UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA INFORMÁTICA

Trabajo de Fin de Grado

Herramienta para la detección de contenido oculto en redes P2P

Autor: Adrián Alonso González
Tutor: Jorge Blasco Alís

Septiembre de 2012

Agradecimientos

Justo ahora no puedo evitar acordarme de todo el esfuerzo y trabajo dedicado a lo largo de los cuatro años que ha durado mi formación universitaria. Ha sido un camino difícil y lleno de obstáculos pero gracias a ciertas personas he podido seguir adelante y este es el momento de agradecérselo.

En primer lugar quiero agradecerle a mi familia todo el apoyo que me han dado, ya que siempre han estado ahí para preguntarme cómo me iba todo, para ayudarme en esos momentos difíciles y para animarme a seguir adelante. Gracias a mis abuelos, a mis tíos, a mis primos y sobre todo a mi madre, sin ellos no sería lo que soy.

Tampoco puedo olvidarme de mis amigos, los cuales me han aportado muchísimo en multitud de facetas de mi vida y a día de hoy lo siguen haciendo. Sin ellos habría tirado la toalla en muchas ocasiones pero siempre conseguían sacarme una sonrisa y animarme para que siguiera adelante. Todos y cada uno de ellos son importantísimos para mí pero hay una persona sin la cual, mi vida no sería igual. A esa persona le quiero dar las gracias por muchos motivos, gracias por preocuparte de como avanzaba con el proyecto, gracias por aguantar mis etapas de agobio y mis frustraciones, gracias por animarme en los malos momentos, pero sobre todo, gracias por ser como eres.

También me gustaría agradecerle a mis compañeros de universidad los buenos momentos pasados, las cosas que aprendí de a ellos y a que hayan hecho más llevaderos estos cuatro años. No podría olvidarme además de dar las gracias a todos los profesores que he tenido a lo largo de la carrera, gracias a todos y cada uno de ellos he podido aprender muchas cosas y en definitiva formarme en lo que siempre había querido.

Por último dar las gracias a mi tutor, Jorge por todo el apoyo que me ha brindado. Gracias a su guía hemos conseguido que este proyecto haya sido un éxito y que todo el esfuerzo invertido haya valido la pena, sin ti nada de esto habría sido posible.

Gracias a todos y lo siento si me he dejado alguien.

Termina una etapa de mi vida que siempre recordaré con mucho cariño pero empieza otra que seguro será igual o más enriquecedora. Mi futuro comienza hoy.

Índice

Índice de Ilustraciones

Índice de tablas

Índice de casos de uso

Índice de requisitos

Índice de pruebas

Abstract

This end of degree project is based on two key parts, steganography and P2P networks. The term steganography comes from the Greek words *steganos*, which means hidden, and *graphos*, which means writing. Therefore, the term, literally, can be interpreted as hidden writing. From a more general point of view the term steganography can be interpreted as the study of the set of techniques and processes that enable to hide information embedding it in other information. Informally, steganography supplies the necessary tools to hide messages inside other messages (which only are supposed to be visible for the message's recipient).

Although it may seem as a recent technique, steganography is a discipline used long since. Throughout history there have been many documented ways of exchanging messages hiding its existence, here are just a few:

- Approximately toward the year 440 B.C, Aristágoras of Mileto (tyrant of Mileto's city), used one of his slaves to provoke a riot against the Persian. The way to exchange the message was the following: first, Aristagoras shaved the head of one of his slaves and tattooed him the message that wanted to send in the slave's scalp. Done that, he just needed to wait for the slave's hair to grow back. When this happened, he sent the slave to meet the receiver of the message. Shaving the slave´s head he was able to access it.
- During Middle Age in Europe a method consisting in superimposing a template (which used to be paper or wood), called Cardan grille and which contained a series of holes on a write was developed.
- During World War II some rather curious methods were developed to hide messages. One of them was to use the characters "j", "i", "t" and "f" to send messages in Morse code within a message, written on a microfilm, which was apparently normal. As can be observed the characters "j" and "i" could be used to transmit the "short beeps" message, while the characters "f" and "t" could be used to transmit the "long beep" message.
- Another method that was developed during the World War II was possible thanks to the technological advances in photography. Thanks to these advances, the micropoints technology was developed. This technology allows reducing a photograph to a point size so that adding these images in a text is possible. If these images are hidden as punctuation marks (points, commas ...), they are almost undetectable.
- Finally, among the most widely used methods throughout history are invisible inks. Text messages can be written with these inks. To have access to these messages is necessary to heat the paper support on which the message is written. The most common inks used can be: lemon juice, milk, urine, ammonia salt... in general, substances containing large amounts of carbon in its composition.

Today, thanks to the information revolution that involves digital images, digital video or digital audio, among other digital content, steganography techniques have also been adapted to these changes. These changes make the hidden information to be more difficult to perceive. In addition to the multimedia content, there are other kinds of files that may contain hidden information, such as: executable files, websites, network packet fields, unused disk space, etc.

Before describing steganographic techniques, the following terms, which are often used, are going to be defined:

- **Object carrier:** This term refers to the object that is used to contain the hidden message: an image, audio, text, video, an executable file, etc.
- **Stego file:** This term refers to the assembly formed by the object carrier and the message object hidden in it.

Among the most widely used techniques for generating stego files are the following:

- **Addition:** In this case, the message is hidden in portions of the object carrier that will be ignored during its processing. An example of this method is a comment inserted into the HTML code of a web page. The web browser does not interpret this HTML, so it will not be displayed on the screen to the user. The main disadvantage of this method is that it increases the size of the object carrier and this can arouse suspicion if it is intercepted.
- **Generation:** In this case, the object carrier is created from the message that is going to be hidden in it, so it is not necessary to previously have an object carrier.
- **Replacement:** In this case, object carrier parts are modified by changing them for secret message parts. For example in steganography with images, the stego file can be built in several ways. The simplest is to use the least significant bits of each pixel of a digital image or, if it is an audio file, the least significant bits of each byte that forms the file. Thus the variation in the image is minimal and it is impossible to detect these changes visually.

Once the use of steganography discipline throughout history and different steganographic techniques have been analyzed, the potential of steganography to hide information can be understood. Furthermore steganography can be combined with cryptography, making it much harder to recover the hidden message.

The other key part of this project are P2P networks. A P2P network or peer-to-peer network is a computer network, in which there are usually no fixed clients or servers, but the computers connected to the network or nodes may act as the two roles, client and server. P2P networks, among other advantages, have the characteristic that typically optimize the bandwidth exploiting the connectivity between users. Using P2P networks any type of information can be exchanged so they have many applications, but the most interesting for this project is file sharing. These applications arise because of the need of the users to exchange information with other users. They, using a P2P application for this purpose, register on the network the files that they want to share identifying them by their title, length, format or other properties so that the other users can find them. This is

the most widespread application of this network type being BitTorrent, eDonkey2000 or Ares the most used protocols.

P2P networks typically do not have great security and facilitate the anonymity of the authors of the content that they store. These features make these networks the ideal medium for sharing fraudulent files. This joined to that, as mentioned above, steganography is an ideal technique to hide information are the main motivations for this project, the development of a tool for detecting hidden content in P2P networks.

The planned and achieved goals for the application developed in this project are:

- **Define a search base criteria to download files being shared on the Internet:** The application is able to search for files containing hidden information in various P2P networks according to different search criteria. The application can search using as criteria data such as the file size, its hash function value, etc. The aim of the search will be to find files that potentially contain hidden information by steganographic techniques.
- **Connect to a P2P network or a website:** The application is able to connect to a P2P network to choose from one of the main used networks nowadays such as eDonkey, BitTorrent, Gnutella, Ares, etc. Finally, it would be a desirable characteristic that the application could connect to a web page.
- **Download content based on the search criteria:** The application is able to connect to a P2P as was described in the second point and according to the criteria defined in the first point it is able to download files.
- **Run detectors on downloaded content:** As mentioned above, downloaded files can have hidden information with steganographic techniques. The application is able to run a set of detectors on the downloaded files to find out if those files actually contain hidden information. The application also can run these detectors during the file download process. The process of detecting this information is called steganalysis.
- **Execute actions according to the result of the detectors:** The application is able to execute a set of actions based on the results generated by the detectors. An example of action that could make the application would be that, in case of detecting fraudulent hidden information in a downloaded file, this information tries to be revealed making various attacks.
- **Write a report on the results of the detectors and the actions taken:** The application is able to write a report as a text file with the result of the execution of the detectors on a file and actions were executed according to the result of these detectors. A report details, among other things, if hidden information has been detected in the file, it includes this information, it indicates which steganography technique has been used and it includes the result of actions that have been executed. Reports can also be generated in PDF format. Continuing with the example discussed in the previous section a possible report would be a text file that reports that hidden information has been detected in a file, that the information found is a plain text file, and that as a result of the detection of this information a bruteforce attack was performed against the downloaded file and the hidden information was extracted.

- **Easy to adapt design to facilitate the addition of new functionality:** The application is designed so that it will be easy to expand with new features such as the inclusion of new search criteria, new P2P networks, new detectors, new actions, etc. To accomplish this, the application takes the form of modules such as the connection module, the search module, the detection module, the actuation module, etc. These modules are able to be easily extended with new functionality.

As a result of the project an application, called Stegocrawler, was developed. This application, built on top of some of the libraries of Vuze, meets all the above objectives. The Stegocrawler application is able to connect to the BitTorrent network to find content and download them. Among the detectors which can be executed over downloaded contents, there are functions, that using steganalysis techniques, can detect if a file contains hidden information. The application also implements actions like bruteforce attacks or dictionary attacks with the aim of extracting this hidden information. Finally the application is able to report, in PDF format or in plain text format, about the results of the detectors and the actions.

Along with the development of the application the present document was developed. Finally, the structure of the present document is described in the following lines:

- **Chapter 1: Introduction.** This chapter introduces the project that was developed. In this chapter the project motivations are detailed, the planned target list is specified and the structure followed in the document is indicated. The development of this chapter was essential, since it served as a contact with the project and to be clear about the project goals.
- **Chapter 2: Analysis.** This chapter is a study of the most important parts of the project. An analysis of P2P networks has been made, explaining how they work, the different ways to classify them and their characteristics. Also the operating mode of two networks in particular has been detailed, the BitTorrent network and the eDonkey network. Then the discipline of steganography has been explained with the different techniques used in it along with examples. Then the concept of steganalysis and the techniques employed in it have been explained. After, a study of the various actions to be implemented by the application has been made. Subsequently different design alternatives have been studied and one of them has been chosen. After, a study of the use cases of the application has been made, showing a use cases diagram and giving details about each of them. The next step is to make an analysis of the requirements both functional and nonfunctional of the application. Also a traceability matrix linking functional requirements with use cases has been constructed. The last step that has been made in the analysis is the development of a test plan. This plan is simply a set of test cases that the application must pass to be accepted. Also next to the test cases catalog a traceability matrix linking test cases and functional requirements has been made. The conclusions at the end of that chapter were very satisfactory as it served to better understand faced problems.

- **Chapter 3: Design.** This chapter details the design aspects of the application. A study of the architecture of the application has been made. Next, the design pattern has been described and the different components of the application have been studied at low level. For each component, the set of classes that compose the component have been described, showing their methods and attributes, and showing a class diagram. Finally sequence diagrams of some operations carried out by the application have been depicted. The realization of the design chapter served to have a clear idea of how the application would be built, which was very useful for its implementation.
- **Chapter 4: Implementation.** In this chapter the project issues related with the implementation of the application have been detailed. The chapter has been divided into two parts: initially, aspects of the implementation of the major functions that the application can perform have been detailed at code level and in the second part the results of the test plan have been shown. The completion of this chapter helped to capture some application errors, and correct them, and to demonstrate the development process that was made.
- **Chapter 5: Project Management.** In this chapter a study of project planning have been done, showing Gantt diagrams about the initial planning and the final planning. Subsequently, the technical resources used in the development of the project, both software and hardware, have been shown and an economic analysis of the project has been made by estimating the initial cost, calculating the real costs and making an analysis of how to sell the application. This chapter was useful for several reasons but mainly because it served to look back and see the difference between the initial and final planning. Analyzing the error in planning will ensure that, in future plannings, this error will be reduced.
- **Chapter 6: Conclusions and future lines.** In this chapter the findings from the project and future lines of work have been exposed. That chapter was useful to analyze the work done, first evaluating the quality of the developed application and secondly analyzing what the performing of the present project meant personally. Besides the development of the future lines section served to have a starting point if, anytime, the project is going to be continued.
- **Appendix A: Application Manual.** In this appendix, a manual for the correct use of the application has been created. Instructions for completion of the various operations that can be executed by the application have been detailed along with screenshots of execution examples of each of them. The development of this appendix served to show certain aspects of the application that were not shown in the rest of the project. In addition it also served as study of the usability of the application (develop an application manual and read it is a great way to check if the application is useful) and helped to capture and to fix bugs and to improve the application usability about certain aspects of the interaction with the user.

Capitulo 1: Introducción

1.1 Motivación del proyecto

El presente proyecto se basa en dos pilares, la esteganografía y las redes P2P. La esteganografía es la ciencia que estudia las técnicas para ocultar información. Ésta, aunque puede ser utilizada para multitud de fines, también puede ser usada por criminales para intercambiar información oculta relacionada con sus crímenes. En (Simmons, 1998) se puede encontrar el planteamiento del problema del prisionero, descrito por G. J. Simmons en 1983. Este problema expone que existen dos prisioneros que únicamente pueden comunicarse a través de mensajes escritos los cuales son revisados por un funcionario de la prisión. Para trazar un plan de fuga, no sería lógico que los mensajes fueran en claro para el funcionario, y si ambos prisioneros cifran los mensajes el funcionario podría sospechar de las intenciones de ambos y cortar la comunicación. Por tanto, los prisioneros deben encontrar una forma de ocultar información relativa a la fuga de la prisión en mensajes que, a ojos del funcionario, parezcan totalmente inofensivos. Como se puede observar este problema describe desde otro punto de vista el objetivo de las técnicas esteganográficas.

A pesar de lo que pueda parecer, la esteganografía es una disciplina empleada desde hace mucho tiempo. Entre las primeras referencias a técnicas que permitían ocultar información se puede destacar la obra *Las Historias* de Heródoto de Halicarnaso. Esta obra está compuesta por un total de nueve libros (disponibles en (Heródoto)) cuyo fin es mostrar una descripción de la historia relativa a las Guerras Médicas y el Antiguo Egipto. A lo largo de la narración de estos libros, se tratan algunas maneras de intercambio de mensajes ocultando su existencia, aunque sólo se citará la siguiente:

- Aproximadamente hacia el año 440 A.C, Aristágoras de Mileto (tirano de la ciudad de Mileto), emplea a uno de sus esclavos para provocar una revuelta contra los persas. La manera de intercambiar el mensaje es la siguiente: en primer lugar, Aristágoras rasuró la cabeza a uno de sus esclavos y tatuó el mensaje que deseaba enviar en el cuero cabelludo del esclavo, hecho esto, sólo le quedaba esperar a que al esclavo le volviera a crecer el cabello. Cuando esto hubo ocurrido, envió al esclavo a encontrarse con el receptor del mensaje, el cual rasurando la cabeza del esclavo podía tener acceso al mismo.

Durante la edad media se desarrolló en Europa un método que consistía en superponer una plantilla (que solía ser de papel o madera), denominada rejilla de Cardano y la cual contenía una serie de orificios, sobre un escrito. Las letras que quedaban bajo los orificios de la plantilla formaban el mensaje. Un método muy similar a este, y también muy utilizado a lo largo de la historia, era practicar pequeños orificios sobre las letras de un escrito. De esta forma, y leyendo el escrito a contraluz, quedaban resaltados los caracteres que conformaban el mensaje. Este método fue utilizado, por ejemplo, por el ejército alemán en la Segunda Guerra Mundial (el texto empleado para

ocultar la información se trataba de un periódico, y, tal como se ha explicado, sobre ciertos caracteres de las noticias que contenía el mismo se practicaban pequeños agujeros).

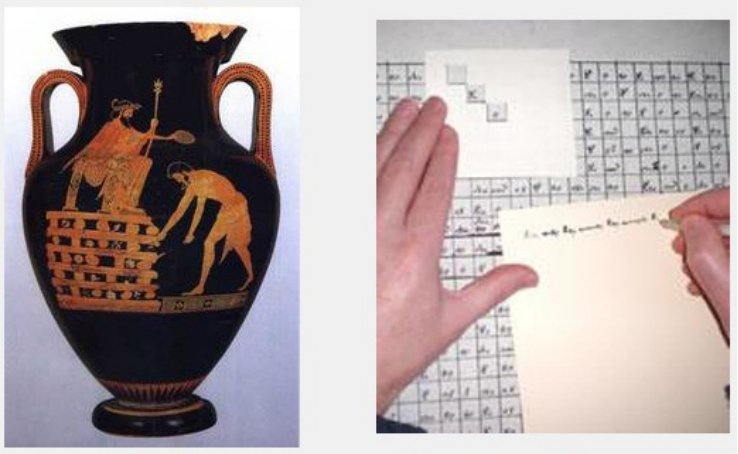


Ilustración 1: A la izquierda representación de la técnica de Aristágoras de tatuar a un esclavo[1] y a la derecha una representación del modo de uso de una rejilla de Cardano[2]

También en la Segunda Guerra Mundial se desarrollaron algunos métodos bastante curiosos para ocultar mensajes:

- Uno de ellos consistía en, dentro de un mensaje, escrito en un microfilm, aparentemente normal, emplear los caracteres "j", "i", "t" y "f" para enviar mensajes en código Morse. Como se puede observar los caracteres "j" e "i" serían empleados para transmitir los "pitidos cortos" del mensaje, mientras que los caracteres "f" y "t" serían empleados para transmitir los "pitidos largos" del mensaje.
- Otro de los métodos desarrollados fue posible por los avances tecnológicos realizados en el campo de la fotografía. Gracias a estos se desarrolló la tecnología de los micropuntos. Ésta permitía reducir una imagen al tamaño de un punto, por lo que era posible añadir estas imágenes en un texto, haciéndolas pasar por signos de puntuación (puntos, comas,…), de manera que pasaban totalmente desapercibidas.

Entre los métodos más utilizados a lo largo de toda la historia se encuentran las tintas invisibles. Con este tipo de tintas se pueden escribir mensajes, y para poder tener acceso a ellos es necesario calentar el soporte de papel sobre el que se escribe el mensaje. Las tintas utilizadas más comunes pueden ser: zumo de limón, leche, orina, sal de amoniaco,…, en general sustancias que contienen grandes cantidades de carbono en su composición.

[1] Imagen obtenida de http://theionianrevolt.weebly.com/synopsis.html
[2] Imagen obtenida de http://www.nature.com/news/2003/031217/full/news031215-5.html

En la actualidad, tras el salto tecnológico que supone el uso de imágenes digitales, vídeo digital o audio digital, entre otros contenidos digitales; las técnicas de esteganografía también se han ido adaptando a estos cambios. Estos cambios hacen que la información oculta en distintos ficheros pase cada vez más desapercibida. Además de los contenidos multimedia, existen gran cantidad de ficheros que pueden contener información oculta, como pueden ser: ficheros ejecutables, páginas web, campos de los paquetes red, espacio de disco no utilizado, etc. Estos son sólo algunos ejemplos, ya que se pueden encontrar muchas otras técnicas.

Una vez se ha analizado el uso de la disciplina de la esteganografía a lo largo de la historia ya se puede alcanzar a comprender el potencial de ésta para ocultar información, pudiéndose combinar además con criptografía, lo que haría mucho más difícil la recuperación del mensaje oculto.

Por otra parte en cuanto a las redes P2P, éstas se empezaron a utilizar de forma general con la aparición de Napster en 1999 (anteriormente existían varios proyectos pero fue dicha red la primera conocida mundialmente). Napster llegó a tener más de 13 millones de usuarios conectados en todo el planeta y su objetivo era la compartición de música. No obstante tras una orden judicial se ordenó su cierre en Julio del 2001, tras esto pasó a ser un servicio de pago y fue condenado al olvido por sus usuarios. Después de Napster apareció AudioGalaxy, otro cliente P2P destinado a compartir música aunque también fue cerrado por orden judicial. En la actualidad AudioGalaxy sigue existiendo aunque ya no como cliente P2P sino como servicio de streaming (véase Glosario - Streaming).

Ambas redes Napster y AudioGalaxy eran de carácter centralizado (más adelante se estudiará que significa esto) y por lo tanto más fáciles de ser controladas. Posteriormente aparecieron las primeras redes P2P totalmente descentralizadas o híbridas como Gnutella, Kazaa, Ares Galaxy, eDonkey2000 o BitTorrent.

Ilustración 2: Evolución de las redes P2P

Estas redes no dependen únicamente de un servidor central y por lo tanto son más difíciles de controlar aunque también ha habido intentos de destruirlas como el cierre de los dos servidores más importantes de eDonkey2000, Razorback 2.0 y Razorback 2.1, por parte de la Motion Picture Association (MPA) o el cierre de multitud de páginas que compartían enlaces torren o enlaces ed2k. Pese estos intentos, todas estas redes siguen funcionando actualmente e incluso hace poco vieron incrementado su número de usuarios debido al cierre del servicio de alojamiento de archivos Megaupload (19 de Enero de 2012) como se muestra en el siguiente gráfico.

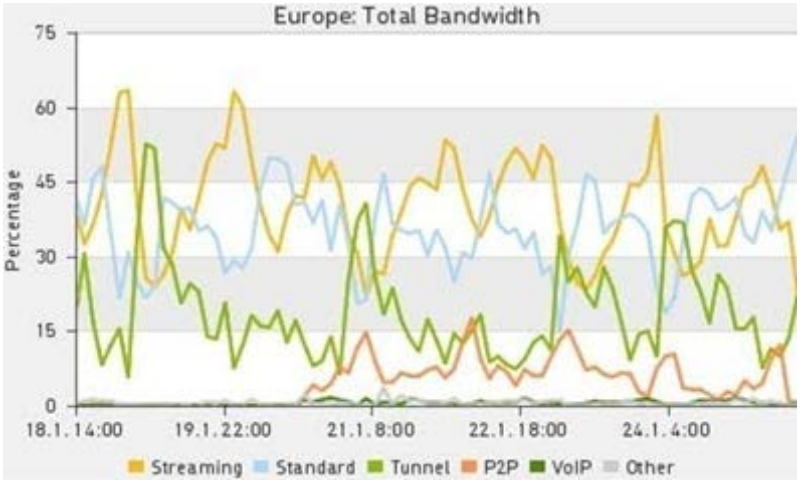


Ilustración 3: Gráfico en el que se observa un aumento del ancho de banda usado en redes P2P a partir del 19 de Enero[3]

Normalmente estas redes no poseen una gran seguridad y facilitan el anonimato de los autores del contenido. Estas características hacen de ellas el medio ideal para la compartición de archivos de carácter fraudulento. Si se une esto al hecho de que tal y como se dijo anteriormente la disciplina de la esteganografía es ideal para ocultar información aparece la principal motivación para la realización de este proyecto, el desarrollo de una herramienta para la detección de contenido oculto en redes P2P. Esta herramienta, la cual se denominará Stegocrawler, será capaz de conectarse a un conjunto de redes P2P (inicialmente sólo será capaz de conectarse a una) y descargar contenidos de la misma detectando aquellos que sean sospechoso de incluir información oculta o tengan características anormales. En el siguiente apartado se detallan los objetivos que debe cumplir la aplicación.

[3] Gráfico propiedad del Internet Observatory

1.2 Objetivos del proyecto

En esta sección se elaborará una lista de objetivos, esta lista sirve para tener claros desde un principio y a lo largo de todo el proyecto una serie de metas que deberá alcanzar la aplicación desarrollada. A continuación se presenta dicha lista de objetivos incluyendo una breve descripción de cada uno de ellos:

- **Definir una base de criterios de búsqueda**: La aplicación deberá ser capaz de buscar archivos con contenido oculto en distintas redes P2P de acuerdo a una serie de criterios de búsqueda. Se podrá buscar mediante criterios como el tamaño del archivo, su función hash, etc. El objetivo de la búsqueda será el de encontrar archivos que potencialmente puedan contener información oculta mediante técnicas esteganográficas.
- **Conectarse a una red P2P o a una web**: La aplicación deberá ser capaz de conectarse a una red P2P a elegir entre las disponibles en el mercado como eDonkey, BitTorrent, Gnutella, Ares, etc. Además se estudiará la posibilidad de que la aplicación pueda conectarse a una página web.
- **Descargar contenido en base a los criterios de búsqueda**: La aplicación se conectará a los medios descritos en el segundo punto y de acuerdo a los criterios de búsqueda definidos en el primer punto será capaz de descargar archivos.
- **Ejecutar detectores sobre el contenido descargado**: Como ya se dijo anteriormente los archivos descargados podrán ser objeto de la ocultación de información mediante técnicas esteganográficas. La aplicación deberá ser capaz de ejecutar una serie de detectores sobre ellos para localizar si dichos archivos realmente contienen información oculta. Además se estudiará la posibilidad de que la aplicación sea capaz de ejecutar estos detectores durante el proceso de descarga del archivo. El proceso de detectar esta información se denomina estegoanálisis.
- **Ejecutar acciones de acuerdo al resultado de los detectores**: La aplicación deberá ser capaz de ejecutar una serie de acciones según el resultado generado por los detectores. Un ejemplo de acción a tomar por la aplicación sería que, en el caso de que se detectara información maligna oculta en un archivo, se iniciara un ataque con el fin de extraer dicha información.
- **Elaborar un informe con el resultado de los detectores y de las acciones llevadas a cabo**: La aplicación deberá ser capaz de elaborar un informe en forma de fichero de texto con el resultado de la ejecución de los detectores sobre un archivo y de las acciones que se llevaron a cabo de acuerdo al resultado de estos detectores. El informe detallará entre otras cosas si se detectó información oculta en el archivo, incluirá dicha información, indicará cual fue la técnica esteganográfica utilizada y el resultado de las acciones que se ejecutaron. Además se estudiará la posibilidad de que el informe pueda ser generado en formato PDF. Siguiendo con el ejemplo desarrollado en el punto anterior un posible informe sería un fichero de texto que informara de que se ha detectado información oculta en el archivo, que la información encontrada se trata de un archivo de texto plano, y que como resultado de la detección de

esta información se lanzó un ataque de fuerza bruta sobre el archivo descargado y se extrajo dicha información oculta.

- **Diseño adecuado para facilitar la adición de nuevas funcionalidades**: La aplicación deberá diseñarse de tal forma que sea fácil añadirle nuevas funcionalidades tales como la inclusión de nuevos criterios de búsqueda, nuevas redes P2P, nuevos detectores, nuevas acciones, etc. Para lograr esto la aplicación se desarrollará en forma de módulos como por ejemplo el módulo de conexión, el módulo de búsqueda, el módulo de detección, el módulo de actuación, etc. Estos módulos tendrán la capacidad de ampliarse fácilmente con nueva funcionalidad.
- **Ejecutar una prueba de la aplicación sobre una red P2P**: Finalmente, la aplicación deberá ser probada sobre una red P2P y deberá funcionar de manera correcta, comprobándose que se cumplen todos los objetivos marcados.

1.3 Estructura del documento

El presente documento está estructurado de la siguiente manera:

- **Capítulo 1: Introducción.** En este capítulo se introduce el proyecto que se va a realizar detallando las motivaciones para la realización del mismo, especificando la lista de objetivos que se deben cumplir e indicando la estructura que seguirá el presente documento.
- **Capítulo 2: Análisis.** En este capítulo se realiza un estudio de los distintos pilares que forman el proyecto:
    - Se explica el funcionamiento de las redes P2P centrándose en algunas en concreto.
    - Se detalla en qué consiste la disciplina de la esteganografía y los métodos más empleados
    - Se estudia la disciplina del estegoanálisis y las distintas técnicas utilizadas.
    - Se realiza un estudio de las diferentes acciones que puede llevar a cabo la aplicación una vez que se ha detectado información oculta en algún archivo descargado.
    - Se muestran distintas alternativas para el futuro diseño de la aplicación.

    Finalmente se especifica el diagrama de casos de uso de la aplicación y se muestra el catálogo de requisitos y de pruebas de aceptación.

- **Capítulo 3: Diseño.** En este capítulo se detalla cómo se realizó el proceso de diseño de la aplicación. Se realiza un análisis de la arquitectura de la aplicación, se muestra el patrón de diseño que se siguió, se estudian a bajo nivel los diferentes componentes que forman la aplicación y finalmente se muestran diagramas de secuencia de algunas operaciones que puede ejecutar la aplicación.
- **Capítulo 4: Implementación.** En este capítulo del proyecto se profundiza en temas relacionados con la implementación de la aplicación desarrollada. El

capítulo se divide en dos partes: inicialmente, se detallan a nivel de código aspectos de la implementación de las principales funciones que puede realizar la aplicación y en la segunda parte se muestran los resultados del plan de pruebas realizado en el capítulo de análisis.

- **Capítulo 5: Gestión del proyecto.** En este capítulo se realiza un estudio de la planificación del proyecto mostrándose diagramas de Gantt tanto de la planificación inicial como de la que finalmente se llevó a cabo. A continuación se muestran los medios técnicos empleados en el desarrollo del proyecto, tanto software como hardware, y se realiza un análisis económico del mismo mediante una estimación de costes inicial, un cálculo de los costes reales y un análisis de la forma de venta de la aplicación.
- **Capítulo 6: Conclusiones y líneas futuras.** En este capítulo se exponen las conclusiones extraídas de la realización del proyecto y las líneas futuras por las que éste puede continuar.
- **Anexo A: Manual de la aplicación.** En este anexo se desarrolla un manual del correcto uso de la aplicación. Se detallan instrucciones para la ejecución de las distintas operaciones que puede llevar a cabo la aplicación junto con capturas de ejemplos de ejecución de cada una de ellas.
- **Anexo B: Glosario.** En este anexo se incluye una lista con definiciones de algunos términos aparecidos a lo largo del proyecto.

Capitulo 2: Análisis

En este capítulo se realiza un estudio de los distintos pilares que forman el proyecto:

- Se explica el funcionamiento de las redes P2P centrándose en algunas en concreto.
- Se detalla en qué consiste la disciplina de la esteganografía y los métodos más empleados
- Se estudia la disciplina del estegoanálisis y las distintas técnicas utilizadas.
- Se realiza un estudio de las diferentes acciones que puede llevar a cabo la aplicación una vez que se ha detectado información oculta en algún archivo descargado.
- Se muestran distintas alternativas para el futuro diseño de la aplicación.

Finalmente se especifica el diagrama de casos de uso de la aplicación y se muestra el catálogo de requisitos y de pruebas de aceptación.

2.1 Estado de la Cuestión

Lo primero que se va a realizar en esta fase del proyecto es una descripción amplia de diversos conceptos claves en el proyecto. Primeramente, se describirá en qué consiste una red P2P, los distintos tipos de redes según las distintas arquitecturas que implementan y se analizará su seguridad. Además se analizarán distintos protocolos P2P y se realizará un estudio de dos protocolos en concreto, el protocolo BitTorrent y el protocolo eDonkey2000. Posteriormente se explicará en qué consiste la esteganografía y el estegoanálisis y las diversas técnicas empleadas. Finalmente se definirán las posibles acciones a realizar sobre los resultados obtenidos de la búsqueda.

2.1.1 Redes P2P[4]

Una red P2P o red de pares es una red de ordenadores en las que normalmente no hay clientes ni servidores fijos, sino que los ordenadores conectados a la red o nodos[5] pueden actuar según ambos roles, cliente y servidor. Las redes P2P entre otras ventajas tienen la característica de que normalmente el ancho de banda se optimiza aprovechando la conectividad de los usuarios como se explicará más adelante. Mediante las redes P2P

[4] Referencias utilizadas: (Buford, y otros, 2008) , (Steinmetz) y (Ayudabittorrent).

[5] A lo largo de todo el documento se utilizará indistintamente los términos nodo, usuario u ordenador conectado a una red para designar el mismo concepto.

se puede intercambiar información de cualquier tipo por lo que tienen multitud de aplicaciones, entre las cuales destacan las siguientes:

- **Intercambio de archivos:** Surgen debido a la necesidad de los usurarios de intercambiar información con otros usuarios. Éstos utilizando una aplicación P2P para este propósito registran en la red los archivos que desean compartir identificándolos por su título, longitud, formato u otras propiedades de manera que el resto de usuarios pueda buscarlos. Esta es la aplicación más extendida de este tipo de redes siendo los protocolos más utilizados BitTorrent, eDonkey2000 y Ares entre otros.
- **Streaming y telefonía sobre internet o VoIP:** Las primeras aplicaciones de este tipo empiezan a aparecer a mediados de los años 90 y ofrecen a los usuarios un servicio para transmitir voz o videollamadas de ordenador a ordenador. Algunas aplicaciones como Skype o Spotify están construidas sobre protocolos P2P. Además existen las aplicaciones conocidas como P2PTV que se utilizan para compartir video en streaming en tiempo real mediante redes P2P. Un ejemplo de este tipo de aplicaciones es la web Zatto[6].
- **Sistemas de ficheros distribuidos:** Como por ejemplo la red de distribución de información Freenet.
- **Aplicaciones o servicios que necesiten gran cantidad de recursos:** Cualquier aplicación que realice cálculos científicos o matemáticos que necesiten procesar una gran cantidad de datos.

Entre las características que se buscan a la hora de desarrollar redes P2P se encuentran las siguientes:

- **Escalabilidad:** Como se ha dicho anteriormente las redes P2P normalmente optimizan el ancho de banda. Estas redes normalmente son usadas por millones de usuarios y cuando estos usuarios se conectan comparten sus recursos de manera que a medida que van conectándose más usuarios los recursos totales de la red aumentan. Esto es contrario a lo que ocurre en las arquitecturas cliente-servidor típicas en las que los recursos provienen de un número fijo de servidores y cuantos más clientes se conectan a ellos menos recursos se disponen para cada uno de ellos.
- **Robustez:** Debido a la naturaleza distribuida de estas redes se aumenta su robustez ya que no hay un punto crítico que en caso de fallar echara abajo el sistema, como si puede ocurrir en las redes cliente-servidor.
- **Descentralización:** No hay ningún elemento imprescindible en el sistema. Como se verá más adelante esto no se cumple en todas las redes P2P. Aunque la descentralización a priori parece muy buena ya que le permite a la red ser escalable, anónima y robusta, no proporciona algunas ventajas que si pueden llegar a proporcionar las redes centralizadas, sobre todo ventajas económicas. Se puede pensar en ejemplos tales como la red social Facebook que al utilizar una arquitectura centralizada como es la arquitectura de cliente-servidor les

[6] www.zattoo.com/

dan la posibilidad a sus propietarios de ganar dinero mediante publicidad y gestionar la información de los usuarios de la red social con fines económicos .

- **Distribución de costes entre los usuarios:** Cada usuario comparte recursos a cambio de otros recursos. Aparecen los términos de *leechers* y *seeders* de los que se hablará más adelante.
- **Anonimato:** Se desea que exista la posibilidad de que tanto el autor del contenido como el que lo descarga o el servidor que lo alberga sean anónimo. El usuario no necesita registrarse en ningún sitio y por lo tanto no hay una entidad que albergue ningún dato personal nuestro, como ocurre en cualquier página web de descarga. Existen proyectos de redes sociales sobre P2P como PeerBook (Lazalde, 2010) o Diaspora (November, 2010) que pretenden aprovechar esta característica de privacidad así como otras características de las redes P2P.
- **Seguridad:** Aunque debido a su naturaleza anónima estas redes son idóneas para propagar archivos infectados desgraciadamente esta es la característica menos implementada. Además existen otros riesgos como ataques de denegación de servicio (véase Glosario - Ataque de denegación de servicio) o la compartición de información sensible. En ocasiones hay usuarios poco informados o descuidados que comparten información sensible sin su conocimiento debido a una mala configuración de las carpetas compartidas. Entre otros mecanismos que podrían asegurar las redes P2P estarían la identificación de nodos sospechosos, archivos infectados, intervención de las comunicaciones entre nodos, cifrado de las comunicaciones, sandboxing (véase Glosario - Sandboxing). No obstante hay algunos mecanismos que algunos clientes P2P implementan. Entre estos mecanismos se encuentra el empleo de funciones resumen o hash (véase Glosario - Función resumen) de manera que se mantenga la integridad de la información así como sistemas de comentarios que, aunque no son mecanismos de seguridad propiamente dichos, en ocasiones permiten conocer si un archivo está infectado o no, aunque no ofrecen un diagnóstico seguro (ALMALASI, 2008).

Las redes P2P deben enfrentarse a dos problemas principales, estos problemas son provocados porque la mayoría de los nodos en internet no tienen una dirección IP pública fija, es el caso de los nodos que se conectan a través de redes de área local (LAN), detrás de un cortafuegos o NAT (véase Glosario - NAT), o que tienen contratada con su ISP (véase Glosario - ISP) una dirección IP dinámica (la mayoría de los casos). A continuación se describen los dos problemas:

- **Encontrar un nodo que ya está conectado a la red P2P:** Para solucionar este problema se emplea un servidor intermediario o proxy con una dirección IP fija y conocida por el cliente P2P. Este servidor mantiene una lista con todas las direcciones de los nodos que se han conectado, de manera que tras esto los nodos ya están en disposición de intercambiar información con otros nodos sin la intervención del servidor.
- **Conectar dos nodos sin dirección IP pública entre ellos:** Para resolver este problema se utiliza un enfoque parecido al anterior, se emplea un tercer nodo con dirección pública que haga de intermediario entre los dos y envíe la información que le llega de uno al otro. Aquí surge un problema de seguridad y

es que ese nodo intermediario podría captar todo el flujo de información entre los dos nodos que se están comunicando por lo que es necesario implementar mecanismos de seguridad como el cifrado de la información.

2.1.1.1 Clasificación de redes P2P

A continuación se va a hacer una clasificación de las redes P2P de acuerdo a diferentes criterios. Según su grado de centralización las redes P2P se pueden clasificar en:

- **Redes P2P centralizadas:** En este tipo de redes todas las transacciones de información pasan a través de un único servidor, el cual distribuye en qué nodos se almacenan los contenidos. En este tipo de redes el servidor central mantiene un índice de los archivos que hay en la red de manera que cuando un usuario desea descargar un archivo deberá acceder al servidor central el cual consultará el índice y le dirá a que nodo debe conectarse para descargar el archivo. Son las equivalentes a la arquitectura cliente-servidor típica y al igual que ésta tienen la ventaja de que son fáciles de administrar y entre otras desventajas la pérdida de privacidad, una peor escalabilidad, cuellos de botella en el servidor central, existencia de un único punto de fallo, etc. Ejemplos de este tipo de redes son la ya desaparecida Napster y AudioGalaxy.
- **Redes P2P puras:** En estas redes no se necesita la existencia de un servidor central que gestione las comunicaciones y distribuya los contenidos sino que los nodos se comunican directamente entre sí, con ayuda de un nodo intermediario que hace posible la comunicación tal y como se dijo anteriormente, y son los propios nodos los que almacenan la información, existiendo un índice de archivos distribuido. Cuando un usuario realiza una petición para descargar un archivo ésta se va propagando a través de los nodos vecinos (se denominan nodos vecinos a los nodos sobre los que se tiene información) hasta encontrar el archivo deseado, esto se suele implementar mediante tablas de enrutado. Algunos ejemplos de estas redes son Kadenlia, Ares Galaxy, Gnutella, Gnutella 2 y Freenet.
- **Redes P2P híbridas:** Este tipo de redes como su nombre indica son una fusión de los otros dos tipos e intentan obtener las ventajas de cada uno de ellos. Existe un servidor central que al igual que en las redes P2P centralizadas distribuye en qué nodos se almacenan los contenidos y el ancho de banda pero sin saber la identidad de éstos ni almacenar ningún tipo de información. Además en caso de que el servidor o servidores centrales se caigan los nodos pueden seguir intercambiando información entre pares sin la intervención del servidor al igual que en las redes P2P puras. Ejemplos de este tipo de redes son BitTorrent, eDonkey y Direct Connect.

También se pueden clasificar las redes P2P según su estructura, es decir cómo se enlazan unos nodos con otros o cómo se organiza el contenido:

- **No estructuradas:** Se habla de redes no estructuradas cuando los enlaces entre los nodos se forman sin seguir ningún criterio. Al no haber una estructura

marcada cuando un usuario realiza una petición de información esta petición va propagándose entre los nodos y sus vecinos a lo largo de toda la red para recorrer el máximo posible de usuarios, de esta forma las búsquedas de información más popular tendrán más probabilidades de éxito que las búsquedas de información menos popular. Debido a esta inundación (flooding) de tráfico necesaria para encontrar información la eficacia de las búsquedas se ve reducida, aunque se han implementado distintas mejoras a este tipo de redes como un sistema de jerarquías que introducen nodos especiales conocidos como supernodos y que realizan funciones de servidores para agilizar las búsquedas. Ejemplos de redes no estructuradas son Napster, Kazaa y Gnutella.

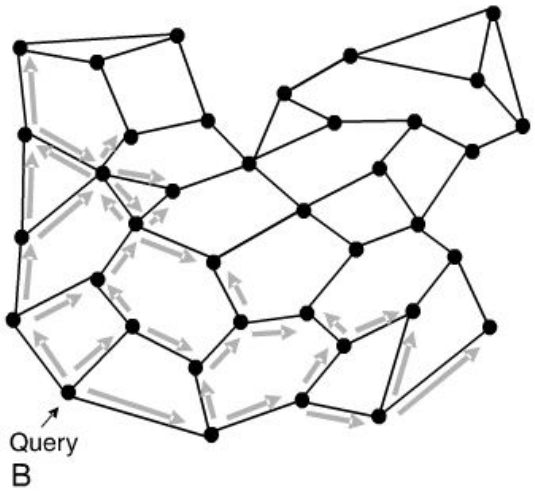


Ilustración 4: Representación de la propagación de las búsquedas en redes no estructuradas[7]

- **Estructuradas:** Estas redes se crearon para paliar la ineficiencia en las búsquedas de las redes no estructuradas. En ellas los nodos forman una capa de red virtual de manera que los nodos se organicen en función del contenido que compartan, es decir, un nodo estará más cerca de otro cuanto más contenidos en común compartan. Una subclase de este tipos de redes son las que implementan tablas de hash distribuidas o DHT (véase Glosario - Tabla de hash distribuida) que se basan en pares de clave-valor. Las claves se computan mediante funciones hash para que todas sean números enteros en el mismo rango. Se valen de un protocolo de manera que cuando un usuario realiza una petición de información se utiliza dicho protocolo para determinar qué nodo posee dicha información y se le conecta con él. Ejemplos de protocolos de redes P2P de búsqueda distribuida son Chord y Pastry P2P Network, además algunos clientes de la red BitTorrent implementan una extensión, que actúa

[7] Imagen obtenida de (Buford, y otros, 2008)

cuando el intermediario que conecta los nodos (denominado Tracker en esta red) se cae, que emplea una DHT.

2.1.1.2 Ejemplos de redes P2P

A continuación se detallan distintos aspectos de las redes P2P BitTorrent y eDonkey2000.

2.1.1.2.1 BitTorrent

BitTorrent es uno de los protocolos de intercambio de archivos sobre P2P más usados en internet, fue diseñado por el programador Bran Cohen en abril de 2001 y se publicó su primera implementación en julio del mismo año. En este protocolo al descargar un archivo, en lugar de descargarlo desde un único servidor, BitTorrent permite a los clientes unirse en grupos de nodos formando un enjambre (swarm en inglés) para descargar y subir el archivo de forma simultánea. A medida que va creciendo el enjambre mayor es la probabilidad de que cualquier nodo pueda descargar el archivo completo. Además ya que la tarea de distribuir un archivo es compartida, el distribuidor original del archivo puede reducir sus recursos de ancho de banda y es más fácil garantizar su anonimato.

Un usuario que quiere subir un archivo, inicialmente crea un archivo *torrent*, el cual contiene la información necesaria para que se pueda descargar, la información más importante que contiene es la dirección del tracker al que el interesado en descargarse el archivo debe conectarse, más adelante se explicará qué son los trackers. Una vez creado el archivo torrent el usuario lo distribuye por los medios convencionales (páginas web, correo electrónico…). A continuación hace que el archivo esté disponible en la red a través de un nodo BitTorrent que actúa como semilla (véase Glosario - Seeder). Los usuarios que quieren descargar el archivo, obtienen el archivo *torrent* y crean otro nodo BitTorrent que actúa como cliente o "sanguijuela" (véase Glosario - Leecher), a continuación establecen una conexión HTTP con el tracker y éste les proporciona una la lista de usuarios que tienen partes (pieces) del archivo a descargar de manera que finalmente comienzan a intercambiar dichas partes con la semilla y con otros clientes, mediante sockets TCP o UDP.

Cuando un usuario descarga una parte puede comenzar a compartirla y convertirse en seeder, liberando así al seed original de tener que enviar una copia de esa parte al resto de usuarios interesados en ese archivo. De esta manera, tal y como se ha dicho anteriormente, la tarea de distribuir un archivo es compartida por todos los nodos interesados en él. Además las partes normalmente no se descargan de manera secuencial, es el cliente BitTorrent el encargado de comprobar que partes ya ha descargado y cuales le faltan por descargar y de ordenarlas. Todas las partes del archivo son del mismo tamaño y son transmitidas de una sola vez. Gracias a este mecanismo se puede detener la descarga de un archivo en cualquier momento y reanudarla posteriormente sin perder la

información previa descargada. Además ya que, como se ha dicho anteriormente las partes no tienen por qué descargarse de manera secuencial, el cliente puede buscar las partes disponibles para descargar fácilmente, sin necesidad de pausar la descarga y esperar que la siguiente parte esté disponible, lo que reduce el tiempo total de descarga.

El protocolo BitTorrent implementa distintos mecanismos para aumentar la homogeneidad y el rendimiento en la descarga de las piezas. Uno de los algoritmos que implementa es el algoritmo de la pieza más rara primero, este algoritmo aprovecha que cada nodo conoce las piezas que ya han descargado sus vecinos y descarga las piezas que se han descargado con menos frecuencia con el objetivo de maximizar la probabilidad de que sus vecinos descargan piezas de él. También se utiliza el algoritmo de bloqueo que consiste en que cada nodo (al que se denominará nodo local), únicamente puede tener un número arbitrario de conexiones activas (normalmente cuatro) y el resto de nodos interesados en compartir información con él (los que se denominarán nodos remotos) son bloqueados. De manera periódica se va comprobando el ratio de descargas proporcionadas al nodo local por los nodos remotos y se elige a los tres nodos con mayor ratio de descargas. Finalmente se elige aleatoriamente y también de forma periódica un cuarto nodo para conectarse de manera que se les permite a los nodos que aún no han descargado ninguna pieza (y por lo tanto aun no pueden compartir nada) empezar la descarga y además se les da una segunda oportunidad a los nodos bloqueados. Con el uso de este algoritmo se penaliza a los nodos que únicamente descargan y no suben nada.

Como mecanismos de integración cada archivo torrent incorpora un hash criptográfico de manera que cualquier modificación sobre alguna parte de un archivo pueda ser detectada y evitar que dichas partes modificadas sean recibidas por el resto de nodos. Un nodo con el archivo torrent completo puede verificar la autenticidad del archivo completo descargado ya que dentro del archivo torrent se codifica un hash de cada una de las piezas de dicho archivo.

Los clientes de BitTorrent más utilizados son Vuze, anteriormente conocido como Azureus y µTorrent. El primero está escrito en Java y destaca por su gran cantidad de opciones y por ser libre y multiplataforma. El segundo está escrito en C++, también es de licencia libre y entre otras cosas destaca por su buen rendimiento.

2.1.1.2.1.1 Estructura

A continuación se muestra una ilustración que representa la estructura que se sigue en una red BitTorrent.

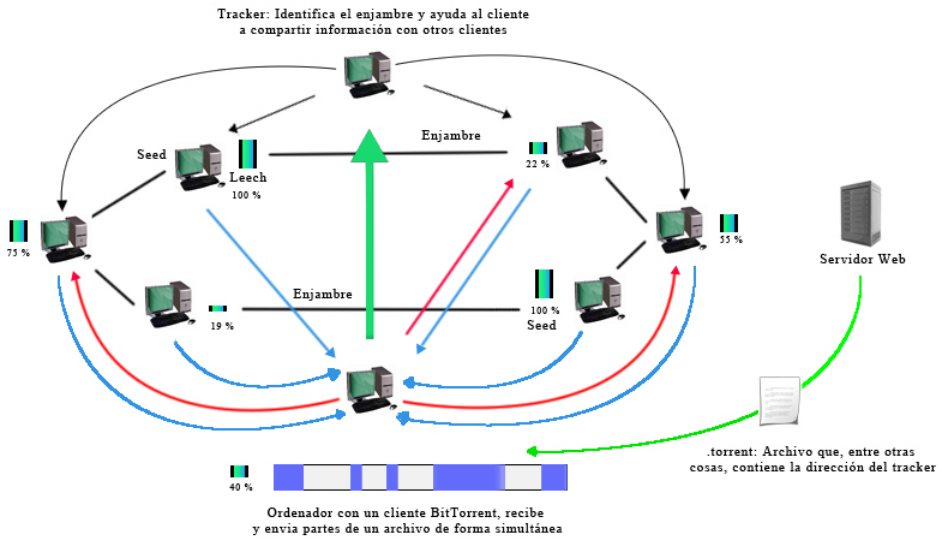


Ilustración 5: Estructura de una red BitTorrent[8]

- **Peers o pares:** Se denomina así a todos los usuarios de la red, tanto seeders como leechers.
- **Seeders o semillas:** Son los usuarios de la red que tienen el archivo completo y por lo tanto lo comparten con los demás.
- **Leechers o sanguijuelas:** Son los usuarios de la red que se encuentran descargando un archivo y no lo tienen aún completo.
- **Trackers:** Son tipos de servidores especiales que contienen la información necesaria para que los peers se conecten entre sí y puedan compartir información. Localiza qué usuarios contienen el archivo que se quiere descargar y les proporciona la lista de dichos usuarios a los que están interesados en descargar dicho archivo.
- **Swarm o enjambre:** Conjunto de usuarios que el Tracker se encarga de buscar. Normalmente los pares se unen a un enjambre para descargar un archivo y le abandonan al poco de habérselo descargado. Se denominan así por su parecido con la forma de comportarse de las abejas.

2.1.1.2.2 EDonkey2000

EDonkey2000 es un protocolo de intercambio de archivos sobre P2P desarrollado por la compañía MetaMachine en septiembre del año 2000. Al igual que la mayoría de redes de intercambio de archivos es una red descentralizada ya que los archivos no se almacenan sino que se intercambian directamente entre usuarios, no obstante existen servidores que se encargan de manejar la información que se transmite por la red por lo que se puede hablar de una red semicentralizada. Cuenta con un sistema de

[8] Imagen basada en una ilustración obtenida de http://es.wikipedia.org/wiki/Archivo:Red_bittorrent.jpg

identificación de archivos basado en hash que utiliza el algoritmo MD4. Este sistema trata los archivos con el mismo contenido pero distinto nombre como el mismo archivo y los archivos con el mismo nombre y distinto contenido como archivos distintos. Cada archivo se divide en partes iguales y por cada una de ellas se realiza un hash de manera que si hay un error de transmisión se puede detectar cual es la parte afectada.

Los clientes eDonkey se conectan a la red para compartir archivos, los servidores eDonkey por su parte actúan como centros de comunicación para los clientes, permitiendo a los usuarios localizar archivos en la red.

- **Software servidor:** El software más usado como servidor en las redes eDonkey2000 es el conocido como servidor Lugdunum. Este servidor fue creado mediante ingeniería inversa y del protocolo eDonkey2000 y posteriormente rediseñado desde cero. Posteriormente se abandonó el desarrollo de este servidor y apareció un nuevo tipo de software de servidor en 2007 conocido como Satan-eDonkey-Server. Este servidor también fue creado mediante ingeniería inversa y está disponible en java y en C++. Ambos servidores son gratuitos.
- **Software cliente:** Existen números clientes que implementan la red eDonkey2000 como por ejemplo eMule, eMule Plus, aMule, JMule, eDonkey2000, IMule, etc.

Los clientes y servidores están disponibles para Windows, Mac OS X, Linux y otros sistemas operativos Unix. Mediante la ejecución de un programa servidor de eDonkey en una máquina conectada a Internet, cualquier usuario puede añadir un servidor a la red. Debido a que el número de servidores y sus direcciones cambian con frecuencia, los programas clientes deben actualizar sus listas de servidores regularmente.

La mecánica de funcionamiento de la red, explicada de manera muy sencilla, es la siguiente:

- **Handshake:** Cuando un cliente se conecta con un servidor le manda una serie de datos, entre ellos información de los archivos que comparte.
- **Búsqueda:** Cuando un cliente busca un archivo en la red, le envía una petición al servidor y éste se encarga de buscar el archivo en la información que los clientes proporcionada durante el handshake. Además el servidor reenvía la petición de búsqueda a otros servidores y éstos realizan el mismo proceso.
- Cuando la búsqueda finaliza con éxito el servidor se encarga de establecer una conexión entre los dos clientes y comienza la descarga del archivo.

El cliente oficial que implementaba este protocolo era eDonkey2000 pero en 2005 quedó descontinuado debido a una denuncia de la RIIA (Recording Industry Association of America) por derechos de autor. El cliente más extendido actualmente que usa la red eDonkey2000 es Emule, el cual añade además una serie de mejoras. Algunos ejemplos de extensiones del protocolo eD2k son el intercambio de pares entre los clientes, la ofuscación del protocolo (que permite evitar la discriminación de los ISPs al uso de programas P2P) y el soporte para archivos mayores de 4 Gigabytes.

2.1.2 Esteganografía[9]

El término "esteganografía" proviene de los vocablos griegos *steganos*, que significa encubierto (oculto), y *graphos*, que significa escribir. Por tanto, el término, de manera literal, puede ser interpretado como "escritura oculta". Desde un punto de vista más general, el término esteganografía es interpretado como el estudio de un conjunto de técnicas y procesos que permiten ocultar información anexándola a otra información. Dicho de modo más coloquial, la esteganografía provee las herramientas necesarias para esconder mensajes (los cuales sólo serán visibles para el receptor del mensaje) dentro de otros mensajes.

A partir de esta definición, pueden surgir ciertas confusiones a la hora de distinguir este término del término "criptografía". Sin embargo, se trata de dos disciplinas totalmente distintas: mientras que la criptografía centra su objetivo en ocultar el contenido del mensaje intercambiado entre un emisor y un receptor, la esteganografía centra su objetivo en ocultar la presencia del mensaje en sí. De esta forma, si un mensaje, al que se le han aplicado técnicas criptográficas (como por ejemplo un cifrado de clave simétrica o un cifrado de clave asimétrica), es interceptado, se conoce de forma inmediata que el mensaje contiene información que el emisor y el receptor no desean que personas ajenas tengan acceso a ella. Sin embargo, si ese mismo mensaje ha sido incorporado, por ejemplo, en los bits menos significativos de cada píxel de una imagen mediante la aplicación de técnicas esteganográficas y dicho mensaje es interceptado, a simple vista (si la técnica aplicada es lo suficientemente buena) no será posible distinguir si ese mensaje contiene más información de la que se muestra a simple vista. A pesar de tratarse de dos disciplinas distintas, la esteganografía y la criptografía pueden ser empleadas de forma complementaria: es posible ocultar un mensaje cifrado, dentro de otro mensaje que no lo esté (siguiendo con el ejemplo anterior), como una imagen.

Antes de comenzar a describir técnicas esteganográficas se van a definir los siguientes términos, los cuales se emplearán a menudo:

- **Objeto portador:** Con este término se hace referencia al objeto que se utilizará para contener el mensaje oculto: una imagen, un audio, un texto, un vídeo, un fichero ejecutable, etc.
- **Esteganograma:** Con este término se hace referencia al conjunto formado por el objeto portador y el mensaje oculto en el mismo.

2.1.2.1 Técnicas

Entre las técnicas más empleadas para generar esteganogramas se encuentran las siguientes:

- **Adición:** En este caso el mensaje se oculta en partes del objeto portador que serán ignoradas durante el procesamiento del mismo. Un ejemplo de este

[9] Referencias utilizadas: (Wayner, 2008), (INTECO, 2010) y (Alonso González, y otros, 2011)

método es un comentario insertado dentro del código HTML de una página web. El navegador web no interpreta ese tipo de código HTML, por lo que no se mostrará al usuario final por pantalla. El principal inconveniente con el que cuenta este método es que aumenta el tamaño del objeto portador, lo que puede levantar sospechas en caso de que el mismo sea interceptado.

- **Generación:** En este caso el objeto portador es creado a partir del mensaje que se desea ocultar en el mismo, por lo que no es necesario disponer de un objeto portador previamente. En (Wayner, 2011) se puede encontrar un applet que genera a partir de una entrada proporcionada un texto para ocultar el mensaje de la entrada. El resultado se puede ver en la siguiente imagen:

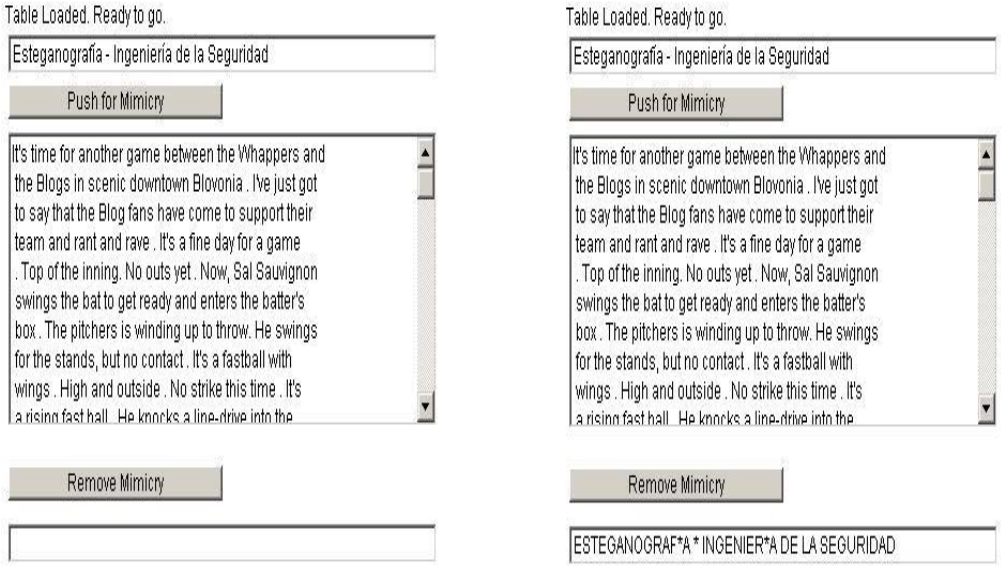


Ilustración 6: Demostración de la técnica de generación[10]

- **Sustitución:** En este caso se modifican partes del objeto portador cambiándolas por partes del mensaje secreto. Por ejemplo en esteganografía en imágenes el esteganograma se puede construir de varias maneras, siendo la más sencilla utilizar los bits menos significativos de cada pixel de una imagen digital o, si se trata de un fichero de audio, los bits menos significativos de cada byte que conforma el fichero. De esta forma la variación en la imagen es mínima y a simple vista resulta imposible detectar dicho cambio. A continuación se plantea un ejemplo:

  Se supone que el mensaje que se quiere ocultar es el siguiente número:

  216

  Se supone también que la imagen contiene los siguientes valores en sus primeros tres píxeles:

  (42 50 71) (218 65 90) (1 255 108)

[10] Imagen obtenida de (Wayner, 2011)

Descomponiendo nuestro mensaje y los bytes de cada píxel de la imagen en bits se obtiene que el mensaje viene dado por

11011000

Y la imagen viene dada por

(00101010 00110010 01000111) (11011010 01000001 01011010) (00000001 11111111 01101100)

Hecho esto, sólo falta sustituir cada LSB (bit menos significativo) de los bytes de la imagen por cada uno de los bits del mensaje, obteniendo el siguiente esteganograma:

(00101011 00110011 01000110) (11011011 01000001 01011010) (00000000 11111110 01101100)

Los nuevos valores de los píxeles con el mensaje oculto son:

(43 51 70) (219 65 90) (0 254 108)

Para que se puedan comprobar las diferencias se facilitan las siguientes figuras:

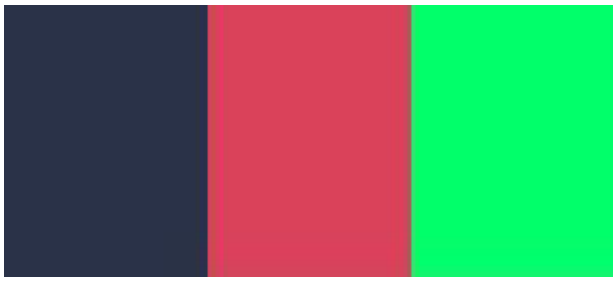


Ilustración 7: Píxeles de la imagen original

Ilustración 8: Píxeles de la imagen con el contenido oculto

2.1.2.2 Generación del esteganograma

Se puede expresar, de manera gráfica, el proceso de obtener (de forma general) un esteganograma de la siguiente forma:

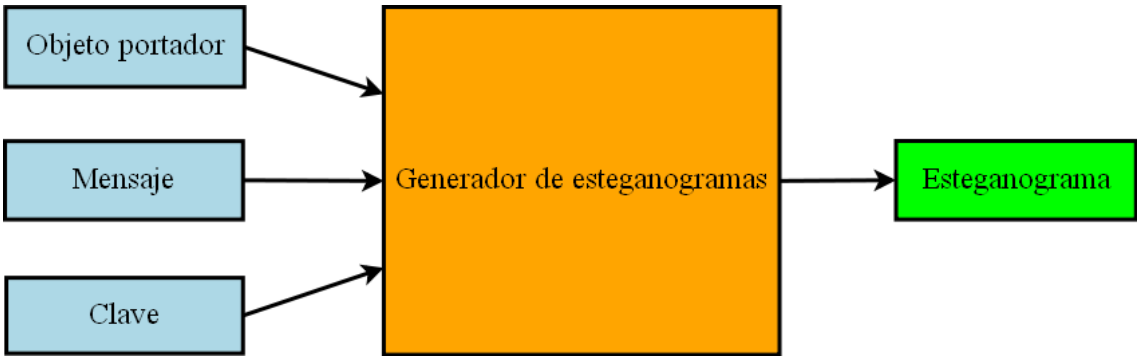


Ilustración 9: Proceso de generación de un esteganograma

En la figura se puede observar que al generador de esteganogramas se le proporcionan un conjunto de entradas (resaltadas de color azul en la figura) que son:

- El objeto portador que contendrá el mensaje oculto.
- El mensaje que se desea ocultar en el objeto portador.
- De manera opcional, una clave, cuya finalidad es la misma que en el caso de las claves utilizadas al generar criptogramas: asegurar que sólo el receptor que disponga de la clave pueda acceder al contenido del mensaje oculto.

El generador de esteganogramas (elemento resaltado en color naranja en la figura), por otra parte, se encarga de calcular la salida, la cual no será más que un esteganograma. Para realizar el cálculo de esta salida se calcula, en primer lugar, los lugares o partes del objeto portador en los que se ubicará el mensaje y, en segundo lugar, se lleva a cabo la inserción de los datos en las posiciones elegidas en el paso anterior.

Parece bastante claro que para que el esteganograma sea aceptable y no revele que oculta un mensaje, debe ser lo más parecido posible al objeto portador, no debe ofrecer indicios de que en el esteganograma existe más información de la que se percibe a simple vista y, sobre todo, el mensaje oculto en el esteganograma sólo debe poder ser recuperado por receptores del mismo que tengan autorización para acceder a él.

2.1.3 Estegoanálisis[11]

Se denomina estegoanálisis a la disciplina encargada del estudio de la detección de mensajes ocultos en fuentes de datos mediante esteganografía. El estegoanálisis es a la esteganografía como el criptoanálisis a la criptografía, sin embargo a diferencia del criptoanálisis donde es necesario descifrar el mensaje para considerar roto un criptosistema en el estegoanálisis basta con demostrar que el archivo contiene

[11] Referencias utilizadas: (Wayner, 2008), (INTECO, 2010) y (Alonso González, y otros, 2011)

información oculta. A continuación se describen los distintos enfoques que suelen ser habituales a la hora de llevar a cabo un estegoanálisis.

Estos enfoques se dividen en:

- Análisis visual
- Análisis estructural
- Análisis estadístico

2.1.3.1 Análisis visual

Éste enfoque es habitual en el estegoanálisis sobre imágenes. Mediante algunas técnicas de análisis se pueden quitar las partes importantes de la imagen de manera que sea posible para un ser humano intentar buscar anomalías visuales. Una prueba común muestra los bits menos significativos de una imagen. La existencia de ruido aleatorio a menudo revela la existencia de un mensaje oculto ya que algunas cámaras, escáneres y otros digitalizadores dejan ecos en los bits menos significativos.

Cuando la imagen es o bien toda blanca o bien toda negra, el bit menos significativo está lejos de ser aleatorio, por lo general, suele ser un cero o un uno. Incluso entonces los tramos no saturados están lejos de ser aleatorios. Las posiciones de los objetos en una imagen y las luces que los iluminan garantizan que a menudo habrá cambios graduales en los colores, estos gradientes también están lejos de ser aleatorios. Además hay también imperfección pura. Las cámaras digitales no siempre cuentan con un total de 24 bits de sensibilidad. El software puede producir imágenes de 24 bits, pero sólo después de rellenar los resultados y añadirles detalles adicionales. Los bits menos significativos no siempre son asignados de forma aleatoria cuando los sensores no tienen la resolución suficiente. Después la información se oculta en los bits menos significativos, aunque, todas estas regiones se vuelven mucho más aleatorias.

El ojo a menudo es la herramienta más rápida para identificar estos cambios. Finalmente se muestra una imagen (obtenida de (Wayner, 2008)) en la que se pueden observar fácilmente los cambios:

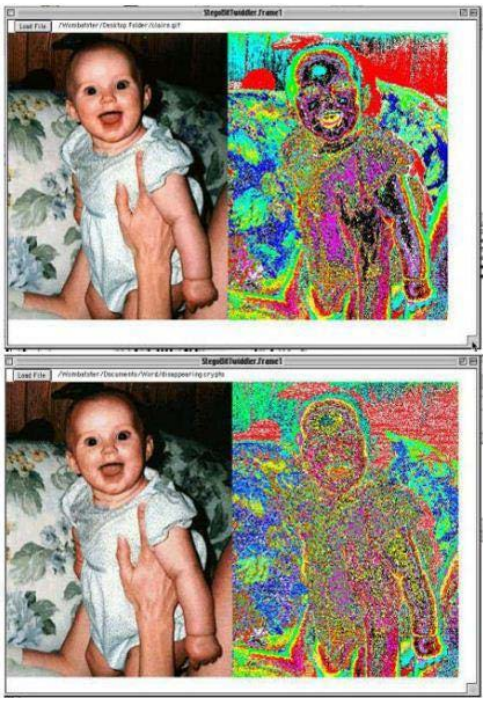


Ilustración 10: La imagen superior muestra una fotografía junto a sus bits más significativos antes de añadirle información oculta. La imagen inferior muestra lo mismo que la anterior después de añadirle información oculta[12]

2.1.3.2 Análisis estructural

El formato del archivo de datos a menudo cambia cuando se le añade información oculta. A menudo, estos cambios pueden ser reducidos a un patrón fácilmente detectable en la estructura de los datos. En algunos casos, los programas de esteganografía utilizan versiones ligeramente diferentes del formato de archivo, lo que les delata. Estos rastros son bastante distintivos y permiten al analizador detectar los algoritmos esteganográficos empleados si se sabe lo que se está buscando. Como se ha dicho anteriormente muchos programas esteganográficos usan estructuras de archivos diferentes, por ejemplo utilizan archivos GIFS con paletas de 256 colores aunque necesiten menos. Esto puede dar lugar a errores en GIFS comprimidos en los que se haya sustituido el bit menos significativo ya que los colores en la paleta no están lo suficientemente juntos unos de otros, puede ocurrir que un valor de 01001001 pertenezca a un azul oscuro y un valor de 01001000 pertenezca a un rosa oscuro, provocando que la imagen se distorsione claramente. No obstante la mayoría de programa esteganográficos implementan algoritmos de ordenamiento de los colores para solventar este error. Si el analista conoce el algoritmo podrá detectar si el archivo contiene información oculta. Una técnica que emplea en el estegoanálisis de imágenes es el empleo de imágenes interpoladas, la cual se describe a continuación.

[12] imagen obtenida de (Wayner, 2008)

2.1.3.2.1 Imágenes interpoladas

Uno de los secretos de la fotografía digital moderna es que una cámara de n megapíxeles no tiene realmente n millones de sensores que detectan la intensidad de la luz, roja, verde y azul sobre un punto. Muchas cámaras tienen sólo un sensor en cada píxel que detecta el rojo, el verde o el azul, los otros dos valores se aproximan haciendo la media de los valores vecinos. Ésta aproximación a menudo es llamada mosaico de filtro de color o interpolación cromática.

Un ejemplo de mosaico de filtrado de color es el conocido como mosaico de Bayer, en tributo a su creador Bryce Bayer, hay más píxeles en verde debido a que el ojo humano es más sensible a este color:

b g b g b
g r g r g
b g b g b
g r g r g
b g b g b
g r g r g

De esta manera el elemento (3,4) será un sensor de píxel de luz verde, el valor rojo de ese elemento se calculará como una media ponderada de los valores rojos vecinos: los valores rojos de los lados, (2,4) y (4,4), los valores rojos superiores, (2,2) y (4,2) y los valores rojos inferiores (2,6) y (4,6). Muchas cámaras emplean un rango mayor de valores para la ponderación pero esto puede incrementar los errores en los bordes o en otras zonas donde la intensidad cambia de valor bruscamente. Se utiliza una media ponderada para dar más valor a los elementos más cercanos a las cámaras y así intentar evitar lo anterior aunque a menudo incluyen funciones para reducir los errores en los bordes.

Alin Popescu y Hany Farid descubrieron que las imágenes retocadas tienen valores que no cumplen lo anterior, es decir un píxel rojo no se calcula como la media de sus vecinos. Esto será más evidente en los bordes. Ambos idearon un algoritmo que invierte las medias ponderadas estimando los valores del sensor original y determina qué píxeles fueron generados de alguna forma que no se corresponde con la media estándar hecha por las cámaras digitales midiendo cual es su desviación con respecto a esta media estándar. Con este algoritmo se puede detectar si una imagen ha sido alterada con Photoshop o con alguna técnica esteganográfica. A continuación se muestra la siguiente figura (obtenida de (Wayner, 2008)) la cual representa una imagen alterada con Photoshop y el resultado de pasarle el algoritmo de Popescu y Farid.

Ilustración 11: La imagen superior muestra una fotografía alterada con Photoshop. La imagen inferior muestra el resultado de aplicar sobre la misma el algoritmo de Popescu y Farid[13]

En la imagen de arriba se pegó con Photoshop un barco en el cielo. La imagen de abajo muestra los valores más sospechosos de haber sido cambiados con un color más oscuro. Los valores más oscuros corresponden al barco, aunque hay otros valores grises que sugieren que hay alguna incongruencia, posiblemente sea un efecto causado por el proceso de compresión.

2.1.3.3 Análisis estadístico

Muchos de los estudios estadísticos se centran en determinar cuando un fenómeno ocurre aleatoriamente y cuando no: Los científicos los emplean para determinar si sus teorías explican correctamente el fenómeno. En la esteganografía se pueden emplear para calcular cuando un archivo contiene información oculta ya que a menudo los mensajes ocultos son más aleatorios que la información que sustituyen. El test más simple que se puede realizar para detectar aleatoriedad es el de Chi cuadrado, el cual suma el cuadrado de las diferencias. De manera que {e0, e1,…} será el número de veces que un evento ocurra. En el caso de la esteganografía puede ser el número de veces que un bit menos significativo es uno o cero. Así $E(e_i)$ será el número de veces que se

[13] imagen obtenida de (Wayner, 2008)

espera que el evento ocurra si la muestra era totalmente aleatoria. La cantidad de aleatoriedad en la muestra es medida con la siguiente ecuación:

$$x^2 = \sum \frac{(e_i - E(e_i))^2}{E(e_i)}$$

Valores altos indican una condición baja de aleatoriedad y valores bajos indican una condición alta de aleatoriedad, es decir, que es probable que la muestra contenga información oculta.

El test de la Chi cuadrado puede aplicarse a cualquier parte del archivo. El bit menos significativo puede analizarse de acuerdo a dos eventos, e0 cuando vale 0 y e1 cuando vale 1. Un valor bajo indica que los bits se producen casi con la misma probabilidad mientras que un valor alto significa que un bit supera al otro. En este caso $E(e_0) = E(e_1) = 0.50$.

Una mejor solución es crear cuatro eventos que se corresponden con el patrón de los bits menos significativos de los valores vecinos. Las imágenes naturales, (entiéndase imágenes naturales como imágenes sin modificar) normalmente tienen unos valores vecinos con el mismo valor en el bit menos significativo. Archivos con información oculta suelen tener valores vecinos con valores distintos en el bit menos significativo.

Normalmente los test de la Chi cuadrado proporcionan un buen resultado en el análisis esteganográfico, aunque mecanismos más complejos para ocultar datos pueden saltarse estos test.

2.1.4 Estudio de las acciones

En esta sección se describen los conceptos de sujeto y acción. Se define el concepto de **acción** como el conjunto de procedimientos que realiza la aplicación ante un evento. Un **sujeto** en cambio es el elemento sobre el que se realiza la acción, ejemplos de sujetos son los archivos descargados o la misma prueba si la acción a realizar es de configuración.

Normalmente los archivos con información oculta mediante técnicas esteganográficas suelen estar protegidos mediante una clave por lo que una primera acción que se podría llevar a cabo sería un ataque de fuerza bruta sobre el archivo para intentar recuperar la clave, estos ataques consisten en probar sucesivamente todas las combinaciones posibles de claves utilizando letras, números, otros caracteres o una combinación de los anteriores de manera que se dé con la clave correcta y de pueda obtener el contenido oculto. Otro posible ataque es el ataque por diccionario consistente en ir recorriendo un archivo de texto denominado diccionario e ir probando con todas las palabras que contenga hasta dar con la clave correcta. Una vez obtenida la clave y si se descubre que el archivo contiene algún tipo de malware, información maliciosa, información sensible o está incumpliendo la ley de alguna forma la aplicación tomará una nueva acción a elegir entre una lista. Entre estas posibles acciones se encuentra la

posibilidad de obtener las direcciones IP de los pares que contienen el archivo contenedor de la información oculta con el fin de avisar de alguna forma a los afectados o informar a la policía del suceso. Para entender mejor los conceptos de acción y sujeto se van a presentar una serie de ejemplos:

- **Escenario:** Se descarga un archivo de la red P2P.
  Acción: Se ejecuta una prueba esteganográfica sobre el archivo descargado.
  Sujeto: Elemento descargado.
- **Escenario:** La prueba realizada detecta que al archivo contiene información oculta.
  Acción: Realizar un ataque de fuerza bruta con el fin de extraer dicha información oculta.
  Sujeto: Elemento descargado.
- **Escenario:** Se consigue extraer la información ocultada en un archivo.
  Acción: Obtener las direcciones IP de las víctimas que descargaron ese archivo (más adelante se detallará el proceso de recuperación de direcciones IP).
  Sujeto: Elemento descargado.

2.2 Alternativas de diseño

En esta sección se va a exponer una serie de alternativas para el futuro diseño de la aplicación listando sus ventajas y desventajas y finalmente se elegirá una de ellas, la cual será implementada. Además utilizando la información sobre redes P2P desarrollada anteriormente se elegirá una red para ser implementada inicialmente.

2.2.1 Aspectos de interfaz

En este apartado se analizarán las distintas alternativas a tomar para el desarrollo de la interfaz de usuario de la aplicación. Éste es un aspecto fundamental de la misma ya que será el elemento con el que interactúe el usuario.

2.2.1.1 Aplicación con interfaz de usuario gráfica

Esta alternativa de diseño consiste en la implementación de una interfaz de usuario o GUI (Graphical User Interface), es decir, un programa que permite la comunicación entre el usuario y la aplicación mediante elementos o iconos que representen las acciones que se pueden llevar a cabo. La interfaz de usuario se desarrollará mediante alguna librería especializada y que trabaje con la interfaz gráfica nativa del sistema operativo donde se desarrolle la aplicación.

Las interfaces de usuario facilitan la interacción con el usuario ya que éste puede manipular directamente los elementos de la interfaz y se da cuenta rápidamente de las acciones que puede realizar con la aplicación. Por otro lado la programación de una interfaz de usuario puede llegar a complicarse mucho y hace falta experiencia para lograr una interfaz intuitiva y fácil de utilizar ya que además de factores tecnológicos entran en juego factores humanos. La elaboración de una buena interfaz es primordial ya que será la capa de presentación de la aplicación y con la que interactuará el usuario, además una mala interfaz puede provocar problemas de usabilidad y que éste rechace usar la aplicación. Por último añadir que el usuario medio está acostumbrado al uso de interfaces gráficas ya que casi todos los programas de uso común tienen una interfaz de este tipo.

Finalmente y a modo de resumen se muestra una lista de las ventajas y desventajas del uso de interfaces gráficas:

Ventajas	**Desventajas**
Facilidad de interacción con el usuario	Pueden ser difíciles de programar
Es intuitiva	Inexperiencia del desarrollador
El usuario medio está acostumbrado a ellas	

2.2.1.2 Aplicación con interfaz de línea de comandos

Esta alternativa de diseño consiste en la implementación de la aplicación utilizando una interfaz de línea de comandos o CLI (Command Line Interface), es decir, un programa que sea capaz de interpretar los comandos o instrucciones introducidos por consola por el usuario. La interfaz se desarrollará mediante alguna librería especializada y que facilite el parseo de las instrucciones dadas por los usuarios.

Ventajas	**Desventajas**
Rapidez	Poca tolerancia a fallos, pueden provocar frustración
Facilidad	El usuario medio puede sentir rechazo hacia ellas
Permiten extenderse mediante lenguajes de scripting	

Las interfaces de líneas de comandos son bastante rápidas y fáciles de implementar ya que simplemente tienen que interpretar comandos, los cuales se reducen a cadenas de texto. Además pueden extenderse mediante lenguajes de scripting y permitir al usuario la programación de diferentes operaciones, por ejemplo se pueden implementar bucles que ejecuten de forma repetida una aplicación, iniciar alguna aplicación de acuerdo a algún evento, etc. Por otro lado las interfaces de líneas de comandos provocan que el usuario tenga que recordar una lista de comandos, lo que aumenta su carga cognitiva, y además son más restrictivas con los fallos de los usuarios lo que puede provocar frustración en éstos. Por último y tal y como se ha dicho al hablar de

las interfaces gráficas el usuario medio está más acostumbrado al uso de éstas y por lo tanto puede sentir rechazo al tener que enfrentarse ante una interfaz de línea de comandos.

2.2.2 Decisión de la alternativa de diseño

Después de valorar las ventajas y las desventajas de las diferentes alternativas la decisión tomada en cuanto a la interfaz es la implementación de la alternativa "aplicación con interfaz de línea de comandos". Los motivos principales para la toma de esta decisión son la inexperiencia del desarrollador en la elaboración de interfaces gráficas y la posibilidad de un elevado gasto de tiempo en el desarrollo de una interfaz gráfica adecuada, además la aplicación no está centrada al usuario medio y por lo tanto es más probable que sus usuarios no rechacen el uso de una interfaz mediante línea de comandos. En cuanto a la red P2P a implementar inicialmente las alternativas barajadas eran las mimas que se estudiaron en el capítulo de análisis, es decir, las redes BitTorrent y eDonkey2000, utilizándose algún cliente de éstas como apoyo para el desarrollo de la aplicación. Finalmente se optó por implementar la red BitTorrent utilizando el cliente Vuze como apoyo por las características de éste ya que está escrito en Java y el desarrollador tiene bastante experiencia con este lenguaje de programación, ofrece bastantes opciones, es de licencia libre y además es uno de los clientes BitTorrent más conocidos y utilizados.

2.3 Casos de uso

Antes de proceder a la extracción de los requisitos se realizará un análisis de los posibles casos de uso de la aplicación, éstos se derivarán de los objetivos de la aplicación. Todos los casos de uso se describirán de una forma concisa. Para mostrar los casos de uso se empleará una tabla a modo de plantilla que contiene la siguiente información:

IDENTIFICADOR	
Nombre	
Actores	
Objetivo	
Descripción	
Pre-condiciones	
Post-condiciones	
Escenario	
Condiciones de fallo	

Tabla 1: Plantilla para la elaboración de casos de uso

Siendo cada campo:

- **Identificador:** Identifica al caso de uso. Se define como CU-XX, donde XX es un número que representa unívocamente el requisito.
- **Nombre**: Nombre que identificará el caso de uso.
- **Actores**: Los posibles roles de usuario que pueden intervenir en el caso de uso.
- **Objetivo**: La descripción de la finalidad que pretende el actor con la realización de este caso de uso.
- **Descripción**: Breve explicación del proceso llevado a cabo en el caso de uso.
- **Pre-condiciones**: Son aquellos hechos que deben ser previamente ciertos para poder llevar a cabo el caso de uso.
- **Post-condiciones**: Son aquellos hechos que la ejecución del caso de uso hace verdaderos o ciertos.
- **Escenario**: Descripción esquemática de las fases que componen el caso de uso.
- **Condiciones de fallo**: describe posibles errores y las respuestas del sistema ante los mismos.

A continuación, se listan los diferentes casos de uso con diagramas representativos de los mismos:

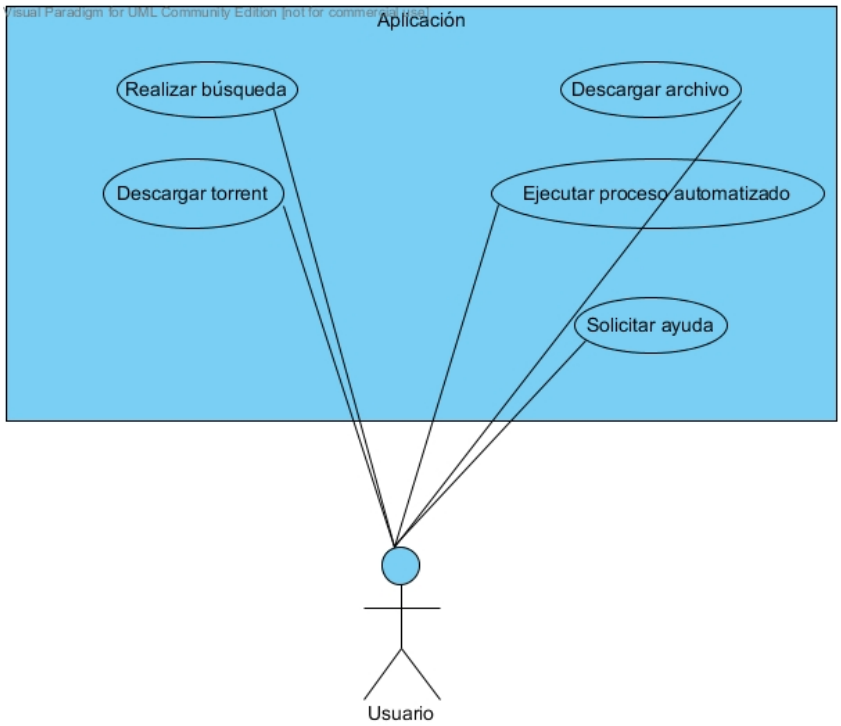


Ilustración 12: Diagramas de casos de uso

CU – 01	
Nombre	Realizar búsqueda
Actores	Usuario
Objetivo	Obtener enlaces para descargar un torrent

Descripción	El usuario introduce una serie de parámetros como la red P2P sobre la que buscar y la cadena de búsqueda y recupera una lista de elementos con, entre otra información, los enlaces a los torrents para descargar el archivo
Pre-condiciones	El usuario debe haber seleccionado la operación[14] de buscar y haber introducido los parámetros necesarios
Post-condiciones	La aplicación devuelve los resultados de la búsqueda
Escenario	1. El usuario selecciona la operación buscar e introduce los parámetros necesarios. 2. La aplicación ejecuta la búsqueda y devuelve los resultados. 3. La aplicación escribe los resultados en un informe
Condiciones de fallo	El usuario no introduce todos los parámetros necesarios o los parámetros introducidos no son correctos. Hay algún problema de red.

Caso de uso 1: Realizar búsqueda

CU – 02	
Nombre	Descargar torrent
Actores	Usuario
Objetivo	Obtener un archivo torrent, necesario para la descarga del archivo
Descripción	El usuario introduce la URL (véase Glosario - URL) que localiza el torrent deseado y la aplicación procede a la descarga de dicho archivo
Pre-condiciones	El usuario debe haber seleccionado la operación de descargar y haber introducido los parámetros necesarios
Post-condiciones	La aplicación descarga el archivo torrent deseado y lo coloca en la carpeta seleccionada
Escenario	1. El usuario selecciona la operación descargar e introduce los parámetros necesarios. 2. La aplicación ejecuta la operación y descarga el archivo torrent deseado y lo coloca en la carpeta seleccionada.
Condiciones de fallo	El usuario no introduce todos los parámetros necesarios o los parámetros introducidos no son correctos. Hay algún problema de red.

Caso de uso 2: Descargar torrent

CU – 03	
Nombre	Descargar archivo
Actores	Usuario
Objetivo	Obtener un archivo que potencialmente contiene información oculta
Descripción	El usuario introduce la ruta al archivo torrent del archivo que se

[14] A partir de ahora y a lo largo de todo el documento se utilizará el término operación como sinónimo del término acción. El motivo de la utilización de este término es la diferenciación de las distintas operaciones que puede ejecutar la aplicación como puede ser la búsqueda o descarga de archivos de las acciones que se ejecutan una vez se ha detectado algún tipo de contenido oculto.

	quiere descargar y la aplicación procede a la descarga
Pre-condiciones	El usuario debe haber seleccionado la operación de descargar y haber introducido los parámetros necesarios
Post-condiciones	La aplicación descarga el archivo deseado y lo coloca en la carpeta seleccionada
Escenario	1. El usuario selecciona la operación descargar e introduce los parámetros necesarios. 2. La aplicación ejecuta la operación y descarga el archivo deseado y lo coloca en la carpeta seleccionada 3. La aplicación comienza la ejecución de los detectores. 4. A partir de los resultados de los detectores la aplicación ejecuta distintas acciones. 5. Con los resultados de los detectores y de las acciones la aplicación elabora un informe.
Condiciones de fallo	El usuario no introduce todos los parámetros necesarios o los parámetros introducidos no son correctos. Hay algún problema de red.

Caso de uso 3: Descargar archivo

CU – 04	
Nombre	Ejecutar proceso automatizado
Actores	Usuario
Objetivo	Obtener un conjunto de archivos que potencialmente pueden contener información oculta
Descripción	El usuario introduce una cadena de búsqueda que se utiliza para descargar un número de archivos determinado por el usuario. Una vez terminada la descarga o durante esta el programa iniciará una serie de detectores con el fin de desvelar si los archivos analizados tienen información oculta. Según el resultado de los detectores se ejecutará una serie de acciones y con los resultados de éstas y de los detectores se elaborará un informe.
Pre-condiciones	El usuario debe haber seleccionado la operación de proceso automatizado y haber introducido los parámetros necesarios
Post-condiciones	La aplicación busca y descarga los archivos deseados y los coloca en la carpeta seleccionada
Escenario	1. El usuario selecciona la operación de proceso automatizado e introduce los parámetros necesarios. 2. La aplicación ejecuta la búsqueda utilizando la query introducida por el usuario. 3. La aplicación descarga el número de archivos torrent que le indicó el usuario. 4. Una vez descargados los archivos torrent la aplicación los utiliza para comenzar la descarga de los archivos. 5. La aplicación comienza la ejecución de los detectores. 6. A partir de los resultados de los detectores la aplicación ejecuta distintas acciones. 7. Con los resultados de los detectores y de las acciones la aplicación elabora un informe.
Condiciones de	El usuario no introduce todos los parámetros necesarios o los

fallo	parámetros introducidos no son correctos. Se bloquea la descarga de un archivo torrent. Hay algún problema de red.

Caso de uso 4: Ejecutar proceso automatizado

CU – 05	
Nombre	Solicitar ayuda
Actores	Usuario
Objetivo	Obtener información acerca del modo de uso del programa
Descripción	El usuario selecciona la opción de obtener ayuda y la aplicación muestra por pantalla instrucciones para el correcto modo de uso del programa
Pre-condiciones	El usuario debe haber introducido la opción de obtener ayuda
Post-condiciones	La aplicación muestra la ayuda por pantalla
Escenario	1. El usuario selecciona la operación obtener ayuda. 2. La aplicación muestra la ayuda por pantalla.
Condiciones de fallo	

Caso de uso 5: Solicitar ayuda

2.4 Requisitos

En este apartado se presentan los requisitos recogidos para el producto a desarrollar, tanto funcionales como no funcionales.

Antes de mostrar los se procederá a explicar los atributos que cada uno de estos incluirá en su definición.

Cada requisito debe contener los siguientes atributos:

- **Identificador:** cada requisito tendrá un identificador único de manera que lo representará de manera unívoca. Este identificador será:
  - RF-XX. Representa un requisito funcional. Será necesario sustituir XX por el número que se utilice para identificar el requisito.
  - RNF-XX. Representa un requisito no funcional. Será necesario sustituir XX por el número que se utilice para identificar el requisito.
- **Fuente:** identifica el caso de uso del que proviene el requisito.
- **Prioridad:** cada requisito incluirá una medida de la prioridad que posee, para facilita la planificación del desarrollo de la aplicación. Este atributo podrá tomar los siguientes valores:
  - Alta.
  - Media.
  - Baja.
- **Riesgo:** es una estimación del riesgo que supone realizar el requisito, está relacionado con la dificultad de la implementación del mismo. Este atributo podrá tomar los siguientes valores:

  - Alto.
  - Medio.
  - Bajo.

  Se deberá prestar especial atención a los requisitos de riesgo alto.
- **Estabilidad:** indica el grado de modificación que puede sufrir un requisito a lo largo del ciclo de desarrollo. Algunos requisitos pueden permanecer estables durante toda la vida esperada del software, otros sin embargo pueden verse modificados. Tales requisitos inestables se deben señalar, para prestarles la atención que requieren a lo largo del desarrollo del proyecto.
- **Verificabilidad:** cada requisito será verificable. Esto significa que debe ser posible comprobar fehacientemente que el requisito se ha incorporado en el diseño, es decir, que se puede probar que el software aplica el requisito. Este atributo podrá tomar los siguientes valores:
  - Alta.
  - Media.
  - Baja.
- **Descripción corta:** sirve para entender rápidamente el significado del requisito.
- **Descripción:** definición extensa del significado del requisito, expresado de manera coherente y clara para evitar ambigüedades en su interpretación. Esta descripción puede ser textual y/o acompañarse de diagramas o prototipos. Sea cual fuere la forma de representar la descripción, esta deberá ser:
  - Clara
  - Verificable

El formato de documentación de un requisito es el siguiente:

IDENTIFICADOR			
Fuente			
Prioridad	Alta	Media	Baja
Riesgo	Alto	Medio	Bajo
Estabilidad			
Verificabilidad	Alta	Media	Baja
Descripción corta			
Descripción			

Tabla 2: Plantilla para la elaboración de requisitos

2.4.1 Requisitos funcionales

Los requisitos funcionales especifican “qué” tiene que hacer el software, su comportamiento y su propósito. A continuación se exponen los requisitos funcionales:

RF – 01			
Fuente	CU – 01		
Prioridad	✓ Alta	Media	Baja
Riesgo	✓ Alto	Medio	Bajo
Estabilidad	Durante toda la vida de la aplicación		
Verificabilidad	✓ Alta	Media	Baja
Descripción corta	Permitir la búsqueda de torrents		
Descripción	La aplicación le permitirá al usuario buscar archivos torrents, para ello éste debe introducir la red que se quiere utilizar, la cadena de búsqueda y el tipo de informe (txt o pdf) en el que se escribirán los resultados obtenidos.		

Requisito 1: Permitir la búsqueda de torrents

RF – 02			
Fuente	CU – 01		
Prioridad	Alta	✓ Media	Baja
Riesgo	Alto	Medio	✓ Bajo
Estabilidad	Durante toda la vida de la aplicación		
Verificabilidad	✓ Alta	Media	Baja
Descripción corta	Permitir ordenar los resultados		
Descripción	La aplicación le permitirá al usuario ordenar los resultados de la búsqueda de acuerdo a diferentes criterios (tamaño, nombre, número de seeds…). Además se podrá elegir el modo de ordenación entre ascendente y descendente		

Requisito 2: Permitir ordenar los resultados

RF – 03			
Fuente	CU – 01		
Prioridad	Alta	✓ Media	Baja
Riesgo	Alto	Medio	✓ Bajo
Estabilidad	Durante toda la vida de la aplicación		
Verificabilidad	✓ Alta	Media	Baja
Descripción corta	Permitir filtrar los resultados		
Descripción	La aplicación le permitirá al usuario filtrar los resultados de la búsqueda de acuerdo a diferentes criterios (tamaño, número de seeds, hash…)		

Requisito 3: Permitir filtrar los resultados

RF – 04			
Fuente	CU – 02		
Prioridad	✓ Alta	Media	Baja
Riesgo	✓ Alto	Medio	Bajo
Estabilidad	Durante toda la vida de la aplicación		
Verificabilidad	✓ Alta	Media	Baja
Descripción corta	Permitir la descarga de torrents		

Descripción	La aplicación le permitirá al usuario descargar archivos torrents, para ello éste debe introducir la red que se quiere utilizar y la URL que localiza al torrent. Opcionalmente se le podrán indicar una URL de referencia para los torrents que lo necesiten

Requisito 4: Permitir la descarga de torrents

RF – 05			
Fuente	CU – 02		
Prioridad	✓ Alta	Media	Baja
Riesgo	Alto	Medio	✓ Bajo
Estabilidad	Durante toda la vida de la aplicación		
Verificabilidad	✓ Alta	Media	Baja
Descripción corta	Aceptar enlaces magnet (véase Glosario - Enlace magnet)		
Descripción	La aplicación permitirá la descarga de archivos torrent utilizando enlaces magnet		

Requisito 5: Aceptar enlaces magnet

RF – 06			
Fuente	CU – 03		
Prioridad	✓ Alta	Media	Baja
Riesgo	✓ Alto	Medio	Bajo
Estabilidad	Durante toda la vida de la aplicación		
Verificabilidad	✓ Alta	Media	Baja
Descripción corta	Permitir la descarga de archivos		
Descripción	La aplicación le permitirá al usuario descargar archivos con el fin de comprobar si tienen información oculta, para ello éste debe introducir la red que se quiere utilizar y la ruta al archivo torrent descargado previamente y el cual contiene la información necesaria para la descarga		

Requisito 6: Permitir la descarga de archivos

RF – 07			
Fuente	CU – 03		
Prioridad	✓ Alta	Media	Baja
Riesgo	Alto	✓ Medio	Bajo
Estabilidad	Durante toda la vida de la aplicación		
Verificabilidad	✓ Alta	Media	Baja
Descripción corta	Permitir reanudar las descargas		
Descripción	La aplicación permitirá la reanudación de las descargas que aun no había completado cuando finalizó la ejecución		

Requisito 7: Permitir reanudar las descargas

RF – 08			
Fuente	CU – 02, CU – 03		
Prioridad	✓ Alta	Media	Baja
Riesgo	Alto	✓ Medio	Bajo
Estabilidad	Durante toda la vida de la aplicación		
Verificabilidad	✓ Alta	Media	Baja
Descripción corta	Permitir cancelar las descargas		
Descripción	La aplicación permitirá la cancelación de la descarga de un archivo que está en ejecución. Al cancelar la descarga de un archivo se borraran los datos que ya se habían descargado		

Requisito 8: Permitir cancelar las descargas

RF – 09			
Fuente	CU – 04		
Prioridad	✓ Alta	Media	Baja
Riesgo	✓ Alto	Medio	Bajo
Estabilidad	Durante toda la vida de la aplicación		
Verificabilidad	✓ Alta	Media	Baja
Descripción corta	Ejecutar proceso automatizado		
Descripción	La aplicación será capaz de, a partir de una cadena de búsqueda introducida por el usuario, descargar un número de torrents determinado por el usuario y proceder con la descarga de los archivos utilizando dichos torrents.		

Requisito 9: Ejecutar proceso automatizado

RF – 10			
Fuente	CU – 03, CU – 04		
Prioridad	✓ Alta	Media	Baja
Riesgo	✓ Alto	Medio	Bajo
Estabilidad	Durante toda la vida de la aplicación		
Verificabilidad	✓ Alta	Media	Baja
Descripción corta	Ejecutar detectores sobre los archivos descargados		
Descripción	La aplicación ejecutará detectores con el fin de desvelar si los archivos descargados contienen información oculta mediante técnicas esteganográficas. Los detectores se valdrán de técnicas de estegoanálisis para este fin		

Requisito 10: Ejecutar detectores sobre los archivos descargados

RF – 11			
Fuente	CU – 03, CU – 04		
Prioridad	Alta	✓ Media	Baja
Riesgo	✓ Alto	Medio	Bajo
Estabilidad	Puede verse modificado en la fase de implementación		
Verificabilidad	✓ Alta	Media	Baja
Descripción corta	Permitir ejecutar detectores durante el proceso de descarga		

Descripción	La aplicación permitirá la ejecución de los detectores sobre un archivo durante la descarga del mismo, para ellos deberán conocer las partes del archivo que ya se han descargado

Requisito 11: Permitir ejecutar detectores durante el proceso de descarga

RF – 12			
Fuente	CU – 03, CU – 04		
Prioridad	✓ Alta	Media	Baja
Riesgo	✓ Alto	Medio	Bajo
Estabilidad	Durante toda la vida de la aplicación		
Verificabilidad	✓ Alta	Media	Baja
Descripción corta	Ejecutar acciones de acuerdo al resultado de los detectores		
Descripción	La aplicación ejecutará una acción a elegir entre una lista. La acción a ejecutar dependerá del resultado de los detectores ejecutados		

Requisito 12: Ejecutar acciones de acuerdo al resultado de los detectores

RF – 13			
Fuente	CU – 03, CU – 04		
Prioridad	✓ Alta	Media	Baja
Riesgo	Alto	Medio	✓ Bajo
Estabilidad	Durante toda la vida de la aplicación		
Verificabilidad	✓ Alta	Media	Baja
Descripción corta	Elaborar informes		
Descripción	La aplicación elaborará informes con información sobre el resultado proporcionado por los detectores y la acción llevada a cabo. El informe detallará entre otras cosas si se detectó información oculta en el archivo, mostrará dicha información, indicará cual fue la técnica esteganográfica utilizada y mostrará el resultado de la acción ejecutada		

Requisito 13: Elaborar informes

RF – 14			
Fuente	CU – 05		
Prioridad	✓ Alta	Media	Baja
Riesgo	Alto	Medio	✓ Bajo
Estabilidad	Durante toda la vida de la aplicación		
Verificabilidad	✓ Alta	Media	Baja
Descripción corta	Permitir la solicitud de ayuda		
Descripción	La aplicación le permitirá al usuario solicitar ayuda en caso de que no conozca el funcionamiento de la misma. Ésta le mostrará al usuario una serie de instrucciones para que conozca su modo de uso		

Requisito 14: Permitir la solicitud de ayuda

2.4.2 Requisitos no funcionales

Los requisitos no funcionales especifican "cómo" va a funcionar la aplicación en términos de rendimiento, interfaz, operación, seguridad, calidad, mantenimiento, recursos, etc. A continuación se especifican los requisitos no funcionales:

2.4.2.1 Requisitos de interfaz

RNF – 01			
Fuente			
Prioridad	✓ Alta	Media	Baja
Riesgo	Alto	Medio	✓ Bajo
Estabilidad	Durante toda la vida de la aplicación		
Verificabilidad	✓ Alta	Media	Baja
Descripción corta	Lenguaje de programación de la aplicación		
Descripción	La aplicación se programará utilizando el lenguaje de programación Java por lo que para su ejecución se necesitará tenerlo instalado en el equipo		

Requisito 15: Lenguaje de programación de la aplicación

RNF – 02			
Fuente			
Prioridad	Alta	✓ Media	Baja
Riesgo	Alto	✓ Medio	Bajo
Estabilidad	Durante toda la vida de la aplicación		
Verificabilidad	✓ Alta	Media	Baja
Descripción corta	Sistemas operativos soportados		
Descripción	La aplicación funcionará en los sistemas operativos Linux, Windows y Mac		

Requisito 16: Sistemas operativos soportados

RNF – 03			
Fuente			
Prioridad	✓ Alta	Media	Baja
Riesgo	Alto	Medio	✓ Bajo
Estabilidad	Durante toda la vida de la aplicación		
Verificabilidad	✓ Alta	Media	Baja
Descripción corta	Interfaz de la aplicación		
Descripción	La aplicación será ejecutada por consola mediante comandos (ver más adelante el apartado alternativas de diseño)		

Requisito 17: Interfaz de la aplicación

RNF – 04			
Fuente			
Prioridad	✓ Alta	Media	Baja
Riesgo	Alto	Medio	✓ Bajo
Estabilidad	Durante toda la vida de la aplicación		
Verificabilidad	✓ Alta	Media	Baja
Descripción corta	Parámetros mediante archivos XML		
Descripción	La aplicación permitirá la introducción de los parámetros mediante archivos XML		

Requisito 18: Parámetros mediante archivos XML

2.4.2.2 Requisitos de comprobación

RNF – 05			
Fuente			
Prioridad	✓ Alta	Media	Baja
Riesgo	Alto	✓ Medio	Bajo
Estabilidad	Durante toda la vida de la aplicación		
Verificabilidad	✓ Alta	Media	Baja
Descripción corta	Verificación de los datos de entrada		
Descripción	La aplicación utilizará una librería para parsear los comandos introducidos por el usuario y comprobar que se han introducido los comandos necesarios. Adicionalmente a las comprobaciones que realice la librería se realizarán comprobaciones adicionales para conocer si los datos introducidos son válidos		

Requisito 19: Verificación de los datos de entrada

2.4.2.3 Requisitos de aceptación de pruebas

RNF – 06			
Fuente			
Prioridad	✓ Alta	Media	Baja
Riesgo	Alto	✓ Medio	Bajo
Estabilidad	Durante toda la vida de la aplicación		
Verificabilidad	✓ Alta	Media	Baja
Descripción corta	Ejecución de pruebas de aceptación		
Descripción	La aplicación deberá superar con éxito pruebas de aceptación para que se pueda dar como aceptada		

Requisito 20: Ejecución de pruebas de aceptación

2.4.2.4 Requisitos de mantenimiento

RNF – 07			
Fuente			
Prioridad	✓ Alta	Media	Baja
Riesgo	Alto	✓ Medio	Bajo
Estabilidad	Durante toda la vida de la aplicación		
Verificabilidad	Alta	✓ Media	Baja
Descripción corta	Aplicación modular		
Descripción	La aplicación se programará de manera modular con el objetivo de facilitar la inclusión de nuevas funcionalidades		

Requisito 21: Aplicación modular

2.4.3 Matriz de trazabilidad requisitos-casos de uso

La siguiente matriz relaciona los casos de uso con los requisitos funcionales derivados de ellos.

	CU-01	CU-02	CU-03	CU-04	CU-05
RF-01	X				
RF-02	X				
RF-03	X				
RF-04		X			
RF-05		X			
RF-06			X		
RF-07			X		
RF-08		X	X		
RF-09				X	
RF-10			X	X	
RF-11			X	X	
RF-12			X	X	
RF-13			X	X	
RF-14					X

Tabla 3: Matriz de trazabilidad requisitos-casos de uso

2.5 Marco regulador

Este apartado no aplica en este proyecto por ser una aplicación centrada en la investigación.

2.6 Restricciones

Este apartado no aplica en este proyecto.

2.7 Plan de pruebas

En este apartado se especificará un listado de pruebas, las cuales cubrirán todos los requisitos expuestos en el apartado anterior. Para considerar que el proyecto ha finalizado con éxito deben superarse todas las pruebas.

2.7.1 Formato del catálogo de pruebas

Los campos que forman cada una de las pruebas son:

- **Identificador:** clave que sirve para identificar una prueba de manera unívoca. El formato para la identificación de las pruebas será el siguiente: PR-XX siendo XX un número utilizado para identificar la prueba.
- **Descripción:** definición detallada de la prueba.
- **Entrada:** parámetros que se le proporcionan a la prueba para su ejecución.
- **Salida esperada:** resultado que se espera obtener al finalizar la prueba.

2.7.2 Catálogo de pruebas de aceptación

A continuación se muestra el catálogo de pruebas de aceptación.

Identificador	PR-01
Descripción	Se ejecutará una operación de búsqueda sobre la red BitTorrent. Además se añadirán argumentos opcionales para realizar la ordenación y el filtrado de los resultados
Entrada	Se deben especificar los siguientes argumentos con los siguientes valores (el significado de los argumentos se detallará en el capítulo de implementación): • **Mode (-m):** command • **Operation (-p):** search • **Network (-n):** torrent • **SearchQuery (-s):** prueba • **InformType (-i):** txt • **Out (--out):** informe • **Size (--size):** 1073741824 (1 GB especificado en bytes)

	• **Nseeds (--nseeds):** 20 • **Sort (--sort):** nseeds • **SortMode (--sortmode):** descendent
Salida esperada	Se deberá obtener un informe en texto plano con los resultados devueltos por la aplicación. Dichos resultados deberán tener un tamaño no superior a 1 GB y un número de seeds mayor a 20. Además deberán estar ordenados por número de seeds de manera descendente

Prueba 1: Prueba de operación de búsqueda

Identificador	PR-02
Descripción	Se introducirán los argumentos necesarios para realizar una operación de descarga de un archivo torrent. La descarga del archivo torrent se realizará a partir de un enlace magnet
Entrada	Se deben especificar los siguientes argumentos con los siguientes valores: • **Mode (-m):** command • **Operation (-p):** download • **Network (-n):** torrent • **Url (-u):** magnet:?xt=urn:btih:HK74OKXSCCTKNLHAMP54NN4ZE4TBISZ7&tr=http://tracker.mininova.org/announce
Salida esperada	Se deberá obtener el archivo torrent descargado en la carpeta de torrents configurada en la aplicación

Prueba 2: Prueba de operación de descarga de archivo torrent

Identificador	PR-03
Descripción	Se introducirán los argumentos necesarios para realizar una operación de descarga de un archivo.
Entrada	Se deben especificar los siguientes argumentos con los siguientes valores: • **Mode (-m):** command • **Operation (-o):** download • **Network (-n):** torrent • **Torrent (-t):** prueba.torrent • **Path (-p):** Downloads • **Dconf (--dconf):** [type:vecna] • **Aconf (--aconf):** [type:bruteforce,charsets:minLetters,initLength:4]
Salida esperada	Se deberá obtener el archivo descargado en la carpeta especificada mediante el argumento Path. Además si el archivo provoca la activación del detector se deberán obtener informes sobre el proceso de detección y sobre la acción llevada a cabo en la carpeta de informes configurada en la aplicación

Prueba 3: Prueba de operación de descarga de archivo

Identificador	PR-04
Descripción	Se ejecutará una operación de descarga de archivo equivalente a la de la prueba anterior y durante el proceso de descarga del mismo se introducirá la orden "exit" para salir de la aplicación dejando la descarga como activa. Finalmente se ejecutará otra operación de descarga de archivo con los mismos argumentos para comprobar que la descarga continúa a partir del mismo punto donde se había quedado anteriormente.
Entrada	Se deben especificar los siguientes argumentos con los siguientes valores: • **Mode (-m):** command • **Operation (-o):** download • **Network (-n):** torrent • **Torrent (-t):** prueba.torrent • **Path (-p):** Downloads • **Dconf (--dconf):** [type:vecna] • **Aconf (--aconf):** [type:bruteforce,charsets:minLetters,initLength:4]
Salida esperada	La descarga se deberá reanudar a partir del mismo punto donde se había quedado antes de introducir la orden "exit"

Prueba 4: Prueba de reanudación de una descarga de archivo

Identificador	PR-05
Descripción	Se ejecutará una operación de descarga de archivo equivalente a la de la prueba anterior y se introducirá la orden "cancel" para cancelar la descarga.
Entrada	Se deben especificar los siguientes argumentos con los siguientes valores: • **Mode (-m):** command • **Operation (-o):** download • **Network (-n):** torrent • **Torrent (-t):** prueba.torrent • **Path (-p):** Downloads • **Dconf (--dconf):** [type:vecna] • **Aconf (--aconf):** [type:bruteforce,charsets:minLetters,initLength:4]
Salida esperada	La carpeta de descargas especificada por el argumento Path no deberá contener ningún dato sobre la descarga

Prueba 5: Prueba de cancelación de una descarga de archivo

Identificador	PR-06
Descripción	Se introducirán los argumentos necesarios para la ejecución de una operación automatizada
Entrada	Se deben especificar los siguientes argumentos con los siguientes valores.

	• **Mode (-m):** command • **Operation (-o):** auto • **Network (-n):** torrent • **SearchQuery (-s):** prueba • **Number (--number):** 2 • **Path (-p):** Downloads • **Dconf (--dconf):** [type:vecna] • **Aconf (--aconf):** [type:bruteforce,charsets:minLetters,initLength:4]
Salida esperada	Se deberán obtener dos archivos descargados en la carpeta especificada mediante el argumento Path. Además alguno de los archivos archivo provoca la activación de un detector se deberán obtener informes sobre el proceso de detección y sobre la acción llevada a cabo en la carpeta de informes configurada en la aplicación

Prueba 6: Prueba de operación automatizada

Identificador	PR-07
Descripción	Se solicitará ayuda a la aplicación mediante los argumentos necesarios
Entrada	Se deben especificar los siguientes argumentos con los siguientes valores. • **Help (--help)**
Salida esperada	La aplicación mostrará la ayuda

Prueba 7: Prueba de solicitud de ayuda

2.7.3 Matriz de trazabilidad pruebas-requisitos

La siguiente matriz relaciona las pruebas con los requisitos funcionales asociados a ellas.

	PR-01	PR-02	PR-03	PR-04	PR-05	PR-06	PR-07
RF-01	X						
RF-02	X						
RF-03	X						
RF-04		X					
RF-05		X					
RF-06			X				
RF-07				X			
RF-08					X		
RF-09						X	
RF-10			X			X	
RF-11			X			X	
RF-12			X			X	

RF-13			X			X	
RF-14							X

Tabla 4: Matriz de trazabilidad pruebas-requisitos

Capitulo 3: Diseño

En este capítulo se detalla cómo se realizó el proceso de diseño de la aplicación. Se realiza un análisis de la arquitectura de la aplicación, se muestra el patrón de diseño que se siguió, se estudian a bajo nivel los diferentes componentes que forman la aplicación y finalmente se muestran diagramas de secuencia de algunas operaciones llevadas a cabo por la aplicación.

3.1 Arquitectura de la aplicación

La aplicación se tratará de un cliente P2P especial el cual será capaz de conectarse a la red BitTorrent para descargar y subir archivos, adicionalmente será capaz de extenderse de manera fácil para conectarse con otras redes P2P. Se dice que será un cliente P2P especial porque, además de las funciones básicas de un cliente P2P convencional de compartición de archivos, contará con funciones para la detección de contenidos ocultos en archivos mediante técnicas esteganográficas y la capacidad para llevar a cabo acciones según el resultado de los detectores.

Ya se ha analizado en profundidad la arquitectura P2P por lo que sólo se dirá brevemente y a modo de recordatorio que esta arquitectura está formada por ordenadores conectados entre sí y que pueden ejercer tanto de cliente como de servidor pudiendo eliminar la necesidad de disponer de un servidor central.

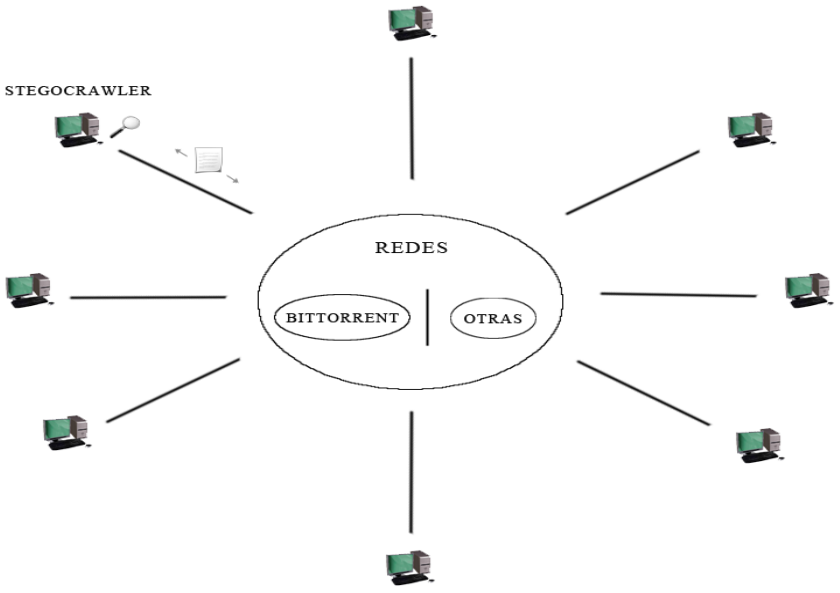


Ilustración 13: Arquitectura de la aplicación

En la ilustración anterior se puede observar un esquema de la arquitectura de la aplicación, en ella se muestra cómo se dispone de un conjunto de redes P2P a las que la aplicación podrá conectarse como un cliente más y compartir archivos, como ya se ha dicho antes, una vez descargado un archivo la aplicación ejecutará sus funciones especiales.

3.2 Patrón

En esta sección se hablará sobre el patrón de diseño que se seguirá para el desarrollo de la aplicación. El patrón a seguir elegido será el de Modelo Vista Controlador (MVC) aunque por las peculiaridades de la aplicación el patrón no se seguirá de manera estricta, tal y como se dijo anteriormente que la aplicación se implementará utilizando una interfaz de línea de comandos y el patrón MVC suele utilizarse para aplicaciones con algún tipo de interfaz gráfica, comúnmente aplicaciones web. Este patrón abstrae el desarrollo del software y lo divide en tres componentes: el modelo, la vista y el controlador, los cuales se describen a continuación:

- **Modelo:** Es la representación de la información de la aplicación. Interactúa con la vista y el controlador utilizando la primera para informar al usuario de lo que está ocurriendo y es accedido por el segundo para operar con los datos, es decir para consultarlos, añadirlos, modificarlos, eliminarlos, etc.
- **Vista:** Es la capa de presentación de la aplicación, es lo que ve y con lo que interactúa el usuario. Es utilizada por el usuario para recibir órdenes y por el modelo para informarle a éste del estado de la aplicación.
- **Controlador:** Es el encargado de tomar decisiones en la aplicación, recibe las órdenes de la vista y es capaz de gestionarlas y actuar en consecuencia. Puede interaccionar con el modelo o la vista modificándolos.

La siguiente ilustración representa la interacción entre la vista, el modelo, el controlador y el usuario:

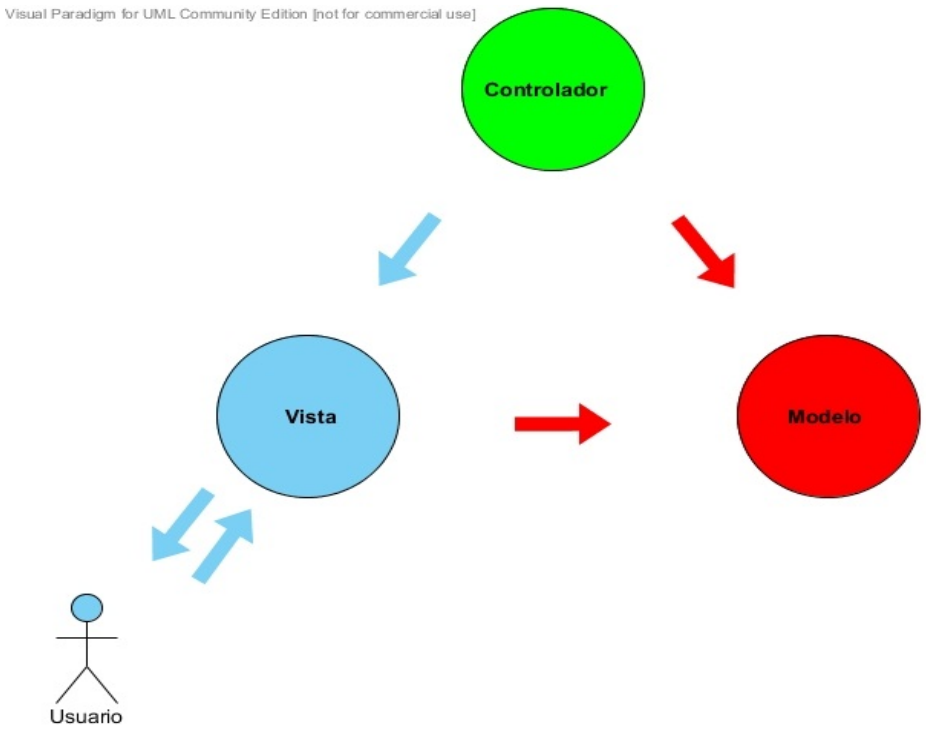


Ilustración 14: Modelo Vista Controlador

Finalmente, para clarificar más los conceptos de modelo vista y controlador se desarrolla el siguiente escenario de ejecución:

1. El usuario mediante la interfaz de línea de comandos **(vista)** introduce la operación que desea realizar, por ejemplo una operación de búsqueda.
2. Una clase encargada de parsear los comandos **(controlador)** recibe la orden introducida por el usuario mencionada en el paso anterior, la parsea, valida que tenga un formato correcto y actúa en consecuencia, en este caso realiza una operación de búsqueda.
3. Como resultado de la operación de búsqueda se obtiene un conjunto de clases que contienen información sobre los resultados de la búsqueda **(modelo)**.
4. Finalmente el controlador, utilizando los datos del modelo, informa al usuario de los resultados de la búsqueda a través de la vista.

3.3 Componentes

En este apartado se va a hablar del diseño de los componentes de la aplicación. La división de los componentes se ha realizado utilizando como criterio las distintas funciones que desempeña la aplicación. Para cada componente se mostrará un diagrama de clases. Finalmente, en cuanto a la relación de los componentes de la aplicación con el patrón MVC, tal y como se dijo en el apartado anterior la aplicación no encaja perfectamente con dicho patrón. No obstante se puede decir que la clase

CommanLineParser, descrita más adelante, será la que ejerza de controlador, la propia interfaz de línea de comandos será la vista y el modelo lo formarán las clases que almacenan los resultados de una búsqueda, la información relativa a una descarga o los resultados de la ejecución de un detector, entre otras, habiendo clases que no puedan encasillarse como modelo, vista o controlador.

La siguiente imagen se muestra como esquema de la estructura de componentes de la aplicación:

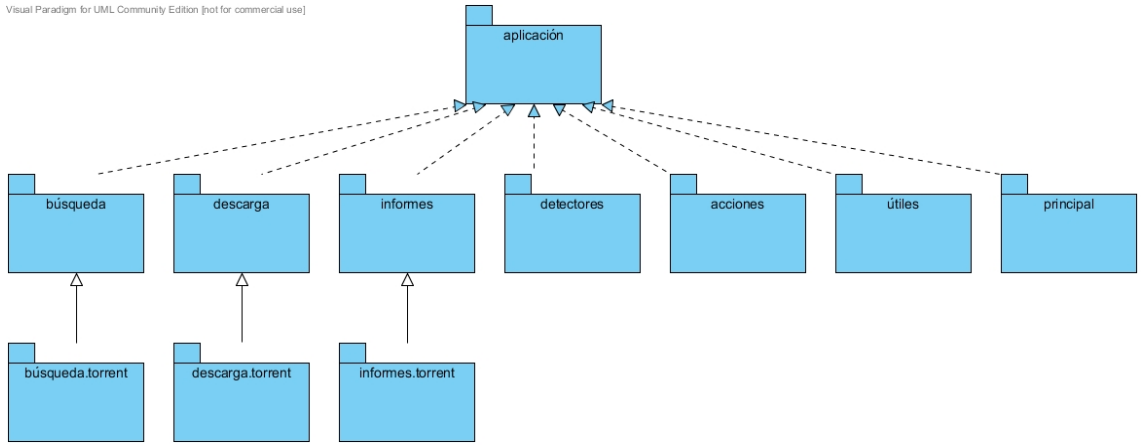


Ilustración 15: Estructura de paquetes de la aplicación

A continuación se van a desglosar los componentes que compondrán la aplicación y su función.

3.3.1 Búsqueda

Este componente es el encargado de ejecutar la función de búsqueda de archivos sobre las distintas redes P2P. Este componente se dividirá en subcomponentes, cada uno de los cuales implementará la función de búsqueda sobre una red concreta siendo la red BitTorrent la implementada inicialmente. Cada subcomponente contendrá además una clase controladora encargada de manejar todo el proceso. La funcionalidad de búsqueda en la red BitTorrent se valdrá de la función de metabúsqueda de Vuze, la cual utiliza unos archivos especiales denominados plantillas, el funcionamiento de estos archivos se detallará en el capítulo de implementación. El componente de búsqueda principal también contendrá una clase controladora que será la encargada de, según la red elegida, instanciar la clase necesaria para realizar la búsqueda y además contendrá una clase genérica para almacenar los resultados de la búsqueda en un formato común a todas las redes. A continuación se especifica un diagrama de clases del componente:

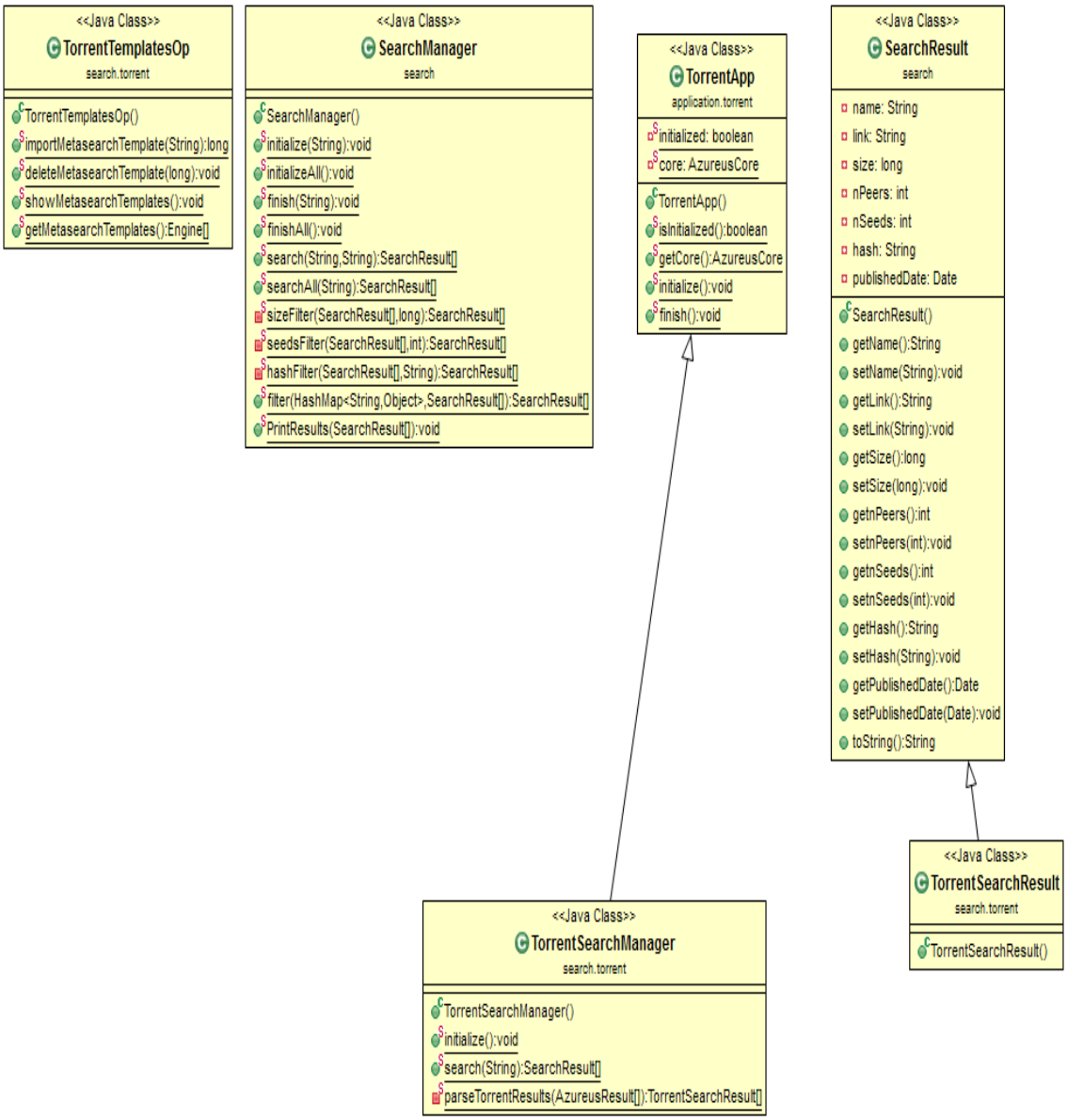


Ilustración 16: Diagrama de clases del componente Búsqueda

La función principal de las distintas clases es:

- **SearchManager:** controla todos los procesos de búsqueda para las distintas redes que soporta la aplicación. Es el núcleo de la operación de búsqueda y utiliza el resto de clases del componente para llevar a cabo dicha operación.
- **TorrentApp:** inicializa el núcleo de la parte de la aplicación relacionada con la red BitTorrent. Esta clase se utiliza en otros componentes así que sólo se comentara en la descripción de éste.
- **TorrentSearchManager:** controla e implementa el proceso de búsqueda para la red BitTorrent.
- **SearchResult:** almacena los resultados devueltos por el proceso de búsqueda para cualquier red.

- **TorrentSearchResult:** especializa la clase SearchResult y su función es extender dicha clase para incluir atributos y métodos adicionales propios de la red BitTorrent.
- **TorrentTemplatesOp:** implementa las operaciones relacionadas con las plantillas de Vuze explicadas anteriormente.

A continuación se detallan los atributos y métodos de las distintas clases:

TorrentApp	
Atributos	
initialized : boolean	Booleano para controlar si el núcleo de Azureus ha sido iniciado
core : AzureusCore	Núcleo del programa Azureus, es el elemento principal del programa y es primordial para cualquier ejecución relacionada con él
Métodos	
isInitialized() : boolean	Devuelve una referencia al atributo initialized y por lo tanto permite conocer si el núcleo de Azureus ha sido iniciado o no
getCore() : AzureusCore	Devuelve una referencia al atributo core
initialize() : void	Consulta si el núcleo de Azureus ha sido previamente inicializado y en caso negativo lo inicia
finish() : void	Consulta si el núcleo de Azureus ha sido previamente inicializado y en caso afirmativo lo para y lo destruye

Tabla 5: Clase TorrentApp

SearchManager	
Atributos	
Métodos	
initialize(String) : void	Inicializa el controlador de búsquedas de la red especificada por parámetro
initializeAll() : void	Inicializa los controladores de búsquedas de todas las redes implementadas
finish(String) : void	Finaliza el controlador de búsqueda de la red especificada por parámetro
finishAll() : void	Finaliza todos los controladores de búsquedas de todas las redes implementadas
search(String,String) : SearchResult[]	Realiza una búsqueda utilizando un determinado término de búsqueda sobre una red específica y devuelve los resultados
searchAll(String) : SearchResult[]	Realiza una búsqueda utilizando el término de búsqueda introducido por parámetro sobre todas las redes implementadas y devuelve los resultados

sizeFilter(SearchResult[],int) : SearchResult[]	Realiza un filtrado de los resultados obtenidos al realizar una búsqueda eliminando los que excedan el tamaño introducido por parámetro (especificado en bytes)
seedsFilter(SearchResult[], int) : SearchResult[]	Realiza un filtrado de los resultados obtenidos al realizar una búsqueda eliminando los que no superen un determinado número de seeds, introducido por parámetro
hashFilter(SearchResult[], String) : SearchResult[]	Realiza un filtrado de los resultados obtenidos al realizar una búsqueda eliminando los que no coincidan con el hash introducido por parámetro
filter(HashMap<String,Object>,SearchResult[]) : SearchResult[]	Permite un filtrado múltiple de los resultados. Se encarga de según los datos introducidos por parámetro llamar a los métodos de filtrado anteriores
PrintResults(SearchResult[]) : void	Imprime por consola los resultados de la búsqueda

Tabla 6: Clase SearchManager

TorrentSearchManager	
Atributos	
Métodos	
initialize() : void	Inicializa el controlador de búsquedas de la red BitTorrent
search(String) : SearchResult[]	realiza una búsqueda del término introducido por parámetro en la red BitTorrent y devuelve un array con los resultados
parseTorrentResults(AzureusResult[]) : TorrentSearchResult[]	Convierte los resultados devueltos por Azureus a un formato adecuado a la aplicación

Tabla 7: Clase TorrentSearchManager

SearchResult	
Atributos	
name : String	Nombre del resultado de búsqueda
link : String	Enlace del resultado de búsqueda
size : long	Tamaño en bytes del resultado de búsqueda
nPeers : int	Número de pares que poseen el resultado de búsqueda
nSeeds : int	Número de semillas que poseen el resultado de búsqueda
hash : String	Función hash del resultado de búsqueda
publishedDate : Date	Fecha de publicación del resultado de búsqueda

Métodos	
getName() : String	Devuelve una referencia al atributo name
setName(String) : void	Establece el valor introducido por parámetro al atributo name
getLink() : String	Devuelve una referencia al atributo link
setLink(String) : void	Establece el valor introducido por parámetro al atributo link
getSize() : long	Devuelve una referencia al atributo size
setSize(long) : void	Establece el valor introducido por parámetro al atributo size
getnPeers() : int	Devuelve una referencia al atributo npeers
setnPeers(int) : void	Establece el valor introducido por parámetro al atributo npeers
getnSeeds() : int	Devuelve una referencia al atributo nseeds
setnSeeds(int) : void	Establece el valor introducido por parámetro al atributo nseeds
getHash() : String	Devuelve una referencia al atributo hash
setHash(String) : void	Establece el valor introducido por parámetro al atributo hash
getPublishedDate() : Date	Devuelve una referencia al atributo publishedDate
setPublishedDate(Date) : void	Establece el valor introducido por parámetro al atributo publishedDate
toString() : String	Devuelve una cadena de texto representando el objeto que hace la llamada a este método

Tabla 8: Clase SearchResult

TorrentTemplatesOp	
Atributos	
Métodos	
importMetasearchTemplate(String) : long	Importa una plantilla de metabúsqueda localizada en la ruta introducida por parámetro y devuelve su identificador
deleteMetasearchTemplate(long) : void	Borra una plantilla de metabúsqueda identificada por el valor numérico introducido por parámetro
showMetasearchTemplates() : void	Muestra por consola las plantillas que se encuentran importadas en la aplicación
getMetasearchTemplates() : Engine[]	Devuelve un array con las plantillas que se encuentran importadas en la aplicación

Tabla 9: Clase TorrentTemplatesOp

3.3.2 Descarga

La estructura de este componente es muy similar a la del componente de búsqueda. Será el encargado de ejecutar la función de descarga de archivos sobre las

distintas redes P2P y también se dividirá en subcomponentes, cada uno de los cuales implementará la función de descarga sobre una red concreta. Cada subcomponente contendrá además una clase controladora encargada de manejar todo el proceso. La red BitTorrent será la que inicialmente se implemente y además se le proporcionará de funcionalidad para descargar archivos torrent (adicionalmente se permitirá su descarga mediante enlaces magnet). El componente de descarga principal también contendrá una clase controladora que será la encargada de, según la red elegida, instanciar la clase necesaria para realizar la descarga. A continuación se especifica un diagrama de clases del componente:

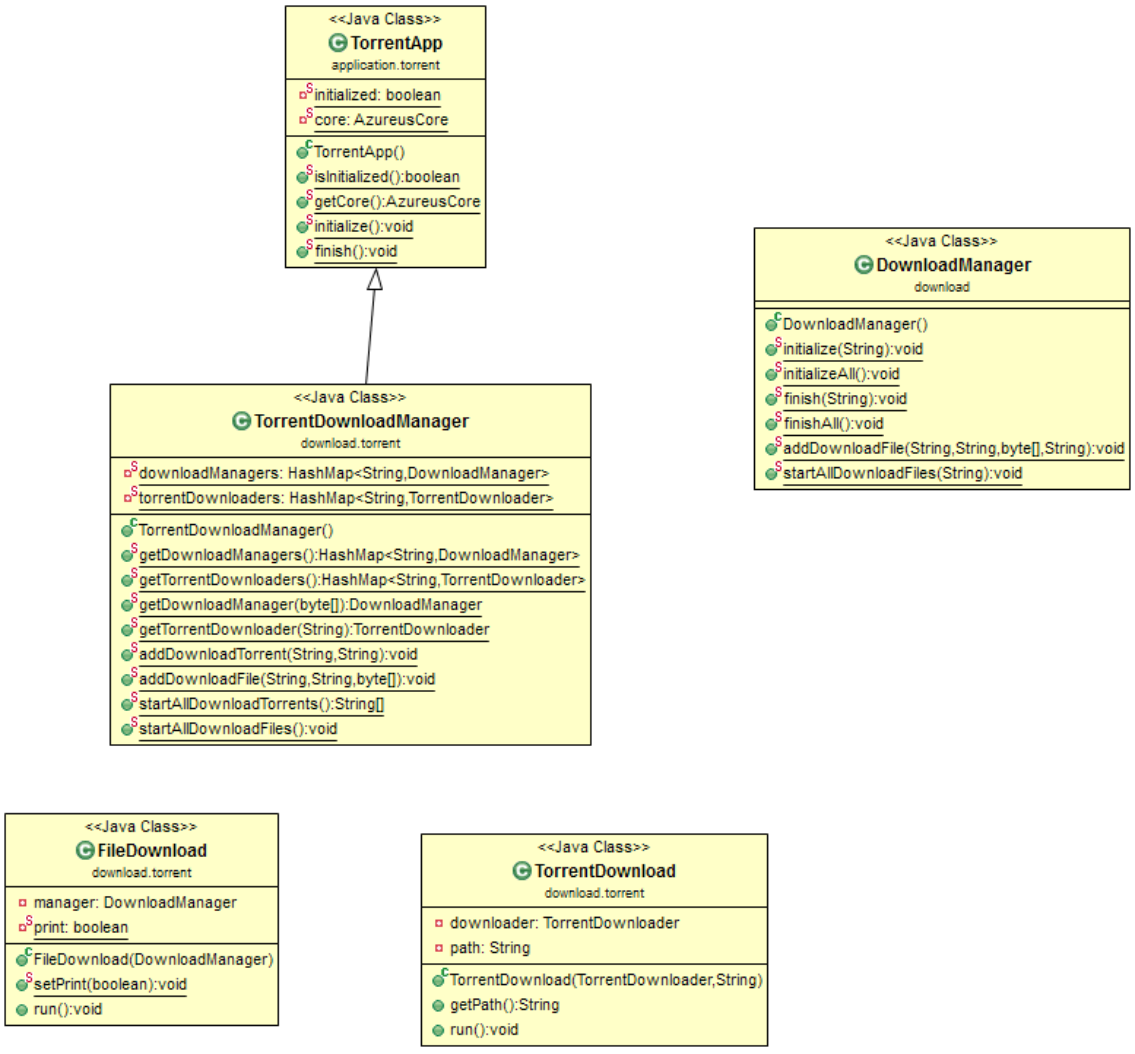


Ilustración 17: Diagrama de clases del componente Descarga

La función principal de las distintas clases es:

- **DownloadManager:** controla todos los procesos de descarga para las distintas redes soportadas por la aplicación. Es el núcleo de la operación de descarga y utiliza el resto de clases del componente para llevar a cabo dicha operación.

- **TorrentDownloadManager:** controla los procesos de descarga específicamente para la red BitTorrent.
- **TorrentDownload:** implementa el proceso de descarga de archivos torrent.
- **FileDownload:** implementa el proceso de descarga de archivos en la red BitTorrent.

A continuación se detallan los atributos y métodos de las distintas clases:

DownloadManager	
Atributos	
Métodos	
initialize(String) : void	Inicializa el controlador de descargas de la red especificada por parámetro
initializeAll() : void	Inicializa todos los controladores de descargas de las redes implementadas en la aplicación
finish(String) : void	Finaliza el controlador de descargas de la red especificada por parámetro
finishAll() : void	Finaliza todos los controladores de descargas de las redes implementadas en la aplicación
addDownloadFile(String, String, byte[], String) : void	Añade la descarga de un archivo sobre una red específica, utilizando un archivo torrent (en el caso de la red BitTorrent) identificado por una ruta y una función hash y eligiendo que se descargue en una determinada ruta
startAllDownlaodFiles(String) : void	Comienza la descarga de todos los archivos que se han programado para descargar en la red especificada por parámetro

Tabla 10: Clase DownloadManager

TorrentDownloadManager	
Atributos	
downloadManagers : HashMap<String, DownloadManager>	Contiene todos los manejadores de descargas de archivos que se están ejecutando actualmente identificados por el hash del archivo a descargar
torrentDownloaders : HashMap<String,TorrentDownloader>	Contiene todos los manejadores de descargas de archivos torrent que se están ejecutando actualmente identificados por la URL del archivo torrent a descargar
Métodos	
getDownloadManagers() : HashMap<String,DownloadManager>	Devuelve una referencia al atributo downloadManagers
getTorrentDownloaders() : HashMap<String,TorrentDownloader>	Devuelve una referencia al atributo torrentDownloaders
getDownloadManager(byte[]) : DownloadManager	Devuelve un manejador de descarga de archivo identificado por el hash

	especificado por parámetro
getTorrentDownloader(String) : TorrentDownloader	Devuelve un manejador de descarga de archivo torrent identificado por la url especificada por parámetro
addDownloadTorrent(String,String) : void	Añade la descarga de un archivo torrent localizado por su url y opcionalmente utilizando una url de referencia
addDownloadFile(String,String,byte[]) : void	Añade la descarga de un archivo utilizando un archivo torrent (en el caso de la red BitTorrent) identificado por una ruta y una función hash y eligiendo que se descargue en una determinada ruta
startAllDownloadTorrents() : String[]	Comienza la descarga de todos los archivos torrent que se han programado para descargar y una vez descargados devuelve las rutas que los localizan
startAllDownloadFiles() : void	Comienza la descarga de todos los archivos que se han programado para descargar en la red BitTorrent

Tabla 11: Clase TorrentDownloadManager

TorrentDownload	
Atributos	
downloader : TorrentDownloader	Manejador de la descarga del archivo torrent
path : String	Ruta completa del archivo torrent descargado
Métodos	
getPath() : String	Devuelve una referencia al atributo path
run() : void	Ejecuta el proceso de descarga del archivo torrent de forma paralela. Durante el proceso se imprime por consola información relacionada con la descarga del archivo torrent

Tabla 12: Clase TorrentDownload

FileDownload	
Atributos	
manager : DownloadManager	Manejador de la descarga del archivo
print : boolean	Indica si se permite la impresión por pantalla de información relacionada con el proceso de descarga. Se utiliza para no mostrar por pantalla dicha información cuando el usuario introduce una orden
Métodos	

setPrint(boolean) : void	Establece un valor para el atributo print
run() : void	Ejecuta el proceso de descarga del archivo de forma paralela. Durante el proceso se imprime por consola información relacionada con la descarga del archivo únicamente si el atributo print es true

Tabla 13: Clase FileDownload

3.3.3 Informes

Este componente será el encargado de ejecutar la función de elaborar informes. Contendrá clases que permitirán la elaboración de informes al menos en formato PDF y en formato de texto plano. Será dividido en subcomponentes, cada uno de los cuales se especializa en generar informes para una red concreta. A continuación se especifica un diagrama de clases del componente:

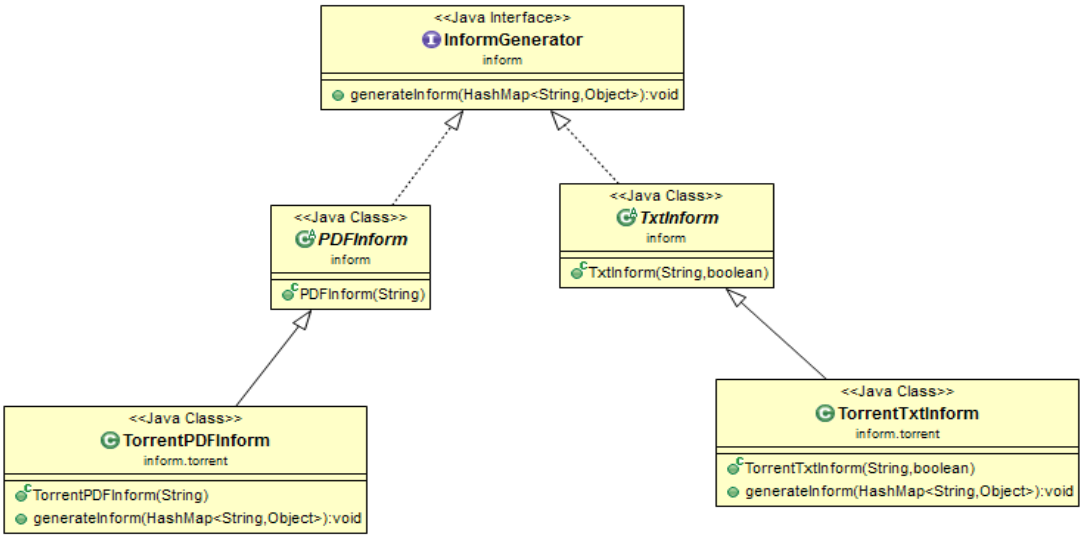


Ilustración 18: Diagrama de clases del componente Informes

La función principal de las distintas clases es:

- **InformGenerator:** interfaz java que implementan las distintas clases que elaboran informes y que contiene la declaración del método que implementa la elaboración del informe.
- **PDFInform:** implementa una clase que representa un informe en formato PDF.
- **TorrentPDFInform:** especializa la clase anterior e implementa el método de elaboración de informes específicamente para la red BitTorrent.
- **TxtInform:** implementa una clase que representa un informe en texto plano.
- **TorrentTxtInform:** especializa la clase anterior e implementa el método de elaboración de informes específicamente para la red BitTorrent.

A continuación se detallan los atributos y métodos de las distintas clases:

InformGenerator	
Atributos	
Métodos	
generateInform(HashMap<String,Object>) : void	Declaración del método que se utilizará para elaborar los informes y que deben implementar las clases encargadas de ello

Tabla 14: Clase InformGenerator

TorrentPDFInform	
Atributos	
Métodos	
generateInform(HashMap<String,Object>) : void	Construye un informe en formato PDF utilizando los datos introducidos por parámetro

Tabla 15: Clase TorrentPDFInform

TorrentTxtInfom	
Atributos	
Métodos	
generateInform(HashMap<String,Object>) : void	Construye un informe en texto plano utilizando los datos introducidos por parámetro

Tabla 16: Clase TorrentTxtInform

3.3.4 Detectores

Este componente se encargará de las funciones de la aplicación relacionadas con la detección de contenido oculto. Contendrá una clase controladora que de manera paralela a la descarga de un archivo se encargue de, de acuerdo a la configuración introducida por el usuario y según una serie de criterios tales como el porcentaje de información descargada, lanzar unos detectores u otros. Ejemplos de detectores a implementar son clases con que se valgan de diferentes técnicas de estegoanálisis para la detección del contenido oculto. El conjunto de detectores es susceptible de ser aumentado. Más adelante en el apartado **4.1.6 Detectores** se mostrará el conjunto de detectores finalmente implementado. A continuación se especifica un diagrama de clases del componente:

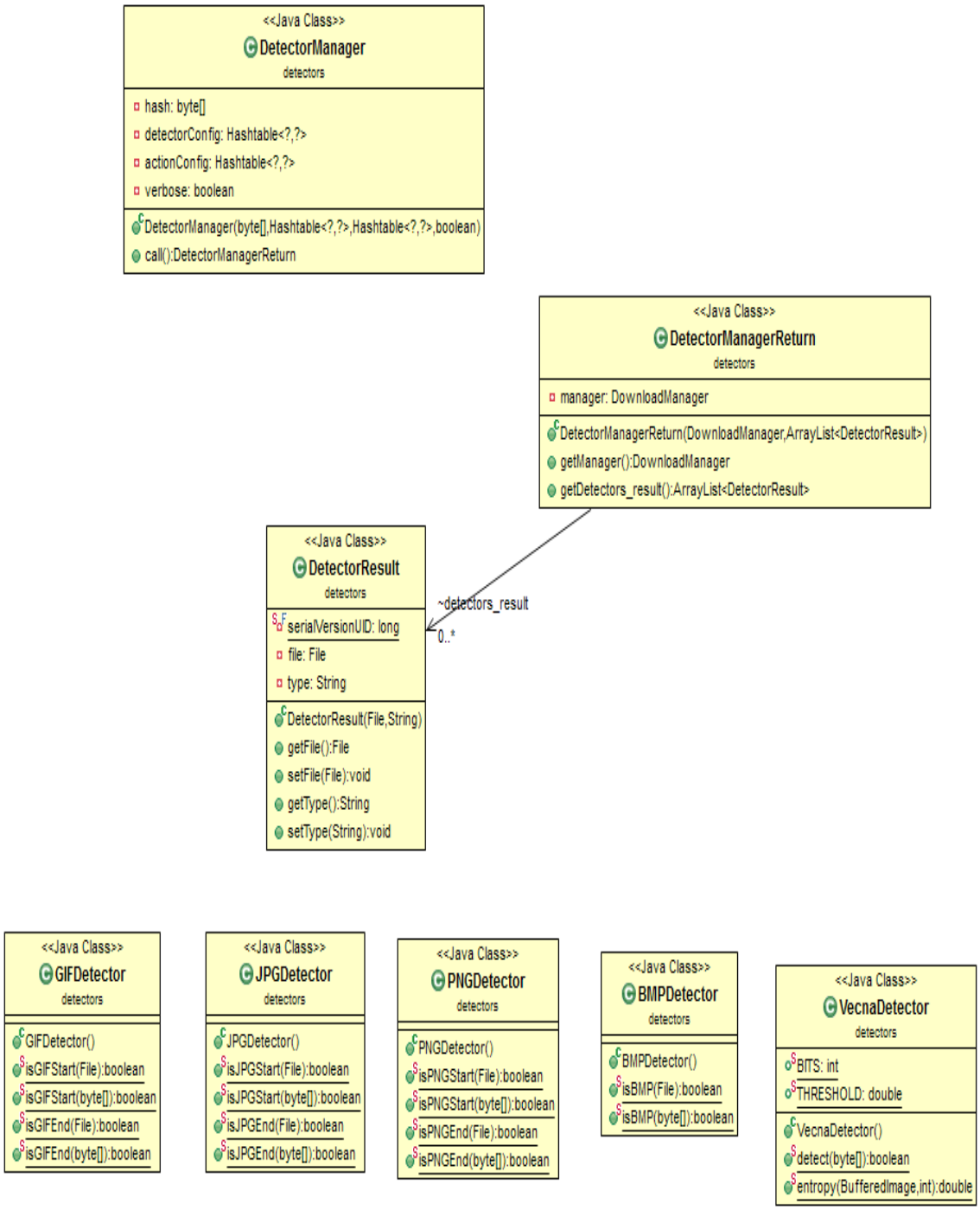


Ilustración 19: Diagrama de clases del componente Detectores

La función principal de las distintas clases es:

- **DetectorManager:** controla la ejecución de los detectores y decide cuál de ellos ejecutar de acuerdo a la configuración introducida por el usuario. Es el núcleo del componente.
- **DetectorManagerReturn:** almacena los resultados devueltos por los detectores ejecutados durante una descarga.

- **DetectorResult:** almacena el resultado de un detector sobre un archivo de una descarga. La clase DetectorManagerReturn contendrá una lista de objetos DetectorResult, uno por cada archivo que dé un resultado positivo al ejecutar un detector sobre él.
- **VecnaDetector:** detecta si un archivo descargado contiene información oculta mediante la técnica del cálculo de la entropía. El proceso de detección está especializado para el programa esteganográfico Vecna.
- **GIFDetector:** comprueba si un archivo se corresponde con un archivo del tipo imagen GIF.
- **JPGDetector:** comprueba si un archivo se corresponde con un archivo del tipo imagen JPG.
- **PNGDetector:** comprueba si un archivo se corresponde con un archivo del tipo imagen PNG.
- **BMPDetector:** comprueba si un archivo se corresponde con un archivo del tipo imagen BMP.

A continuación se detallan los atributos y métodos de las distintas clases:

DetectorManager	
Atributos	
hash: byte[]	Función hash del archivo sobre el que se está ejecutando el detector
detectorConfig : Hashtable<?,?>	Contiene toda la información sobre la configuración del detector o detectores a utilizar
actionConfig: Hashtable<?,?>	Contiene toda la información sobre la configuración de la acción o acciones a ejecutar
verbose : boolean	Indica si debe mostrar por consola información relacionada con el proceso ejecutado por los detectores
Métodos	
call() : DetectorManagerReturn	Implementa todo el proceso de ejecución de los detectores de forma paralela

Tabla 17: Clase DetectorManager

DetectorManagerReturn	
Atributos	
manager : DownloadManager	Manejador de la descarga del archivo
detectors_result : DetectorResult[]	Lista con los resultados devueltos por los detectores durante la descarga. Esta lista únicamente contendrá resultados en los que los detectores devolvieron un resultado positivo
Métodos	

getManager() : DownloadManager	Devuelve una referencia al atributo manager
getDetectors_result() : ArrayList<DetectorResult>	Devuelve una referencia al atributo detectors_result

Tabla 18: Clase DetectorManagerReturn

DetectorResult	
Atributos	
file : File	Referencia al archivo sobre el que se ejecutó el detector
type : String	Tipo de detector ejecutado
Métodos	
getFile() : File	Devuelve una referencia al atributo file
setFile(File) : void	Establece el valor especificado por parámetro al atributo file
getType() : String	Devuelve una referencia al atributo type
setType(String) : void	Establece el valor especificado por parámetro al atributo type

Tabla 19: Clase DetectorResult

VecnaDetector	
Atributos	
BITS : int	Número de bits menos significativos a analizar en el cálculo de la entropía
THRESHOLD : double	Límite para detectar un fichero como sospechoso utilizado en el cálculo de la entropía
Métodos	
detect(byte[]) : boolean	Detecta si un archive es sospechoso de contener contenido oculto o no mediante la técnica del cálculo de la entropía
entropy(BufferedImage,int) : double	Realiza el cálculo de la entropía para una imagen utilizando un número específico de bits

Tabla 20: Clase VecnaDetector

GIFDetector	
Atributos	
Métodos	
isGIFStart(File) : boolean	Comprueba si el archivo especificado por parámetro empieza como un archivo GIF
isGIFStart(byte[]) : boolean	Comprueba si el array de bytes especificado por parámetro empieza como un archivo GIF

isGIFEnd(File) : boolean	Comprueba si el archivo especificado por parámetro termina como un archivo GIF
isGIFEnd(byte[]) : boolean	Comprueba si el array de bytes especificado por parámetro termina como un archivo GIF

Tabla 21: Clase GIFDetector

JPGDetector	
Atributos	
Métodos	
isJPGStart(File) : boolean	Comprueba si el archivo especificado por parámetro empieza como un archivo JPG
isJPGStart(byte[]) : boolean	Comprueba si el array de bytes especificado por parámetro empieza como un archivo JPG
isJPGEnd(File) : boolean	Comprueba si el archivo especificado por parámetro termina como un archivo JPG
isJPGEnd(byte[]) : boolean	Comprueba si el array de bytes especificado por parámetro termina como un archivo JPG

Tabla 22: Clase JPGDetector

PNGDetector	
Atributos	
Métodos	
isPNGStart(File) : boolean	Comprueba si el archivo especificado por parámetro empieza como un archivo PNG
isPNGStart(byte[]) : boolean	Comprueba si el array de bytes especificado por parámetro empieza como un archivo PNG
isPNGEnd(File) : boolean	Comprueba si el archivo especificado por parámetro termina como un archivo PNG
isPNGEnd(byte[]) : boolean	Comprueba si el array de bytes especificado por parámetro termina como un archivo PNG

Tabla 23: Clase PNGDetector

BMPDetector	
Atributos	
Métodos	
isBMP(File) : boolean	Comprueba si el archivo especificado

	por parámetro se corresponde con un archivo BMP
isBMP(byte[]) : boolean	Comprueba si el array de bytes especificado por parámetro se corresponde con un fichero BMP

Tabla 24: Clase BMPDetector

3.3.5 Acciones

Este componente se encargará de las funciones de la aplicación relacionadas con la ejecución de acciones una vez que se ha detectado contenido oculto. Contendrá una clase controladora que de acuerdo a la configuración introducida por el usuario y de acuerdo al resultado de los detectores lance unas acciones u otras. Ejemplos de acciones a implementar son ataques de fuerza bruta o de diccionario a la posible clave utilizada para ocultar un archivo mediante alguna técnica esteganográfica. El conjunto de acciones es susceptible de ser aumentado. Más adelante en el apartado **4.1.7 Acciones** se mostrará el conjunto de acciones finalmente implementado. A continuación se especifica un diagrama de clases del componente:

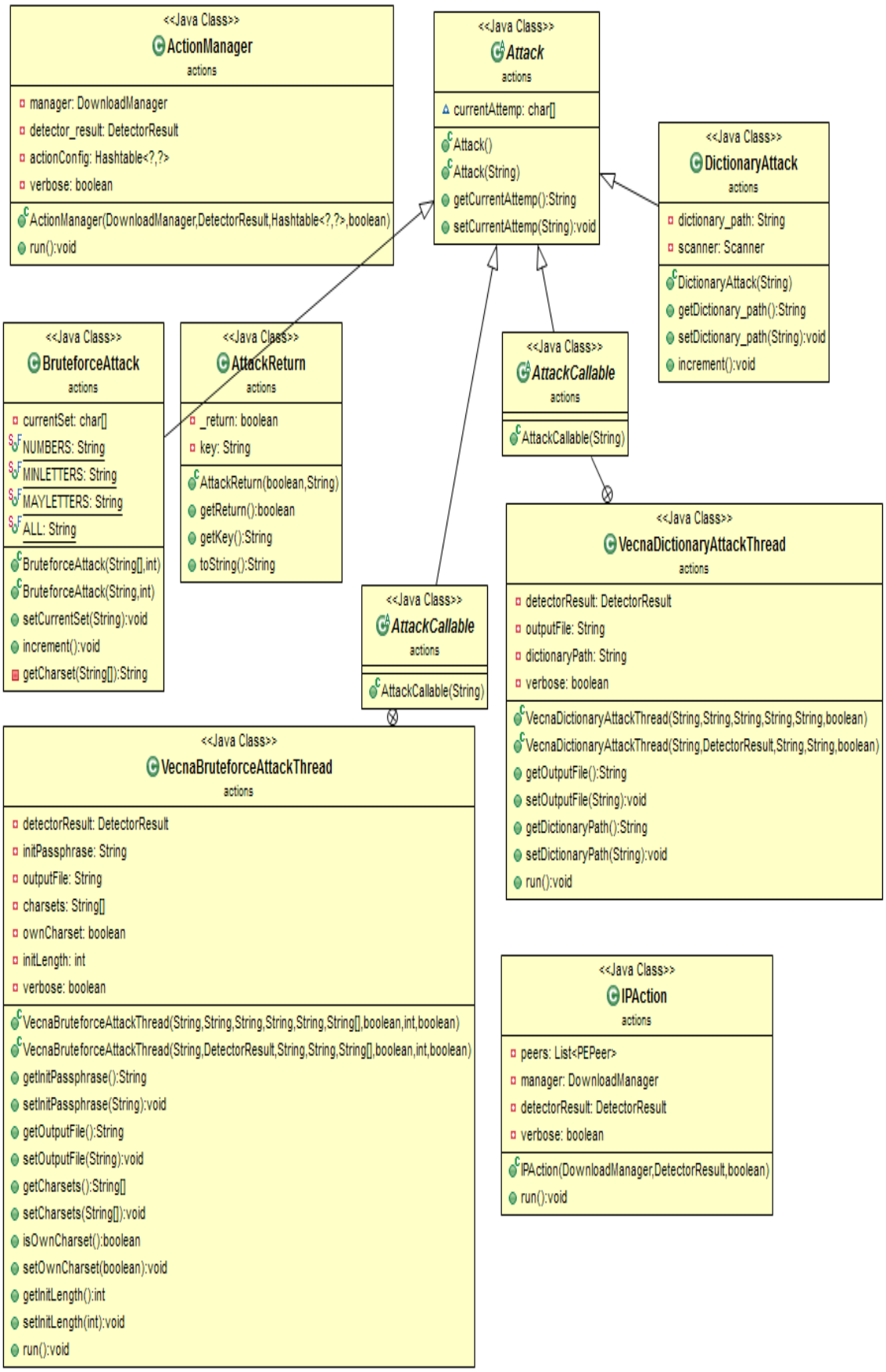


Ilustración 20: Diagrama de clases del componente Acciones

La función principal de las distintas clases es:

- **ActionManager:** controla la ejecución de las acciones y decide cuál de ellas ejecutar de acuerdo a la configuración introducida por el usuario. Es el núcleo del componente.
- **Attack:** implementa una clase que representa un ataque hacia una imagen con posible contenido oculto.
- **DictionaryAttack:** especialización de la clase Attack que implementa un ataque de diccionario.
- **BruteforceAttack:** especialización de la clase Attack que implementa un ataque de fuerza bruta.
- **VecnaDictionaryAttackThread:** utiliza la clase DictionaryAttack junto con funciones del programa especializado en esteganografía Vecna para realizar un ataque de diccionario de manera paralela a la descarga de un archivo.
- **VecnaBruteforceAttackThread:** utiliza la clase BruteforceAttack junto con funciones del programa especializado en esteganografía Vecna para realizar un ataque de fuerza bruta de manera paralela a la descarga de un archivo.
- **IPAction:** obtiene el conjunto de direcciones IP de los pares que están compartiendo un archivo.
- **AttackReturn:** almacena el resultado de un ataque realizado contra un archivo.
- **AttackCallable:** especializa la clase Attack para ejecutarla de forma paralela y se utiliza en las clases VecnaDictionaryAttackThread y VecnaBruteforceAttackThread.

A continuación se detallan los atributos y métodos de las distintas clases:

ActionManager	
Atributos	
manager : DownloadManager	Manejador de la descarga del archivo
detectors_result : DetectorResult	Resultado de la ejecución del detector que provocó la ejecución de la acción
actionConfig : Hashtable<?,?>	Contiene toda la información sobre la configuración de la acción o acciones a ejecutar
verbose : boolean	Indica si debe mostrar por consola información relacionada con el proceso ejecutado por las acciones
Métodos	
run() : void	Implementa todo el proceso de ejecución de las acciones de forma paralela

Tabla 25: Clase ActionManager

Attack	
Atributos	
currentAttemp : char[]	Intento de clave actual que está ejecutando el ataque
Métodos	
getCurrentAttemp() : String	Devuelve una referencia al atributo currentAttemp
setCurrentAttemp(String) : void	Establece el valor especificado por parámetro al atributo currentAttemp

Tabla 26: Clase Attack

DictionaryAttack	
Atributos	
dictionary_path : String	Ruta al diccionario que se utilizará en el ataque
scanner : Scannner	Objeto utilizado para leer el archive que ejercerá de diccionario
Métodos	
getDictionary_path() : String	Devuelve una referencia al atributo dictionary_path
setDictionary_path(String) : String	Establece el valor especificado por parámetro al atributo dictionary_path
increment() : void	Calcula el intento siguiente al que se está ejecutando actualmente. En el caso de un ataque por diccionario simplemente lee la siguiente palabra del archivo que está ejerciendo de diccionario

Tabla 27: Clase DictionaryAttack

BruteforceAttack	
Atributos	
currentSet : char[]	Conjunto de caracteres que se está utilizando para realizar el ataque
NUMBERS : String	Constante que representa el conjunto de caracteres de los números
MINLETTERS : String	Constante que representa el conjunto de caracteres de las letras minúsculas
MAYLETTERS : String	Constante que representa el conjunto de caracteres de las letras mayúsculas
ALL : String	Constante que representa el conjunto de todos los caracteres
Métodos	
setCurrentSet(String) : void	Establece el valor especificado por parámetro al atributo currentSet
increment() : void	Calcula el intento siguiente al que se está ejecutando actualmente de acuerdo al juego de

	caracteres que se está utilizando
getCharset(String[]) : String	Devuelve un juego de caracteres de acuerdo al array de cadenas de texto introducido por parámetro. Por ejemplo si se ejecuta getCharset(new String[]{MINLETTERS,NUMBERS}) se obtiene el juego de caracteres “abc…z0123456789”

Tabla 28: Clase BruteforceAttack

VecnaDictionaryAttackThread	
Atributos	
detectors_result : DetectorResult	Resultado de la ejecución del detector que provocó la ejecución de la acción. Contiene la ruta de la imagen descargada y sobre la cual se ejecutará el ataque y el tipo de detector que fue activado
outputFile : String	Ruta al archivo en el que se extraerá la información oculta en caso de que se encuentre
dictionaryPath : String	Ruta al diccionario que se utilizará en el ataque
verbose : boolean	Indica si debe mostrar por consola información relacionada con el proceso ejecutado por la acción
Métodos	
getOutputFile() : String	Devuelve una referencia al atributo outputFile
setOutputFile(String) : void	Establece el valor especificado por parámetro al atributo outputFile
getDictionaryPath() : String	Devuelve una referencia al atributo dictionaryPath
setDictionaryPath(String) : void	Establece el valor especificado por parámetro al atributo dictionaryPath
run() : void	Ejecuta el ataque de diccionario de manera paralela

Tabla 29: Clase VecnaDictionaryAttackThread

VecnaBruteforceAttackThread	
Atributos	
detectors_result : DetectorResult	Resultado de la ejecución del detector que provocó la ejecución de la acción. Contiene la ruta de la imagen descargada y sobre la cual se ejecutará el ataque y el tipo de detector que fue activado
initPassphrase: String	Intento inicial que se utilizará para realizar el ataque
outputFile : String	Ruta al archivo en el que se extraerá la

	información oculta en caso de que se encuentre
charsets : String[]	Conjuntos de caracteres que se utilizarán en el ataque
ownCharset : Boolean	Indica si se utilizará un charset personalizado
initLength : int	Longitud inicial de la primera clave con la que se iniciará el ataque
verbose : boolean	Indica si debe mostrar por consola información relacionada con el proceso ejecutado por la acción
Métodos	
getInitPassphrase() : String	Devuelve una referencia al atributo initPassphrase
setInitiPassphrase(String) : void	Establece el valor especificado por parámetro al atributo initPassphrase
getOutputFile() : String	Devuelve una referencia al atributo outputFile
setOutputFile(String) : void	Establece el valor especificado por parámetro al atributo outputFile
getCharsets() : String[]	Devuelve una referencia al atributo charsets
setCharsets(String[]) : void	Establece el valor especificado por parámetro al atributo charsets
isOwnCharset() : boolean	Devuelve una referencia por parámetro al atributo ownCharset
setOwnCharset(boolean) : void	Establece el valor especificado por parámetro al atributo ownCharset
getInitLength() : int	Devuelve una referencia por parámetro al atributo initLength
setInitLength(int) : void	Establece el valor especificado por parámetro al atributo initLength
run() : void	Ejecuta el ataque de fuerza bruta de manera paralela

Tabla 30: Clase VecnaBruteforceAttackThread

IPAction	
Atributos	
peers : List<PePeer>	Conjunto de pares que comparten el archivo descargado
manager : DownloadManager	Manejador de la descarga del archivo
detectors_result : DetectorResult	Resultado de la ejecución del detector que provocó la ejecución de la acción
verbose : boolean	Indica si debe mostrar por consola información relacionada con el proceso ejecutado por la acción
Métodos	
run() : void	Ejecuta el ataque de fuerza bruta de manera paralela

Tabla 31: Clase IPAction

AttackReturn	
Atributos	
_return : Boolean	Indica si el ataque encontró o no información oculta
key : String	Clave que se utilizó para realizar el ataque
Métodos	
getReturn() : boolean	Devuelve una referencia al atributo _return
getKey() : String	Devuelve una referencia al atributo key

Tabla 32: Clase AttackReturn

3.3.6 Útiles

Este componente se encargará de ejecutar funciones tales como operaciones sobre archivos, cadenas de texto, números y en definitiva cualquier función que pueda ser útil para el resto de la aplicación. En este componente se incluirán las clases encargadas de parsear y validar los parámetros de ejecución procedentes de archivos XML y de la consola. Para parsear los comandos introducidos por consola se utilizará una librería especializada como ya se ha dicho anteriormente, además para la configuración de los detectores y las acciones se creará un formato específico el cual luego será validado mediante expresiones regulares. También se elaborará un esquema XML para validar los parámetros introducidos mediante este sistema. A continuación se especifica un diagrama de clases del componente:

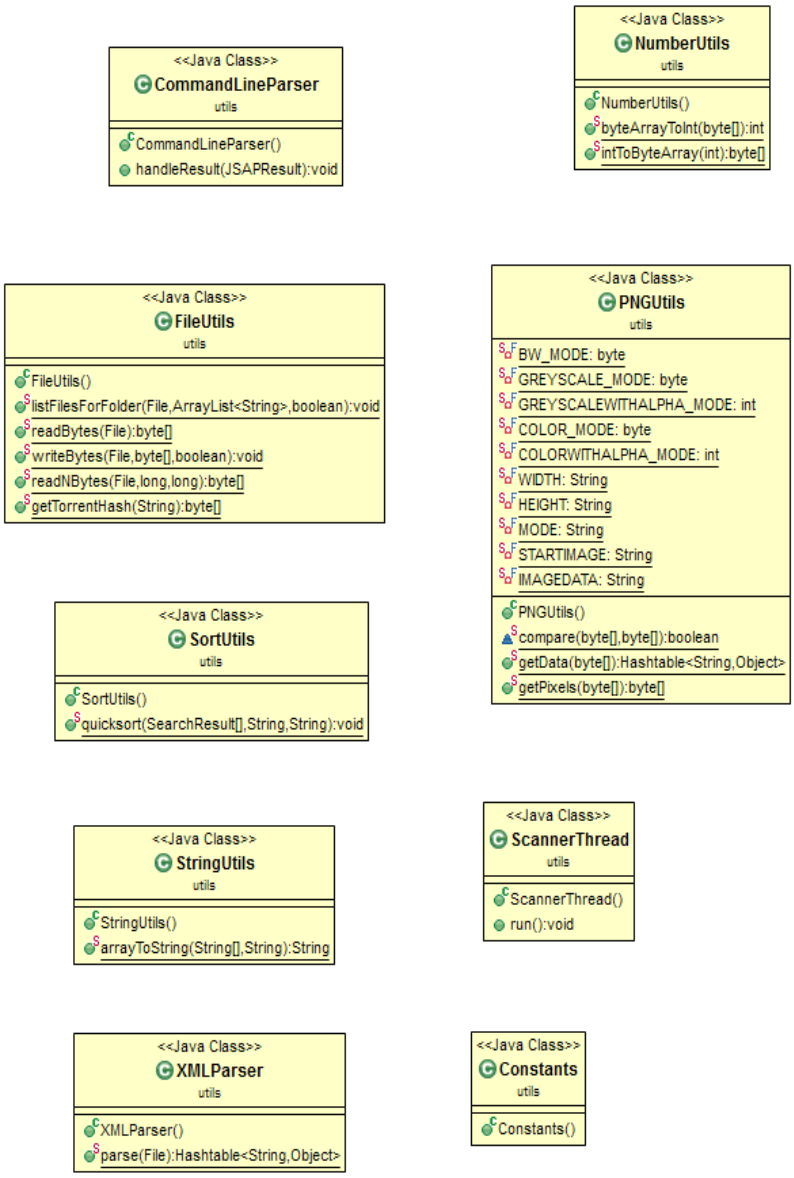


Ilustración 21: Diagrama de clases del componente Útiles

La función principal de las distintas clases es:

- **CommandLineParser:** se encarga de interpretar los comandos introducidos por el usuario y actuar en consecuencia. Como ya se ha dicho al principio de esta sección esta clase ejercerá de controlador.
- **NumberUtils:** realiza funciones relacionadas con números.
- **FileUtils:** realiza funciones relacionadas con archivos.
- **SortUtils:** realiza funciones relacionadas con la ordenación de arrays.
- **StringUtils:** realiza funciones relacionadas con cadenas de texto.
- **PNGUtils:** realiza funciones específicas para archivos PNG.
- **ScannerThread:** se encarga de leer por teclado instrucciones introducidas por el usuario relacionadas con la cancelación de las descargas o la salida de la aplicación.

- **XMLParser:** se encarga de validar que los archivos XML especificados por el usuario se adecúan al esquema XML que utiliza la aplicación y en ese caso parsea el archivo y devuelve un objeto con los datos introducidos por el usuario de forma que los pueda interpretar la aplicación.
- **Handler:** es utilizada por la clase XMLParser para realizar el parseo.
- **Constants**: almacena constantes que serán utilizadas por los distintos componentes de la aplicación. Dichas constantes se mostrarán en el **Capitulo 4: Implementación**.

A continuación se detallan los atributos y métodos de las distintas clases:

CommandLineParser	
Atributos	
Métodos	
handleResult(JSAPResult) : void	Parsea los comandos introducidos por el usuario mediante líneas de comandos y en el caso de una ejecución mediante archivo XML llama a la clase encargada de pasear este tipo de archivos. Una vez que ya obtiene la información sobre la operación que quiere realizar el usuario llama a las clases encargadas de ejecutarla

Tabla 33: Clase CommandLineParser

NumberUtils	
Atributos	
Métodos	
byteArrayToInt(byte[]) : int	Convierte el array de cuatro bytes especificado por parámetro a su correspondiente entero
intToByteArray(int) : byte[]	Convierte el entero especificado por parámetro a su correspondiente array de cuatro bytes

Tabla 34: Clase NumberUtils

FileUtils	
Atributos	
Métodos	
listFilesForFolder(File,ArrayList<String>,boolean) : void	Lista todos los archivos contenidos en un directorio, incluyendo subdirectorios y añade sus rutas a una lista. Las rutas a los subdirectorios se incluyen o no según el valor del booleano especificado por parámetro
readBytes(File) : byte[]	Lee el archivo especificado por parámetro como un array de bytes

writeBytes(File,byte[],boolean) : void	Escribe un array de bytes en un determinado archivo. Mediante el valor del booleano especificado por parámetro se decide si se va a vaciar el archivo antes de escribir en él o se va a escribir a continuación de lo que ya tenga escrito
readNBytes(File,long,long) : byte[]	Lee un determinado número de bytes a partir de una determinada posición en un archivo
getTorrentHash(String) : byte[]	Obtiene la función hash del archivo localizado por la ruta especificada por parámetro

Tabla 35: Clase FileUtils

SortUtils	
Atributos	
Métodos	
quicksort(SearchResult[],String,String) : void	Ordena unos resultados de búsqueda utilizando un determinado criterio (nombre,fecha,tamaño…) y modo de ordenación (ascendente o descendente)

Tabla 36: Clase SortUtils

StringUtils	
Atributos	
Métodos	
arrayToString(String[],String) : String	Convierte un array de cadenas de texto a su representación como cadena de texto utilizando un separador específico. Por ejemplo arrayToString(new String[]{“Pedro”,”Raul”},”|”) devuelve el siguiente String “Pedro|Raul”

Tabla 37: Clase StringUtils

PNGUtils	
Atributos	
BW_MODE : byte	Constante que representa una imagen en blanco y negro
GREYSCALE_MODE : byte	Constante que representa una imagen en escala de grises
GREYSCALEWITHALPHA_MODE : int	Constante que representa una imagen en escala de grises con transparencias
COLOR_MODE : byte	Constante que representa una imagen en color

COLORWITHALPHA_MODE : int	Constante que representa una imagen en color con transparencias
Métodos	
compare(byte[],byte[]) : boolean	Compara los arrays de bytes especificados por parámetro para comprobar si son iguales
getData(byte[]) : Hashtable<String,Object>	Obtiene información acerca de la imagen especificada por parámetro (representada como un array de bytes) como su ancho, su alto, el conjunto de píxeles de la imagen, etc
getPixels(byte[]) : byte[]	Devuelve un array de bytes con los píxeles de la imagen especificada por parámetro (representada como un array de bytes)

Tabla 38: Clase PNGUtils

ScannerThread	
Atributos	
Métodos	
run() : void	Lee las órdenes introducidas por el usuario mediante teclado de forma paralela y actúa en consecuencia.

Tabla 39: Clase ScannerThread

XMLParser	
Atributos	
Métodos	
parse(File) : Hashtable<String,Object>	Valida el archivo XML especificado por parámetro para comprobar que su formato se adecúa al esquema XML utilizado por la aplicación y después parsea dicho archivo y devuelve un objeto con los datos introducidos por el usuario de manera que los pueda entender la aplicación

Tabla 40: Clase XMLParser

3.3.7 Principal

Este componente será el que sirva como punto de entrada de la aplicación y el que realice las llamadas a las clases que implementan las distintas funcionalidades de la aplicación. A continuación se especifica un diagrama de clases del componente:

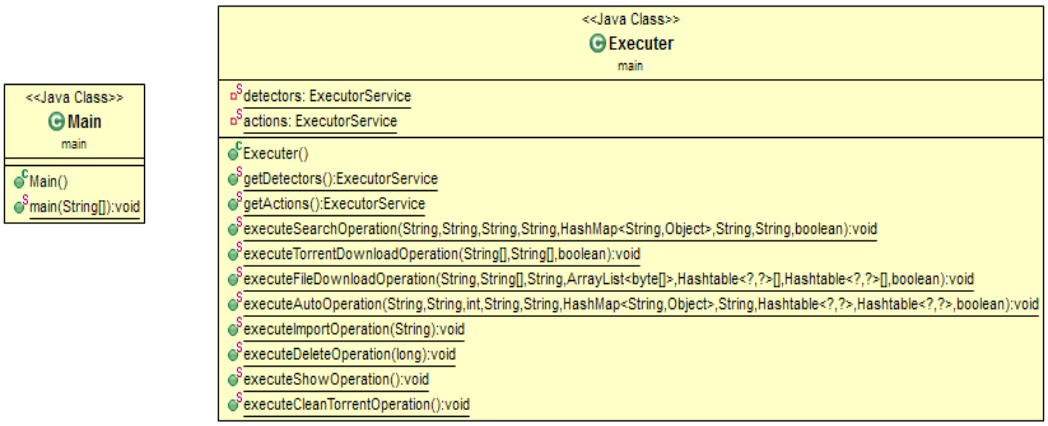


Ilustración 22: Diagrama de clases del componente Principal

La función principal de las distintas clases es:

- **Main:** sirve como punto de entrada de la aplicación y es en esta clase donde se inician todas las ejecuciones de ésta.
- **Executer:** recoge todas las operaciones que puede ejecutar la aplicación y se encarga de llamar a los distintos componentes para ejecutar una determinada operación.

A continuación se detallan los atributos y métodos de las distintas clases:

Main	
Atributos	
Métodos	
main(String[]) : void	Punto de entrada de la aplicación, realiza las labores de configuración necesarias y llama a la clase encargada de parsear los comandos introducidos por el usuario

Tabla 41: Clase Main

Executer	
Atributos	
detectors : ExecutorService	Pool de hilos que se utiliza para la ejecución de los detectores
actions : ExecutorService	Pool de hilos que se utiliza para la ejecución de las acciones
Métodos	
getDetectors() : ExecutorService	Devuelve una referencia al atributo detectors
getActions() : ExecutorService	Devuelve una referencia al atributo actions
executeSearchOperation(String,String,String,String,HashMap<String,Object>,String,String,boolean) : void	Ejecuta una operación de búsqueda utilizando un determinado término de búsqueda sobre una red específica, opcionalmente ordenando los resultados según algún criterio y modo de ordenación, opcionalmente filtrándolos de

	acuerdo a unos datos especificados por el usuario y escribiendo los resultados de la búsqueda en un informe de un determinado tipo y localizado por una ruta específica. El booleano introducido por parámetro se utiliza para indicar si se desea imprimir por consola información adicional sobre el proceso
executeTorrentDownloadOperation(String[],String[],boolean) : void	Ejecuta la descarga de un conjunto de archivos torrent localizados por sus URLs y opcionalmente utilizando URLs de referencia. El booleano introducido por parámetro se utiliza para indicar si se desea imprimir por consola información adicional sobre el proceso
ExecuteFileDownloadOperation(String,String[],String,ArrayList<byte[]>,Hashtable<?,?>[],Hashtable<?,?>[],boolean : void	Ejecuta el proceso de descarga de un conjunto de archivos sobre una red específica, utilizando sus archivos torrent (en el caso de la red BitTorrent) identificados por una ruta y una función hash específicos, eligiendo que se descargue en una ruta específica y utilizando una determinada configuración de detectores y acciones. El booleano introducido por parámetro se utiliza para indicar si se desea imprimir por consola información adicional sobre el proceso
executeAutoOperation(String,String,int, String,String, HashMap>String,Object>,String,Hashtable<?,?>,Hashtable<?,?>,boolean) : void	Ejecuta el proceso de operación automatizada. Este proceso consiste en que la aplicación inicialmente ejecuta una operación de búsqueda equivalente a la que se ha explicado anteriormente. Una vez ha finalizado la búsqueda de resultados la aplicación comienza la operación de descarga de un determinado número de ellos. El booleano introducido por parámetro se utiliza para indicar si se desea imprimir por consola información adicional sobre el proceso
executeImportOperation(String) : void	Ejecuta la operación de importar una plantilla de metabúsqueda localizada por la ruta especificada por parámetro
executeDeletecOperation(long) : void	Ejecuta el borrado de una plantilla de metabúsqueda identificada por el valor numérico especificado por parámetro
executeShowOperation() : void	Muestra las plantillas de metabúsqueda importadas en la aplicación
executeCleanTorrentOperation() : void	Cancela todas las descargas que se estén ejecutando y borra toda la información asociada.

Tabla 42: Clase executer

3.4 Diagramas de secuencia

A continuación se van a mostrar diagramas de secuencia de algunas de las operaciones que puede llevar a cabo la aplicación:

3.4.1 Operación de búsqueda

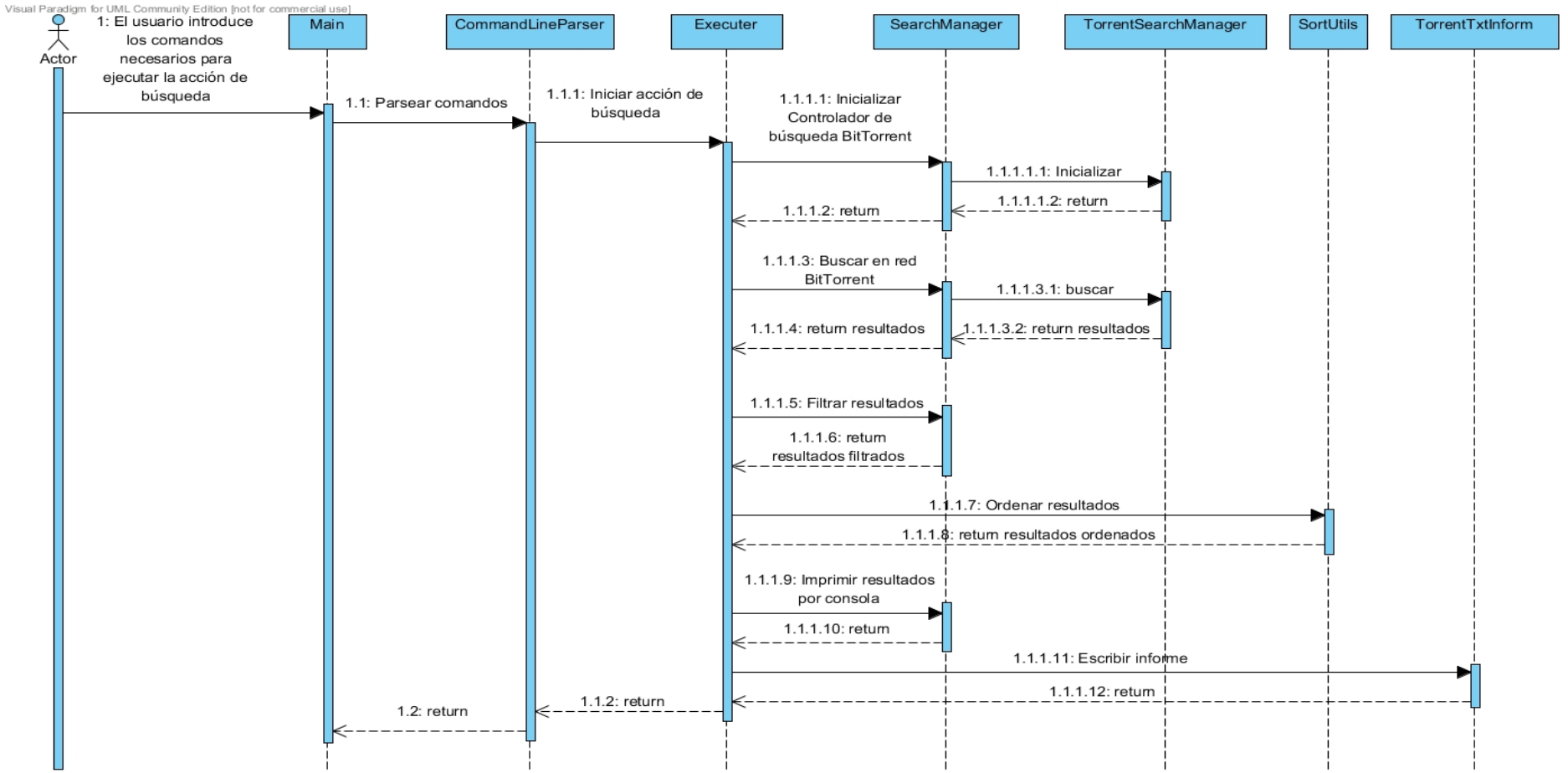


Ilustración 23: Diagrama de secuencia de la operación de búsqueda

3.4.2 Operación de descarga

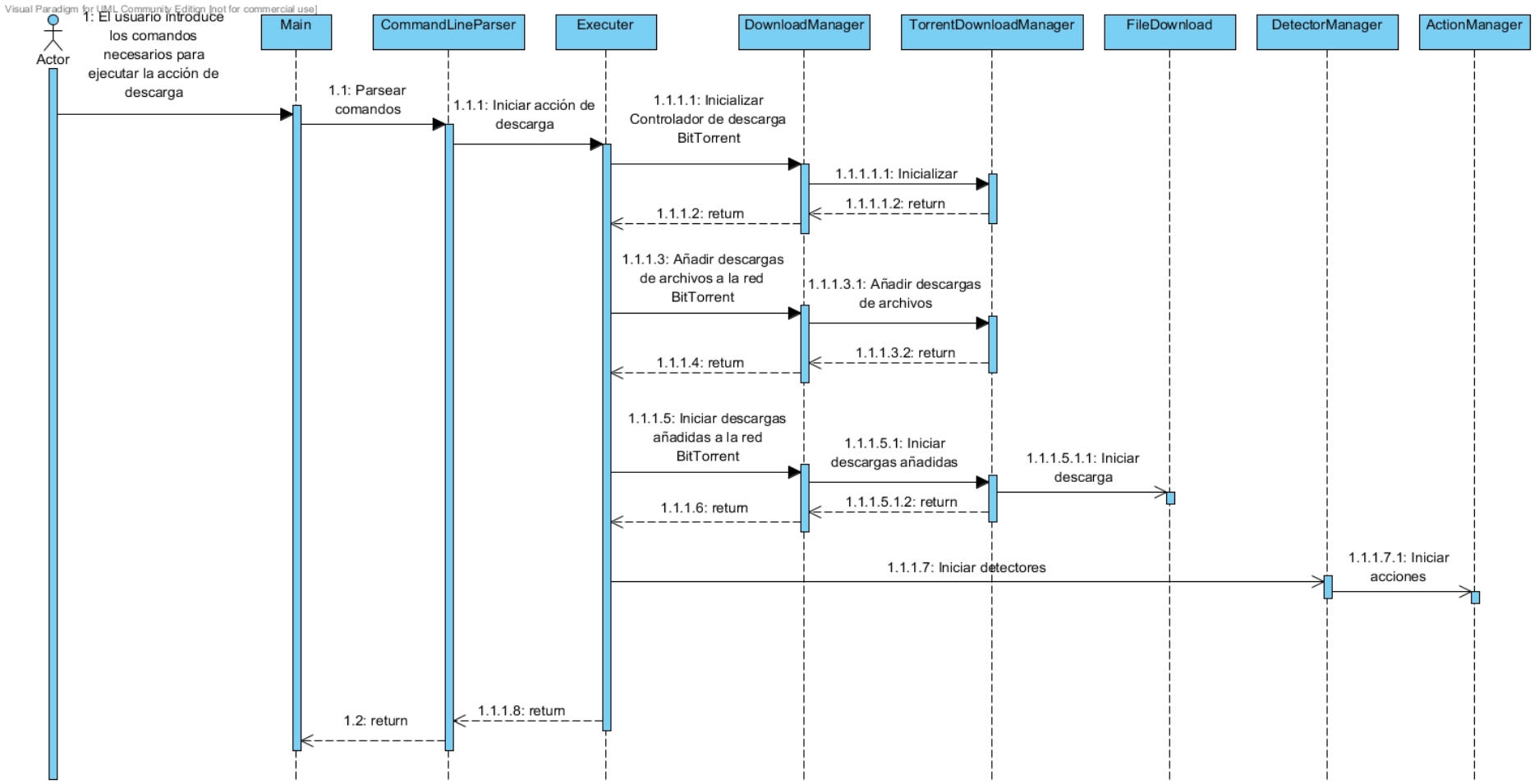


Ilustración 24: Diagrama de secuencia de la operación de descarga

Capitulo 4: Implementación

En este capítulo del proyecto se va a profundizar en temas relacionados con la implementación de la aplicación desarrollada. El capítulo se dividirá en dos partes: inicialmente, se detallarán aspectos de la implementación de las principales funciones que puede realizar la aplicación y en la segunda parte se mostrarán los resultados del plan de pruebas realizado en el capítulo de análisis.

4.1 Aspectos de la implementación

En esta sección se detallarán los aspectos principales de implementación de la aplicación desarrollada. Ésta se trata de una aplicación escrita en el lenguaje de programación Java y por lo tanto está estructurada en paquetes, los cuales se corresponden con el diseño de componentes realizado en el capítulo anterior. La aplicación se ha desarrollado de manera modular en la medida de lo posible haciendo más fácil la inclusión de nuevas funcionalidades como nuevos detectores o acciones, nuevas redes P2P soportadas, etc. Además, tal y como se dijo en el capítulo de diseño, la aplicación utiliza el programa Vuze a modo de librería externa como apoyo para el desarrollo de las funciones genéricas de un cliente P2P, como son la búsqueda y compartición de archivos en la red.

A continuación se van a mostrar los aspectos de implementación de las principales operaciones ejecutadas por la aplicación detallando las partes de código que resultan más importantes. Todas las operaciones se ejecutan sobre la red BitTorrent ya que es la única que se encuentra implementada inicialmente.

4.1.1 Parseo de los comandos de usuario

Como ya se dijo en el capítulo de diseño, la aplicación funciona mediante una interfaz de línea de comandos por lo que se necesita de un mecanismo para interpretar dichos comandos. El parseo de los comandos del usuario es una parte esencial en la aplicación desarrollada ya que es mediante este sistema mediante el cual se interactuará con el usuario, es decir mediante el cual éste dirá que operaciones quiere que realice la aplicación. Para el desarrollo de esta función se ha utilizado la librería java *JSAP* (Java Simple Argument Parser) especializada en el parseo de argumentos especificados por el usuario mediante línea de comandos. Mediante esta librería se puede especificar el tipo de los parámetros que debe introducir el usuario, valores por defecto, ayuda a mostrar en caso de que el usuario la solicite, etc.

Para ejecutar cualquier operación primero se debe especificar el modo de ejecución a utilizar por la aplicación mediante el argumento *mode* (-m). La aplicación

tiene dos modos de ejecución: por comandos (command) o mediante archivos XML (xml). En el **Anexo A: Manual de la aplicación** se detallará el modo de uso de ambos modos. La aplicación además cuenta con el comando *help* (--help) para solicitar ayuda sobre el funcionamiento de la aplicación y el comando *verbose* (-v|--verbose) para solicitar que se informe información adicional acerca del proceso que se está realizando.

4.1.1.1 Modo de funcionamiento

El parseo de los comandos es llevado a cabo por la clase *CommandLineParser* y su funcionamiento consiste en lo siguiente:

1. Se crea una instancia de la clase *CommandLineParser*.
2. Se parsean los comandos introducidos por el usuario, los cuales se corresponden con la variable *args* del método *main*.
3. Una vez completado el parseo se validan si los comandos tienen un formato correcto y se procede a la ejecución de la operación deseada. Dentro del proceso de validación se comprueba si el usuario desea ejecutar la aplicación mediante archivos XML, en ese caso se procederá al parseo del archivo XML y una vez completado el proceso se continuará con la validación.

Desde el punto de vista de código lo anterior se traduce en:

```
public static void main(String[] args) {
        …
        CommandLineParser parser = new CommandLineParser();
        JSAPResult result = parser.parse(args);
        parser.handleResult(result);
}
```

La primera línea de código se corresponde con la instancia de la clase *CommandLineParser*, después se llama al método *parse* de dicha clase para parsear la variable *args*, la cual contiene los argumentos que el usuario introdujo al ejecutar la aplicación. Como resultado del método *parse* se obtiene un objeto *JSAPResult*, el cual contiene una lista con todos los comandos introducidos debidamente parseados. Finalmente se llama al método *handleResult*[15] el cual valida dichos comandos y ejecuta la operación deseada. A continuación se muestra un ejemplo sencillo y resumido del parseo y la validación de los comandos correspondientes a una operación de búsqueda (en este apartado sólo se indican los comandos necesarios para ejecutar dicha operación pero más adelante se explicará su significado).

[15] El método *handleResult* es de creación propia, es decir, es ajeno a la librería *JSAP*.

Un ejemplo de una posible ejecución de una operación de búsqueda seria el dado por la siguiente instrucción: "java –jar stegocrawler.jar –m command –o search –n torrent –s "prueba" –i txt –-out informe".

El proceso de ejecución sería el siguiente:

1. La variable *args* toma el siguiente valor: "[-m, command, -o, search, -n, torrent, -s, prueba, -i, txt, --out, informe]"
2. Se llama al método de parseo y como resultado se tiene un objeto *JSAPResult* con la siguiente forma:

Comando	Valor
mode	command
operation	search
network	torrent
searchQuery	prueba
informType	txt
out	informe

Tabla 43: Ejemplo de objeto JSAPResult

3. Se llama al método *handleResult* para validar los comandos. A continuación se muestra pseudocódigo del proceso de validación:

```
public void handleResult(JSAPResult result) {
    …
    String mode = result.getString("mode");
    //se comprueba si el usuario está ejecutando la aplicación en modo por comandos
    IF (mode == "command") {
        String operation = result.getString("operation");
        if(operation == "search")){
            boolean error = false;
            //se comprueba que el usuario haya introducido los parámetros necesarios
            if(results.contains("network") == false || result.contains("searchQuery") ==
            false || result.contains("informType") == false){
                //de no ser así se muestra un error
                error = true;
                showError();
            }
            else{
                String network = result.getString("network");
                String searchQuery = result.getString("searchQuery");
                String informType = result.getString("informType");
                //en este caso ya que el usuario no introdujo un valor para el parámetro
                out este tomará el valor "searchInform" por defecto
                String out = result.getString("out");
```

```
                    //se construye la variable que contiene la ruta donde se almacenará el
                    informe, en este caso toma el valor "informs\searchs\informe"
                    String informPath = INFORMS_FOLDER + FILE_SEPARATOR + "Searchs" +
                        FILE_SEPARATOR + out;
                    //se comprueba si el usuario introdujo un valor válido para el parámetro
                    network, es decir, si éste se corresponde con una de las redes
                    implementadas
                    if(binarySearch(VALID_NETWORKS,network) == false) {
                        //de no ser así se muestra un error
                        error = true;
                        showError();
                    }
                    //se comprueba si el usuario introdujo un valor válido para el parámetro
                    searchquery, es decir, si éste no es una cadena de texto vacía
                    if(searchQuery == "") {
                        //si es así se muestra un error
                        error = true;
                        showError();
                    }
                    //se comprueba si el usuario introdujo un valor válido para el parámetro
                    informType, es decir, si éste se corresponde con una de los tipos de informe
                    implementados
                    if(binarySearch(VALID_INFORM_TYPES,informType) == false) {
                        //de no ser así se muestra un error
                        error = true;
                        shorError();
                    }
                    if(error == false){
                        executeSearchOperation(network,searchQuery,informType,informPat
                        h);
                    }
                }
            }
            …
        }
        …
}
```

4.1.2 Clase Constants

La aplicación implementa una clase denominada *Constants* que contiene un conjunto de constantes que son utilizadas por el resto de clases de la aplicación. Esta clase facilita la programación ya que todas las clases tienen acceso a ella y un cambio en ella es visible por el resto de clases de la aplicación. A continuación se muestra el conjunto de constantes contenidas en la clase *Constants*, esta tabla servirá para entender mejor el código mostrado para detallar los aspectos de implementación de las distintas operaciones:

Nombre	Valor
FILE_SEPARATOR	System.*getProperty*("file.separator")[16]
APP_NAME	"stegocrawler"
INFORMS_FOLDER	""
DEFAULT_DOWNLOADS_FOLDER	""
TORRENTS_FOLDER	""[17]
SETTINGS_FOLDER	System.*getProperty*("user.dir") + *FILE_SEPARATOR* + "Settings"
MODE	"mode"
COMMAND_MODE	"command"
XML_MODE	"xml"
MODES	{COMMAND_MODE,XML_MODE}
OPERATION	"operation"
AUTO_OPERATION	"auto"
SEARCH_OPERATION	"search"
DOWNLOAD_OPERATION	"download"
IMPORT_OPERATION	"import"
DELETE_OPERATION	"delete"
SHOW_OPERATION	"show"
CLEAN_OPERATION	"clean"
OPERATIONS	{SEARCH_OPERATION, DOWNLOAD_OPERATION, AUTO_OPERATION, IMPORT_OPERATION, DELETE_OPERATION, SHOW_OPERATION, CLEAN_OPERATION}
TORRENT	"torrent"
ALL	"all"
NETWORKS	{ALL,TORRENT}
PDF	"pdf"
TXT	"txt"
INFORM_TYPES	{PDF,TXT}
NAME	"name"
LINK	"link"
DATE	"date"
SIZE	"size"
NPEERS	"npeers"
NSEEDS	"nseeds"
ASCENDENT	"ascendent"
DESCENDENT	"descendent"
NUMBER	"number"
SORT_MODES	{ASCENDENT,DESCENDENT}
SORTS	{DATE,NAME,NPEERS,NSEEDS,SIZE}
PATH	"path"

[16] Mediante esta propiedad del sistema se obtiene el carácter que separa los componentes en la ruta de un archivo según el SO que se está utilizando. En Windows se utiliza el carácter "\" y en los sistemas tipo Unix se utiliza el carácter "/".

[17] Las rutas de los informes, de descarga de archivos torrent y de descarga de archivos por defecto se configuran mediante el sistema de configuración del cual se hablará más adelante.

ID	“id”
NETWORK	“network”
URL	“url”
INFORM_TYPE	“informType”
RESULTS	“results”
HASH	“hash”
SORT	“sort”
SORT_MODE	“sortMode”
REFERRER	“referrer”
SEARCHQUERY	“searchQuery”
DETECTOR_CONF	“detectorConf”
ACTION_CONF	“actionConf”
BMP	“bmp”
GIF	“gif”
JPG	“jpg”
PNG	“png”
VECNA	“vecna”
DETECTOR_TYPES	{VECNA}
GET_OWNER_IPS_ACTION	“ip”
BRUTEFORCE_ACTION	“bruteforce”
DICTIONARY_ACTION	“dictionary”
ACTION_TYPES	{BRUTEFORCE_ACTION, DICTIONARY_ACTION, GET_OWNER_IPS_ACTION}
MAGICNUMBER_JPG_START	“ffd8”
MAGICNUMBER_JPG_END	“ffd9”
MAGICNUMBER_PNG_START	"89504e470d0a1a0a"
MAGICNUMBER_PNG_END	"49454e44ae426082"
MAGICNUMBER_GIF89A	"474946383961"
MAGICNUMBER_GIF87A	"474946383761"
MAGICNUMBER_GIF_END	“3b”
MAGICNUMBER_BMP_START	“424d”

Tabla 44: Tabla de constantes

4.1.3 Operación de búsqueda

El objetivo de la operación de búsqueda es la recopilación de información acerca de archivos torrent, con la intención de que sean descargados en un futuro. Dicha función se apoya en el uso de la función metabúsqueda del programa Vuze, la cual tal y como se explicó en la sección componentes del capítulo de diseño se vale de unos archivos especiales denominados plantillas. En este apartado se tratarán dos cuestiones, primero se detallarán los detalles de implementación de la función de búsqueda en sí y posteriormente se hará lo mismo con las funciones relacionadas con las plantillas de metabúsqueda.

4.1.3.1 Modo de funcionamiento

Al igual que el resto de las operaciones ejecutadas por la aplicación la operación de búsqueda es llevada a cabo por la clase *Executer*, la cual se apoya en otras cases para llevar a cabo la operación. El conjunto de clases que intervienen en el proceso es el siguiente:

Clases: Main, CommandLineParser, Executer, TorrentApp, SearchManager, TorrentSearchManager, SearchResult, TorrentSearchResult, SortUtils**,** TorrentTxtInform y TorrentPDFInform.

El funcionamiento de la operación a grandes rasgos es el siguiente:

1. Inicialmente, el usuario introduce los argumentos necesarios para ejecutar la operación de búsqueda, estos se parsean y validan, tal y como se vio en el apartado de parseo de los comandos de usuario, y llama a la clase *Executer* para que inicie la operación de búsqueda. Los argumentos necesarios para ejecutar cada operación se pueden ver en el **Anexo A: Manual de la aplicación**.
2. Como primer paso a realizar en la operación de búsqueda se inicializa la clase controladora del proceso de búsqueda, la clase *SearchManager*, ésta a su vez inicializará la clase controladora del proceso de búsqueda sobre una red concreta, en el caso de la red BitTorrent se trata de la clase *TorrentSearchManager.*
3. Se invoca al método *search* de la clase *SearchManager*, la cual a su vez llamará al correspondiente método *search* de la clase *TorrentSearchManager*. Una vez que ha terminado el proceso de búsqueda se obtiene un conjunto de objetos *SearchResult*.
4. Una vez que se han obtenido los resultados se procede al proceso de filtrado de los mismos en caso de que el usuario lo haya especificado.
5. A continuación se procede a la ordenación de los resultados según el criterio y el modo especificados por el usuario, si éste no ha especificado ninguno se utilizan unos por defecto.
6. En el caso de que el usuario haya solicitado ejecutar la aplicación en modo de salida extensa se imprimen por consola los resultados.
7. Finalmente se procede al proceso de elaboración del informe de búsqueda.

A continuación se muestra un diagrama de actividad de la operación para comprender mejor el proceso.

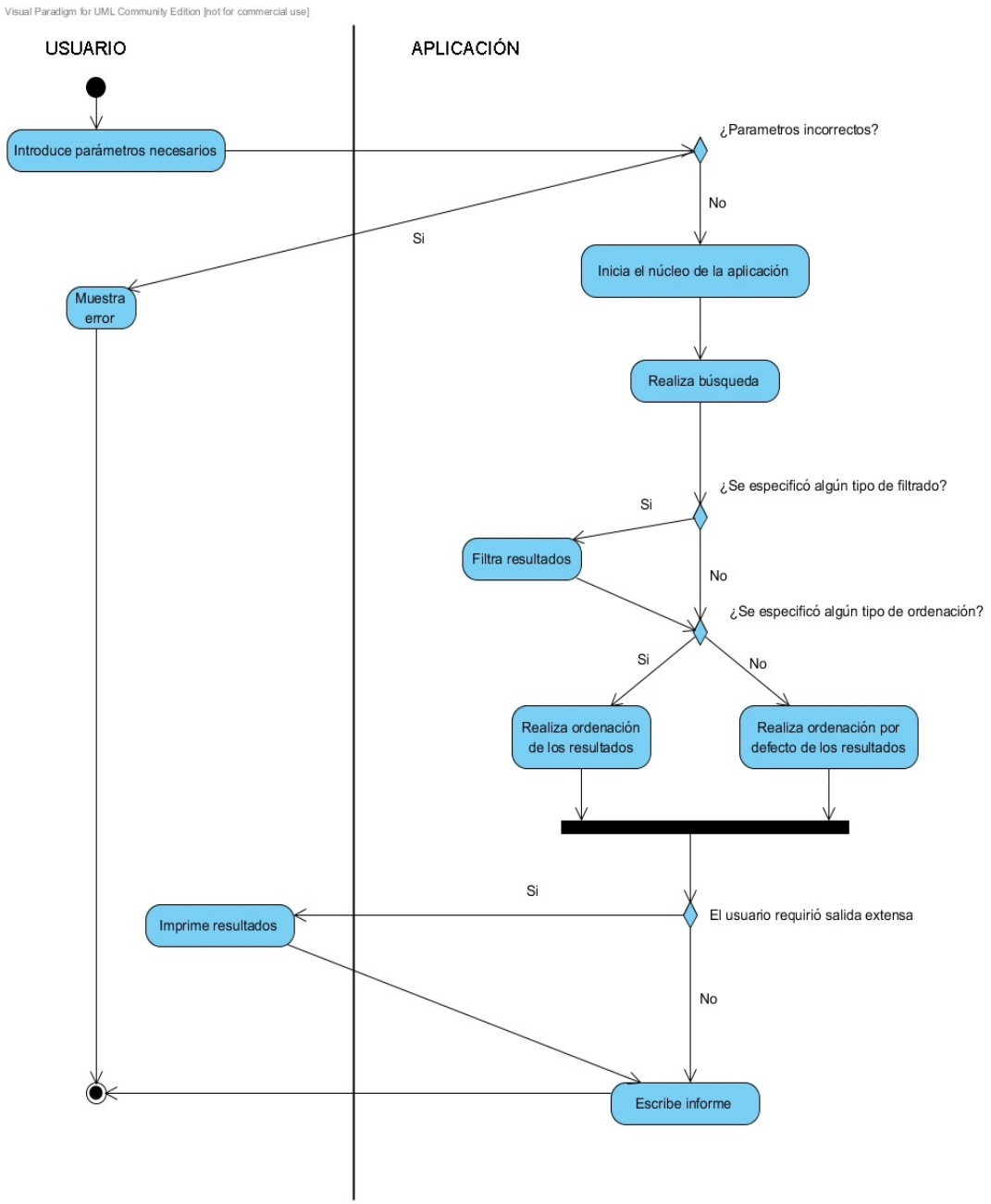


Ilustración 25: Diagrama de actividad de la operación de búsqueda

Finalmente se van a explicar algunas partes del proceso más detalladamente.

4.1.3.1.1 Proceso de búsqueda

El proceso de búsqueda de resultados se controla mediante las clases *SearchManager* y *TorrentSearchManager*. La aplicación contiene una clase controladora de búsquedas para cada red soportada, inicialmente la única red implementada es la red BitTorrent y la clase controladora de búsquedas para esta red

es la clase *TorrentSearchManager*. Esta clase se encarga de, mediante apoyo del programa Vuze, realizar el proceso de búsqueda y obtener un conjunto de resultados en el formato entendido por Vuze. A continuación se recorren estos resultados y se convierten a objetos *TorrentSearchResult*, los cuales son una especialización de la clase *SearchResult* y su función es almacenar información acerca de dichos resultados como su nombre, el enlace para descargarlo, su tamaño, etc. Por otra parte la clase *SearchManager* se encarga de, según la red a utilizar especificada por el usuario, inicializar una clase controladora u otra e invocar al método de búsqueda de dicha clase.

A continuación se muestra el código de los métodos *initialize* y *search*[18].

```
public static void initialize(String p2pnetwork){
    if(p2pnetwork.equalsIgnoreCase(Constants.TORRENT)){
        TorrentSearchManager.initialize();
    }
    else if(p2pnetwork.equalsIgnoreCase(Constants.ALL)){
        initializeAll();
    }
}
```

```
public static SearchResult[] search(String searchText,String p2pnetwork){
    if(p2pnetwork.equalsIgnoreCase(Constants.TORRENT)){
        return TorrentSearchManager.search(searchText);
    }
    else if(p2pnetwork.equalsIgnoreCase(Constants.ALL)){
        return searchAll(searchText);
    }
    else{
        return null;
    }
}
```

Como se puede comprobar ambos métodos funcionan de la misma forma, reciben por parámetro la red P2P a utilizar y de acuerdo a esta información llaman a los métodos *initialize* o *search* de la clase correspondiente. La implementación de dichos métodos no se mostrará debido a que únicamente se limitan a realizar llamadas a métodos de Vuze y no se consideran relevantes. Además en el código anterior se observa que se realizan llamadas a unos métodos denominados *initializeAll* y *searchAll*, estos métodos le proporcionan a la aplicación la posibilidad de realizar búsquedas en la totalidad de las redes implementadas. A continuación se muestra el código de dichos métodos.

[18] En caso de duda sobre cualquier porción de código se recomienda revisar el apartado **3.3 Componentes** ya que hay se muestra el diseño de las clases que forman la aplicación.

```
public static void initializeAll(){
	for(String network : Constants.NETWORKS){
		if(network.equalsIgnoreCase(Constants.ALL) == false){
			initialize(network);
		}
	}
}
```

```
public static SearchResult[] searchAll(String searchText){
	ArrayList<SearchResult> results_a = new ArrayList<SearchResult>();
	for(String network : Constants.NETWORKS){
		if(network.equalsIgnoreCase(Constants.ALL) == false){
			for(SearchResult result : search(searchText,network)){
				results_a.add(result);
			}
		}
	}
	return results_a.toArray(new SearchResult[results_a.size()]);
}
```

Como se puede observar en el código el método *initializeAll* únicamente recorre la constante que representa el conjunto de redes implementadas e invoca al método *initialize* con cada una de ellas. El método *searchAll* funciona de manera parecida, invoca al método *search* para cada red implementada y almacena los resultados obtenidos en una lista.

4.1.3.1.2 Filtrado de resultados

Una vez que se han obtenido los resultados de una búsqueda la aplicación proporciona la opción de filtrarlos para eliminar resultados no relevantes para el usuario. El proceso de filtrado lo lleva a cabo la clase *SearchManager* y se proporciona la posibilidad de filtrar los resultados por tamaño, por número de semillas y por hash.

En el filtrado por tamaño se eliminan aquellos resultados que excedan un determinado tamaño (en bytes) especificado por el usuario.

A continuación se muestra el código encargado de realizar el proceso de filtrado por tamaño:

```
private static SearchResult[] sizeFilter(SearchResult[] results,long size){
	ArrayList<SearchResult> results_a = new ArrayList<SearchResult>();
	for(SearchResult result : results){
		if((long)(result.getSize()) <= (long)(size)){
			results_a.add(result);
		}
	}
	return results_a.toArray(new SearchResult[results_a.size()]);
}
```

Como se observa en el código el proceso es sencillo, primero se crea una lista a la que se añadirán los resultados filtrados, después se va recorriendo la lista de resultados de búsqueda comprobando para cada uno de los cuales si su tamaño es menor o igual al especificado por parámetro y si es así se añade a la lista de resultados filtrados.

En el proceso de filtrado por número de semillas se eliminan aquellos resultados que no tengan un determinado número de semillas especificado por el usuario.

A continuación se muestra el código encargado de realizar el proceso de filtrado por número de seeds:

```
private static SearchResult[] seedsFilter(SearchResult[] results,int nseeds){
	ArrayList<SearchResult> results_a = new ArrayList<SearchResult>();
	for(SearchResult result : results){
		if(result.getnSeeds() >= nseeds){
			results_a.add(result);
		}
	}
	return results_a.toArray(new SearchResult[results_a.size()]);
}
```

Como se puede observar el código es muy parecido al del proceso de filtrado por tamaño, únicamente cambia en que ahora la comprobación verifica si el número de semillas para cada resultado es mayor o igual que uno especificado por parámetro.

En el proceso de filtrado por hash se eliminan aquellos resultados cuyo hash no coincida con uno especificado por el usuario.

A continuación se muestra el código encargado de realizar el proceso de filtrado por hash:

```
private static SearchResult[] hashFilter(SearchResult[] results,String hash){
	ArrayList<SearchResult> results_a = new ArrayList<SearchResult>();
	for(SearchResult result : results){
		if(result.getHash() == hash){
			results_a.add(result);
		}
	}
	return results_a.toArray(new SearchResult[results_a.size()]);
}
```

Como se puede observar el código es muy parecido a los dos anteriores excepto que ahora la comprobación verifica si la función hash para cada resultado es igual que una especificada por parámetro.

Finalmente la aplicación permite un proceso de filtrado múltiple, es decir es posible por ejemplo filtrar los resultados por tamaño y una vez filtrados volver a filtrarlos por número de sedes.

El código encargado de realizar este proceso es el siguiente:

```
private static SearchResult[] filter_results;

public static SearchResult[] filter(HashMap<String,Object> data,SearchResult[] results){
	if(results == null || data.size() == 0){
		return results;
	}
	if(data.containsKey(Constants.SIZE)){
		long size = (Long)data.get(Constants.SIZE);
		if(filter_results == null){
			filter_results = sizeFilter(results,size);
		}
		else{
			filter_results = sizeFilter(filter_results,size);
		}
		data.remove(Constants.SIZE);
		filter(data,results);
	}
	else if(data.containsKey(Constants.NSEEDS)){
		int nseeds = (Integer)data.get(Constants.NSEEDS);
		if(filter_results == null){
			filter_results = seedsFilter(results,nseeds);
		}
```

```
            else{
                filter_results = seedsFilter(filter_results,nseeds);
            }
            data.remove(Constants.NSEEDS);
            filter(data,results);
        }
        else if(data.containsKey(Constants.HASH_FILTER)){
            String hash = String.valueOf(data.get(Constants.HASH_FILTER));
            if(filter_results == null){
                filter_results = hashFilter(results,hash);
            }
            else{
                filter_results = hashFilter(filter_results,hash);
            }
            data.remove(Constants.HASH_FILTER);
            filter(data,results);
        }
        return filter_results;
    }
```

Para el proceso de filtrado múltiple se utiliza un objeto *HashMap* el cual contiene todos los datos que introdujo el usuario para filtrar, dentro del método se va comprobando qué tipos de filtrado solicitó el usuario y se van llamando a los métodos encargados de realizarlos, una vez que se ha terminado de realizar un filtrado se eliminan los datos del objeto *HashMap* y se vuelve a llamar al método *filter* para realizar el siguiente tipo de filtrado.

4.1.3.1.3 Ordenación de resultados

Una vez que se ha realizado el proceso de filtrado la aplicación proporciona la opción de ordenar los resultados de búsqueda de acuerdo a un cierto criterio y modo de ordenación. Los criterios por los cuales se permite la ordenación de los resultados son el nombre, el tamaño, el número de peers, el número de seeds y la fecha de publicación. Los modos de ordenación son ordenación ascendente y ordenación descendente. El proceso de ordenación de los resultados se lleva a cabo en la clase *SortUtils* y el algoritmo utilizado es el *quicksort*.

A continuación se muestran fragmentos del código encargado de realizar el proceso de ordenación, no se muestra la totalidad del código porque no se considera relevante:

```
public static void quicksort(SearchResult[] results,String comparer, String mode) {
	if(results == null || results.length == 0){
		return;
	}
	if(comparer.equalsIgnoreCase(Constants.NSEEDS)){
		if(mode.equalsIgnoreCase(Constants.ASCENDENT)){
			Sort.quicksort(results,  new Compare(){
				@Override
				public int doCompare(Object arg0, Object arg1) {
					// TODO Auto-generated method stub
					SearchResult a = (SearchResult)arg0;
					SearchResult b = (SearchResult) arg1;
					if(a.getnSeeds() > b.getnSeeds()){
						return 1;
					}
					else if(a.getnSeeds() < b.getnSeeds()){
						return -1;
					}
					else return 0;
				}
			});
		}
		else if(mode.equalsIgnoreCase(Constants.DESCENDENT)){
			Sort.quicksort(results,  new Compare(){
				@Override
				public int doCompare(Object arg0, Object arg1) {
					// TODO Auto-generated method stub
					SearchResult a = (SearchResult)arg0;
					SearchResult b = (SearchResult) arg1;
					if(a.getnSeeds() > b.getnSeeds()){
						return -1;
					}
					else if(a.getnSeeds() < b.getnSeeds()){
						return 1;
					}
					else return 0;
				}
			});
		}
	}
	else if(comparer.equalsIgnoreCase(Constants.NAME)){
		if(mode.equalsIgnoreCase(Constants.ASCENDENT)){
			Sort.quicksort(results, new Compare(){
				@Override
				public int doCompare(Object arg0, Object arg1) {
					// TODO Auto-generated method stub
					SearchResult a = (SearchResult)arg0;
					SearchResult b = (SearchResult) arg1;
```

```
                        return -1 *
                            (a.getName().toLowerCase().compareTo(b.getName().toLowerC
                            ase()));
                    }
                });
            }
            else if(mode.equalsIgnoreCase(Constants.DESCENDENT)){
                Sort.quicksort(results, new Compare(){
                    @Override
                    public int doCompare(Object arg0, Object arg1) {
                        // TODO Auto-generated method stub
                        SearchResult a = (SearchResult)arg0;
                        SearchResult b = (SearchResult) arg1;
                        return
                            a.getName().toLowerCase().compareTo(b.getName().toLowerC
                            ase());
                    }
                });
            }
        }
        …
}
```

Como se muestra en el código para el proceso de ordenación se utiliza el método *quicksort* proporcionado por la jdk de Java. Según el criterio y el modo de ordenación especificado por parámetro se utiliza un comparador u otro. El objeto comparador es muy sencillo, simplemente implementa un método denominado *doCompare* que recibe dos argumentos y devuelve un número, si el primer argumento es mayor que el segundo según algún criterio específico se retornará 1, si el segundo es mayor que el primero se retornará -1 y si son iguales se retornará 0. En el caso de la ordenación utilizando el nombre como criterio se utiliza el método *compareTo* de la clase *String* el cual compara dos cadenas de texto y retorna 1,-1 o 0 según el mismo criterio explicado anteriormente.

4.1.3.1.4 Elaboración del informe

Una vez que se ha realizado el proceso de ordenación de los resultados la aplicación elabora un informe incluyendo información sobre éstos. Los tipos de informes que soporta la aplicación son archivos de texto plano y archivos PDF. Para la construcción de archivos PDF la aplicación se apoyará en la librería *itextpdf*, especializada en la elaboración de dichos archivos. Las clases encargadas de realizar el proceso de elaboración de informes son las clases *TxtInform*, *PDFInform*, *TorrentTxtInform*, *TorrentPDFInform* y la interfaz *InformGenerator*. Las dos primeras son clases abstractas que representan informes de tipo txt y PDF respectivamente,

primera extiende de la clase *PrintWriter* y la segunda extiende de la clase *Document*. Las clases *TorrentTxtInfom* y *TorrentPDFInform* a su vez heredan de las dos anteriores e implementan el método *generateInform*, declarado en la interfaz *InformGenerator*, y en el cual se desarrolla el proceso de elaboración del informe de búsqueda, especializado para la red BitTorrent. Para elaborar un informe simplemente se instancia la clase *TorrentTxtInform* o la clase *TorrentPDFInform* proporcionándoles la ruta hacia el archivo donde se escribirá el informe como argumento pasado al constructor. Una vez que se ha instanciado la clase se invoca al método *generateSearchInform*.

A continuación se muestra el código encargado de elaborar un informe de búsqueda de tipo txt:

```
public void generateSearchInform(HashMap<String, Object> data) {
	// TODO Auto-generated method stub
	SearchResult[] results = (SearchResult[])data.get(Constants.RESULTS);
	if(results == null){
		results = new SearchResult[0];
	}
	String searchQuery = String.valueOf(data.get(Constants.SEARCHQUERY));
	this.println("Informe de búsqueda");
	this.println("Cadena de búsqueda: " + searchQuery + "\n");
	this.println("Se encontraron " + results.length + " resultados");
	this.println("Lista de resultados:");
	for(int i = 0;i < results.length;i++){
		this.println((i+1) + "--> " + results[i].toString());
	}
	this.println("Fin del informe");
	this.flush();
}
```

Para el proceso de elaborar un informe se utiliza un objeto *HashMap* el cual contiene datos relativos al proceso de búsqueda como son la cadena de búsqueda utilizada y los resultados devueltos. Como se observa en el código mediante el método *println* se van imprimiendo los datos en el informe.

A continuación se muestra el código encargado de elaborar un informe de búsqueda de tipo PDF:

```
public void generateSearchInform(HashMap<String, Object> data) {
	// TODO Auto-generated method stub
	SearchResult[] results = (SearchResult[])data.get(Constants.RESULTS);
	if(results == null){
		results = new SearchResult[0];
	}
	String searchQuery = String.valueOf(data.get(Constants.SEARCHQUERY));
	Paragraph paragraph = new Paragraph();
	Font title = new Font();
	title.setSize(20);
	title.setStyle(0);
	Font title2 = new Font();
	title2.setSize(16);
	title2.setStyle(0);
	paragraph.setFont(title);
	paragraph.add("Informe de búsqueda\n");
	paragraph.add("\n");
	paragraph.setFont(new Font());
	paragraph.add("Cadena de búsqueda: " + searchQuery + "\n");
	paragraph.add("\n");
	paragraph.add("Se encontraron " + results.length + " resultados\n");
	paragraph.add("\n");
	paragraph.setFont(title2);
	paragraph.add("Lista de resultados\n");
	paragraph.add("\n");
	paragraph.setFont(new Font());
	for(int i = 0;i < results.length;i++){
		paragraph.add((i+1) + "--> " + results[i].toString()+".\n");
	}
	paragraph.setFont(title);
	paragraph.add("\n");
	paragraph.add("Fin del informe");
	try {
		this.add(paragraph);
	}
	catch (DocumentException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}
}
```

El proceso de elaboración de un informe de tipo PDF es muy similar al de elaboración de un informe de tipo txt salvo que para elaborar informes PDF se utilizan objetos *Paragraph*, los cuales representan párrafos en el documento, y objetos *Font* para establecer las propiedades de la fuente de lo que se va a escribir.

4.1.3.2 Plantillas de metabúsqueda

Como ya se ha dicho anteriormente la operación de búsqueda utiliza unos archivos especiales propios de Vuze denominados plantillas. Estos archivos se valen de expresiones regulares que se utilizan para parsear distintas webs en búsqueda de torrents. Estas plantillas se pueden encontrar por internet ya que el propio programa Vuze tiene una opción para elaborarlas y los usuarios las comparten. La aplicación desarrollada proporciona operaciones para importar plantillas, borrarlas y mostrar las plantillas importadas. Los argumentos necesarios para ejecutar las operaciones anteriores se muestran en el **Anexo A: Manual de la aplicación**. No se mostrará el código encargado de realizar estas operaciones ya que únicamente realiza llamadas a métodos propios de Vuze y no se considera relevante.

4.1.4 Operación de descarga

La función de descarga al igual que la función de búsqueda se apoya en funciones del programa Vuze. La operación se divide en dos, por un lado la aplicación desarrollada permite la descarga de archivos torrent a partir de su enlace y por otro lado permite la descarga de archivos a partir del archivo torrent previamente descargado. A continuación se procede a detallar ambas funcionalidades.

4.1.4.1 Descarga de archivos torrent

La aplicación permite la descarga de archivos torrent, los cuales son necesarios para la descarga de cualquier archivo. Estos archivos torrent son localizados mediante URLs, adicionalmente también se permite la descarga de archivos torrent mediante enlaces magnet. En este apartado se estudian los aspectos relativos a la implementación de esta operación.

4.1.4.1.1 Modo de funcionamiento

El conjunto de clases que intervienen en el proceso es el siguiente:

Clases: Main, CommandLineParser, Executer, TorrentApp, TorrentDownloadManager, TorrentDownload.

El funcionamiento de la operación a grandes rasgos es el siguiente:

1. Inicialmente, el usuario introduce los argumentos necesarios para ejecutar la operación de descarga de un archivo torrent, estos se parsean y validan, tal y como se vio en el apartado de parseo de los comandos de usuario, y llama a la clase *Executer* para que inicie dicha operación.
2. Como primer paso a realizar en la operación de descarga se inicializa la clase controladora del proceso de descarga sobre la red BitTorrent, la clase *TorrentDownloadManager*. Para la ejecución de esta operación se prescinde de la clase controladora *DownloadManager* ya que el proceso de descarga de archivos torrent únicamente es válido para la red BitTorrent.
3. Se invoca al método *addDownloadTorrent* de la clase *TorrentDownloadManager* para cada URL o enlace magnet especificado por el usuario.
4. Una vez que se ha añadido la configuración de todos los archivos torrent que se desea descargar se invoca al método *startAllDownloadTorrents* de la clase *TorrentDownloadManager* para iniciar la descarga de todos ellos. La descarga de cada archivo torrent es llevada a cabo por una instancia de la clase *TorrentDownload*.
5. Finalmente se instancia y se inicia la clase *ScannerThread* la cual se utiliza para leer de la consola posibles instrucciones especificadas por el usuario.

A continuación se muestra un diagrama de actividad de la operación para comprender mejor el proceso.

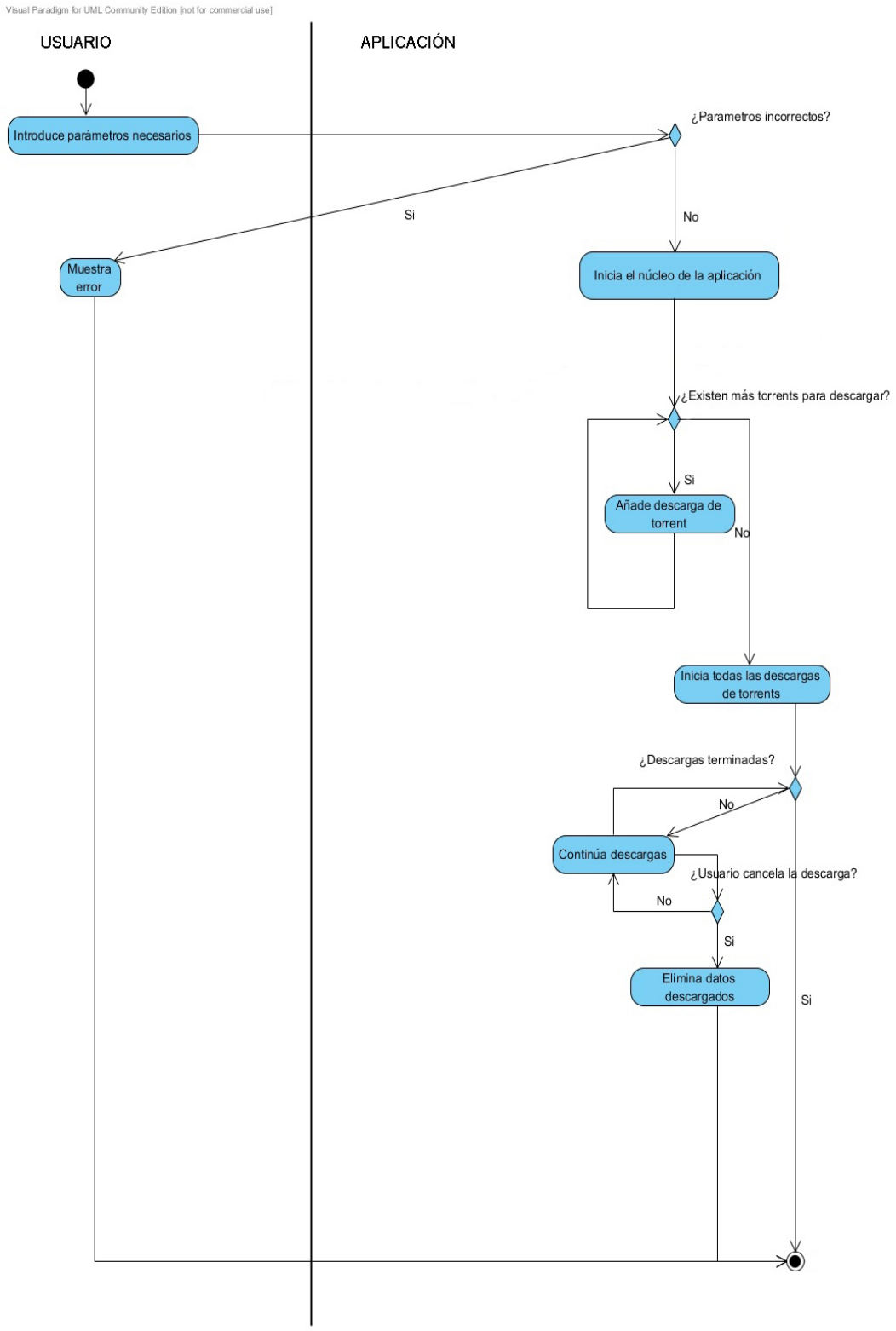


Ilustración 26: Diagrama de actividad de la operación de descarga de un archivo torrent

Finalmente se van a explicar algunas partes del proceso más detalladamente.

4.1.4.1.1.1 Proceso de descarga del archivo torrent

El proceso de descarga de archivos torrent se controla mediante la clase *TorrentDownloadManager*. La aplicación permite la descarga de múltiples archivos torrent en una ejecución. Tal y como se dijo anteriormente primero se añade la configuración del conjunto de archivos torrent que se desean descargar, incluyendo para cada uno de ellos la URL o enlace magnet que los localiza y opcionalmente una URL de referencia si es necesaria. Una vez que se han configurado todas las descargas de archivos torrent a realizar se procede a iniciar el proceso de descarga para todas ellas. Cada descarga de un archivo torrent es llevada a cabo por la clase *TorrentDownload*. Esta clase extiende de la clase *Thread* de java y realiza el proceso de descarga de forma paralela al resto de la ejecución de la aplicación. Además durante la descarga de un archivo torrent se le informa al usuario sobre el estado del proceso. No se muestra el código del proceso de descarga de un archivo torrent puesto que únicamente contiene llamadas a funciones del programa Vuze y no se considera relevante.

4.1.4.1.1.2 Lectura de instrucciones del usuario

Durante el proceso de descarga de un archivo torrent la aplicación le permite al usuario la cancelación de dicho proceso escribiendo la orden *cancel* en la consola. Si el usuario especifica esta orden la aplicación para el proceso de descarga y elimina los datos que ya se hubieran descargado. Además la aplicación también le permite al usuario emplear la orden *exit* mediante la cual finaliza la ejecución de la aplicación marcando como pendientes las descargas que se están ejecutando de manera que puedan ser continuadas cuando se inicie de nuevo la aplicación. La clase *ScannerThread*, la cual hereda de la clase *Thread* de java, se encarga de, de forma paralela al proceso de descarga, leer las órdenes especificadas por el usuario y, en caso de que se trate de alguna de las permitidas, actuar en consecuencia.

4.1.4.2 Descarga de archivos

La aplicación permite la descarga de archivos, el objetivo de esta operación es comprobar si dichos archivos contienen información oculta y en ese caso proceder de alguna forma. En este apartado se estudian los aspectos relativos a la implementación de esta operación.

4.1.4.2.1 Modo de funcionamiento

El conjunto de clases que intervienen en el proceso es el siguiente:

Clases: Main, CommandLineParser, Executer, TorrentApp, DownloadManager, TorrentDownloadManager, FileDownload.

Además intervienen en el proceso las clases relacionadas con los detectores y las acciones pero éstas se explicarán en una sección posterior.

El funcionamiento de la operación a grandes rasgos es el siguiente:

1. Inicialmente, el usuario introduce los argumentos necesarios para ejecutar la operación de descarga de un archivo, estos se parsean y validan, tal y como se vio en el apartado de parseo de los comandos de usuario, y llama a la clase *Executer* para que inicie dicha operación.
2. Como primer paso a realizar en la operación de descarga se inicializa la clase controladora de dicho proceso, la clase *DownloadManager*, ésta a su vez inicializará la clase controladora del proceso de descarga sobre una red concreta, en el caso de la red BitTorrent se trata de la clase *TorrentDownloadManager*.
3. Se invoca al método *addDownloadFile* de la clase *DownloadManager*, para cada ruta hacia un archivo torrent especificado por el usuario. La clase *DownloadManager* a su vez llama al correspondiente método *addDownloadFile* de la clase *TorrentDownloadManager*.
4. Una vez que se ha añadido la configuración de todos los archivos torrent que se desea descargar se invoca al método *startAllDownloadFiles* de la clase *DownloadManager* para iniciar la descarga de todos ellos. La clase *DownloadManager* a su vez llama al correspondiente método *startAllDownloadFiles* de la clase *TorrentDownloadManager*. La descarga de cada archivo es llevada a cabo por una instancia de la clase *FileDownload*.
5. A continuación se instancia y se inicia la clase *ScannerThread* la cual se utiliza para leer de la consola posibles instrucciones especificadas por el usuario. El proceso de leer las instrucciones de usuario es el mismo que el realizado en la descarga de un archivo torrent.
6. Es durante el proceso de descarga del archivo cuando se ejecutan los detectores y las acciones. Los aspectos de implementación de ambos se detallarán en una sección posterior.

A continuación se muestra un diagrama de actividad de la operación para comprender mejor el proceso.

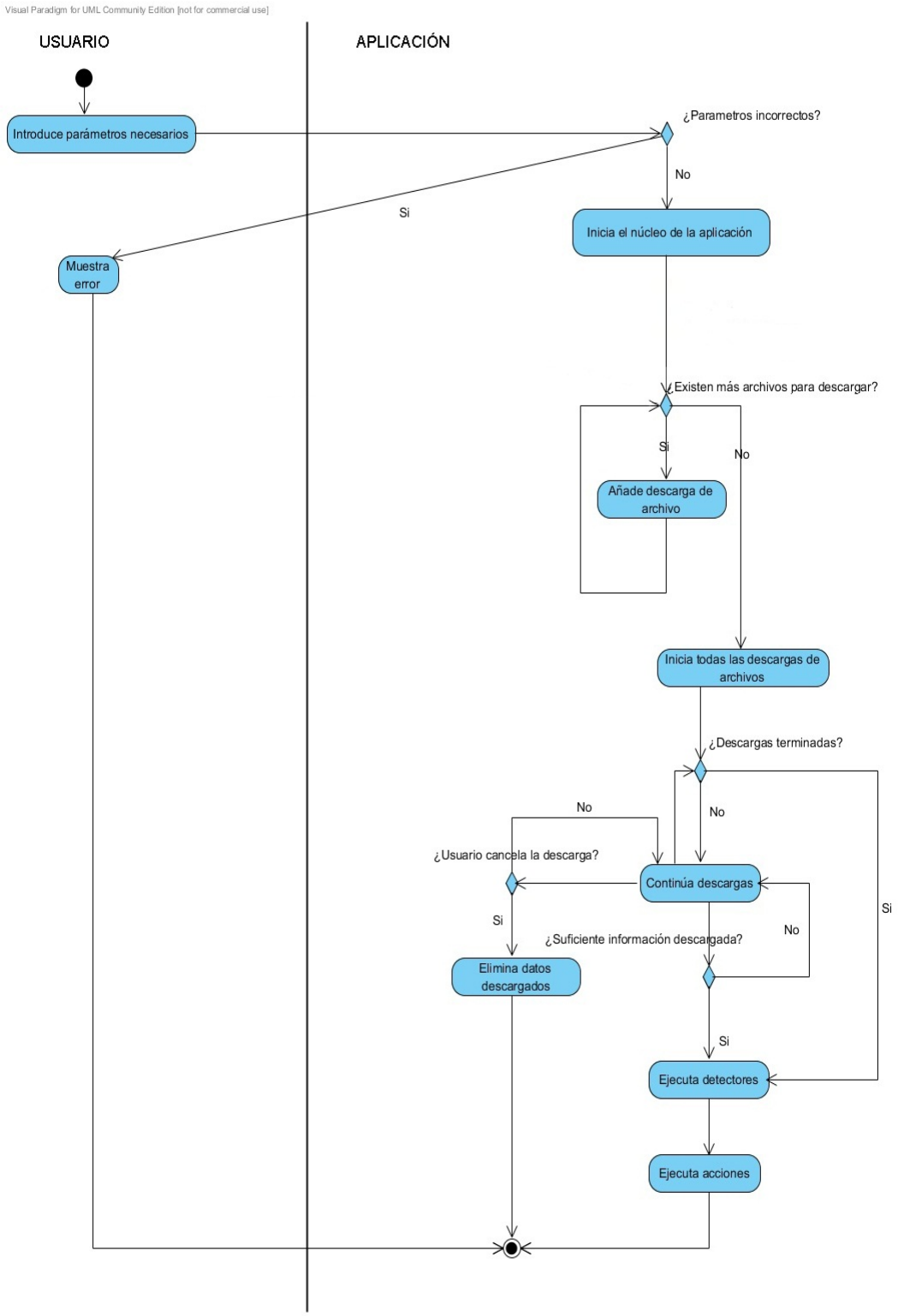


Ilustración 27: Diagrama de actividad de la operación de descarga de un archivo

Finalmente se van a explicar algunas partes del proceso más detalladamente.

4.1.4.2.1.1 Proceso de descarga del archivo

El proceso de descarga de archivos se controla mediante las clases *DownloadManager* y *TorrentDownloadManager*. El funcionamiento de estas clases es equivalente al de las clases *SearchManager* y *TorrentSearchManager*, explicado en el apartado relativo a la implementación de la operación de búsqueda. La aplicación contiene una clase controladora de descargas para cada red soportada, inicialmente la única red implementada es la red BitTorrent y la clase controladora de descargas para esta red es la clase *TorrentDownloadManager*. Esta clase se encarga de, mediante apoyo del programa Vuze, realizar el proceso de descarga. Por otra parte la clase *DownloadManager* se encarga de, según la red a utilizar especificada por el usuario, inicializar una clase controladora u otra e invocar al método de descarga de dicha clase. No se muestra el código de los métodos de inicialización y descarga de la clase *DownloadManager* ya que funcionan de la misma forma que los de la clase *SearchManager* ya mostrados.

La aplicación permite la descarga de múltiples archivos en una ejecución. Tal y como se dijo anteriormente primero se añade la configuración del conjunto de archivos que se desean descargar, incluyendo para cada uno de ellos la ruta al archivo torrente necesario para la descarga. Una vez que se han configurado todas las descargas de archivos a realizar se procede a iniciar el proceso de descarga para todas ellas. Cada descarga de un archivo es llevada a cabo por la clase *FileDownload*. Esta clase extiende de la clase *Thread* de java y realiza el proceso de descarga de forma paralela al resto de la ejecución de la aplicación. Además durante la descarga de un archivo se le informa al usuario sobre el estado del proceso, incluyo datos sobre el porcentaje del archivo descargado, el número de semillas y pares que lo están compartiendo, el estado del Tracker, etc. No se muestra el código del proceso de descarga de un archivo puesto que únicamente contiene llamadas a funciones del programa Vuze y no se considera relevante.

4.1.5 Operación automatizada

La operación automatizada es una combinación de las operaciones de búsqueda y descarga. La aplicación busca una serie de resultados a partir de un término de búsqueda especificado por el usuario. A continuación descargará un determinado número de ellos de acuerdo a las instrucciones especificadas por el usuario. Al igual que en la operación de descarga durante el proceso de descarga de los archivos la aplicación ejecuta una serie de detectores configurados por el usuario y en caso de que éstos den un resultado positivo iniciar una serie de acciones también de acuerdo a la configuración especificada por el usuario.

4.1.5.1 Modo de funcionamiento

El conjunto de clases que intervienen en el proceso es el siguiente:

Clases: Main, CommandLineParser, Executer, TorrentApp, SearchManager, TorrentSearchManager, SearchResult, TorrentSearchResult, DownloadManager, TorrentDownloadManager, TorrentDownload, FileDownload.

Además intervienen en el proceso las clases relacionadas con los detectores y las acciones pero éstas se explicarán en una sección posterior.

El funcionamiento de la operación a grandes rasgos es el siguiente:

1. Inicialmente, el usuario introduce los argumentos necesarios para ejecutar la operación automatizada, estos se parsean y validan, tal y como se vio en el apartado de parseo de los comandos de usuario, y llama a la clase *Executer* para que inicie dicha operación.
2. Como primer paso a realizar en la operación automatizada se realiza una operación de búsqueda utilizando el término de búsqueda especificado por el usuario (ver apartado **4.1.3 Operación de búsqueda**). En la operación automatizada se omite el paso de elaboración del informe.
3. Se instancia y se inicia la clase *ScannerThread* la cual se utiliza para leer de la consola posibles instrucciones especificadas por el usuario.
4. Una vez obtenidos los resultados de búsqueda se procede a descargar un determinado número de ellos, especificado por el usuario (ver apartado **4.1.4.1 Descarga de archivos torrent**).
5. Cuando ya ha terminado el proceso de descarga de los archivos torrent y utilizando la información proporcionada por ellos se procede a la descarga de los archivos (ver apartado **4.1.4.2 Descarga de archivos**).
6. Al igual que ocurre en la operación de descarga de un archivo es durante el proceso de descarga del cual cuando se ejecutan los detectores y las acciones.

A continuación se muestra un diagrama de actividad de la operación para comprender mejor el proceso.

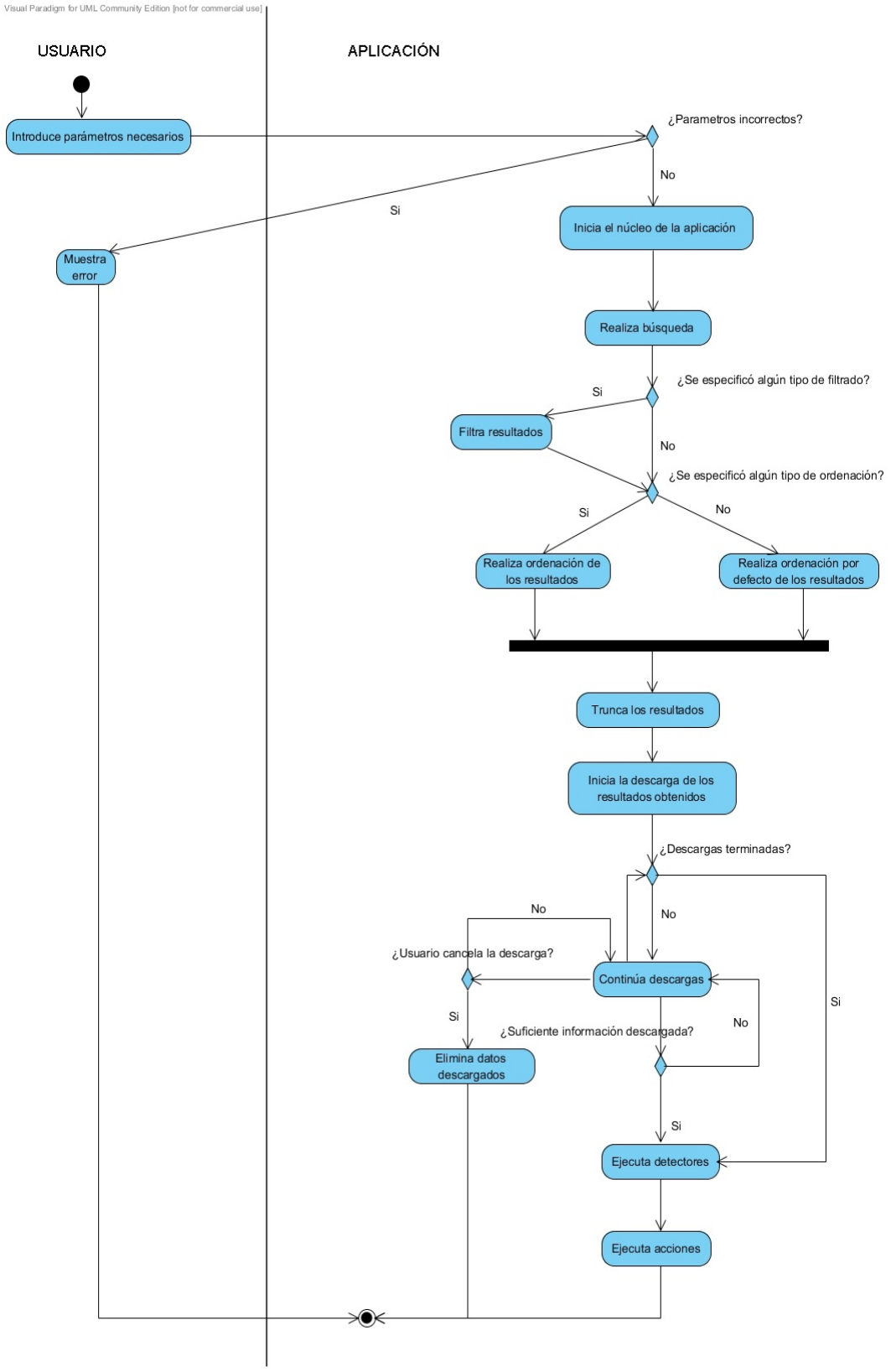


Ilustración 28: Diagrama de actividad de la operación automatizada

Las distintas partes del proceso no necesitan explicación ya que se trataron con anterioridad en el análisis de las operaciones anteriores.

4.1.6 Detectores

Durante la descarga del mismo se inicia el proceso de detección de contenido oculto. Todos los detectores se ejecutan de forma paralela a la descarga del archivo y se ejecutan de acuerdo a la configuración especificada por el usuario, tal y como se verá en el **Anexo A: Manual de la aplicación**. La configuración especificada por el usuario se valida de acuerdo a la siguiente expresión regular.

```
((\\[type:vecna\\]))+
```

Como se puede observar la expresión regular es extremadamente sencilla y realmente no tiene sentido su utilización para validar la configuración especificada por el usuario. No obstante se ha decidido validar dicha configuración de esta forma para facilitar la inclusión de nuevos detectores y su configuración.

Una vez que se valida que el usuario ha introducido una configuración válida se analiza dicha configuración y se parsea a un formato comprendido por la aplicación.

La clase encargada de controlar el lanzamiento de los detectores es la clase *DetectorManager*. Esta clase se encarga de, controlar el proceso de lanzamiento de los detectores en base a la configuración del usuario y al número de piezas (partes de un archivo) descargadas. No se muestra el código que ejecuta dicha operación ya que únicamente se trata de estructuras de control y llamadas a funciones del programa Vuze y no se considera relevante.

A continuación se describe el único detector que implementa la aplicación, el detector Vecna.

4.1.6.1 Detector Vecna

El detector Vecna especializa la detección de contenido oculto en imágenes mediante la técnica del cálculo de la entropía para el programa especializado en esteganografía Vecna. Para la detectar si un archivo es una imagen se vale de las clases *BMPDetector, JPGDetector, GIFDetector* y *PNGDetector.*

La entropía, también llamada entropía de Shannon mide la incertidumbre promedio en una fuente de información. En esteganografía se utiliza como una medida de la aleatoriedad del archivo, si al realizar el cálculo de la entropía de dicho archivo se obtiene un valor superior a un cierto umbral (calculado en base a resultados anteriores) es posible que dicho archivo haya sido alterado de alguna forma.

El cálculo de la entropía se realiza mediante la siguiente fórmula:

$$H(X) = -\sum p(x_i) * \log_2 p(x_i)$$ [19]

Siendo $p(x_i)$ la probabilidad de aparición de los estados de un determinado fenómeno. En el caso que nos ocupa $p(x_i)$ será la probabilidad de aparición de cada uno de los componentes RGB de un color en el conjunto de píxeles de una imagen. Por ejemplo, si en una cierta imagen aparecen los siguientes colores:

(200 150 225) (200 230 240)

El cálculo de los diferentes valores de $p(x_i)$ sería el siguiente:

$p(x_{200}) = \frac{2}{6} = 0.33,\ p(x_{150}) = \frac{1}{6} = 0.16$, y así sucesivamente.

La clase de implementar el detector Vecna es la clase *VecnaDetector*, a continuación se muestran los aspectos más relevantes de su código.

```
public static int BITS = 8;
public static double THRESHOLD = 7.825;
```

Los atributos de la clase mostrados anteriormente son esenciales para la ejecución del detector. El atributo *BITS* representa el número de bits menos significativos a utilizar para el cálculo de la entropía. El atributo *THRESHOLD* por otro lado representa el valor con el que se comparará la entropía calculada para determinar si el archivo contiene información oculta o no.

```
public static boolean detect(byte[] downloadedData) throws IOException{
      BufferedImage imagenAux = ImageIO.read(new
          ByteArrayInputStream(downloadedData));
      double measuredEntropy = entropy(imagenAux,VecnaDetector.BITS);
      boolean detected;
      if (imagenAux != null && measuredEntropy > VecnaDetector.THRESHOLD){
            detected = true;
      }
      else{
            detected = false;
      }
      return detected;
}
```

Como se puede observar en el código primero se crea un objeto *BufferedImage* a partir de un array de bytes, dicho objeto contendrá todos los datos relativos a la imagen como su ancho, su alto, el conjunto de sus píxeles, etc. Posteriormente se

[19] Obtenida de: (Shannon entropy calculator, 2012)

calcula la entropía de los píxeles de la imagen llamando al método *entropy* y utilizando el número de bits especificado por el argumento *BITS*. Finalmente se compara el valor de la entropía calculada con el valor del atributo *THRESHOLD,* si es superior a dicho atributo se considera que la imagen contiene contenido oculto, en caso contrario se considera que la imagen no ha sido alterada y por lo tanto no contiene contenido oculto.

```
public static double entropy(BufferedImage imagen, int bits){
    int width = imagen.getWidth();
    int height = imagen.getHeight();
    int tamanyo = width*height;
    int total = tamanyo*3;
    int tam_entropia = 1<<bits;
    int []probabilidades = new int[tam_entropia];
    for(int i=0;i<width;i++){
        for(int j=0;j<height;j++){
            int pixel = imagen.getRGB(i, j);
            for(int k=0; k<3;k++){
                int component = (pixel >> (8*k)) & (tam_entropia-1);
                probabilidades[component]++;
            }
        }
    }
    //calculo de la entropía:
    double entropy=0.0;
    for(int i=0; i< probabilidades.length; i++){
        entropy = entropy - ((double)probabilidades[i]/total) *
            (Math.log(((double)probabilidades[i]/total))/Math.log(2));
    }
    return entropy;
}
```

El código anterior representa el cálculo de la entropía. La primera operación que se realiza en el código es el cálculo de las probabilidades de aparición de los distintos componentes RGB de los colores que forman la imagen. Para realizar dicho cálculo se declara un array de enteros de 2^{BITS} posiciones. A continuación se recorre completamente la imagen obteniendo una representación numérica de cada uno de los píxeles que forman la imagen. A partir de dicha representación numérica se calculan los diferentes componentes RGB que forman el píxel y se actualiza su probabilidad de aparición

Una vez calculadas las probabilidades de aparición se procede con el cálculo de la entropía. Dicho cálculo simplemente consiste en la implementación de la fórmula presentada anteriormente, el bucle *for* se utiliza para realizar la operación de incremento y la instrucción *probabilidades[i]/total) * (Math.log(probabilidades[i]/total)/Math.log(2))* equivale a la operación $p(x_i) * \log_2 p(x_i)$.

4.1.7 Acciones

Una vez que se ha detectado que un archivo descargado (como resultado de una operación de descarga o una operación automatizada) es una imagen y además contiene información oculta se inicia el proceso de ejecución de las acciones. Todas las acciones se ejecutan de forma paralela a la descarga del archivo y se ejecutan de acuerdo a la configuración especificada por el usuario, tal y como se verá en el **Anexo A: Manual de la aplicación**. La configuración especificada por el usuario se valida de acuerdo a la siguiente expresión regular.

```
((\\[type:dictionary,dictionaryPath:[^,\\]]+(,outputFile:[^,\\]]+)?\\])|(\\[type:ip\\])|(\\[type:bruteforce,((charsets:(minLetters|mayLetters|numbers|all)(\\|minLetters|\\|mayLetters|\\|numbers|\\|all)*,initLength:\\d+)|(ownCharset:[^,\\]]+))(,initPassphrase:[^,\\]]+)?(,outputFile:[^,\\]]+)?\\]))+
```

Una vez que se valida que el usuario ha introducido una configuración válida se analiza dicha configuración y se parsea a un formato comprendido por la aplicación. Cada parte sombreada se corresponde con una de las tres acciones que implementa la aplicación.

La clase encargada de controlar el lanzamiento de las acciones es la clase *ActionManager*. Esta clase se encarga de, a partir de la configuración de las acciones especificada por el usuario, lanzar unas clases u otras. No se muestra el código que ejecuta dicha operación ya que únicamente consiste en un conjunto de estructuras de control que se utilizan para decidir a qué clase llamar y por lo tanto no se considera relevante.

A continuación se describen las distintas acciones que implementa la aplicación.

4.1.7.1 Ataque por diccionario

La correcta configuración de esta acción por parte del usuario se especificará en el **Anexo A: Manual de la aplicación**.

La acción de ataque de diccionario consiste en recorrer un fichero de texto plano especial denominado diccionario con el fin de ir probando claves (a partir de ahora se denominará intento a la clave probada) con el fin de obtener aquella con la que se ha ocultado información en un archivo descargado. El diccionario no es más que un fichero de texto plano con un conjunto de palabras, normalmente propias de alguna lengua en concreto aunque no es necesario. Estos ataques son más eficientes que los ataque de fuerza bruta puesto que muchos usuarios utilizan palabras de su propia lengua como contraseña.

Este ataque se ha instanciado para el programa Vecna, especializado en esteganografía. Para cada intento se llama a la función *decode* del programa, cuyo cometido es el de extraer la información ocultada en la imagen descargada, utilizando dicho intento como posible clave para la extracción. La clase encargada de implementar dicha acción es la clase *VecnaDictionaryAttackThread*.

A continuación se muestra parte del código que implementa la acción de ataque por diccionario.

```
public void run(){
	DictionaryAttack dic = null;
	String currentKey = null;
	boolean finished = false;
	int i = 0;
	int numWorkers = Runtime.getRuntime().availableProcessors()*2;
	ExecutorService tpes = Executors.newFixedThreadPool(numWorkers);
	AttackCallable workers[] = new AttackCallable[numWorkers];
	Future<?> futures[] = new Future[numWorkers];
	TorrentTxtInform inform = null;
	try {
		dic = new DictionaryAttack(dictionaryPath);
		inform = new TorrentTxtInform(Constants.INFORMS_FOLDER +
		    Constants.FILE_SEPARATOR + "Actions" + Constants.FILE_SEPARATOR +
		    downloadName + "DictionaryActionInform.txt", true);
		inform.println("INICIO DEL INFORME, FECHA: " + new Date());
		while (finished == false) {
			currentKey = dic.getCurrentAttemp();
			if(currentKey == null){
				finished = true;
				tpes.shutdown();
			}
			else{
				workers[i%numWorkers] = new AttackCallable(currentKey){
					@Override
					public AttackReturn call() throws Exception {
						// TODO Auto-generated method stub
						boolean retorno = VecnaUtils.decodeImage(inputImage,
						outputFile, getCurrentAttemp());
						return new AttackReturn(retorno,getCurrentAttemp());
					}
				};
				futures[i%numWorkers]= (Future<?>)
				    tpes.submit(workers[i%numWorkers]);
				dic.increment();
				i++;
				if((i % numWorkers) == 0){
					AttackReturn retorno;
					for (int j = 0; j < numWorkers; j++) {
						try {
							retorno = (AttackReturn) futures[j].get();
```

```
                                    if(retorno.getReturn()){
                                        if(verbose){
                                            System.out.println("Ataque finalizado, clave
                                                utilizada: " + retorno.getKey());
                                        }
                                        inform.println("File: " +
                                            detectorResult.getFile().getName() + ", Type: " +
                                            detectorResult.getType() + ", Ataque con éxito,
                                            información extraida en " + new
                                            File(outputFile).getAbsolutePath() + ", clave
                                            utilizada: " + retorno.getKey());
                                        finished = true;
                                        tpes.shutdown();finished = true;
                                        tpes.shutdown();
                                    }
                                } catch (Exception e) {}
                            }
                            retorno = null;
                        }
                    }
                }
            } catch (FileNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (IOException e) {
                // TODO Bloque catch generado automáticamente
                e.printStackTrace();
            }
            finally{
                inform.println("FIN DEL INFORME");
                inform.close();
            }
}
```

A continuación se detallan algunos aspectos del código:

- Será la clase *DictionaryAttack* la encargada de generar las claves utilizando la configuración especificada por el usuario.
- Mediante la clase *ExecutorService*, propia de Java, se crea un pool de hilos. El tamaño del pool se establece como el doble de procesadores de la máquina en la que se está ejecutando la aplicación.
- Los objetos *AttackCallable* se utilizan para ejecutar un ataque de forma paralela tal y como se dijo en el apartado **3.4.5 Acciones.**
- El objeto *Future* se utiliza para recuperar el valor de retorno devuelto por los objetos *AttackCallable*.
- Los valores de retorno se almacenan en la clase *AttackReturn*. Esta clase almacena un booleano que indica si el ataque tuvo éxito o no y la clave utilizada en el ataque.

4.1.7.2 Ataque por fuerza bruta

La correcta configuración de esta acción por parte del usuario se especificará en el **Anexo A: Manual de la aplicación**.

La acción de ataque por fuerza bruta consiste en a partir de un juego de caracteres ir generando claves con el fin de obtener aquella con la que se ha ocultado información en un archivo descargado. Supóngase que se configura una acción de ataque de fuerza bruta para utilizar el juego de caracteres de las letras minúsculas, el conjunto de claves generadas sería el siguiente (dicho proceso únicamente termina cuando se cierra la aplicación o cuando se encuentra la clave correcta):

a
b
c
…
z
aa
ab
…

Al igual que en el ataque por diccionario dicho ataque se ha instanciado para el programa Vecna. El proceso a seguir es exactamente el mismo que en dicha acción utilizando cada clave generada como intento de clave para la extracción. La clase encargada de implementar dicha acción es la clase *VecnaBruteforceAttackThread*.

El código encargado de realizar la acción de ataque de fuerza bruta no se mostrará ya que es equivalente al mostrado en la descripción de la acción de ataque por diccionario salvo que en este caso la generación de claves se hace utilizando la clase *BruteforceAttack*.

4.1.7.3 Obtención de direcciones IP

La correcta configuración de esta acción se especificará en el **Anexo A: Manual de la aplicación**.

La acción de obtención de direcciones IP consiste en utilizar funciones del programa Vuze para obtener las direcciones IP de los pares que están compartiendo el archivo descargado contenedor de información oculta. Como resultado de la ejecución de esta acción se obtendrá un fichero de texto plano con todas las direcciones IP. El objetivo de esta acción sería el de, utilizando dichas direcciones IP, informar a dichos pares de que el archivo que están descargando contiene información oculta o el de denunciar el hecho a la policía. No obstante estas acciones finalmente no se implementaron por los motivos expuestos en el **Capitulo 6: Conclusiones y líneas futuras**. La clase encargada de implementar dicha acción es la clase *IPAction.*

El código de esta acción no se mostrara ya que únicamente contiene llamadas a funciones propias de Vuze.

4.1.8 Operación de limpieza

La operación de limpieza consiste en el borrado de todas las descargas que se encuentran en proceso incluyendo los datos parcialmente descargados y otros datos asociados a ellas.

4.1.8.1 Modo de funcionamiento

El conjunto de clases que intervienen en el proceso es el siguiente:

Clases: Main, CommandLineParser, Executer, TorrentApp.

El funcionamiento de la operación a grandes rasgos es el siguiente:

1. Inicialmente, el usuario introduce los argumentos necesarios para ejecutar la operación de limpieza estos se parsean y validan, tal y como se vio en el apartado de parseo de los comandos de usuario, y llama a la clase *Executer* para que inicie dicha operación.
2. La aplicación recupera la lista de descargas activas.
3. Para cada descarga activa se cancela su proceso y se elimina de la lista.
4. A continuación se elimina información de uso interno de la aplicación relacionada con las descargas
5. Finalmente se eliminan los datos parcialmente descargados.

4.1.9 Sistema de configuración

La aplicación cuenta con un sistema de configuración mediante archivos XML y que le permite al usuario la configuración de ciertas propiedades de la misma.

4.1.9.1 Modo de funcionamiento

Para cambiar la configuración de la aplicación basta con editar el archivo properties.xml situado dentro de la carpeta *Settings* de la aplicación. Las distintas propiedades que se pueden editar se mostrarán en el **Anexo A: Manual de la aplicación**.

A continuación se muestra el código encargado de leer y almacenar las propiedades, este código se ejecuta en la clase *Main*:

```
Properties properties = new Properties();
try {
	properties.loadFromXML(new FileInputStream(Constants.SETTINGS_FOLDER +
	    Constants.FILE_SEPARATOR + "properties.xml"));
}
catch (FileNotFoundException e2) {
	properties.put("Informs_folder", System.getProperty("user.dir")+
	    Constants.FILE_SEPARATOR + "Informs");
	properties.put("Torrents_folder", System.getProperty("user.dir") +
	    Constants.FILE_SEPARATOR + "Torrents");
	properties.put("Default_downloads_folder", System.getProperty("user.dir") +
	    Constants.FILE_SEPARATOR + "Downloads");
	try {
		properties.storeToXML(new FileOutputStream(Constants.SETTINGS_FOLDER +
		    Constants.FILE_SEPARATOR + "properties.xml"),
		    Constants.PROPERTIES_COMMENTS);
	}
	catch (FileNotFoundException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}
	catch (IOException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}
}
 catch (IOException e2) {
	// TODO Auto-generated catch block
	e2.printStackTrace();
}
```

Como se observa en el código inicialmente se leen las propiedades del archivo properties.xml, si éste no existe se crean unas propiedades por defecto y se almacenan en dicho archivo properties.xml.

4.1.10 Ejecución mediante archivos XML

La aplicación además del modo de ejecución mediante comandos implementa un modo de ejecución mediante archivos XML. En estos archivos se deben incluir los parámetros necesarios para la ejecución de la operación deseada, además la aplicación utiliza un esquema XML para validar que los archivos XML especificados por los usuarios cumplen un determinado formato.

4.1.10.1 Modo de funcionamiento

Los argumentos necesarios para ejecutar la aplicación en el modo de ejecución mediante archivos XML se mostrarán en el **Anexo A: Manual de la aplicación**. El modo de funcionamiento del modo de ejecución mediante archivos XML es el siguiente:

Una vez que la clase *CommandLineParser* ya ha parseado los comandos especificados por el usuario utiliza la clase *XMLParser* para validar y parsear el archivo XML. Como resultado del parseo del archivo XML se obtiene un objeto *Hashtable* con todos los argumentos necesarios para ejecutar la operación deseada por el usuario. A continuación se valida si dichos argumentos tienen un formato correcto y se procede a la ejecución de la operación deseada.

El formato que deben seguir los archivos XML se mostrará en el **Anexo A: Manual de la aplicación** junto con ejemplos para la correcta ejecución de las distintas operaciones ejecutadas por la aplicación.

4.2 Resultados del plan de pruebas

El resultado del plan de pruebas ha sido satisfactorio ya que se superaron todas y cada una de ellas tal y como se muestra en la siguiente tabla.

Identificador	Resultado
PR-01	Superada
PR-02	Superada
PR-03	Superada
PR-04	Superada
PR-05	Superada
PR-06	Superada
PR-07	Superada

Tabla 45: Resultado de las pruebas

Capitulo 5: Gestión del proyecto

En este capítulo se realizará un estudio de la planificación del proyecto mostrándose diagramas de Gantt tanto de la planificación inicial como de la que finalmente se llevó a cabo. A continuación se mostrarán los medios técnicos empleados en el desarrollo del proyecto, tanto software como hardware, y se realizará un análisis económico del mismo mediante una estimación de costes inicial, un cálculo de los costes reales y un análisis de la forma de venta de la aplicación.

5.1 Planificación del proyecto

En este apartado se detallarán las tareas de las que se ha compuesto el proyecto desarrollado junto con el esfuerzo en días necesitado para llevarlas a cabo. Se incluirá además una representación de lo anterior en forma de diagrama de Gantt. Finalmente se compararán la planificación inicial del proyecto y la planificación real, es decir la planificación que finalmente se cumplió.

5.1.1 Planificación inicial

A continuación se detallan las distintas tareas que inicialmente estaban planificadas para ser llevadas a cabo durante el desarrollo del proyecto. Como se podrá ver después la planificación real difirió bastante de la planificación inicial.

- **Inicio**
  - **Lista de objetivos:** en esta tarea se marcaron los distintos objetivos que la aplicación debía cumplir para darse por finalizada.
- **Análisis**
  - **Estado de la cuestión:** se realizó un análisis de aspectos importantes para la realización del proyecto. Se realizó un estudio sobre las redes P2P, analizando dos redes en concreto, se profundizó en los conceptos de esteganografía y estegoanálisis y se realizó un estudio de las posibles acciones que podía llevar a cabo la aplicación una vez detectado algún contenido sospechoso.
  - **Casos de uso:** en esta tarea se realizó una especificación de los casos de uso de la aplicación.
  - **Requisitos:** aquí se realizó una especificación de los requisitos de la aplicación.
  - **Definición de pruebas:** aquí se definieron las distintas pruebas que debían superarse para poder dar la aplicación por finalizada.

- **Diseño**
  - **Arquitectura:** en esta tarea se estudió la arquitectura a utilizar por la aplicación desarrollada.
  - **Diseño de componentes:** aquí se estudiaron los distintos componentes que iban a integrar la aplicación.
  - **Diseño detallado:** en esta sección se estudiaron las distintas clases que iban a componer la aplicación y se realizó un diagrama de clases de las mismas.
- **Implementación**
  - **Programación:** esta tarea consistió en el desarrollo del código de la aplicación.
- **Pruebas**
  - **Ejecución de pruebas:** aquí se realizó el conjunto de pruebas definido en la tarea definición de pruebas.
  - **Toma de resultados:** en esta tarea se recogieron los resultados devueltos por la tarea anterior.
- **Documentación:** esta tarea consistió en la realización del presente documento. Se programaron diferentes entregables con el fin de mejorar el seguimiento del proyecto.

A continuación se puede ver el diagrama Gantt inicialmente planificado.

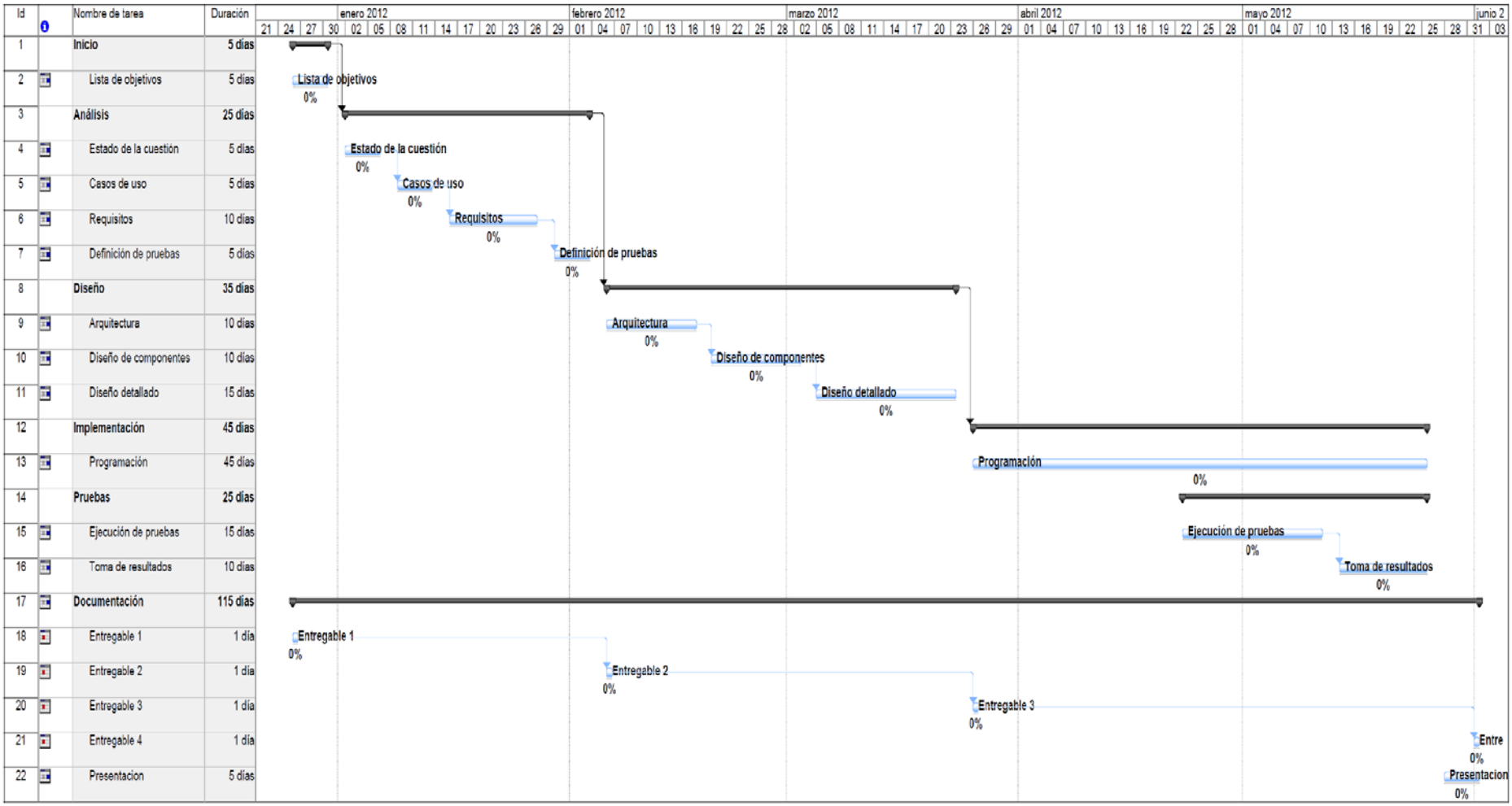


Ilustración 29: Diagrama de Gantt inicial

5.1.2 Planificación real

La planificación inicialmente planteada no se pudo llevar a cabo debido a que el tiempo que fue necesario dedicarle a las tareas del curso académico fue más del esperado y no se dispuso del tiempo deseado para el desarrollo del proyecto, lo que obligó a tener que retrasarlo para su presentación en la segunda sesión.

A continuación se puede ver el diagrama Gantt de la planificación que finalmente se llevó a cabo.

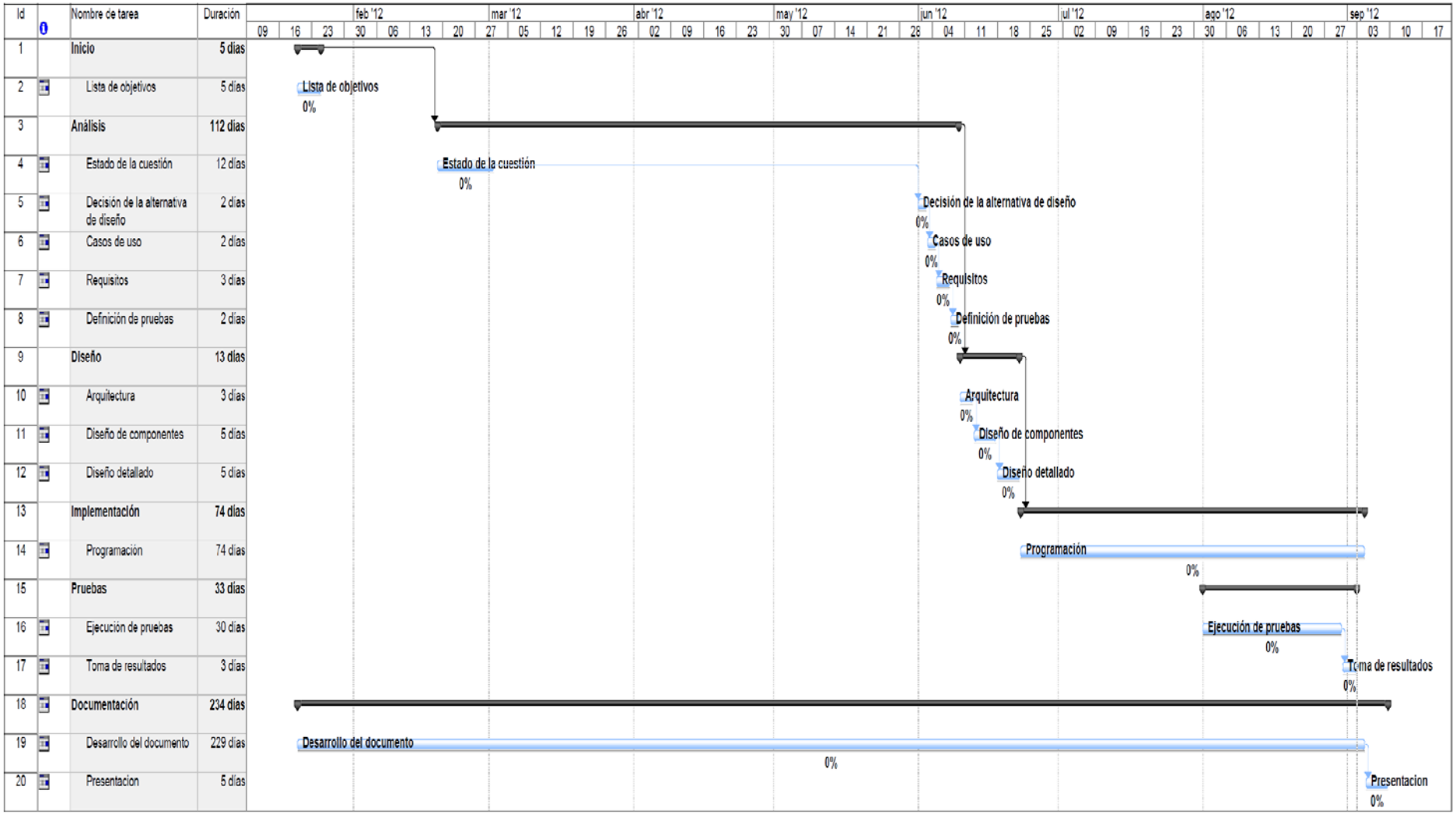


Ilustración 30: Diagrama de Gantt final

Como se puede observar en el diagrama varias tareas variaron su duración con respecto a la estimada. La tarea de análisis se inició durante la primera etapa del proyecto y no se finalizó hasta la segunda etapa, además se añadió la tarea de decisión de la alternativa de diseño. Dicha tarea consistió en el estudio de una serie de alternativas de diseño en base a diferentes criterios y la decisión de una de ellas. Además la tarea de diseño duró menos de lo esperado, pudiéndose dedicar más tiempo a las tareas de implementación y pruebas.

Finalmente se observa que se eliminaron los entregables programados, esto fue debido a que se siguió un desarrollo más libre en la elaboración del documento.

5.2 Medios técnicos empleados

En esta sección se describen los medios utilizados tanto hardware como software que se han utilizado para la realización del proyecto.

5.2.1 Hardware

Para la realización del proyecto únicamente se ha utilizado un ordenador de sobremesa con las siguientes características.

- **Procesador:** Intel(R) Pentium(R) D CPU 3.00 GHz
- **Memoria RAM:** 2 GB

5.2.2 Software

En la siguiente tabla se muestra el software utilizado para la realización del presente proyecto.

Tipo	Nombre	Página web
Sistema operativo	Windows 7 Ultimate	http://windows.microsoft.com/es-ES/windows/shop/windows-7/ultimate
Entorno de programación	Eclipse Classic 4.2	http://www.eclipse.org/
Gestión del proyecto	Microsoft Project Standard 2010	http://www.microsoft.com/project/en-us/project-standard-2010.aspx
Tratamiento de imágenes	Adobe Photoshop CS6 Extended Student and Teacher Edition	http://www.adobe.com/es/products/photoshop.html
Diagramas	Visual Paradigm for UML 10.0 Community Edition	http://www.visual-paradigm.com/product/vpuml/

Procesador de textos	Microsoft Office Word 2010	http://office.microsoft.com/es-es/word/
Presentaciones	Microsoft Office Power Point 2010	http://office.microsoft.com/es-es/powerpoint/

Tabla 46: Software utilizado

5.3 Análisis económico

En esta sección se detallarán los costes que suponen la realización del proyecto, estos costes son los derivados de las distintas partes que lo componen, es decir recursos humanos (RRHH), hardware y software. Además se comparará el presupuesto inicial con el presupuesto final para calcular la desviación sufrida.

Finalmente se realizará un análisis de la estrategia de venta más acorde con el tipo de aplicación desarrollada.

5.3.1 Metodología de estimación de costes

Para realizar la estimación de costes del proyecto éstos se dividirán en cuatro categorías: personal, hardware, software y costes indirectos. La adición de estos cuatro costes supone una aproximación muy buena al verdadero coste del proyecto. A continuación se analizará cada una de las categorías.

- **Personal:** el coste en personal se basa en el salario del Ingeniero Informático. Para el cálculo de dicho salario se consultaron los perfiles profesionales en informática en la página web de la ALI (Asociación de Titulados Universitarios Oficiales en Informática, (ALI, 2012)). No obstante el salario definitivo se decidió de forma un poco arbitraria ya que como se puede observar en el documento anterior el sueldo del ingeniero informático depende mucho del puesto que desempeñe, de su experiencia y además no se ofrece una tabla de salarios.
- **Hardware:** en el caso del hardware, se aplicará el reparto correspondiente de cada elemento hardware según el tiempo que se haya utilizado en el proyecto.
- **Software:** en este apartado se estimará el coste mediante el precio de las licencias de cada uno de los programas utilizados.
- **Costes indirectos:** aquí se tendrán en cuenta, entre otros, los siguientes gastos:
  - Gastos de luz.
  - Gastos de redes de comunicaciones

5.3.2 Análisis de costes estimados

Aquí se mostrará el coste del proyecto que se había calculado antes de iniciarse el mismo utilizando la metodología de estimación de costes especificada anteriormente.

5.3.2.1 Estimación del coste de personal

Los costes de personal o recursos humanos se componen únicamente de los honorarios del Ingeniero informático encargado del desarrollo del proyecto. Inicialmente, se planificó el proyecto con una carga horaria de unas 5 horas diarias de media durante 5 meses (supóngase que un mes equivale a 20 días laborables).

Concepto	Horas	Honorarios	Coste RRHH
Ingeniero informático	500 horas	20 €/hora	10000 €

Tabla 47: Coste de personal estimado

5.3.2.2 Estimación del coste del Hardware

A continuación se especificará el precio del hardware (precios con 18 % de IVA) utilizado en la elaboración del proyecto para poder estimar el coste que supone el hardware en el mismo.

Concepto	Unidades	Precio unitario	Vida útil estimada	Tiempo de uso	Coste para el proyecto
Ordenador con procesador Intel(R) Pentium(R) D CPU 3.00 GHz y 2 GB de memoria RAM	1	635,45 €	48 meses	5 meses	66,19 €

Tabla 48: Coste del Hardware estimado

5.3.2.3 Estimación del coste del Software

A continuación se hará un análisis económico del Software utilizado. Tal y como se dijo anteriormente los costes se calcularán utilizando el precio de las licencias del software.

Concepto	Licencia	Vida útil estimada	Tiempo de uso	Coste para el proyecto
Windows 7 Ultimate	319 €	120 meses	5 meses	13,29 €
Eclipse Classic 4.2	0 €	-	5 meses	0 €
Microsoft Project Standard 2010	775 €	120 meses	1 mes	6,46 €
Adobe Photoshop CS6 Extended Student and Teacher Edition	199 €	12 meses	3 meses	49,75 €
Visual Paradigm for UML 10.0 Community Edition	0 €	-	3 meses	0 €
Microsoft Office Hogar y Estudiantes 2010 (incluye Word 2010, Excel 2010, PowerPoint 2010 y OneNote 2010)	99 €	120 meses	4 meses	3,3 €
				72,80 €

Tabla 49: Coste del Software estimado

5.3.2.4 Estimación de los costes indirectos

A continuación se va a proceder a realizar una estimación de los costes indirectos del proyecto, como se dijo anteriormente estos costes se componen de los gastos de luz y de los gastos de redes de comunicaciones.

Para calcular los costes de redes de comunicaciones se supone que la tarificación mensual impuesta por el proveedor de servicios de internet (ISP) es de 46 €/mes. Para el cálculo de los gastos de luz se supone un gasto mensual de 70 €/mes. Teniendo en cuenta que la duración estimada del proyecto inicialmente era de 4 meses se obtiene la siguiente tabla de costes indirectos estimados.

Concepto	Precio mensual	Tiempo de uso	Coste para el proyecto
Conexión a internet	46 €	5 meses	230 €
Luz	70 €	5 meses	350 €
			580 €

Tabla 50: Costes indirectos estimados

5.3.2.5 Estimación del coste total

El cálculo del coste total estimado del proyecto simplemente se realiza calculando la suma de los costes calculados previamente.

Concepto	Coste
Coste de personal	10000 €
Coste de Hardware	66,19 €
Coste de Software	72,80 €
Costes indirectos	580 €
	10718,99 €

Tabla 51: Costes totales estimados

5.3.3 Análisis de costes reales

En este apartado se calculará el coste real del proyecto una vez este ha finalizado. La principal diferencia con los costes iniciales estimados es la diferencia de horas.

Tal y como se explicó en el apartado planificación real la duración final del proyecto fue de 8 meses, durante los primeros cinco meses del proyecto (enero 2012 – mayo 2012) se le dedicó al proyecto solamente una media de 1 hora diaria debido a motivos académicos. Durante la segunda parte del proyecto (junio 2012 – agosto 2012) se le dedicó una media de 5 horas útiles diarias. Por lo tanto el cálculo de horas totales es el siguiente.

Etapa del proyecto	Horas diarias dedicadas	Meses	Horas totales
Primera etapa	1 hora	5 meses	100 horas
Segunda etapa	5 horas	3 meses	300 horas
			400 horas

Tabla 52: Cálculo de las horas totales

Como se puede observar el número de horas dedicadas al proyecto ha sido el inferior al planificado inicialmente, por lo que los cotes en personal se reducirán. El coste en hardware, software y costes indirectos será mayor debido al mayor número de meses que se han utilizado estos bienes. A continuación se calculan las diferencias en los siguientes apartados.

5.3.3.1 Coste real de personal

En lo relativo al personal el aumento de horas dedicadas al proyecto implica un aumento de los costes de personal tal y como se muestra en la siguiente tabla.

Concepto	Horas	Honorarios	Coste RRHH inicial	Coste RRHH real	Diferencia
Ingeniero informático	400 horas	20 €/hora	10000 €	8000 €	2000 € (-)

Tabla 53: Coste de personal real

5.3.3.2 Coste real del Hardware

En cuanto a los costes reales de hardware, software y costes indirectos tal y como se dijo anteriormente éstos sufren un incremento debido al aumento de duración del proyecto.

Concepto	Precio unitario	Vida útil estimada	Tiempo de uso	Coste HW inicial	Coste HW real	Diferencia
Ordenador con procesador Intel(R) Pentium(R) D CPU 3.00 GHz y 2 GB de memoria RAM	635,45 €	48 meses	8 meses	66,19 €	105,91 €	39,72 € (+)

Tabla 54: Coste de Hardware real

5.3.3.3 Coste real del Software

Concepto	Licencia	Vida útil estimada	Tiempo de uso	Coste SW inicial	Coste SW real	Diferencia
Windows 7 Ultimate	319 €	120 meses	8 meses	13,29 €	21,27 €	7,98 € (+)
Eclipse Classic 4.2	0 €	-	8 meses	0 €	0 €	0 €
Microsoft Project Standard 2010	775 €	120 meses	2 meses	6,46 €	12,92 €	6,46 € (+)
Adobe Photoshop CS6 Extended Student and	199 €	12 meses	3 meses	49,75 €	49,75 €	0 €

Teacher Edition						
Visual Paradigm for UML 10.0 Community Edition	0 €	-	3 meses	0 €	0 €	0 €
Microsoft Office Hogar y Estudiantes 2010 (incluye Word 2010, Excel 2010, PowerPoint 2010 y OneNote 2010)	99 €	120 meses	7 meses	3,3 €	5,77 €	2,47 € (+)
				72,80 €	89,71 €	16,91 € (+)

Tabla 55: Coste de Software real

5.3.3.4 Costes indirectos reales

Concepto	Precio mensual	Tiempo de uso	Coste indirecto inicial	Coste indirecto real	Diferencia
Conexión a internet	46 €	8 meses	230 €	368 €	138 € (+)
Luz	70 €	8 meses	350 €	560 €	210 € (+)
			580 €	928 €	348 € (+)

Tabla 56: Costes indirectos reales

5.3.3.5 Costes totales reales

Concepto	Coste total inicial	Coste total real	Diferencia
Coste de personal	10000 €	8000 €	2000 € (-)
Coste de Hardware	66,19 €	105,91 €	39,72 € (+)
Coste de Software	72,80 €	89,71 €	16,91 € (+)
Costes indirectos	580 €	928 €	348 € (+)
	10718,99 €	9123,62 €	1595,37 € (-)

Tabla 57: Costes totales reales

Como se puede observar el coste final del proyecto ha sido de **9.123,62 €,** menos que el estimado inicialmente, con una desviación de **1.595,37 €** con respecto a

la planificación inicial. **Esta desviación supone un 14,88 % de ahorro sobre el presupuesto inicial de 10.718,99 €**. Como se puede observar la desviación es bastante pequeña comparada con la gran diferencia entre la planificación inicial del proyecto y la que se llevó finalmente a cabo.

5.3.4 Análisis de la forma de venta de la aplicación

El objetivo del desarrollo de la aplicación es obtener un beneficio de la misma para ello se utilizará un sistema de donaciones. La aplicación utiliza una licencia pública general de GNU o GNU General Public License (GNU GPL) en inglés. Este tipo de licencia fomenta el uso del software libre, permitiendo su libre distribución y modificación. Se ha elegido utilizar este tipo de licencia para facilitar la inclusión de nuevas funcionalidades como nuevos detectores, acciones o redes P2P implementadas.

Se supone que se desea obtener un retorno de la inversión o ROI[20] del 30 %, es decir, se desea obtener un importe de 14.000 €[21]. La siguiente tabla muestra el número de donaciones que se deberían recibir con distintos importes para lograr el importe deseado.

Importe	Número de donaciones necesarias	Donaciones al día	Años para ROI = 30 %
1,00 €	14000,00	10	3,84
3,00 €	4666,67	10	1,28
5,00 €	2800,00	10	0,77
10,00 €	1400,00	10	0,38

Tabla 58: Años para ROI del 30 % para distintos importes de donaciones

Calculando la media aritmética de los distintos importes de las donaciones se obtiene la siguiente tabla.

Importe	Número de donaciones	Donaciones al día	Años para ROI = 30 %
4,75 €	2947,37	10	0,81

Tabla 59: Años para ROI del 30 % para un importe medio por donación

[20] El ROI es un porcentaje que se calcula como el coeficiente entre beneficio y los costes iniciales. Para más información consultar (Denney, 2012).

[21] El cálculo del importe se realiza mediante la siguiente fórmula: $10718{,}99 * 1.30 \cong 14000$ €.

Si se realizan unas diez donaciones al día con unos 4,75 € de media por donación se logrará el valor del ROI deseado en poco menos de un año, es decir al finalizar ese periodo de tiempo se habrá recibido un importe de 14.000 €. El beneficio real de la aplicación se calcula como la resta entre dicho importe y los costes reales totales del desarrollo del proyecto:

$$14000 - 9123{,}62 = 4876{,}38\ €$$

Por lo tanto al finalizar dicho periodo se habrían obtenido unos beneficios de **4.876,38 €**.

Capitulo 6: Conclusiones y líneas futuras

En este capítulo se expondrán las conclusiones extraídas de la realización del proyecto y las líneas futuras por las que éste puede continuar.

6.1 Conclusiones sobre el proyecto

Cuando se inició el desarrollo del proyecto se marcaron una serie de objetivos los cuales debían ser cumplidos para darse por finalizado el mismo. En este punto, viendo el resultado final obtenido las conclusiones extraídas son en general bastante satisfactorias y positivas. Se ha logrado desarrollar una aplicación capaz de lo siguiente:

- Conectarse a la red BitTorrent.
- Buscar contenido en dicha red.
- Descargar archivos.
- Ejecutar detectores sobre los archivos descargados.
- Ejecutar acciones según el resultado de los detectores.
- Elaborar informes con el resultado de los detectores y las acciones.
- Ser ampliada con nuevas funcionalidades con facilidad.

Como se puede observar la aplicación cumple todos los objetivos inicialmente planificados. Además junto a las funcionalidades mencionadas anteriormente se incorporaron algunas funcionalidades más a la aplicación. Por un lado se implementó un sistema de configuración mediante archivos XML que le permite al usuario establecer ciertas propiedades de la aplicación como las carpetas donde almacenar las descargas, los informes, etc. Además se implementó un modo de funcionamiento mediante archivos XML, complementario al modo por comandos planificado inicialmente y que hace más ágil y automatizable la ejecución de la aplicación.

Por otro lado algunas funcionalidades, que aunque no se consideraban esenciales para la aceptación del proyecto sí que se estudió su implementación, no pudieron llevarse a cabo debido a cuestiones temporales. Entre estas se encontraba la posibilidad de conectarse a páginas webs con el mismo objetivo que busca la aplicación en su estado actual, es decir la descarga de archivos sospechosos para la detección de información oculta. Además entre las acciones a realizar por la aplicación se estudió la posibilidad de que ésta en caso de que detectara información oculta en un archivo recuperara las direcciones IP de los pares que poseen dicho archivo y se pusiera en contacto con ellos de alguna forma o avisara a la policía del suceso. Ambas cuestiones serán tratadas en el apartado **6.3 Líneas futuras.**

6.2 Conclusiones a nivel personal

La realización del proyecto ha supuesto una gran labor de superación personal para mí ya que jamás me había adentrado en solitario en el proceso de desarrollo de una aplicación pasando por todas sus fases, desde el análisis y diseño hasta llegar al fin a la implementación. A lo largo de la carrera he participado en algunos desarrollos grupales pero no ha sido hasta ahora que me he dado cuenta de la increíble labor que supone llevar a cabo un proyecto software y del orgullo que se siente una vez ves el producto finalizado y cumpliendo tus expectativas.

Por otra parte la experiencia ha sido muy enriquecedora ya que además de los conocimientos adquiridos en desarrollo de proyectos software he adquirido diversos conocimientos sobre temas en los que no había profundizado a lo largo de la carrera. Por un lado he adquirido conocimientos sobre las redes P2P en general como las distintas clasificaciones que hay y sobre la red BitTorrent en particular y su funcionamiento. También analizando el código del programa Vuze y pese al desconcierto inicial (el programa tiene un gran número de funcionalidades y esto se ve reflejado en el código con una cantidad enorme de clases y paquetes java) he adquirido algunas habilidades para el análisis de un programa de tal y magnitud y he aprendido algunos aspectos de la implementación de un cliente P2P. Por otro lado he profundizado en cuestiones como las disciplinas de esteganografía y estegoanálisis, las cuales han sido de un gran interés para mí. Finalmente he adquirido experiencia en la programación en java, sobre todo en cuestiones como la paralelización.

Finalmente el desarrollo del proyecto también me ha hecho obtener valores positivos a nivel personal como la responsabilidad y una mayor habilidad de planificación.

6.3 Líneas futuras

En este apartado se explican los diferentes aspectos por los que se podría continuar el desarrollo del proyecto con el fin de mejorarlo e incluirle nuevas funcionalidades.

6.3.1 Nuevas redes P2P

Aunque inicialmente la aplicación sólo implementa la red BitTorrent está diseñada para que la inclusión de nuevas redes se realice con facilidad. La red más interesante para ser incluida debido a sus características y a que es una de las más usadas es la red eDonkey2000. Para su implementación el desarrollador se apoyaría en algún cliente conocido y de licencia GPL de dicha red como puede ser Emule, Amule o Jmule siendo este el preferido al estar desarrollado en java al igual que el resto de la

aplicación. No obstante, si se desea trabajar con Amule o Emule (ambos escritos en C++), Java dispone del framework JNI (Java Native Interface) que permite que un programa escrito en Java se comunique con otros escritos en C, C++ o ensamblador aunque no se recomienda su uso por su dificultad y porque se pierden algunas características esenciales del lenguaje Java como es su portabilidad o el recolector de basura.

6.3.2 Nuevos detectores

En cuanto a la detección de contenido oculto hay distintas opciones por las que continuar el desarrollo de la aplicación. Por un lado se podrían desarrollar detectores para nuevos algoritmos esteganográficos como el algoritmo F5 (Westfeld, 2012). Dicho algoritmo posee una alta capacidad esteganográfica, una alta eficiencia mediante una codificación por matrices, previene ataques visuales, resiste ataques estadísticos, es de código abierto y utiliza imágenes JPEG como objetos portadores.

Adicionalmente al desarrollo de nuevos detectores para algoritmos esteganográficos específicos se podrían desarrollar técnicas para la detección de contenido oculto en otros contenidos que no sean imágenes, como por ejemplo audio o vídeo.

6.3.3 Nuevas acciones

Como ya se ha dicho anteriormente la aplicación inicialmente estaba planificada para ejecutar algunas acciones más de las que dispone actualmente pero desgraciadamente por cuestiones de falta de tiempo no se pudieron implementar. Actualmente la aplicación una vez que descarga un archivo y detecta que éste posee información oculta ofrece la posibilidad de recopilar el conjunto de direcciones IP de los pares que se encuentran compartiendo dicho archivo y guardarlas en un archivo de texto plano. Como extensión a lo anterior se había planificado que la aplicación pudiera ejecutar dos acciones, las cuales se detallarán a continuación.

6.3.3.1 Comunicación con los usuarios

Una vez obtenido el conjunto de direcciones IP la aplicación utilizaría dicho conjunto para ponerse en contacto con los usuarios que hubieran descargado el archivo con la información oculta y les informaría de la existencia de dicha información. Ante la imposibilidad de conocer la naturaleza de la información esta acción requeriría la participación del usuario para que éste analizara si dicha información es de carácter fraudulento o no y diera su autorización para la ejecución de la acción.

Algunos clientes P2P implementan un sistema de mensajería y sería éste el mecanismo mediante el cual se realizaría la comunicación con los usuarios.

6.3.3.2 Denuncia a la policía

Una vez obtenido el conjunto de direcciones IP y en caso de que la información oculta sea de carácter fraudulento la aplicación contactaría con la policía para informarle del suceso y de los usuarios que están en posesión del archivo. Ante la imposibilidad de conocer la naturaleza de la información esta acción requeriría la participación del usuario para que éste analizara si dicha información es de carácter fraudulento o no y diera su autorización para la ejecución de la acción. Además la propia policía podría hacer uso de la aplicación de forma activa.

La tramitación de la denuncia se haría a través del Twitter oficial de la policía *@policia* y utilizando la librería de java *twitter4j* para realizar el proceso de autenticación con Twitter y el envío de los tweets. Para el proceso de autenticación con Twitter además de poseer una cuenta en dicha red social se debe ingresar en la sección *desarrolladores* y registrar una aplicación con el fin de obtener los credenciales de autenticación.

La librería se puede descargar en la siguiente dirección http://twitter4j.org/en/index.html. En dicha web se puede encontrar además documentación y códigos de ejemplo.

Anexo A: Manual de la aplicación

En este anexo se desarrolla un manual del correcto uso de la aplicación. Se detallan instrucciones para la ejecución de las distintas operaciones que puede llevar a cabo la aplicación junto con capturas de ejemplos de ejecución de cada una de ellas.

La aplicación tiene dos modos de ejecución, mediante comandos y mediante archivos XML. Para elegir el modo de ejecución de la aplicación se utiliza el argumento *mode* (-m). Si dicho argumento toma el valor *command* se le indica a la aplicación que se debe utilizar el modo de ejecución mediante comandos, además cada operación a ejecutar por la aplicación tiene unos argumentos que son necesarios para su ejecución y deben ser especificados por éste y otros argumentos opcionales que pueden incluirse o no. Si el argumento *mode* toma el valor XML se le indica a la aplicación que se debe utilizar el modo de ejecución mediante archivos XML, además se le debe especificar el argumento *path* con la ruta que localiza al archivo XML a utilizar y el cual contiene los argumentos para ejecutar una operación concreta. La aplicación además cuenta con el comando *help* (--help) para solicitar ayuda sobre el funcionamiento de la aplicación y el comando *verbose* (-v|--verbose) para solicitar que se informe información adicional acerca del proceso que se está realizando.

A continuación se detalla el correcto modo de ejecución de las distintas operaciones llevadas a cabo por la aplicación para ambos modos junto con capturas con ejecuciones de ejemplo. Además se explica el sistema de configuración que utiliza la aplicación.

A.1 Operación de búsqueda

La función de esta operación es la búsqueda de archivos para descargar en la red elegida. A continuación se especifica su modo de uso para los distintos modos de ejecución junto con capturas de ejemplo.

A.1.1 Modo por comandos

Los argumentos necesarios para ejecutar una operación de búsqueda son los siguientes (obtenidos de la ayuda de la aplicación):

- **Mode (-m):** Especifica el modo de ejecución: [command|xml]. Debe tomar el valor command.
- **Operation (-o):** Especifica la operación a realizar: [search|download|auto|import|delete|show|clean]. Debe tomar el valor search.

- **Network (-n):** Especifica la red P2P a utilizar: [all|torrent]. Puede tomar el valor torrent o el valor all.
- **SearchQuery (-s):** Especifica la cadena de búsqueda a utilizar en una operación de búsqueda o en una operación automatizada.
- **InformType (-i):** Especifica el tipo de informe búsqueda a utilizar en una operación de búsqueda: [pdf|txt].

Opcionalmente se pueden añadir los siguientes argumentos:

- **Out (--out):** Especifica el nombre del informe búsqueda a utilizar en una operación de búsqueda. (default: SearchInform).
- **Size (--size):** Especifica el tamaño límite de la búsqueda en bytes utilizado para filtrar en una operación de búsqueda o en una operación automatizada.
- **Nseeds (--nseeds):** Especifica el número mínimo de seeds utilizado para filtrar en una operación de búsqueda o en una operación automatizada.
- **Hash (--hash):** Especifica el hash utilizado para filtrar en una operación de búsqueda o en una operación automatizada.
- **Sort (--sort):** Especifica el criterio de ordenación de los resultados a utilizar en una operación de búsqueda o en una operación automatizada: [date|name|npeers|nseeds|size]. (default: nseeds).
- **SortMode (--sortmode):** Especifica el modo de ordenación de los resultados a utilizar en una operación de búsqueda o en una operación automatizada: [ascendent|descendent]. (default: descendent).

A continuación se incluyen distintos ejemplos de ejecución:

Comando de ejecución	Descripción
java -jar stegocrawler.jar -m command -o search -n torrent -s “game of thrones” -i txt	Operación de búsqueda básica utilizando el termino de búsqueda “game of thrones” y utilizando un archivo de texto plano para guardar los resultados de la búsqueda
java -jar stegocrawler.jar -m command -o search -n torrent -s “game of thrones” -i txt --nseeds 20	Operación de búsqueda utilizando el termino de búsqueda “game of thrones”, utilizando un archivo de texto plano para guardar los resultados de la búsqueda y filtrando los resultados para obtener únicamente los que tengan más de 20 seeds
java -jar stegocrawler.jar -m command -o search -n torrent -s “jackson” -i pdf -o “informeBusqueda” --sort name --sortmode descendent –size 1073741824 –nseeds 10	Operación de búsqueda utilizando el termino de búsqueda “jackson”, utilizando un archivo pdf para guardar los resultados de la búsqueda y al que se llamará “informeBusqueda”, filtrando los resultados eliminando los que ocupen más de 1 GB (1 GB = 1073741820 bytes) o tengan menos de 10 seeds y ordenándolos por nombre de manera descendente

Tabla 60: Ejemplos de ejecución de operaciones de búsqueda

A.1.2 Modo por archivos XML

A continuación se va a mostrar una representación del esquema XML utilizado para la validación. Únicamente se mostrará la parte del esquema relacionada con la operación de búsqueda.

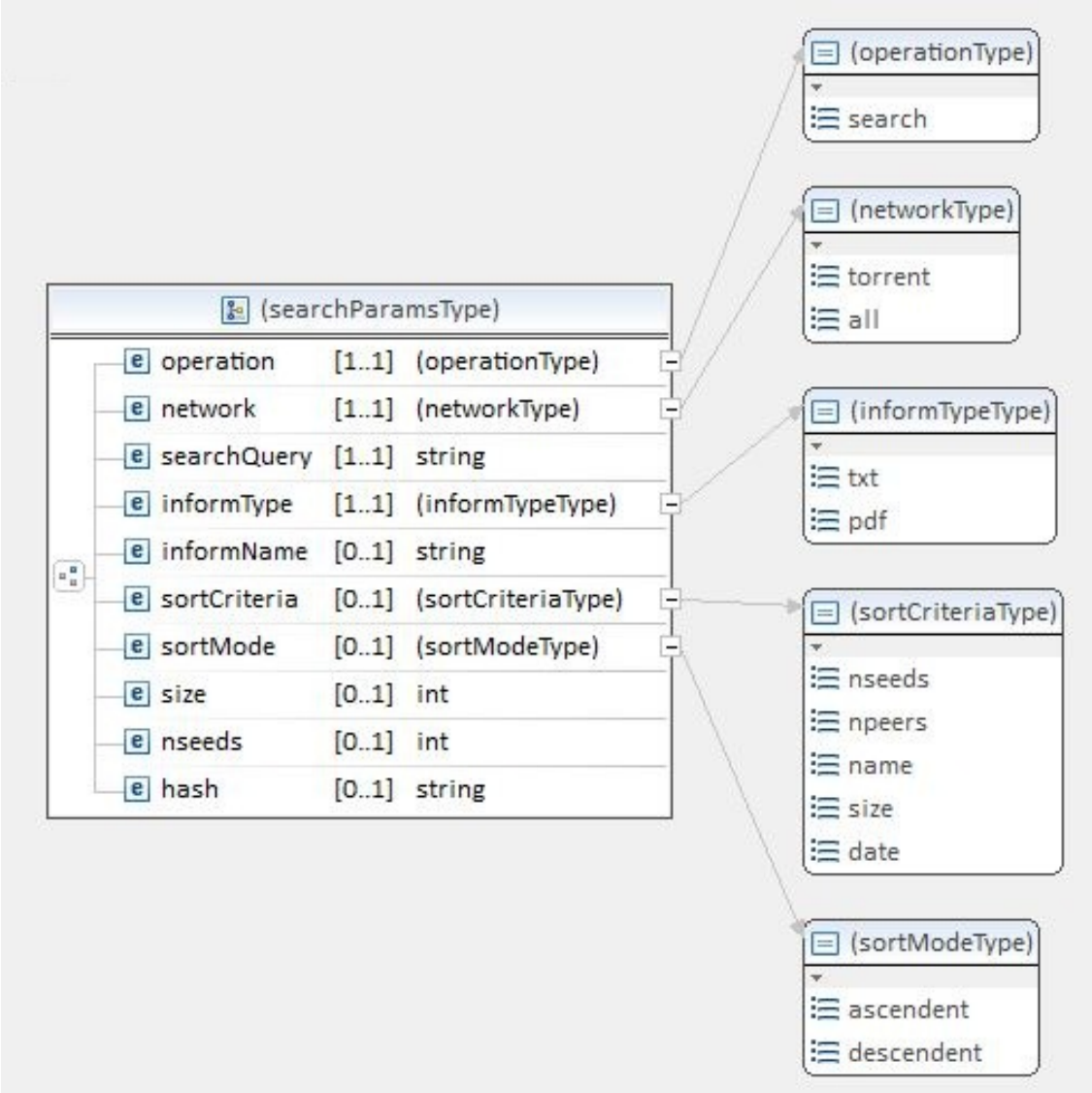


Ilustración 31: Esquema XML de la operación de búsqueda

Como se puede observar los argumentos necesarios son los mismos que para realizar la operación mediante el modo de ejecución mediante comandos.

Finalmente se muestra un ejemplo de un archivo XML con los argumentos necesarios para realizar una operación de búsqueda.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:xmlConfig xmlns:tns="http://www.example.org/XMLConfigSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.org/XMLConfigSchema ../src/resource
s/XMLConfigSchema.xsd ">
  <tns:params>
      <tns:searchParams>
            <tns:operation>search</tns:operation>
            <tns:network>torrent</tns:network>
            <tns:searchQuery>a</tns:searchQuery>
            <tns:informType>txt</tns:informType>
            <tns:informName>informe</tns:informName>
            <tns:sortCriteria>name</tns:sortCriteria>
            <tns:sortMode>descendent</tns:sortMode>
            <tns:hash>7261B5B80BC56224D5403DBEF0E118DE35253078
            </tns:hash>
            <tns:size>1073741824</tns:size>
            <tns:nseeds>20</tns:nseeds>
            </tns:searchParams>
  </tns:params>
  <tns:verbose>true</tns:verbose>
</tns:xmlConfig>
```

A.1.3 Capturas

Las siguientes ilustraciones son capturas de un ejemplo de ejecución de una operación de búsqueda.

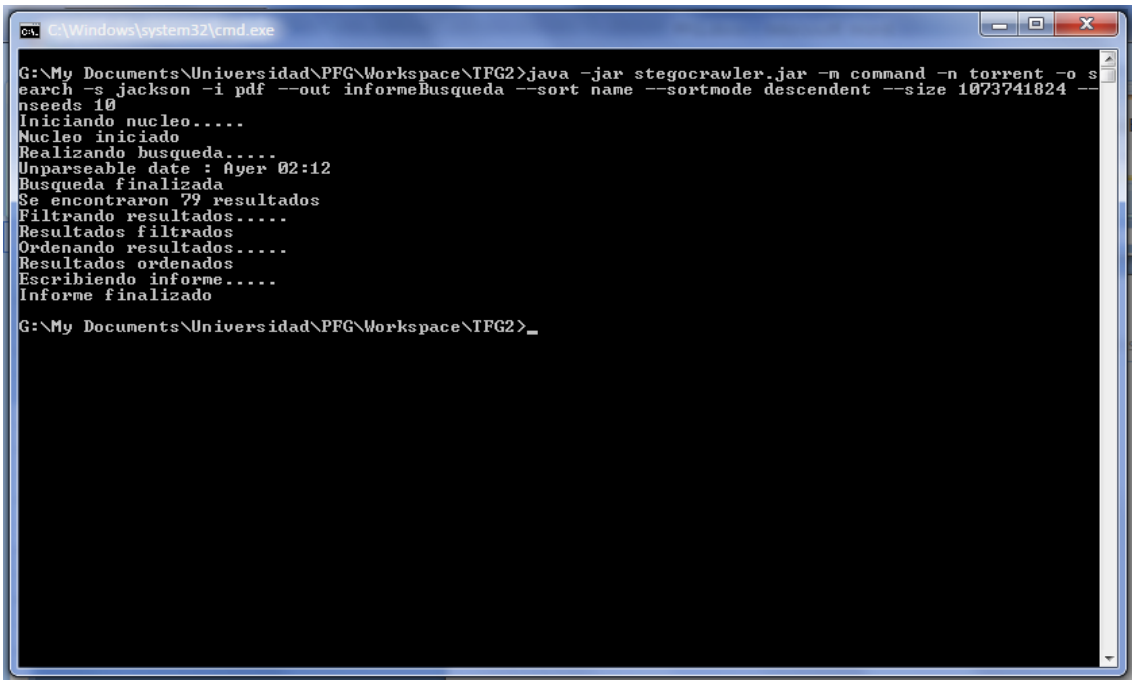
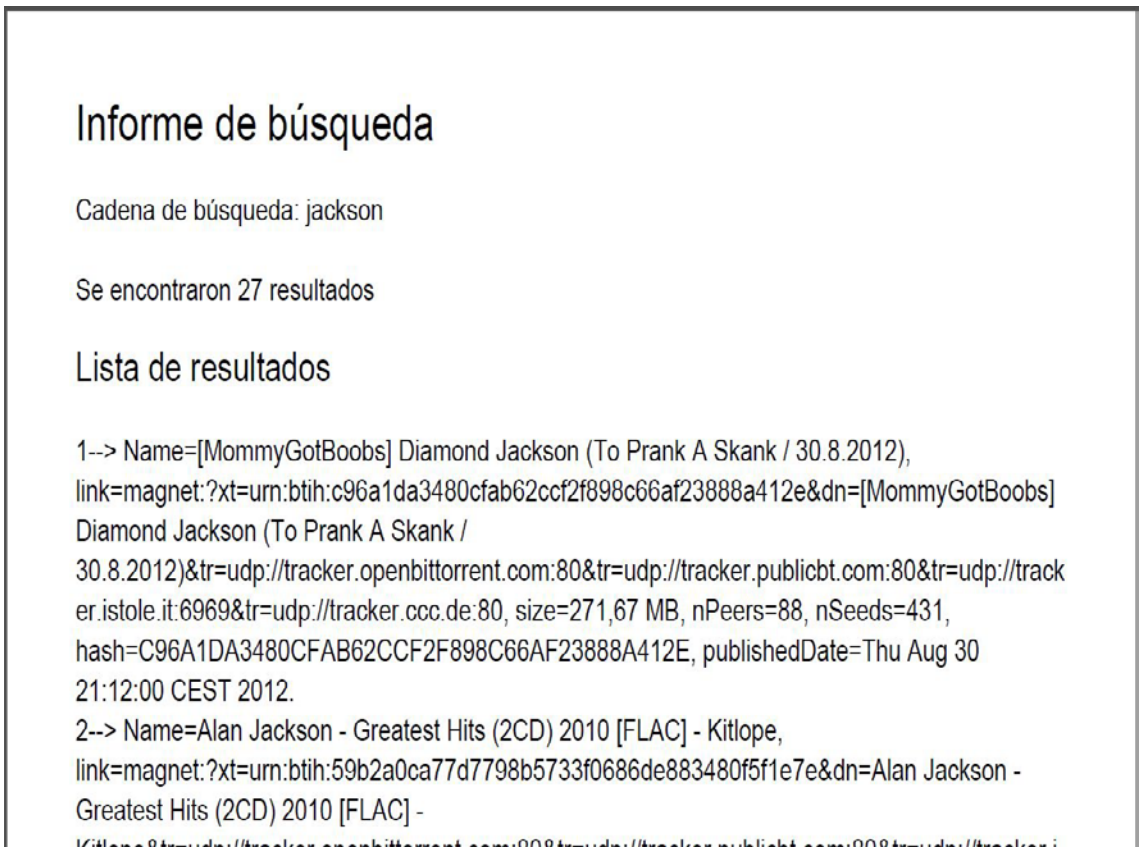


Ilustración 32: Salida mostrada por la operación de búsqueda

En la ilustración anterior se puede observar la salida mostrada por la aplicación al ejecutar una operación de búsqueda.


Informe de búsqueda

Cadena de búsqueda: jackson

Se encontraron 27 resultados

Lista de resultados

1--> Name=[MommyGotBoobs] Diamond Jackson (To Prank A Skank / 30.8.2012), link=magnet:?xt=urn:btih:c96a1da3480cfab62ccf2f898c66af23888a412e&dn=[MommyGotBoobs] Diamond Jackson (To Prank A Skank / 30.8.2012)&tr=udp://tracker.openbittorrent.com:80&tr=udp://tracker.publicbt.com:80&tr=udp://tracker.istole.it:6969&tr=udp://tracker.ccc.de:80, size=271,67 MB, nPeers=88, nSeeds=431, hash=C96A1DA3480CFAB62CCF2F898C66AF23888A412E, publishedDate=Thu Aug 30 21:12:00 CEST 2012.
2--> Name=Alan Jackson - Greatest Hits (2CD) 2010 [FLAC] - Kitlope, link=magnet:?xt=urn:btih:59b2a0ca77d7798b5733f0686de883480f5f1e7e&dn=Alan Jackson - Greatest Hits (2CD) 2010 [FLAC] -

Ilustración 33: Informe de búsqueda generado

En la ilustración anterior se muestra el informe en formato PDF generado por la aplicación como resultado de la ejecución de la operación de búsqueda.

A.2 Operación de descarga

La función de esta operación es la descarga de archivos de la red elegida para localizar una posible información oculta. Esta operación se divide en dos, descarga de archivos torrent y descarga de archivos. Cualquier operación de descarga puede ser cancelada escribiendo la orden "cancel" en la consola y pulsando la tecla intro, eliminando así cualquier porción del archivo descargada. Adicionalmente se podrá finalizar la ejecución del programa escribiendo la orden "exit" y pulsando la tecla intro durante el proceso de descarga de algún archivo. Finalmente cualquier descarga que no haya sido finalizada podrá ser reanudada volviendo a seleccionar el mismo archivo torrent en otra ejecución.

A continuación se describen los modos de uso de la operación de descarga de archivos torrent y de la operación de descarga de archivos junto con capturas de ejemplo.

A.2.1 Descarga de archivo torrent

La función de esta operación es la descarga de archivos torrent los cuales contienen la información necesaria para la descarga de un archivo. A continuación se especifica su modo de uso para los distintos modos de ejecución junto con capturas de ejemplo.

A.2.1.1 Modo por comandos

Los argumentos necesarios para ejecutar una operación de descarga de un archivo torrent son los siguientes:

- **Mode (-m):** Especifica el modo de ejecución: [command|xml]. Debe tomar el valor command.
- **Operation (-o):** Especifica la operación a realizar: [search|download|auto|import|delete|show|clean]. Debe tomar el valor download.
- **Network (-n):** Especifica la red P2P a utilizar: [all|torrent]. Debe tomar el valor torrent.
- **Url (-u):** Especifica la url o el enlace magnet que localiza al archivo torrent en una operación de descarga de archivo torrent. Se permite especificar un conjunto de urls para la descarga múltiple de archivos torrent.

Opcionalmente se pueden añadir los siguientes argumentos:

- **Referrer (-r):** Especifica una url de referencia para la descarga del archivo torrent en caso de que sea necesaria en una operación de descarga de archivo torrent. Se permite especificar un conjunto de urls de referencia para la descarga múltiple de archivos torrent.

A continuación se incluye un ejemplo de ejecución:

Comando de ejecución	Descripción
java -jar stegocrawler.jar -m command -o download -n torrent -u “magnet:?xt=urn:btih:HK74OKXSCCTKNLHAMP54NN4ZE4TBISZ7&tr=http://tracker.mininova.org/announce”	Operación básica de descarga de un archivo torrent utilizando la URL especificada por el argumento “-u”

Tabla 61: Ejemplo de ejecución de una operación de descarga de archivo torrent

A.2.1.2 Modo por archivos XML

A continuación se va a mostrar una representación del esquema XML utilizado para la validación. Únicamente se mostrará la parte del esquema relacionada con la operación de descarga, válido tanto para la operación de descarga de archivos torrent como para la operación de descarga de archivos.

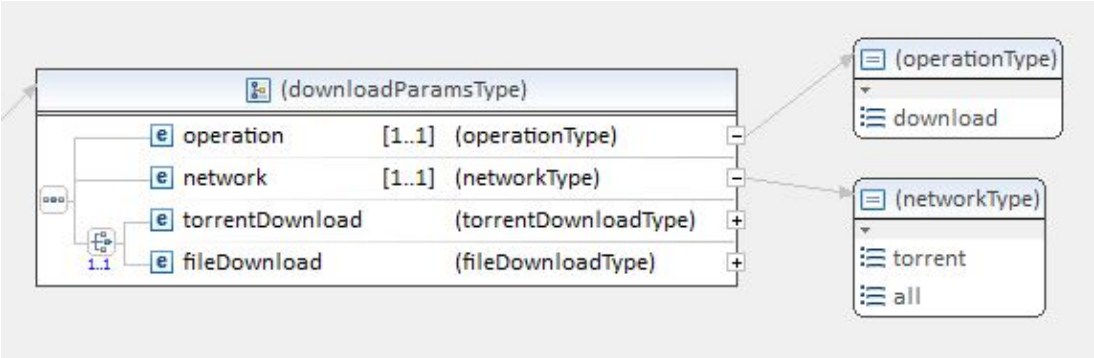


Ilustración 34: Esquema XML de la operación de descarga 1

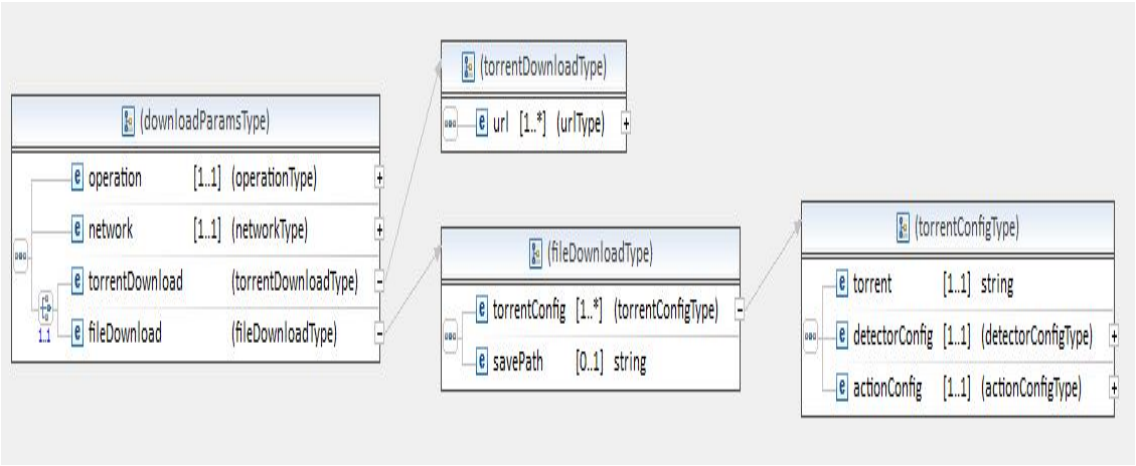


Ilustración 35: Esquema XML de la operación de descarga 2

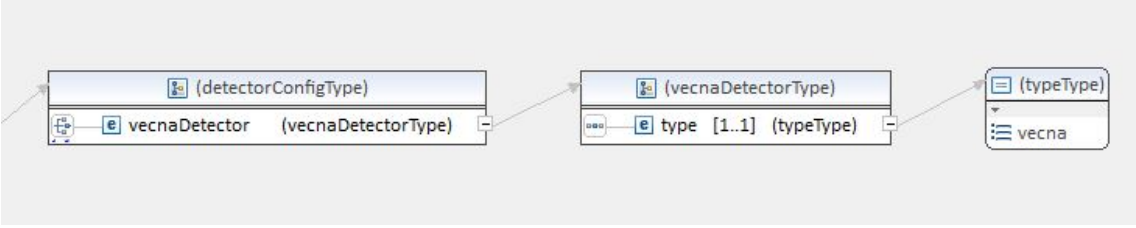


Ilustración 36: Esquema XML de la operación de descarga 3

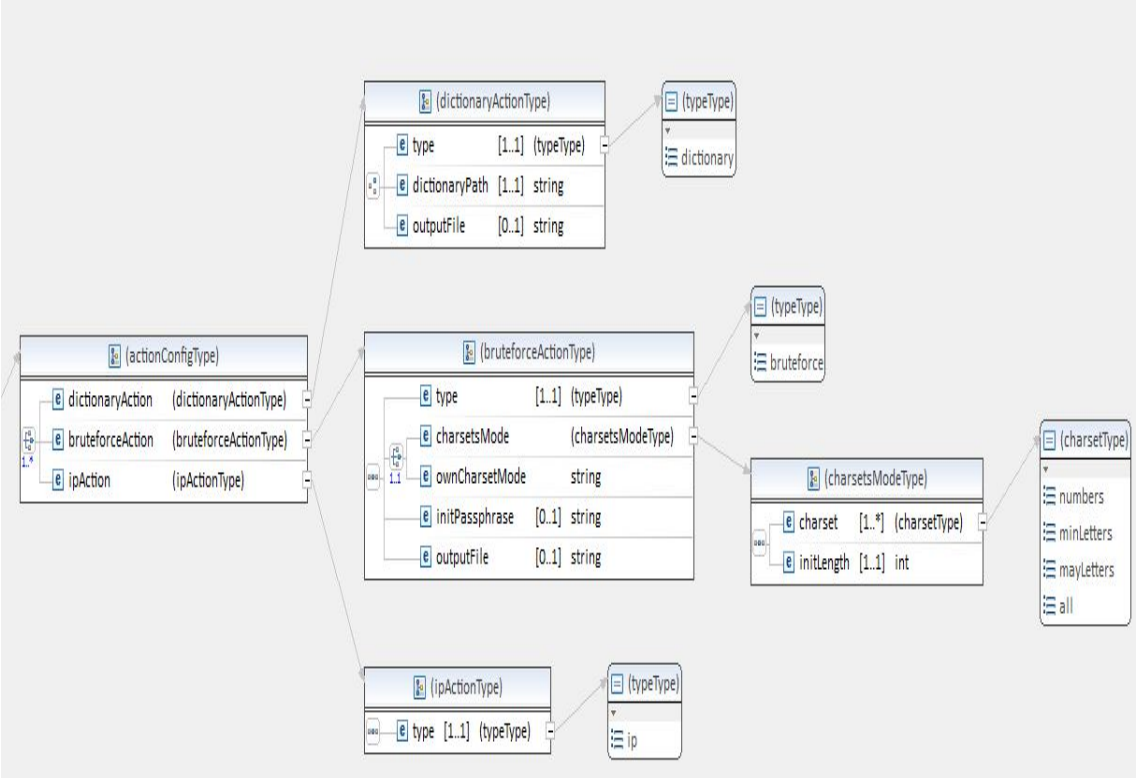


Ilustración 37: Esquema XML de la operación de descarga 4

Finalmente se muestra un ejemplo de un archivo XML con los argumentos necesarios para realizar una operación de descarga de archivo torrent.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:xmlConfig xmlns:tns="http://www.example.org/XMLConfigSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.org/XMLConfigSchema ../src/resource
s/XMLConfigSchema.xsd ">
  <tns:params>
      <tns:downloadParams>
            <tns:operation>download</tns:operation>
            <tns:network>torrent</tns:network>
            <tns:torrentDownload>
                <tns:url>
                magnet:?xt=urn:btih:HK74OKXSCCTKNLHAMP54NN4ZE4TBISZ7&tr=ht
                tp://tracker.mininova.org/announce
                </tns:url>
            </tns:torrentDownload>
      </tns:downloadParams>
  </tns:params>
  <tns:verbose>true</tns:verbose>
</tns:xmlConfig>
```

A.2.1.3 Capturas

En la siguiente ilustración se puede observar la salida mostrada por la aplicación al ejecutar una operación de descarga de un archivo torrent.

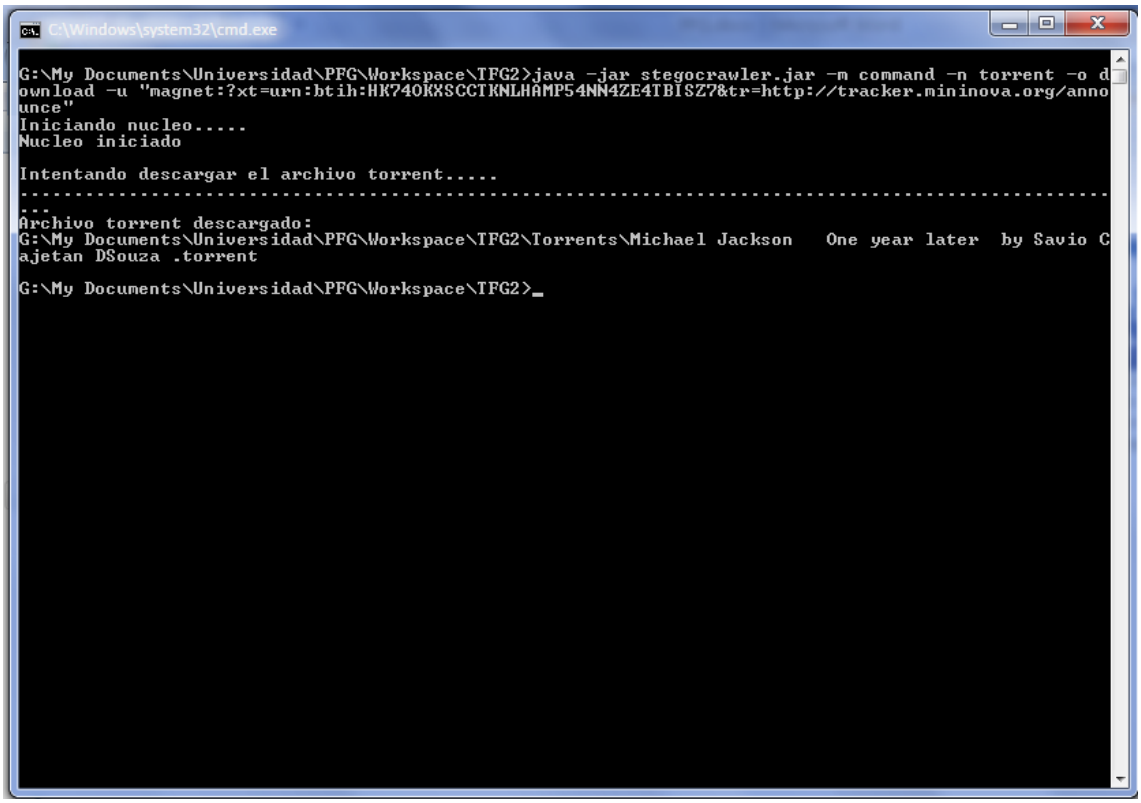


Ilustración 38: Salida mostrada por la operación de descarga de archivo torrent

A.2.2 Descarga de archivo

La función de esta operación es la descarga de archivos con el fin de detectar si contienen información oculta. A continuación se especifica su modo de uso para los distintos modos de ejecución junto con capturas de ejemplo.

A.2.2.1 Modo por comandos

Los argumentos necesarios para ejecutar una operación de descarga de un archivo son los siguientes:

- **Mode (-m):** Especifica el modo de ejecución: [command|xml]. Debe tomar el valor command.

- **Operation (-o):** Especifica la operación a realizar: [search|download|auto|import|delete|show|clean]. Debe tomar el valor download.
- **Network (-n):** Especifica la red P2P a utilizar: [all|torrent]. Puede tomar el valor torrent y el valor all.
- **Torrent (-t):** Especifica la ruta a un archivo torrent o un directorio contenedor de archivos torrent en una operación de descarga de archivo. Estos archivos contienen la información necesaria para la descarga de los archivos. Se permite especificar un conjunto de rutas a archivos torrent para la descarga múltiple de archivos.
- **DetectorConf (--dconf):** Especifica la configuración de los detectores. El correcto modo de uso de este argumento se mostrará en el apartado **A.4 Detectores**.
- **ActionConf (--aconf):** Especifica la configuración de las acciones. El correcto modo de uso de este argumento se mostrará en el apartado **A.5 Acciones**.

Opcionalmente se pueden añadir los siguientes argumentos:

- **Path (-p):** Especifica la ruta en la que se descargarán los archivos en una operación de descarga de archivo o en una operación automatizada. Si el usuario no introduce una ruta de descarga o introduce una ruta que no es válida la aplicación le pregunta si desea usar la ruta de descargas por defecto configurada o si desea introducir una nueva ruta de descargas.

A continuación se incluyen distintos ejemplos de ejecución:

Comando de ejecución	Descripción
java -jar stegocrawler.jar -m command -o donwload -n torrent -t “prueba.torrent” --dconf [type:vecna] --aconf [type:bruteforce,charsets:minLetters,initLength:4]	Operación básica de descarga de un archivo utilizando el archivo torrent especificado por el argumento “-t”, especificando que se utilice un detector de tipo detección Vecna y una acción de tipo ataque por fuerza bruta utilizando el conjunto de caracteres de las letras minúsculas con una longitud de 4 para la clave inicial
java -jar stegocrawler.jar -m command -o donwload -n torrent –t “GameOfThrones.torrent” –t “House.torrent” –dconf [type:vecna] –dconf [type:vecna] --aconf [type:dictionary,dictionaryPath:dictionary.txt] –aconf [type:ip]	Operación de descarga de múltiples archivos utilizando los archivos torrent especificados por el argumento “-t”, especificando que se utilicen detectores de tipo detección Vecna para ambas descargas y una acción de tipo ataque por diccionario utilizando para ello el archivo especificado mediante el parámetro “dictionaryPath” para la primera descarga y una acción de obtener las direcciones IP de los usuarios poseedores del archivo para la segunda

Tabla 62: Ejemplo de ejecución de una operación de descarga de archivo

A.2.2.2 Modo por archivos XML

A continuación se muestra un ejemplo de un archivo XML con los argumentos necesarios para realizar una operación de descarga de archivo. Los esquemas XML de la operación de descarga de un archivo se mostraron en la sección **A.2.1: Descarga de archivo torren**t.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:xmlConfig xmlns:tns="http://www.example.org/XMLConfigSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.org/XMLConfigSchema ../src/resource
s/XMLConfigSchema.xsd ">
  <tns:params>
      <tns:downloadParams>
            <tns:operation>download</tns:operation>
            <tns:network>torrent</tns:network>
            <tns:fileDownload>
                  <tns:torrentConfig>
                        <tns:torrent>prueba.torrent</tns:torrent>
                        <tns:detectorConfig>
                              <tns:vecnaDetector>
                                    <tns:type>vecna</tns:type>
                              </tns:vecnaDetector>
                        </tns:detectorConfig>
                        <tns:actionConfig>
                              <tns:dictionaryAction>
                                    <tns:type>dictionary</tns:type>
                                    <tns:dictionaryPath>dictionary.txt
                                    </tns:dictionaryPath>
                                    <tns:outputFile>extracted
                                    </tns:outputFile>
                              </tns:dictionaryAction>
                              <tns:bruteforceAction>
                                    <tns:type>bruteforce</tns:type>
                                    <tns:charsetsMode>
                                        <tns:charset>minLetters
                                        </tns:charset>
                                        <tns:charset>numbers
                                        </tns:charset>
                                        <tns:initLength>4
                                        </tns:initLength>
                                    </tns:charsetsMode>
                                    <tns:outputFile>extracted
                                    </tns:outputFile>
                              </tns:bruteforceAction>
                              <tns:ipAction>
                                    <tns:type>ip</tns:type>
                              </tns:ipAction>
                         </tns:actionConfig>
                  </tns:torrentConfig>
                  <tns:savePath>Descargas</tns:savePath>
            </tns:fileDownload>
      </tns:downloadParams>
  </tns:params>
  <tns:verbose>true</tns:verbose>
</tns:xmlConfig>
```

A.2.2.3 Capturas

Las siguientes capturas corresponden a una operación de descarga de una imagen en formato PNG. En dicha imagen se ocultó mediante el programa Vecna información conocida utilizando una clave también conocida. Posteriormente se creó un archivo torrent llamado “prueba.torrent” a partir de dicha imagen utilizando el programa Vuze.

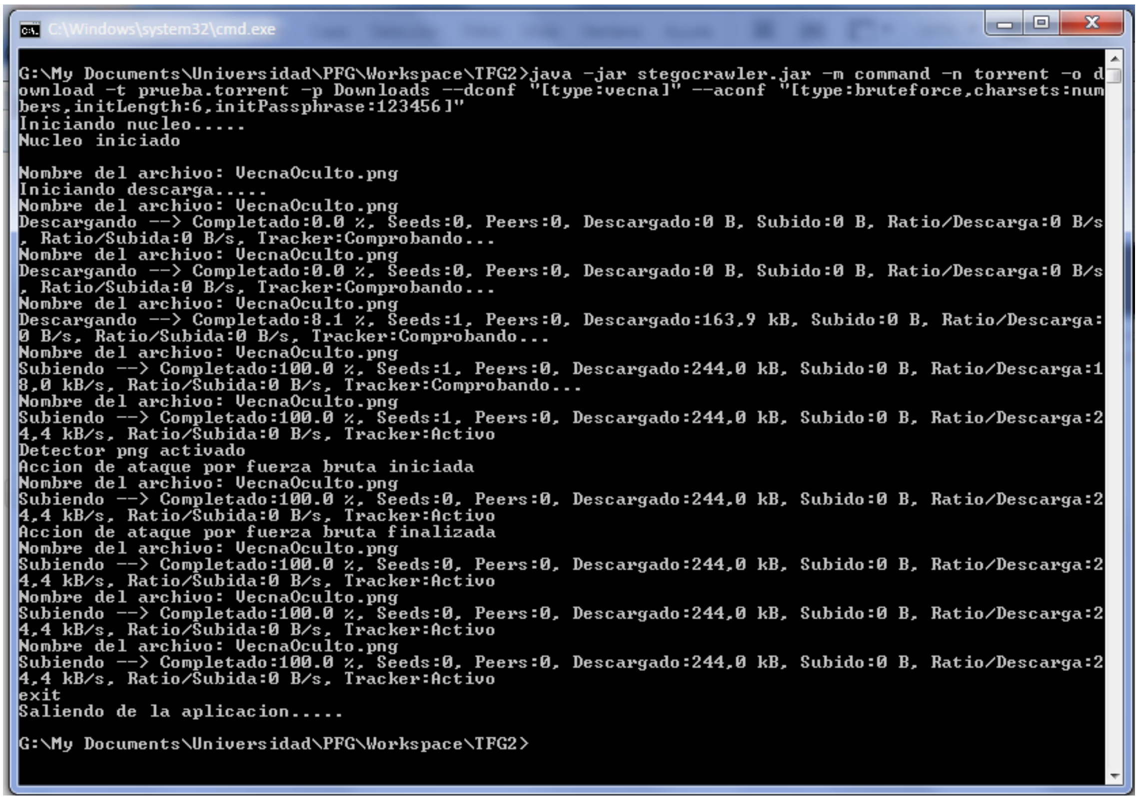


Ilustración 39: Salida mostrada por la operación de descarga de archivo

La ilustración anterior muestra la salida mostrada por pantalla como resultado de la ejecución de la operación.

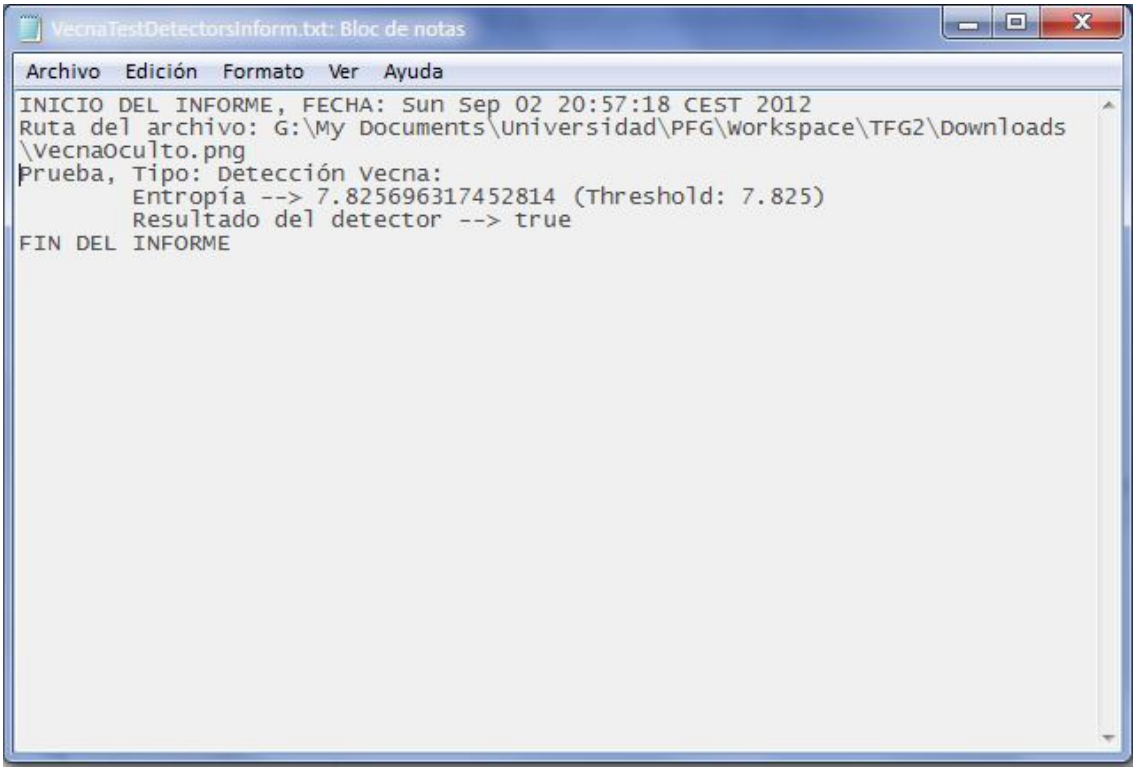


Ilustración 40: Informe de detectores

En la ilustración anterior se muestra el informe generado por la aplicación como resultado de la ejecución de los detectores configurados. Como se puede observar el detector lanzado detectó información oculta en el archivo descargado.

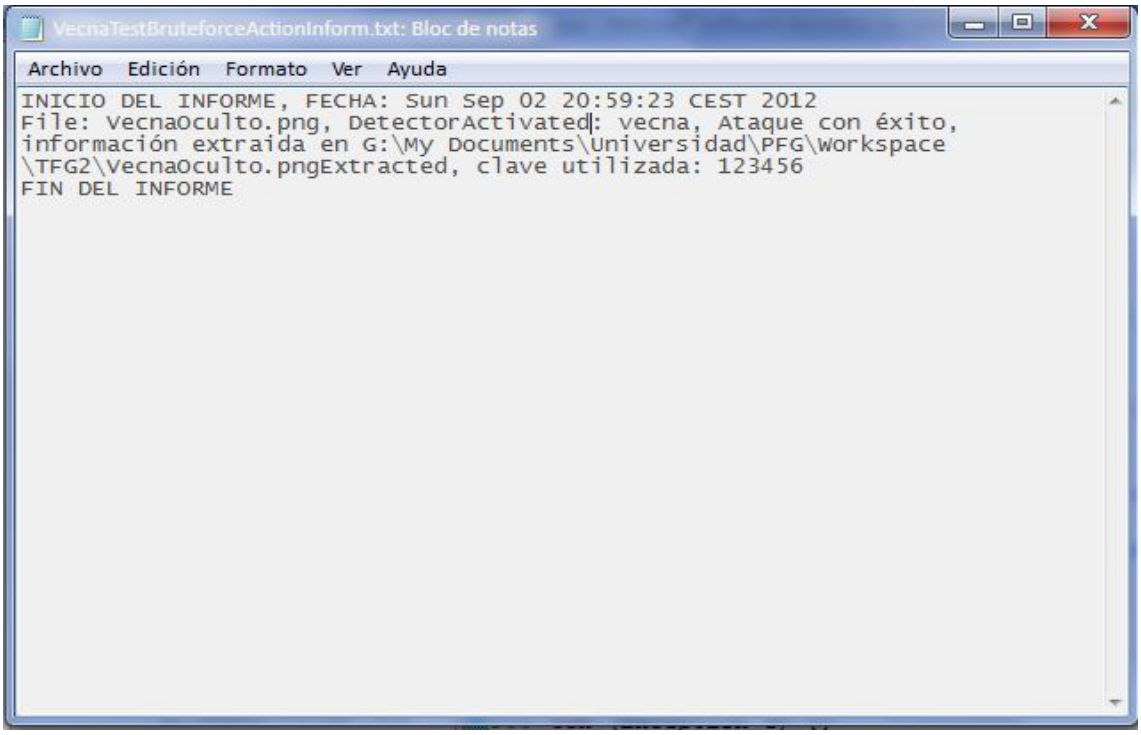


Ilustración 41: Informe de acciones

En la ilustración anterior se muestra el informe generado por la aplicación como resultado de la ejecución de las acciones configuradas. Como se puede observar se realizó un ataque de fuerza bruta con éxito y se extrajo la información previamente ocultada utilizando la clave "123456". A continuación se muestra el archivo extraído, el cual, como se puede observar, se trata de una imagen:

Ilustración 42: Información extraída

A.3 Operación automatizada

La función de esta operación es la de realizar un proceso de búsqueda para obtener unos resultados, posteriormente descargarse un determinado número de ellos y finalmente procesarlos en busca de posible información oculta. A continuación se especifica su modo de uso para los distintos modos de ejecución junto con capturas de ejemplo.

A.3.1 Modo por comandos

Los argumentos necesarios para ejecutar una operación automatizada son los siguientes (obtenidos de la ayuda de la aplicación):

- **Mode (-m):** Especifica el modo de ejecución: [command|xml]. Debe tomar el valor command.

- **Operation (-o):** Especifica la operación a realizar: [search|download|auto|import|delete|show|clean]. Debe tomar el valor auto.
- **Network (-n):** Especifica la red P2P a utilizar: [all|torrent]. Debe tomar el valor torrent ya que esta operación únicamente es válida en la red BitTorrent.
- **SearchQuery (-s):** Especifica la cadena de búsqueda a utilizar en una operación de búsqueda o en una operación automatizada.
- **Number (--number):** Especifica el número de resultados de búsqueda a descargar en una operación automatizada.
- **DetectorConf (--dconf):** Especifica la configuración de los detectores. El correcto modo de uso de este argumento se mostrará en el apartado **A.4 Detectores**.
- **ActionConf (--aconf):** Especifica la configuración de las acciones. El correcto modo de uso de este argumento se mostrará en el apartado **A.5 Acciones**.

Opcionalmente se pueden añadir los siguientes argumentos:

- **Size (--size):** Especifica El tamaño límite de la búsqueda en bytes utilizado para filtrar en una operación de búsqueda o en una operación automatizada.
- **Nseeds (--nseeds):** Especifica el número mínimo de seeds utilizado para filtrar en una operación de búsqueda o en una operación automatizada.
- **Hash (--hash):** Especifica el hash utilizado para filtrar en una operación de búsqueda o en una operación automatizada.
- **Sort (--sort):** Especifica el criterio de ordenación de los resultados de búsqueda a utilizar en una operación de búsqueda o en una operación automatizada: [date|name|npeers|nseeds|size]. (default: nseeds).
- **SortMode (--sortmode):** Especifica el modo de ordenación de los resultados de búsqueda a utilizar en una operación de búsqueda o en una operación automatizada: [ascendent|descendent]. (default: descendent).
- **Path (-p):** Especifica la ruta en la que se descargarán los archivos en una operación de descarga de archivo o en una operación automatizada. Si el usuario no introduce una ruta de descarga o introduce una ruta que no es válida la aplicación le pregunta si desea usar la ruta de descargas por defecto configurada o si desea introducir una nueva ruta de descargas.

A continuación se incluye un ejemplo de ejecución:

Comando de ejecución	Descripción
java -jar stegocrawler.jar -m command -o auto -n torrent -s “jackson” --number 2 –p Downloads --dconf [type:vecna] --aconf [type:bruteforce,charsets:minLetters,initLength:4]	Operación automatizada utilizando el termino de búsqueda “jackson”, eligiendo descargar 2 archivos (por defecto se eligen los 2 primeros archivos con mas seeds), configurando la ejecución de un detector del tipo detección Vecna y una acción de tipo ataque por fuerza bruta utilizando el conjunto de caracteres de las letras minúsculas con una longitud de 4 para la clave inicial

Tabla 63: Ejemplo de ejecución de una operación automatizada

A.3.2 Modo por archivos XML

A continuación se va a mostrar una representación del esquema XML utilizado para la validación. Únicamente se mostrará la parte del esquema relacionada con la operación automatizada.

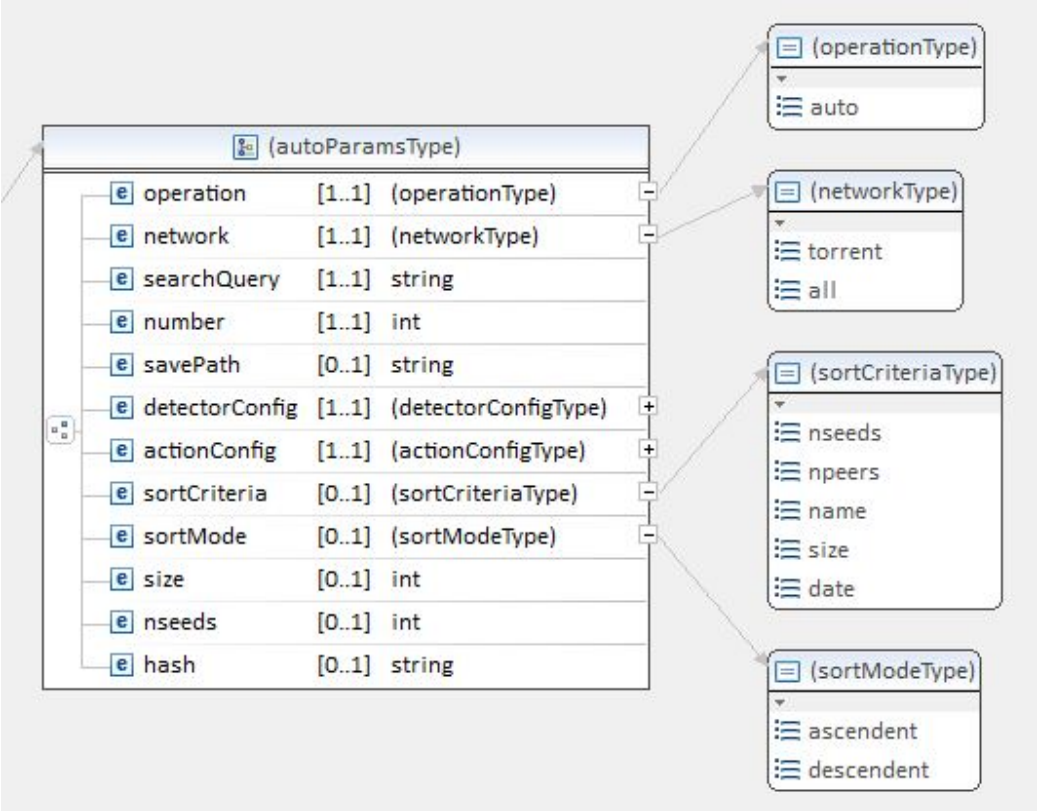


Tabla 64: Esquema XML de la operación automatizada 1

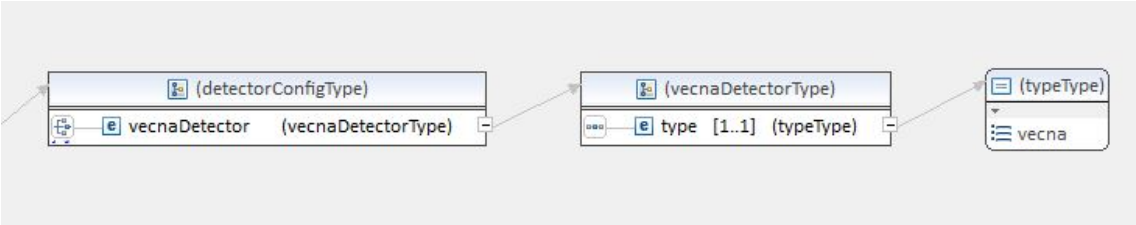


Tabla 65: Esquema XML de la operación automatizada 2

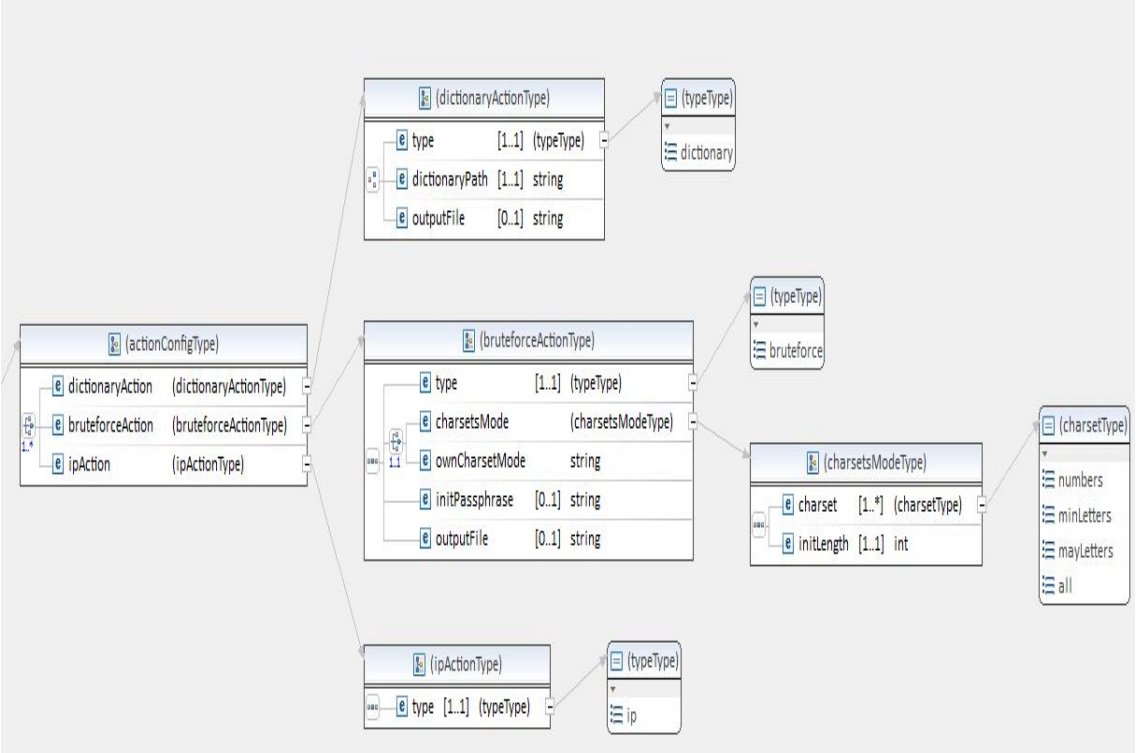


Tabla 66: Esquema XML de la operación automatizada 3

Finalmente se muestra un ejemplo de un archivo XML con los argumentos necesarios para realizar una operación automatizada.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:xmlConfig xmlns:tns="http://www.example.org/XMLConfigSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.org/XMLConfigSchema ../src/resource
s/XMLConfigSchema.xsd ">
  <tns:params>
      <tns:autoParams>
          <tns:operation>auto</tns:operation>
          <tns:network>torrent</tns:network>
          <tns:searchQuery>prueba</tns:searchQuery>
          <tns:number>5</tns:number>
          <tns:detectorConfig>
              <tns:vecnaDetector>
                  <tns:type>vecna</tns:type>
              </tns:vecnaDetector>
          </tns:detectorConfig>
          <tns:actionConfig>
              <tns:dictionaryAction>
                  <tns:type>dictionary</tns:type>
                  <tns:dictionaryPath>dictionary.txt</tns:dictionaryPath>
                  <tns:outputFile>extracted</tns:outputFile>
              </tns:dictionaryAction>
```

```
            <tns:bruteforceAction>
                <tns:type>bruteforce</tns:type>
                <tns:charsetsMode>
                    <tns:charset>minLetters</tns:charset>
                    <tns:charset>numbers</tns:charset>
                    <tns:initLength>4</tns:initLength>
                </tns:charsetsMode>
                <tns:outputFile>extracted</tns:outputFile>
            </tns:bruteforceAction>
            <tns:ipAction>
                <tns:type>ip</tns:type>
            </tns:ipAction>
            </tns:actionConfig>
            <tns:sortCriteria>name</tns:sortCriteria>
            <tns:sortMode>descendent</tns:sortMode>
            <tns:hash>7261B5B80BC56224D5403DBEF0E118DE35253078
            </tns:hash>
            <tns:size>1073741824</tns:size>
            <tns:nseeds>20</tns:nseeds>
        </tns:autoParams>
    </tns:params>
    <tns:verbose>true</tns:verbose>
</tns:xmlConfig>
```

A.3.3 Capturas

En la siguiente ilustración se puede observar la salida mostrada por la aplicación al ejecutar una automatizada.

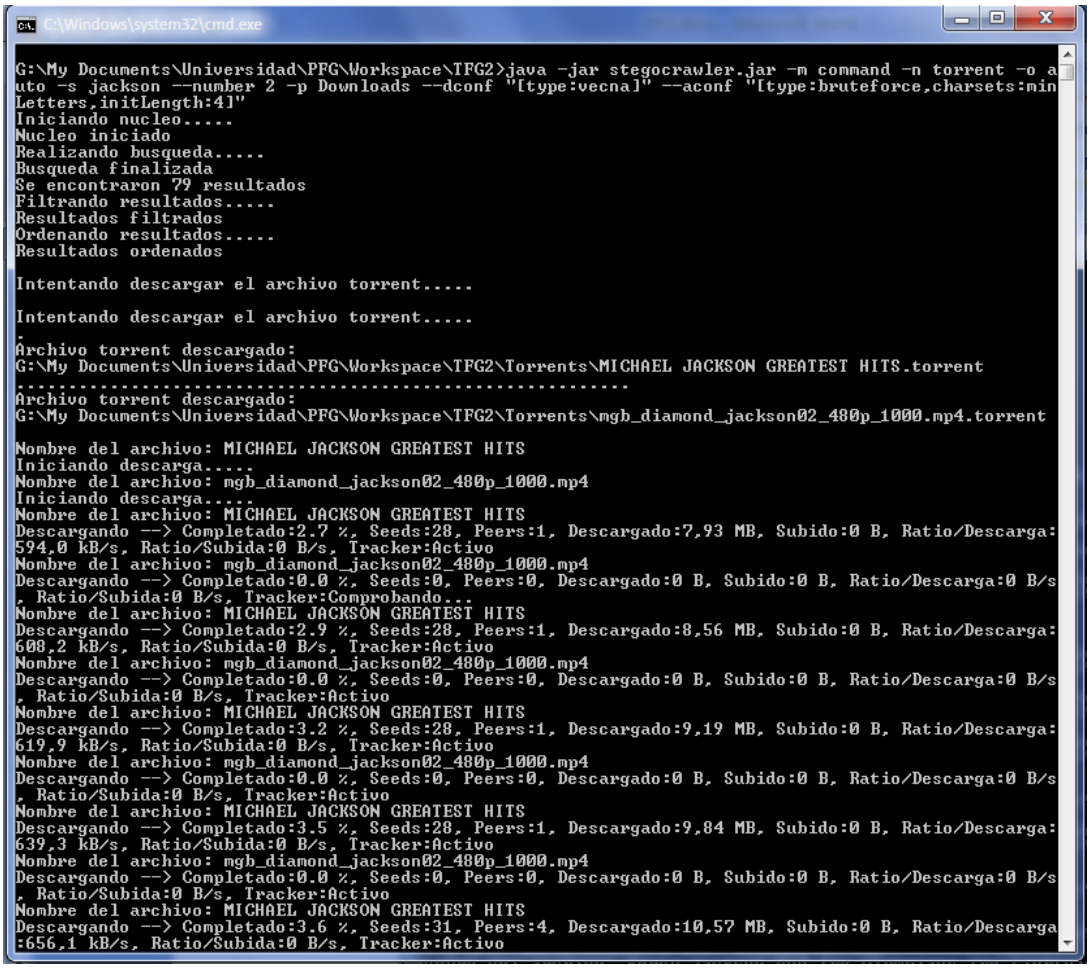


Ilustración 43: Salida mostrada por la operación automatizada

A.4 Detectores

Los detectores se configuran mediante el argumento Dconf (--dconf). Los distintos detectores que implementa la aplicación son los siguientes:

- **Detector Vecna:** El detector Vecna especializa la detección de contenido oculto en imágenes mediante la técnica del cálculo de la entropía para el programa especializado en esteganografía Vecna.

Formato
[type:vecna]
Ejemplo
--dconf [type:vecna]

Tabla 67: Modo de uso del detector Vecna

Además aunque únicamente se encuentra implementado un detector, se permite la configuración de múltiples de ellos para su ejecución Por ejemplo, si existiera un detector denominado F5, la siguiente configuración sería posible:

--dconf [type:vecna][type:f5]

A.5 Acciones

Las acciones se configuran mediante el argumento Aconf (--aconf). Las distintas acciones que implementa la aplicación son las siguientes:

- **Acción ataque por diccionario:** La acción de ataque de diccionario consiste en recorrer un fichero de texto plano especial denominado diccionario con el fin de ir probando claves con el fin de obtener aquella con la que se ha ocultado información en un archivo descargado. La siguiente tabla muestra su correcto modo de uso.

Formato	
[type:dictionary,dictionaryPath:<ruta_diccionario> [,outputFile:<ruta_archivo_salida>]]	
Campos obligatorios	**Campos opcionales**
• **Type:** especifica el tipo de acción. En este caso toma el valor dictionary. • **DictionaryPath:** especifica la ruta al archivo que se utilizará como diccionario.	• **OutputFile:** especifica la ruta al archivo donde se almacenará la información extraída en caso de que el ataque tenga éxito.
Ejemplo	
--aconf [type:dictionary,dictionaryPath:dictionary.txt]	

Tabla 68: Modo de uso de la acción ataque por diccionario

- **Acción ataque por fuerza bruta:** La acción de ataque por fuerza bruta consiste en a partir de un juego de caracteres ir generando claves con el fin de obtener aquella con la que se ha ocultado información en un archivo descargado. La siguiente tabla muestra su correcto modo de uso.

Formato	
[type:bruteforce,{(charsets:{minLetters|mayLetters|numbers|all}+,initLength:<longitud_inicial>)|(ownCharset:<charset_propio>)}[,initPassphrase:<clave_inicial>][,outputFile:<ruta_archivo_salida]]	
Campos obligatorios	**Campos opcionales**
• **Type:** especifica el tipo de acción. En este caso toma el valor bruteforce. • Se debe introducir una de las dos configuraciones:	• **InitPassphrase:** especifica la clave con la que se iniciara el ataque. • **OutputFile:** especifica la ruta al archivo donde se almacenará la información

o **Charset estándar:** consiste en usar charsets (conjuntos de caracteres) predefinidos como las letras mayúsculas, los números…Se deben especificar los siguientes campos: ▪ **Charsets:** especifica conjunto de charsets a utilizar en el ataque a elegir entre minLetters,mayLetters,numbers y all y separados por el carácter |. ▪ **InitLength:** especifica la longitud de la clave con la que se iniciara el ataque. o **Charset propio:** consiste en utilizar un charset especificado por el usuario. Se deben especificar los siguientes campos: ▪ **OwnCharset:** especifica un conjunto de caracteres personalizado para ejecutar el ataque.	extraída en caso de que el ataque tenga éxito.
Ejemplos	
--aconf [type:bruteforce,charsets:minLetters|numbers,initLength:4,initPassphrase:adaa] --aconf [type:bruteforce,ownCharset:abcdefgh123456789,outputFile:salida]	

Tabla 69: Modo de uso de la acción ataque por fuerza bruta

- **Acción obtener direcciones IP:** La acción de obtención de direcciones IP consiste en utilizar funciones del programa Vuze para obtener las direcciones IP de los pares que están compartiendo el archivo descargado contenedor de información oculta. La siguiente tabla muestra su correcto modo de uso.

Formato	
[type:ip]	
Campos obligatorios	**Campos opcionales**
• **Type:** especifica el tipo de acción. En este caso toma el valor ip.	
Ejemplo	
--aconf [type:ip]	

Tabla 70: Modo de uso de la acción obtener direcciones IP

Además se permite la configuración de múltiples acciones para su ejecución, es decir, la siguiente configuración es posible:

--aconf [type:dictionary,dictionaryPath:dictionary.txt][type:ip]

Dicha configuración indica que se debe realizar una acción de ataque por diccionario y una acción de obtener direcciones IP.

A.6 Operación de importar plantilla de metabúsqueda

La función de esta operación es importar plantillas de metabúsqueda. El modo de funcionamiento de estas plantillas ya se detallo en la sección **4.1.3.2 Plantillas de metabúsqueda**. A continuación se especifica su modo de uso para los distintos modos de ejecución junto con capturas de ejemplo.

A.6.1 Modo por comandos

Los argumentos necesarios para ejecutar una operación de importar plantilla de metabúsqueda son los siguientes:

- **Mode (-m):** Especifica el modo de ejecución: [command|xml]. Debe tomar el valor command.
- **Operation (-o):** Especifica la operación a realizar: [search|download|auto|import|delete|show|clean]. Debe tomar el valor import.
- **Network (-n):** Especifica la red P2P a utilizar: [all|torrent]. Debe tomar el valor torrent ya que esta operación únicamente es válida para dicha red.
- **Path (-p):** Especifica la ruta que localiza un template para su importación en una operación de importar plantilla de metabúsqueda.

A continuación se incluye un ejemplo de ejecución:

Comando de ejecución	Descripción
java -jar stegocrawler.jar -m command -o import -n torrent -p “PirateBay.vuze”	Operación de importar una plantilla localizada en la ruta especificada mediante el argumento “-p”.

Tabla 71: Ejemplo de ejecución de una operación de importar plantilla de metabúsqueda

A.6.2 Modo por archivos XML

A continuación se va a mostrar una representación del esquema XML utilizado para la validación. Únicamente se mostrará la parte del esquema relacionada con la operación de importar una plantilla de metabúsqueda.

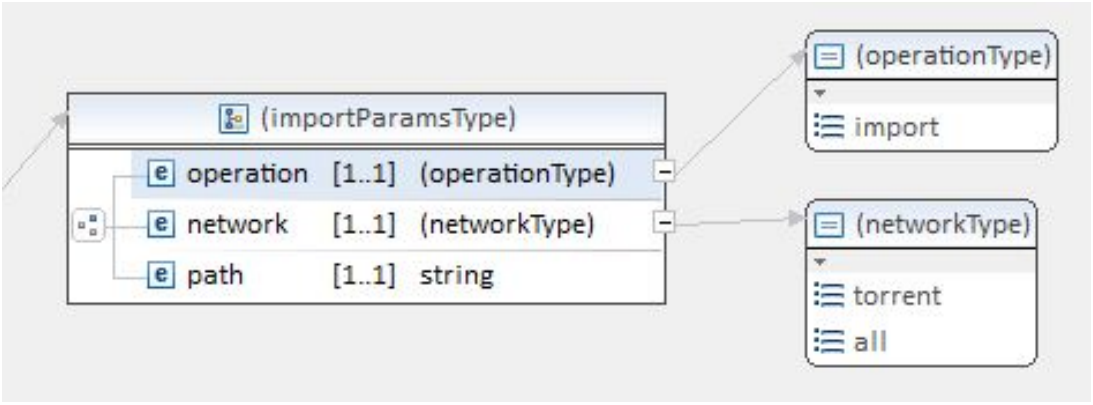


Ilustración 44: Esquema XML de la operación de importar plantilla de metabúsqueda

Finalmente se muestra un ejemplo de un archivo XML con los argumentos necesarios para realizar una operación de importar una plantilla de metabúsqueda.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:xmlConfig xmlns:tns="http://www.example.org/XMLConfigSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.org/XMLConfigSchema ../src/resource
s/XMLConfigSchema.xsd ">
  <tns:params>
      <tns:importParams>
            <tns:operation>import</tns:operation>
            <tns:network>torrent</tns:network>
            <tns:path>PirateBay.vuze</tns:path>
      </tns:importParams>
  </tns:params>
  <tns:verbose>true</tns:verbose>
</tns:xmlConfig>
```

A.6.3 Capturas

En la siguiente ilustración se puede observar la salida mostrada por la aplicación al ejecutar una operación de importar plantilla de metabúsqueda.

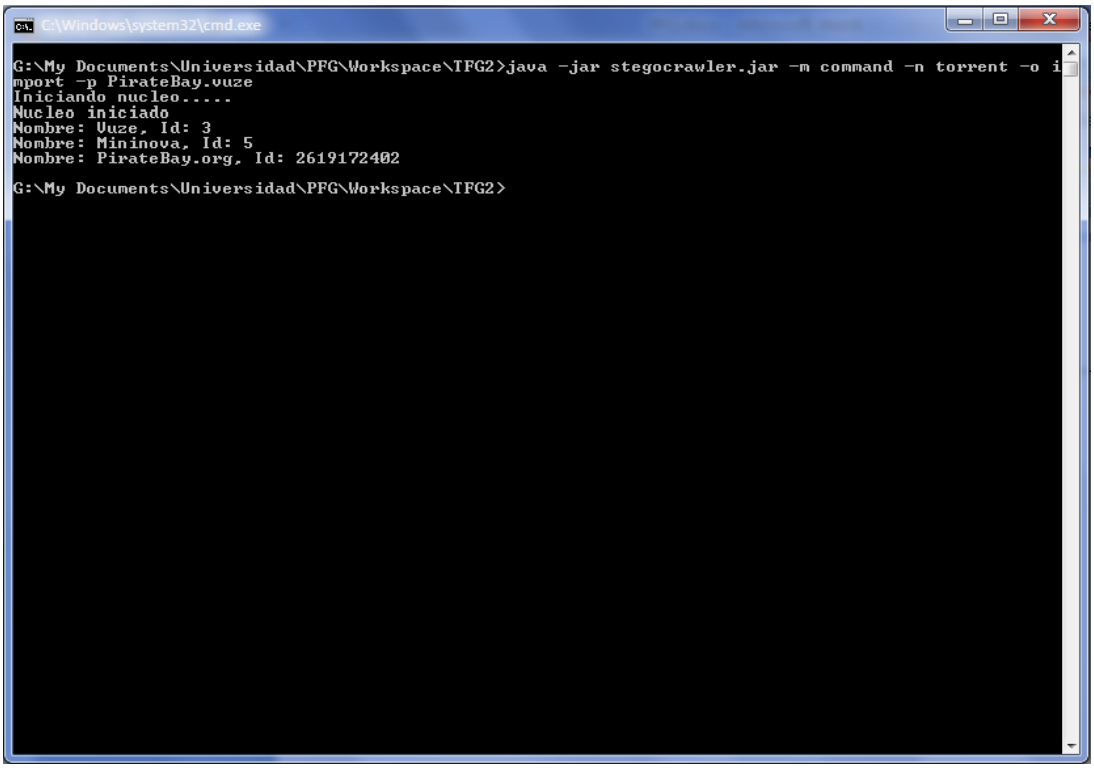


Ilustración 45: Salida mostrada por la operación de importar plantilla de metabúsqueda

A.7 Operación de borrar plantilla de metabúsqueda

La función de esta operación es eliminar plantillas de metabúsqueda importadas previamente a partir de sus identificadores (ids). Para conocer los ids de las plantillas ya importadas se debe ejecutar la operación de mostrar plantillas, explicada más adelante. A continuación se especifica su modo de uso para los distintos modos de ejecución junto con capturas de ejemplo.

A.7.1 Modo por comandos

Los argumentos necesarios para ejecutar una operación de borrar plantilla de metabúsqueda son los siguientes:

- **Mode (-m):** Especifica el modo de ejecución: [command|xml]. Debe tomar el valor command.
- **Operation (-o):** Especifica la operación a realizar: [search|download|auto|import|delete|show|clean]. Debe tomar el valor delete.

- **Network (-n):** Especifica la red P2P a utilizar: [all|torrent]. Debe tomar el valor torrent ya que esta operación únicamente es válida para dicha red.
- **Id (--id):** Especifica el id del template a borrar en una operación de borrado de template. Para conocer el id de las plantillas se debe ejecutar la operación de mostrar plantillas importadas.

A continuación se incluye un ejemplo de ejecución:

Comando de ejecución	Descripción
java -jar stegocrawler.jar -m command -o delete -n torrent --id 2619172402	Operación de eliminar una plantilla importada previamente e identificada por el id especificado mediante el argumento "—id".

Tabla 72: Ejemplo de ejecución de una operación de borrar plantilla de metabúsqueda

A.7.2 Modo por archivos XML

A continuación se va a mostrar una representación del esquema XML utilizado para la validación. Únicamente se mostrará la parte del esquema relacionada con la operación de borrar una plantilla de metabúsqueda.

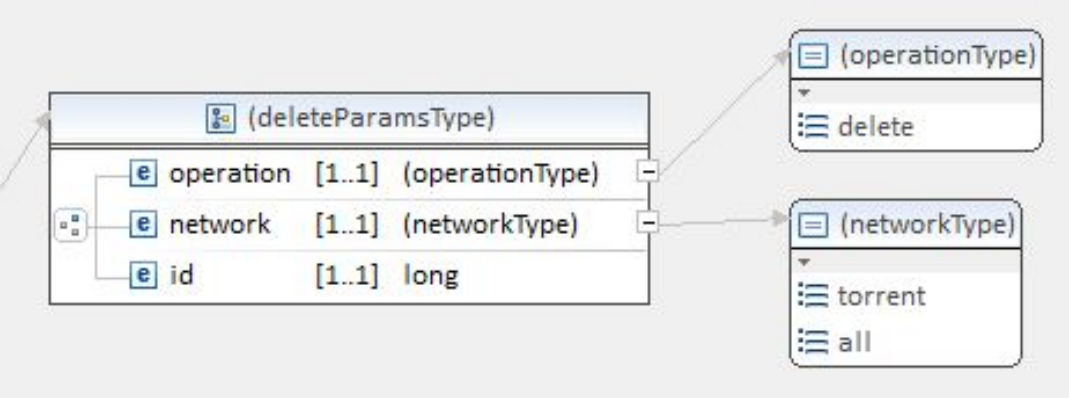


Ilustración 46: Esquema XML de la operación de borrar plantilla de metabúsqueda

Finalmente a continuación se muestra un ejemplo de un archivo XML con los argumentos necesarios para realizar una operación de borrar una plantilla de metabúsqueda.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:xmlConfig xmlns:tns="http://www.example.org/XMLConfigSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.org/XMLConfigSchema ../src/resource
s/XMLConfigSchema.xsd ">
  <tns:params>
      <tns:deleteParams>
            <tns:operation>delete</tns:operation>
            <tns:network>torrent</tns:network>
            <tns:id>2619172402</tns:id>
      </tns:deleteParams>
  </tns:params>
  <tns:verbose>true</tns:verbose>
</tns:xmlConfig>
```

A.7.3 Capturas

En la siguiente ilustración se puede observar la salida mostrada por la aplicación al ejecutar una operación de borrar plantilla de metabúsqueda.

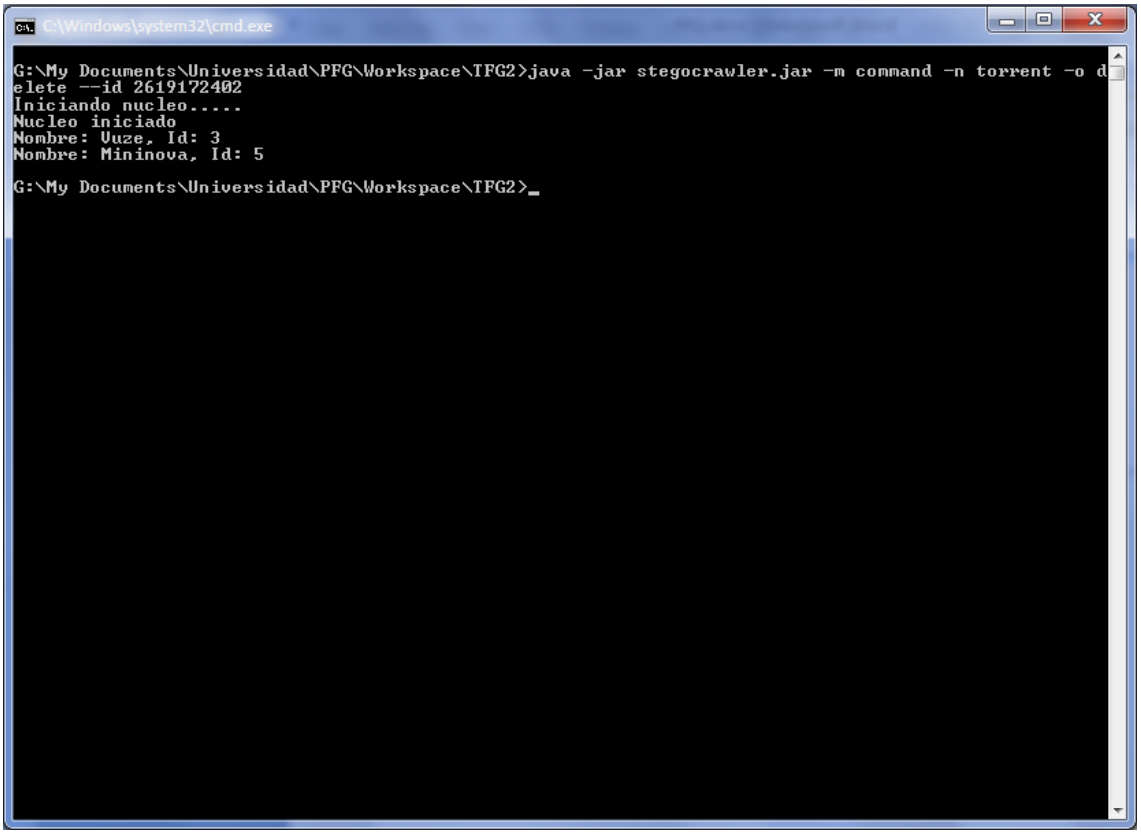


Ilustración 47: Salida mostrada por la operación de borrar plantilla de metabúsqueda

A.8 Operación de mostrar plantillas de metabúsqueda importadas

La función de esta operación es mostrar todas las plantillas que ya se han importado a la aplicación. A continuación se especifica su modo de uso para los distintos modos de ejecución junto con capturas de ejemplo.

A.8.1 Modo por comandos

Los argumentos necesarios para ejecutar una operación de mostrar plantillas de metabúsqueda importadas son los siguientes:

- **Mode (-m):** Especifica el modo de ejecución: [command|xml]. Debe tomar el valor command.
- **Operation (-o):** Especifica la operación a realizar: [search|download|auto|import|delete|show|clean]. Debe tomar el valor show.
- **Network (-n):** Especifica la red P2P a utilizar: [all|torrent]. Debe tomar el valor torrent ya que esta operación únicamente es válida para dicha red.

A continuación se incluye un ejemplo de ejecución:

Comando de ejecución	Descripción
java -jar stegocrawler.jar -m command -o show -n torrent.	Operación de mostrar plantillas de metabúsqueda

Tabla 73: Ejemplo de ejecución de una operación de mostrar plantillas de metabúsqueda

A.8.2 Modo por archivos XML

A continuación se va a mostrar una representación del esquema XML utilizado para la validación. Únicamente se mostrará la parte del esquema relacionada con la operación de mostrar las plantillas de metabúsqueda importadas.

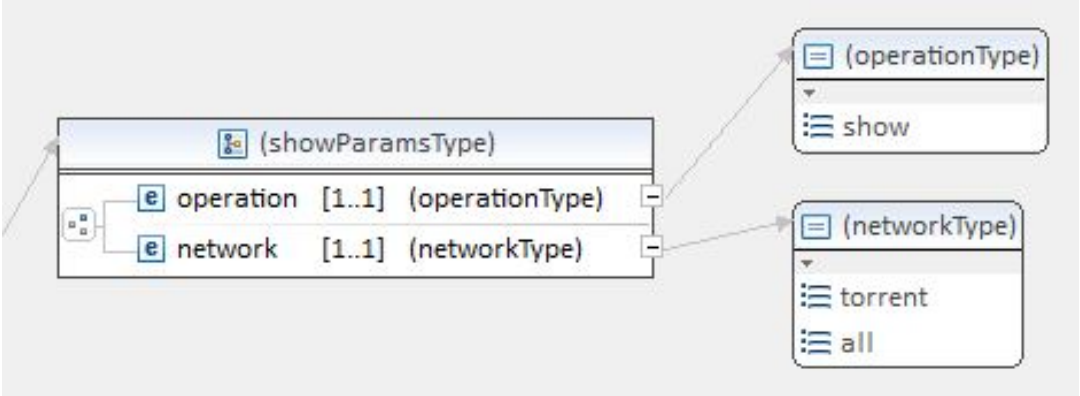


Ilustración 48: Esquema XML de la operación de mostrar plantillas de metabúsqueda importadas

Finalmente a continuación se muestra un ejemplo de un archivo XML con los argumentos necesarios para realizar una operación de mostrar las plantillas de metabúsqueda importadas.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:xmlConfig xmlns:tns="http://www.example.org/XMLConfigSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.org/XMLConfigSchema ../src/resource
s/XMLConfigSchema.xsd ">
  <tns:params>
      <tns:showParams>
            <tns:operation>show</tns:operation>
            <tns:network>torrent</tns:network>
      </tns:showParams>
  </tns:params>
  <tns:verbose>true</tns:verbose>
</tns:xmlConfig>
```

A.8.3 Capturas

En la siguiente ilustración se puede observar la salida mostrada por la aplicación al ejecutar una operación de mostrar plantillas de metabúsqueda.

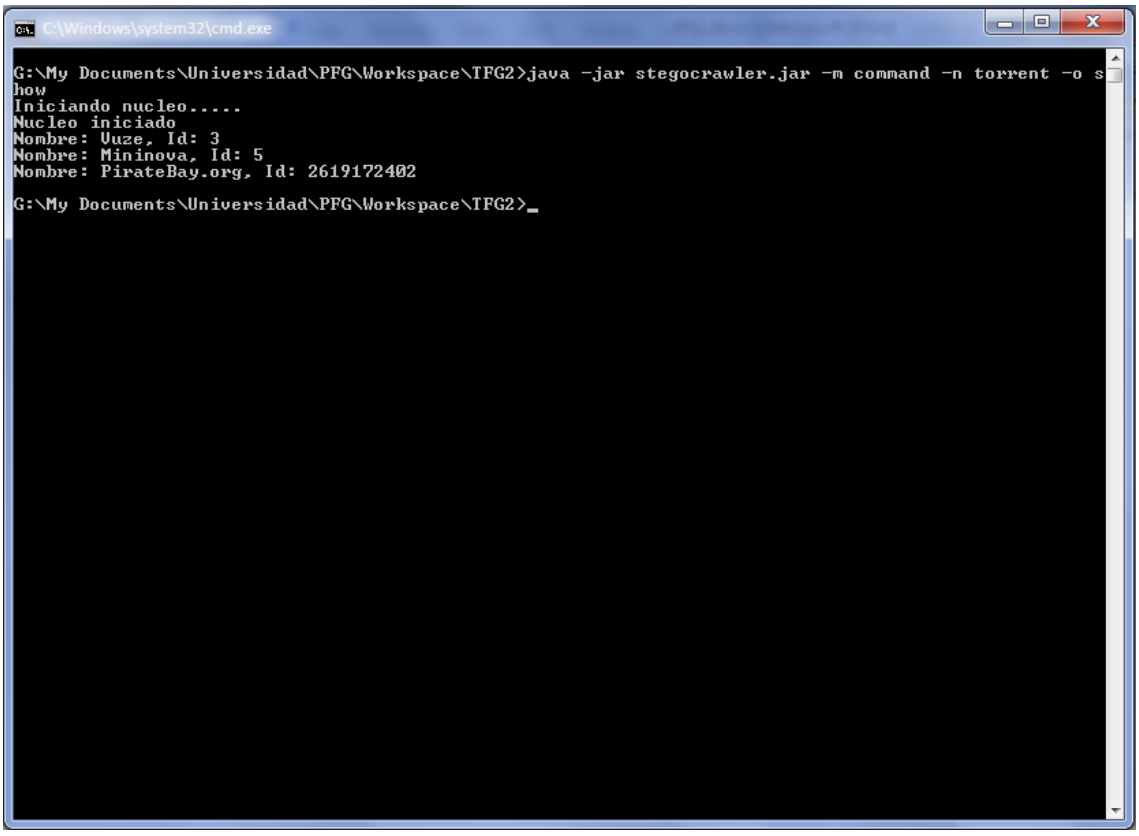


Ilustración 49: Salida mostrada por la operación de mostrar plantillas de metabúsqueda

A.9 Operación de limpieza

La función de esta operación es la de eliminar todas las descargas que se estén realizando actualmente, incluyendo cualquier porción del archivo que ya se haya descargado. A continuación se especifica su modo de uso para los distintos modos de ejecución junto con capturas de ejemplo.

A.9.1 Modo por comandos

Los argumentos necesarios para ejecutar una operación de limpieza son los siguientes:

- **Mode (-m):** Especifica el modo de ejecución: [command|xml]. Debe tomar el valor command.
- **Operation (-o):** Especifica la operación a realizar: [search|download|auto|import|delete|show|clean]. Debe tomar el valor clean.

- **Network (-n):** Especifica la red P2P a utilizar: [all|torrent]. Debe tomar el valor torrent ya que esta operación únicamente es válida en la red BitTorrent.

A continuación se incluye un ejemplo de ejecución:

Comando de ejecución	Descripción
java -jar stegocrawler.jar -m command -o clean -n torrent	Operación de limpieza

Tabla 74: Ejemplo de ejecución de una operación de limpieza

A.9.2 Modo por archivos XML

A continuación se va a mostrar una representación del esquema XML utilizado para la validación. Únicamente se mostrará la parte del esquema relacionada con la operación de limpieza.

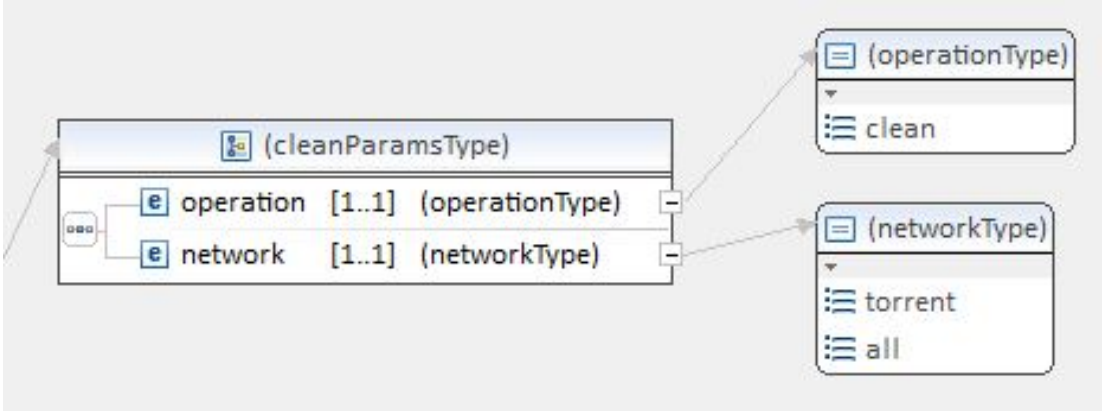


Ilustración 50: Esquema XML de la operación de limpieza

Finalmente a continuación se muestra un ejemplo de un archivo XML con los argumentos necesarios para realizar una operación de limpieza.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:xmlConfig xmlns:tns="http://www.example.org/XMLConfigSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.org/XMLConfigSchema ../src/resource
s/XMLConfigSchema.xsd ">
  <tns:params>
       <tns:cleanParams>
             <tns:operation>clean</tns:operation>
             <tns:network>torrent</tns:network>
       </tns:cleanParams>
  </tns:params>
  <tns:verbose>true</tns:verbose>
</tns:xmlConfig>
```

A.9.3 Capturas

En la siguiente ilustración se puede observar la salida mostrada por la aplicación al ejecutar una operación de limpieza.

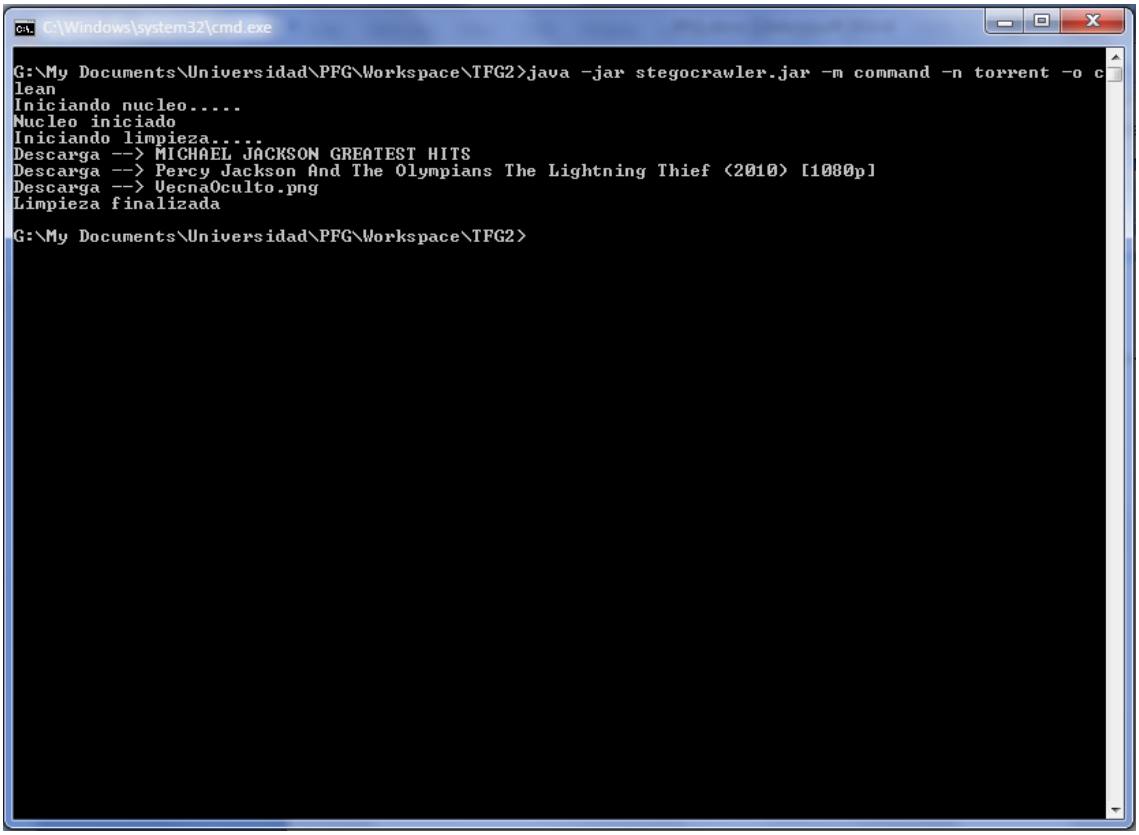


Ilustración 51: Salida mostrada por la operación de limpieza

A.10 Sistema de configuración

Tal y como se dijo en la sección **4.1.9 Sistema de configuración** la aplicación permite la configuración de distintas propiedades de la misma mediante la edición del archivo properties.xml situado en la carpeta *Settings*. A continuación se detallan las distintas propiedades que se pueden configurar.

- **Default_downloads_folder:** especifica la ruta tomada por defecto por la aplicación para almacenar las descargas de archivos.
- **Informs_folder:** especifica la ruta a utilizar por la aplicación para almacenar los informes generados.
- **Torrents_folder:** especifica la ruta a utilizar por la aplicación para almacenar los archivos torrent descargados.

Además de las propiedades anteriores la aplicación permite la configuración de propiedades propias del programa Vuze aunque es tarea del usuario conocer el nombre de la propiedad a configurar y un valor válido para ella.

Anexo B: Glosario

En este anexo se incluye una lista con definiciones de algunos términos aparecidos a lo largo del proyecto.

Ataque de denegación de servicio: Se conocen como ataques de denegación de servicio o DoS a los ataques que consisten en negar la disponibilidad de un cierto servicio a usuarios legítimos. Los ataques más empleados son los de inundación (flooding), ya sea de paquetes ICMP, de paquetes TCP/SYN, etc.

Enlace magnet: Son enlaces formados de acuerdo al esquema URI que proporciona el estándar abierto MAGNET. Estos enlaces se caracterizan por identificar archivos mediante funciones hash. Estos enlaces se están haciendo muy populares actualmente por su utilidad en los entornos P2P ya que por ejemplo les permite a las páginas que almacenan torrents en sus servidores dejar de almacenarlos y únicamente compartir enlaces magnet. Los clientes utilizando el hash contenido en el enlace pueden conectarse a sus nodos vecinos para descargar el archivo torrent, que sigue siendo necesario para la descarga del archivo (Parfeni).

Expresión regular: Una expresión regular, a menudo llamada también patrón, es una cadena de texto especial que describe un patrón de búsqueda (Regular-expressions).

Función resumen: Como cualquier función es un algoritmo que mapea un conjunto de elementos de entrada a un rango de elementos de salida más compacto. Normalmente son algoritmos de bajo coste. También se les llama funciones hash.

ISP: Un proveedor de servicios de Internet (o ISP, por sus siglas en inglés, *Internet Service Provider*) es una empresa que proporciona conexión a Internet a sus clientes.

Leecher: Un leecher, o sanguijuela en español, es un nodo que está descargando algún archivo torrent y aún no se ha completado su descarga. En el momento en el que esta descarga se completa el nodo comienza a compartir el archivo y se convierte en un seeder. El término leecher fuera del ámbito de la red BitTorrent es un término peyorativo ya que se aplica a usuarios que únicamente descargan información de internet y no suben nunca o casi nunca información para que otros puedan descargarla.

NAT: (Network Address Translation, Traducción de Dirección de Red) Básicamente permite que un único dispositivo, como un router, actuar como agente entre Internet y una red local. Esto significa que únicamente se necesita una dirección IP para representar a un grupo de ordenadores frente a cualquiera fuera de su red (Cisco, 2011).

Sandboxing: El sandboxing consiste en el aislamiento de recursos. Normalmente se usa para aislar procesos (programas en ejecución) que provienen de autores que no son de confianza para evitar posibles daños en el sistema.

Seeder: Un seeder es un nodo de la red BitTorrent que ha descargado completamente un archivo torrent o alguna parte de él y ahora lo comparte para que otros nodos lo puedan descargar. Para que un archivo pueda ser descargado completamente debe haber al menos un seeder que lo esté compartiendo.

Streaming: Se denomina streaming a la forma de contenido multimedia que puede reproducirse sin previamente haberlo descargado completamente (TechTerms).

Tabla de hash distribuida: Una tabla de hash distribuida o DHT es una estructura superpuesta que utiliza un enrutamiento basado en claves para las operaciones de poner y obtener objetos del índice. Cada par es designado para mantener una porción del índice de la DHT (Buford, y otros, 2008).

URI: (Uniform Resource Identifier) Es una secuencia de caracteres que identifica el nombre un recurso de acuerdo a un formato uniforme (TechTerms).

URL: (Uniform Resource Locator) Es una secuencia de caracteres de acuerdo a un estándar que sirve para localizar recursos en internet. Es una subclase de URI.

Bibliografía

ALI. 2012. Perfiles profesionales en informática. *Asociación de Titulados Universitarios Oficiales en Informática.* [En línea] 2012. http://www.ali.es/modules/miprofesion/visit.php?fileid=72.

ALMALASI. 2008. Riesgos en la seguridad y confidencialidad al utilizar programas P2P. *Configurarequipos.* [En línea] 31 de Octubre de 2008. http://www.configurarequipos.com/doc986.html.

Alonso González, Adrián y Barra Muñoz, Joel. 2011. *Esteganografía en imágenes.* 2011.

Ayudabittorrent. Funcionamiento del protocolo del BitTorrent. *Ayudabittorrent.* [En línea] http://www.ayudabittorrent.com/funcionamiento-protocolo-bittorrent.

Buford, John, Yu, Heather y Lua, Eng Keong. 2008. P2P Networking and Applications. *Safari Books.* [En línea] 12 de Diciembre de 2008. http://proquest.safaribooksonline.com/book/-/9780123742148.

Cisco. 2011. How NAT Works. *Web de CISCO.* [En línea] 29 de Marzo de 2011. http://www.cisco.com/en/US/tech/tk648/tk361/technologies_tech_note09186a008009 4831.shtml.

Denney, Richard. 2012. *Stickyminds.* [En línea] 2012. http://www.stickyminds.com/sitewide.asp?ObjectId=6525&Function=edetail&ObjectType=ART.

García Crespo, Ángel y López Cuadrado, Jose Luís. 2011. *Teoría de la Asignatura Dirección de Proyectos de Software.* 2011.

Heródoto. Las Historias de Heródoto de Halicarnaso. *Dominiopúblico.* [En línea] http://www.dominiopublico.es/libros/H/Herodoto/.

INTECO. 2010. Esteganografía, el arte de ocultar información. *INTECO.* [En línea] 17 de Febrero de 2010. http://www.inteco.es/Seguridad/Observatorio/Articulos/Esteganografia_el_arte_de_ocultar_informacion.

Lazalde, Alan. 2010. PeerBook: la versión P2P de Facebook que hace lo que éste no puede. *Bitelia.* [En línea] 25 de Junio de 2010. http://bitelia.com/2010/06/peerbook-la-version-p2p-de-facebook-que-hace-lo-que-este-no-puede.

November, Charlie. 2010. ¿Por qué Diaspora es mejor que Facebook? *Alt1040.* [En línea] 19 de Mayo de 2010. http://alt1040.com/2010/05/diaspora-una-nueva-esperanza.

Parfeni, Lucian. BitTorrent Magnet Links Explained. [En línea] http://news.softpedia.com/news/BitTorrent-Magnet-Links-Explained-132536.shtml.

Regular-expressions. Web acerca de las expresiones regulares. *Regular-expressions.* [En línea] http://www.regular-expressions.info/.

Shannon entropy calculator. 2012. *Shannon entropy calculator.* [En línea] 2012. http://www.shannonentropy.netmark.pl/.

Simmons, Gustavus J. 1998. Department of Mathematics and Computer Science. *North Carolina Central University.* [En línea] 1998. http://www.cs.nccu.edu.tw/~raylin/UndergraduateCourse/ComtenporaryCryptography/Spring2009/ThePrisonerProblem.pdf.

Steinmetz, Ralf. *Peer-to-peer systems and applications.*

TechTerms. Diccionario de términos informáticos. *TechTerms.* [En línea] http://www.techterms.com/.

Wayner, Peter. 2008. *Disappearing Cryptography: Information Hiding: Steganography & Watermarking, Third Edition.* s.l. : Morgan Kaufmann, 2008.

—. 2011. Sitio web de Peter Wayner. [En línea] 2011. http://www.wayner.org/texts/mimic/.

Westfeld, Andreas. 2012. Htw-dresden. [En línea] 2012. http://www2.htw-dresden.de/~westfeld/publikationen/f5.pdf.