finales

Tras realizar todos los pasos para obtener una imagen procesada, se ha utilizado a modo de ejemplo una serie de algoritmos, proporcionados por mis tutores, para demostrar como resultado final de las imágenes satelitales. Estas operaciones de post-procesado consisten en aplicar filtros para reducir el contenido de ruido en las imágenes. Se aplicaron dos filtros muy empleados en este campo de trabajo: el de mediana y el de difusión anisotrópica. El resultado se muestra en las ilustraciones 84 y 83. Con esto simplemente se quiere mostrar que la herramienta diseñada da cabida de forma sencilla a post-procesados, bien sea vía Matlab u otras aplicaciones. En este caso los filtros se han implementado en C++ y convertidos a ejecutables y lanzados fuera de la herramienta objeto de este TFG.

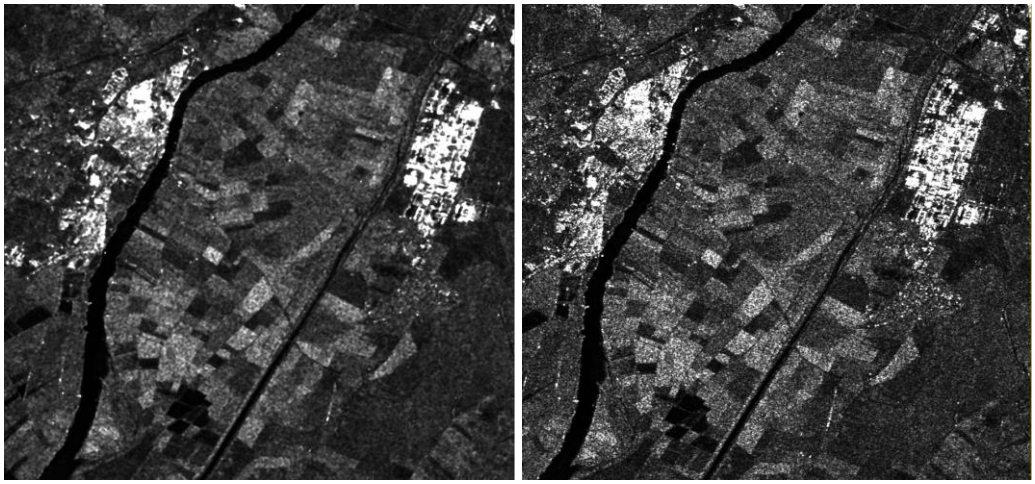


Ilustración 84 - Izquierda, imagen SAR aplicando filtro de mediana; Derecha, imagen sin aplicar filtro

Ilustración 85 - Izquierda, imagen SAR aplicando filtro de difusión anisotrópica; Derecha, imagen sin aplicar filtro

14.2 Conclusiones y trabajos futuros

Tras desarrollar esta primera versión de la aplicación, resulta claro que todavía me queda un gran recorrido en cuanto a adquisición de conocimiento respecto a desarrollo de aplicaciones software de escritorio. Tengo bastante experiencia en el desarrollo web, y en otros entornos, pero para este proyecto partía de lo que había aprendido durante la carrera, con la complejidad de que no se utilizaba JavaFX, sino Swing.

Pese a ser una pequeña aplicación, la gran parte del tiempo la he invertido en cómo hacer las cosas en vez de directamente a ponerme a hacerlas. Con este proyecto me he dado cuenta de la gran importancia de tener conocimientos sobre cómo desarrollar una aplicación software en sí, cómo dividir los diferentes componentes, qué responsabilidades debe tener cada uno y cómo comunicar cambios al resto.

Dichas preguntas son las que me han aparecido durante la gran mayoría de días que he estado desarrollando la aplicación, y con el tiempo, se van respondiendo, pero no en su totalidad. Quizás volver a este nivel de desarrollo de interfaz, sin ningún tipo de ayuda respecto a controlar eventos, me permita mejorar la velocidad de desarrollo con otras aplicaciones que si cuenten con ello, gracias a la experiencia de tener que crear estos controladores por mi cuenta.

Respecto al campo del procesado de imágenes, he hecho grandes avances desde mis prácticas externas. Contar con esta aplicación, que permite obtener las imágenes y poder usarlas en poco tiempo es bastante reconfortante.

Y que además esto es simplemente un punto de partida. Tal como está implementada la aplicación, está lista para añadir nuevos tipos de productos, nuevos portales de obtención de imágenes, nuevos modos de procesado, etc.

Otra posible mejora es explotar la base de datos en MongoDB. Se podría crear una aplicación móvil que se centre únicamente en la búsqueda y guardado de productos en listas. Con búsquedas automáticas, avisar al usuario/a que se ha añadido un nuevo producto en una zona que él ha marcado como área de interés, o incluso con un conjunto de filtros.

También se podría incorporar el módulo de post-procesado, que, sobre las imágenes resultantes, se ejecute una serie de algoritmos, y obtener las respuestas de éstos.

En definitiva, este proyecto me ha servido para darme cuenta de que es necesario tener una idea principal sobre cómo los diferentes componentes del software van a interaccionar entre ellos, quién tiene que avisar a quién y cómo.

En general, estoy contento con el producto final, al igual que mis tutores, y espero que pueda ser de utilizad para aquellas personas que, ya sea porque no tienen tiempo, o porque no tienen el conocimiento necesario para procesar a un nivel alto imágenes satelitales, encuentren a esta aplicación como la solución perfecta para dichos problemas.

15 Bibliografía

[1] ESA, «El Programa Copérnico,» ESA, [En línea]. Available: https://www.esa.int/Space_in_Member_States/Spain/El_programa_Copernico.

[2] MatWorks, «Matlab,» [En línea]. Available: https://es.mathworks.com/products/matlab.html.

[3] ESA, «ESA Snap,» [En línea]. Available: https://step.esa.int/main/toolboxes/snap/.

[4] «Gisgeography,» 4 10 2020. [En línea]. Available: https://gisgeography.com/multispectral-vs-hyperspectral-imagery-explained/.

[5] A. M. M. O. I. C. C. M. Biescas E., «APLICACIONES DE LA INTERFEROMETRÍA SAR PARA LA MEDIDA DE,» *Instituto de Geomática,* pp. 1-7, 2005.

[6] NASA LandSat Science, «LandSat Science,» [En línea]. Available: https://landsat.gsfc.nasa.gov/.

[7] Agencia Espacial Europea, «ESA Sentinel 2,» [En línea]. Available: https://www.esa.int/Space_in_Member_States/Spain/SENTINEL_2.

[8] Asociación Geoinnova, «Asociación Geoinnova,» 2018. [En línea]. Available: https://geoinnova.org/blog-territorio/gis-aplicacion-snap-imagenes-aereas/.

[9] Maven, «Apache Maven,» [En línea]. Available: https://maven.apache.org/.

[10] Oracle, «Oracle Java 8 Documentation,» [En línea]. Available: https://docs.oracle.com/javase/8/docs/api/overview-summary.html.

[11] Oracle, «Oracle JavaFX 8,» 2014. [En línea]. Available: https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm.

[12] GeoTools, «GeoTools,» [En línea]. Available: https://geotools.org/.

[13] ESA Senbox-org, «Github Senbox-org,» 15 10 2020. [En línea]. Available: https://github.com/senbox-org/.

[14] Morphia, «Morphia,» [En línea]. Available: https://morphia.dev/1.6.0/.

[15] «FasterXML Jackson,» [En línea]. Available: https://github.com/FasterXML/jackson.

[16] GitHub, «ControlsFC,» [En línea]. Available: https://github.com/controlsfx/controlsfx.

[17] Pixelduke, «Pixelduke - JavaFX Theme Jmetro,» [En línea]. Available: https://pixelduke.com/java-javafx-theme-jmetro/.

[18] GitHub, «GitHub JFoenix,» 2020. [En línea]. Available: https://github.com/jfoenixadmin/JFoenix.

[19] J. Deters, «FontawesomeFX,» [En línea]. Available: https://bitbucket.org/Jerady/fontawesomefx.

[20] Tutorialspoint, «MVC Framework Introduction,» [En línea]. Available: https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm.

[21] Agile Manifesto, «Agile Manifesto,» 2001. [En línea]. Available: https://agilemanifesto.org/.

[22] Baeldung , «Baeldung Jackson,» [En línea]. Available: https://www.baeldung.com/jackson.

[23] Agencia Espacial Europea, «Documentación Snap-Engine,» [En línea]. Available: http://step.esa.int/docs/v6.0/apidoc/engine/index.html.

[24] F. Filipponi, «Sentinel-1 GRD Preprocessing Workflow,» *Multidisciplinary Digital Publishing Institute Proceedings,* pp. Vol. 18, No. 1, p. 11, 2019.

[25] Github Senbox Org, «Github Senbox Org S1TBX,» 2020. [En línea]. Available: https://github.com/senbox-org/s1tbx.

[26] D. V. D. S. B. N. R. V. V. S. K. K. V. R. Y. S. &. B. A. Mandal, «Sentinel-1 SLC preprocessing workflow for polarimetric applications: a generic practice for generating dual-pol covariance matrix elements in SNAP S-1 toolbox,» 2019.

[27] Github Senbox Org, «Github Senbox Org S2TBX,» 2020. [En línea]. Available: https://github.com/senbox-org/s2tbx.

[28] «Centre for Remote-Image Sensing & Processing,» 2001. [En línea]. Available: https://crisp.nus.edu.sg/~research/tutorial/optical.htm.

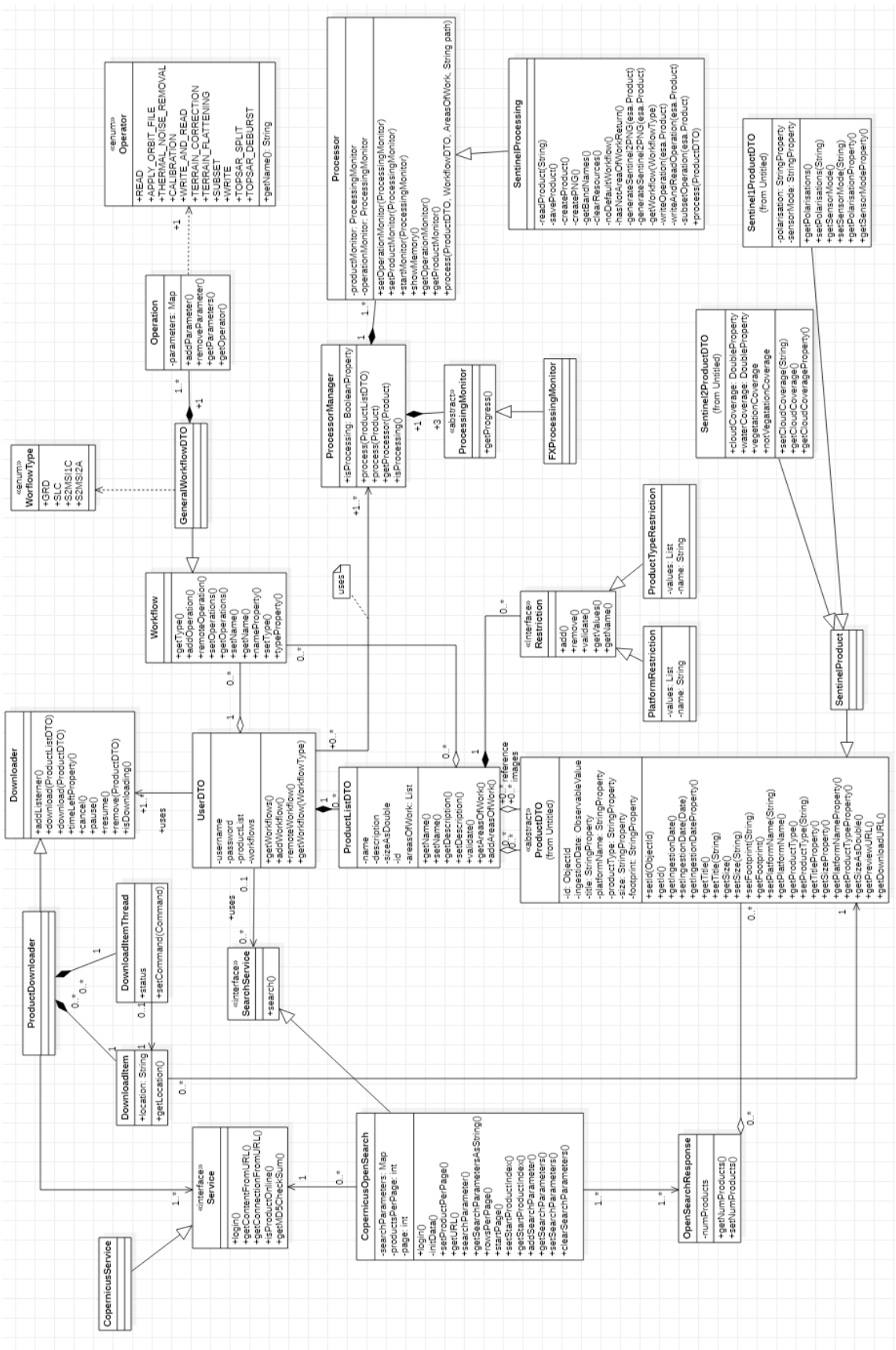
[29] E. A. M. M. O. I. C. &. C. M. Biescas, «Aplicaciones de la interferometría SAR para la medida de deformaciones del terreno,» *Instituto de Geomática, Castelldefels, Espana.,* 2003.

16 Anexos

16.1 Anexo I: Operaciones y parámetros de procesado

- **Apply Orbit File**: Utilizado para obtener los datos de órbita del satélite en el momento de adquisición. Su objetivo es georreferenciar el producto.
    - **Orbit Type**: Tipo de archivo de órbita
    - **PolyDegree:** Grado del polinomio
- **ThermalNoiseRemoval:** Reducción del ruido térmico sobre la imagen.
- **Calibration:** Calibración de la imagen
    - **Output beta:** Obtener una banda utilizando en ángulo de incidencia Beta.
    - **Output sigma:** Obtener una banda utilizando en ángulo de incidencia Sigma.
    - **Output gamma:** Obtener una banda utilizando en ángulo de incidencia Gamma.
    - **Output dB:** Obtener la salida en DB.
    - **Output complex:** Obtener las bandas complejas (parte real e imaginaria)
- **TerrainCorrection:** Corrección de terreno con la utilización de mapas de elevación
    - **Digital Elevation Model**: Mapa de elevación digital disponibles en procesar una imagen
    - **DEM Resampling Method**: Método de reensamblado para obtener la elevación del DEM.
    - **Image Resampling Method**: Método de reensamblado para obtener la elevación del DEM.
- **Write:** Formato de salida, normalmente en TIFF.
    - **Format:** Formato de salido o escritura
- **TopSARSplit**: Elección del área a trabajar
    - **Swath**: Elección de uno de los 3 Swath disponibles
    - **First index 1-9:** Primer Deburst en los que se divide un Swath.
    - **Last index 1-9:** Último Deburst.
- **TopSarDeburst**: Se utiliza para eliminar las divisiones entre los 3 Swath que cuenta la imagen.
- **Resampling**: Obtener la misma resolución espacial para todas las bandas del producto (imagen multiespectral).
    - **Banda**: Seleccionar banda como referencia para el remuestreo.
    - **Resolution**: Seleccionar resolución como referencia para el remuestreo
    - **Upsampling**: Método para pasar de resoluciones altas a bajas
    - **Downsampling**: Método para pasar de bajas resoluciones a altas
- **TerainFlattenning:** Resolución de errores creados por la topografía de la superficie
    - **Digital Elevation Model**: Mapa de elevación digital disponibles en procesar una imagen.
    - **DEM resampling Method:** Método de reensamblado para obtener la elevación del DEM.
    - **Overlap Percentage**: Porcentaje para obtener pixeles adyacentes para corregir los errores causados por la topografía.
    - **Oversampling Multiple:** Remuestreo del mapa de elevación, por si no está a la misma resolución de la imagen.

16.2 Anexo II: Diagrama de las clases más importantes del modelado



16.3 Anexo III: Diagrama de las principales clases de la interfaz