UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

ESCOLA TÉCNICA SUPERIOR DE ENXEÑARÍA

Descubrimiento de patrones frecuentes en flujos de trabajo

Autor:
David Chapela de la Campa

Tutores:
Manuel Mucientes Molina
Manuel Lama Penín
Borja Vázquez Barreiros

Grado en Ingeniería Informática

Julio 2015

Trabajo de Fin de Grado presentado en la Escuela Técnica Superior de Ingeniería de la Universidad de Santiago de Compostela para la obtención del Grado en Ingeniería Informática

Dr. Manuel Mucientes Molina, Profesor del Departamento de Electrónica y Computación de la Universidad de Santiago de Compostela, **Dr. Manuel Lama Penín**, Profesor del Departamento de Electrónica y Computación de la Universidad de Santiago de Compostela y **D. Borja Vázquez Barreiros**, Investigador predoctoral de la Universidad de Satiango de Compostela.

INFORMAN:

Que la presente memoria, titulada *Descubrimiento de patrones frecuentes en flujos de trabajo*, presentada por **D. David Chapela de la Campa** para superar los créditos correspondientes al Trabajo de Fin de Grado de la titulación del Grado en Ingeniería Informática, se realizó bajo nuestra dirección en el Departamento de Electrónica y Computación de la Universidad de Santiago de Compostela.

Y para que así conste a los efectos oportunos, expiden el presente informe en Santiago de Compostela, a 10 de Julio de 2015:

El Tutor,	El cotutor,	El cotutor,	El alumno,
Manuel Mucientes Molina	Manuel Lama Penín	Borja Vázquez Barreiros	David Chapela de la Campa

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Prefacio

El objetivo de la **minería de datos** es la extracción de patrones, o datos de frecuente aparición, de grandes volúmenes de información que, a priori, no pueden ser interpretados por su elevado tamaño. Aplicando técnicas estadísticas y de inteligencia artificial, se consigue extraer información de gran importancia para todo tipo de entidades, actividad que resultaría inviable mediante otros métodos. La aplicación de técnicas de minería de datos a grafos se conoce como **minería de grafos**, y su objetivo es la identificación de patrones representativos, no solo analizando la frecuencia, sino también la relevancia dentro del grafo.

Actualmente el volumen de algoritmos diseñados para la minería de grafos es muy reducido, la gran mayoría de ellos se centran en *sequential pattern mining*, que consiste en la búsqueda de patrones en forma de secuencia dentro de conjuntos de datos. Además, el soporte en este campo a grafos que contengan bucles es prácticamente inexistente. Esta última característica es una de las más comunes en estas estructuras, puesto que en un escenario real la probabilidad de que una tarea, o un conjunto de ellas, se realicen varias veces en la misma ejecución es muy alta. Siempre está presente la posibilidad de 'volver atrás', tanto para realizar de nuevo una tarea como para tomar otro camino.

Por estos motivos, el presente TFG se centrará en la implementación y adaptación de un algoritmo que permita reconocer patrones frecuentes en grafos restringidos[1], dado un conjunto de trazas y un grafo al que pertenecen. Se partirá de una definición de alto nivel del algoritmo *w-find* [10], y se ampliará y modificará lo necesario para optimizar la búsqueda de patrones frecuentes en grafos restringidos, y dar soporte a flujos de trabajo con bucles.

Debido a que la búsqueda que se desea realizar comprende subestructuras de los grafos que pueden presentar bifurcaciones en el flujo, los resultados obtenidos no serán secuencias de ejecución, por lo que es necesario una representación que permita simular la ejecución de distintos caminos del grafo. Para esto, el algoritmo se basará en redes de Petri, un potente formalismo orientado a la descripción del comportamiento de la dinámica de los sistemas.

[1] Un grafo restringido es aquel que contiene reglas que relacionan tareas, de modo que la ejecución de una puede condicionar la ejecución de otra (*p.ej.* estructuras AND u OR).

1.1. Objetivos

Como se ha comentado previamente, el objetivo general de este proyecto es la implementación y adaptación de un algoritmo de minería de grafos que permita el reconocimiento de patrones frecuentes en grafos restringidos, dando soporte a bucles. Para cumplir con este fin, se han elaborado los siguientes objetivos:

1. **Implementación de un algoritmo para el reconocimiento de patrones frecuentes en grafos restringidos a partir de un fichero de registro con diversos flujos de trabajo**. Para esto se implementará, adaptará y mejorará el algoritmo w-find [10], comprobando su eficacia con el problema a resolver.
2. **Ampliación del algoritmo para dar soporte a grafos restringidos con bucles**. Se mejorará el algoritmo de partida ampliando sus funcionalidades para dar soporte a la búsqueda de patrones frecuentes en grafos restringidos que posean bucles.
3. **Interfaz web sobre una arquitectura basada en big data para la visualización de los resultados**. De esta forma se facilitará la interacción y visualización por parte de usuarios de los resultados del algoritmo.

1.2. Organización del documento

En esta memoria se documentan las diferentes etapas, fases y desarrollos que se han llevado a cabo para resolver los objetivos definidos en el TFG, explicando cada una de estas tareas y justificando las decisiones tomadas.

- El ***capítulo 2*** realiza una introducción a la minería de grafos y a la problemática que se aborda en este Trabajo Fin de Grado, explicando la base del algoritmo de partida de este proyecto.
- En el ***capítulo 3*** se explican en detalle las características del algoritmo implementado, así como las mejoras introducidas en el mismo para su ampliación.
- El ***capítulo 4*** muestra el análisis de requisitos realizado en este proyecto, así como la definición del alcance del mismo.
- El ***capítulo 5*** describe la gestión realizada a lo largo del proyecto, comprendiendo el análisis de riesgos, metodología aplicada, planificación, cálculo del coste del proyecto, gestión de la configuración y gestión de las comunicaciones.
- En el ***capítulo 6*** se propone una arquitectura de alto nivel del sistema, definiendo cada una de las partes de las que consta, y las herramientas que se han utilizado para el desarrollo del presente TFG.
- El ***capítulo 7*** contiene la implementación y diseño a bajo nivel de la arquitectura propuesta.
- En el ***capítulo 8*** se detallan las pruebas realizadas para la validación de la funcionalidad del algoritmo, de la interfaz de usuario, y de los componentes de los mismos.

- El ***capítulo 9*** recoge las conclusiones derivadas de la realización del presente proyecto, así como las posibles ampliaciones a realizar, de cara a una continuidad del mismo.
- Por último, se añaden los apéndices con el manual de instalación y el manual de uso.

Capítulo 2

Introducción

Los grafos son estructuras de datos muy potentes que permiten representar, de forma gráfica, la unión o enlace entre diferentes entidades que mantienen algún tipo de interacción. Estas estructuras fueron inventadas en el siglo XVIII, y desde entonces han sido ampliamente utilizadas.

Un tipo concreto de grafos son los grafos dirigidos, estos se caracterizan por poseer direcciones definidas en los arcos, lo que facilita el modelar flujos de ejecución en los que unos nodos deben preceder a otros. Con este tipo de estructuras se puede modelar desde las posibles secuencias de tareas efectuadas por clientes en un centro comercial, hasta el flujo de trabajo (*workflow*) propio de un proceso de negocio. En todos estos casos puede ser interesante conocer, dado un conjunto de ejecuciones, cuáles son las tareas que más se ejecutan, y en qué orden, con el fin de obtener información de interés en relación al proceso.

Algunos ejemplos de lo que se podría hacer con esta información serían *i)* rediseñar la distribución del centro comercial para mejorar las ventas, *ii)* optimizar los procesos de negocio redistribuyendo las tareas según su frecuencia de ejecución, *iii)* y hasta descubrir el comportamiento más frecuente entre los alumnos de una asignatura, para poder modificar el diseño de la unidad educativa, mejorando así los resultados.

Aquí es donde entra en juego el objetivo del presente Trabajo Fin de Grado. A partir de un registro de actividades (*log*), y un flujo de trabajo asociado a él, se obtienen los patrones frecuentes[1] ofreciendo información que a priori no está al alcance de los usuarios. Debido a la escasez de aproximaciones que realicen esta minería de grafos, este proyecto plantea la implementación y mejora del algoritmo w-find [10], que introduce la base para la búsqueda de patrones frecuentes en grafos dirigidos con restricciones.

[1] En este TFG se entenderá por frecuente aquel patrón que aparece en un porcentaje de los registros mayor que el umbral definido.

2.1. Flujos de trabajo

La Workflow Management Coalition[2] en su glosario [18], define flujo de trabajo, (*workflow* en inglés) como:

> *La automatización, completa o en parte, de un proceso de negocio durante el cual se define la secuencia de acciones, actividades o tareas utilizadas para la ejecución del proceso, incluyendo el conjunto de información intercambiada entre sus participantes.*

De esta definición podemos deducir que un flujo de trabajo explica *qué* es lo que va a suceder en un proceso, *quién* lo va a hacer y *cómo* se va a realizar. Tras la detección de un *workflow*, la extracción de los patrones que más frecuentemente ocurren puede, por ejemplo, permitir al encargado del proceso la realización de modificaciones con mucho más conocimiento sobre lo que está ocurriendo.

Un ejemplo de flujo de trabajo simple, de carácter secuencial, sería el siguiente:

1. Un profesor sube una actividad a realizar mediante una herramienta de aprendizaje.
2. Un alumno accede a la herramienta y comprueba las actividades pendientes.
3. El alumno realiza la actividad.
4. El profesor evalúa la actividad.

Los workflows que se han analizado en este TFG han sido proporcionados por el algoritmo de minería de procesos ProDiGen [16], que genera flujos de trabajo a partir de registros.

2.2. Redes de Petri

Uno de los principales problemas a la hora de implementar un algoritmo de minería de grafos, que proporciona una especificación de las operaciones a realizar desde un alto nivel, es la representación que se hará de los datos. La decisión que se tome para este cometido condicionará la dificultad de implementar algunas funcionalidades en futuras fases.

Primero se deben estudiar las necesidades del algoritmo, en este caso la representación debe ser únicamente de grafos restringidos o flujos de trabajo, que se analizarán a medida que son ejecutados. La necesidad del control, paso a paso, de la ejecución de las trazas sobre el grafo requiere una estructura que permita conocer en todo momento las tareas que están activas, y las que no lo están. Por este motivo se ha decidido utilizar las redes de Petri [9] como representación de los grafos, ya que todo grafo restringido puede ser representado con una estructura de este tipo. A continuación se introducen las características de este tipo de estructuras.

[2] www.wfmc.org

2.2.1. Elementos y funcionamiento

Una red de Petri es un grafo orientado donde se pueden distinguir tres tipos de elementos:

- Las **plazas**, representadas por círculos, se corresponden con las pre o post-condiciones de una tarea o evento. Para representar el estado (activo o inactivo) de estas condiciones se utilizarán lo que se denomina **tokens** o **marcas**, representados como puntos dentro de estas plazas.
- Las **transiciones** o **tareas**, representadas por cuadrados, se corresponden con las posibles acciones a realizar cuando se cumplen sus pre-condiciones.
- Los **arcos**, representados por flechas, siempre deben unir una plaza con una transición, y nunca dos plazas o dos tareas entre sí.

La definición formal de una red de este tipo se efectúa con una tupla $N = (P, T, F)$ donde: *i)* P es un conjunto finito de plazas, *ii)* T es un conjunto finito de transiciones y *iii)* F es el conjunto de arcos dirigidos.

En la Figura 2.1 se puede observar un ejemplo de una red de este tipo, en este caso formada por dos tareas organizadas de forma secuencial.

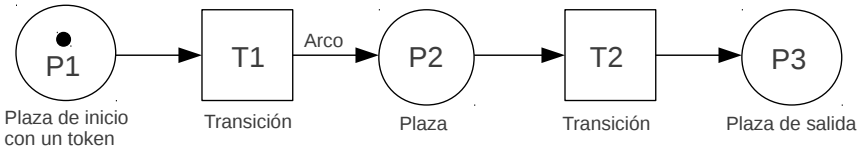

Figura 2.1: Ejemplo secuencial de una Red de Petri.

Utilizando la misma red se introducirán, a continuación, las reglas sobre la ejecución de transiciones:

I Una transición t está *activa si y sólo si* cada una de sus plazas de entrada contienen, al menos, un token. Esto es, se cumplen todas sus pre-condiciones.

II Una transición t sólo se puede *disparar* si está *activa*. En caso de dispararse, esta consumirá una marca por cada plaza de entrada que tenga, y generará una marca en cada una de sus plazas de salida, o post-condiciones.

La Figura 2.1 representa un comportamiento denominado *secuencia*, en el que una tarea debe su causalidad únicamente a otra tarea. A continuación se introducirán dos tipos de estructuras más complejas, con las que se trabajará en el proyecto, y que permiten a las redes de Petri representar flujos de proceso mucho más sofisticados.

- **Selección**: esta estructura se compone de varias ramas de ejecución, pero únicamente una de ellas va a ser ejecutada (post-condición) o ya lo ha sido (pre-condición) en relación con una transición. Esta estructura se correspondería con la operación lógica X-OR, ya que solo permite la activación de una de las ramas. Como ejemplo de esta construcción se muestra la Figura 2.2.

- **OR-split** (Figura 2.2a): en esta estructura el flujo de ejecución tomará una salida de un conjunto de ellas. La plaza activa, cuando posee un token, todas las tareas que la tienen como pre-condición, pero al dispararse una de ellas, se consume el token y se desactivan el resto de transiciones.
- **OR-join** (Figura 2.2b): para mantener la consistencia en la red, tras una selección debe haber un punto de sincronización posterior. Esta construcción se consigue con una plaza precedida de un conjunto formado por más de una transición, de forma que la activación de la plaza depende de la ejecución de únicamente una tarea de las de entre todas sus entradas.

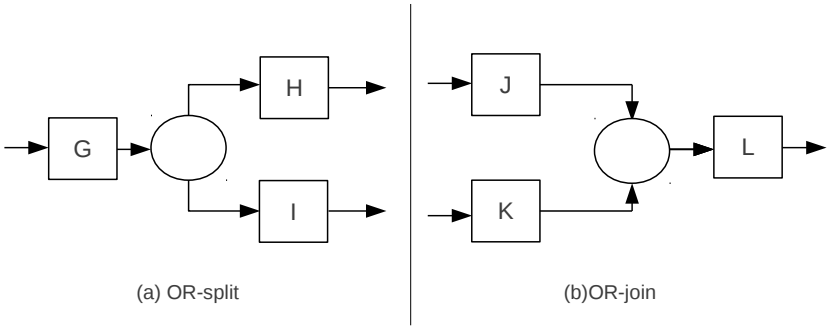

Figura 2.2: Patrón de selección (a) y sincronización (b).

- **Paralelización**: mostrada en la la Figura 2.3, representa la ejecución de varias ramas de flujo simultáneamente. Su operador lógico equivalente en el Álgebra de Boole sería el AND. A continuación se explican sus dos variantes:
  - **AND-split** (Figura 2.3a): se corresponde con una transición t con más de una plaza de salida. Cuando la transición se dispara, se genera una marca en cada plaza de salida.
  - **AND-join** (Figura 2.3b): al igual que en el caso de la selección, para mantener la consistencia en la red es preciso que tras una palarelización haya una sincronización. Este tipo de estructuras están formadas por una transición t que posee varias plazas de entrada.

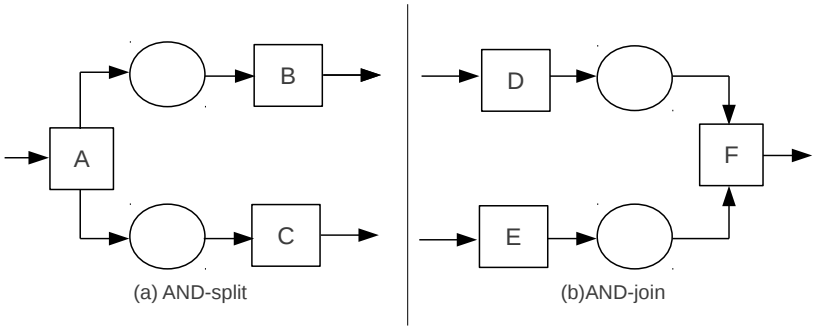

Figura 2.3: Patrón de paralelización (a) y sincronización (b).

Las redes de Petri manejadas en este proyecto para representar los grafos tendrán una única plaza de inicio, y una única plaza de fin. En cambio, es necesario que los patrones que se van

generando, y que se devolverán como resultado, tengan la posibilidad de comenzar y finalizar en diferentes plazas, aumentando así la complejidad del análisis de los mismos.

2.3. Minería de grafos

El objetivo de la minería de grafos es la extracción automática de patrones (subgrafos) pertenecientes a un grafo mayor, que tienen un peso o importancia elevados dentro de la estructura a la que pertenecen. La importancia de un patrón con respecto al grafo al que pertenece puede ser interpretada de diversas formas. Una vertiente de la minería de grafos analiza las apariciones de estos patrones dentro de un grafo, con el objetivo de detectar aquellos que más se repiten, y poder simplificar así la vista del grafo final sustituyendo estos patrones por estructuras más simples.

Otra variante considera el peso de un patrón como la frecuencia con la que este ha sido ejecutado, es decir, una aproximación del número de veces que esa estructura aparece en un conjunto de trazas (*log*) de ejecución. El presente TFG se centrará en esta última interpretación, obteniendo así los patrones pertenecientes a un grafo que superen un umbral de ejecuciones. Para ello, es necesario contar con un modelo de *workflow* y un registro, o *log*, asociado a él.

Se asume que estos registros permiten almacenar las actividades correspondientes a una ejecución de tal modo que:

I Cada evento hace referencia a una actividad (un paso definido en el proceso).

II Cada evento se corresponde con un caso (una instancia del proceso).

III Se almacenan de forma secuencial, independientemente de la existencia de procesos paralelos.

Los registros con los que se trabajará pueden contener otra información como rol, fecha de ejecución, etc. pero en este proyecto esta información adicional es irrelevante, por lo que se omitirá en los análisis realizados. Una vez definido un método que extraiga resultados útiles, se podría contemplar la posibilidad de ampliar la búsqueda considerando otros datos a mayores.

En la Tabla 2.1 se muestran un ejemplo de un log con 5 ejecuciones que pertenece al workflow que se modela en la Figura 2.4. Se trata de un caso sencillo con dos posibles secuencias, que servirá para explicar la base de la búsqueda de patrones frecuentes seguida por este TFG.

Identificador del registro	Secuencia de tareas
R1	A - B - D - E
R2	A - C - E
R3	A - B - D - E
R4	A - B - D - E
R5	A - C - E

Tabla 2.1: Registro de eventos.

Como se trata de un ejemplo simple, las secuencias del registro pueden ser simplificadas en dos, una de ellas con los registros *R1*, *R3* y *R4*, y la otra con *R2* y *R5*.

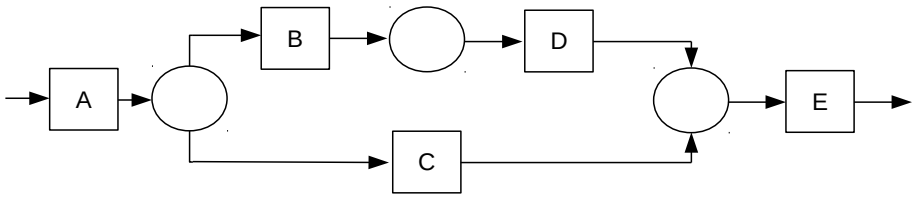

Figura 2.4: Red de Petri con una selección.

La información que se puede extraer de este registro es que la ejecución del patrón representado en la Figura 2.5a, tiene una frecuencia sobre los registros de un 60 %. Por otro lado, el patrón de la Figura 2.5b presenta una frecuencia del 40 %.

Con esta información se puede deducir que, tras un análisis con un umbral definido en el intervalo (40-60] daría como resultado el patrón de la Figura 2.5a, ya que es el máximo que supera el umbral. Otro ejemplo de resultado podría ser el de los patrones formados únicamente por una tarea que resultarían en el caso en el que el umbral superase el 60. El resultado en este caso sería un patrón formado por A, y otro formado por E, ya que la frecuencia de esos es del 100 %.

Por último, cabe resaltar que, en el caso en el que el umbral fuese inferior o igual a 40, el resultado sería el grafo completo, ya que todas sus posibles variantes superan el 40 % de frecuencia.

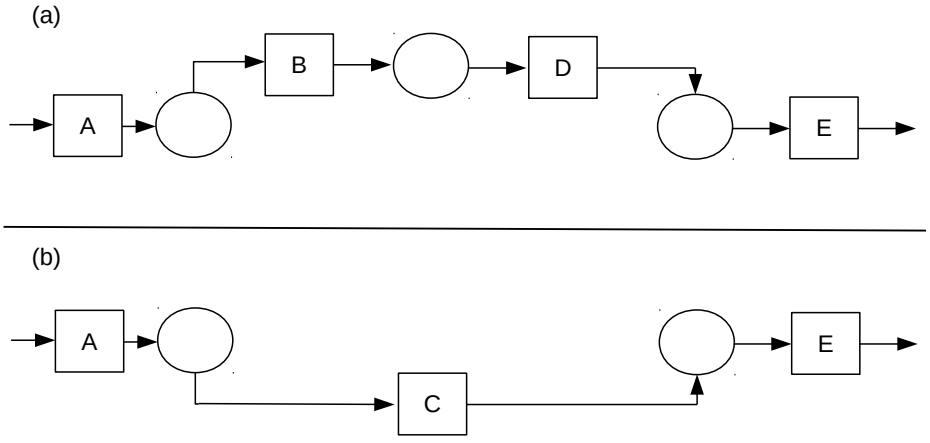

Figura 2.5: Resultado de una búsqueda en una red de petri con una selección.

Con este ejemplo se pretende introducir las bases y la finalidad del algoritmo que se planea desarrollar en este proyecto, con objetivos de búsqueda en grafos y registros de mayor tamaño y complejidad, haciendo imposible su análisis por medios convencionales.

2.3.1. Aproximaciones actuales

Tal y como se menciona en la publicación de Gianluigi Greco [10], en los últimos años se han realizado algunas publicaciones relacionadas con la minería de grafos. Como por ejemplo, estudios que aplican búsquedas voraces para encontrar patrones de un grafo [4, 14]. También han sido propuestas aproximaciones que aplican métodos de proyección o de búsqueda level-wise para conseguir el conjunto completo de patrones [1, 5].

A pesar de esto, y haciendo referencia a la misma publicación:

> *In principle, many of the above approaches could be used to mine constrained graphs. However, the adaptation of the above mentioned methods to workflow mining is a challenging task, and it results unpractical from both the expressiveness and the efficiency viewpoint [...]*

La aplicación de los métodos mencionados a la minería de procesos es complicada, además de resultar impracticable desde el punto de vista de eficiencia y expresividad. Por este motivo se ha decidido escoger el algoritmo w-find para realizar, no solo una implementación del mismo, sino una mejora y ampliación para dar soporte a procesos con bucles.

Capítulo 3

Algoritmo de minería de grafos

En este capítulo se describirá el algoritmo de minería de grafos implementado en este Trabajo Fin de Grado, especificando el pseudocódigo necesario, y explicando las mejoras realizadas sobre el algoritmo del que se ha partido.

A continuación se describirán: *(i)* La representación interna utilizada en este proyecto, *(ii)* el algoritmo de partida (w-find), *(iii)* Las funciones a mayores que se han tenido que implementar, pero que no están especificadas en el algoritmo w-find y *(iv)* las mejoras realizadas sobre el algoritmo w-find.

3.1. Representación interna

Como ya se ha explicado en la introducción de la memoria, la estructura elegida para representar los grafos a analizar son las redes de Petri convencionales.

Una decisión importante es qué representación interna utilizar para almacenar y modificar la red de Petri en el algoritmo, ya que una mala elección puede suponer un incremento en la dificultad de la implementación. En [16] se propone un algoritmo para generar flujos de trabajo a partir de grafos, y se ha observado la eficiencia que proporciona una representación mediante *matriz causal*. Por ello, se ha decidido utilizar esta estructura de forma interna.

A continuación se explicará en qué consiste una matriz causal, así como un ejemplo de uso, que facilitará la comprensión por parte del lector.

En la Tabla 3.1 se muestra una matriz causal que expresa, desde el punto de vista de la dependencia entre tareas, el mismo comportamiento que la red de Petri de la Figura 3.1 que representa.

En una representación con matriz causal, se pueden determinar las dependencias entre tareas a través de las funciones *Entrada(Tarea)* y *Salida(Tarea)* (*I(t)* y *O(t)* respectivamente de aquí en adelante). El resultado de estas funciones está formado por un conjunto de subconjuntos, que contienen las diferentes tareas del modelo, pudiendo establecer las siguientes reglas a partir de ellas:

Tarea	**I(Tarea)**	**O(Tarea)**
A	{}	{{B},{C,D}}
B	{A}	{E}
C	{A}	{E}
D	{A}	{E}
E	{{B},{C,D}}	{}

Tabla 3.1: Matriz causal equivalente a la red de Petri de la Figura 3.1.

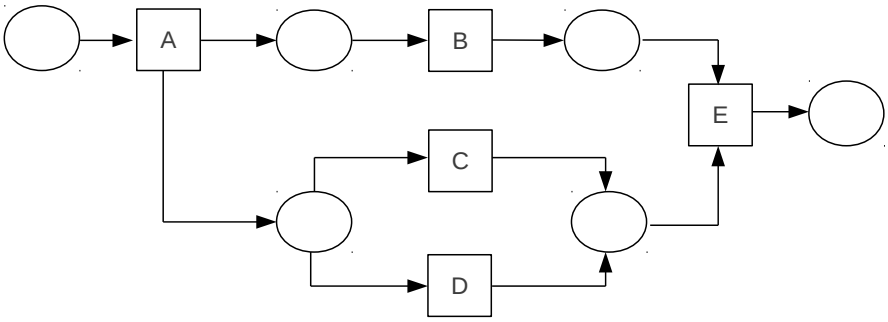

Figura 3.1: Red de Petri original.

- Tareas del **mismo subconjunto**:
  - de la función **I(t)** se corresponden con una **relación OR-join**
  - de la función **O(t)** se corresponden con una **relación OR-split**
- **Relaciones entre diferentes subconjuntos**:
  - de la función **I(t)** se corresponden con una **relación AND-join**
  - de la función **O(t)** se corresponden con una **relación AND-split**

A pesar de no encontrarse en este ejemplo, puede darse el caso de la aparición de una misma tarea en diferentes subconjuntos dentro del mismo conjunto, es decir, la intersección entre subconjuntos del mismo conjunto no tiene por qué ser el conjunto vacío. En cambio, una tarea no puede aparecer más de una vez en el mismo subconjunto.

Tomando como ejemplo la tarea A de la red de Petri de la Figura 3.1, la matriz causal indica que *O(A) = {{B}, {C, D}}*. Esto significa que, tras la activación de *A*, se realiza una construcción *AND-split* que activa las entradas *A* de las tareas *B* y *C* o *D* (*OR-split*). Como las dependencias *I(t)* de las tareas *B*, *C* y *D* están formadas únicamente por *A*, todas las tareas pasarían a estar activas.

3.2. Algoritmo w-find

Como ya se ha mencionado en secciones previas, el algoritmo desarrollado en el presente TFG parte de la especificación de un algoritmo, llamado w-find [10], siendo los objetivos de este proyecto la implementación y mejora del mismo.

w-find es un algoritmo de búsqueda de patrones frecuentes en grafos restringidos basado en los fundamentos del algoritmo Apriori[1]. La estructura principal del w-find se muestra en la Figura 3.2, donde se pueden apreciar las diferentes fases del mismo, que serán comentadas en los siguientes apartados.

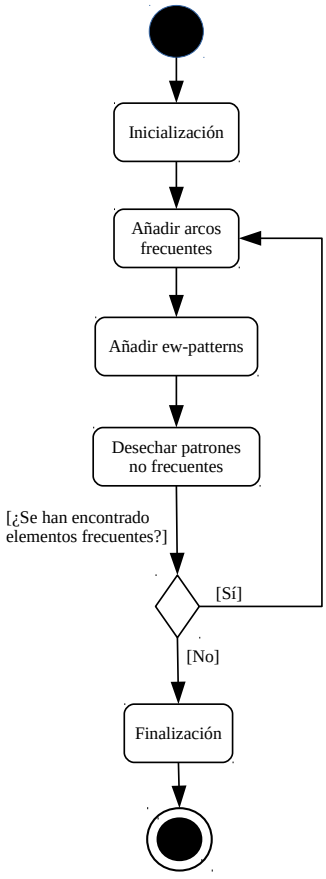

Figura 3.2: Diagrama de flujo de datos del algoritmo w-find.

En [10] se introducen conceptos necesarios para la comprensión del algoritmo diseñado en este TFG, que serán explicados a continuación, y se especifica el pseudocódigo base para el desarrollo del mismo.

[1]El algoritmo apriori se basa en el conocimiento previo, o a priori, de los conjuntos frecuentes, y permite reducir el espacio de búsqueda evitando la exploración por caminos que se sabe que no van a resultar fructíferos.

3.2.1. Conceptos básicos

En este apartado se explicarán los conceptos y pseudocódigo del algoritmo w-find, necesarios para la comprensión del funcionamiento del mismo y de las modificaciones realizadas.

El primer concepto que se introduce es el de *weak pattern* o *w-pattern*, el cual se define como todo subgrafo que satisface las restricciones del grafo al que pertenece. Es decir, se considerará *w-pattern* de G todo subgrafo de G que no rompa ninguna de las restricciones que este posee. En la Figura 3.3 se muestra una red de Petri de la que se han generado dos subgrafos (Figura 3.4). El caso (a) es un w-pattern válido, ya que satisface las restricciones de los nodos que contiene. En cambio, el ejemplo de (b) no lo es, ya que viola dos de las restricciones, una es el XOR de la tarea A, que solo permite realizar una elección entre B y C, y la otra es la restricción AND-split impuesta en C, que en este caso solo cuenta con E como salida, cuando deberían aparecer E y F.

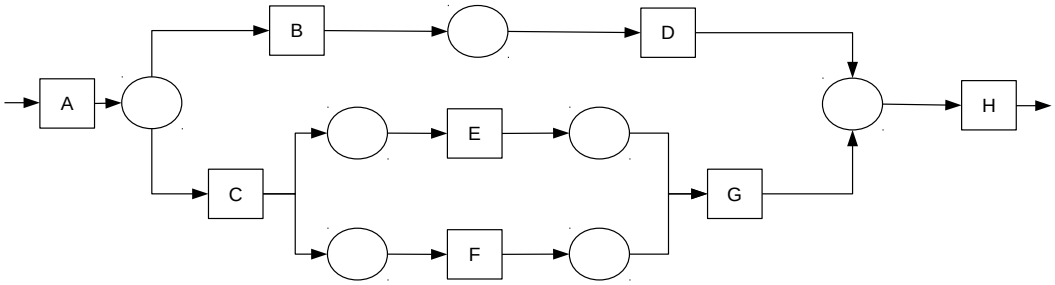

Figura 3.3: Red de petri con selección y paralelización.

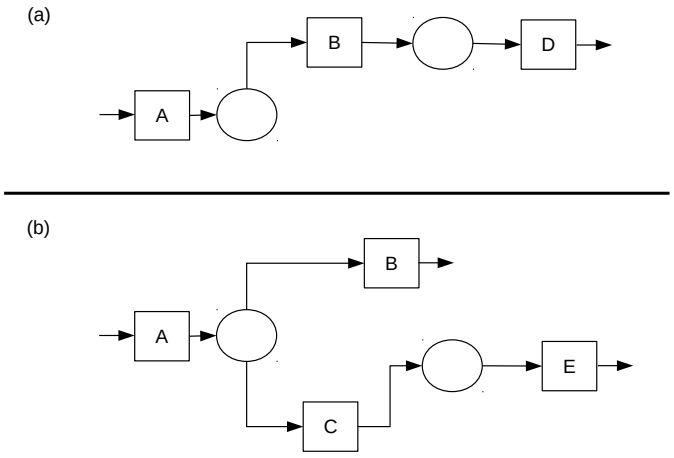

Figura 3.4: Ejemplos de w-patterns de la Figura 3.3.

Los w-pattern serán los patrones que generará el algoritmo, haciéndolos crecer por medio de otro tipo de patrones que se definen a continuación.

Estos patrones se denominan *elementary weak patterns* o *ew-pattern*, y se corresponden con los w-pattern mínimos del grafo. Es decir, para cada nodo, hallando su cierre determinista se obtendrá el *ew*-pattern correspondiente. Estos patrones son la base del algoritmo, ya que se parte de ellos en un inicio. Para aclarar el concepto de *ew*-pattern se proporcionan tres ejemplos en la Figura 3.5;

estos tres subgrafos son patrones pertenecientes al grafo de la Figura 3.3. En este caso, (a) y (b) son dos *ew*-patterns válidos, mientras que (c) no lo es. De hecho, (a) y (b) se corresponden con los cierres deterministas de los nodos A y C respectivamente. Mientras que (c) no se corresponde con el cierre determinista ni de F ni de G.

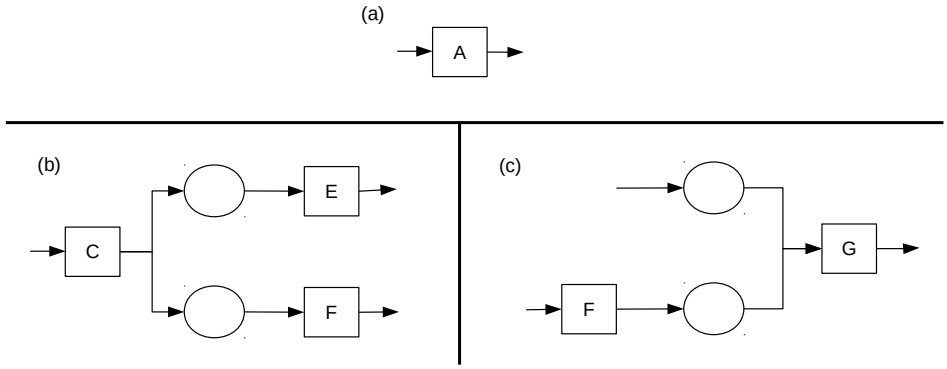

Figura 3.5: Ejemplos de *ew*-patterns de la Figura 3.3.

El valor devuelto por este algoritmo es el de todos los patrones, indistintamente de su tamaño, que superen el umbral especificado de frecuencia, esto es, que aparezcan en un porcentaje de registros mayor o igual al umbral especificado. Por ejemplo, *w-find(0.7)* devuelve el conjunto de patrones que aparezcan en el 70 % o más de los registros del log.

Para esto, el algoritmo aplica una aproximación apriori, que asume que, si un patrón X no es frecuente, los patrones que lo contengan tampoco lo serán. Bajo esta asunción, el algoritmo parte de los *ew*-patterns frecuentes del grafo, y los va expandiendo, iteración tras iteración, uniéndolos entre sí o añadiéndoles arcos del grafo frecuentes.

Un concepto que no se ha explicado todavía es el de los arcos frecuentes, ya que a pesar de que dado su nombre se puede entender que está formado por los arcos que son frecuentes, en realidad se excluyen aquellos cuyo origen es un *fork determinista*. Es decir, se tendrán en cuenta como arcos frecuentes aquellos que superen el umbral de frecuencia y cuyo origen no sea un *AND-split*.

Como se ha mencionado anteriormente, el algoritmo expande, iteración a iteración, el conjunto de patrones frecuentes. Concretamente, el algoritmo parte de los *ew*-patterns frecuentes del grafo, y los expande generando nuevos patrones, en la siguiente iteración serán estos patrones los que se sometan a la expansión, y así sucesivamente, hasta que el conjunto se queda vacío. Esta expansión mencionada se realiza de dos formas: *i)* insertando arcos frecuentes y *ii)* añadiendo *ew*-patterns al patrón.

3.2.2. Pseudocódigo del algoritmo

Una vez explicados los conceptos básicos del algoritmo, para complementar la explicación de alto nivel dada en los párrafos anteriores, se mostrará y explicará el pseudocódigo proporcionado en [10].

En la Figura 3.6 se muestra el pseudocódigo de la estructura base del algoritmo. La entrada del algoritmo consta de un workflow o grafo *WS* y un conjunto de instancias del grafo, *F*. Como

salida se devuelven los *F-patterns* (patrones frecuentes) resultantes.

En las primeras cuatro líneas se inicializan las variables, L_0 se inicializa con todos los *ew*-patterns frecuentes con respecto a las instancias de *F*, *FrequentArcs* contendrá todos los arcos cuyo origen no es un AND-split y que sean frecuentes con respecto a *F*. Por último, *E* contendrá la intersección entre los arcos cuyo origen no es un AND-split y los arcos almacenados en *frequentArcs*[2].

Input: A workflow Graph $\mathcal{WS}$, a set $\mathcal{F} = \{I_1, \ldots, I_N\}$ of instances of $\mathcal{WS}$.
Output: A set of frequent $\mathcal{F}$-patterns.
Method: Perform the following steps:

```
1   L_0 := {e|e ∈ EW, e is frequent w.r.t. F};
2   k := 0, R := L_0;
3   FrequentArcs := {(a,b)|(a,b) ∈ E^⊆, ⟨{a,b},{(a,b)}⟩ is frequent w.r.t. F};
4   E^⊆_f := E^⊆ ∩ FrequentArcs;
5   repeat
6     U := ∅;
7     forall p ∈ L_k do begin
8       U := U ∪ addFrequentArc(p);
9       forall e ∈ Compl(EW_p) ∩ L_0 do
10        U := U ∪ addFrequentEWPattern(p, e);
11    end
12    L_{k+1} := {p|p ∈ U, p is frequent w.r.t. F};
13    R := R ∪ L_{k+1};
14  until L_{k+1} = ∅;
15  return R;
```

Figura 3.6: Pseudocódigo principal del algoritmo *w*-find (Fuente [10]).

En las siguientes líneas se repite la parte iterativa del algoritmo, hasta que el conjunto generado esté vacío. En primer lugar, para cada patrón del conjunto, generará los patrones resultantes de añadirle los arcos frecuentes. A continuación, para cada *ew*-pattern que no esté contenido en el patrón, y que pertenezca a los frecuentes, generará los patrones resultantes de su unión con el patrón actual. Por último almacenará en el conjunto de la siguiente iteración los patrones generados que sean frecuentes, y los añadirá al conjunto final.

En la Figura 3.7 se especifica el pseudocódigo de algunas funcionalidades utilizadas en la estructura base. La función *addFrequentEWPattern* recibe dos patrones como argumentos, en la primera línea realiza la unión de los dos, mezclando las actividades y los arcos, y comprueba si el resultado es conexo. Si esto es así, significa que el patrón resultado es válido, por lo que se devuelve. En caso contrario, se realiza una llamada a la función *addFrequentConnection*.

En *addFrequentConnection* se parte de tres patrones, uno contiene un patrón frecuente, otro un *ew-pattern* frecuente, y el otro la mezcla de los dos, que no ha resultado ser conexa. En esta función se recorren los arcos frecuentes que no pertenecen al patrón que se está explotando, pero que tienen el origen en uno de ellos, y el destino en el otro, es decir, que une los dos patrones. Si el resultado de añadir este arco satisface las restricciones del grafo se añade al conjunto de resultados, que se devolverá al finalizar la función.

Por último, la función *addFrequentArc* recibe un patrón, y devuelve el conjunto de patrones

[2]Dado que frequentArcs es un subconjunto de los arcos cuyo origen no es un AND-split, realizar la intersección con este conjunto generará los arcos de frequentArcs. Esto se explicará en posteriores apartados, junto a otros errores encontrados en el algoritmo *w*-find.

Function *addFrequentEWPattern*$(p = \langle A_p, E_p\rangle, e = \langle A_e, E_e\rangle)$: **w-pattern**;
$p' := \langle A_p \cup A_e, E_p \cup E_e\rangle$;
if p' is *connected*, **then return** p' **else return** *addFrequentConnection*(p', p, e);

Function *addFrequentConnection*$(p' = \langle A_{p'}, E_{p'}\rangle, p = \langle A_p, E_p\rangle, e = \langle A_e, E_e\rangle)$: **w-pattern**;
$S := \emptyset$
forall frequent $(a, b) \in E_f^{\subseteq} - E_p$ s.t. $(a \in A_p,\ b \in A_e) \vee (a \in A_e,\ b \in A_p)$ **do begin**
$q := \langle A_{p'}, E_{p'} \cup (a, b)\rangle$;
if $\mathcal{WS} \models q$ **then** $S := S \cup \{q\}$;
end
return S

Function *addFrequentArc*$(p = \langle A_p, E_p\rangle)$: **pattern**;
$S := \emptyset$
forall frequent $(a, b) \in E_f^{\subseteq} - E_p$ s.t. $a \in A_p, b \in A_p$ **do begin**
$p' := \langle A_p, E_p \cup (a, b)\rangle$
if $\mathcal{WS} \models p'$ **then** $S := S \cup \{p'\}$;
end
return S

Figura 3.7: Pseudocódigo adicional del algoritmo *w*-find (Fuente [10]).

resultantes después de añadir, uno a uno, los arcos frecuentes que tienen origen y destino en el patrón, pero que no están contenidos en este.

Esta es la especificación del algoritmo *w*-find que se obtiene de la publicación [10], y de la que se parte en el presente TFG. A continuación se realizará una explicación de las funcionalidades necesarias para este algoritmo que se han tenido que implementar a mayores, por falta de especificación, y las mejoras realizadas para dar soporte a bucles, y eliminar operaciones innecesarias.

3.2.3. Funcionalidades Implementadas

En esta sección se explicarán e introducirán las implementaciones realizadas para el funcionamiento del algoritmo, y que no están especificadas en la publicación [10] que se ha tomado como referencia en este Trabajo Fin de Grado.

En el apartado anterior se ha introducido el algoritmo base, y como se puede apreciar especifica, de una forma muy superficial, el funcionamiento del algoritmo nombrando funcionalidades como la obtención de los arcos y *ew*-patterns frecuentes pero sin proporcionar el pseudocódigo. En esta sección se explicarán en profundidad estas funcionalidades que se han tenido que implementar y se introducirán las mejoras que se han hecho sobre las mismas.

3.2.3.1. Búsqueda de *ew*-patterns frecuentes

Antes de mostrar el pseudocódigo correspondiente a la implementación de esta funcionalidad se realizará una breve introducción. Posteriormente se explicará el funcionamiento del algoritmo creado con ejemplos gráficos, para facilitar la comprensión del mismo. El objetivo de este algoritmo es generar los *ew*-patterns correspondientes al grafo, que superen el umbral de frecuencia establecido, es decir, que sean frecuentes. Como resumen del funcionamiento se podría decir que el algoritmo realiza una búsqueda en la que para cada nodo del grafo, crea un patrón con él y

expande la explotación cuando es necesario para construir los w-patterns mínimos que contienen al nodo correspondiente.

La explicación del algoritmo en este aparado cuenta con las modificaciones realizadas en este TFG que amplían la definición de *ew*-pattern, aumentando la eficiencia y eficacia del algoritmo, como se explica en el último apartado de este capítulo. En los Algoritmos 1 y 2 se muestra el pseudocódigo correspondiente al algoritmo ***getEwPatterns***, que genera los *elementary weak patterns* de un grafo.

Algorithm 1: *getEwPatterns*: obtiene los *ew*-patterns frecuentes de un grafo.

```
   Input: Un conjunto de nodos N[] con las entradas y salidas correspondientes, un conjunto de
          registros R[] correspondientes al grafo a minar.
   Output: El conjunto de ew-patterns del grafo.
 1 Function getEwpatterns(N, R)
 2     patterns ← ∅
 3     forall the n ∈ N[] do
 4         pattern ← empty pattern
 5         add n to pattern
 6         exploitConnections(true, pattern, n)
 7         exploitConnections(false, pattern, n)
 8         candPatterns ← ∅
 9         if pattern has or connections then
10             add pattern to candPattern
11         else
12             add combinations of pattern to candPatterns
13         forall the candPattern ∈ candPatterns do
14             if (candPattern ∉ candPatterns) AND (candPattern is frequent w.r.t. R[]) then
15                 add candPattern a patterns
16     return patterns
17
18 Function exploitConnections(explInputs, pattern, node)
19     if explInputs then
20         set ← inputs of node
21     else
22         set ← outputs of node
23     if set is an AND then
24         forall the subset ∈ set do
25             if subset.size = 1 then
26                 connection ← only element in subset
27                 addArcAndContinue(explInputs, pattern, node, connection)
28             else
29                 forall the connection ∈ subset do
30                     if the arc is frequent w.r.t.R[] then
31                         addArcAndContinue(explInputs, pattern, node, connection)
32     else if set is an OR then
33         if set.size = 1 then
34             connection ← only element in set
35             if !isAndWithOr(connection) then
36                 addArcAndContinue(explInputs, pattern, node, connection)
```

Algorithm 2: Continuación del código del Algoritmo 1.

```
1  Function addArcAndContinue(explInputs, pattern, node, connection)
2      if explInputs then
3          arc ← arc from connection to node
4      else
5          arc ← arc from node to connection
6      if node hasn't been exploited yet then
7          set node as exploited
8          exploitConnections(true, pattern, connection)
9          exploitConnections(false, pattern, connection)
10
11 Function isAndWithOr(node)
12     isAndWithOr ← false
13     fifo ← node
14     exploited ← ∅
15     while fifo ≠ ∅ AND !isAndWithOr do
16         n ← first of fifo
17         add n to exploited
18         if n.inputs in an AND then
19             forall the subset ∈ n.inputs do
20                 if subset.size > 1 then
21                     isAndWithOr ← true
22                 else
23                     connection ← only element in subset
24                     if connection ∉ exploited then
25                         add connection to fifo
26         if n.outputs in an AND then
27             forall the subset ∈ n.outputs do
28                 if subset.size > 1 then
29                     isAndWithOr ← true
30                 else
31                     connection ← only element in subset
32                     if connection ∉ exploited then
33                         add connection to fifo
34     return isAndWithOr
```

Se debe destacar que, debido a la representación que se hace de los grafos restringidos en el algoritmo *w*-find, no se han contemplado un tipo de estructuras que presentan un problema a la hora de seleccionar los *ew*-patterns de un grafo. Este tipo de estructuras se reflejan en la Figura 3.8, y están formadas por una estructura AND, que posee una elección OR en alguna de las ramas.

Cuando las conexiones de un patrón forman una estructura AND (línea 29 del Algoritmo 1), se deben recorrer las diferentes ramas, añadiendo las conexiones sin elección, y analizando la frecuencia de los arcos pertenecientes a una rama con una elección OR. Esto haría que para el caso de la Figura 3.5, cuando se analizasen las salidas de la tarea A, se añadiesen B y E, pero se comprobase la frecuencia de los arcos $A \rightarrow C$ y $A \rightarrow D$, añadiendo solo las conexiones que resulten frecuentes (línea 30 del Algoritmo 1). En el caso en el que ambos superen el umbral de frecuencia se generarían las combinaciones del patrón (línea 12 del Algoritmo 1), obteniendo así dos *ew*-patterns, uno con cada arco.

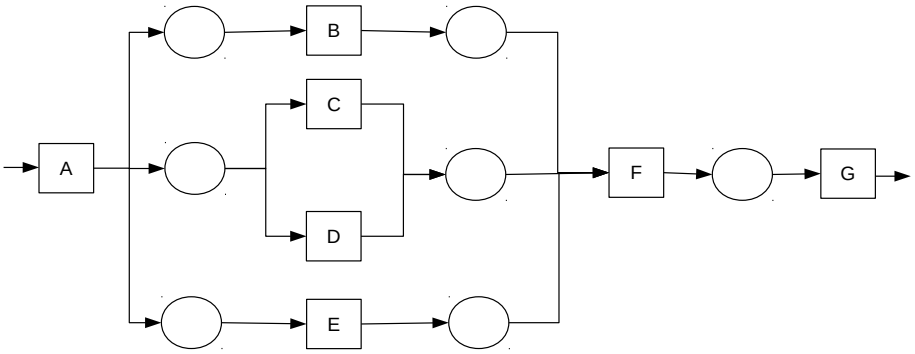

Figura 3.8: Estructura AND con OR en una rama.

Se deben contemplar también los casos en los que las entradas o salidas de un nodo están formadas únicamente por un OR. Si es así, y este solo tiene una elección, no se diferencia de una estructura AND que posea una sola rama, ya que esta siempre se ejecutará. Por este motivo se ha decidido ampliar el concepto de *ew*-pattern incluyendo las conexiones en las que solo hay una (línea 33 del Algoritmo 1).

Para una mejor comprensión se hace referencia a la Figura 3.9, de la que se extraerán tres ejemplos de *ew*-patterns que pasarán por las condiciones recién explicadas. Si partimos del nodo A, es decir, generamos su *ew*-pattern, nos encontraremos con B y C como selección en las salidas. En este caso la expansión se pararía, quedando únicamente como *ew*-pattern el de la Figura 3.10a. En el caso del nodo B, como sus entradas están formadas por una sola decisión, estas se añaden, pero no es el caso de sus salidas, generando así el *ew*-pattern de la Figura 3.10c.

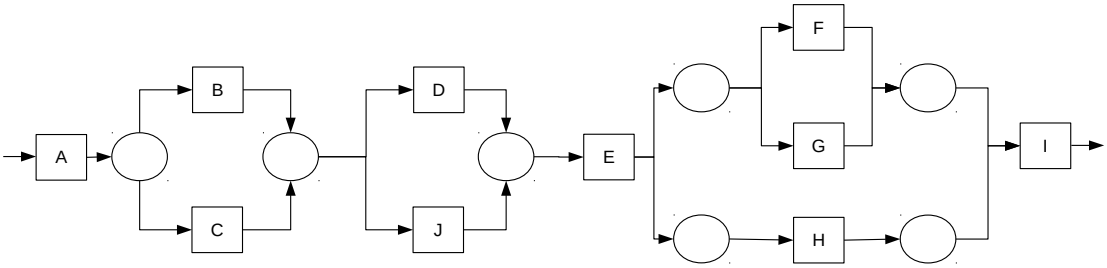

Figura 3.9: Red de petri para ejemplificar la elección de *ew*-patterns.

Por último, y para que se comprenda la necesidad de comprobar que la conexión no es un nodo AND con una rama en la que aparece un OR (línea 35 del Algoritmo 1), se efectuará el análisis partiendo del nodo D. En este caso, al analizar sus entradas se encuentra un OR-join, por lo que no se añade ninguna conexión y se continúa analizando las salidas. En este paso se debe evaluar si la conexión que se va a añadir es un AND con un OR en alguna de las ramas. Como esto es así, no se añade el nodo E, generando el *ew*-pattern de la Figura 3.10b.

La necesidad de la comprobación introducida en el párrafo anterior reside en que, si se añade el *ew*-pattern correspondiente a E se están recortando los resultados. Para ejemplificar lo explicado se propone el siguiente caso:

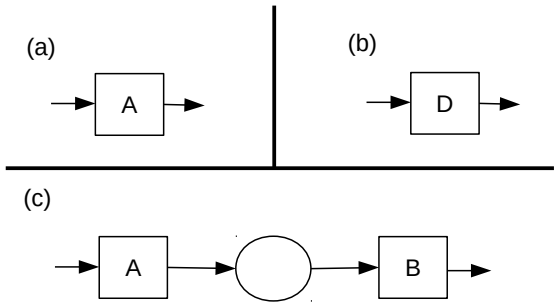

Figura 3.10: Ejemplos de *ew*-patterns correspondientes a la Figura 3.9.

- Suponiendo que la ejecución de B es en un 80 % de los registros, y que D se ejecuta en el 100 % de estos, se tiene una frecuencia de $B \rightarrow D$ del 80 %.
- Si suponemos que $E \rightarrow F$ y $E \rightarrow G$ se ejecutan cada uno el 50 % de las trazas en las que se pasó por B, obtenemos que la frecuencia del patrón que contiene $B - D - E - F$ y el que contiene $B - D - E - G$ tienen una frecuencia del 40 %.
- Suponiendo un umbral el 41 %.

Si permitimos que el *ew*-pattern de D incluya las combinaciones del patrón de E, estamos generando los *ew*-patterns correspondientes a la Figura 3.11b, a través de los cuales nunca se podrá llegar al patrón formado por $B \rightarrow D$, ya que la mezcla de cualquiera de estos patrones elementales contiene a B y alguna de F o G, proporcionando un umbral por debajo del establecido.

Si, en cambio, se deja como patrón elemental de D el propio D, se obtendrán los *ew*-patterns de la Figura 3.11a, a partir de los cuales sí se podrá generar el patrón $B \rightarrow D$ en una iteración simple.

Como se puede apreciar en la línea 33 del Algoritmo 2, se realiza una propagación en el análisis para comprobar que no se trata de una estructura AND con un OR en alguna rama. Esto es debido a que, en un caso como el de la Figura 3.12, si analizamos el nodo A, tenemos que saber que todos los componentes de sus conexiones en AND tampoco son un AND con una selección en alguna rama. En este caso, la inclusión de A implicaría la inclusión del resto de nodos de la estructura, ya que cada vez que se ejecuta A, dispara la ejecución de B y de C, esta última las de D y E, y previo a la ejecución de E se debe haber disparado F. En el caso de que alguno de estos nodos, en una rama de la estructura AND que forman, tuviesen una selección, se debería cancelar la explotación de A.

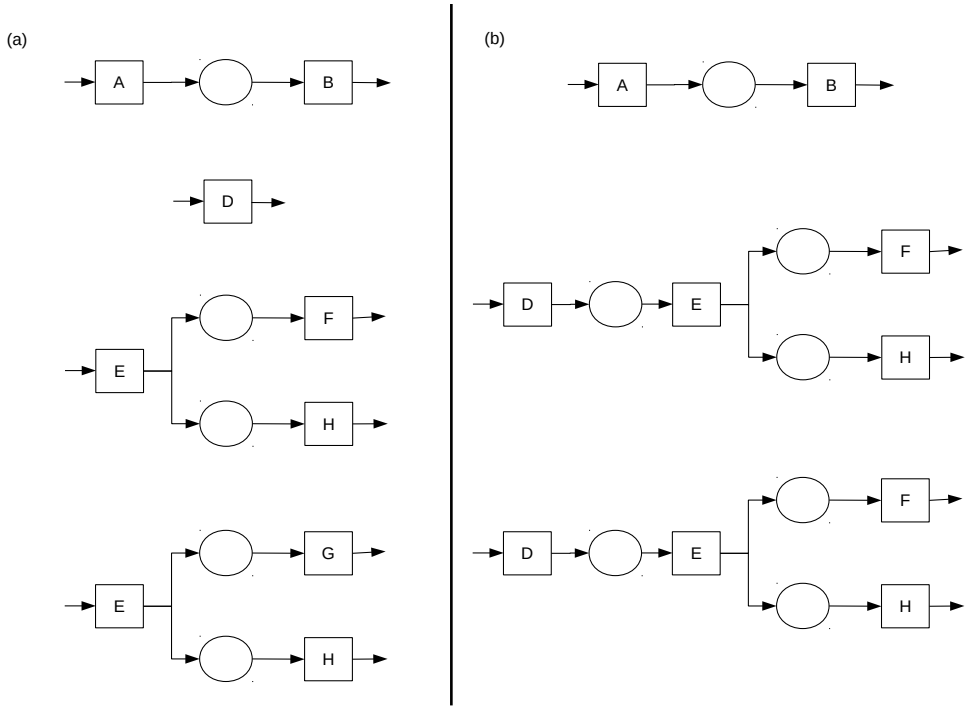

Figura 3.11: Comparación de *ew*-patterns correspondientes a la Figura 3.9

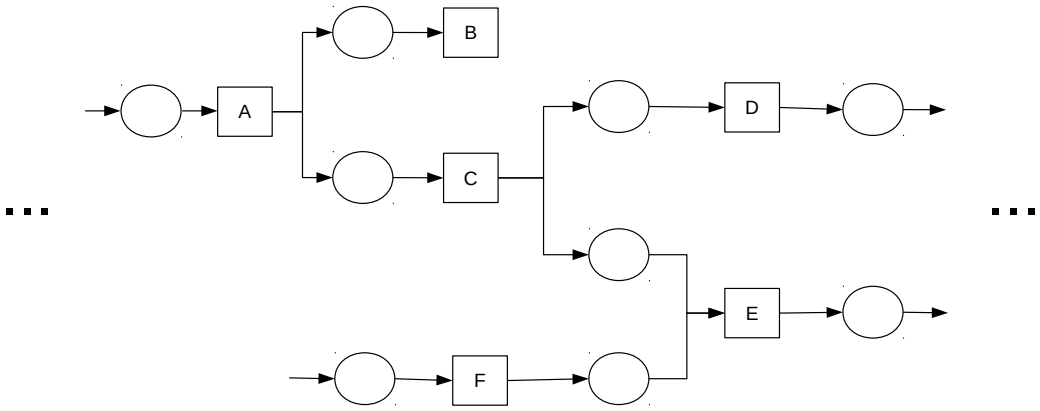

Figura 3.12: Ejemplo para el corte de la expansión en la búsqueda de *ew*-patterns

3.2.3.2. Busqueda de arcos frecuentes

En la línea 14 del pseudocódigo del algoritmo ***getEwPatterns***, se hace mención a los arcos que son frecuentes con respecto a R, esto es, los arcos que son ejecutados en un porcentaje de elementos de R mayor o igual al umbral. Este algoritmo parte de un log compuesto por ejecuciones del grafo, representadas mediante una secuencia de tareas. Debido a este formato, saber qué arco se está ejecutando en casos en los que el flujo recorre varios caminos a la vez es una tarea complicada.

Para lograr este cometido se ha utilizado la clase *CMMarking* [12] para simular la ejecución de los registros, e identificar así cada arco a medida que se ejecutan las tareas.

El procedimiento seguido por el algoritmo diseñado consiste en recorrer los registros de ejecución, averiguando las tareas que disparan la ejecución de la actual, y almacenando, sólo una vez por registro, los arcos ejecutados. En la ampliación que se realizará para dar soporte a bucles, cabe la posibilidad de que un arco se ejecute más de una vez en un mismo registro, por lo que se ha decidido contar únicamente una aparición por cada uno de ellos. Esto se debe a que se quiere registrar la frecuencia con respecto a las ejecuciones realizadas, y que no se distorsione el resultado debido a que algún usuario realice una tarea un número muy elevado de veces, dado que esto no refleja la visión global del comportamiento de todos los usuarios.

A continuación se muestra el pseudocódigo desarrollado para el cálculo de la frecuencia de los arcos de un grafo, dado un conjunto de registros de los que se ha extraído el mismo.

Algorithm 3: ***measureFrequencyArcs***: Medición de la frecuencia de los arcos de un grafo restringido.

Input: un conjunto de registros $R[]$ correspondientes al grafo a minar.
Output: una matriz $A[][]$ que almacena las frecuencias de los arcos.

```
1  Function measureFrequencyArcs(R)
2      A ← ∅
3      forall the r ∈ R do
4          executedArcs ← ∅
5          forall the t ∈ r do
6              execute task t
7              combination ← get combinationExecuted
8              if combination ∉ executedArcs then
9                  add combination to executedArcs
10                 forall the t_o ∈ combination do
11                     A[t_o][t] ++
12     return A
```

En la línea 5 del algoritmo ***measureFrequencyArcs*** se recorren las tareas de cada registro, ejecutándolas y obteniendo la combinación de las tareas origen y la tarea disparada. Si esta combinación no se ha dado en el registro que se está analizando (línea 8), se aumentan los valores de las posiciones de los arcos en la matriz de frecuencias (línea 11).

3.2.3.3. Medición de la frecuencia de un patrón con bucles

La funcionalidad de mayor complejidad es la encargada de calcular la frecuencia de un patrón, y comprobar así si es frecuente o no. Debido a que en la descripción del algoritmo w-find no se

proporciona ninguna aproximación de este método, y dado que se informa explícitamente de que dicho algoritmo no soporta bucles, en este TFG se ha desarrollado directamente una forma que evalúa si un patrón es frecuente o no, dando soporte a grafos y patrones con bucles.

La dificultad que presenta esta funcionalidad por el hecho de soportar grafos con bucles reside en la posibilidad de que una ejecución '*entre en el patrón*' en cualquier punto del mismo, acto seguido '*salga*' volviendo a entrar luego, o ejecutándolo completamente después de dar error en diversas ocasiones.

Las primeras ideas que se contemplaron pretendían analizar la frecuencia de un patrón completo, aunque este tuviera diferentes caminos o combinaciones. Una de ellas se basó en el registro de los arcos que se ejecutaban, comprobando si estos se contenían en el patrón, y marcándolos cuando lo hicieran. Pero en la etapa de pruebas todas presentaron errores con diferentes tipos de grafos.

Finalmente, se ha logrado diseñar el algoritmo ***isFrequentPattern*** que es capaz de calcular la frecuencia de un patrón dado un conjunto de registros y comprobar si este es frecuente. Para el funcionamiento de este algoritmo, que se explicará a continuación, ha sido necesaria la creación de dos algoritmos a mayores, uno que genera las combinaciones de un patrón (***getPatternCombinations***), y otro que detecta los bucles en un grafo (***searchStartEndLoopArcs***). Ambos métodos serán comentados en sus respectivos apartados, pero primero se describirá el funcionamiento del algoritmo que calcula la frecuencia de un patrón, realizando una abstracción sobre la implementación de estas dos funcionalidades.

La elección de generar las combinaciones y analizar la frecuencia de las mismas ha sido propiciada por el estudio de un patrón como el de la Figura 3.13, en esta se puede observar un grafo con un bucle en una de las ramas de una estructura AND, que tiene una salida y dos posibles entradas. En la figura se resalta en negro el patrón a analizar, y con un tono más claro el resto del grafo.

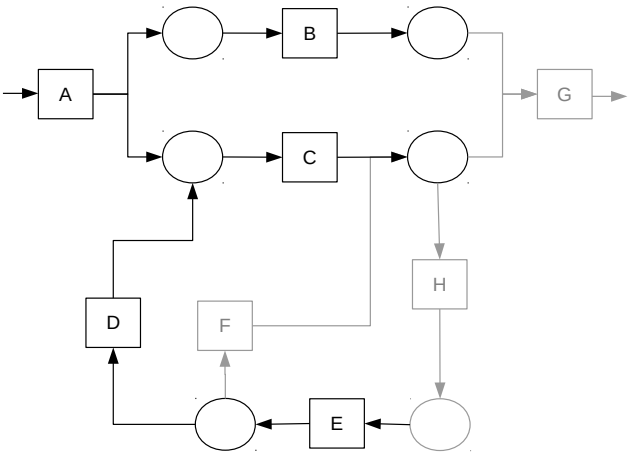

Figura 3.13: Patrón resaltado en una red de Petri con bucle.

Como ejemplo para mejorar la comprensión de esta decisión se realizará una simulación con una aproximación simple, como las que se tuvieron en cuenta en un inicio. En estas se marcan los arcos que se ejecutan, considerando el patrón como ejecutado cuando todos sus arcos han sido marcados, y en caso de error se reinicia el marcado.

Con un análisis del patrón y grafo pertenecientes a la Figura 3.13 se podría dar el caso en el las

Arcos marcados	**Arco a ejecutar**	**Acción**
—	$A \rightarrow C$	Pertenece al patrón, por lo que se acepta y continúa.
$A \rightarrow C$	$C \rightarrow H$	El origen es un nodo final, por lo que se continúa.
$A \rightarrow C$	$H \rightarrow E$	El destino es un nodo inicial, por lo que se continúa.
$A \rightarrow C$	$E \rightarrow F$	El origen se ha ejecutado y el arco no pertenece al patrón, por lo que se reinicia.
—	$F \rightarrow H$	No se hace nada.
—	$H \rightarrow E$	El destino es un nodo inicial, por lo que se continúa.
—	$E \rightarrow D$	Pertenece al patrón, por lo que se acepta y continúa.
$E \rightarrow D$	$D \rightarrow C$	Pertenece al patrón, por lo que se acepta y continúa.
$E \rightarrow D$ $D \rightarrow C$	$A \rightarrow B$	Pertenece al patrón, por lo que se acepta y continúa.
$E \rightarrow D$ $D \rightarrow C$ $A \rightarrow B$	$B \rightarrow G$ $C \rightarrow G$	Los orígenes son nodos finales, por lo que se continúa.
$E \rightarrow D$ $D \rightarrow C$ $A \rightarrow B$	—	Finalmente se comparan los arcos ejecutados y los arcos del patrón. Como el patrón contiene un arco que no ha sido ejecutado ($A \rightarrow C$) no se cumple.

Tabla 3.2: Simulación fallida de la comprobación de la frecuencia de un patrón.

dos combinaciones por las que está formado el patrón se ejecuten, pero en un orden que provoca que si no se realiza el análisis de forma individual para cada combinación el resultado sea erróneo. Este suceso se muestra en la Tabla 3.2.

En este caso, la existencia de un bucle crea la posibilidad de que, mientras se ha ejecutado parcialmente una estructura AND, se registre un error por el camino del bucle en una de ellas, que provoca el reinicio del marcado, perdiendo la ejecución que se había registrado hasta el momento de la estructura AND.

Si en el ejemplo explicado se analizase la aparición de cada una de las combinaciones por separado se detectaría con éxito la ejecución. Esto es debido a que durante el análisis de la combinación de la estructura AND no se registrarían como errores los ocurridos en el bucle, evitando pérdidas de información como las ocurridas en el ejemplo anterior.

Algorithm 4: ***isFrequentPattern***: Comprobación de si un patrón es o no frecuente.

Input: un patrón para analizar *pattern*, un conjunto de registros $R[]$ correspondientes al grafo a minar y el umbral de frecuencia *threshold*.
Output: un valor booleano que indica si el patrón supera el umbral o no.

```
1 Function isFrequentPattern(pattern, R, threshold)
2     combinations ← getPatternCombinations(pattern)
3     frequencies ← ∅
4     forall the combination ∈ combinations do
5         numExecutions ← getCombinationFrequency(combination, R)
6         frequency ← numExecutions/R.size()
7         add frequency to frequencies
8     min ← minimum value in frequencies
9     return min > threshold
```

Algorithm 5: Continuación del código del Algoritmo 4.

```
1  Function getCombinationFrequency(combination, R)
2      n ← 0
3      forall the r ∈ R do
4          if parseRegister(combination, r) then
5              n++
6      return n
7
8  Function parseRegister(pattern, R)
9      lastExecutedNodes ← ∅
10     executedArcs ← ∅
11     cpyInitialNodes ← pattern.initialNodes
12     forall the destiny ∈ r do
13         if !isWorkflowExecuted(pattern, executedArcs, lastExecutedNodes) then
14             execute destiny
15             sources ← get combinationExecuted
16             failArc ← false
17             forall the source ∈ sources do
18                 if !failArc then
19                     if source ∈ pattern AND destiny ∈ pattern then
20                         arcBetweenPatternNodes(pattern, source, destiny, failArc,
                           cpyInitialNodes, executedArcs, lastExecutedNodes)
21                     else if destiny ∈ pattern then
22                         arcWithDestinyInPattern(pattern, source, destiny, failArc,
                           cpyInitialNodes, executedArcs, lastExecutedNodes)
23                     else if source ∈ pattern then
24                         arcWithSourceInPattern(pattern, source, destiny, failArc,
                           cpyInitialNodes, executedArcs, lastExecutedNodes)
25                 else
26                     restartExecution(pattern, source, destiny, cpyInitialNodes, executedArcs,
                       lastExecutedNodes)
27             if sources = ∅ AND destiny ∈ cpyInitialNodes then
28                 remove destiny from cpyInitialNodes
29                 executionOk(pattern, source, destiny, executedArcs, lastExecutedNodes)
30     return isWorkflowExecuted(pattern, executedArcs, lastExecutedNodes)
31
32 Function arcBetweenPatternNodes(pattern, source, destiny, failArc, cpyInitialNodes,
   executedArcs, lastExecutedNodes)
33     arc ← executed arc
34     if arc ∈ pattern AND source ∈ lastExecutedNodes then
35         executionOk(pattern, source, destiny, executedArcs, lastExecutedNodes)
36     else
37         if source ∈ cpyInitialNodes AND destiny ∈ pattern.finalNodes then
38             remove source from cpyInitialNodes
39             executionOk(pattern, source, destiny, executedArcs, lastExecutedNodes)
40         else
41             failArc ← true
42             restartExecution(pattern, source, destiny, cpyInitialNodes, executedArcs,
               lastExecutedNodes)
43             if source ∈ cpyInitialNodes then
44                 remove source from cpyInitialNodes
45                 executionOk(pattern, source, destiny, executedArcs, lastExecutedNodes)
```

Algorithm 6: Continuación del código del Algoritmo 5.

```
1  Function arcWithDestinyInPattern(pattern, source, destiny, failArc, cpyInitialNodes,
   executedArcs, lastExecutedNodes)
2      if destiny ∈ cpyInitialNodes then
3          remove destiny from cpyInitialNodes
4          executionOk(pattern, source, destiny, executedArcs, lastExecutedNodes)
5      else
6          failArc ← true
7          restartExecution(pattern, source, destiny, cpyInitialNodes, executedArcs,
           lastExecutedNodes)
8
9  Function arcWithSourceInPattern(pattern, source, destiny, failArc, cpyInitialNodes,
   executedArcs, lastExecutedNodes)
10     if source ∉ pattern.endNodes then
11         failArc ← true
12         restartExecution(pattern, source, destiny, cpyInitialNodes, executedArcs,
           lastExecutedNodes)
13
14 Function executionOk(pattern, source, destiny, executedArcs, lastExecutedNodes)
15     if source ∈ pattern then
16         arc ← executed arc
17         add arc to executedArcs
18         forall the input ∈ source.inputs do
19             remove source from lastExecutedNodes
20         if source.inputs = ∅ then
21             remove source from lastExecutedNodes
22     forall the output ∈ destiny.outputs do
23         remove destiny from lastExecutedNodes
24     if destiny.outputs = ∅ then
25         remove destiny from lastExecutedNodes
26
27 Function restartExecution(pattern, source, destiny, cpyInitialNodes, executedArcs,
   lastExecutedNodes)
28     arc ← executed arc
29     if source ∈ pattern then
30         propagateRestart(source, pattern, executedArcs, lastExecutedNodes,
           cpyInitialNodes)
31     if destiny ∈ pattern then
32         propagateRestart(destiny, pattern, executedArcs, lastExecutedNodes,
           cpyInitialNodes)
```

Algorithm 7: Continuación del código del Algoritmo 6.

```
1  Function propagateRestart(node, pattern, executedArcs, arc, lastExecutedNodes, cpyInitialNodes)
2      toDelete ← ∅
3      forall the executedArc ∈ executedArcs do
4          if node = arc.destiny then
5              add executedArc to toDelete
6      forall the arcToDelete ∈ toDelete do
7          remove arcToDelete from executedArcs
8          add arcToDelete.source to lastExecutedNodes
9          propagateRestart(arcToDelete.source, pattern, executedArcs, arcToDelete,
           lastExecutedNodes, cpyInitialNodes)
10     remove node from executedArcs
11     if node ∈ pattern.initialNodes then
12         add node to cpyInitialNodes
13
14 Function isWorkflowExecuted(pattern, executedArcs, lastExecutedNodes)
15     isExecuted ← false
16     if executedArcs = pattern.arcs then
17         cpyLastExecutedNodes ← lastExecutedNodes without repetitions
18         if cpyLastExecutedNodes = pattern.endNodes then
19             isExecuted ← true
20     return isExecuted
```

La función ***isFrequentPattern*** (Algoritmo 4) genera las combinaciones del patrón a analizar (línea 2) y comprueba, una a una, la frecuencia de las mismas sobre las trazas (línea 5). Por último, se comprueba si la mínima de las frecuencias es superior o igual al umbral de frecuencia (línea 9), ya que en este TFG se considera un patrón como frecuente cuando todas sus combinaciones superan este umbral.

La base de la función ***parseRegister*** (línea 8 del Algoritmo 5) desarrollado para la comprobación de si un patrón es ejecutado dentro de un registro consiste en ir ejecutando la traza del grafo, obteniendo los arcos ejecutados y comprobando que estos se encuentran en el patrón. Ante la ejecución de un arco que no se encuentra en el patrón, se debe realizar un análisis para saber si es conveniente reiniciar el marcado, continuar o realizar alguna acción diferente. A continuación se exponen los casos posibles y las respuestas diseñadas ante ellos:

- Ninguno de los nodos pertenece al grafo y, por tanto, el arco tampoco se encuentra en el mismo. La acción en este caso es avanzar, ya que se trata de un arco ajeno al patrón, que se está ejecutando en alguna rama paralela a la del patrón.
- Solo el destino se encuentra en el patrón (línea 21 del Algoritmo 5), por lo que el arco no. En este caso, se debe analizar si el nodo destino es un nodo inicial del patrón, si es así, significa que la ejecución del mismo está dando comienzo, por lo que se registra. En caso de que no se trate de un nodo inicial, el registro está entrando en el patrón por un nodo no inicial, por lo que se reinicia la cuenta.
- Solo el origen del arco se encuentra en el patrón (línea 23 del Algoritmo 5). La acción es similar al caso en el que el nodo destino es el que se encuentra en el patrón, pero en este caso se debe comprobar si el nodo origen del arco es un nodo final del patrón.

- Ambos nodos pertenecen al patrón, y el arco se encuentra en el mismo (línea 34 del Algoritmo 5). En este caso se evalúa si el nodo origen ha sido uno de los últimos nodos en ejecutarse, es decir, que no ha disparado otra tarea, y en caso positivo se registra el avance.
- Ambos nodos pertenecen al patrón, pero el arco no se encuentra en el mismo (línea 36 del Algoritmo 5). Este caso es el más complejo, y para su explicación se recurrirá a la Figura 3.14 en la que se puede apreciar una red de Petri con un patrón resaltado. Asumiendo que este es el que se está analizando, en el caso en el que se ejecute el arco $D \rightarrow G$ se detectarán ambos nodos en el patrón, pero el arco no. Por este motivo, cuando se da este caso se debe comprobar si el nodo origen del arco es un nodo final del patrón y si el destino del arco es un nodo de inicio. En este caso se acepta la ejecución y no se reinicia.

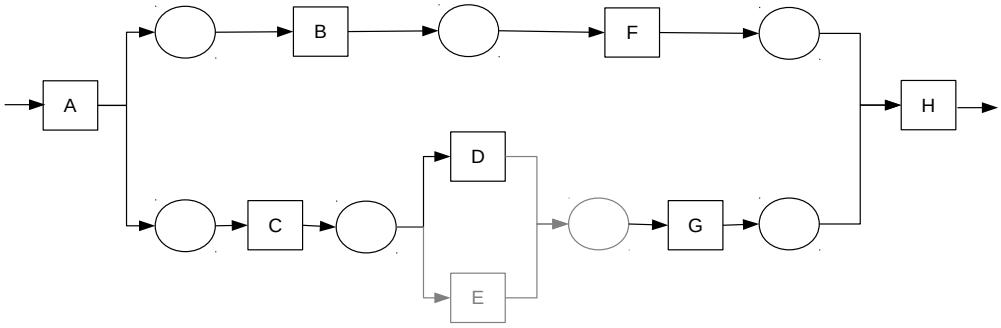

Figura 3.14: Patrón resaltado en una red de Petri.

3.2.3.4. Generación de combinaciones

Como se ha comentado previamente y se refleja en la línea 2 del Algoritmo 4, para la comprobación de la frecuencia de un patrón es necesario generar las combinaciones del mismo. Con este objetivo se ha creado el algoritmo ***getPatternCombinations*** que dado un patrón genera las combinaciones del mismo. Este algoritmo se basa recorrer el grafo generando combinaciones diferentes cuando se encuentra una selección (OR), de forma que se parte de un nodo, y se amplía la combinación con sus entradas y salidas, aplicando recursividad en la expansión. Cuando las entradas o salidas contienen una selección, se realiza una copia del patrón, y cada una seguirá una rama de la selección diferente.

A continuación, en los Algoritmos 8 y 9 se muestra el código correspondiente al algoritmo ***getPatternCombinatios***, que se ha diseñado en este TFG para cumplir con los objetivos de esta funcionalidad.

Algorithm 8: *getPatternCombinations*: Obtiene las combinaciones de un patrón dado.

Input: Patrón *initialPattern* del que se generarán las combinaciones.
Output: Conjunto de patrones, correspondientes con las combinaciones de *pattern*.

```
1  Function getCombinations(initialPattern)
2      startNodes ← initialPattern.initialNodes
3      candPatterns ← ∅
4      listExploited ← ∅
5      forall the startNode ∈ startNodes do
6          pattern ← empty pattern
7          add startNode to pattern
8          add pattern to candPatterns
9          add ∅ to listExploited
10         index ← index of pattern in candPatterns
11         exploitNode(node, candPatterns, listExploited, index)
12     combinations ← ∅
13     forall the combination ∈ candPatterns do
14         if combination ∉ combinations AND combination is compliant with initialPattern then
15             add combination to combinations
16     return combinations
17
18 Function exploitNode(node, candPatterns, listExploited, indexPattern)
19     exploited ← listExploited[indexPattern]
20     if node ∉ exploited then
21         add node to exploited
22         exploitSet(true, candPatterns, listExploited, indexPattern, node.inputs, 0)
23         continueWithOutputs(node, candPatterns, listExploited, indexPattern)
24
25 Function continueWithOutputs(node, candPatterns, listExploited, indexPattern)
26     cpyPatterns ← ∅
27     cpyExploited ← ∅
28     for i = indexPattern to candPatterns.size do
29         add candPatterns[i] to cpyPatterns
30         add listExploited[i] to cpyExploited
31     remove elements in candPatterns and listExploited added in previous loop
32     forall the pattern ∈ cpyPatterns do
33         exploited ← element with same index than pattern but in cpyExploited
34         newCandPatterns ← ∅
35         newListExploited ← ∅
36         add pattern to newCandPatterns
37         add exploited to newListExploited
38         exploitSet(false, newCandPatterns, newListExploited, 0, node.outputs, 0)
39         add all newCandPatterns to candPatterns
40         add all newListExploited to listExploited
```

Algorithm 9: Continuación del código del Algoritmo 8.

```
1  Function exploitSet(explInputs, node, candPatterns, listExploited, indexPattern, set, indexSet)
2      if set.size > indexSet then
3          subset ← set[indexSet]
4          branchAlreadyExploited ← false
5          forall the connection ∈ subset do
6              if !branchAlreadyExploited then
7                  if explInputs then
8                      arc ← arc from connection to node
9                  else
10                     arc ← arc from node to connection
11                 pattern ← candPatterns[indexPattern]
12                 exploited ← listExploited[indexPattern]
13                 if arc ∉ pattern then
14                     newPattern ← deep copy of pattern
15                     newExploited ← deep copy of exploited
16                     if connection ∉ newPattern then
17                         add connection to newPattern
18                     add arc to newPattern
19                     if newPattern is a combination of initialPattern then
20                         add newPattern to candPatterns
21                         add newExploited to listExploited
22                         newIndexPattern ← index of pattern in candPatterns
23                         exploitNode(connection, candPatterns, listExploited, newIndexPattern)
24                         continueExploiting(explInputs, node, candPatterns, listExploited,
                           newIndexPattern, set, indexSet + 1)
25                 else if (arc is not endLoopArc or startLoopArc) OR (loop is not closed) then
26                     branchAlreadyExploited ← true
27         if branchAlreadyExploited then
28             exploitSet(explInputs, candPatterns, listExploited, indexPattern, set, indexSet + 1)
29         else if indexPattern = (candPatterns.size − 1) then
30             remove candPatterns[indexPattern]
31             remove listExploited[indexPattern]
```

El primer problema que se encuentra es la selección del nodo de partida, debido a que las combinaciones se realizan sobre patrones del grafo, no existe un solo nodo de inicio, pudiendo haber construcciones como las mostradas en la Figura 3.15, que necesitan de dos nodos de inicio para expandir las combinaciones, ya que en caso de partir de uno solo, al pertenecer a una selección, no se podrá tener en cuenta los otros caminos de la misma.

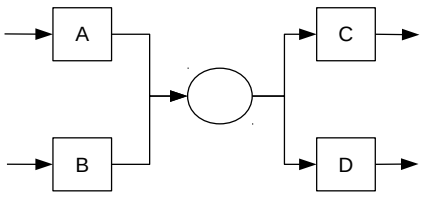

Figura 3.15: Patrón formado por una doble selección.

Se ha ideado un algoritmo que, partiendo de los nodos iniciales, genera la lista de caminos posibles y escoge el conjunto mínimo de forma que de cada camino, al menos se hubiera recogido un nodo. Con este conjunto se esperaba poder generar las combinaciones del patrón, partiendo del mínimo número de nodos posible, pero han aparecido patrones con irregularidades que forzaron a la toma de otro método.

Lo que se ha realizado en última instancia ha sido la generación de las combinaciones a partir de todos los nodos de inicio, eliminando a posteriori las que estuvieran repetidas. El funcionamiento básico es expandir la búsqueda desde cada nodo (línea 11 del Algoritmo 8), realizando una copia de la combinación cuando hay una selección (línea 14 del Algoritmo 9), y continuando con el siguiente nodo en todas las copias. De esta forma se consiguen recorrer todos los caminos del grafo, generando combinaciones cada vez que se descubren dos posibles caminos, y continuando en cada uno.

Cabe destacar que, cuando se están explotando las diferentes ramas de un patrón, es necesario realizar la explotación de la siguiente rama con todas las copias generadas hasta el momento. Para explicar más claramente este requisito se hace referencia a la Figura 3.16, donde se muestra un patrón con tres ramas en paralelo, y teniendo una de ellas una selección. En este caso, empezando la explotación por el nodo A, se comenzarían a expandir por las ramas de sus salidas. Después de añadir el nodo de la primera rama y no poder continuar por él, se pasa a la segunda, donde se encuentra una selección, por lo que genera una copia para cada nodo y continúa con C en uno y con D en el otro. Como ambos carecen de más conexiones, se pasa a la siguiente, pero es aquí donde se debe tener en cuenta que ahora hay dos combinaciones que deben explorar la tercera rama del AND-split, esto se realiza con la llamada a la función ***ContinueExploiting*** en la línea 24 del Algoritmo 9. Esta función continúa la expansión de la misma forma que la función ***continueWithOutputs*** (línea 25 del Algoritmo 8) pero continuando con el siguiente conjunto de conexiones, en vez de empezar con las salidas, como hace esta.

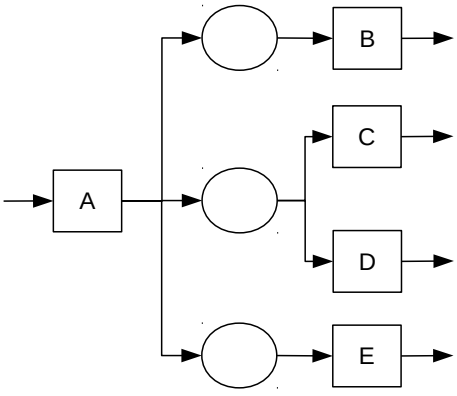

Figura 3.16: Patrón AND con una rama OR.

Otra característica que se debe tener en cuenta es que los patrones pueden contener tareas de un bucle sin tener el bucle entero. Como se muestra en la Figura 3.17, el patrón superior y el patrón inferior pertenecen al mismo grafo, de hecho el patrón (b) es un subpatrón del (a), pero cuando se trata de generar las combinaciones, el patrón (a) generará una sola combinación, con el bucle completo, mientras que el patrón (b) deberá generar dos combinaciones, ya que es como si

se tratase de una selección.

El problema con este tipo de patrones reside en cómo hacer que un algoritmo detecte que, en el patrón de la Figura 3.17a el arco $D \rightarrow B$ no debe provocar una bifurcación de la combinación, mientras que en el patrón 3.17b sí. Para esto se ha necesitado la ayuda del algoritmo ***searchStartEndLoopArcs***, que se explicará posteriormente, y que detecta los arcos de inicio y fin de un bucle.

Sabiendo que $D \rightarrow B$ es un arco de fin de bucle (o *endLoopArc*) y $B \rightarrow C$ un arco de inicio de bucle (*startLoopArc*), podemos establecer una condición que haga que, para separar una combinación en dos, es necesario que ambos arcos no sean *startLoopArc* o *endLoopArc* de un bucle cerrado (línea 25 del Algoritmo 9). En caso de serlo, pero de un bucle abierto, como en la Figura 3.17b, se generarán las combinaciones como si se tratase de una selección.

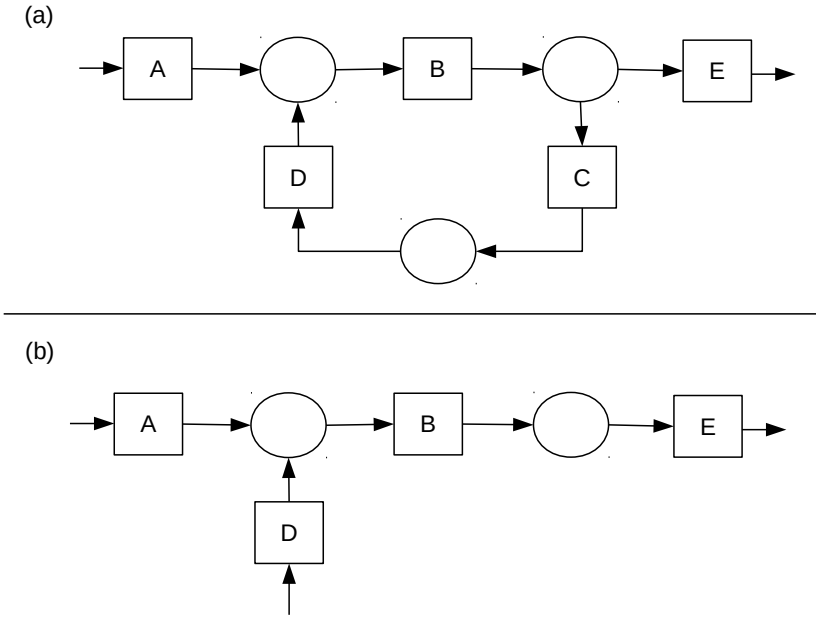

Figura 3.17: (a) Red de Petri con un bucle. (b) Patrón de la red de Petri con una parte del bucle

3.2.3.5. Detección de arcos de bucle

Uno de los objetivos del proyecto, de cara al soporte de bucles en la búsqueda de los patrones frecuentes, es diseñar un algoritmo que detectase la existencia de bucles en un grafo, identificando los arcos que suponen el inicio y fin de los mismos (*startLoopArc* y *endLoopArc* a partir de ahora). Este algoritmo no solo sería de utilidad en el presente proyecto, sino que se podría utilizar para mejorar el postprocesado de los grafos generados por el algoritmo ProDiGen [12].

Debido a la inexistencia de aproximaciones que realicen dicho cometido, y a la poca documentación sobre este tipo de problemas, se ha tenido que realizar el estudio con la base adquirida en la fase inicial del proyecto. Por este motivo, se han realizado diferentes aproximaciones que fueron presentando errores en las fases de pruebas, provocando que se volviese a diseñar el algoritmo en

varias ocasiones.

A continuación se explicarán algunas de las aproximaciones que se han realizado y los motivos de sus descartes, finalizando con el algoritmo definitivo, y el pseudocódigo asociado al mismo.

Primera aproximación

La primera prueba consistió en recorrer el grafo desde el inicio en anchura, y marcar como *endLoopArc* el arco con el que se llegase a un nodo ya alcanzado previamente. Pero esta aproximación presenta problemas cuando hay selecciones, ya que alcanza el nodo OR-join tantas veces como selecciones haya, y en las siguientes a la primera se detectaría como bucle.

Por ello, se decidió utilizar la misma aproximación pero parar la explotación cuando se llegase a un nodo que no tenía todos sus orígenes explotados, y se retomase su explotación cuando fuera alcanzado por todas sus ramas. De esta forma, los OR-join serían una sincronización que forzaría a la espera del análisis de todas sus ramas, y haría que los bucles como los de la Figura 3.18 se detectasen sin problema, ya que aunque se llegase antes a G por la rama superior, para su explotación se tendría que esperar por la rama inferior, que detectaría el bucle y avanzaría. El problema en este caso apareció cuando se implementó la solución, ya que al parar la explotación cuando no todas las entradas fueron alcanzadas hace que los destinos de un *endLoopArc* como D también quedasen parados, haciendo imposible la distinción de bucles o simples OR-join.

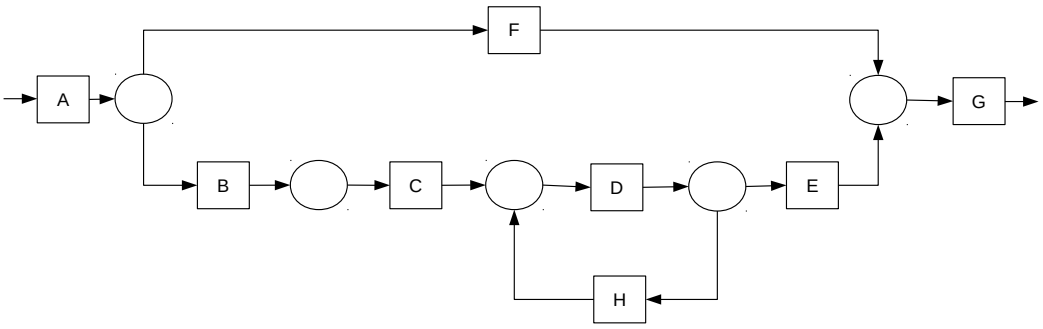

Figura 3.18: Red de Petri con una selección y un bucle en una de las ramas.

Segunda aproximación

Para esto se pensó en que la información tendría que residir en el origen de los datos, por lo que se ideó una fase de '*recuperación*', en la que se analizarían los nodos pendientes con los registros. Si seguimos la ejecución del algoritmo con la red de Petri de la Figura 3.18, tendríamos un avance que se pararía al llegar a G en primer lugar (por ser recorrido en anchura), y a D en segundo. En este punto no se podría avanzar más por lo que se entraría en fase de recuperación, y se analizarían los arcos de uno de los dos nodos. En caso de ser D el nodo elegido, se comprobaría en las ejecuciones de los registros, qué nodo aparece en primer lugar, siendo siempre D. Debido a este resultado, en el que la entrada del nodo en el que se ha parado la ejecución aparece siempre después, se puede deducir que se trata de un bucle, ya que, según los registros, es necesaria la ejecución de D, para que se efectúe la de H.

Si el nodo escogido para análisis fuera G, el resultado sería el mismo, ya que se analizaría en las trazas si E precede a G, o viceversa. Los resultados serían que E siempre precede a G, lo cual es normal, por lo que se sabría que no se trata de un bucle.

Una vez implementado este algoritmo se consiguieron resultados positivos en un gran número de pruebas, pero presentó problemas con casos como el reflejado en la Figura 3.19. En este grafo, se pasaría a la fase de recuperación con F y con E, pero al analizar ambos casos, se darían registros en los que se ejecuta primero la entrada, y otros en los que se ejecuta primero la salida. Esto es debido a que si se coge el camino superior, H se ejecutaría antes que E, y si se coge el inferior, sería E el que precediese a H. Por este motivo se pensó en tomar un umbral, que dependiendo del porcentaje de trazas en las que se ejecutase uno u otro, se decidiese una cosa u otra. Pero podría darse el caso en el que en el 90 % de las trazas, incluso más, se tomase el camino superior en primer lugar, invalidando la teoría de un umbral.

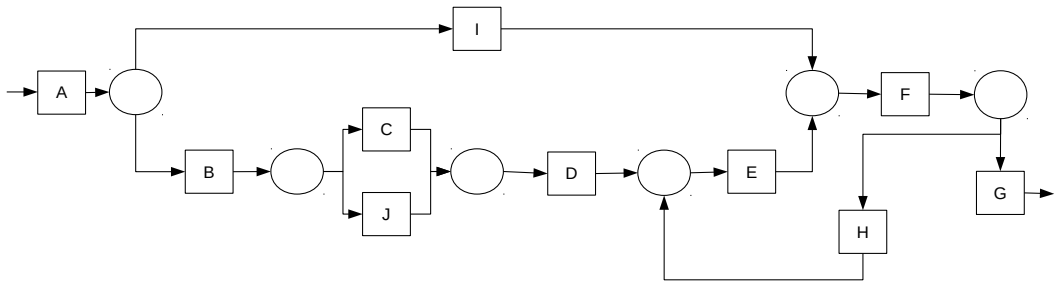

Figura 3.19: Red de Petri con bucles que saltan desde fuera de una selección al interior de la misma.

Tercera aproximación

Después de numerosos intentos, todos con casos especiales que anulaban las teorías realizadas, se llegó a un modelo que fue capaz de detectar los bucles con éxito en todas las pruebas a las que ha sido sometido.

El algoritmo se basa en tres sencillos pasos:

1. Se comienza avanzando desde el nodo inicio, siempre hacia adelante.
2. Cuando se encuentra un nodo con más de una salida (AND-split u OR-split) se analiza si cada arco es un startLoopArc.
   - Para esto se avanza y si se llega al final del nodo se marca el arco como normal.
   - Si, en cambio, se alcanza el origen del arco que se está analizando se trata de un *startLoopArc*.
3. Si mientras se analiza un arco se encuentra otro nodo con más de una salida, se analiza cada uno antes de continuar.

El funcionamiento de este algoritmo reside en la base de que, para detectar un *startLoopArc* primero debe haber recorrido todo un camino, sin haber pasado por un *startLoopArc*, y haber llegado al origen del arco. Como cuando se encuentra un posible *startLoopArc* el análisis pasa a este (análisis LIFO), antes de escoger un camino se sabe si se trata de un bucle o no.

Para una mejor comprensión, previamente a mostrar el pseudocódigo correspondiente, en la Tabla 3.3 se muestra un ejemplo del algoritmo aplicado al grafo de la Figura 3.19.

Estado	Arco a analizar	Acción a realizar
Inicio del algoritmo	$A \to I$	Como A tiene varios outputs, se debe analizar si $A \to I$ es un *startLoopArc*.
Analizando $A \to I$	$I \to F$	Se avanza, pues I solo tiene un output.
Analizando $A \to I$	$F \to H$	Como F tiene varios outputs, se analizará cada uno de ellos.
Analizando $F \to H$	$H \to E$	Se avanza con normalidad.
Analizando $F \to H$	$E \to F$	Se encuentra un bucle, el destino del arco a analizar es el origen del arco que se está analizando. Se fija $F \to H$ como *startLoopArc*.
Analizando $A \to I$	$F \to G$	Como se ha detectado que $F \to H$ es un inicio de bucle, ese camino no se toma, y se continúa analizando otro output.
Analizando $F \to G$	—	G es el nodo final, por lo que no es un *startLoopArc*,
Analizando $A \to I$	$F \to G$	Como ya se han analizado todos los outputs de F se continúa avanzando. Como $F \to H$ es un *startLoopArc*, no se continúa por él. Se continúa por $F \to G$.
Analizando $A \to I$	—	G es el nodo final y no hay más caminos, por lo que $A \to I$ no es un *startLoopArc*.
Analizando $A \to B$	$B \to C$	B tiene varios outputs, por lo que se procederá al análisis de los mismos.
Analizando $B \to C$	$C \to D$	Se avanza con normalidad.
Analizando $B \to C$	$D \to E$	Se avanza con normalidad.
Analizando $B \to C$	$E \to F$	Se avanza con normalidad.
Analizando $B \to J$	$F \to G$	Como F ya ha sido analizado, se avanza solo por los caminos que no son *startLoopArc*. En este caso únicamente $F \to G$.
Analizando $B \to J$	—	Se ha llegado al final y no hay más caminos, por lo que $B \to J$ no es un *startLoopArc*. Se sigue con el análisis de $A \to B$por una de las ramas que no son *startLoopArc*.
Analizando $A \to B$	$C \to D$	Se avanza con normalidad. Se repite el proceso analizado antes para $C \to D$ y $J \to D$ se llega al final.
Analizando $A \to B$	$J \to D$	Se avanza con normalidad. Se repite el proceso analizado antes para $C \to D$ y $J \to D$ se llega al final.No quedan más caminos para examinar, por lo que $A \to B$ no es un *startLoopEnd*.

Tabla 3.3: Simulación de la búsqueda de *startLoopArcs*.

Una vez detectados los *startLoopArcs*, la detección de los *endLoopArcs* se vuelve trivial. El procedimiento consiste en coger cada *startLoopArc* y avanzar desde su nodo destino, en cada nodo se realiza un análisis, si retrocediendo se llega al inicio del grafo sin pasar por un *startLoopArc* es que el arco que se acaba de analizar es un *endLoopArc*, ya que hay un camino que llega a él desde el inicio del grafo sin pasar por un bucle. Se muestra en la Tabla 3.4 el ejemplo de la ejecución de este algoritmo para la figura 3.19.

Estado	Arco a analizar	Acción a realizar
Se inicia con el arco $F \to H$, analizando $F \to H$.	H	Solo tiene un input, y el arco que los une es un *startLoopArc* por lo que se continúa.
Analizando $H \to E$	E	Tiene dos inputs, se comienza con D.
Analizando $H \to E$	C	$D \to E$ no es *startLoop*, por lo que se retrocede a C.
Analizando $H \to E$	B	$C \to D$ no es *startLoop*, por lo que se retrocede a B.
Analizando $H \to E$	A	$B \to C$ no es *startLoop*, por lo que se retrocede a A.
Analizando $H \to E$	—	$A \to B$ no es *startLoop*, por lo que se intenta retrocederpero A es el nodo inicio, por lo que $H \to E$ es un *endLoopArc*.

Tabla 3.4: Simulación de la búsqueda de *endLoopArcs*.

A continuación, una vez ya explicado el algoritmo ***searchStartEndLoopArcs***, se muestra el pseudocódigo diseñado que lo implementa, realizando una serie de comentarios para mejorar la comprensión del mismo:

Algorithm 10: Identifica los arcos *startLoopArc* y *endLoopArc* de un grafo.

Input: Grafo *graph* sobre el que realizar la búsqueda.
Output: void. Identifica los arcos de inicio y fin de bucle.

```
1  Function exploreArcs(graph)
2      searchStartLoopArcs(graph)
3      searchEndLoopArcs(graph)
4
5  Function searchStartLoopArcs(graph)
6      exploited ← ∅
7      go forward from initial node until found node with more than one output
8      node ← get node
9      sortedOutputs ← node.outputs sorted by id
10     forall the output ∈ sortedOutputs do
11         analizeOr(graph, node, output, exploited)
```

Algorithm 11: Continuación del código del Algoritmo 10.

```
1  Function analizeOr(graph, source, destiny, exploited)
2      found ← false
3      lifo ← ∅
4      exploited ← ∅
5      add destiny to lifo
6      add destiny to exploited
7      while fifo ≠ ∅ AND !found do
8          node ← lifo.pop
9          if node ∉ exploited AND node.outputs > 1 then
10             extendExploration(graph, node, exploited)
11         reversedSortedOutputs ← node.outputs sorted reversely by id
12         forall the output ∈ reverseSortedOutputs do
13             arc ← arc from node to output
14             if output = source AND arc is not startLoopArc then
15                 mark arc as startLoopArc
16                 found ← true
17             else if arc is not startLoopArc then
18                 add output to lifo
19
20 Function extendExploration(graph, node, exploited)
21     allStartLoopArcs ← true
22     sortedOutputs ← node.outputs sorted by id
23     forall the output ∈ sortedOutputs do
24         arc ← arc from node to output
25         if arc is not startLoopArc then
26             analizeOr(graph, node, output, exploited)
27         if arc is not startLoopArc then
28             allStartLoopArcs ← false
29     if allStartLoopArcs then
30         unmark startLoopArc from arcs with node as source
```

La función ***searchStartLoopArcs*** especificada en el Algoritmo 10 se corresponde con la búsqueda de *startLoopArcs* en el grafo, para ello comienza avanzando hasta el primer nodo que tenga más de una salida (línea 7 del Algoritmo 10) y lanza, por cada arco, la función *analizeOr*. Esta realiza la búsqueda en profundidad, lanzando la función *extendExploration* cuando se encuentra un nodo con más de una salida, para analizarlos (línea 10 del Algoritmo 11). De esta forma se ejecuta el algoritmo previamente explicado.

Cabe destacar, que el orden de análisis es importante (líneas 9 del Algoritmo 10 y líneas 11 y 22 del Algoritmo 11), debido a que si no se realiza de forma ordenada, al analizar un arco en búsqueda de bucle, se podrán escoger caminos antes de lo debido, provocando detecciones erróneas de bucles.

Algorithm 12: Continuación del código del Algoritmo 11.

```
1  Function searchEndLoopArcs(graph)
2      forall the arc ∈ graph.arcs do
3          if arc is startLoopArc then
4              if arriveStartWithoutLoop(graph, node) then
5                  mark arc as endLoopArc
6              else
7                  continueWithOutputs(graph, node)
8
9  Function arriveStartWithoutLoop(graph, node)
10     fifo ← ∅
11     exploited ← ∅
12     add node to fifo
13     reached ← false
14     while fifo ≠ ∅ AND !reached do
15         actualNode ← fifo.get
16         forall the input ∈ actualNode.inputs do
17             arc ← arc from input to actualNode
18             if input = graph.getInitialNode then
19                 reached ← true
20             else if input ∉ exploited AND arc is not startLoopArc then
21                 add input to exploited
22                 add input to fifo
23     return reached
24
25 Function continueWithOutputs(graph, node)
26     forall the output ∈ node.outputs do
27         arc ← arc from node to output
28         if arriveStartWithoutLoop(graph, output) then
29             mark arc as endLoopArc
30         else
31             continueWithOutputs(graph, output)
```

La función ***searchEndLoopArcs*** del Algoritmo 12 recorre los arcos marcados como *startLoopArc*, primero comprueba si este es también un arco fin de bucle, caso que se muestra en la Figura 3.20[3], en la que hay un arco de C a B, y en caso negativo expande la búsqueda con las salidas del destino del arco.

Para comprobar si se alcanza el inicio del grafo se realiza un bucle con una estructura *FIFO* (*first in first out*), que retrocede, sin pasar por arcos *startLoopArc*, y devuelve *true* si alcanza el inicio del grafo (línea 14 del Algoritmo 12).

3.2.4. Mejoras realizadas

A continuación se introducirán las mejoras realizadas sobre el algoritmo. Cabe destacar que, en la sección anterior se han explicado las funcionalidades implementadas que no estaban especificadas en el algoritmo w-find y gran parte de estas se han mejorado con ampliaciones necesarias para

[3]En esta figura aparece una tarea *silenciosa* (color negro). Estas tareas se caracterizan por no suponer ninguna acción, utilizándose únicamente para control del flujo.

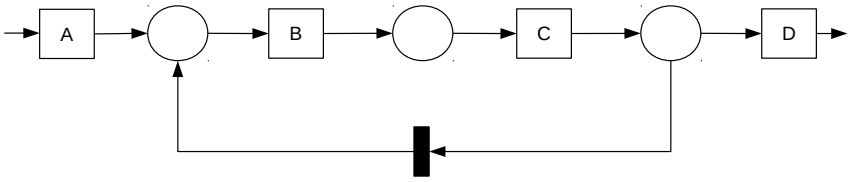

Figura 3.20: Red de Petri con un bucle de longitud 1.

dar soporte a bucles, siendo la única que se ha implementado sin mejoras la búsqueda de arcos frecuentes.

3.2.4.1. Concepto de *w*-pattern

Uno de los conceptos utilizados en el algoritmo *w*-find es el de weak pattern, o *w*-pattern, definiéndolos como:

> *[...] connected subgraphs p which are "closed" w.r.t. local and global constraints, i.e., such that* $p \models c$ *for all* $c \in C_L \cup C_G$*. We shall denote such graphs weak patterns, or simply w-patterns*

A partir de dicha definición, se entiende que un *w*-pattern se corresponde con un patrón que cumple todas las restricciones globales y locales del grafo. Las restricciones globales se corresponden con dependencias entre tareas no relacionadas directamente, como por ejemplo la relación que existe entre las tareas f y m de la Figura 3.21. En este caso, aunque el grafo lo permita, si se ejecuta la tarea de registrar un cliente, no se podrá ejecutar en la misma instancia la aplicación de un descuento por fidelidad, ya que el cliente es nuevo, esta relación es una restricción global.

Por otro lado, las restricciones locales las define como las relaciones directas entre tareas, como la que obliga a la tarea b a ejecutar una sola salida.

A partir de estas definiciones, entendemos que los *w*-pattern generados por el algoritmo *w*-find se corresponden con subpatrones del grafo, que no rompen ninguna de esas restricciones, por lo que no contempla patrones con combinaciones en caso de que tanto $b \rightarrow c$ como $b \rightarrow i$ sean frecuentes, mostrando dos patrones separados.

En el desarrollo de este TFG se ha ampliado la concepción de *w*-pattern generando patrones con varios caminos en una selección, siempre y cuando todas las combinaciones del patrón superen el umbral (Algoritmo 4).

3.2.4.2. Concepto de *ew*-pattern

Otro concepto utilizado en el algoritmo *w*-find es el de *ew*-pattern, definiéndolo en el mismo como:

> *A pattern is an elementary w-pattern (cf. ew-pattern) for a node a if it is the minimal (w.r.t. set inclusion) w-pattern containing a.*

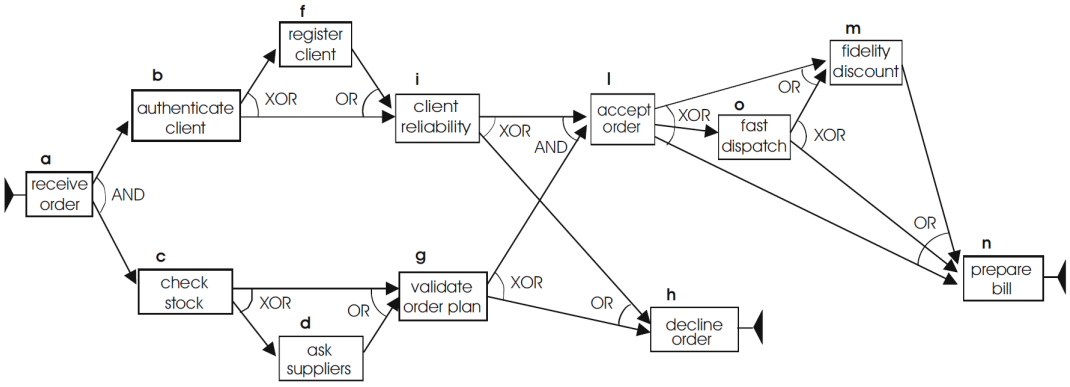

Figura 3.21: Workflow para un proceso *OrderManagement* [10].

A partir de esto se extrae la definición de *elementary weak pattern* de un nodo como el patrón mínimo que cumple con las restricciones del grafo, y que contiene a un nodo. Así se podrían extraer como ejemplos de *ew*-patterns del grafo de la Figura 3.21, los mostrados en la Figura 3.22.

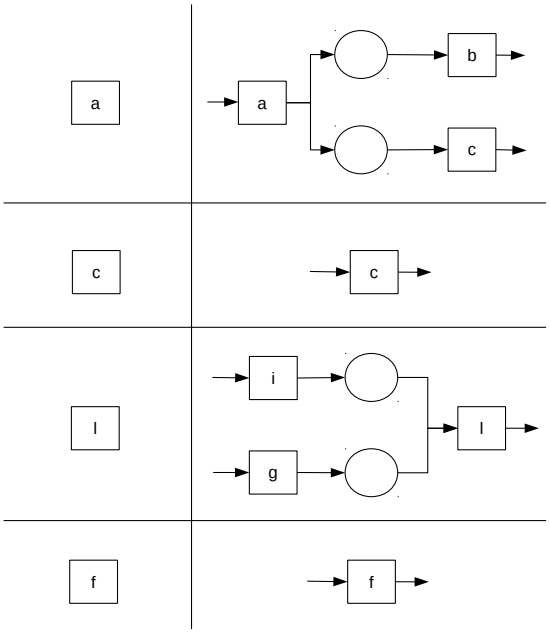

Figura 3.22: Ejemplos de *ew*-patterns del grafo de la Figura 3.21.

Una vez implementado el algoritmo con esta definición, se han observado resultados del grafo de la Figura 3.21 como el mostrado en la 3.23. Esto es debido a que en el patrón resultado, una de las combinaciones es $f \rightarrow i$ y otra $d \rightarrow g$, siendo estas frecuentes, pero si analizamos la naturaleza del grafo, cada vez que se ejecuta f, es porque se ha ejecutado b con anterioridad, por lo que sería razonable que las apariciones de f como patrón frecuente fueran acompañadas de b, al igual que en una construcción AND como la de a, se añaden los elementos de sus ramas ya que siempre se ejecutarán cuando a lo haga.

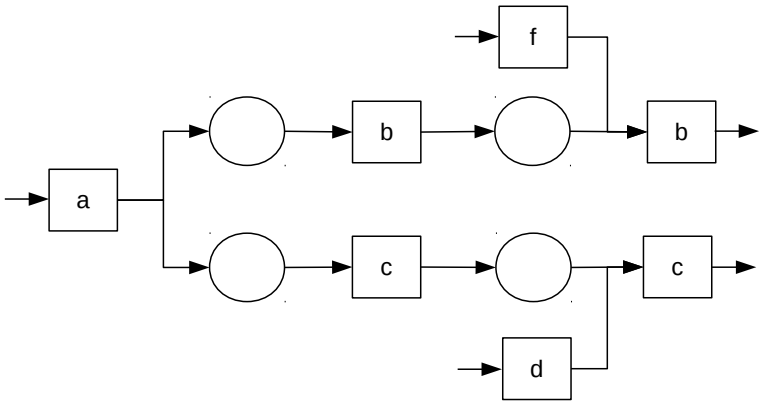

Figura 3.23: Patrón resultado de una ejecución de w-find sobre la red de Petri de la Figura 3.21.

Por este motivo, y con objetivo de evitar resultados como los anteriormente comentados, se ha redefinido el concepto de ew-pattern de forma que, en caso de que un nodo tenga una sola conexión, al ser equivalente a un AND de una sola rama, esta se incluya en su *elementary weak pattern*. En la Figura 3.24 se muestra un conjunto de ew-patterns correspondientes a la Figura 3.21, pero con el nuevo concepto de patrón elemental.

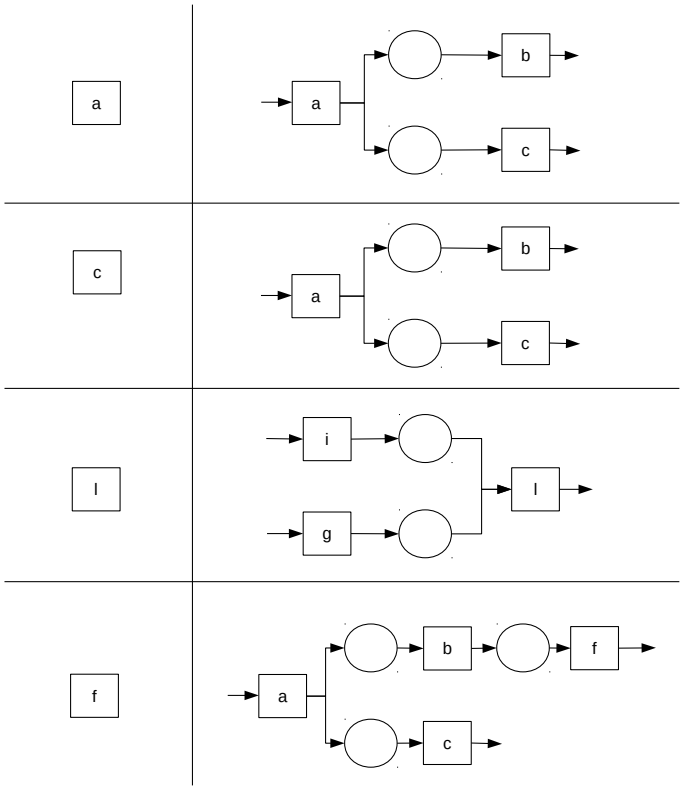

Figura 3.24: Ejemplos de ew-patterns de la Figura 3.22 con el nuevo concepto.

De esta forma, no solo se evitan resultados como los comentados en el párrafo anterior, sino que se reduce el espacio de búsqueda y el número de iteraciones del algoritmo debido a la existencia de patrones iniciales de mayor tamaño.

3.2.4.3. Conjunto de arcos frecuentes

En el algoritmo w-find se utiliza un conjunto de arcos denominado *frequent arcs* con los que se expanden los patrones en cada iteración. Este conjunto está formado por todos los arcos cuyo origen no sea un '*fork determinista*', es decir, un AND-split.

Esta exclusión de dichos arcos del conjunto se debe a que, los arcos cuyo origen pertenezca a una estructura AND siempre van a estar en el patrón elemental de su origen, haciendo que la inclusión del arco en una iteración se haga al añadirle el ew-pattern. Por ejemplo, en la Figura 3.21, no se incluiría el arco $a \rightarrow c$ ya que este siempre irá ligado al patrón elemental correspondiente al nodo a.

Por este mismo motivo, los arcos cuyo destino sea un AND-join también deberían ser excluídos, ya que se añadirán en conjunto cuando se utilice el ew-pattern correspondiente.

3.2.4.4. Acciones suprimidas

Tras realizar un análisis minucioso sobre la publicación del algoritmo w-find [10] se ha detectado una instrucción cuya realización no aporta nada al algoritmo, esta es la efectuada en la línea 4 del pseudocódigo de la Figura 3.6. En esta línea se realiza la intersección entre la variable *frequentArcs* y el conjunto de arcos $E^{\subseteq}$. En la línea anterior, los arcos de la variable *frequentArcs* se extraen del conjunto de $E^{\subseteq}$, por lo que esta variable contendrá el conjunto o un subconjunto del mismo. Por este motivo, realizar la intersección entre un conjunto, y un subconjunto del mismo carece de finalidad, ya que el resultado siempre será el del subconjunto.

Otro caso de optimización está relacionado con las comprobaciones que se hacen en las funciones *addFrequentConnection* y *addFrequentArc* ($WS \models q$). Esta instrucción comprueba que el patrón generado cumple con las restricciones del grafo, tanto globales como locales. Dentro de las locales se encuentra la posibilidad de que se haya añadido un arco perteneciente a una selección, o perteneciente a un AND-join, ya que estos no fueron descartados de los arcos frecuentes.

En el caso mejorado del algoritmo de este TFG, los arcos a añadir solo forman parte de estructuras OR, y como los resultados del algoritmo permiten a un patrón presentar varias ramas dentro de una selección, la comprobación que se hace pierde su finalidad.

Para una mejor comprensión se aclarará que, en el algoritmo diseñado en el presente TFG, la forma de ampliar patrones es añadiéndoles arcos de una selección, o añadiéndole patrones elementales. Como los patrones elementales contienen las estructuras AND completas y los arcos pertenecen a selecciones, la única comprobación que se debe hacer en este caso, cuando la expansión se realizó añadiendo un patrón elemental, es ver si el patrón resultante es conexo.

De esta forma se reduce el tiempo de computación por liberar cálculos innecesarios. Se ha de mencionar que esto es debido a la exclusión de las restricciones globales del grafo. Se han decidido excluir estas restricciones ya que, al recoger de los registros los patrones frecuentes, si una restricción no permite a dos tareas ejecutarse a la vez, esta relación nunca aparecerá en las ejecuciones, no permitiendo que el algoritmo detecte como frecuente un patrón que las contenga.

3.2.4.5. Filtrado de los resultados

El resultado devuelto por el algoritmo w-find está compuesto por el conjunto de todos los patrones frecuentes encontrados, en cualquiera de las iteraciones, por lo que contenía un alto índice de redundancia. Para poder facilitar la búsqueda de construcciones al usuario, en la aplicación desarrollada por este TFG se ha aplicado un filtro al conjunto de patrones resultado que elimina todos aquellos que estén contenidos en algún patrón del conjunto.

3.2.4.6. Análisis de frecuencia con bucle

Como ya se ha explicado en la sección anterior, uno de los objetivos de este Trabajo Fin de Grado es el de detectar patrones frecuentes en grafos restringidos con bucles, para esto se ha realizado un análisis elaborando el algoritmo ***isFrequentPattern*** (Algoritmo 4) que consigue calcular la frecuencia de un patrón perteneciente a un grafo con bucles.

3.2.4.7. Generación de combinaciones

Para el funcionamiento del algoritmo ***isFrequentPattern*** ha sido necesaria la creación de otro algoritmo (***getPatternCombinations***) que genera las combinaciones de un patrón, que también ha sido explicado en profundidad en la sección anterior (Algoritmo 8).

3.2.4.8. Detección de arcos de bucles

Por últmo, para la generación de las combinaciones de un patrón, ha sido necesaria la creación de un algoritmo que detectase los arcos de inicio y fin de bucle, sirviendo el mismo para campos distintos al que aborda este TFG, como la minería de procesos con el algoritmo ProDiGen [12]. Este algoritmo, llamado ***searchStartEndLoopArcs***, se ha especificado en la sección anterior (Algoritmo 10).

Capítulo 4

Análisis de requisitos

Haciendo referencia al PMBOK, la recopilación de requisitos se define como:

> "*[...]el proceso que consiste en definir y documentar las necesidades de los interesados a fin de cumplir con los objetivos del proyecto. El éxito del proyecto depende directamente del cuidado que se tenga en obtener y gestionar los requisitos del proyecto y del producto*"

Con esta definición se puede deducir que esta fase es una de las más importantes de la ingeniería de software, por lo que debe ser llevada a cabo con extremo cuidado y minuciosidad. Si en un proyecto la definición e identificación de los requisitos no se hace correctamente las consecuencias pueden ser catastróficas, llevando incluso, en algunos casos, a la cancelación del mismo.

Por estos motivos se hará, a continuación, un análisis del alcance del proyecto, estudiando las características del mismo, funcionalidades, objetivos deseados y realizando una identificación de los requisitos que debe cumplir, y una descripción de los mismos.

4.1. Identificación de los requisitos del sistema

Para la realización de un análisis de requisitos correcto, primero se debe comprender correcta y enteramente los objetivos y alcance del proyecto. Por este motivo se han llevado a cabo una serie de reuniones con los interesados, en este caso los tutores, en las que se han discutido y definido los requisitos necesarios para la completitud del proyecto. En sucesivas reuniones se han refinado los requisitos con el objetivo de reducir al máximo la ambigüedad que estos pueden presentar.

4.1.1. Actores

Dentro del análisis y definición de requisitos, el primer paso es identificar qué actores van a intervenir en la correcta ejecución del sistema. Un actor representa un rol interpretado por una persona, dispositivo o sistema externo con capacidades de interacción. La identificación de los actores en este proyecto es directa: el único rol a definir es el del **usuario que interacciona**

con el algoritmo que se describe en la presente memoria. Siguiendo una especificación formal, la descripción de los actores se detalla en la tabla que se presenta a continuación:

ID	Actor-01
Nombre	Usuario
Descripción	Este actor representa al usuario que interaccionará con la aplicación, definiendo los parámetros necesarios y lanzando la ejecución del algoritmo, así como realizando la visualización de los resultados.

4.1.2. Casos de uso

Para poder efectuar una correcta identificación de los requisitos del sistema, el primer paso implica identificar los casos de uso del mismo. Un caso de uso [2] es una descripción de un conjunto de secuencias de acciones, incluyendo variantes, que ejecuta un sistema para producir un resultado observable de valor para un actor. Es así como los os casos de uso se utilizan para capturar el comportamiento deseado en el desarrollo, pero sin la especificación sobre la forma en la que este comportamiento es implementado. Esto permite a los clientes y al equipo de desarrollo abstraerse de la problemática de bajo nivel y conseguir definir los requisitos con la mayor precisión posible.

De este modo, una vez identificados los actores, el siguiente paso es reconocer las acciones, es decir, casos de uso, que este puede llevar a cabo cuando utilice el sistema que se pretende desarrollar. Siguiendo una especificación informal del flujo de ejecución del algoritmo, podemos identificar las siguientes etapas. En el primer paso, el sistema debe permitirle escoger al usuario el grafo y registro (log) con los que se efectuará la búsqueda (**seleccionar origen del grafo y log**), además de **especificar el umbral** que se utilizará en el algoritmo. A continuación, el usuario podrá **iniciar el algoritmo** y proceder a su detención (**detener algoritmo**) cuando lo crea oportuno. Por último, el usuario podrá **visualizar los resultados** generados por el algoritmo con un conjunto de casos de uso asociados que le permiten realizar diversas acciones.

Debido al número de casos de uso existentes se ha decidido dividir los diagramas de casos de uso en dos subsistemas. Un subsistema es un sistema perteneciente al proyecto, de tamaño inferior, que engloba un conjunto de casos de uso relacionados entre sí. En este contexto, se han identificado los subsistemas de *(i)* 'visualización de resultados' (VR), que engloba los casos de uso referentes a la visualización de los resultados, y (ii) el subsistema de 'interacción con el algoritmo' (IA) en el que se engloban los casos de uso que tratan acciones relacionadas con el algoritmo.

4.1.2.1. Subsistema de interacción con el algoritmo

En este subapartado se mostrará el diagrama de casos de uso referente al subsistema de interacción con el algoritmo, así como la definición formal de los casos de uso. La notación utilizada para la identificación de los casos de uso de este subsistema será CU_IA-*N*.

En la Figura 4.1 se muestra el diagrama de casos de uso del subsistema de interacción con el algoritmo, en el que se han resaltado con color los casos de uso que tienen características en común con la finalidad de mejorar la comprensión del mismo. En este caso se muestra en verde los casos

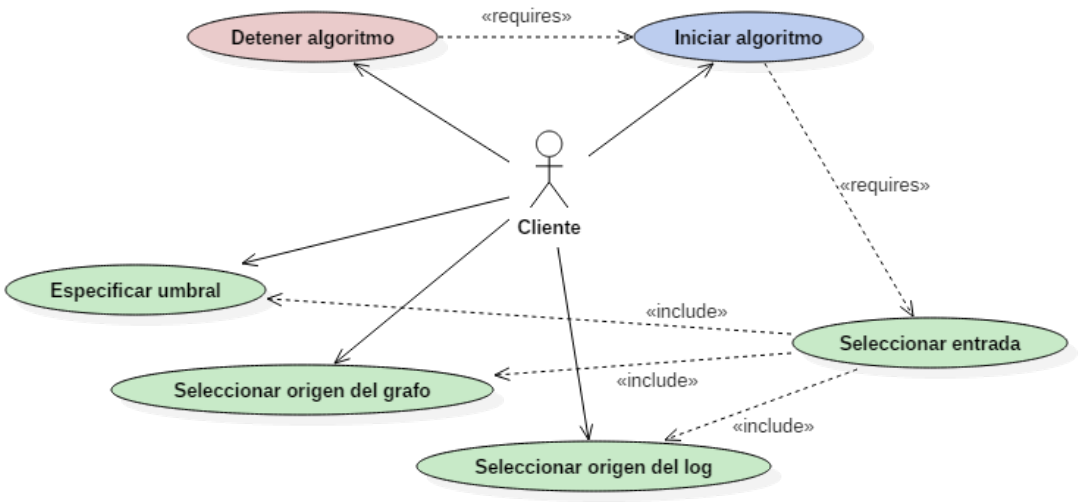

Figura 4.1: Diagrama UML con los casos de uso del subsistema de interacción con el algoritmo.

de uso en los que se selecciona la entrada con un tono verde, el que inicia el algoritmo en azul, y en rojo la detención del mismo.

A continuación se muestra la especificación formal de los casos de uso del subsistema de interacción con el algoritmo.

ID	CU_IA-01
Nombre	Seleccionar origen del grafo
Descripción	El usuario selecciona el archivo con la especificación del grafo.
Prioridad	Vital

ID	CU_IA-02
Nombre	Seleccionar origen del log
Descripción	El usuario selecciona el archivo con las trazas de ejecución.
Prioridad	Vital

ID	CU_IA-03
Nombre	Especificar umbral
Descripción	El usuario especifica el valor del umbral que se utilizará para la ejecución del algoritmo.
Prioridad	Vital

ID	CU_IA-04
Nombre	Seleccionar entrada
Descripción	El usuario especifica el conjunto de entradas para el algoritmo, formadas por el registro, modelo del grafo y umbral.
Prioridad	Vital

ID	CU_IA-05
Nombre	Iniciar algoritmo
Descripción	El usuario inicia el algoritmo.
Prioridad	Vital

ID	CU_IA-06
Nombre	Detener algoritmo
Descripción	El usuario detiene el algoritmo.
Prioridad	Deseable

4.1.2.2. Subsistema de visualización de los resultados

En este subapartado se mostrará el diagrama de casos de uso del subsistema relacionado con la visualización de los resultados, además de la definición formal de los casos de uso asociados a dicho subsistema. La notación utilizada para la identificación de los casos de uso de este subsistema será CU_VR-*N*, donde *N* hace referencia al número del caso de uso.

En la Figura 4.2 se muestra el diagrama de casos de uso del subsistema de interacción con el usuario, en este se ha resaltado con colores los casos de uso que suponen funcionalidades pertenecientes a un mismo conjunto, con el objetivo de facilitar la comprensión y futura identificación de los casos de uso. En este caso se muestran en verde los casos de uso relacionados con la ordenación de patrones, con color azul los que representan funcionalidades de listado de patrones, y en rojo las que modifican la visualización del grafo.

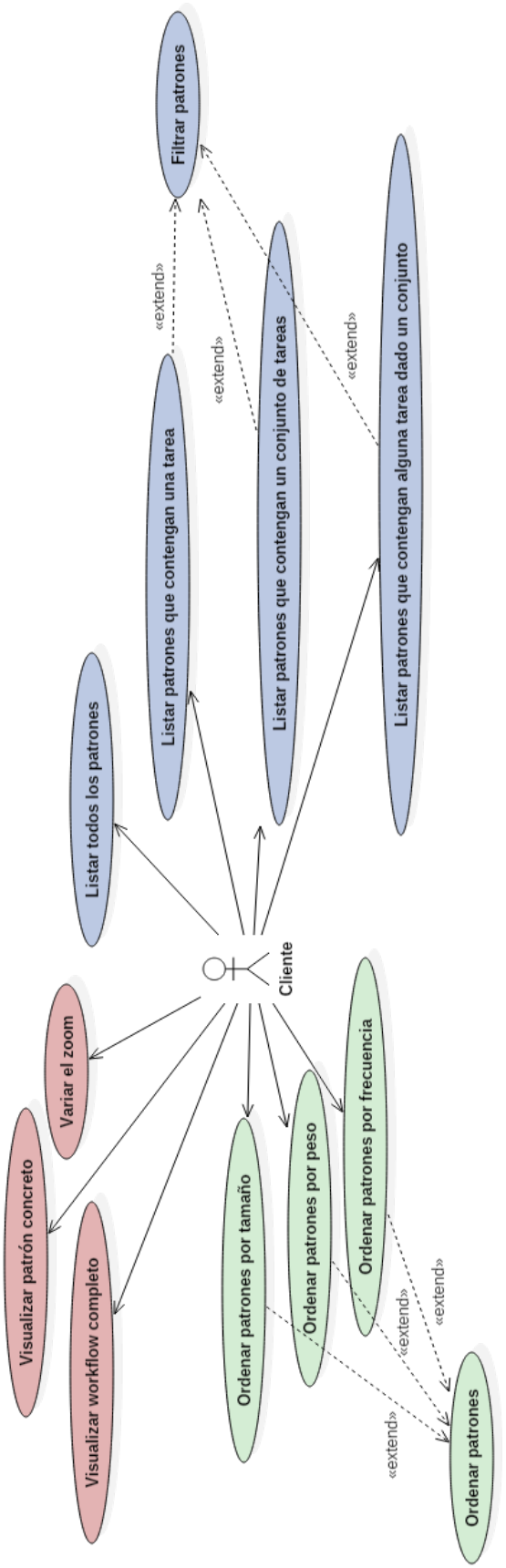

Figura 4.2: Diagrama UML con los casos de uso del subsistema de visualización de los resultados.

A continuación se muestra una descripción simple y formal de los casos de uso. De estos se extraerán los requisitos del sistema posteriormente.

ID	CU_VR-01
Nombre	Visualizar workflow completo
Descripción	El usuario visualiza el workflow completo en la interfaz.
Prioridad	Vital

ID	CU_VR-02
Nombre	Visualizar patrón concreto
Descripción	El usuario visualiza el patrón concreto dentro del grafo en la interfaz.
Prioridad	Vital

ID	CU_VR-03
Nombre	Variar el zoom
Descripción	El usuario aumenta o disminuye el zoom del grafo.
Prioridad	Importante

ID	CU_VR-04
Nombre	Listar todos los patrones
Descripción	El usuario visualiza la lista de todos los patrones resultantes de la ejecución del algoritmo.
Prioridad	Vital

ID	CU_VR-05
Nombre	Listar patrones que contengan una tarea
Descripción	El usuario visualiza la lista de los patrones resultado que tengan una tarea determinada.
Prioridad	Importante

ID	CU_VR-06
Nombre	Listar patrones que contengan un conjunto de tareas
Descripción	El usuario visualiza la lista de patrones resultado que contienen un conjunto de tareas.
Prioridad	Importante

ID	CU_VR-07
Nombre	Listar patrones que contengan alguna tarea dado un conjunto
Descripción	El usuario visualiza la lista de patrones resultado que contengan alguna tarea dado un subconjunto.
Prioridad	Deseado

ID	CU_VR-08
Nombre	Filtrar patrones
Descripción	El usuario puede filtrar los patrones resultados con diferentes métodos.
Prioridad	Vital

ID	CU_VR-09
Nombre	Ordenar patrones por tamaño
Descripción	El usuario visualiza los patrones ordenados por tamaño de forma ascendente o descendente.
Prioridad	Importante

ID	CU_VR-10
Nombre	Ordenar patrones por peso
Descripción	El usuario visualiza los patrones ordenados por peso de forma ascendente o descendente.
Prioridad	Deseado

ID	CU_VR-11
Nombre	Ordenar patrones por frecuencia
Descripción	El usuario visualiza los patrones ordenados por frecuencia de forma ascendente o descendente.
Prioridad	Vital

ID	CU_VR-12
Nombre	Ordenar patrones
Descripción	El usuario puede seleccionar el orden que siguen los patrones del resultado.
Prioridad	Vital

4.1.3. Especificación de requisitos

Una vez definidos los casos de uso, se procede a un análisis de los mismos en el que se extraerán los diferentes requisitos del sistema. En esta sección, los requisitos se clasificarán y describirán de forma completa según el estándar IEEE830 [6], haciendo una distinción entre requisitos de información (RI-N), requisitos funcionales y requisitos no funcionales (RNF-N).

Debido a la separación en subsistemas que se ha hecho en la identificación de casos de uso, se optará por realizar la misma distinción para la identificación de los requisitos funcionales, generando así requisitos funcionales del subsistema de visualización de los resultados (RF_VR-N) y requisitos funcionales del subsistema de interacción con el algoritmo (RF_IA-N).

Las descripciones de los requisitos varían en función del tipo de los mismos, habiendo campos comunes como el identificador y nombre, la descripción, o la criticidad del mismo. Esta última será clasificada siguiendo una escala con los siguientes valores:

– **Vital**: Fundamental para que el proyecto se considere completado.

– **Importante**: Aumenta la calidad del producto y su funcionamiento, pero no es indispensable.

– **Deseable**: Extiende la definición del producto.

4.1.3.1. Requisitos de información

Los requisitos de información especifican los datos que debe almacenar el sistema para el cumplimiento, parcial o total, de requisitos de otros tipos, a continuación se detallan los requisitos de información referentes a este proyecto:

ID	RI-01
Nombre	Grafos analizados
Requisitos asociados	RF_IA-10 Reutilización de resultados previamente obtenidos
Descripción	El sistema debe almacenar los grafos o workflows que se carguen.
Datos específicos	- Identificador del workflow - Fichero del workflow - ID o Alias asignado al workflow
Criticidad	Importante

ID	RI-02
Nombre	Logs analizados
Requisitos asociados	RF_IA-10 Reutilización de resultados previamente obtenidos
Descripción	El sistema deberá almacenar los logs con las trazas de ejecución que se carguen
Datos específicos	- Identificador de las trazas - Trazas
Criticidad	Importante

ID	RI-03
Nombre	Umbrales utilizados
Requisitos asociados	RF_IA-10 Reutilización de resultados previamente obtenidos
Descripción	El sistema deberá almacenar los umbrales utilizados en los análisis.
Datos específicos	- Valor del umbral
Criticidad	Importante

ID	RI-04
Nombre	RF_IA-10 Reutilización de resultados previamente obtenidos
Requisitos asociados	Resultados obtenidos
Descripción	El sistema deberá almacenar los resultados obtenidos en cada ejecución.
Datos específicos	- ID del workflow - ID del log - Valor del umbral utilizado - Patrones resultados
Criticidad	Importante

4.1.3.2. Requisitos funcionales

Un requisito funcional define, en esencia, una función que un sistema debe ofrecer, definiendo el comportamiento interno del software: cálculos, detalles técnicos, manipulación de datos y otras funcionalidades específicas que muestran cómo los casos de uso serán llevados a la práctica.

Subsistema de interacción con el algoritmo

En este subsistema se incluyen los requisitos relacionados con la interacción del usuario con el algoritmo, y los requisitos del propio algoritmo.

ID	RF_IA-01	
Nombre	Seleccionar origen del grafo	
Requisitos asociados	RF_IA-04 Iniciar algoritmo	
Descripción	El sistema deberá permitir al usuario seleccionar o cargar el archivo del grafo o workflow en formato *.hn*.	
Precondición	Se debe estar en la pantalla para cargar otro modelo.	
Secuencia normal	Paso	Acción
	1	El usuario selecciona la casilla para cargar otro workflow.
	2	El usuario selecciona el archivo del workflow deseado en su equipo.
Postcondición	Queda seleccionado el archivo escogido por el usuario como origen del workflow a analizar.	
Criticidad	Vital	
Criterio de validación	Se considerará cumplido cuando el usuario pueda seleccionar un archivo como origen del grafo, y este se cargue correctamente.	

ID	RF_IA-02	
Nombre	Seleccionar origen del log	
Requisitos asociados	RF_IA-04 Iniciar algoritmo	
Descripción	El sistema deberá permitir al usuario seleccionar o cargar el archivo del log en formato .xes.	
Precondición	Se debe estar en la pantalla para cargar otro modelo.	
Secuencia normal	Paso	Acción
	1	El usuario selecciona la casilla para cargar otro log.
	2	El usuario selecciona el archivo del log deseado en su equipo.
Postcondición	Queda seleccionado el archivo escogido por el usuario como origen del log a analizar.	
Criticidad	Vital	
Criterio de validación	Se considerará cumplido cuando el usuario pueda seleccionar un archivo como origen del log, y este se cargue correctamente.	

ID	RF_IA-03	
Nombre	Especificar umbral de frecuencia	
Requisitos asociados	RF_IA-04 Iniciar algoritmo	
Descripción	El sistema deberá permitir al usuario definir el umbral de frecuencia que desee, siendo un número con un decimal entre 0 y 100.	
Precondición	Se debe estar en la pantalla de cargar otro modelo, o en la de definir otro umbral.	
Secuencia normal	Paso	Acción
	1	El usuario introduce el valor del nuevo umbral
Postcondición	Queda fijado el nuevo umbral, que se utilizará en la siguiente ejecución.	
Criticidad	Vital	
Criterio de validación	Se considerará cumplido cuando el usuario pueda especificar un umbral entre 0 y 100 con dos decimales.	

ID	RF_IA-04	
Nombre	Iniciar algoritmo	
Requisitos asociados	RF_IA-01 Seleccionar origen del grafo RF_IA-02 Seleccionar origen del log RF_IA-03 Especificar umbral de frecuencia	
Descripción	El sistema deberá permitir al usuario iniciar el algoritmo.	
Precondición	El algoritmo no puede estar en ejecución.	
Secuencia normal	Paso	Acción
	1	El usuario pulsa el botón de iniciar algoritmo desde la pantalla de carga de otro modelo.
Secuencia alternativa 1	Paso	Acción
	1	El usuario pulsa el botón de iniciar algoritmo desde la pantalla de redefinición del umbral.
Postcondición	Se inicia el algoritmo.	
Excepciones secuencia normal	Paso	Acción
	1	El grafo introducido no tiene el formato correcto, se avisa al usuario y no se inicia el algoritmo.
	1	El log introducido no tiene el formato correcto, se avisa al usuario y no se inicia el algoritmo.
	1	El umbral introducido es incorrecto, se avisa al usuario y no se inicia el algoritmo.
Excepciónes secuencia alternativa 1	Paso	Acción
	1	El umbral introducido es incorrecto, se avisa al usuario y no se inicia el algoritmo.
Criticidad	Vital	
Criterio de validación	Se considerará este requisito como aceptado cuando el usuario pueda iniciar el algoritmo desde la interfaz gráfica si los argumentos son correctos, en caso contrario se avisa al usuario.	

ID	RF_IA-05	
Nombre	Detener algoritmo	
Requisitos asociados	RF_IA-04 Iniciar algoritmo	
Descripción	El sistema deberá permitir al usuario detener el algoritmo una vez este haya iniciado.	
Precondición	El algoritmo debe estar en ejecución.	
Secuencia normal	Paso	Acción
	1	El usuario pulsa el botón de detener algoritmo desde la pantalla de espera.
Postcondición	Se para el algoritmo y se vuelve al inicio de la aplicación.	
Criticidad	Deseable	
Criterio de validación	Se considerará este requisito como aceptado cuando el usuario pueda detener el algoritmo desde la interfaz gráfica una vez este se haya iniciado, pero antes de que finalice.	

ID	RF_IA-06	
Nombre	Detectar patrones máximos que superen el umbral	
Descripción	El sistema deberá generar como patrones resultado los máximos cuya frecuencia sea mayor o igual que el umbral.	
Precondición	Se obtiene la lista de resultados con todos los patrones frecuentes.	
Secuencia normal	Paso	Acción
	1	El sistema aplica un filtrado a los patrones.
Postcondición	El conjunto resultado no posee patrones que se encuentran como subpatrones de otros.	
Criticidad	Vital	
Criterio de validación	Se considerará este requisito como cumplido cuando los patrones que devuelva el algoritmo superen el umbral definido, y no haya un patrón contenido en otro.	

ID	RF_IA-07	
Nombre	Detección de patrones con bucles	
Requisitos asociados	RF_IA-08 Detección de arcos de inicio y fin de bucle RF_IA-09 Generación de combinaciones	
Descripción	El sistema deberá contemplar patrones con bucles.	
Precondición	Dado un umbral, un grafo con bucles y sus registros correspondientes.	
Secuencia normal	Paso	Acción
	1	El sistema realiza el análisis.
Postcondición	El resultado devuelto es el correcto, soportando patrones con bucles.	
Criticidad	Importante	
Criterio de validación	Se considerará este requisito como cumplido cuando el algoritmo funcione correctamente para grafos con bucles.	

ID	RF_IA-08	
Nombre	Detección de arcos de inicio y fin de bucle	
Requisitos asociados	RF_IA-07 Detección de patrones con bucles	
Descripción	El sistema deberá detectar los arcos de inicio y fin de bucle.	
Precondición	Dado un grafo con bucles.	
Secuencia normal	Paso	Acción
	1	El algoritmo inicia el análisis.
Postcondición	El resultado es el grafo con los arcos de inicio y fin de bucle detectados.	
Criticidad	Importante	
Criterio de validación	Se considerará cumplido este requisito cuando el algoritmo sea capaz de detectar los patrones de inicio y fin de bucle de los grafos que posean bucles.	

ID	RF_IA-09	
Nombre	Generación de combinaciones	
Requisitos asociados	RF_IA-07 Detección de patrones con bucles	
Descripción	El sistema deberá poder generar las combinaciones de un patrón, esto son los diferentes caminos que se pueden tomar en el patrón.	
Precondición	Pasado un patrón.	
Secuencia normal	Paso	Acción
	1	El algoritmo genera sus combinaciones.
Postcondición	El resultado son las combinaciones del patrón.	
Criticidad	Importante	
Criterio de validación	Se considerará este requisito como cumplido cuando el algoritmo sea capaz de generar las combinaciones de los patrones.	

ID	RF_IA-10	
Nombre	Reutilización de resultados previamente obtenidos	
Requisitos asociados	RI-01 Grafos analizados RI-02 Logs analizados RI-03 Umbrales utilizados RI-04 Resultados obtenidos	
Descripción	El sistema deberá reutilizar los resultados de una ejecución cuando se vuelva a lanzar una con los mismos parámetros.	
Precondición	Un usuario realiza una petición ya existente.	
Secuencia normal	Paso	Acción
	1	El usuario hace una petición
	2	El sistema detecta que la petición ya se ha realizado con anterioridad.
Postcondición	Se muestra el resultado del análisis.	
Criticidad	Deseado	
Criterio de validación	Se considerará validado este requisito cuando, ante una petición de ejecución que ya se ha realizado, el sistema recupera la solución en lugar de volver a ejecutar el algoritmo.	

Subsistema de visualización de los resultados

ID	RF_VR-01	
Nombre	Visualizar workflow completo	
Requisitos asociados	RF_VR-02 Visualizar patrón concreto RF_VR-03 Variar el zoom	
Descripción	El sistema debe permitir al usuario visualizar el workflow completo, resaltando el patrón seleccionado en otro color.	
Precondición	El algoritmo se ha lanzado y se visualiza el modelo en la pantalla.	
Secuencia normal	Paso	Acción
	1	El usuario selecciona la opción para visualizar el grafo completo.
Postcondición	La vista realiza el ajuste necesario para que se visualice el grafo completo.	
Criticidad	Vital	
Criterio de validación	Se considerará cumplido este algoritmo cuando el usuario pueda visualizar el grafo completo a su petición.	

ID	RF_VR-02	
Nombre	Visualizar patrón concreto	
Descripción	El sistema deberá permitir al usuario visualizar un patrón concreto, esto es, ampliar la vista automáticamente enfocando al patrón que se seleccione.	
Precondición	El algoritmo se ha lanzado y se visualiza el modelo en la pantalla.	
Secuencia normal	Paso	Acción
	1	El usuario selecciona un patrón de la lista de resultados.
Postcondición	Se hace zoom sobre el patrón seleccionando viéndolo en pantalla.	
Criticidad	Vital	
Criterio de validación	Se considerará este requisito como validado cuando el usuario pueda seleccionar un patrón y la aplicación haga zoom automáticamente hacia el mismo.	

ID	RF_VR-03	
Nombre	Variar el zoom	
Descripción	El sistema deberá permitir que el usuario pueda aumentar y disminuir el zoom sobre el workflow.	
Precondición	El algoritmo se ha lanzado y se visualiza el modelo en la pantalla.	
Secuencia normal	Paso	Acción
	1	El usuario selecciona aumentar o disminuir el zoom.
Postcondición	El zoom sobre el modelo visualizado se aumenta o disminuye dependiendo de la acción que se realice.	
Criticidad	Importante	
Criterio de validación	Se considerará este requisito como validado cuando el usuario pueda realizar tanto aumento como disminución del zoom en el modelo mostrado en la interfaz gráfica.	

ID	RF_VR-04	
Nombre	Listar todos los patrones	
Requisitos asociados	RF_VR-05 Listar patrones que contengan una tarea RF_VR-06 Listar patrones que contengan un conjunto de tareas RF_VR-07 Listar patrones que contengan alguna tarea dado un conjunto	
Descripción	El sistema deberá permitir al usuario la visualización de todos los patrones resultado por medio de una lista en la que se enumeren.	
Precondición	El algoritmo se ha lanzado y ha terminado devolviendo un conjunto de patrones como resultado.	
Secuencia normal	Paso	Acción
	1	El usuario selecciona la opción de visualizar todos los resultados.
Postcondición	En la lista de resultados se muestran todos los patrones resuntantes.	
Criticidad	Vital	
Criterio de validación	Se considerará este requisito como cumplido cuando el usuario pueda seleccionar la opción de listado de todos los patrones y aparezcan en la lista todos los que forman el conjunto de resultados.	

ID	RF_VR-05	
Nombre	Listar patrones que contengan una tarea	
Descripción	El sistema deberá permitir al usuario escoger una tarea y mostrar los patrones resultado que contienen a la tarea seleccionada.	
Precondición	El algoritmo se ha lanzado y ha terminado devolviendo un conjunto de patrones como resultado.	
Secuencia normal	Paso	Acción
	1	El usuario selecciona la opción de filtrar por tareas
	2	Selecciona una tarea
Postcondición	Se filtran los patrones apareciendo solo los que contengan la tarea seleccionada.	
Criticidad	Importante	
Criterio de validación	Se considerará validado este requisito cuando el usuario pueda seleccionar una tarea y se listen los patrones resultantes que contengan dicha tarea.	

ID	RF_VR-06	
Nombre	Listar patrones que contengan un conjunto de tareas	
Descripción	El sistema deberá permitir al usuario escoger un conjunto de tareas y mostrar los patrones resultado que contienen todas las tareas seleccionadas.	
Precondición	El algoritmo se ha lanzado y ha terminado devolviendo un conjunto de patrones como resultado.	
Secuencia normal	Paso	Acción
	1	El usuario selecciona la opción de filtrar por tareas
	2	Selecciona un conjunto de ellas
Postcondición	Se filtran los patrones apareciendo solo los que contengan el conjunto de tareas.	
Criticidad	Importante	
Criterio de validación	Se considerará este requisito como cumplido cuando el usuario pueda seleccionar un conjunto de tareas y la interfaz liste los patrones resultantes que contengan el conjunto completo.	

ID	RF_VR-07	
Nombre	Listar patrones que contengan alguna tarea dado un conjunto	
Descripción	El sistema deberá permitir al usuario escoger un conjunto de tareas y mostrar los patrones resultado que contienen alguna de las tareas seleccionadas.	
Precondición	El algoritmo se ha lanzado y ha terminado devolviendo un conjunto de patrones como resultado.	
Secuencia normal	Paso	Acción
	1	El usuario selecciona la opción de filtrar por tareas
	2	Selecciona un conjunto de ellas
Postcondición	Se filtran los patrones apareciendo solo los que contengan alguna tarea del conjunto.	
Criticidad	Deseado	
Criterio de validación	Se considerará este requisito como cumplido cuando el usuario pueda seleccionar un conjunto de tareas y la interfaz liste los patrones resultantes que contengan alguna tarea del conjunto.	

ID	RF_VR-08	
Nombre	Ordenar patrones por tamaño	
Requisitos asociados	RF_VR-09 Ordenar patrones por peso RF_VR-10 Ordenar patrones por frecuencia	
Descripción	El sistema deberá permitir al usuario ordenar la lista de los patrones resultado por tamaño, de forma ascendente y descendente.	
Precondición	El algoritmo se ha lanzado y ha terminado devolviendo un conjunto de patrones como resultado.	
Secuencia normal	Paso	Acción
	1	El usuario selecciona la opción de ordenar por tamaño
Postcondición	La lista de patrones se ordena por tamaño	
Criticidad	Importante	
Criterio de validación	Se considerará validado este requisito cuando el usuario selecciones la opción de ordenar los patrones por tamaño y la lista se ordene.	

ID	RF_VR-09	
Nombre	Ordenar patrones por peso	
Requisitos asociados	RF_VR-08 Ordenar patrones por tamaño RF_VR-10 Ordenar patrones por frecuencia	
Descripción	El sistema deberá permitir al usuario ordenar la lista de los patrones resultado por peso, de forma ascendente y descendente. El peso se obtiene a partir de la siguiente fórmula: $Peso = N_a + 2N_t$ Donde N_a es el número de arcos y N_t el número de tareas	
Precondición	El algoritmo se ha lanzado y ha terminado devolviendo un conjunto de patrones como resultado.	
Secuencia normal	Paso	Acción
	1	El usuario selecciona la opción de ordenar por peso
Postcondición	La lista de patrones se ordena por peso	
Criticidad	Deseable	
Criterio de validación	Se considerará este requisito como cumplido cuando el usuario selecciones la opción de ordenar los patrones por peso y la lista se ordene.	

ID	RF_VR-10	
Nombre	Ordenar patrones por frecuencia	
Requisitos asociados	RF_VR-08 Ordenar patrones por tamaño RF_VR-09 Ordenar patrones por peso	
Descripción	El sistema deberá permitir al usuario ordenar la lista de los patrones resultado por frecuencia, de forma ascendente y descendente.	
Precondición	El algoritmo se ha lanzado y ha terminado devolviendo un conjunto de patrones como resultado.	
Secuencia normal	Paso	Acción
	1	El usuario selecciona la opción de ordenar por frecuencia
Postcondición	La lista de patrones se ordena por frecuencia	
Criticidad	Vital	
Criterio de validación	Se considerará validado este requisito cuando el usuario selecciones la opción de ordenar los patrones por frecuencia y la lista se ordene.	

4.1.3.3. Requisitos no funcionales

Un requisito no funcional especifica criterios que pueden usarse para juzgar la operación de un sistema en lugar de sus comportamientos específicos, es decir, rendimiento, disponibilidad tiempos de respuesta, capacidad de almacenamiento, etc.

ID	RNF-01
Nombre	Interfaz gráfica independiente del motor
Descripción	El algoritmo debe ser implementado de forma separada a la interfaz, debido a que es posible que en un futuro se cambie la interacción con el cliente.
Criticidad	Vital
Criterio de validación	Se considerará cumplido este requisito si el algoritmo se desarrolla independientemente de la interfaz, de forma que pueda ser llamado sin la necesidad de esta última.

ID	RNF-02
Nombre	Facilidad de pruebas
Descripción	El desarrollo, en especial del algoritmo, se realizará con la implementación de excepciones para facilitar la detección de errores durante las etapas de pruebas.
Criticidad	Vital
Criterio de validación	Se considerará el requisito como validado si la implementación, en especial del algoritmo, contiene excepciones que se lanzan en casos que no deberían darse, facilitando así la identificación de los errores que puedan surgir.

ID	RNF-03
Nombre	Multiplataforma
Descripción	El algoritmo debe ser desarrollado en un lenguaje que permita la ejecución sobre cualquier tipo de plataforma, con mayor prioridad en Linux y Windows.
Criticidad	Importante
Criterio de validación	Se considerará como cumplido este requisito si el desarrollo del algoritmo se realiza en un lenguaje no dependiente de la plataforma sobre la que se ejecuta.

ID	RNF-04
Nombre	Extensibilidad
Descripción	El código debe estar diseñado de forma que sea posible efectuar, de forma simple, un cambio del algoritmo para analizar la frecuencia de un patrón o la detección de los bucles.
Criticidad	Importante
Criterio de validación	Se considerará validado este requisito cuando el diseño del algoritmo integre el patrón Strategy haciendo que el cambio del mismo sea lo menos costo posible.

ID	RNF-05
Nombre	Documentación
Descripción	La documentación deberá ser lo más clara posible, priorizando la simplicidad siempre y cuando sea posible.
Criticidad	Importante
Criterio de validación	Se considerará este requisito como cumplido si la documentación carece de apartados o párrafos que no aportan ningún contenido necesario.

ID	RNF-06
Nombre	Interfaz usable
Descripción	La interfaz de usuario diseñada deberá ser usable para facilitar la adaptación de los usuarios, cumpliendo con el mayor número de principios de Nielsen posibles.
Criticidad	Importante
Criterio de validación	Se considerará como validado este requisito cuando supere un cuestionario SUS realizado por 3 usuarios.

ID	RNF-07
Nombre	Seguridad en la aplicación
Descripción	El sistema debe implementar medidas de seguridad para que los usuarios no puedan atacar la aplicación.
Criticidad	Importante
Criterio de validación	Se considerará superado este requisito cuando se implementen medidas contra ataques por parte de los usuarios de inyecciones SQL. Además de comprobar el *MIME/type* de los archivos subidos por el usuario para verificar que el formato es correcto.

4.1.3.4. Matriz de trazabilidad

Para finalizar con el análisis de requisitos, en la Tabla 4.1 se muestra la matriz de trazabilidad que representa las correspondencias entre los casos de uso y los requisitos funcionales del sistema.

	CU_-IA-01	CU_-IA-02	CU_-IA-03	CU_-IA-04	CU_-IA-05	CU_-IA-06	CU_-VR-01	CU_-VR-02	CU_-VR-03	CU_-VR-04	CU_-VR-05	CU_-VR-06	CU_-VR-07	CU_-VR-08	CU_-VR-09	CU_-VR-10	CU_-VR-11	CU_-VR-12
RF_IA-01	X			X														
RF_IA-02		X		X														
RF_IA-03			X	X														
RF_IA-04					X													
RF_IA-05						X												
RF_IA-06					X													
RF_IA-07					X													
RF_IA-08					X													
RF_IA-09					X													
RF_IA-10					X													
RF_VR-01							X		X									
RF_VR-02								X	X									
RF_VR-03									X									
RF_VR-04										X								
RF_VR-05											X							
RF_VR-06												X						
RF_VR-07													X	X				
RF_VR-08															X			X
RF_VR-09																X		X
RF_VR-10																	X	X

Tabla 4.1: Matriz de trazabilidad que enfrenta los requisitos funcionales contra los casos de uso.

Capítulo 5

Gestión del proyecto

Cuando se realiza un proyecto de grandes dimensiones, este debe ser gestionado de forma correcta y definida, reduciendo así la probabilidad de cometer errores durante el desarrollo del mismo. Esta gestión consiste en la aplicación de técnicas, habilidades y conocimientos, con el objetivo de iniciar, supervisar y controlar los proyectos a lo largo de su ciclo de vida, y de este modo cumplir con el alcance del proyecto dentro de las restricciones temporales, económicas y relativas a la calidad del producto.

Considerando ésta como una fase clave para asegurar el éxito del producto, en este capítulo se describen: (i) el análisis y gestión de riesgos, (ii) la gestión de la configuración, (iii) la selección de la metodología de desarrollo, (iv) la planificación temporal (v) la gestión de costes y (vi) el plan de las comunicaciones.

5.1. Gestión de riesgos

Siguiendo la definición dada por el PMBOK:

> *Un riesgo de un proyecto es un evento o condición inciertos que, si se produce, tiene un efecto positivo o negativo sobre al menos un objetivo del proyecto, como tiempo, coste, alcance o calidad.*

Las causas de que un riesgo se produzca pueden ser más de una, y en caso de producirse, los impactos también pueden ser diversos. Sabiendo esto se puede deducir que los riesgos de un proyecto son algo que no se debe subestimar, ya que un riesgo no previsto puede hacer que el proyecto se tenga que cancelar, o que el tiempo/coste se eleve considerablemente. Para esto se debe realizar una correcta gestión de riesgos, en la que se identifiquen todos los que pongan en peligro algún objetivo del proyecto, diseñando planes para evitarlos, mitigarlos, aceptarlos o transferirlos. Concretamente, cada una de estas posibles respuestas ante un riesgo se pueden definir como:

- *Evitar un riesgo* consiste en cambiar la planificación del proyecto para eliminar la amenaza.
- *Mitigar* implica realizar o planificar acciones para reducir la probabilidad o el impacto que un riesgo puede tener sobre los activos a los que afecta.

- *Aceptar un riesgo* es una estrategia común en la gestión de proyectos, en casos en los que no se puede cambiar el plan de proyecto y no se sabe cómo reducir la probabilidad del riesgo, la solución puede ser aceptarlo y asumir que puede ocurrir. Esta aceptación puede ser pasiva (no hacer nada) o activa (reservar un fondo para contingencias que se utilizará si el riesgo aparece).

- *Transferir un riesgo* consiste en derivar la responsabilidad del mismo a un tercero. Esto no elimina el riesgo, sino que traslada a otra entidad el trabajo de tratarlo.

En el presente proyecto, al ser desarrollado por un solo trabajador, se tratará de evitar la transferencia como respuesta a un riesgo, ya que es una estrategia propensa a conllevar costes adicionales. Por el mismo motivo, la aceptación de un riesgo se aplicará solo cuando esta sea la única estrategia posible a seguir.

El primer paso de esta fase es la identificación de los riesgos, en esta etapa se deberán identificar el mayor número de riesgos posibles, para luego analizar su importancia, y establecer estrategias para los más importantes en caso de no poder tratar todos.

5.1.1. Especificación de riesgos

A continuación se muestran las especificaciones formales de los riesgos, en la que se muestra una descripción de los riesgos, junto a la probabilidad de ocurrencia, el impacto que tendrán en caso de ocurrir, y los planes diseñados para cada uno de ellos. Para la especificación de la probabilidad e impacto se utilizará una escala con los siguientes valores: muy bajo, bajo, medio, alto, muy alto.

ID	RIS-01
Nombre	Retraso en la planificación
Descripción	Debido a la inexperiencia del desarrollador del proyecto, es posible que las planificaciones se realicen erróneamente, con lo que el proyecto presentaría retrasos.
Probabilidad de ocurrencia	Alta
Impacto	Muy Alto
Plan de prevención	Establecer reuniones periódicas para que los tutores verifiquen el avance en el proyecto.
Plan de contingencia	Volver a definir el alcance del proyecto, eliminando objetivos no vitales.

ID	RIS-02
Nombre	Dificultad de comprensión del ámbito de aplicación
Descripción	Es posible que el desarrollador del proyecto no sepa desenvolverse con la fluidez necesaria y presente problemas con la comprensión del algoritmo y de otras referencias.
Probabilidad de ocurrencia	Media
Impacto	Muy alto
Plan de contingencia	Solicitar reuniones con los tutores para resolución de dudas y explicación de conceptos.

ID	RIS-03
Nombre	Cambio en los requisitos
Descripción	Además de la probabilidad inherente a todo proyecto, al ser uno de investigación es posible que los requisitos se vean alterados en función de las particularidades del algoritmo a implementar.
Probabilidad de ocurrencia	Alta
Impacto	Alto
Plan de prevención	Diseño modular y adaptable para que un cambio en los requisitos no suponga una gran refactorización del proyecto.
Plan de contingencia	Al aplicar una metodología ágil como la programación extrema este tipo de cambios están previstos, y son soportados sin problemas.

ID	RIS-04
Nombre	Inaplicabilidad del algoritmo inicial en la práctica
Descripción	El algoritmo del que se parte está especificado de forma teórica, pero no se tiene seguridad de su éxito en la práctica. Cabe la posibilidad de que una vez implementado, los resultados no sean los esperados, y no haya posibilidad de mejora ni aplicación.
Probabilidad de ocurrencia	Media
Impacto	Muy Alto
Plan de contingencia	Redefinición del alcance del proyecto estableciendo otro algoritmo como inicio del mismo.

ID	RIS-05
Nombre	Baja del desarrollador del proyecto
Descripción	Es posible que, por motivos físicos, económicos o mentales el único desarrollador del proyecto tenga que dejar de trabajar en él.
Probabilidad de ocurrencia	Baja
Impacto	Muy Alto
Plan de contingencia	Se aplazaría el proyecto hasta que el desarrollador vuelva a estar disponible. Si esto nunca fuese a ocurrir se asignaría a otro desarrollador.

ID	RIS-06
Nombre	Baja de los tutores
Descripción	Es posible que, por motivos económicos, físicos o mentales alguno de los tutores deje de trabajar en el proyecto.
Probabilidad de ocurrencia	Baja
Impacto	Bajo
Plan de prevención	Con objetivo de reducir el impacto se tratará de terminar lo antes posible las fases de alcance y diseño.
Plan de contingencia	En caso de que la baja sea de una parte de los tutores, se seguirá trabajando con los restantes. Si la baja es de todos, se iniciará un proceso de búsqueda de un tutor provisional.

ID	RIS-07
Nombre	Inutilización del equipo de trabajo
Descripción	Por una diversidad de causas, ya sean accidente o problemas que presente el equipo, este podría dejar de funcionar, haciendo que el avance en el proyecto se vea afectado.
Probabilidad de ocurrencia	Baja
Impacto	Muy Alto
Plan de contingencia	El desarrollador, al tratarse de un alumno de la USC, puede acceder a los equipos de la universidad para trabajar, por lo que en caso de fallo recurrirá a estas aulas.

ID	RIS-08
Nombre	Imposibilidad de comunicación interfaz-algoritmo eficiente
Descripción	Cabe la posibilidad de que no se encuentre una forma de interacción entre el algoritmo y la interfaz que no suponga un tiempo de espera excesivo, debido a la cantidad de información que se tiene que intercambiar.
Probabilidad de ocurrencia	Muy Baja
Impacto	Muy Alto
Plan de contingencia	En este caso la aceptación es pasiva, no se hace nada y en caso de ocurrir el rendimiento de la aplicación será bajo.

ID	RIS-09
Nombre	Excesiva dificultad de representar grafos en la interfaz
Descripción	En caso de que no funcionen las posibilidades que se tienen pensadas la representación de los grafos en la interfaz puede verse complicada altamente.
Probabilidad de ocurrencia	Baja
Impacto	Alto
Plan de prevención	En caso de no encontrar un modo de representación seguro se estudiaran varias opciones, para tener preparadas formas alternativas en caso de error.
Plan de contingencia	Se redefinirá el alcance del proyecto para ajustar el tiempo, eliminando tareas secundarias de ser posible.

ID	RIS-10
Nombre	Pérdida del código y documentación
Descripción	Puede darse el caso de que el equipo de trabajo falle perdiéndose así la documentación avanzada hasta el momento.
Probabilidad de ocurrencia	Baja
Impacto	Muy Alto
Plan de prevención	Para reducir la probabilidad de perder por completo la documentación realizada se utilizará Dropbox, teniéndolo sincronizado también con el equipo personal del desarrollador.
Plan de contingencia	Replanificación y redefinición del alcance para ajustar el proyecto y poder rehacer lo perdido.

ID	RIS-11
Nombre	Mala usabilidad de la aplicación web
Descripción	Existe el riesgo de que el diseño de la interfaz desarrollada no tenga la suficiente usabilidad para que los usuarios la puedan utilizar sin problemas.
Probabilidad de ocurrencia	Media
Impacto	Muy Alto
Plan de prevención	Para reducir la probabilidad de ocurrencia de este riesgo se elaborará un test de usabilidad, comprobando así la facilidad de uso de la aplicación.

5.2. Gestión de la configuración

Como se explica en la *guía práctica de gestión de configuración* publicada por INTECO, la gestión de la configuración es un proceso que pretende establecer y mantener la integridad de los productos de trabajo estableciendo una serie de pautas:

1. Se deben identificar los elementos o productos que van a ser controlados, para poder acceder a ellos de forma eficaz y eficiente.

2. Debe existir un único procedimiento para el control de los cambios sobre los productos, es decir, no debe haber un método de control de cambios ambiguo que permita diferentes controles en un mismo caso.

3. Se debe llevar un informe o registro del estado de los productos, para saber en todo momento en qué estado están.

4. Deben establecerse auditorías de configuración, en las que se asegurará el cumplimiento de los requisitos por parte de los elementos de la configuración.

La gestión de la configuración es un proceso muy importante dentro de un proyecto, esta permite asegurar la integridad de los productos desarrollados reduciendo el riesgo de efectuar una entrega errónea. Con una correcta gestión de configuración se evitan sobreesfuerzos causados por problemas de integridad, proporciona la capacidad de controlar los cambios y garantiza que todo el equipo de desarrollo trabaja sobre la misma versión.

En este caso, al haber un solo desarrollador, el proceso para asegurar la integridad ante modificaciones por parte de los trabajadores se simplifica, puesto que solo es necesario almacenar las versiones de los documentos de forma eficiente.

Una mala gestión de configuración provoca, de una u otra forma, la pérdida de tiempo, dinero o recursos. El no establecer un control sobre los cambios y una buena identificación de los elementos de la configuración pueden derivar en la pérdida de versiones, la realización de un trabajo más de una vez debido a que la versión sobre la que se realizó no era la correcta, la no disponibilidad de la última versión de un elemento, etc.

5.2.1. Herramientas

En esta sección se realiza una introducción de las herramientas utilizadas para llevar a cabo los procesos relacionados con la gestión de la configuración, describiendo su uso en los mismos en el siguiente apartado.

Dropbox[1]: Es un servicio de almacenamiento de archivos en la nube, permite el alojamiento de ficheros en la web, sincronizándolos con todos los equipos que tengan la cuenta asignada.

Subversion (SVN)[2]: herramienta para el control de versiones basada en repositorio. SVN permite llevar un control de cambios teniendo la posibilidad de deshacer uno siempre que se desee.

LaTeX[3]: es un sistema de composición de textos de alta calidad, es un estándar de facto para la creación de documentos y publicaciones de carácter científico.

5.2.2. Descripción del soporte de las herramientas al proceso

Para la edición de los documentos asociados al proyecto se utilizará LaTeX y ya la documentación será elaborada con este sistema de composición, se optará por su uso desde el inicio del proyecto, forzando así a los desarrolladores a la adaptación y conocimiento del mismo desde un principio.

Para el control de versiones se utilizarán dos servicios. Por una parte, para los documentos y archivos relacionados con la memoria, el soporte ofrecido por una herramienta como Dropbox es suficiente. Se dispondrá de replicación de los archivos en la nube, además de una autosincronización en los equipos que lo tengan instalado, pudiendo acceder desde la web a versiones anteriores para su restauración.

En cuanto a los archivos referentes al código, se utilizará un sistema de control de versiones Subversion, que permitirá un control de los cambios más preciso y un sistema de unión en caso de conflictos. Estos últimos serán muy poco probables, ya que solo habrá un desarrollador trabajando en el proyecto. Para el trabajo con SVN se dispondrá de las extensiones que presentan los entornos de desarrollo como NetBeans[4] que integran por completo las funcionalidades de los sistemas de control de versiones.

5.2.3. Nomenclatura

Para un fácil reconocimiento de los elementos referentes a la documentación del proyecto se establecerá una nomenclatura común. Esta deberá permitir el reconocimiento del documento y versión de una forma cómoda y sin tener la necesidad de abrirlo o ejecutarlo.

```
<nombre_documento>_<aaMMdd>_v<version>.<extension>
```

nombre_documento representa el identificativo del documento, que permitirá su reconocimiento. **aaMMdd** se corresponde con la fecha de la última modificación, en formato año, mes,

[1] www.dropbox.com/
[2] subversion.apache.org
[3] www.latex-project.org/
[4] www.netbeans.org

día. **version** es la versión interna del documento. Y por último, **extension** sería la extensión del archivo que indica el formato.

Un ejemplo de nombre para un documento sería el siguiente:

memoriaTFG_150408_v1.3.tex

Este nombre se correspondería con la memoria del Trabajo fin de grado, modificada por última vez el 8 de Abril de 2015, y que se encuentra en su versión 1.3. Podemos saber que el formato del documento es LaTeX debido a su extensión (*tex*). Se ha de mencionar que dicha nomenclatura no será utilizada para los archivos relacionados con el código, puesto que se utilizará el sistema de gestión de versiones SVN, como ha sido explicado anteriormente.

5.3. Metodología de desarrollo

Como ya se ha mencionado con anterioridad, una de las mayores causas de los abandonos o cancelaciones en proyectos de software es el sobrecoste que producen, estando este presente también en gran número de proyectos terminados. Este sobrecoste está estrechamente ligado con el tiempo y la planificación del proyecto, que dependen altamente de la metodología de desarrollo a seguir. Por ello, la elección de una metodología correcta o adecuada al tipo de proyecto que se va a desarrollar es una de las fases más importantes. Una mala elección de la metodología a seguir para el desarrollo de un proyecto software puede provocar diversos problemas: desde el descontento del cliente por no poder disfrutar de una versión operativa hasta el final del desarrollo, hasta la imposibilidad de continuar con el proyecto por la naturaleza cambiante de los requisitos no contemplada en la metodología.

En proyectos de pequeña envergadura la existencia de metodología puede no ser un punto muy importante, pero cuando los proyectos poseen una complejidad elevada, esta elección debe permitir la estructuración, planificación y control del proceso de desarrollo favoreciendo la minimización de costes y maximización de la calidad del producto final, ofreciendo al equipo de desarrollo e interesados, información sobre el estado del proyecto en todo momento.

Queda aclarado que un proyecto software consta de varias etapas, por las que se pasa desde el inicio hasta el fin (siendo fin el desuso del mismo). Este paso por las diferentes etapas es lo que se entiende como *ciclo de vida*, que establece lo que se tiene que conseguir en cada estado, y qué se debe realizar después. Cabe destacar que una metodología no solo responde a esto, sino que también ofrece el cómo se deben obtener estos objetivos, siguiendo uno o varios modelos de ciclo de vida. En lugar de analizar y comparar las diferentes metodologías de desarrollo disponibles, se describirán las características de este proyecto y se presentará la metodología escogida para el desarrollo del mismo.

Este TFG se caracteriza por ser desarrollado de forma individual, lo que implica que no habrá equipos de trabajo que tendrán que comunicarse entre sí, establecer una repartición de tareas, etc. Además, se trata de un proyecto de una duración aproximada de 412.5 horas, lo que favorece la elección de una metodología que no requiera excesivo tiempo en etapas de control y que permita avanzar de forma rápida. Otra característica es que este proyecto, aunque busca replicar los datos de un algoritmo ya publicado, también tiene como objetivo su mejora. Estas mejoras del algoritmo sólo se pueden detectar en estados avanzados del proyecto, por lo que se necesitarán modificar

los requisitos según este avance (*RIS-03*) propiciando la elección de una metodología que permita cierto grado de flexibilidad.

Por estos motivos, en especial el último comentado, la elección debe estar orientada a aquellas metodologías ágiles, que permiten acelerar el desarrollo adaptándolo a los cambios en los requisitos, cuando estos son necesarios, buscando la simplicidad en los procesos.

Dentro de las metodologías estudiadas, las que más se ajustaban al presente proyecto son Scrum y Programación Extrema (XP), ambas presentan las características de las metodologías ágiles adaptándose correctamente a las necesidades del proyecto, pero analizando sus pequeñas diferencias se ha optado por escoger la **Programación Extrema**.

Los tiempos de duración de los intervalos o sprints son variados en ambos casos, y aunque Scrum no presenta un conjunto de buenas prácticas, siempre se pueden aplicar especificándolas en el plan del proyecto. La diferencia que causa la elección de la Programación Extrema es su flexibilidad ante los cambios dentro de los Sprints o intervalos, ya que en esta última está permitido que el equipo de trabajo haga ajustes necesarios, mientras que en el Scrum esto está más restringido y deben ser aprobados por el cliente.

Entre los motivos de la elección de la Programación Extrema, aunque la gran mayoría también están soportados por Scrum, cabe destacar los siguientes:

- **Integración continua**: implementación progresiva de las funcionalidades del software, y su integración a medida que se completan.
- **Refactorización**: se promueve la corrección y mejora del código continuamente, aunque esto consuma tiempo, ya que en un futuro evitará problemas por fallos, dificultad en el mantenimiento, etc.
- **Diseño simple**: prima la simplicidad del diseño, siempre y cuando este cumpla con los requisitos del proyecto.
- **Mantener el bienestar del programador**: es importante que el programador, en este caso único desarrollador del proyecto, esté descansado ya que el software que produce será de baja calidad en caso contrario.
- **Facilidad de división del sistema** en varios subsistemas independientes.
- Se fomenta la **reutilización de los elementos**.

5.4. Planificación Temporal

En este apartado se mostrará y explicará la planificación temporal realizada para el proyecto que se está a desarrollar. Lo primero en esta fase es analizar el objeto a medir, con el fin de identificar las tareas de cada fase. Para la realización de dicha identificación se ha utilizado una herramienta ampliamente extendida en la gestión de proyectos, la *Estructura de Descomposición Factorial* o *EDT*. Un EDT consiste en una descomposición jerárquica que representa el trabajo requerido para completar un proyecto. En la Figura 5.1 se muestra el diagrama EDT correspondiente al presente

TFG, con los diferentes apartados que representan las fases del proyecto, extraídos del análisis de requisitos.

Para una comprensión más fácil de la planificación del proyecto se detallará a continuación el caso del proyecto completo, y cada una de las fases, complementando con un cronograma en cada caso. La Figura 5.2 muestra el cronograma correspondiente a la planificación realizada para el proyecto completo, en este aparecen las diferentes fases replegadas, ya que estas se explicarán en detalle en los siguientes apartados, y consiguiendo así una mayor comprensión.

Como se ha explicado en el apartado anterior, la metodología aplicada en este proyecto es Programación Extrema, esta caracteriza su ciclo de vida por la existencia de iteraciones en la que se construyen partes funcionales del software y se entregan al cliente. Estas iteraciones suelen durar entre una y tres semanas. En este proyecto se han planificado 6 iteraciones, las dos primeras y la última con una duración de tres semanas, y las tres restantes de dos semanas de duración.

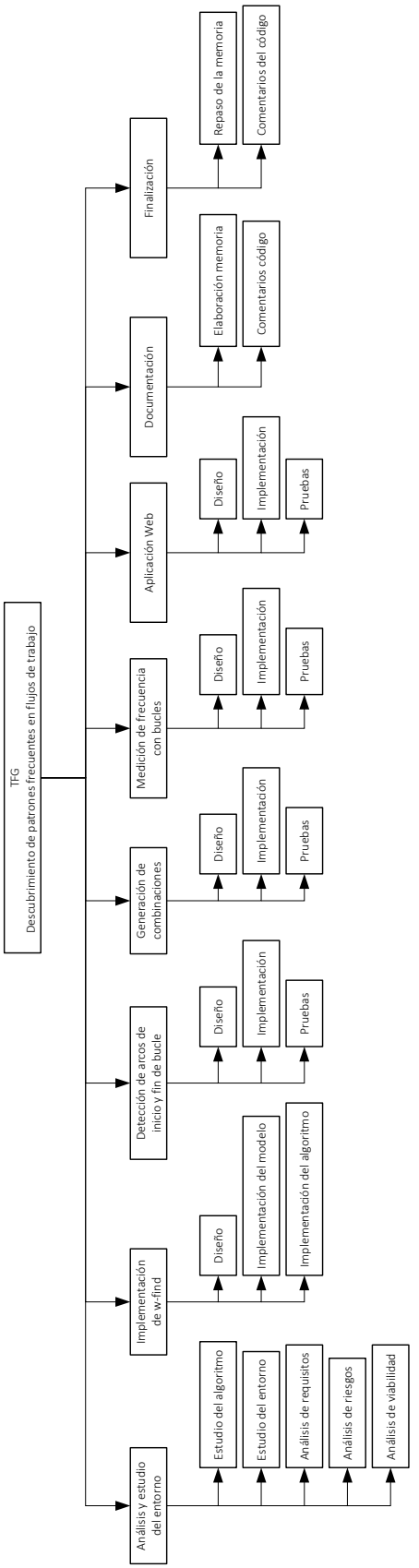

Figura 5.1: Estructura de descomposición de trabajo del proyecto.

Con esta aclaración se puede comprobar en el cronograma de la Figura 5.2 las cuatro interacciones planificadas de forma secuencial, terminando el proyecto con un período de finalización, y teniendo, durante todo el proyecto, un proceso de documentación en paralelo a la codificación.

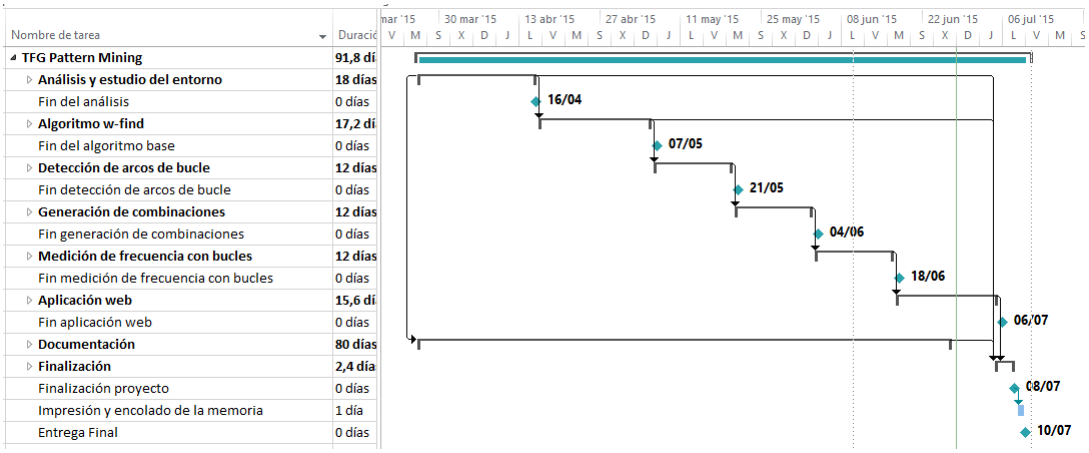

Figura 5.2: Diagrama de Gantt correspondiente a la planificación del proyecto.

Se debe puntuar que todas las fases del proyecto son realizadas por un único recurso, el desarrollador del mismo. A continuación se explicarán cada una de las fases de las que consta la planificación, con su respectivo cronograma desglosado.

Análisis y estudio del entorno

En la Figura 5.3 se muestra el cronograma de la interacción correspondiente al análisis y estudio del entorno. Debido a que el proyecto abarca un campo nuevo para el desarrollador, el conocimiento de este sobre el ámbito en el que va a trabajar es muy escaso, por este motivo es necesaria una fase de estado del arte en la que se familiarizará con el algoritmo y las aproximaciones existentes, para estudiar la forma en la que abordar el problema.

Como se puede apreciar en el cronograma, primero se efectuará una fase de estudio del algoritmo, seguida la cual se comenzará con la identificación de los requisitos, casos de uso y análisis de riesgo, finalizando con un análisis de viabilidad. En paralelo a estos procesos, se encuentra el estudio del entorno, ya que a lo largo de toda esta fase se deberán consultar y comprender diferentes conceptos del ámbito de aplicación.

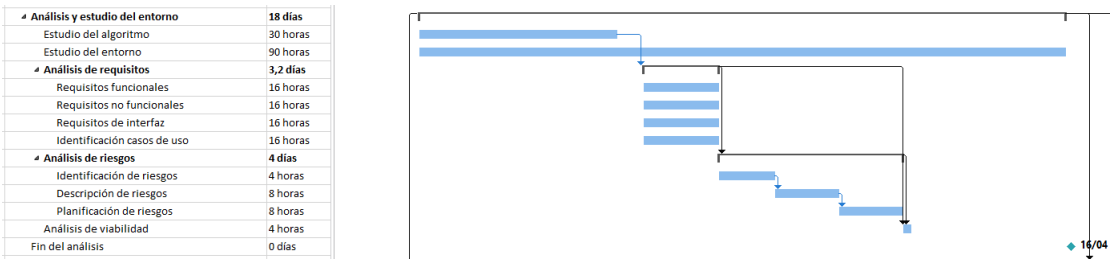

Figura 5.3: Diagrama de Gantt correspondiente a la interacción del análisis.

Cabe destacar que, en el análisis de los requisitos, se han dividido en requisitos funcionales, no

funcionales y de interfaz para diferenciar la búsqueda de los requisitos relacionados con la interfaz, aunque en la práctica todos serán clasificados en requisitos funcionales o no funcionales.

Algoritmo w-find

Una vez se ha estudiado el algoritmo lo suficiente como para conocerlo en profundidad, y siempre y cuando el análisis de viabilidad no detecte una imposibilidad del mismo, se da comienzo a la fase de implementación del algoritmo w-find.

En la Figura 5.4, se muestran las distintas subfases de la que está formado este incremento, comenzando por una fase de diseño, en la que se realiza la especificación de los requisitos identificados en el anterior incremento, así como el diseño del algoritmo base. A continuación comenzará la implementación del modelo de datos para dar soporte al algoritmo, así como de la implementación del mismo. La fase de pruebas unitarias se desarrolla en paralelo al final de la implementación del modelo, y a lo largo de la mayoría del tiempo de la implementación del algoritmo. Por último, la fase de pruebas de integración se realiza al final de la implementación del algoritmo.

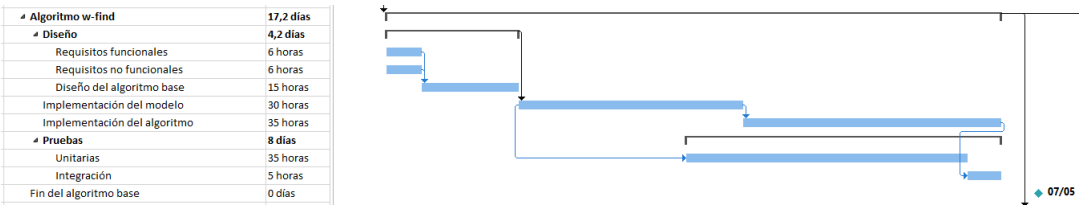

Figura 5.4: Diagrama de Gantt correspondiente a la interacción del algoritmo inicial.

Detección de arcos de bucle

El siguiente incremento a la elaboración del algoritmo base es la creación de un algoritmo que detecte los arcos de inicio y fin de bucle de un grafo. En este caso, debido al desconocimiento de si será posible crear un algoritmo que realice dicho cometido, el desarrollo del proyecto en esta fase consistirá en un diseño constante en el que se crearán diferentes aproximación hasta encontrar una que cumpla con los requisitos.

La fase de implementación da comienzo poco después de que lo haga la de diseño, ya que primero es necesario un estudio y diseño sin realizar implementación ninguna, lo mismo ocurre con la fase de pruebas unitarias, que empieza una vez haya pasado suficiente tiempo de implementación como para poder realizar pruebas. Las pruebas de integración comprenden el fin del intervalo, en el que se analizará el funcionamiento de los diferentes submódulos en conjunto.

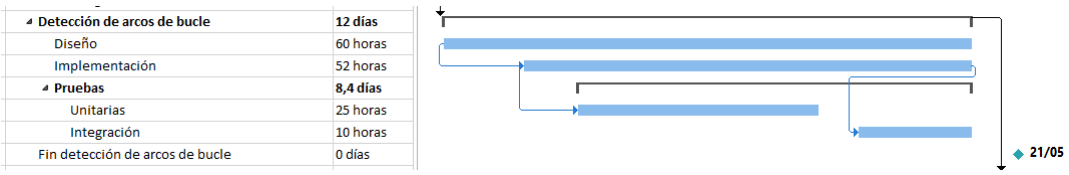

Figura 5.5: Diagrama de Gantt de la fase de desarrollo del algoritmo para detección de arcos de bucle.

Generación de combinaciones

En la siguiente interacción, debido a la similitud con la anterior, ya que el cometido es el de crear un algoritmo que genere las combinaciones de un patrón, la planificación del mismo es similar.

Generación de combinaciones	**12 días**
Diseño	60 horas
Implementación	52 horas
Pruebas	**8,4 días**
Unitarias	25 horas
Integración	10 horas
Fin generación de combinaciones	0 días

Figura 5.6: Diagrama de Gantt de la fase de desarrollo del algoritmo para generación de combinaciones.

Medición de frecuencia con bucles

Al igual que con las dos interacciones anteriores, el objetivo de esta es la creación de un algoritmo que, recibiendo una combinación de un patrón perteneciente a un grafo sin bucles, calcule su frecuencia en las trazas. Por este motivo la planificación es similar, conteniendo las fases de diseño, en la que se crearán y diseñarán posibles algoritmos, y las de implementación y pruebas, para validarlos.

Medición de frecuencia con bucles	**12 días**
Diseño	60 horas
Implementación	52 horas
Pruebas	**8,4 días**
Unitarias	25 horas
Integración	10 horas
Fin medición de frecuencia con bucles	0 días

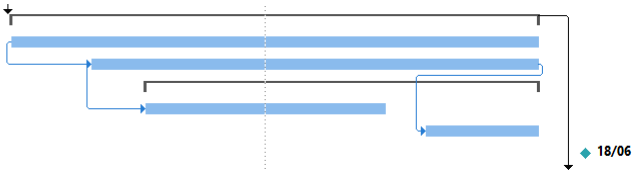

Figura 5.7: Diagrama de Gantt de la fase de desarrollo del algoritmo para medición de frecuencia.

Aplicación web

Una vez finalizadas las fases en las que se implementa el algoritmo, la fase de desarrollo de la aplicación web da comienzo. Esta comienza con el diseño, donde se especificarán los requisitos que estén relacionados con la interfaz, ya identificados en el primer incremento, y se realiza el desarrollo de la interfaz gráfica.

Aplicación web	**15,6 días**
Diseño	**1,6 días**
Requisitos de interfaz	8 horas
Diseño de la interfaz	4 horas
Implementación	**14 días**
Vista	40 horas
Lógica del back-end	70 horas
Pruebas	**12 días**
Unitarias	50 horas
Integración algoritmo e interfaz	15 horas
Fin aplicación web	0 días

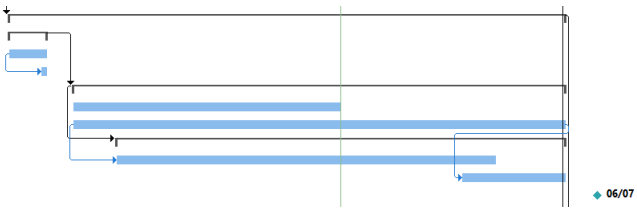

Figura 5.8: Diagrama de Gantt correspondiente a la interacción de la aplicación web.

A continuación dan comienzo las fases de implementación, en las que se desarrolla la vista o *front-end* y la lógica del *back-end*. Al poco de dar comienzo estas dos fases empieza la de pruebas

unitarias, finalizando, como en el resto de casos, con las de integración que validarán el correcto funcionamiento de la interfaz gráfica con la lógica.

Documentación y finalización

Por último se encuentra la fase de finalización, en la que se realizará un repaso de la memoria y del código, junto con el proceso de preparación de la memoria para la posterior entrega. En esta especificación se ha añadido la fase de documentación, que se da en paralelo a todas las interacciones, ya que es un proceso que se realiza a lo largo de todo el proyecto.

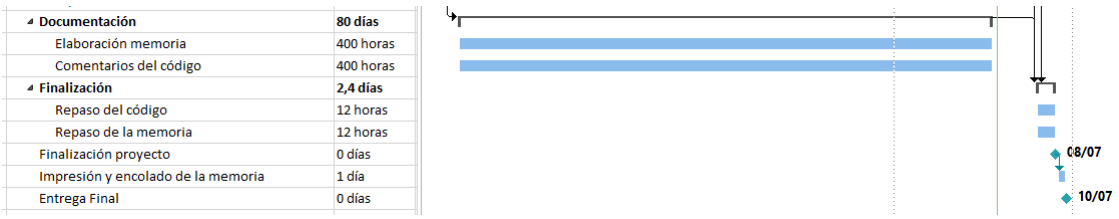

Figura 5.9: Diagrama de Gantt correspondiente a la documentación y fase final del proyecto.

5.5. Análisis de costes

En este apartado se realizará un análisis de costes, el objetivo de este es organizar el presupuesto del proyecto para efectuar un buen control de los gastos. En este análisis se detectarán dos tipos de costes:

- **Directos**: que se pueden asignar de forma clara y definida al proyecto, como por ejemplo las horas trabajadas por un empleado.
- **Indirectos**: se reparten entre diferentes proyectos de forma que su cálculo es de elevada dificultad, como por ejemplo el consumo eléctrico.

Aunque este proyecto es de ámbito académico y será desarrollado únicamente por un alumno, se debe hacer especial hincapié en este apartado, ya que un buen análisis de costes es vital para la correcta realización de un proyecto. Como ejemplo de esta importancia se puede citar los resultados publicados por *The Standish Group* [13] que muestran que aproximadamente el 31 % de los proyectos de software serán cancelados antes de que terminen por problemas de costes, y más de la mitad finalizarán con un sobrecoste del 189 %. En cuanto a los proyectos que terminan sin prolongaciones de tiempo ni costes, estos abarcan aproximadamente el 16 %.

Una vez aclarada la importancia de esta sección, se procederá al cálculo de los costes que supondrían llevar a cabo este proyecto en un ámbito no educativo. Para esto se contabilizarán los costes directos e indirectos que este proyecto conlleva.

En los costes directos, para un proyecto de estas características, únicamente se incluye el salario correspondiente al desarrollador, puesto que los tutores del proyecto asumen un rol de cliente en este TFG. Para esto se tomará un valor medio extraído de un estudio salarial en el sector TIC

de Galicia realizado por la empresa Vitae [17]. En este caso el desarrollador realiza un trabajo de analista programador en JAVA, y dado que no tiene experiencia previa se asumirá un salario Junior (17.000€ brutos anuales). Asumiendo un coste de la seguridad social de un 31 % sobre el salario bruto, se muestra en la siguiente tabla el coste por hora que supondría[5]:

Bruto anual	**Seg. Social**	**Total anual**	**Total mensual**	**Coste/hora**
17.000 €	5270 €	22.270 €	1590 €	9.95 €

Como coste directo se añadirá también el precio de la impresión de la memoria y creación de los CDs con el código, necesarios para la entrega final del proyecto. Calculando aproximadamente 150 páginas de la memoria, se estima un precio aproximado de 100€ para la generación de los documentos de la entrega final.

En cuanto a los costes indirectos, en este grupo entraría el ordenador de trabajo. Para el coste del equipo se tomará el precio del mismo (680€) y se estimará una vida media de 5 años con un uso únicamente de trabajo, teniendo en cuenta que el proyecto tiene una duración aproximada de un tercio de año, se muestra en la siguiente tabla el coste del equipo.

Meses de vida	**Meses de utilización**	**Coste total (€)**	**Coste en el proyecto (€)**
60	4	680	45.4€

En cuanto al coste indirecto correspondiente a servicios como internet, la electricidad del lugar de trabajo, el agua consumida por el usuario, etc. se tomará una estimación del 10 % del coste final, como es común realizar.

El software que se ha utilizado tiene la característica de ser software libre, o trabajar bajo licencias de prueba, por lo que no supone ningún coste. Se muestra a continuación una tabla con el análisis de coste final:

Elemento	**Coste (€)**
Analista programador JAVA	4094.5
Equipo de trabajo	45,4
Apache Tomcat 7.0	0
MySQL server 5.6	0
MySQL Workbench 6.3	0
NetBeans 8.0	0
Java JDK 7	0
StarUML 2.0	0
Linux Mint 17	0
LibreOffice 4.4.3	0
Sublime Text 3	0
Microsoft Project 2013	0
LaTeX	0
Impresión memoria	100
Costes indirectos	10 %
Total	**4663.89**

[5]Coste a la empresa en 14 pagas, 8 horas diarias, 20 días laborables al mes

Esto deja un coste total del proyecto de 4663.89€.

5.6. Plan de Gestión de las Comunicaciones

En este apartado se definen los protocolos a seguir en caso de que se desee realizar un intercambio de información entre el desarrollador del presente proyecto y los interesados del mismo. También se encontrará la información correspondiente a cada una de las personas con las que se deba establecer comunicación.

5.6.1. Gestión de interesados

El PMBOK define como interesados de un proyecto:

> *Las personas y organizaciones que participan de forma activa en el proyecto o cuyos intereses pueden verse afectados como resultado de la ejecución del proyecto o de su conclusión.*

El primer proceso es la identificación de los interesados, este es un proceso continuo que puede resultar difícil. Es muy importante realizar una correcta identificación de los interesados, así como de su influencia en el proyecto y del interés que tienen en el mismo. En este caso, como el TFG que se está desarrollando es de investigación, el número de interesados es reducido, estando este grupo formado únicamente por los tutores del proyecto, el trabajador que lo desarrolla y los potenciales usuarios del sistema.

Para la identificación de los interesados se ha construido una matriz (Tabla 5.1), en la que se muestra toda la información relacionada con ellos: contactos, disponibilidad, etc.

5.6.2. Planificación de la información y comunicaciones

El objetivo principal de esta etapa es garantizar que todos los implicados en el proyecto disponen de la información necesaria en el momento en el que lo sea. Para ello, se deben definir claramente conceptos como quién necesita la información, qué información se distribuye, cuándo se proporciona, etc.

Como métodos de comunicación se definen 3 tipos: la **comunicación interactiva**, en la que se realiza un intercambio de información multidireccional entre dos o más partes. La comunicación de tipo '**push**' es enviada a receptores específicos que necesitan conocer la información. Y el tipo '**pull**' en el que la información se pone a disposición de los receptores, y estos la recogen cuando sea necesario.

El tipo de información utilizada en este proyecto es de tipo interactiva, en la que todos los trabajadores se comunican entre sí, de forma multidireccional y cuando sea necesario. Se ha construido una matriz (Tabla 5.2) para gestionar todas estas características como tipo de comunicación, importancia, interesado, etc.

5.6.3. Reuniones

Para que las reuniones sean efectivas es necesario que primero tengan un objetivo y un tiempo límite determinado. Motivos como falta de objetivo, falta de participación de los asistentes o reuniones muy largas hacen que esta no sea productiva. Es por tanto necesario, a la hora de planificar una reunión cuidar dichos detalles.

Las reuniones se deben realizar en un lugar habitual, donde no haya distracciones y donde todos los miembros se puedan ver entre ellos. En este caso, por defecto se realizarán en una de las aulas de reuniones de las que dispone el CiTIUS [6], ya que la mayor parte de los integrantes de las mismas tienen su lugar de trabajo en dicho edificio.

La periodicidad de las reuniones será variable, dependiendo del grado y cantidad de problemas con los que se encuentre el proyecto a lo largo de su desarrollo, pero se planea una frecuencia alta al principio y una bajada en el número de reuniones a medida que avanza el proyecto. Esto es debido a que durante las etapas de análisis deberá haber un mayor nivel de interacción con los interesados.

[6] Centro Singular de Investigación en Tecnoloxías da Información de la Universidad de Santiago de Compostela (www.citius.usc.es/)

Identifiación					**Evaluación**				**Clasificación**	
Nombre	**Empresa/ grupo**	**Localización**	**Rol en el proyecto**	**Información de contacto**	**Requisitos**	**Expectativas**	**Influencia potencial**	**Fase mayor interés**	**Externo/ interno**	**Apoyo/ neutral/ opositor**
Tutores del proyecto	Grupo	Santiago de Compostela	Tutores	manuel.mucientes@usc.es manuel.lama@usc.es borja.vazquez@usc.es	Tutorización y resolución de dudas	Logro del objetivo del proyecto	Alta	Todo el proyecto	Externo	Apoyo
Desarrollador del proyecto	Equipo del proyecto	Santiago de Compostela	Único desarrollador	david.chapela.delacampa @rai.usc.es	Compromiso con el desarrollo y elaboración	Culminación exitosa del proyecto	Alta	Todo el proyecto	Interno	Apoyo
Potenciales usuarios	Usuarios	Sin localización determinada	Cliente	No hay información de contacto	——	Logro del objetivo del proyecto	Baja	Finalización del proyecto	Externo	Neutral

Tabla 5.1: Matriz de interesados del proyecto.

Identifiación	**Grupo**	**Propósito de la comunicación**	**Contenido**	**Nivel de detalle**	**Importancia**	**Periodicidad**	**Emisor**	**Receptor**	**Formato**	**Idioma**	**Métodos y tecnologías**
Interesado 1	Tutores del proyecto	Informar sobre los avances en el proyecto y consultar dudas sobre el mismo.	Información de índole técnica	Alto	Esencial	No es una comunicación periódica, se trata de reuniones puntuales.	Jefe del proyecto	Uno de los tutores del proyecto	Comunicación electrónica	Español	Correo electrónico.
Interesado 2	Desarrollador del proyecto	Conocer el estado del desarrollo del proyecto en todo momento.	Información exhaustiva sobre el proyecto.	Muy Alto	Esencial	Comunicación continua, informando del estado del proyecto.	Desarrollador	Desarrollador	No procede	Español	No procede
Interesado 3	Potenciales usuarios	Conocer las necesidades de los usuarios	Informe sobre las funcionalidades del software	Alto	Media	No es una comunicación periódica, se trata de charlas puntuales	Jefe del proyecto	Usuarios	Comunicación electrónica	Español	Correo electrónico

Tabla 5.2: Matriz de comunicaciones del proyecto.

Capítulo 6

Arquitectura y Herramientas de Desarrollo

Después de tener una idea formada de los objetivos y requisitos del presente proyecto, se debe describir una arquitectura a alto nivel que proporcione una solución tecnológica al problema. De esta forma, con un nivel elevado de abstracción, se desliga totalmente la arquitectura de una implementación concreta para, a posteriori, estudiar y escoger las tecnologías y herramientas que se consideren adecuadas sin depender del diseño realizado en este apartado.

Tras la definición de la arquitectura se introducirán las tecnologías y herramientas escogidas para el desarrollo del proyecto, teniendo en cuenta un desarrollo orientado a objetos.

6.1. Arquitectura del sistema

Una vez definidos los requisitos del sistema, y teniendo bien claros los objetivos del proyecto, se procede al diseño de la arquitectura a alto nivel. Es importante tener claro en este diseño aspectos como la separación que debe haber entre la interfaz y el algoritmo (requisito *RNF-01*), ya estos se deberán ver reflejados desde el inicio.

La solución adoptada pretende crear una interfaz gráfica para la interacción con el algoritmo y la visualización de los resultados de la ejecución del mismo. Además de la interfaz, será necesario un componente que proporcione la funcionalidad del algoritmo, estando este formado por un conjunto de servicios RESTful que permitirán el acceso a la base de datos para comprobar si los resultas ya han sido obtenidos, y al algoritmo para iniciarlo y detenerlo cuando sea preciso.

Previamente a detallar la arquitectura, y para realizar la separación de conceptos que exigen los requisitos, favoreciendo así también la reutilización de código, se aplicará un patrón de arquitectura denominado *Modelo-Vista-Controlador* (MVC). El MVC es un patrón de arquitectura de software ampliamente extendido en aplicaciones con interfaces de usuario, establece un desacoplamiento entre la vista de la aplicación, y la lógica que opera con los datos de la misma, estableciendo un controlador que redirige la interacción del usuario con la vista.

En la Figura 6.1 se muestra la estructura genérica del patrón MVC, como se puede apreciar se

realiza una separación entre la vista, el controlador y el modelo, consiguiendo así desacoplar las distintas partes de la aplicación, y permitiendo el reemplazo de cualquiera de ellas de forma más simple. La implementación de este patrón favorece el requisito *RNF-01*, en el que se requiere cierta separación entre el algoritmo desarrollado y la interfaz elaborada.

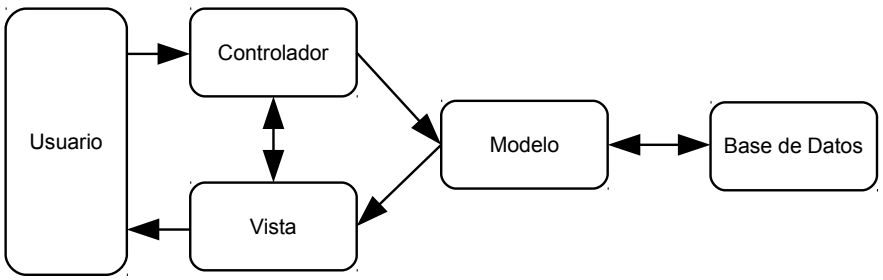

Figura 6.1: Arquitectura del patrón MVC.

Las distintas partes que forman la arquitectura de la aplicación con el patrón MVC son las siguientes:

- **Usuario**: representa los usuarios de la aplicación, que interactuarán con el controlador, recibiendo la información a través de la vista.
- **Controlador**: este componente recibe los eventos de entrada, y es el encargado de decidid qué hacer, pudiendo realizar llamadas al modelo o a la vista.
- **Vista**: ésta recibe los datos del modelo a través del controlador, mostrándoselos a los usuarios de la forma apropiada. Tanto el controlador como la vista serán implementadas en la interfaz de usuario.
- **Modelo**: componente con la lógica de la aplicación, responsable de acceder a la base de datos y de realizar gran parte de las funcionalidades del sistema.
- **Base de Datos**:

Teniendo en cuenta lo especificado anteriormente, a continuación se detallará la arquitectura de los componentes que se desarrollaron a lo largo del proyecto. En la Figura 6.2 se muestra un diagrama en el que se reflejan los elementos que conforman la arquitectura diseñada en este TFG.

La arquitectura del sistema ha sido diseñada desde el punto de vista del patrón arquitectónico MVC, a continuación se explicarán los diferentes componentes que lo forman:

- **Vista**: la vista es la parte encargada de mostrar la información de la aplicación al usuario, este a su vez puede interactuar con la misma para realizar las acciones deseadas, actualizándose así la información a mostrar. En este caso, la vista está formada por la interfaz gráfica (GUI), que está formada por tres vistas, explicadas en el apartado de diseño.
- **Controlador**: El controlador es el elemento que recibe las peticiones de la vista, y decide *qué* hacer como respuesta a cada interacción, interactuando con el modelo cuando es necesario, para realizar las operaciones oportunas y devolver el control a la vista. En este caso, el controlador está formado por un controlador y un *helper*. En este último se delega la elaboración de las acciones necesarias, mientras que en el controlador reside la elección de éstas.

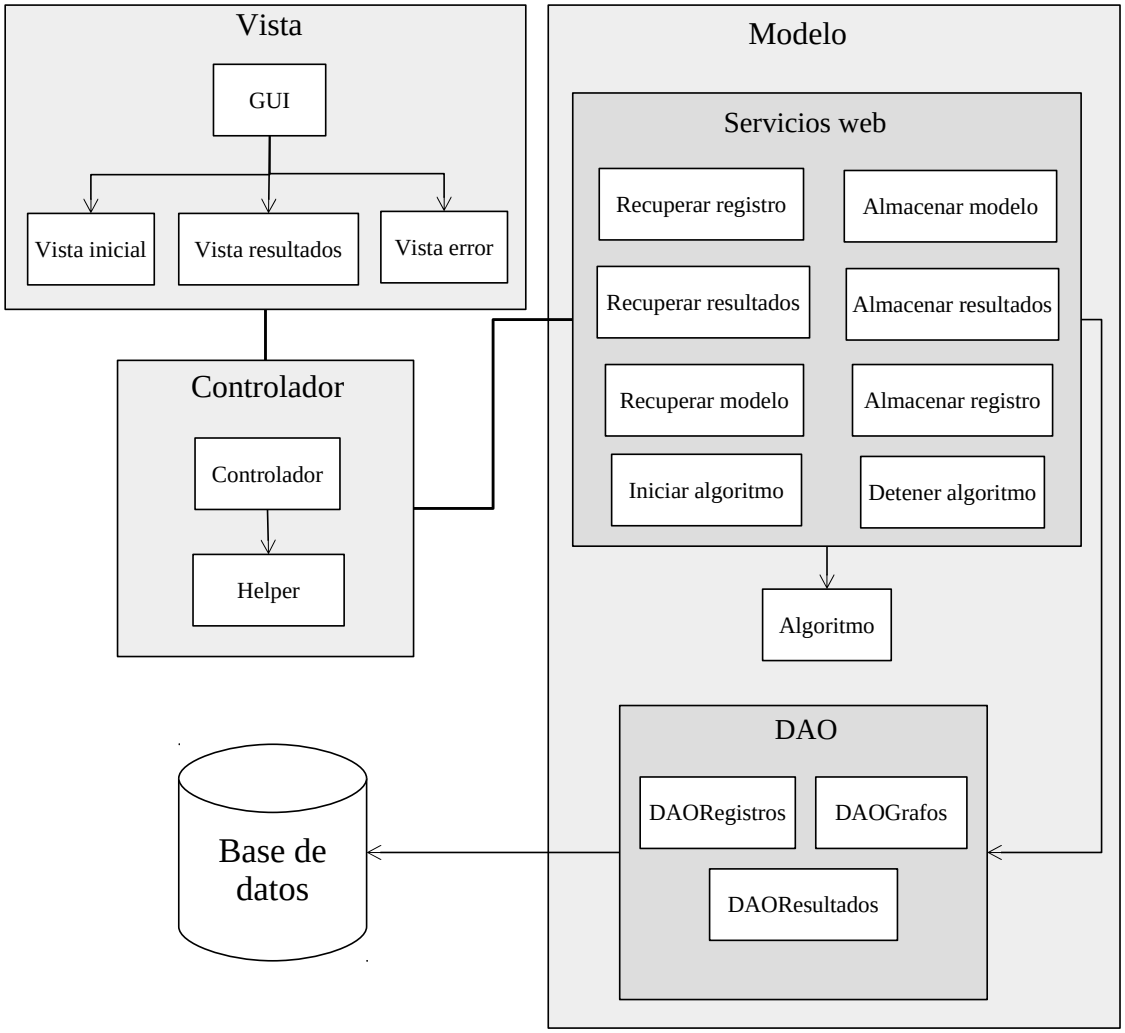

Figura 6.2: Diagrama de arquitectura del sistema.

- **Modelo**: finalmente, el modelo es el encargado de realizar las operaciones relacionadas con la lógica de la aplicación, accediendo a la base de datos, y devolviendo los resultados al controlador. En el diseño de este proyecto se refleja la interacción entre el controlador y el modelo, la cual se realiza a través de los servicios web implementados, que se encargarán de acceder a la base de datos por medio de los DAOs (*Data Access Object*)[1] o de inicial el algoritmo cuando sea preciso.

Cabe resaltar la importancia del algoritmo en este proyecto, ya que se trata de la sección de mayor complejidad, y que más tiempo requerirá. De hecho, como se ha comentado previamente, es posible que en un futuro se integre el algoritmo en otra aplicación, por ello es necesario un diseño que permita su traslado de forma sencilla, y por ello se ha aplicado el patrón Modelo-Vista-Controlador.

[1]Los DAOs son componentes software que proporcionan una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos.

6.1.1. Servicios Web

Como se puede apreciar en la Figura 6.2, las llamadas realizadas por las vistas, después de pasar por el controlador, recaen en servicios web. Un servicio web es una tecnología que establece un conjunto de protocolos y estándares para intercambiar datos entre aplicaciones, independientemente de la plataforma en la que estas se ejecutan. Con la inclusión de un servicio web se añade, no solo una organización y desacoplamiento entre las diferentes llamadas realizadas por las vistas, sino un estándar de interoperabilidad con una serie de llamadas que pueden realizar diferentes aplicaciones, independientemente de su plataforma. De esta forma, se puede realizar un cliente con otra tecnología que realice las llamadas al servicio, sin necesidad de implementar de nuevo todo el modelo y la lógica de la aplicación.

El tipo de servicio escogido para implementar el controlador ha sido un servicio REST (*Representational State Transfer*). REST es un tipo de arquitectura software que utiliza el protocolo HTTP para la obtención de datos o indicación de ejecución de operaciones en cualquier formato (JSON, XML...). Esta libertad en la elección del formato de los datos ha sido la que ha impulsado a la elección de REST como arquitectura para elaborar el servicio de esta aplicación.

Se ha pensado la elaboración del servicio web utilizando SOAP (*Simple Object Access Protocol*), éste es, a diferencia de REST, un protocolo, y define el método de comunicación entre dos objetos utilizando XML como formato de los datos. Finalmente, se ha descartado debido al requisito de encapsular los datos en formato XML, lo cual no es necesario para los objetivos del algoritmo, y que aumentaría considerablemente su tamaño, haciendo más lentas las transferencias.

6.2. Herramientas de Diseño

Para realizar el modelado del software de un modo formal, se utilizará *UML* (*Unified Modeling Language*). UML es el lenguaje de modelado de sistemas software más expandido en la actualidad y respaldado por el *Object Management Group*[2], por lo que se asegura la mayor compatibilidad a la hora de especificar diferentes ámbitos del sistema, desde la lógica de negocio hasta la estructura hardware.

Debido a la importancia que suponen los modelos en un proceso de desarrollo, y para poder definir correctamente los esquemas del sistema software, UML ofrece 13 diagramas de modelado. En el presente TFG únicamente serán necesarios los siguientes:

- **Diagramas de casos de uso**: para capturar el comportamiento e interacción del sistema con sus actores.
- **Diagramas de secuencia**: para modelar los casos de uso representando la interacción entre los objetos ante el paso del tiempo.
- **Diagramas de clases**: donde se define la estructura del sistema mostrando sus clases, atributos, etc. y las interacciones entre ellas.

[2]El OMG es un consorcio dedicado al cuidado y establecimiento de diversos estándares de tecnologías orientadas a objetos.

6.2.1. Herramientas

Para realizar los diagramas correspondientes a los diseños UML, que se muestran en la presente memoria, se utilizará el software *StarUML 2.0*[3]. El principal motivo de su elección es el amplio uso en la carrera de una versión anterior del mismo, durante el cual no se han experimentado limitaciones ni problemas que propiciasen la búsqueda de otro software para su reemplazo.

6.2.2. Patrones de diseño

Como ya se ha comentado previamente, el proyecto que se documenta en esta memoria se acogerá bajo el paradigma OO (orientado a objetos), este tipo de proyectos presentan ciertas dificultades a la hora de realizar un buen diseño, y puesto que un objetivo es el bajo acoplamiento entre los diferentes componentes que forman el sistema, se debe de tener cuidado en este proceso de forma que se favorezca esta característica.

Cuando se desarrolla un sistema de alta complejidad, como el que se está presentando en esta memoria, no se debe abordar el problema directamente y diseñar soluciones de forma rápida, ya que se puede correr el riesgo de construir un sistema con una dificultad muy elevada a la hora de realizar cambios, ya sean de mantenimiento o modificaciones en el comportamiento. Para evitar estos problemas se debe realizar un diseño de la forma más precisa posible, prestando la máxima atención a la reutilización de soluciones ya creadas y probadas.

Al igual que en una empresa se suele reutilizar una estructura, código o componente en varios proyectos, debido a que tiempo invertido en su creación y validación asegura su correcto funcionamiento, en cualquier tipo de proyecto es conveniente utilizar estructuras ampliamente conocidas y probadas, que no solo garanticen una adecuación al sistema, sino que permitirán a posibles futuros desarrolladores realizar una abstracción de ciertos módulos del sistema facilitándoles la comprensión del proyecto. Estos modelos idóneos para la reutilización son los patrones de diseño.

Un patrón de diseño se podría definir como una solución eficaz, y reutilizable, que se puede aplicar en un gran ámbito de situaciones, y que, gracias a su extendido uso, asegura un funcionamiento correcto y fácil reconocimiento del mismo por parte de otros desarrolladores.

A continuación se introducirán los patrones aplicados en el diseño del sistema que se presenta en esta memoria. Esta breve descripción complementará la explicación en la siguiente sección de los patrones, ya introducidos en el diseño del sistema.

Patrón *Abstract Factory*

El patrón creacional *Abstract Factory*[4] introduce una modularización en un sistema creando una estructura que facilita el reemplazo, creación y ampliación de objetos dentro de un mismo entorno.

Para una mejor comprensión se proporciona la Figura 6.3, que refleja la abstracción que se ha explicado en el párrafo anterior, mostrando el diagrama de clases genérico del patrón. En este se pueden observar dos tipos de productos reflejados en las clases abstractas *AbstractProductA* y *AbstractProductB*, estos serán utilizados por el cliente pero delegando su creación en la clase

[3] www.staruml.io/
[4] http://www.oodesign.com/abstract-factory-pattern.html

AbstractFactory. De esta forma, si se desea cambiar la familia de productos que se está utilizando, solo es necesario cambiar la implementación de la clase en la que se delegó la creación, y esta será la encargada de instanciar productos del mismo tipo, pero diferente familia.

La utilidad de este patrón, como se explicará en el apartado de diseño, aparece cuando se tiene una estructura con diferentes tipos de productos, que pueden ser reemplazados en conjunto por productos similares, pero de diferente implementación. Con este patrón el reemplazo se realiza cambiando únicamente la instanciación de la *ConcreteFactory* que genere los productos de una familia por otra implementación que los genere de la familia deseada.

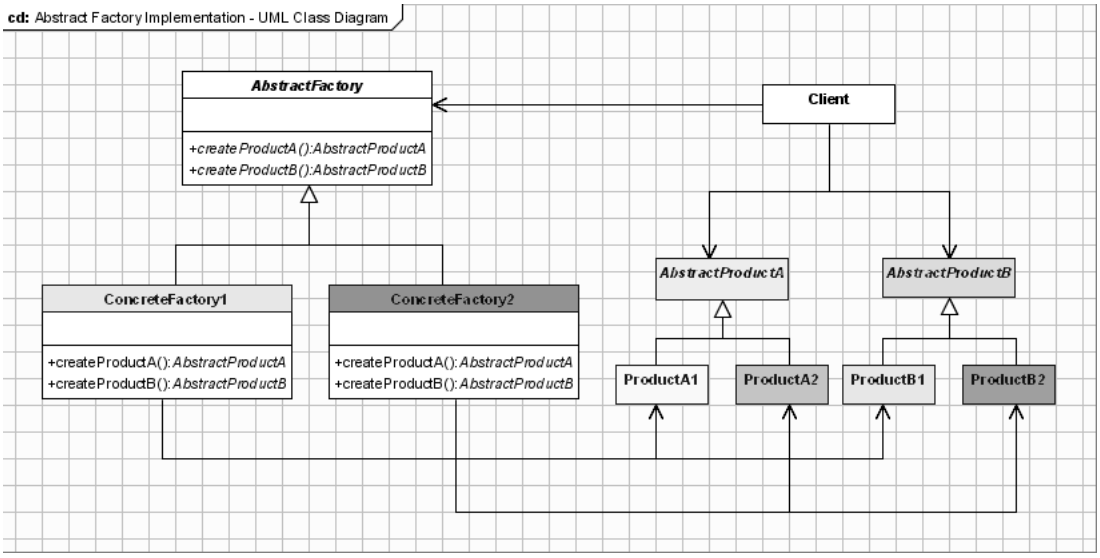

Figura 6.3: Diagrama de clases del patrón *Abstract Factory* (Fuente: [11]).

- ***AbstractFactory***: define la interfaz para la creación de los diferentes tipos de productos de las clases *AbstractProduct*.
- ***ConcreteFactory***: implementa las operaciones de *AbstractFactory* instanciando productos de su familia, que hereden de las clases *AbstractProduct*.
- ***AbstractProduct***: Declara la interfaz de los tipos de productos que se generarán.
- ***ConcreteProduct***: define los productos que serán creados, implementando los métodos que la clase *AbstractProduct* le especifique.
- ***Client***: utiliza la interfaz declarada por *AbstractFactory* para obtener productos de las clases *AbstractProduct*.

Patrón *Observer*

Con el objetivo de reducir las dependencias y desacoplar al máximo los objetos entre sí nace el patrón de comportamiento *Observer*[5], que ofrece una solución eficiente cuando uno o varios objetos quieren suscribirse y ser notificados de ciertos cambios en otro objeto.

[5]http://www.oodesign.com/observer-pattern.html

En la Figura 6.4 se muestra la estructura de clases correspondiente a este patrón. La clase abstracta *Observable* será la encargada de notificar a todos los que quieran ser notificados cuando sea necesario, para ello almacenara un conjunto de objetos *Observer* y ejecutará su método *notify()* cuando esto se requiera. La clase que define el comportamiento de notificación es *ConcreteObservable*, y como observadores concretos se tienen las clases que extienden de *Observer*, denominadas en el diagrama *ConcreteObservableA* y *ConcreteObservableB*. Con este diseño se permite separar la acción de notificación de las implementaciones de los objetos que observan, ya que, mientras estos implementen o hereden de *Observer*, serán tratados de igual forma por el objeto que debe notificar.

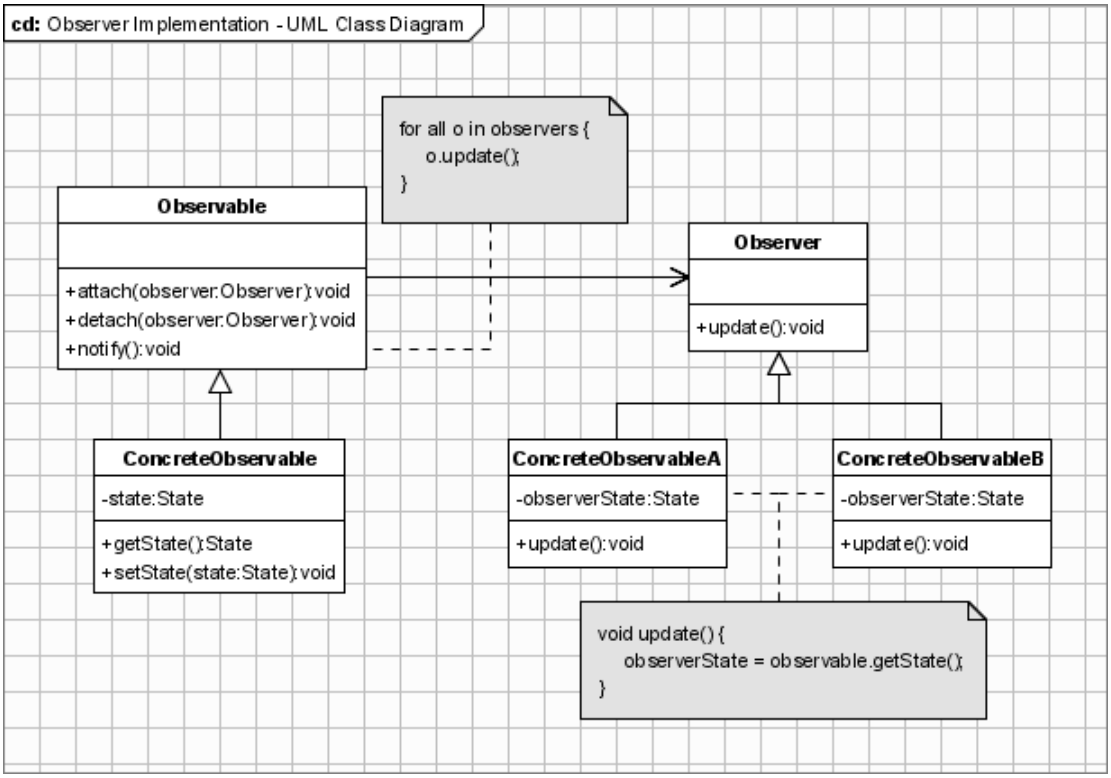

Figura 6.4: Diagrama de clases del patrón *Observer* (Fuente: [11]).

- ***Observable***: clase abstracta o interfaz que define los métodos para suscribirse, anular la suscripción y notificar a los observadores.
- ***ConcreteObservable***: clase que implementa o hereda de Observable, es el que mantiene el estado, y notifica a sus suscriptores cuando este cambia.
- ***Observer***: interfaz o clase abstracta que define el método para ser notificado.
- ***ConcreteObserverA, ConcreteObserverB***: implementación de la clase *Observer*.

Patrón *Strategy*

Hay casos en los que se tiene diferentes variantes de un algoritmo, en las que la única diferencia es el comportamiento del mismo. En estos casos sería útil un método que permitiese efectuar el cambio de algoritmo de forma sencilla, sin tener que adaptar todo el código donde este aparezca

o sea referenciado. Esta solución la ofrece el patrón de comportamiento *Strategy*[6], este patrón consiste en una interfaz que define los métodos comunes de las estrategias, pudiendo cambiar la estrategia únicamente donde se instancia.

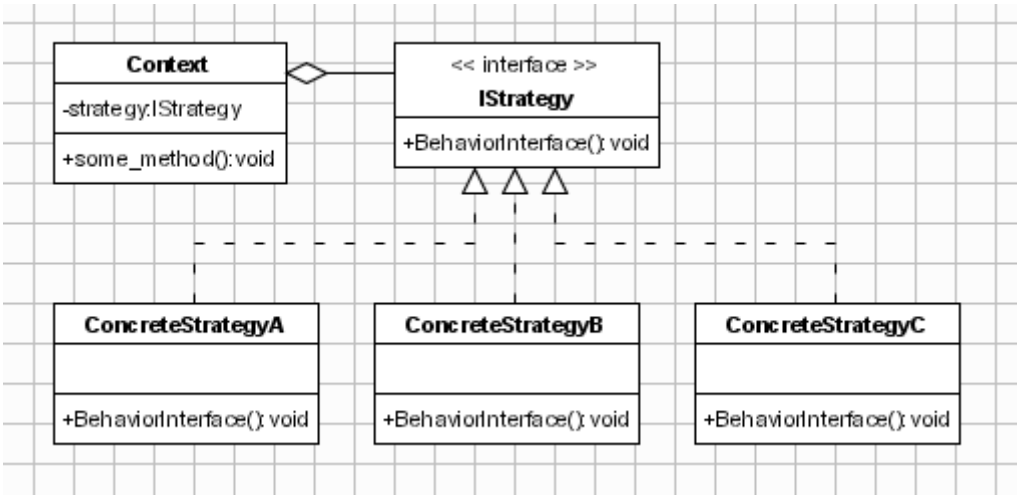

Figura 6.5: Diagrama de clases del patrón *Strategy* (Fuente: [11]).

- ***Context***: esta clase es la encargada de interactuar con el cliente (el que llama al algoritmo) y de instanciar y ejecutar el método de la estrategia deseada.
- ***IStrategy***: interfaz común a todos los diferentes algoritmos o estrategias.
- ***ConcreteStrategy***: implementaciones de la interfaz, cada una con una estrategia diferente.

6.3. Herramientas de desarrollo

Un punto también importante es la elección de las herramientas que se utilizarán para el desarrollo del proyecto. Se debe hacer una selección teniendo en cuenta el tiempo del que se dispone para el aprendizaje de su uso, en caso de ser nuevas, y también las facilidades que presentan.

Por estos motivos, uno de los puntos de mayor importancia a la hora de elegir las herramientas necesarias ha sido el conocimiento previo de las mismas, siendo en algunos casos, junto con la ausencia de problemas encontrados, los motivos claves para la elección del software a utilizar. A continuación se describe el entorno de desarrollo utilizado para la realización del presente TFG:

- **Sistema operativo**: Linux Mint 17 (64 bits)
- **Framework**: Java JDK 7u80, JQuery, Bootstrap.
- **IDE de programación**: NetBeans 8.0.2
- **Editor de textos**: Sublime Text 3
- **Base de datos**: MySQL

[6] http://www.oodesign.com/strategy-pattern.html

6.3.1. Realización de la memoria

A continuación se introducirán las herramientas, y el motivo de su elección, que se han utilizado para la elaboración de la presente memoria.

- **ShareLaTeX**[7]: para el desarrollo de la memoria se ha utilizado LaTeX, un potente sistema de composición de textos que se convirtió en un estándar de facto para la elaboración de documentos técnicos y científicos. Debido al desconocimiento de la estructura de los documentos en LaTeX y por consejo de los tutores, se ha utilizado el editor en línea ShareLaTeX[8], concretamente el que ofrece el CiTIUS en un página web.
- **LibreOffice 4.4.3**: para la elaboración de las figuras y esquemas necesarios para la memoria se ha utilizado el paquete LibreOffice, concretamente el Draw, debido a su conocimiento previo, que agilizó la creación de las mismas.
- **Microsoft Project 2013**: Debido al uso de licencias de prueba que permitieron extender el uso del programa a lo largo del proyecto, y aprovechando su compatibilidad con programas similares de software libre para futuros usos, se ha decidido desarrollar la planificación del Trabajo Fin de Grado con este software, debido a su clara superioridad ante otras soluciones.

6.3.2. Desarrollo

En este apartado se muestran las herramientas utilizadas para el desarrollo del proyecto, y se explica el *por qué* de su elección:

- **NetBeans IDE 8.0.2**: aunque existe un amplio conjunto de entornos de desarrollo integrados (IDEs), los más conocidos son NetBeans y Eclipse, se ha optado por el uso de NetBeans por las siguientes razones:
  - Conocimiento del IDE: aunque a lo largo de la carrera se han utilizado ambos, el uso de NetBeans ha sido considerablemente mayor que el de Eclipse, y el desarrollador posee un conocimiento de comodidades mucho más inculcado.
  - Soporte para SVN: NetBeans, al igual que Eclipse, ofrece la posibilidad de instalar una extensión para el control de versiones Subversion, control que se deseaba ejercer tal y como ha sido explicado en apartados anteriores.
  - Profiler: Netbeans cuenta con un profiler integrado que permite analizar el consumo de memoria y CPU, primordial para analizar el coste de ejecución del algoritmo genético.
- **Apache Tomcat 7.0**: Para el alojamiento de la aplicación web que se va a desarrollar se utilizará este contenedor de aplicaciones de software libre, y que permite una integración con el entorno de desarrollo automatizando el despliegue de la aplicación.
- **MySQL Workbench 6.3**: Para la realización del modelo de la base de datos, y facilitar la gestión de la misma, se ha decidido utilizar esta herramienta visual open source.

[7] https://es.sharelatex.com/
[8] https://sharelatex.citius.usc.es/

- **Chrome 43.0**: Para la elaboración de la aplicación web se ha necesitado un explorador web sobre el que realizar las pruebas, para esto se ha escogido Chrome debido a su extendido uso por la comunidad de usuarios y al conocimiento previo del desarrollador sobre el mismo.

6.3.3. Tecnologías

Una vez definida la arquitectura del sistema es necesario seleccionar los lenguajes de programación y tecnologías a utilizar para el desarrollo del mismo. A la hora de escoger las tecnologías y lenguajes de programación se han tenido en cuenta los objetivos del proyecto, así como las dificultades que presenta el uso de cada elemento, analizando si resultará eficiente utilizarlas. Teniendo en cuenta esto, a continuación se explican las tecnologías y lenguajes seleccionados, junto a la justificación o motivo de la elección de los mismos.

- **JAVA JDK 7u40**: los lenguajes contemplados para la elaboración del código fueron C++ y Java, pero por lo motivos comentados a continuación, se ha desechado la opción de utilizar C++:
  - El motivo más importante era la posibilidad de integración que hay con el algoritmo ProDiGen [12], que está desarrollado en Java y que es el encargado de generar los workflows de entrada del presente algoritmo. Para esta unión era más que conveniente que el proyecto se desarrollase en el mismo lenguaje, facilitando así el acoplamiento de ambos algoritmos.
  - La inexperiencia en el lenguaje C++ por parte del desarrollador supondría dedicarle tiempo del proyecto al aprendizaje de un nuevo lenguaje de programación.
  - Otro de los motivos de elección es la importancia de la gestión de los recursos que se necesita para este algoritmo, y dado que Java ofrece un *recolector de basura* propio, el desarrollador no se tiene que preocupar de la labor de liberar la memoria una vez se ha dejado de utilizar.
  - Por último, Java es un lenguaje que se traduce a bytecode y, posteriormente, es interpretado por una *Máquina Vitual Java* (JVM), esta es la que se encarga de analizar y optimizar el código gracias al sistema de compilación *Just-In-Time* (JIT) [20]. Esto hace posible que el código se optimice independientemente de la arquitectura sobre la que va a ser ejecutado, por lo que esta característica es idónea para cumplir el requisito RNF-05.
- **Servicio RESTful** (*Representational State Transfer*): este término se utiliza para sistemas que siguen los principios REST. Esta es una tecnología ligera y escalable que, mediante una interfaz HTTP, define un conjunto de principios arquitectónicos para la elaboración de servicios web enfocado a los recursos del sistema. Con esta arquitectura se da soporte al intercambio de información sobre HTTP a un conjunto de clientes, independientemente de la plataforma en la que estos se ejecuten, o el lenguaje en el que estén implementados. Este intercambio de información se puede realizar en diversos formatos, como XML o JSON, siendo este último el utilizado en el presente TFG. Por último, para justificar la elección de esta tecnología frente a otras se presentan a continuación las ventajas que el uso de esta proporciona:

  - **Estándar**: se trata de una tecnología estándar hoy en día, con un amplio uso que proporciona documentación realizada por la comunidad que puede ser consultada a la hora de resolver posibles problemas que surjan.
  - **Bajo acoplamiento**: La separación entre el servicio implementado con REST y los clientes que se pueden desarrollar está constituida por una interfaz HTTP, que permite desacoplar ambos componentes, tal y como especifica el requisito *RNF-01*.
  - **Interoperabilidad**: debido a la interfaz sobre el protocolo HTTP que aportan este tipo de servicios, la interacción entre el mismo con los clientes no depende de la implementación de los mismos, pudiendo ser desarrollados en diferentes plataformas y lenguajes.

- **Bootstrap 3.3.5**: Para el desarrollo de las vistas de la aplicación web se ha utilizado el conjunto de herramientas que proporciona el Bootstrap, que facilita el desarrollo de interfaces gráficas ofreciendo plantillas, componentes, estilos, etc. utilizando HTML5 y CSS3, que simplifican el trabajo a realizar. Una de las características más detacables de Bootstrap es que ofrece un sistema de rejilla que permite realizar una construcción de interfaces redimensionables de manera sencilla, lo cual no es imprescindible para la interfaz a desarrollar en este proyecto, pero que puede ser interesante para facilitar una posible extensión del mismo. Además, cuenta con un gran número de complementos jQuery que ofrecen funcionalidades a mayores sin necesidad de implementarlas desde 0.

  Debido al conocimiento del desarrollador de este *framework* por la realización de cursos con anterioridad se ha decidido utilizar para la elaboración de la interfaz gráfica. A continuación se resumen los motivos principales por los que se ha optado por la utilización de este marco de desarrollo en el proyecto:

  - **Componentes reutilizables**: además de los elementos propios de HTML5, Bootstrap ofrece una serie de componentes con estilos propios que facilitan la elaboración de características avanzadas como listas desplegables, barras de progreso, etc.
  - **Estilos propios**: este marco de desarrollo cuenta con un gran número de estilos creados y extendidos por la web que disminuyen el tiempo de trabajo dedicado a la maquetación de la aplicación, pudiendo reutilizar así los ya existentes.
  - **Plug-ins de JavaScript**: la existencia de diversos complementos realizados en jQuery como las tablas dinámicas, diálogos y demás, podrá facilitar la realización de ciertas funcionalidades de la interfaz.

- **JavaScript**: lenguaje de programación [8] interpretado, con soporte a la programación orientada a objetos, basado en prototipos que se ejecuta en el navegador del cliente sin necesidad de interacción con el servidor, aunque permite la implementación de dicha comunicación. Con JavaScript se ofrecerá dinamismo a la interfaz web, pudiendo ejecutar llamadas a servicios y manipular los datos obtenidos sin necesidad de recargas de la página. Además, el uso de este lenguaje proporciona la posiblidad de aplicar librerías para facilitar el procesamiento y visualización de la información.

- **HTML5** (*HyperText Markup Language, versión 5*): se trata de un lenguaje de marcado interpretado por el navegador y utilizado para definir la estructura de la interfaz web. Concretamente HTML5 ofrece un conjunto mejorado de etiquetas que facilitan la inclusión de

determinados componentes e introduce un contenido semántico a la estructura de la web, facilitando así la interpretación de la misma por parte de aplicaciones informáticas, ofrecuendo una estructura adecuada con la Web Semántica (o Web 3.0)[9].

- **CSS3** (Cascading Style Sheets): es un lenguaje utilizado para definir y establecer la presentación de documentos estructurados como HTML o XML. Con este lenguaje se consigue separar la estructura de un documento de su presentación.
- **JSON** (JavaScript Object Notation): es un formato estándar para el intercambio de información caracterizado por su ligereza, característica que propicia su elección frente a XML, ya que este último aumentaría el tamaño de la información intercambiada entre los servicios web y la interfaz, empeorando el rendimiento de la misma. JSON está formado por dos estructuras: *(i)* un conjunto de pares nombre/valor que simbolizan un objeto o una estructura y *(ii)* una lista de valores como podría ser un array o un vector.
- **AJAX** (Asynchronous JavaScript And XML): se trata de una técnica de desarrollo web que permite crear aplicaciones interactivas que se ejecutan en el cliente, y mantienen una comunicación asíncrona con el servidor en segundo plano. De esta forma, es posible realizar actualizaciones en las páginas sin necesidad de recargarlas, mejorando así la experiencia de uso de la aplicación.
- **MySQL server 5.6**: Debido al acceso a base de datos de la aplicación para almacenar resultados previos, es necesario utilizar un sistema gestor de bases de datos. Se ha escogido este sistema por su sencillez y aplicabilidad cuando los requisitos de acceso a datos no son muy exigentes, además de por el uso previo que se le ha dado en la carrera durante el cual se ha adquirido una fluidez considerable comparado con otros sistemas gestores.

6.3.4. Librerías

- **JQuery 1.11**: Para facilitar la programación en JavaScript de las vistas de la aplicación web se ha utilizado la librería JQuery. Esta permite simplificar la interacción con los documentos HTML, la manipulación del árbol DOM, realización de llamadas AJAX... JQuery ha sido utilizada durante la carrera y presenta una documentación de fácil acceso y comprensión, por lo que ha sido elegida para este cometido.
- **graphViz**: graphViz es un paquete de software libre con herramientas para la representación de grafos. Concretamente, en este TFG se ha utilizado una librería que facilita la representación de grafos en formato DOT mediante JavaScript. Se ha decidido utilizar esta herramienta para facilitar la representación de los grafos, a pesar del desconocimiento sobre el uso de la misma por parte del desarrollador.

[9]La Web Semántica es un conjunto de actividades con el fin de crear tecnologías para la pulicación de datos legibles por aplicaciones informáticas añadiendo un carácter semántico y ontológico a la *World Wide Web* (WWW)

Capítulo 7

Diseño e Implementación

Una vez establecida la arquitectura del sistema, el siguiente paso es realizar el diseño e implementación del proyecto. Para la implementación de parte de la lectura de los datos iniciales se reutilizarán una serie de clases desarrolladas por Borja Vázquez Barreiros en su Trabajo Fin de Grado [16], por lo que se comenzará explicando el diseño del acceso a datos. A continuación se mostrarán los diseños de las secciones restantes que conforman el algoritmo, finalizando por el diseño de la aplicación web que proporciona la interfaz de usuario.

7.1. Diagramas de clases

En este proyecto se ha querido realizar un diseño modular, que permitiese realizar el reemplazo de cualquiera de los componentes implicando la mínima carga de trabajo posible. A continuación se muestran y describen los diagramas de clases de los diferentes paquetes, haciendo una breve explicación de la estructura y los patrones aplicados.

7.1.1. Lectura de los datos

El algoritmo desarrollado por este proyecto tiene como entrada un grafo y un conjunto de ejecuciones del mismo (*log*) sobre las cuales se ejecuta el algoritmo. Se ha decidido dar soporte a la carga de los datos de archivos, por lo que es necesario un módulo que realice dicha funcionalidad.

En la Figura 7.1 se representa el diagrama de clases correspondiente a este paquete, que consta de una interfaz y una clase que la implementa. Se ha de mencionar la aparición de dos paquetes adicionales, explicados en apartados posteriores, debido al uso, por la lectura de los datos, de las clases en ellos situadas. La implementación de la interfaz también utiliza dos clases adicionales, pertenecientes al algoritmo ProDiGen, que ofrecen la funcionalidad de la lectura e interpretación de los datos.

Se ha decidido realizar un diseño con interfaz, aplicando el patrón *Strategy* (Figura 6.5, capítulo Arquitectura y Herramientas de Desarrollo), para facilitar la posibilidad de un futuro reemplazo de la implementación de lectura de ficheros por una versión que no dependa de clases como *CMIndividual*, provenientes de otros proyectos y que se deben adaptar al que se está desarrollando.

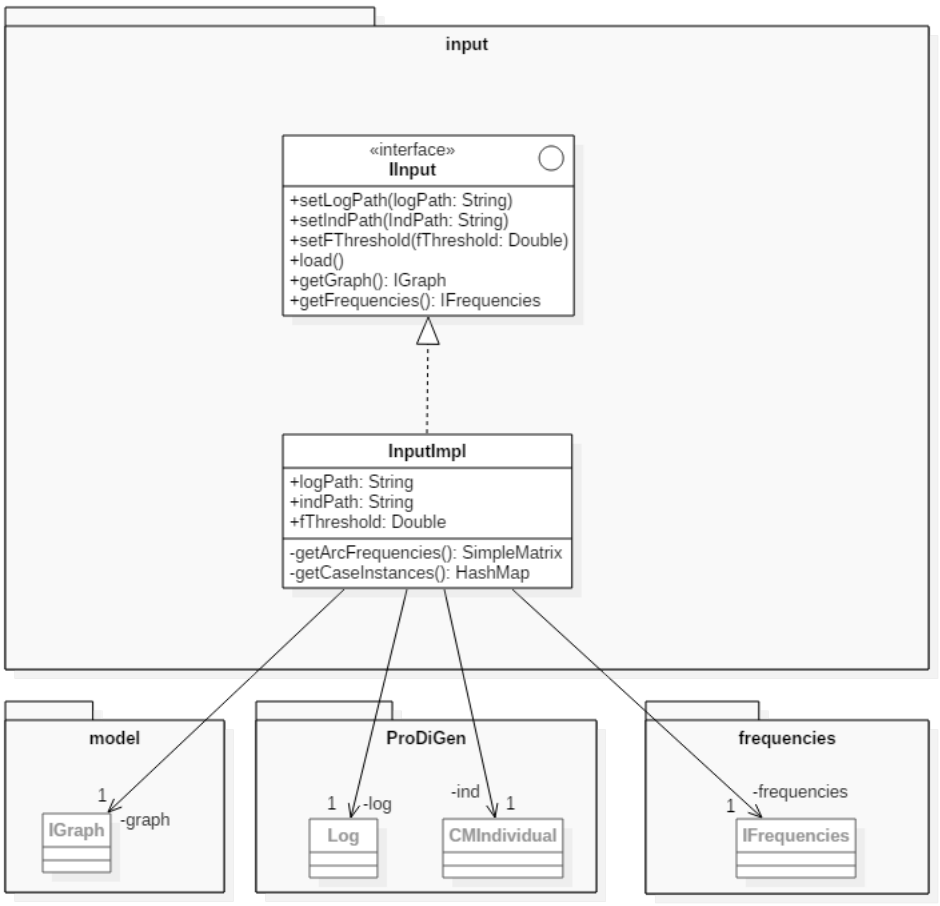

Figura 7.1: Diagrama de clases del paquete de lectura de los datos.

7.1.2. Algoritmo

En este proyecto se partió de la definición en pseudocódigo del algoritmo w-find [10]. Esta definición establece las bases del algoritmo, abstrayéndose de operaciones concretas, y definiendo el esqueleto que proporciona el funcionamiento básico.

Por este motivo se ha decidido separar la implementación de esta base del modelo creado para su funcionamiento y mejora. En este paquete (Figura 7.2) se encuentra la codificación del algoritmo w-find [10] y se reflejan también los dos patrones aplicados al proyecto, *Strategy* y *Observer*. Con el patrón Strategy se podrá ampliar la aplicación con otras aproximaciones realizadas por otros algoritmos de forma sencilla, únicamente tendrá que implementar la interfaz *IGraphMining* y ser instanciados en lugar de la clase *WFind*.

El patrón *Observer* se ha implementado por la necesidad de la aplicación de recibir notificaciones sobre los cambios de estado del algoritmo, indicando así la fase en la que está; lo cual es muy útil para el desarrollo de la interfaz gráfica, ya que recibe las notificaciones del algoritmo cuando tienen lugar, y las muestra al usuario. Por este motivo se ha creado la clase abstracta *AlgorithmNotifier* de la que hereda el algoritmo, y la clase abstracta *StateListener* que implementarán los observadores que quieran ser notificados y que deberán suscribirse.

En el diagrama aparecen dos referencias a clases utilizadas pertenecientes a otros paquetes, que serán explicados posteriormente, su aparición es debido al uso que hace de ellas la implementación del algoritmo w-find.

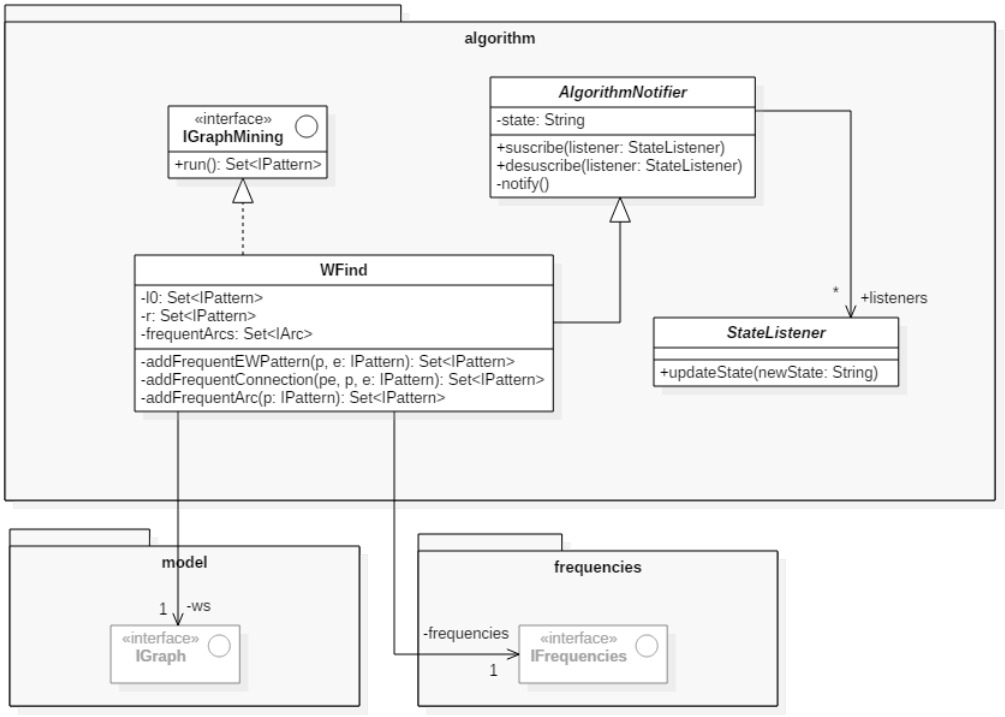

Figura 7.2: Diagrama de clases del paquete del algoritmo.

7.1.3. Medición de frecuencias

Para la ejecución de este algoritmo es necesaria la medición de la frecuencia de los patrones, nodos o arcos. Debido a la dificultad de esta tarea, se ha decidido crear un paquete separado que gestiona dichas consultas. Este paquete se muestra a continuación.

Siguiendo la estructura del patrón *Strategy* (Figura 6.5 del capítulo Arquitectura y Herramientas de Desarrollo), el paquete principal contiene una interfaz que establece los métodos para la medición de la frecuencia de cada elemento. De esta forma, si se desea reemplazar la implementación en un futuro, no se tendrá que modificar cada fragmento de código donde se utilice esta clase, sino que solo se modificará el constructor al inicio de la aplicación, ya que en toda la ejecución se utiliza la misma instancia.

La implementación de la medición de las frecuencias realizada en este proyecto se ha introducido en un subpaquete debido a que está basada en la clase *CMIndividual* ajena a esta aplicación. La funcionalidad que detecta qué arco se ha ejecutado, necesaria para el algoritmo de medición, pertenece a la implementación de ProGiGen [12], por lo que aparecen las clases externas *CaseInstance* y *CMMarking*. La clase *ArcFrequencyMeter* es la encargada de calcular las frecuencias de los arcos al inicio del algoritmo, mientras que *FreqCMIndividual* realiza el cálculo de las frecuencias de un patrón o nodo, utilizando *ParserGraph* para el análisis de los registros (*log*).

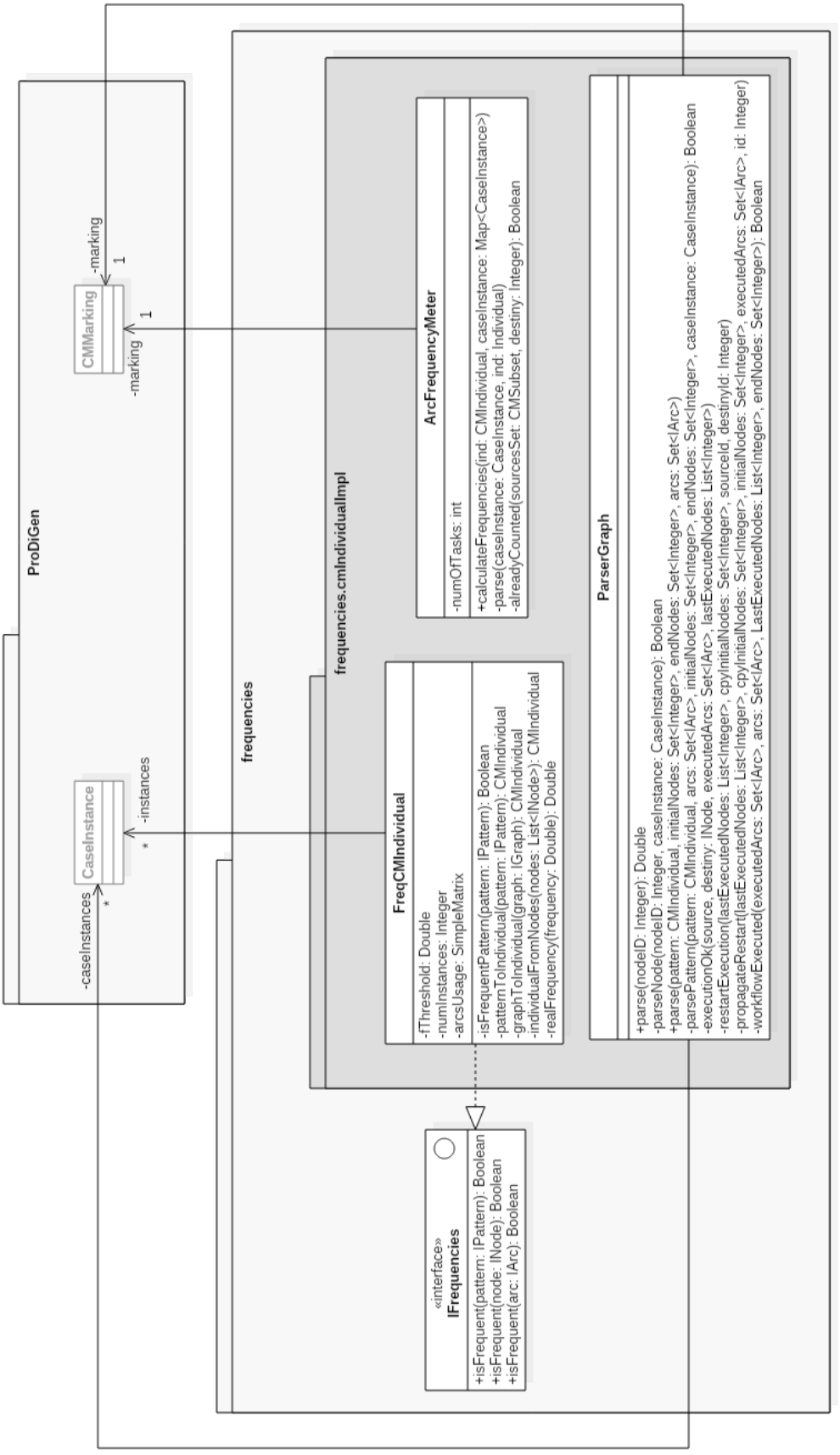

Figura 7.3: Diagrama de clases del paquete de la medición de las frecuencias.

7.1.4. Modelo

El modelo del proyecto está formado por las clases que se utilizan para representar los datos con los que se trabaja, en este caso se trata de grafos, patrones, arcos y nodos, con sus respectivas funcionalidades. Este paquete es uno de los de mayor importancia del proyecto, conteniendo la estructura de los datos y la mayor parte de la lógica.

Como se puede observar en la Figura 7.4, este paquete está formado por cuatro interfaces correspondientes al grafo, patrón, arco y nodo. Cada interfaz tiene su implementación, aplicando así una separación conceptual entre los métodos necesarios por la clase que represente cada objeto, y los métodos necesarios para la codificación, que podrán variar si se desea cambiar la implementación.

Debido a que la clase *Node* es un contenedor de datos sin funcionalidades adicionales, y que la clase *CMTask* de ProDiGen [12] contenía los mismos datos, se ha decidido efectuar una herencia sobrescribiendo los métodos para adaptarlos a la implementación de este proyecto.

En cuanto a la clase *Pattern*, conceptualmente representa un patrón del grafo sobre el que se realizarán operaciones adicionales, pero que contienen la misma información, incluso estructurada de la misma forma, que un grafo; es decir, un patrón es un tipo de grafo. Por este motivo se ha realizado una herencia de *Graph* que libera de la implementación de ciertas funcionalidades a la clase *Pattern*.

Además de las clases que representan la estructura del grafo, distinguimos dos clases adicionales: *Simplifier* y *PatternCombinations*. Esta última se corresponde con la clase que efectúa los cálculos necesarios para la generación de las combinaciones de un patrón. Como ya se ha explicado en la Sección 2, es necesario generar las combinaciones de un patrón, y para simplificar así la consulta del código, se ha decidido introducir en una clase toda la funcionalidad necesaria para generar estas combinaciones. Por último, la clase *Simplifier* es la encargada de realizar las operaciones de filtrado y simplificación de los resultados del algoritmo, siendo en este caso el método que extrae los patrones de mayor tamaño del conjunto de patrones resultado.

Se ha de mencionar que, tanto en la Figura 7.3 como en la Figura 7.4 se ha realizado una simplificación de los métodos menos significativos, y de algunos argumentos, con el fin de mejorar la visualización del mismo en un formato no digital. Si se desea consultar el diagrama completo de cualquiera de los dos modelos, estos se incluyen en el *Apéndice C*.

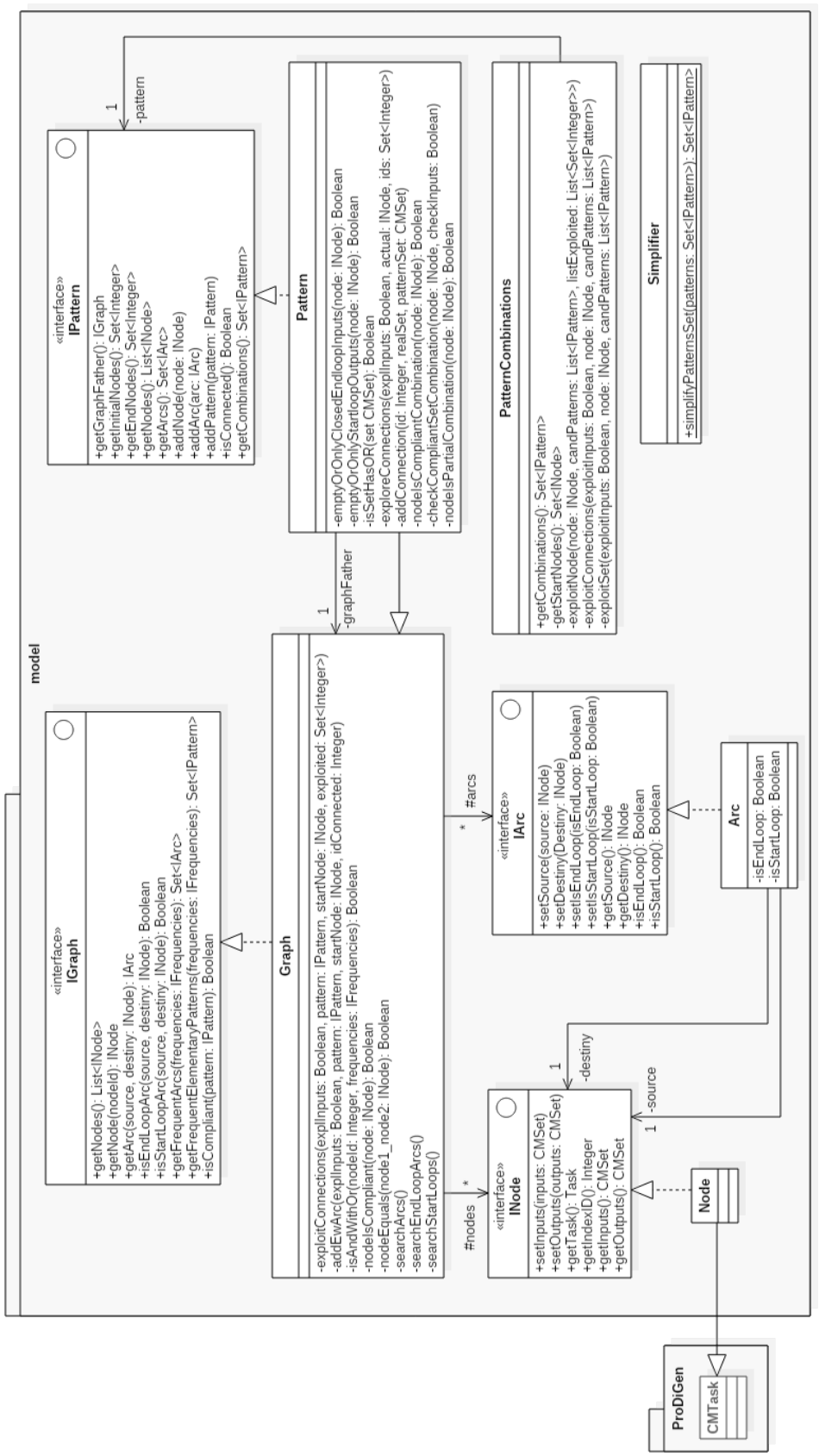

Figura 7.4: Diagrama de clases del paquete del modelo.

7.1.5. Aplicación Web

En la figura 7.5 se muestra el diagrama de clases correspondiente con la aplicación web desarrollada, encargada de mostrar los resultados e interactuar con el usuario. En el diagrama se distinguen diferentes paquetes. *Algorithm* se corresponde con el paquete con contiene toda la lógica del algoritmo. Para facilitar la comprensión se ha incluido únicamente la interfaz con la que interactúa la aplicación web y la clase de la que se extiende para recibir las notificaciones.

El paquete *controller* es el correspondiente al controlador del sistema. Este paquete está formado por la clase *Controller*, que recibe las llamadas de las vistas y devuelve las respuestas necesarias después de realizar las acciones necesarias, y la clase *Helper*, que ayuda al controlador delegando en él las tareas de mayor complejidad como las que requieren llamadas a servicios.

El paquete *fileaccess* contiene una sola clase, con la que se manejarán los ficheros de la aplicación. Y el paquete *dao* contiene las clases que implementan el acceso a datos, con el patrón *Abstract Factory* (6.3, capítulo de Arquitectura y Herramientas de Desarrollo) implementado, de forma que se ha elaborado la clase abstracta *DAOFactory* que generará los productos, en este caso las clases que implementarán el acceso a datos, y que deberán heredar de las clases abstractas *DAOGraph*, *DAOLog* y *DAOResults*. De esta manera, una de las implementaciones se tiene en el subpaquete *dao.mysql*. Con este patrón, en caso de querer cambiar a otro sistema gestor de base de datos, o incluso a almacenamiento a fichero, solo se tendrían que construir las clases que heredasen de las mencionadas anteriormente y modificar la instanciación de la fábrica de DAOs.

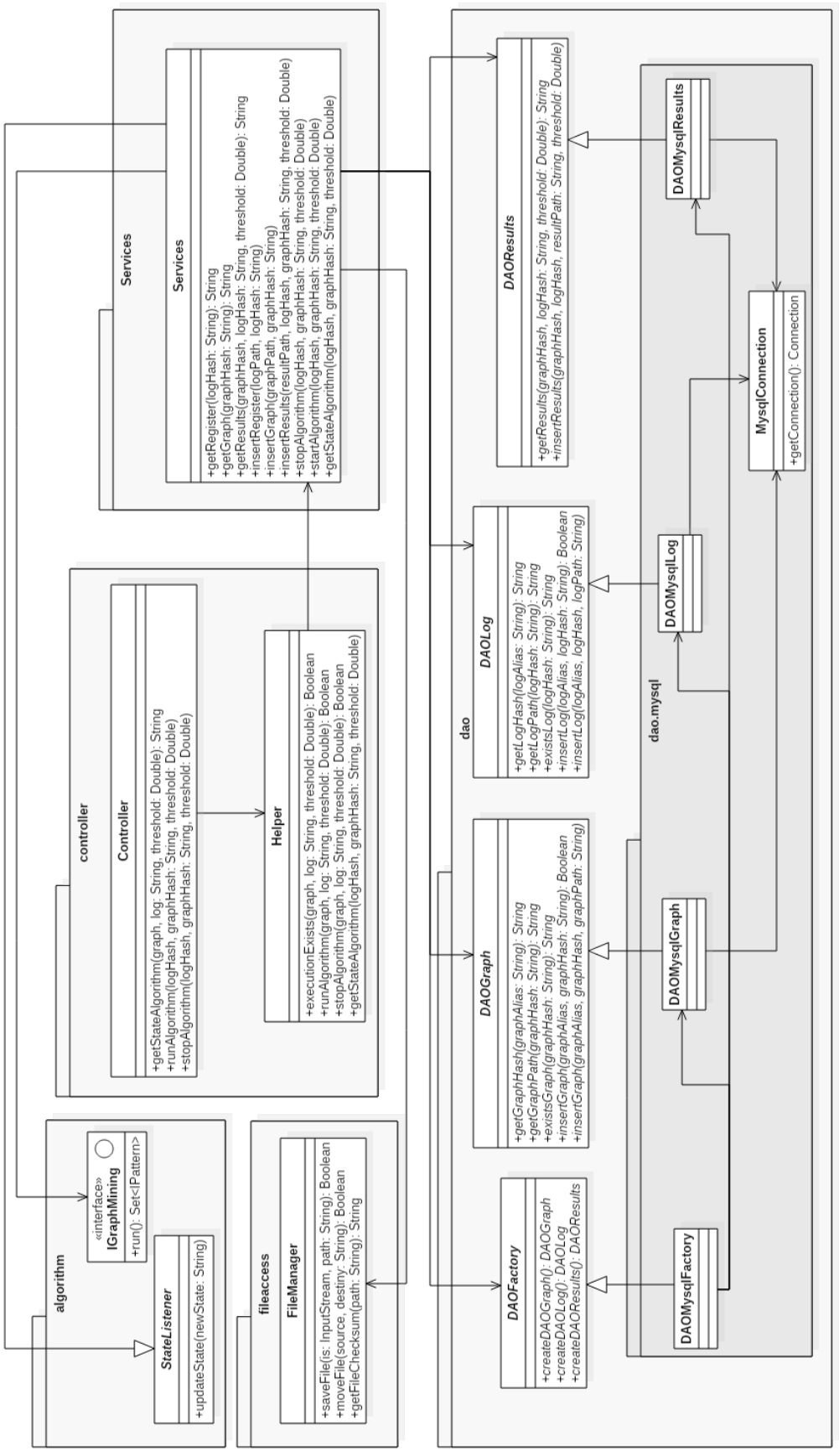

Figura 7.5: Diagrama de clases de la aplicación web.

7.1.6. Diagrama de flujo de datos

Los diagramas de flujo de datos son una representación gráfica del flujo de los datos que tiene lugar en un sistema. Presentan una visión del sistema de alto nivel, que permite realizar una abstracción de la implementación, y conocer el funcionamiento básico y el flujo de información que tiene lugar en el sistema.

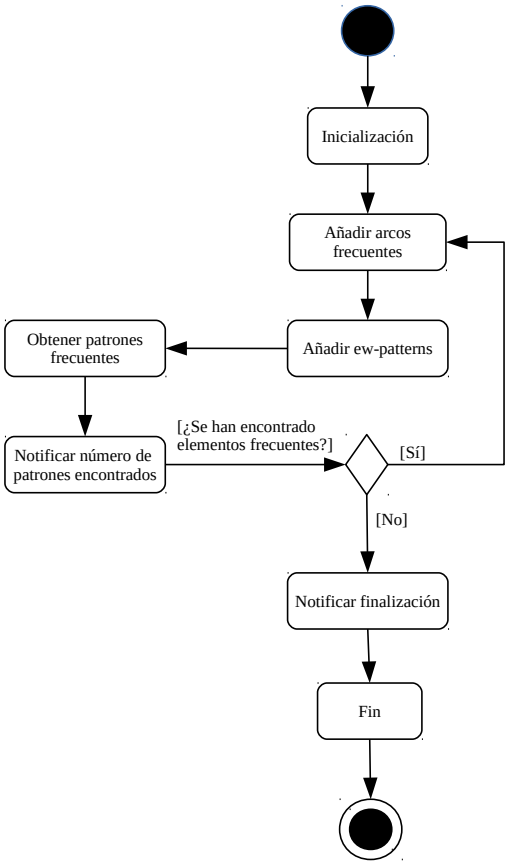

Figura 7.6: Diagrama de flujo de datos del algoritmo final.

La Figura 7.6 se corresponde con el diagrama de flujo de datos del algoritmo una vez aplicado el patrón *Observer*, donde se pueden apreciar las diferentes fases del mismo. El flujo comienza con la inicialización del algoritmo, donde se cargan los arcos y *ew*-patterns frecuentes, a continuación comienza el proceso de iteración, en el que se expandirán los patrones recién obtenidos añadiendo arcos frecuentes, en el siguiente paso se realiza la expansión agregando los *ew*-patterns. Para finalizar la iteración se obtienen, de los patrones generados, aquellos que son frecuentes, y se notifica a los observadores del número de patrones encontrados hasta el momento. En este momento se realiza una comprobación, si se ha encontrado algún patrón frecuente nuevo se reinicia el bucle, y en caso negativo se notifica a los observadores, y se finaliza el algoritmo.

Este diagrama propone una idea a alto nivel de la implementación del algoritmo, una vez aplicados los patrones diseñados para el mismo, y como se puede observar, la única diferencia con la Figura 3.2 (Capítulo 1. Algoritmo) es la inclusión de dos fases: *Notificar número de patrones*

encontrados y *Notificar finalización*. Estas dos fases han sido añadidas para satisfacer el diseño realizado, en el que se incluía el patrón *Observer*, y consisten en una notificación a los observadores que se han suscrito de los patrones encontrados y del resultado, respectivamente. Se ha de mencionar que, en la implementación realizada, un posible observador será la interfaz gráfica, que actualizará la información sobre el estado de la ejecución a medida que esta cambia.

7.2. Diagramas de interacción

Los diagramas de secuencia o interacción son una forma de representar la comunicación entre un cliente (actor) y los objetos (clases), lo que implica la especificación de los sucesos en el interior del sistema como respuesta a un evento.

A continuación se muestran los diagramas de interacción correspondientes a los casos de uso especificados en el capítulo 4, divididos en los mismos subsistemas. Se ha de mencionar que los casos de uso como el *CU_IA -04 Seleccionar entrada* no se representarán con diagramas de interacción debido a que son casos de uso que simplemente engloban otros, y serán estos últimos de los que se realicen diagramas.

7.2.1. Subsistema de interacción con el algoritmo

CU_IA-01 Seleccionar origen del grafo y CU_IA-02 Seleccionar origen del log

Los casos de uso *CU_IA-01* y *CU_IA-02* tienen la única diferencia de que el formato de archivo aceptado es diferente, ya que en cada uno se selecciona un archivo distinto. Pero debido a que el resto de la secuencia es similar, se realizará únicamente el diagrama de interacción para uno de ellos.

En la figura 7.7 se encuentra el diagrama de interacción para el *CU_IA-01 Seleccionar origen del grafo*, y que también puede representar el caso de uso *CU_IA-02 Seleccionar origen del log*. El flujo comienza con el usuario realizando la acción para seleccionar archivo, la vista inicial procesa la solicitud abriendo, a través del navegador, un explorador de archivos en el sistema operativo, para que así el usuario pueda navegar por los directorios hasta encontrar el grafo y seleccionarlo. Acto seguido el explorador devuelve el archivo a la vista, que lo procesa y comprueba el formato del mismo.

En este diagrama de interacción se ha representado el caso en el que un usuario selecciona mal el archivo, realizando posteriormente la acción correcta, para reflejar así ambos comportamientos. La vista le muestra un error al usuario, que vuelve a realizar la acción para seleccionar un archivo. Al igual que en el otro caso se abre el explorador, el usuario navega y selecciona el archivo deseado; en este caso la comprobación de formatos es correcta por lo que se acepta el archivo, finalizando así el flujo de interacción.

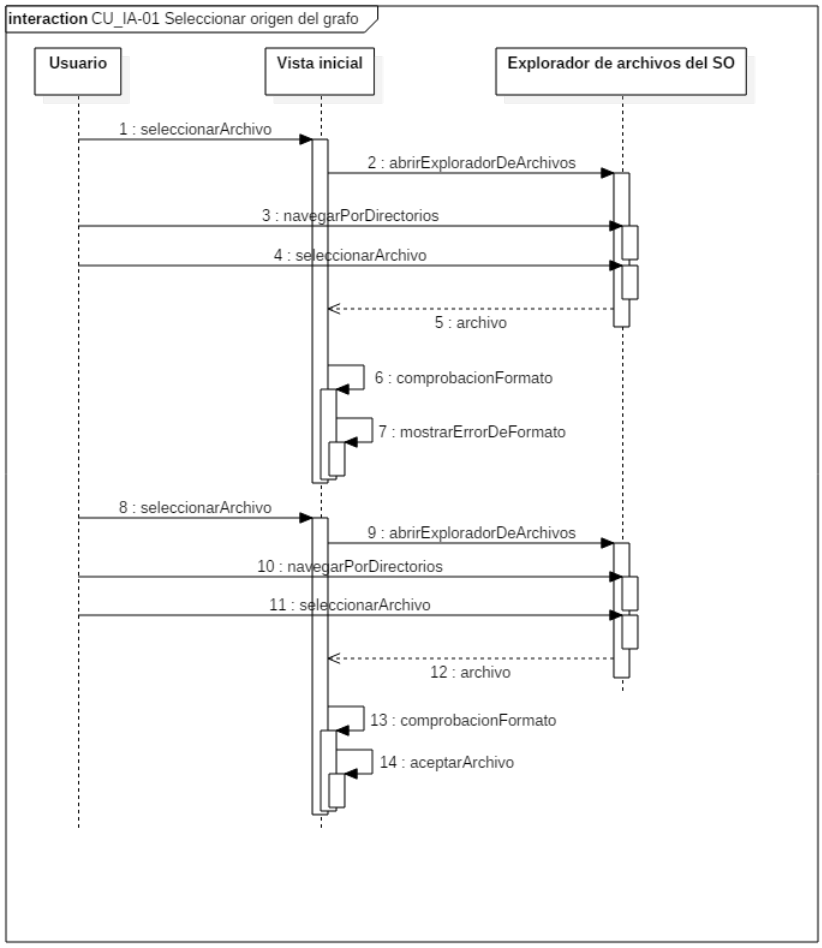

Figura 7.7: Diagrama de interacción del caso de uso *CU_IA-01*.

CU_IA-03 Especificar umbral

El diagrama de interacción de la Figura 7.8 modela el flujo de señales del caso de uso *CU_IA-03 Especificar umbral*, en el que el usuario introduce un valor para el umbral. En este diagrama, la vista inicial analiza el formato, e informa del error debido a que este no es correcto; acto seguido el usuario vuelve a introducir un valor, esta vez con un formato correcto, y la vista acepta el valor introducido, terminando así la interacción.

CU_IA-05 Iniciar algoritmo

El caso de uso más complejo es, sin duda alguna, el de iniciar el algoritmo. Ante este evento la aplicación debe realizar diversas acciones, como comprobar la existencia de la ejecución, comprobar la existencia del grafo o del log, almacenarlos si no existen, lanzar la el algoritmo, comprobar periódicamente el estado del mismo desde la vista, etc.

Para no complicar excesivamente el diagrama, en la Figura 7.9 se ha reflejado una ejecución en la que el usuario inicia el algoritmo desde la vista principal, ésta comprueba la validez de los parámetros, y realiza la llamada al *helper*. Acto seguido, el *helper* propaga las llamas al servicio web que comprueba si ya existe una ejecución con esos parámetros, para lo cual instancia, a través de *DAOFactory*, el *DAO* de los resultados, realiza la petición y éste la propaga a la base de datos.

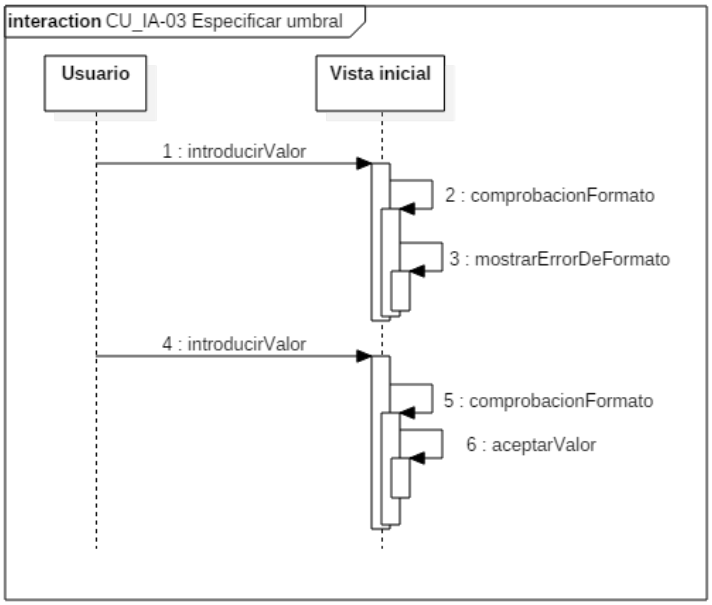

Figura 7.8: Diagrama de interacción del caso de uso *CU_IA-03*.

En este diagrama, el resultado es negativo, por lo que el controlador llama, de nuevo con ayuda del *helper*, al servicio que inserta el grafo, y acto seguido al que inserta los *logs*. Estos servicios instancian, a través de *DAOFactory*, los *DAOs* de grafos y registros, y hacen la petición de almacenamiento.

A continuación, el *helper* hace una llamada al servicio que inicia el algoritmo, y éste crea la instancia, suscribe al controlador a los eventos, e inicia el algoritmo. Acto seguido, el controlador crea la vista de resultados y se la devuelve al cliente.

En este momento, el algoritmo notifica periódicamente al controlador el número de patrones encontrados en cada iteración, y por último en la finalización. Mientras se ejecuta el algoritmo, la vista de los resultados hace peticiones al controlador para saber el estado del algoritmo, éste le devuelve los patrones frecuentes recibidos, y, cuando finaliza, la ruta al archivo con los patrones. La vista de resultados, ante cada respuesta, actualiza la vista con los nuevos datos.

CU_IA-06 Detener algoritmo

El diagrama de interacción relacionado con el caso de uso *CU_IA-06 Detener algoritmo* se muestra en la Figura 7.10. Este comienza con la instrucción del usuario de detener el algoritmo, que pasa por el controlador y se propaga al servicio web correspondiente, el cual recupera el hilo de ejecución del algoritmo, y lo para. Acto seguido, el controlador crea la vista de los resultados indicando que se ha parado, y le devuelve la vista al usuario.

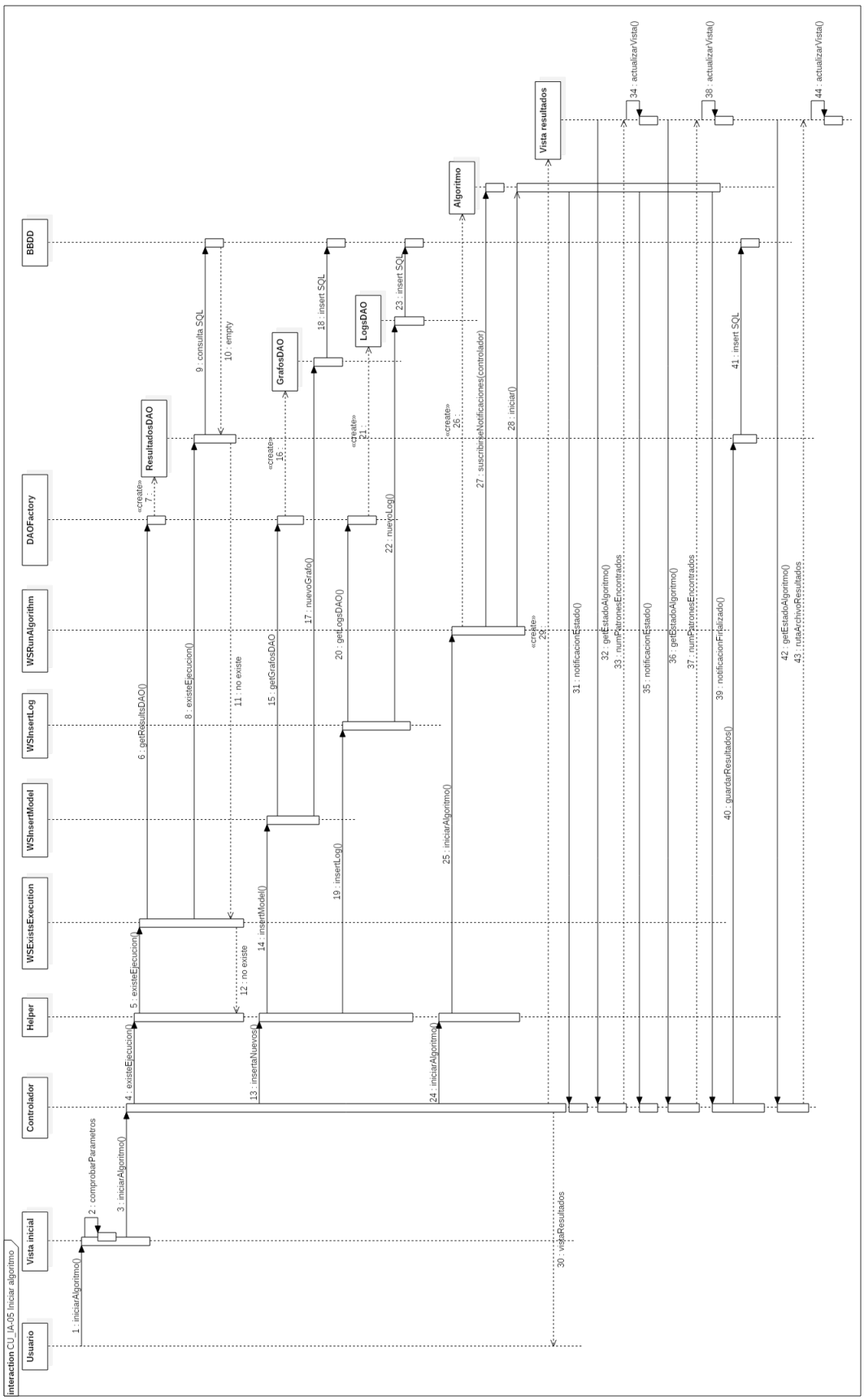

Figura 7.9: Diagrama de interacción del caso de uso *CU_IA-05*.

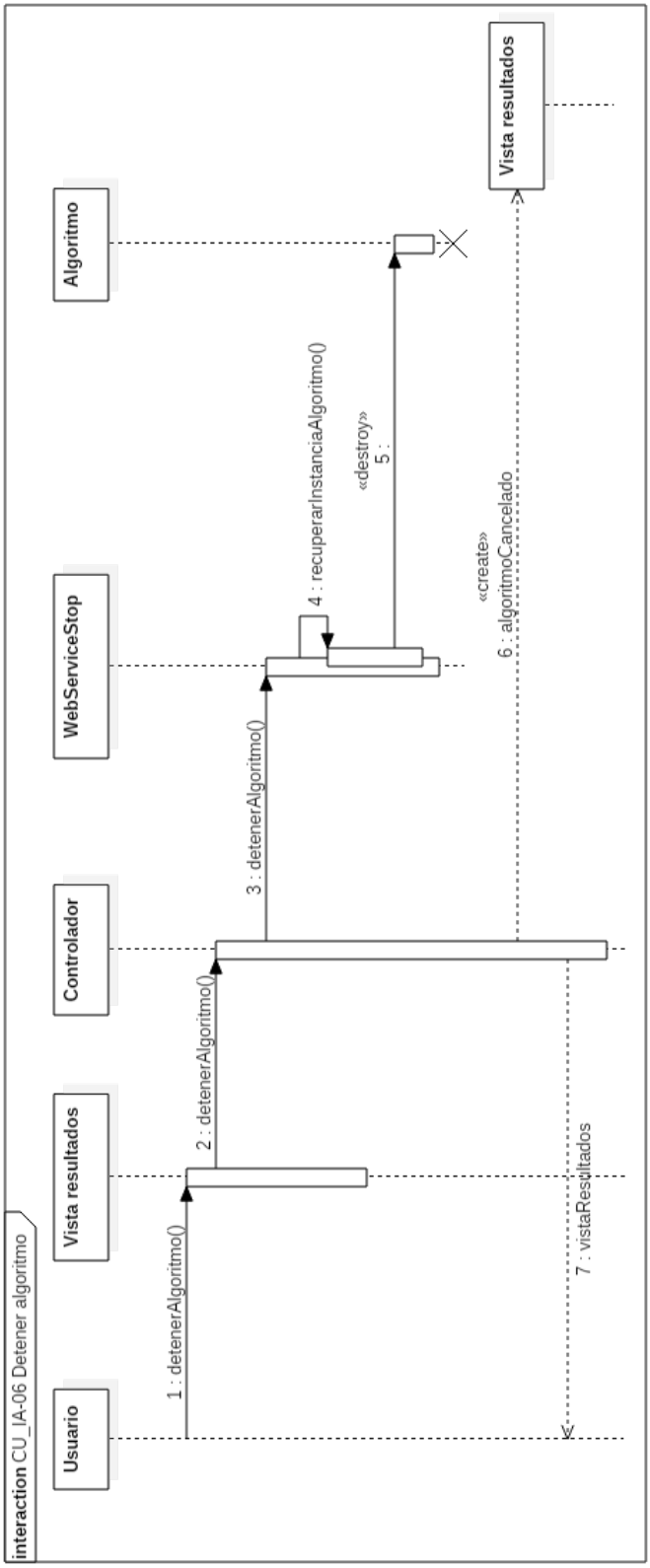

Figura 7.10: Diagrama de interacción del caso de uso *CU_IA-06*.

7.2.2. Subsistema de visualización de los resultados

CU_VR-01 Visualizar workflow completo

En este diagrama de interacción, que se muestra en la figura 7.11, el usuario se comunica con la vista ejecutando la acción para visualizar el flujo de trabajo (o grafo) completo, la vista ajusta el zoom de forma que se visualice el workflow entero y la actualiza.

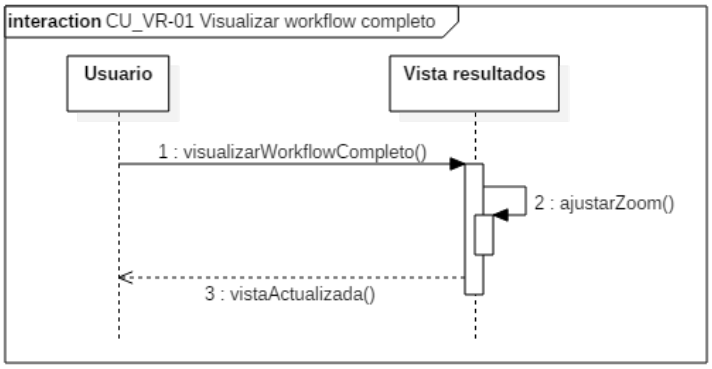

Figura 7.11: Diagrama de interacción del caso de uso *CU_VR-01*.

CU_VR-02 Visualizar patrón concreto

El diagrama de interacción de este caso de uso se refleja en la Figura 7.12, en este se puede apreciar el flujo de interacciones que se producen, comenzando con la petición de visualizar el patrón concreto. La vista calcula las coordenadas del patrón en la representación, ajusta el zoom y devuelve la vista actualizada.

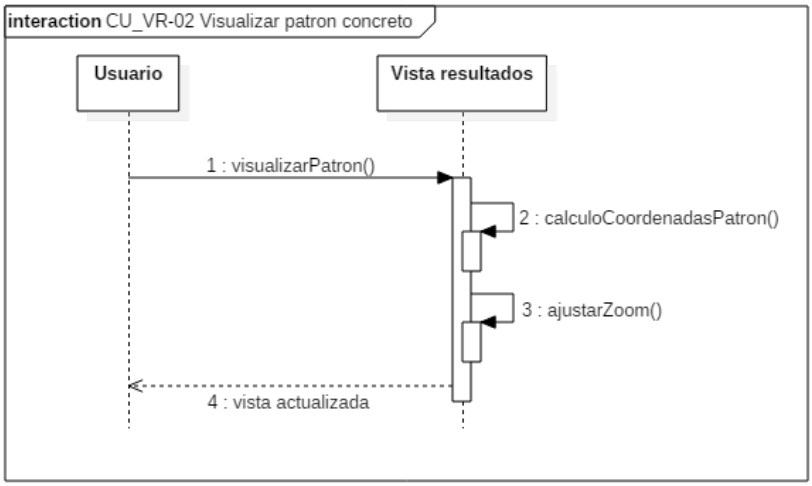

Figura 7.12: Diagrama de interacción del caso de uso *CU_VR-02*.

CU_VR-03 Variar el zoom

La secuencia de interacciones para este caso de uso se refleja en la Figura 7.13. En él se muestran las dos posibilidades de acción: en primer lugar, el usuario realiza la acción de disminuir el zoom, la vista aplica los cálculos necesarios y actualiza la vista; y en segundo lugar, el usuario puede

realizar la acción de aumentar el zoom, siguiendo la misma secuencia de operaciones, pero para aumentar el tamaño de los elementos.

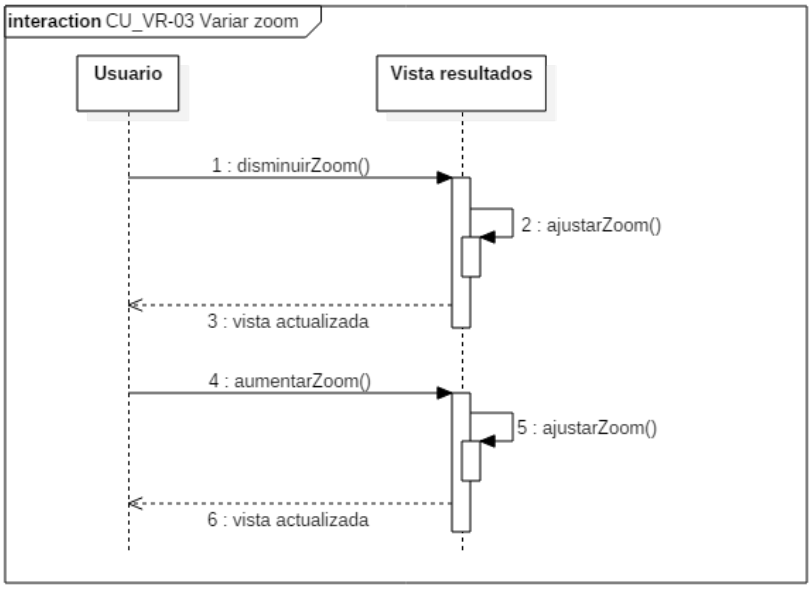

Figura 7.13: Diagrama de interacción del caso de uso *CU_VR-03*.

CU_VR-04 Listar todos los patrones

El diagrama de interacción para mostrar todos los patrones del conjunto de resultados se muestra en la Figura 7.14. En este diagrama el usuario realiza la petición a la vista, ésta muestra todos los patrones que estaban ocultos y actualiza la vista.

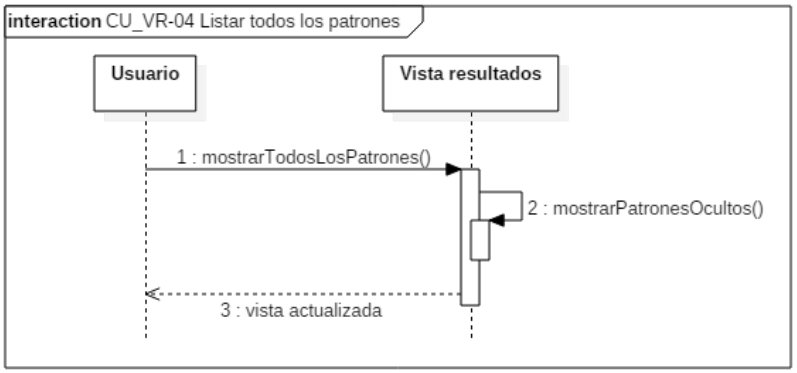

Figura 7.14: Diagrama de interacción del caso de uso *CU_VR-04*.

CU_VR-05 Listar patrones que contengan una tarea

En el diagrama de interacción de la Figura 7.15 se muestra la secuencia de acciones relacionada con el caso de uso *CU_VR-05*. En este diagrama, el usuario comienza enviándole una señal a la vista para seleccionar una tarea, ésta se actualiza para que el usuario pueda seleccionarla. Acto seguido, el usuario envía a la vista la tarea, que busca los patrones que la contienen y oculta los que no lo hacen. Por último, actualiza la vista con los nuevos patrones a mostrar.

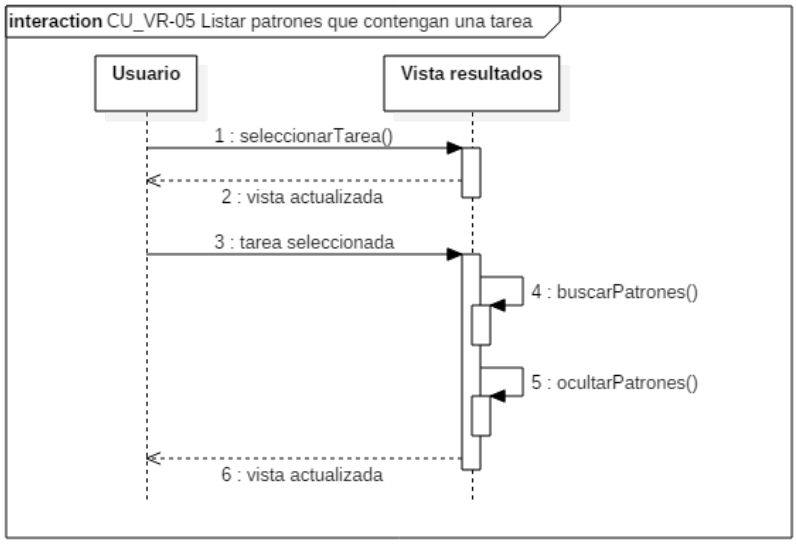

Figura 7.15: Diagrama de interacción del caso de uso *CU_VR-05*.

CU_VR-06 Listar patrones que contengan un conjunto de tareas y CU_VR-07 Listar patrones que contengan alguna tarea dado un conjunto

Los requisitos *CU_VR-06* y *CU_VR-07* pueden ser representados por el mismo diagrama de interacción, ya que la interacción realizada en ambos casos es similar, con la única diferencia de que en el primer requisito la búsqueda de patrones descarta todos los que no contengan todas las tareas del conjunto, mientras que el segundo requisito descarta los patrones que no contienen ninguna.

El flujo sería una petición del usuario para seleccionar un conjunto, la vista se actualiza y el usuario selecciona el conjunto, le envía la información a la vista, junto con la decisión para el caso de uso que es, y ésta realiza la búsqueda y oculta los patrones que no cumplen el requisito. Por último, se actualiza la vista.

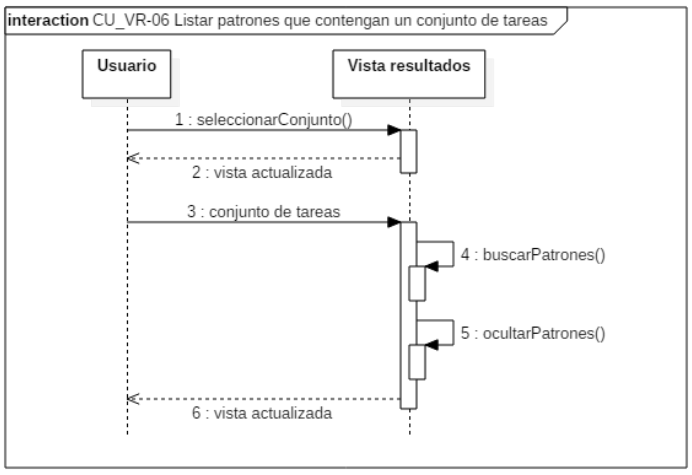

Figura 7.16: Diagrama de interacción del caso de uso *CU_VR-06*.

CU_VR-09 Ordenar patrones por tamaño, CU_VR-10 Ordenar patrones por peso y CU_VR-11 Ordenar patrones por frecuencia

Para la especificación de la secuencia correspondiente a estos tres casos de uso se utilizará el mismo diagrama de interacción, esto es debido a que la única diferencia reside en que el peso para el caso de uso CU_VR-10 necesita ser calculado, mientras que los valores de los otros casos de uso se obtienen de forma directa.

En el diagrama se puede observar cómo el usuario comienza enviando una señal a la vista para que ordene los resultados, ésta, en función de los parámetros recibidos, realizará el cálculo del peso (esto solo para el caso de uso CU_VR-10) si se ordena por este campo, y luego ordenará los patrones, finalizando con una actualización de la vista.

Se ha de mencionar que el cálculo del peso solo se realiza la primera vez que el usuario solicita la ordenación, ya que los valores se almacenan con los patrones para usos en posteriores ordenaciones.

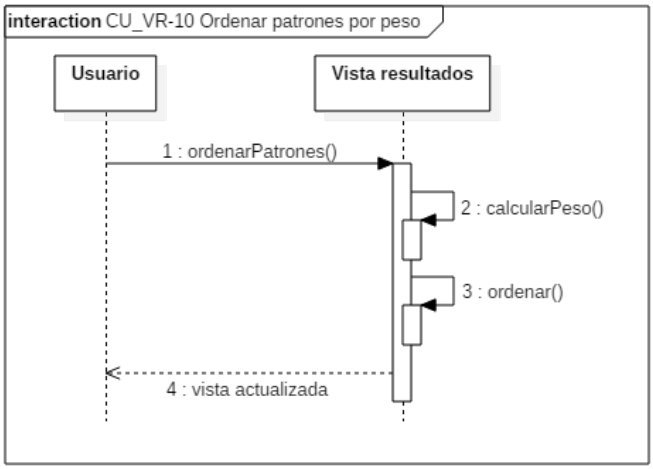

Figura 7.17: Diagrama de interacción del caso de uso *CU_VR-10*.

7.3. Diseño de la interfaz gráfica

Como se ha mencionado anteriormente y especifica el requisito *RNF-01*, la interfaz gráfica debe ser implementada de forma ajena al motor que ejecuta el algoritmo. Por este motivo se decide separar el diseño de la lógica y vistas de la interfaz, del diseño interno del algoritmo, explicado en la sección anterior.

Cuando uno de los objetivos del proyecto consiste en la realización de un mecanismo de interacción con el usuario debe existir un proceso de creación minucioso y que asegure la mayor comodidad a los usuarios de cara a la utilización del sistema. Esto es debido a que, a pesar de la complejidad y utilidad del producto, si la interfaz con la que deben interactuar los clientes no resulta cómoda y usable, es muy probable que el proyecto fracase.

Por esta razón, la realización de la interfaz de usuario es un proceso que se debe realizar con sumo cuidado. En este proyecto se creó una interfaz comenzando con un prototipo del papel, y centrándose en que la aplicación cumpla lo máximo posible con los 10 principios de usabilidad de

Nielsen[1] y sometiéndola a pruebas como los cuestionarios SUS[2] (*System Usability Scale*)

Para minimizar el esfuerzo de realizar cambios en la implementación de la interfaz durante la etapa de diseño, se ha comenzado realizando bocetos en papel, y luego una elaboración de maquetas. En la Figura 7.18, se muestra un *mockup*[3] o maqueta de la ventana de la interfaz que mostraría los resultados, en la que reside la mayor complejidad de esta sección.

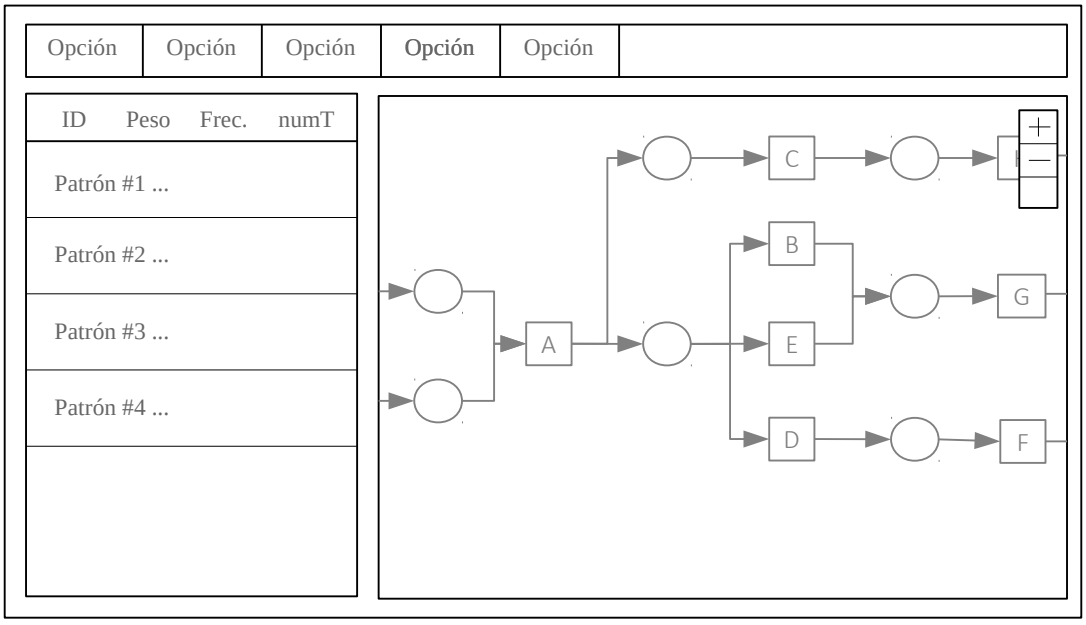

Figura 7.18: *Mockup* de la pantalla principal de la interfaz gráfica.

En ella se puede observar la sección de los resultados a mano izquierda, con un menú con las opciones en la parte superior, y la representación del modelo en la derecha. Esta distribución ha sido diseñada basándose en la regla de la proporción áurea, dando unas proporciones, en anchura, aproximadas del 70 % para una sección, y el 30 % para otra.

Tras un análisis de este diseño y el diseño de las otras secciones de la aplicación web, en el que se consultó la opinión de usuarios sin conocimientos avanzados en la informática, se continuó realizando el diseño directamente en el navegador.

Se ha contemplado la opción de seguir un proceso de diseño en el que primero se realizarían un *wireframe*[4] con el ordenador, después se crearían las maquetas con un editor de imágenes, con sus respectivos *storyboards*[5]. Pero esta ha sido descartada debido a que en un proyecto de este tipo la

[1] Jakob Nielsen es un referente en cuanto a temas de usabilidad web, los 10 principios de usabilidad de Nielsen son 10 reglas que Nielsen diseñó a partir del estudio de 249 problemas de usabilidad.

[2] El cuestionario SUS, desarrollado en 1968, ofrece un test fácil de completar, sencillo de puntuar y con el que se obtiene un único número representando la medida de usabilidad del sistema a probar.

[3] Un *mockup* es un modelo a escala o tamaño real de un diseño, utilizado para la demostración y evaluación del diseño facilitando así la realización de cambios en el mismo.

[4] Un *wireframe*, o esquema de página, es una guía visual para representar la estructura base, o esqueleto, de un sitio web. Representa el diseño enfocándose en el tipo de información mostrada, y supone un paso intermedio entre el prototipo básico y el diseño final.

[5] *Storyboard*, conjunto de diseños de la web que representan la secuencia y disposición del contenido a lo largo de un caso de uso. Esencialmente son similares a los diagramas de interacción, pero muestran la secuencia visual ocurrida al realizar acciones con la interfaz gráfica.

interfaz es un elemento secundario, aunque importante, el cual se utilizará para interactuar con el algoritmo y parte principal del Trabajo Fin de Grado que se está desarrollando.

7.4. Diseño de la Base de Datos

En este apartado se mostrará y explicará el diagrama de clases realizado para la base de datos del sistema, así como el modelo creado con MySQL Workbench para su creación. Con el objetivo de almacenar los registros y grafos ya cargados, y sus respectivas ejecuciones, se ha diseñado una base de datos que almacene la información necesaria sobre los grafos y *logs*, siguiendo el siguiente esquema:

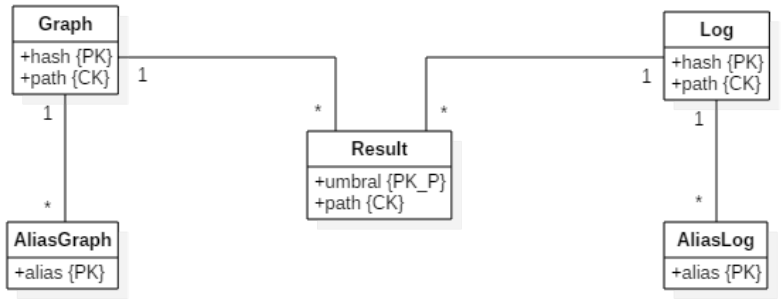

Figura 7.19: Diagrama de clases de la base de datos.

En la Figura 7.19 se refleja la relación entre las diferentes tablas, existen las tablas *Graph* y *Log*, que poseen el hash[6] del fichero y la ruta al mismo, como se observa ambos son únicos para cada entrada, pero se ha escogido como clave primaria el hash debido a su característica no cambiante a pesar de que se mueva el archivo. La anotación CK indica que se trata de una clave candidata, es decir, podría ser una clave primaria.

Cada tabla recién explicada posee una tabla con los alias, esta únicamente contiene, como clave primaria, el alias asignado, ya que no se podrán repetir los alias de cada elemento.

Por último, la tabla *Result* contiene los resultados de las ejecuciones exitosas, esta almacena el umbral como clave primaria parcial, junto a los identificadores de log y grafo, y el camino al archivo, que también se trata de una clave candidata.

Las relaciones entre las tablas de log y grafo con sus respectivos alias es de *1..** debido a que un grafo o log puede tener mucho alias, pero un alias pertenece solo a un log o grafo. Y la tabla *Results* tiene una relación **..1* con *Graph* y *Log* por el mismo motivo, dado un grafo se pueden tener muchas ejecuciones, pero una ejecución pertenece únicamente a un grafo.

A continuación, en la Figura 7.20, se muestra el modelo resultado del esquema previamente explicado, para su creación se tuvo que convertir las relaciones *1..** introduciendo la clave primaria del lado con multiplicidad 1 en el extremo con multiplicidad *.

[6]El hash se obtiene a partir de aplicar una función *hash* o de resumen sobre el archivo, generando una clave identificativa del mismo.

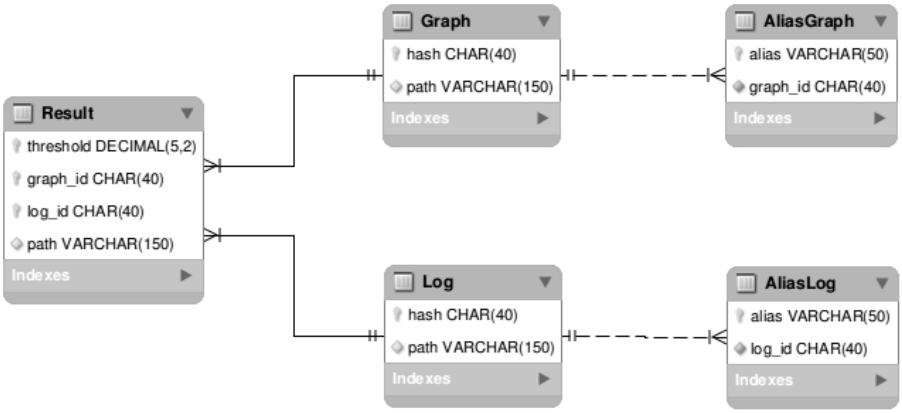

Figura 7.20: Diagrama de clases de la base de datos.

Capítulo 8

Validación y pruebas

En esta sección se documentará el contenido y resultado de la etapa de validación y pruebas. El objetivo principal de esta etapa [7] es verificar no sólo que el software funciona de acuerdo a su especificación, sino también comprobar que cumple con las expectativas del cliente. Para ello es necesario elaborar una serie de pruebas para asegurar el cumplimiento tanto de los requisitos funcionales, como de los no funcionales, prestando atención al criterio de validación realizado en la especificación de cada uno.

De acuerdo a la metodología de Programación Extrema, la cual ha sido aplicada en este proyecto, una de las fases más importantes de todo proyecto software es la validación y ejecución de pruebas. Siguiendo esta metodología se dividen las pruebas en tres grupos:

- Pruebas unitarias: ideadas para verificar que el código se comporta de forma esperada y cumple con los requisitos especificados.
- Pruebas de integración: concebidas para validar el funcionamiento en conjunto de los diferentes módulos unitarios del sistema.
- Pruebas de validación del sistema o aceptación: encargadas de validar que el software desarrollado en el proyecto cumple con las expectativas del cliente.

Debido a que el algoritmo presenta una solución novedosa en su ámbito de aplicación, la importancia que le da la metodología escogida a las pruebas es un punto a favor. Además, la creación de un algoritmo de estas características, formado por una gran variedad de módulos, tiene una alta probabilidad de que haya un fallo en alguno de estos elementos que provoque un mal funcionamiento del mismo.

A lo largo del ciclo de vida del proyecto se han realizado diferentes fases de pruebas, esto es debido a que la metodología aplicada distribuye el tiempo en diferentes intervalos en los que se desarrolla por completo un conjunto de funcionalidades. A continuación se comentarán las pruebas más relevantes que se han realizado en dichas fases, así como la validación efectuada del algoritmo una vez finalizado.

También se especificarán las pruebas a las que se ha sometido la interfaz, para verificar su nivel de usabilidad, cumpliendo así con el requisito *RNF-06*.

8.1. Pruebas unitarias

A continuación se especificarán las pruebas realizadas para asegurar el cumplimiento de cada requisito, tanto funcional como no funcional. Los requisitos de información no conllevan pruebas debido a que su cometido es el almacenamiento de información para das soporte a algún requisito, por lo que al evaluar la completitud del requisito asociado, se evalúa la validez del requisito de información.

8.1.1. Requisitos Funcionales

- **Requisito funcional RF_IA-01: *Seleccionar origen del grafo***
  - **Descripción de la prueba**: se realiza la interacción del usuario, en la que este elige la opción de seleccionar un grafo y se carga un fichero.
  - **Resultado esperado**: la vista se actualiza con el fichero en el campo del formulario.
  - **Estado**: cumplido.
- **Requisito funcional RF_IA-02: *Seleccionar origen del log***
  - **Descripción de la prueba**: se realiza la interacción del usuario, en la que este elige la opción de seleccionar un log y se carga un fichero.
  - **Resultado esperado**: la vista se actualiza con el fichero en el campo del formulario.
  - **Estado**: cumplido.
- **Requisito funcional RF_IA-03: *Especificar umbral de frecuencia***
  - **Descripción de la prueba**: se realiza la interacción del usuario, en la que este define el umbral como un número con un decimal entre 0 y 100.
  - **Resultado esperado**: la vista actualiza el campo del formulario con el nuevo valor.
  - **Estado**: cumplido.
- **Requisito funcional RF_IA-04: *Iniciar algoritmo***

  Para validar este requisito, tal como indica su criterio de validación, se deben probar los casos en los que el formato de los argumentos es erróneo, por lo que se efectuarán diferentes pruebas, descritas a continuación:

  - **Descripción de la prueba**: el usuario pulsa el botón para iniciar el algoritmo, pero un subconjunto de los argumentos está vacío.
  - **Resultado esperado**: la interfaz comunica al usuario la falta de los argumentos.
  - **Estado**: cumplido.
  - **Descripción de la prueba**: el usuario pulsa el botón de iniciar el algoritmo, pero todos los argumentos están vacíos.
  - **Resultado esperado**: la interfaz comunica al usuario la falta de los argumentos.
  - **Estado**: cumplido.

- **Descripción de la prueba**: el usuario pulsa el botón de iniciar el algoritmo, pero la extensión del archivo cargado para el grafo no es correcto.
- **Resultado esperado**: la interfaz comunica al usuario el error en la extensión del archivo del grafo.
- **Estado**: cumplido.

- **Descripción de la prueba**: el usuario pulsa el botón de iniciar el algoritmo, pero la extensión del archivo cargado para el log no es correcto.
- **Resultado esperado**: la interfaz comunica al usuario el error en la extensión del archivo del log.
- **Estado**: cumplido.

- **Descripción de la prueba**: el usuario pulsa el botón de iniciar el algoritmo, pero el formato del umbral no es de un número perteneciente al intervalo (0-100] y con un decimal como máximo.
- **Resultado esperado**: la interfaz comunica al usuario el error en el formato del umbral.
- **Estado**: cumplido.

- **Requisito funcional RF_IA-05:** ***Detener algoritmo***
  - **Descripción de la prueba**: el usuario pulsa la opción de detener el algoritmo cuando esta se muestra una vez iniciado el mismo.
  - **Resultado esperado**: el algoritmo se detiene y se informa al usuario.
  - **Estado**: cumplido.
- **Requisito funcional RF_IA-06:** ***Detectar patrones máximos que superen el umbral***
  - **Descripción de la prueba**: a partir de un conjunto resultado del algoritmo, se recogen los patrones que no estén contenidos en otros del conjunto.
  - **Resultado esperado**: los patrones devueltos son aquellos que no forman parte de otro patrón del conjunto.
  - **Estado**: cumplido.
- **Requisito funcional RF_IA-07:** ***Detección de patrones con bucles***
  - **Descripción de la prueba**: ante una entrada de un grafo con bucles y sus respectivos registros, se lanza el algoritmo para obtener los patrones frecuentes con un umbral que debería devolver un bucle como patrón frecuente.
  - **Resultado esperado**: el algoritmo devuelve patrones con bucles que superan el umbral de frecuencia.
  - **Estado**: cumplido.
- **Requisito funcional RF_IA-08:** ***Detección de arcos de inicio y fin de bucle***
  - **Descripción de la prueba**: dado un grafo con bucles, se lanza el algoritmo que busca los arcos de inicio y fin de bucle.

  - **Resultado esperado**: los arcos identificados como inicio y fin de bucle son los correctos.
  - **Estado**: cumplido.

- **Requisito funcional RF_IA-09:** ***Generación de combinaciones***
  - **Descripción de la prueba**: dado un patrón con diferentes caminos posibles, se generan las combinaciones del mismo.
  - **Resultado esperado**: el algoritmo devuelve un conjunto de patrones formado por las combinaciones del patrón inicial.
  - **Estado**: cumplido.

- **Requisito funcional RF_IA-10:** ***Reutilización de resultados previamente obtenidos***
  - **Descripción de la prueba**: se lanza, desde la interfaz gráfica, el algoritmo con unos parámetros con los que ya se ha realizado una ejecución previamente.
  - **Resultado esperado**: el algoritmo detecta la ejecución previa, recoge los resultados y los muestra por pantalla.
  - **Estado**: cumplido.

- **Requisito funcional RF_VR-01:** ***Visualizar workflow completo***
  - **Descripción de la prueba**: una vez visualizados los resultados, el usuario pulsa el botón para visualizar el grafo completo.
  - **Resultado esperado**: La interfaz realiza el ajuste de zoom necesario para que el grafo completo se pinte en pantalla.
  - **Estado**: cumplido.

- **Requisito funcional RF_VR-02:** ***Visualizar patrón concreto***
  - **Descripción de la prueba**: una vez visualizados los resultados, el usuario pulsa el botón para visualizar el un patrón concreto.
  - **Resultado esperado**: la interfaz realiza el ajuste de zoom necesario para centrar el patrón en la pantalla.
  - **Estado**: cumplido.

- **Requisito funcional RF_VR-03:** ***Variar el zoom***
  - **Descripción de la prueba**: el usuario realiza peticiones de aumento y disminución de zoom.
  - **Resultado esperado**: la vista responde correctamente, aumentando el zoom en el modelo cuando esta petición es activada, y disminuyéndolo en caso contrario.
  - **Estado**: cumplido.

- **Requisito funcional RF_VR-04:** ***Listar todos los patrones***
  - **Descripción de la prueba**: el usuario escoge la opción de listar todos los patrones del conjunto de resultados.

- **Resultado esperado**: los patrones mostrados por la interfaz son el conjunto completo del resultado.
- **Estado**: cumplido.

- **Requisito funcional RF_VR-05:** ***Listar patrones que contengan una tarea***
  - **Descripción de la prueba**: el usuario escoge la opción de listar los patrones que contengan una tarea y selecciona la tarea.
  - **Resultado esperado**: la vista actualiza los resultados mostrando únicamente los patrones que contienen la tarea seleccionada.
  - **Estado**: cumplido.
- **Requisito funcional RF_VR-06:** ***Listar patrones que contengan un conjunto de tareas***
  - **Descripción de la prueba**: el usuario selecciona la opción para listar los patrones que contengan un conjunto de tareas y selecciona el conjunto.
  - **Resultado esperado**: la vista se actualiza mostrando los patrones resultado que contienen todas las tareas del conjunto.
  - **Estado**: cumplido.
- **Requisito funcional RF_VR-07:** ***Listar patrones que contengan alguna tarea dado un conjunto***
  - **Descripción de la prueba**: el usuario selecciona la opción para listar los patrones que contengan alguna tarea de un conjunto y selecciona el conjunto.
  - **Resultado esperado**: la vista se actualiza mostrando los patrones resultado que contienen alguna de las tareas del conjunto.
  - **Estado**: cumplido.
- **Requisito funcional RF_VR-08:** ***Ordenar patrones por tamaño***
  - **Descripción de la prueba**: el usuario selecciona la opción de ordenar tareas por tamaño.
  - **Resultado esperado**: la vista se actualiza ordenándolas de forma ascendente o descendente según el usuario haya seleccionado.
  - **Estado**: cumplido.
- **Requisito funcional RF_VR-09:** ***Ordenar patrones por peso***
  - **Descripción de la prueba**: el usuario selecciona la opción de ordenar tareas por peso.
  - **Resultado esperado**: la vista se actualiza ordenándolas de forma ascendente o descendente según el usuario haya seleccionado.
  - **Estado**: cumplido.
- **Requisito funcional RF_VR-10:** ***Ordenar patrones por frecuencia***
  - **Descripción de la prueba**: el usuario selecciona la opción de ordenar tareas por frecuencia.

- **Resultado esperado**: la vista se actualiza ordenándolas de forma ascendente o descendente según el usuario haya seleccionado.
- **Estado**: cumplido.

8.1.2. Requisitos No Funcionales

- **Requisito no funcional RNF-01:** ***Interfaz gráfica independiente del motor***
  - **Descripción de la prueba**: es posible realizar una separación del algoritmo de la interfaz sin necesidad de modificar el código del algoritmo.
  - **Resultado esperado**: existe una forma simple de modificar el entorno sin necesidad de cambiar nada relacionado con el algoritmo.
  - **Estado**: cumplido, gracias al patrón MVC el algoritmo es completamente independiente de la aplicación web.
- **Requisito no funcional RNF-02:** ***Facilidad de pruebas***
  - **Descripción de la prueba**: se comprueba que el código desarrollado cuenta con un trato de excepciones en casos que no se deberían dar.
  - **Resultado esperado**: el código contiene excepciones en los casos necesarios.
  - **Estado**: cumplido.
- **Requisito no funcional RNF-03:** ***Multiplataforma***
  - **Descripción de la prueba**: se comprueba si la aplicación puede ser ejecutada en Linux y en Windows realizando ambos despliegues.
  - **Resultado esperado**: la aplicación funciona correctamente en ambos.
  - **Estado**: cumplido, la realización del proyecto en Java, junto a HTML5 para la parte web permiten el despliegue de la misma independientemente de la arquitectura sobre la que se haga.
- **Requisito no funcional RNF-04:** ***Extensibilidad***
  - **Descripción de la prueba**: se comprueba la facilidad de cambio de algoritmo en la aplicación
  - **Resultado esperado**: el cambio de algoritmo no supone la realización de modificaciones complejas en el código.
  - **Estado**: cumplido, gracias al patrón Strategy aplicado se puede cambiar la implementación del algoritmo fácilmente.
- **Requisito no funcional RNF-05:** ***Documentación***
  - **Descripción de la prueba**: se revisa la documentación, en búsqueda de secciones cuyo aporte no sea lo suficientemente importante.
  - **Resultado esperado**: no se encuentran apartados que carecen de importancia.
  - **Estado**: cumplido.

- **Requisito no funcional RNF-06:** ***Interfaz usable***
  - **Descripción de la prueba**: se realiza un cuestionario SUS con tres usuarios, explicado en un apartado posterior.
  - **Resultado esperado**: la aplicación consigue un resultado aceptable en la escala de usabilidad.
  - **Estado**: cumplido.
- **Requisito no funcional RNF-07:** ***Seguridad en la aplicación***
  - **Descripción de la prueba**: se intentan realizar inyecciones SQL en la aplicación web.
  - **Resultado esperado**: el sistema no se ve afectado debido a las medidas de seguridad implantadas.
  - **Estado**: completado.

8.2. Pruebas de integración

Una vez realizadas las pruebas unitarias, y teniendo la certeza del correcto funcionamiento de los distintos módulos por separado es necesario verificar que la unión de ciertas funcionalidades no presentan errores, debido a que el correcto funcionamiento individual de cada componente no garantiza que en conjunto se comporten del modo esperado.

A continuación se detallan las pruebas de integración realizadas, identificadas como PI-*N*.

- **PI-1:** ***lectura del registro y modelo y creación del grafo a partir de ellos***
  - **Descripción de la prueba**: se lanzará el algoritmo con un log y una especificación de un modelo.
  - **Resultado esperado**: el algoritmo lee los ficheros y construye correctamente el grafo con la implementación de la clase *Graph*.
  - **Estado**: cumplido.
- **PI-2:** ***expansión de un patrón con arcos frecuentes***
  - **Descripción de la prueba**: se analizará la expansión de un patrón con arcos frecuentes en la que se debe efectuar correctamente la extracción de arcos frecuentes, ampliación de un patrón con un arco y comprobación de restricciones.
  - **Resultado esperado**: los patrones obtenidos son los esperados.
  - **Estado**: cumplido.
- **PI-3:** ***expansión de un patrón con*** *ew*-***patterns***
  - **Descripción de la prueba**: se analizará la expansión de un patrón con otros *ew*-patterns en la que se debe efectuar correctamente la extracción de arcos frecuentes, ampliación de un patrón con otro y comprobación de restricciones.
  - **Resultado esperado**: los patrones obtenidos son los esperados.

- **Estado**: cumplido.

- **PI-4:** ***Frecuencia de un patrón con bucles***
  - **Descripción de la prueba**: se comprobará la frecuencia de un patrón con bucles, pero enlazando las funcionalidades que buscan los arcos de bucle en el grafo, genera las combinaciones, y analiza la frecuencia de una combinación.
  - **Resultado esperado**: la frecuencia devuelta se corresponde con la aparición en las trazas del patrón.
  - **Estado**: cumplido.

- **PI-5:** ***Algoritmo***
  - **Descripción de la prueba**: se realizará una prueba del algoritmo integrando todas las funcionalidades.
  - **Resultado esperado**: los patrones obtenidos forman superan el umbral de frecuencia pasado.
  - **Estado**: cumplido.

- **PI-6:** ***aplicación web***
  - **Descripción de la prueba**: se realizará una prueba de lanzamiento del algoritmo desde la aplicación web, este deberá devolver los resultados correctamente, y la interfaz realizar la visualización de los mismos.
  - **Resultado esperado**: una vez finaliza el algoritmo, los patrones obtenidos se muestran en la interfaz.
  - **Estado**: cumplido.

8.3. Pruebas de interfaz gráfica

Como se ha comentado previamente, en la prueba unitaria de verificación del requisito *RNF-06: Interfaz usable*, para validar la usabilidad de la interfaz gráfica diseñada se ha desarrollado un test o prueba que efectuarán 3 usuarios que no hayan tenido contacto previo con la aplicación.

Este método es conocido por el nombre de cuestionario SUS [15], y consiste en la obtención de una puntuación perteneciente a una escala en función de la experiencia de diferentes usuarios. Una característica de esta prueba reside en la sencillez que debe presentar el cuestionario, haciendo que la sensación de aburrimiento, cansancio o desgana del usuario al responder las preguntas sea la mínima. Además, las preguntas deben ir alternadas entre una positiva y una negativa, es decir, no debe haber dos preguntas seguidas cuya respuesta positiva suponga un beneficio para la interfaz a analizar.

Los usuarios deben contestar a las preguntas una vez han terminado de utilizar la aplicación, pero antes de realizar ningún informe o discusión, y deben responder rápido la pregunta, obteniendo así las primeras impresiones que se han llevado. La forma de realizar la puntuación es elegir un valor entero del 1 al 5, escogiendo 3 en caso de que el usuario no sepa contestar a la pregunta. Por último, una vez realizado el cuestionario, se obtiene el resultado de la siguiente forma: Se deben

sumar primero las contribuciones de cada pregunta. La contribución de cada una valdrá entre 0 y 4. Así, para las preguntas impares la contribución será el valor de la respuesta menos 1, mientras que en las preguntas pares (negativas) la contribución será 5 menos el valor de la respuesta. Se multiplica la suma de los resultados por 2.5 para obtener el valor global del SUS. El resultado estará entre 0 (nada usable) y 100 (totalmente usable).

A continuación se muestran las preguntar pertenecientes al cuestionario diseñado para evaluar la usabilidad de la interfaz:

1. Creo que el sitio web es fácil de usar.
2. Me sentí perdido a la hora de encontrar dónde estaba cada acción.
3. La visualización del modelo resultado es cómoda.
4. Encontré el sitio innecesariamente complejo.
5. No me resultó difícil encontrar las opciones que buscaba.
6. Necesitaría apoyo de un experto para navegar por el portal.
7. Fui capaz de completar las acciones que deseaba.
8. No me pareció agradable la forma de interactuar con el modelo.
9. Me encuentro cómodo utilizando el sitio web.
10. No volvería a frecuentar la aplicación de no ser la única opción.

Se ha realizado el cuestionario a tres usuarios ajenos al ámbito de aplicación y se ha calculado el resultado de sus respuestas. En la tabla 8.1 se detallan los valores de las respuestas recibidas:

Preguntas	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Usuario 1	5	1	4	1	5	1	5	1	5	1
Usuario 2	5	1	4	1	5	1	5	2	5	1
Usuario 3	5	1	5	1	5	2	4	1	4	1
Contribución	12	12	10	12	12	11	11	11	11	12
Contribución media	4	4	3.3	4	4	3.6	3.6	3.6	3.6	4
Final (sobre 100)	94.25									

Tabla 8.1: Puntuaciones obtenidas en la realización del cuestionario SUS.

8.4. Validación del algoritmo

Una vez se dispone del sistema completo, y validado por las pruebas unitarias y de integración, es necesario comprobar su comportamiento ante datos reales o sintéticos, pero de complejidad mayor.

Para poder realizar de una forma más segura la validación del comportamiento de los algoritmos creados en este TFG se han sometido tanto al algoritmo (***searchStartEndLoopArcs***), como al algoritmo principal, que genera los patrones frecuentes de un grafo, a una serie de pruebas en

grafos de alta complejidad extraídos de la tesis doctoral en minería de procesos de Ana Karla Alves de Medeiros [3]. También se han realizado pruebas con registros de flujos de trabajo reales, pertenecientes al ámbito del *e-learning*. A continuación, en la tabla de la Figura 8.1, se muestran las características de los grafos sobre los que se han realizado las pruebas.

Net	Figure	Sequence	Choice	Parallelism	Length-One Loop	Length-Two Loop	Structured Loop	Arbitrary Loop	Invisible Tasks	Unbalanced AND-split/join
g2	C.1	✓	✓	✓		✓	✓		✓	
g3	C.2	✓	✓	✓			✓	✓	✓	
g4	C.4	✓	✓	✓		✓				✓
g5	C.6	✓	✓	✓				✓	✓	
g6	C.7	✓	✓	✓		✓			✓	
g7	C.8	✓	✓	✓			✓		✓	
g8	C.9	✓	✓	✓		✓	✓		✓	✓
g9	C.11	✓	✓	✓		✓	✓		✓	
g10	C.13	✓	✓	✓				✓	✓	
g12	C.15	✓	✓	✓		✓		✓	✓	
g13	C.16	✓	✓	✓	✓	✓			✓	✓
g14	C.18	✓	✓	✓			✓		✓	✓
g15	C.20	✓	✓		✓	✓			✓	
g19	C.22	✓	✓	✓		✓			✓	✓
g20	C.24	✓	✓		✓	✓		✓	✓	
g21	C.25	✓	✓					✓	✓	
g22	C.26	✓	✓	✓			✓		✓	✓
g23	C.28	✓	✓	✓		✓				✓
g24	C.30	✓	✓	✓				✓	✓	✓
g25	C.32	✓	✓	✓	✓				✓	

Figura 8.1: Características de los grafos utilizados para las pruebas (Fuente: [3]).

8.4.1. Detección de arcos startLoopArc y endLoopArc

Para verificar el correcto funcionamiento del algoritmo ***searchStartEndLoopArcs***, que detecta los arcos de inicio y fin de bucle en un grafo con este tipo de estructuras, se ha ejecutado sobre los 20 modelos especificados en la Figura 8.1, obteniendo en todos los casos los arcos identificados de forma manual.

En la Figura 8.2 se muestra el flujo de trabajo *g3* de la Tabla 8.1 una vez aplicado el algoritmo ***searchStartEndLoopArcs***. En el flujo de trabajo se pueden apreciar los arcos de inicio y fin de bucle detectados: *(i)* el *startLoopArc* 11 → 12, *(i)* el *endLoopArc* 15 → 2 y *(iii)* los *startLoopArc y endLoopArc* (bucles de longitud 1) 26 → 23, 24 → 21 y 10 → 4.

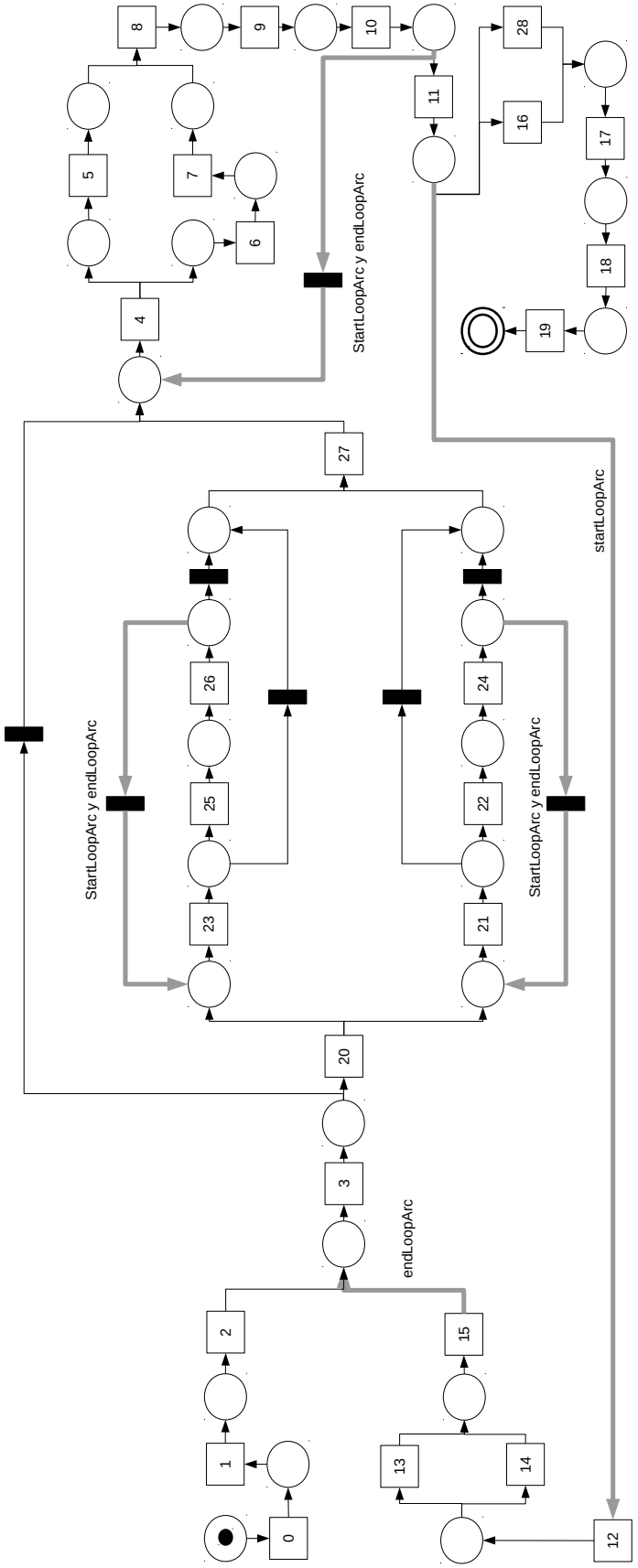

Figura 8.2: Flujo de trabajo $g3$ con los arcos de inicio y fin de bucle detectados por el algoritmo ***searchStartEndLoopArcs***.

8.4.2. Algoritmo general

En cuanto al algoritmo principal, el encargado de la búsqueda de patrones frecuentes en grafos restringidos con bucles, se han realizado dos procesos de validación que se muestran y describen a continuación.

8.4.2.1. Grafos pertenecientes a la Tesis de de Medeiros

Para la validación efectuada con los grafos especificados en la Figura 8.1 se ha lanzado el algoritmo desde la interfaz gráfica, analizando los resultados obtenidos, y asegurándose de que la frecuencia de los patrones obtenidos se correspondía con la calculada por el algoritmo. En la tabla 8.2 se muestra el número de patrones obtenidos para las ejecuciones de los flujos de trabajo mencionados, con cuatro umbrales de frecuencia diferentes.

	Umbral			
	40 %	**55 %**	**70 %**	**85 %**
g2	3	2	2	2
g3	3	4	2	3
g4	3	4	3	4
g5	3	4	2	2
g6	3	2	2	2
g7	4	5	3	3
g8	4	2	2	2
g9	3	2	2	2
g10	1	2	2	2
g12	3	2	2	2
g13	1	2	2	2
g14	3	3	3	3
g15	7	4	4	2
g19	1	2	2	2
g20	6	4	3	3
g21	2	3	2	2
g22	1	2	2	2
g23	2	2	2	2
g24	4	4	3	3
g25	3	5	5	5

Tabla 8.2: Número de patrones resultado de la ejecución de los grafos de 8.1 con diferentes umbrales.

Como ejemplo se propone el flujo de trabajo de la Figura 8.2, sobre el que se ha ejecutado el algoritmo con una frecuencia del 60 %, obteniendo los patrones representados en las Figuras 8.3, 8.4, 8.5 y 8.6.

Se ha de resaltar que, debido a la inexistencia de conjuntos verificados científicamente por la comunidad de investigadores, las pruebas realizadas han tenido que efectuarse sobre flujos de trabajo existentes, y la verificación se ha realizado de forma manual, analizando los registros y comprobando la frecuencia de cada patrón mediante expresiones regulares[1].

[1]Una expresión regular está formada por una secuencia de caracteres, creando un patrón de búsqueda. Principalmente son utilizadas para la búsqueda de patrones de cadenas de caracteres.

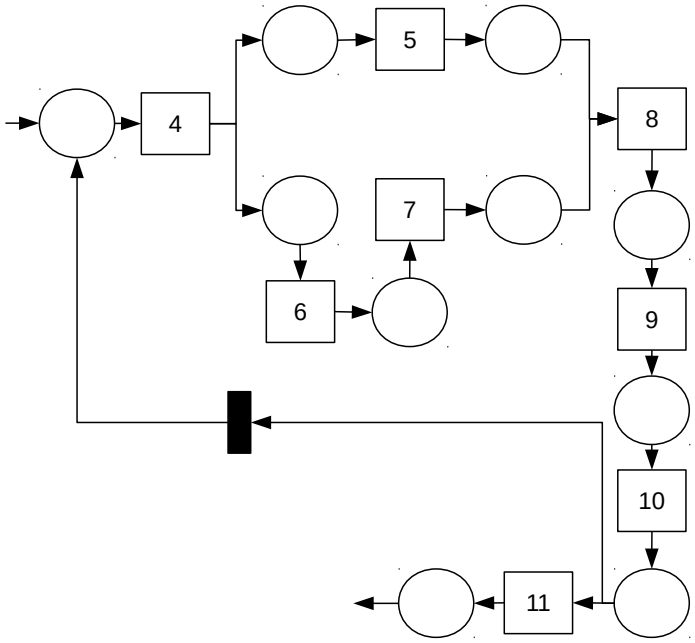

Figura 8.3: Primer resultado del algoritmo w-find mejorado sobre el flujo de trabajo de la Figura 8.2 con un umbral del 60 %.

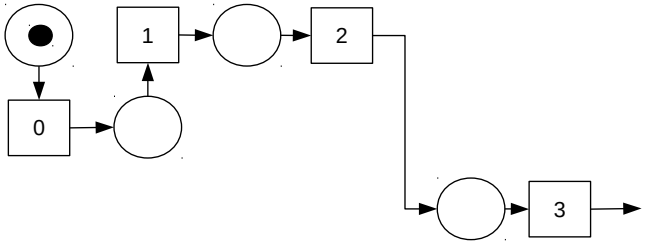

Figura 8.4: Segundo resultado del algoritmo w-find mejorado sobre el flujo de trabajo de la Figura 8.2 con un umbral del 60 %.

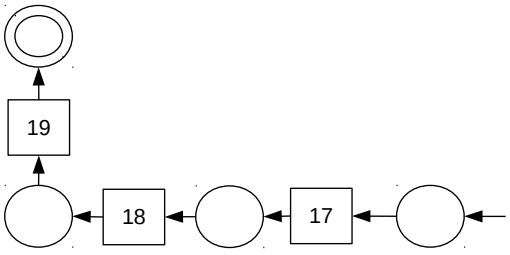

Figura 8.5: Tercer resultado del algoritmo w-find mejorado sobre el flujo de trabajo de la Figura 8.2 con un umbral del 60 %.

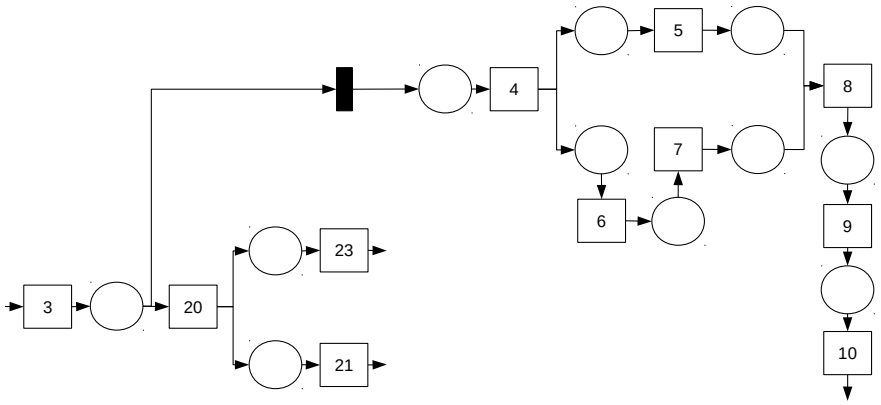

Figura 8.6: Cuarto resultado del algoritmo *w*-find mejorado sobre el flujo de trabajo de la Figura 8.2 con un umbral del 60 %.

8.4.2.2. Registros reales extraídos de SoftLearn

Además de la validación del algoritmo realizada con el conjunto de grafos que se ha explicado en el apartado anterior, se han podido conseguir registros reales extraídos del ámbito del *e-learning* correspondientes a la interacción de diferentes alumnos con las herramientas del curso. En concreto, se trata de registros de la interacción de 72 alumnos durante 4 meses en la asignatura de Tecnología Educativa del Grado en Pedagogía de la Universidad de Santiago de Compostela.

Primeramente, se ha generado el flujo de trabajo correspondiente a los registros con el algoritmo ProDiGen [12] obteniendo el resultado mostrado en la Figura 8.7. Como se puede apreciar, la información que este flujo de trabajo proporciona es escasa, debido a que está formado enteramente por bucles, lo cual provoca que la comprensión de *qué* está pasando en el proceso sea imposible de realizar.

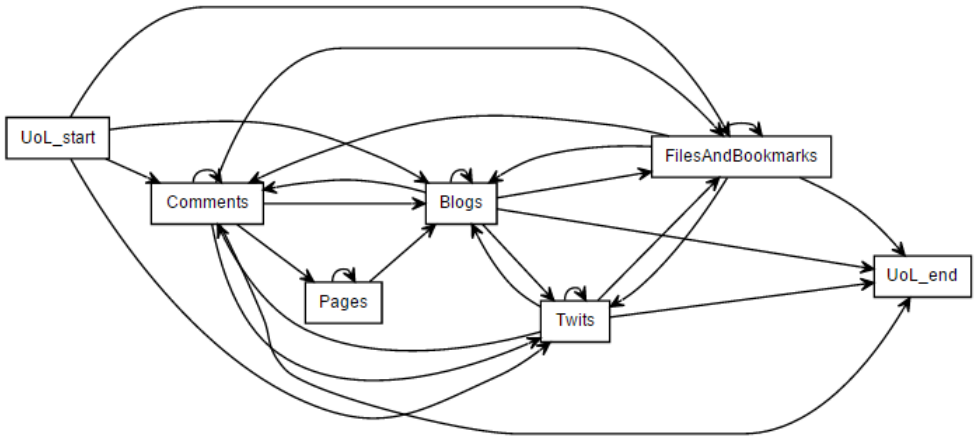

Figura 8.7: Flujo de trabajo extraido de trazas reales en *e-learning*.

Partiendo de los registros mencionados anteriomente, y del flujo de trabajo representado en la Figura 8.7 se ha ejecutado el algoritmo desarrollado en el presente TFG con un umbral del 70 %, obteniendo patrones frecuentes como los que se muestran en la Figura 8.8. Con estos resultados,

la información extraída del flujo de trabajo proporciona un conocimiento mayor, obteniendo así comportamientos frecuentes realizados por los alumnos.

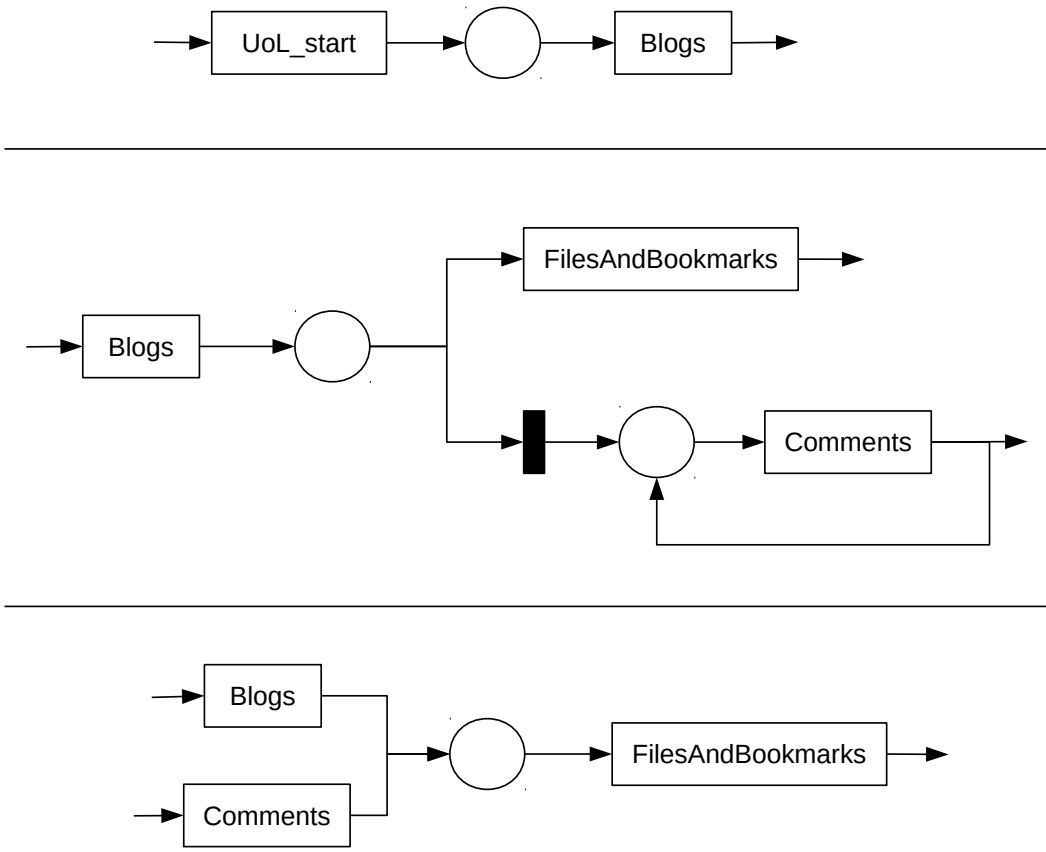

Figura 8.8: Resultados del algoritmo w-find mejorado sobre el flujo de trabajo de la Figura 8.7 con un umbral del 70 %.

Capítulo 9

Conclusiones

Como ya se ha comentado anteriormente, la minería de grafos se podría encuadrar en dos líneas: *(i)* la búsqueda de patrones representativos en el grafo, analizando la aparición de estos dentro del mismo, o *(i)* la búsqueda en las ejecuciones, residiendo la importancia del patrón en el número de ejecuciones que ha tenido.

El objetivo del presente TFG se centró en la búsqueda de patrones frecuentes teniendo en cuenta la cantidad de ejecuciones en las que se disparaba dicho patrón. Para esto se ha partido de una especificación teórica sobre un algoritmo [10], que establecía las bases para una búsqueda de patrones en grafos restringidos, pero sin soporte a bucles, y se ha implementado, analizado y mejorado, obteniendo así el algoritmo presentado en esta memoria.

En concreto, el trabajo abordado por el proyecto, que se refleja en esta memoria, se resume en los siguientes puntos:

1. Se ha implementado un algoritmo que supone la base para la búsqueda de patrones frecuentes en grafos restringidos.
2. Se han implementado una serie de métodos necesarios para el algoritmo [10], pero que no fueron especificados por los autores del mismo.
3. Se ha analizado el algoritmo y su funcionamiento, detectando acciones innecesarias y mejorando así su rendimiento.
4. Se ha redefinido el concepto de patrón elemental, consiguiendo así patrones resultantes de mayor legibilidad y significado.
5. Se ha creado un algoritmo para la detección de bucles en grafos.
6. Se ha mejorado el algoritmo de forma que soporte grafos con bucles, teniéndolos en los patrones resultantes, e identificándolos en el grafo origen.
7. Se ha elaborado una interfaz web para la interacción con el algoritmo y visualización de los resultados por parte del usuario.
8. Se ha validado el algoritmo desarrollado tanto con casos reales como el de *e-learning*, como con numerosos casos sintéticos.

Además, el método desarrollado para la identificación de bucles en los grafos originales no solo tendrá aplicación en este algoritmo de búsqueda, sino que también se llevará a otras áreas de investigación, como la minería de flujos de trabajo, donde se probará su eficacia ayudando a la detección de tareas duplicadas.

9.1. Mejoras y Ampliaciones

A pesar de que en este TFG se ha hecho un gran avance ampliando un algoritmo de búsqueda de patrones frecuentes para obtener mejores resultados, y dar soporte a grafos con bucles, el tiempo dispuesto para la elaboración de este proyecto no ha permitido realizar un análisis minucioso del algoritmo resultante, quedando así como posibles ampliaciones diferentes mejoras e investigaciones de variantes que se comentan a continuación.

9.1.1. Eficiencia

La eficiencia es uno de los aspectos más críticos cuando se trata de aplicaciones reales en las que pueden existir miles de trazas, lo que además da lugar a grafos complejos que es necesario analizar. Por este motivo, uno de los aspectos más claros que admite mejoras es el de la eficiencia:

- El análisis de la frecuencia de los patrones en este algoritmo se realiza comprobando el conjunto completo de los registros. El crecimiento de la complejidad temporal a medida que se aumentan el número de entradas en el log es lineal, por lo que, en caso de que se disponga de registros de tamaños elevados, el rendimiento podría ser muy bajo.

  Para contrarrestar esto, como posible ampliación si se quiere dar soporte a análisis de registros masivos, se podría realizar una comprobación de la frecuencia sobre subconjuntos de las trazas, disminuyendo la veracidad de los resultados, pero aumentando el rendimiento.

- Durante el desarrollo de este TFG, y como se ha explicado en secciones anteriores, se han analizado variantes para el cálculo de la frecuencia de patrones con bucles. Algunas de estas variantes se han tenido que desechar por falta de tiempo para su estudio, por lo que una posible ampliación sería retomar la investigación de dichos métodos, para intentar conseguir algoritmos más eficientes para el análisis de la frecuencia de un patrón.

- Debido al poco tiempo del que se dispone para la realización de este TFG, algunos métodos no han podido ser revisados en profundidad, por lo que una mejora importante que afectaría al rendimiento de forma positiva sería revisar métodos como el que simplifica los resultados, o el que obtiene las combinaciones de un grafo, para evitar la realización de operaciones innecesarias.

9.1.2. Filtrado de los resultados

El algoritmo devuelve los patrones frecuentes de todos los tamaños que encuentra, esto implica que hay patrones en el conjunto de los resultados que están contenidos dentro de otros patrones.

Esto es debido a que primero se descubre un patrón, y luego se expande y se descubre otro que lo contiene.

El algoritmo actual aplica un filtrado para extraer los patrones que están contenidos dentro de otros del conjunto de resultados. Una posible mejora sería añadir otras funciones de filtrado. Por ejemplo, un posible filtrado de interés sería extraer los patrones que tienen fragmentos de bucles, pero no el bucle entero, ya que hay contextos en los que la información que se le da al cliente si se aporta una fracción de un bucle es parcial.

9.1.3. Ampliación de funcionalidades en la interfaz

En cuanto a la interfaz gráfica, se ha decidido realizar un diseño simple, que permitiese comprobar el funcionamiento del algoritmo, y visualizar de forma cómoda los resultados. Pero dependiendo del ámbito de aplicación, puede ser interesante presentar funcionalidades adicionales. Por ejemplo, se podría ofrecer la posibilidad de que el usuario consulte las trazas en las que se da un determinado patrón, por si desea conocer más en profundidad el entorno por el cual éste se ha dado.

Una característica que se ha descubierto, aunque no se ha podido estudiar, es el comportamiento del número de patrones que se descubren en cada iteración del algoritmo. En algunas ejecuciones se ha observado que, a medida que avanzan las iteraciones, el número de patrones encontrados se comporta de la siguiente forma: aumenta hasta llegar a un máximo y disminuye con un comportamiento similar al del aumento. Otra posible ampliación sería estudiar este comportamiento, con la posibilidad de ofrecer al usuario una aproximación del tiempo restante, analizando los patrones encontrados en las iteraciones ya ejecutadas.

9.1.4. Cambio del objetivo

Por la estructura que caracteriza a este algoritmo, y las propiedades compatibles de la minería de grafos, se podría estudiar una modificación para que el algoritmo buscase patrones frecuentes a nivel de grafo. Es decir, que la frecuencia o peso de un patrón no se vea en las trazas, sino en las repeticiones que éste presenta sobre el grafo.

Esto podría ser interesante desde el punto de vista de reducir la complejidad de grafos *spagetti*[1], detectando estructuras repetidas, que se podrían simplificar, facilitando así la interpretación del grafo por parte de los usuarios.

Para esto sería necesario redefinir los conceptos de patrón frecuente, y nodos frecuentes, para que se ejerciese la búsqueda sobre el grafo, y no sobre los registros. Por el carácter modular del diseño, y como la base del algoritmo se podría adaptar fácilmente, este objetivo se podría abordar con una adaptación sencilla del algoritmo presentado en este TFG.

9.1.5. Integración con ProDiGen

Dado que ProDiGen genera flujos de trabajo (o grafos) a partir de *logs* de registro, se podrían integrar ambos algoritmos consiguiendo una aplicación que, partiendo de unas trazas, obtenga el

[1] Un grafo *spagetti* es un grafo con un número de nodos y arcos muy elevado.

grafo a través de ProDiGen, y acto seguido sus patrones frecuentes gracias al algoritmo desarrollado en este TFG.

9.1.6. Escalabilidad

Debido a que la importancia de este TFG residía en la creación del algoritmo, no se ha dedicado tiempo a la escalabilidad de la aplicación web. Una posible ampliación sería rediseñar la misma, en especial la comunicación y esquema de la base de datos, para dar soporte en caso de que el flujo de peticiones aumente. Debido al patrón *Abstract Facotry* y al diseño con MVC que se ha implementado, las modificaciones en este campo requerirían esfuerzos mínimos.

Apéndice A

Manual técnico

Este manual está diseñado para que cualquier tipo de usuario pueda configurar y poner en funcionamiento el algoritmo de minería de grafos y la aplicación web diseñadas en este TFG. Para realizar la instalación y configuración del sistema se tomará como base el Sistema Operativo Linux Mint 17.2, pero tanto las herramientas utilizadas, como los pasos a seguir son similares en cualquier otro S.O.

A.1. Instrucciones de instalación

El software necesario consta de una aplicación web empaquetada en un archivo con extensión **.war* que requiere de un contenedor web para ser desplegado, pudiendo acceder así al mismo. La instalación que se explicará a continuación tiene como finalidad poder probar la aplicación en local, este manual no muestra el caso de despligeue sobre un servidor para dar acceso a través de la Web, ya que esto requiere un nivel más avanzado por parte del usuario.

Para poder ejecutar la aplicación es necesario tener instalados correctamente en el sistema los siguientes componentes software:

- **MySQL-server 5.6** o superior: puede ser descargado desde: `https://dev.mysql.com/downloads/mysql/`.
- **Apache Tomcat 7.0**: puede ser descargado desde: `http://tomcat.apache.org/download-70.cgi`.
- **Google Chrome 43.0** o superior: puede ser descargado desde: `https://www.google.es/chrome/browser/desktop/index.html`.

Partiendo de una versión recién instalada de Linux Mint 17.2, es decir, sin paquetes adicionales, se deben instalar los componentes software listados en el párrafo anterior. En este manual no se explicarán las instalaciones de dicho software debido a la gran cantidad de documentación que se encuentra en la web, y a que no se tiene la certeza de que los métodos de instalación no variarán.

A.1.1. Instalación de la Base de Datos

Una vez se tiene instalado en el equipo el servidor de MySQL se procederá a cargar el modelo de la base de datos. Para esto se puede utilizar una terminal o un programa que aporte una interfaz gráfica para realizar la carga de un script SQL, como por ejemplo MySQL Workbench[1]. A continuación se mostrarán los pasos a seguir para realizar la creación de la base de datos desde una terminal:

1. Abrir una terminal.
2. Situarse en el directorio donde se aloja el script SQL[2].
3. Iniciar sesión como root ('*mysql -u root -p*').
4. Cargar el archivo ejecutando '*source GraphMining.sql*'.

A.1.2. Despliegue de la aplicación

Una vez se ha creado la estructura de la base de datos se debe proceder a desplegar la aplicación para permitir así el acceso a través del navegador web. Para esto se debe tener instalado en el equipo un contenedor web. En este manual se utilizará Apache Tomcat 7.0.

Para esto se debe situar el archivo comprimido con la aplicación en el directorio '*TOMCAT_DIR/webapps/*' siendo '*TOMCAT_DIR*' la ruta al directorio donde se encuentra instalado Tomcat. Acto seguido se inicial el contenedor web ejecutando el scripte bash '*TOMCAT_DIR/bin/startup.sh*'.

Una vez realizados los pasos especificados previamente, la aplicación se encuentra desplegada y puede ser accedida dirigiéndose a la ruta `localhost:8080/GraphMiner/` en el navegador.

[1] `http://dev.mysql.com/downloads/workbench/`

[2] El script para la creación de la base de datos se incluye en el CD de instalación bajo el directorio *BBDD* (GraphMining.sql)

Apéndice B

Manual de usuario

Aunque la aplicación se ha realizado primando la usabilidad y comodidad de la misma, se realizará una explicación del su uso, con el fin de aclarar posibles dudas aportando imágenes para apoyar las descripciones.

B.1. Ejecutar algoritmo

B.1.1. Inicio de la aplicación

Hay varias formas de cargar un modelo o grafo en la aplicación para lanzar el algoritmo. Al acceder al inicio se muestra la pantalla principal (Figura B.1), en ella se observa un formulario con los siguientes campos:

- **Workflow alias**: alias que será asignado al flujo de trabajo subido para futuras consultas.
- **Workflow**: fichero *.hn* con la descripción del grafo.
- **Log alias**: alias que será asignado al registro subido para futuras consultas.
- **Log**: fichero *.xes* con las descripción de los registros.
- **Frequency threshold**: umbral de frecuencia (valor entre 0.0 y 100.0).

En esta pantalla se deben cubrir todos los campos estableciendo así los alias utilizados para identificar al grafo y sus registros, los ficheros que contienen dichos elementos y el umbral de frecuencia con el que será ejecutado el algoritmo. Una vez completos se debe pulsar el botón '*Load Model*' para iniciar el algoritmo.

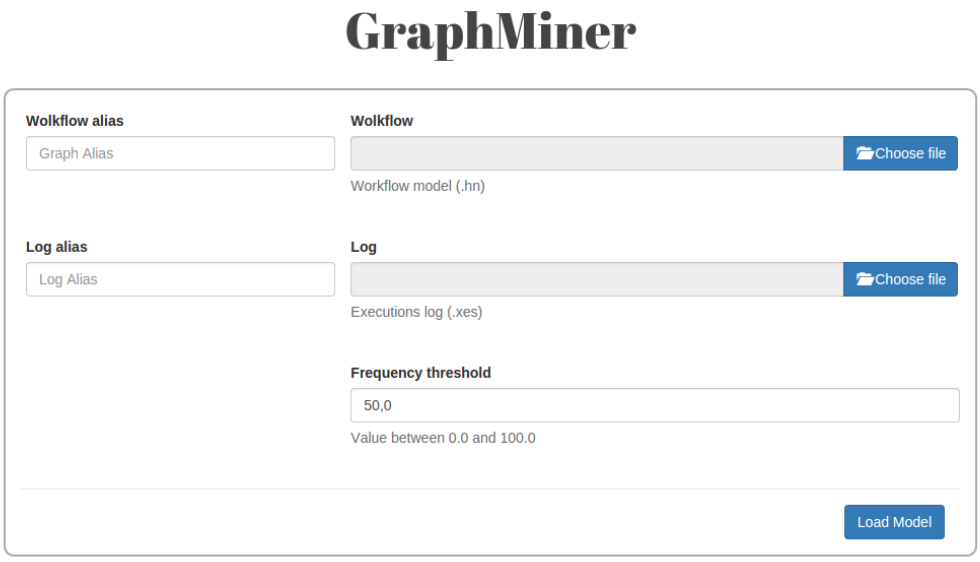

Figura B.1: Pantalla principal de la aplicación.

B.1.2. Desde resultados

Además de iniciar el algoritmo desde la pantalla principal, la aplicación permite el cargado de otro modelo desde la pantalla de resultados. Para esto se debe pulsar la opción '*Load Model*' del menú de opciones (Figura B.2) situado en la esquina superior izquierda de la pantalla de los resultados. Una vez pulsado este botón se muestra un formulario similar al del inicio de la aplicación. Para cargar el modelo se debe pulsar el botón '*Run algorithm*' y si, por el contrario, se quiere volver a los resultados, se debe pulsar el botón '*Cancel*'.

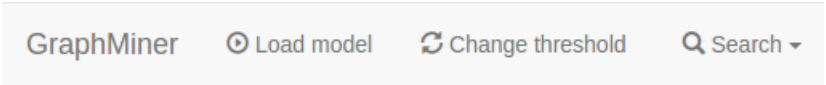

Figura B.2: Menú de opciones de la aplicación.

B.1.3. Accediendo a la URL

La última forma posible de lanzar el algoritmo es accediendo a través de la URL de una ejecución ya lanzada. De esta forma, accediendo directamente a la dirección de la Figura B.3 se obtienen los resultados de la base de datos, informando del estado como en la Figura B.4.

Figura B.3: Aceso directo a una ejecución previa.

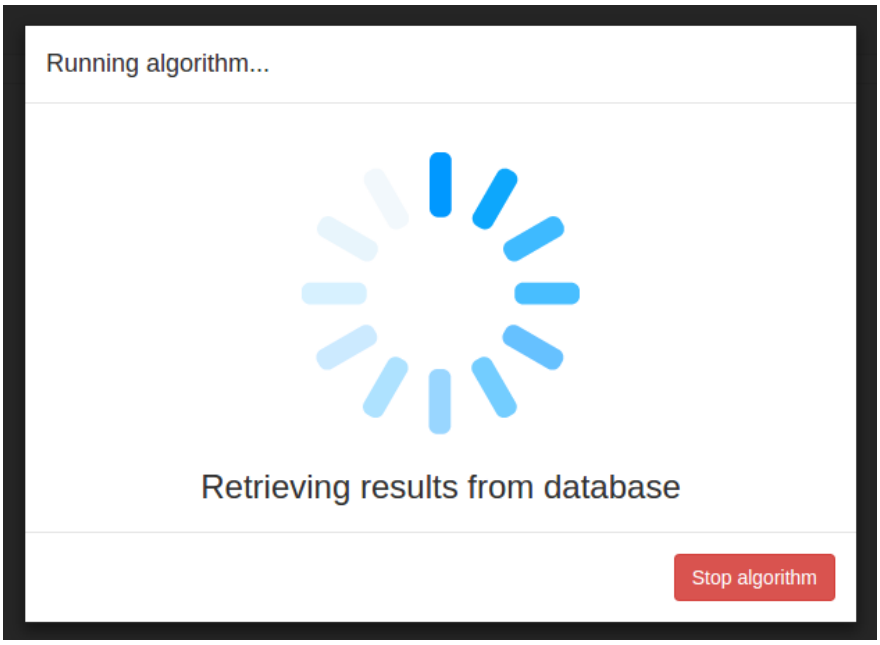

Figura B.4: Recuperación de los resultados de la Base de Datos.

B.1.4. Cancelar ejecución

Si se desea cancelar la ejecución del algoritmo, porque el tiempo de espera es demasiado, o porque se quiere cambiar algún parámetro se debe pulsar el botón '*Stop algorithm*' que se muestra en la Figura B.5. Esta acción disparará una pregunta de confirmación (Figura B.6), que se debe aceptar en caso de querer cancelar el algoritmo.

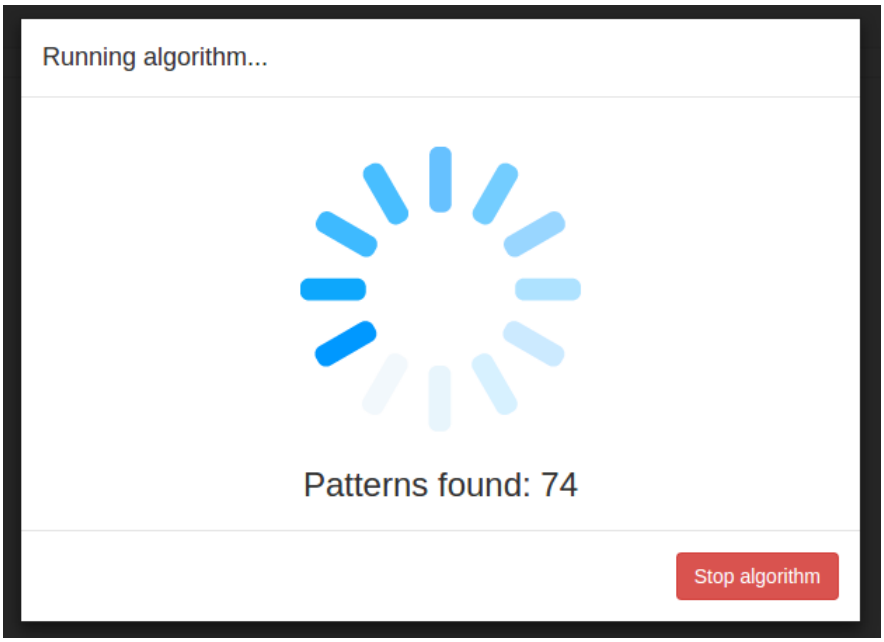

Figura B.5: Algoritmo en ejecución.

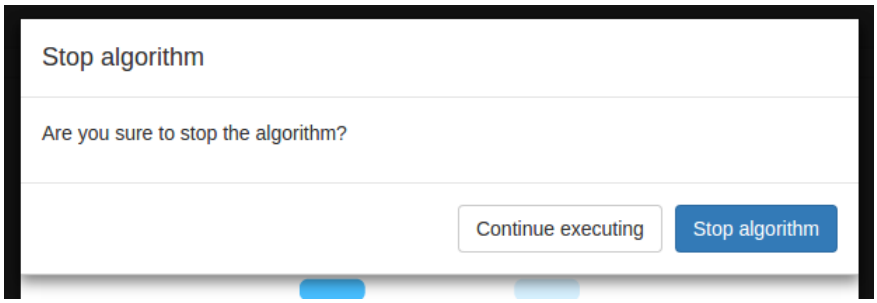

Figura B.6: Confirmación de cancelación del algoritmo.

B.1.5. Visualizar resultados

Una vez se ha ejecutado el algoritmo se carga la pantalla en la que se visualizan los resultados. Para una mejor visualización, la aplicación permite efectuar zoom y desplazar el grafo con el ratón, además de los botones que se muestran en la esquina superior derecha de la Figura B.7 realizan las funcionalidades de: *(i)* ampliar zoom, *(i)* reducir zoom, *(iii)* centrar el patrón seleccionado y *(iv)* centrar el grafo. En la Figura B.7 también se puede apreciar la funcionalidad que permite seleccionar y restaltar un patrón de los resultantes.

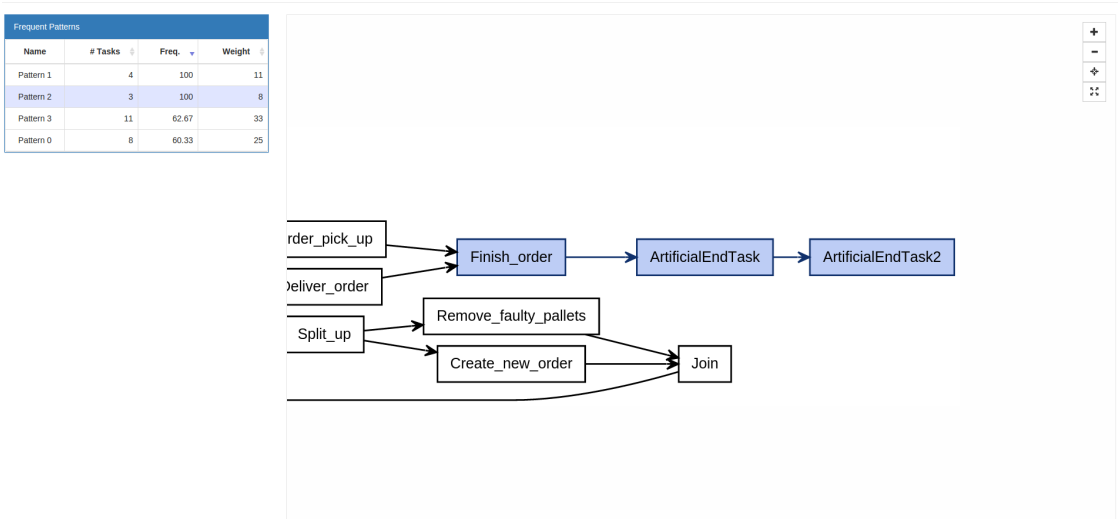

Figura B.7: Pantalla ampliada con los resultados.

En la Figura B.8 se muestra la tabla con los patrones frecuentes que forman el resultado de la ejecución, como se puede apreciar esta se puede ordenar de forma ascendente y descendente por el número de tareas, peso y frecuencia de aparición haciendo clic en cada título.

Frequent Patterns

Name	# Tasks	Freq.	Weight
Pattern 1	4	100	11
Pattern 2	3	100	8
Pattern 3	11	62.67	33
Pattern 0	8	60.33	25

Figura B.8: Tabla con los patrones frecuentes.

B.1.6. Filtrar resultados

Si se desean consultar patrones en función de las tareas que contienen se puede realizar con la opción que se muestra en la Figura B.9. Esta permite mostrar todos los patrones frecuentes que han sido encontrados, o realizar un filtrado dependiendo de las tareas que contengan (Figura B.10). La opción '*All in the group*' muestra los patrones frecuentes cuyas tareas se encuentran en el grupo seleccionado en el *combobox* anterior, mientras que la opción '*Any in the group*' filtra los resultados mostrando los patrones que contengan alguna tarea del conjunto seleccionado

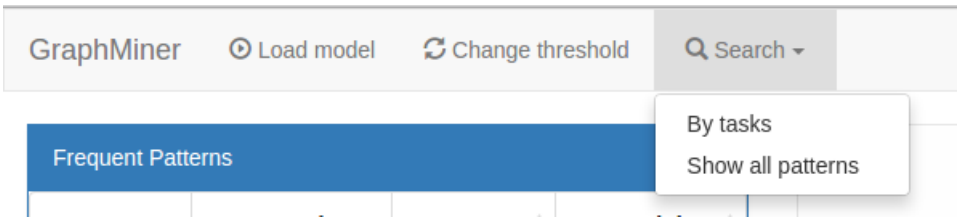

Figura B.9: Opción del menú para filtrar tareas.

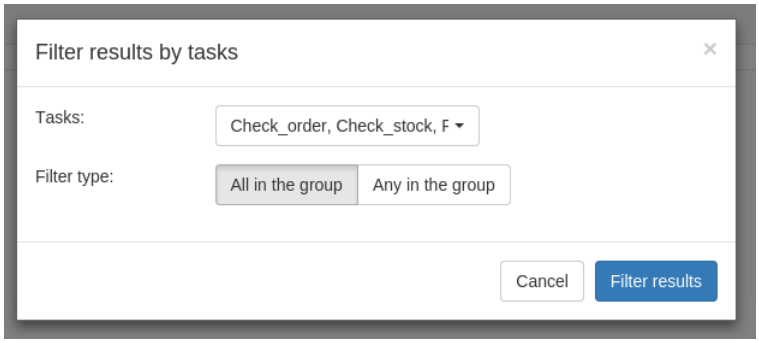

Figura B.10: Opciones de filtrado de tareas.

B.1.7. Cambiar umbral

La opción de cambiar umbral le permite al usuario lanzar una ejecución con el mismo modelo y mismos registros que la actual, pero cambiando únicamente el umbral, esto evita la acción de volver a cargar los ficheros necesarios, agilizando el proceso. Para esto debe seleccionar la opción '*Change threshold*' del menú; la aplicación mostrará una pantalla como la de la Figura B.11 en la que se encuentra un campo para introducir el nuevo umbral.

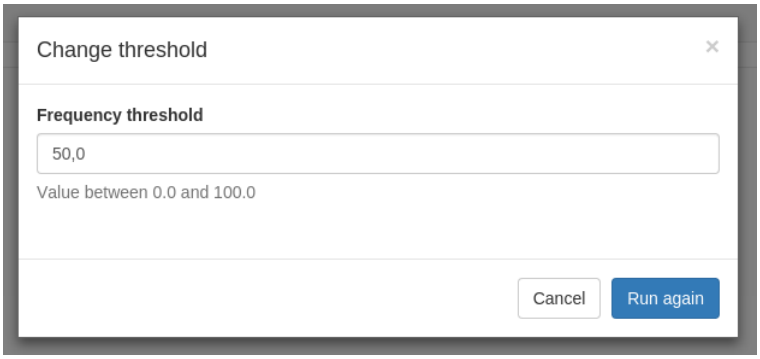

Figura B.11: Cambiar umbral de ejecución.

Apéndice C

Diagramas de clases completos

En este apéndice se incluyen los diagramas de clases del sistema que han sido simplificados en sus respectivos apartados debido a la imposibilidad de la visualización en formato impreso de los mismos. Para ofrecer los diagramas completos en caso de disponer de un soporte digital en el que ampliar las imágenes, se muestran a continuación los diagramas completos.

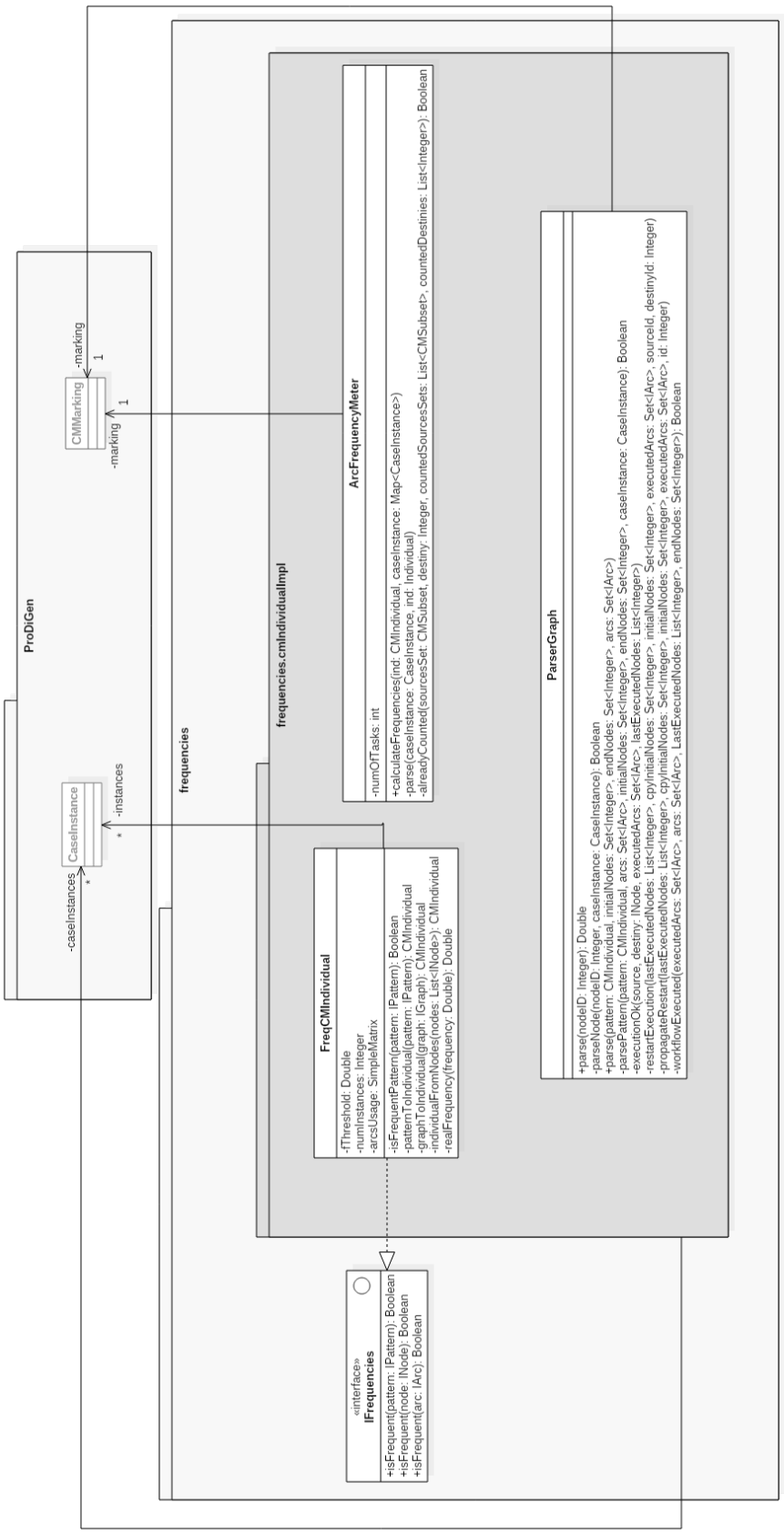

Figura C.1: Diagrama de clases del paquete de la medición de las frecuencias completo.

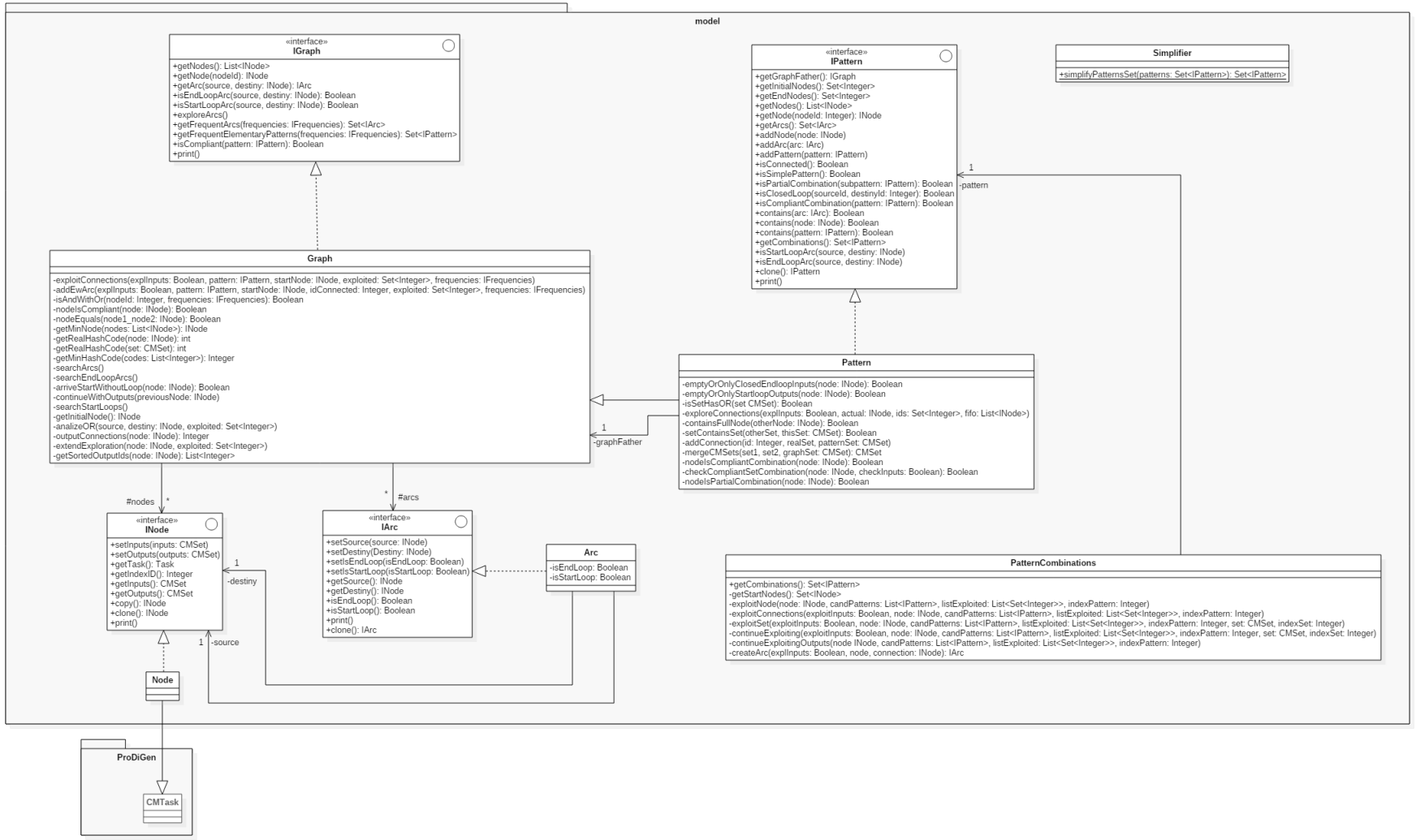

Figura C.2: Diagrama de clases del paquete del modelo completo.

Bibliografía

[1] X. Yan and J. Han. *CloseGraph: Mining closed frequent graph patterns.* In Proc. ACM Int. Conf. on Knowledge Discovery and Data Mining (KDD'03), pages 286–295, 2003.

[2] Grady Booch, James Rumbaugh, and Ivar Jacobson. *El lenguaje unificado de modelado. Manual de referencia.* Addison Wesley, 1999.

[3] *Genetic Process Mining.* Ana Karla Alves de Medeiros. Eindhoven, Technische Universiteit Eindhoven, 2006. Thesis.

[4] K. Yoshida, H. Motoda, and N. Indurkhya. *Graph- based induction as a unified learning framework.* Journal of Applied Intel., 4:297–328, 1994.

[5] X. Yan and J. Han. *gSpan: Graph-based substructure pattern pining.* In Proc. IEEE Int. Conf. on Data Mining (ICDM'02), 2001. An extended version appeared as UIUC-CS Tech. Report: R-2002-2296.

[6] *IEEE recommended practice for software requirements specifications.* Technical report, 1998.

[7] Ian Sommerville. *Ingeniería del Software* (7th ed.). Pearson Educación, 2005.

[8] David Flanagan. *JavaScript: The definitive guide.* (6th ed.). O'Reilly, 2011. 3.0

[9] *Lectures on Petri Nets I: Basic Models*, Lecture Notes in Computer Science, vol. 1491, W. Reisig and G. Rozenberg, eds., Berlin: Springer-Verlag, 1998

[10] Gianluigi Greco, Antonella Guzzo, Giuseppe Manco, Luigi Pontieri, and Domenico Saccà: *Mining Constrained Graphs: The Case of Workflow Systems*

[11] *Object Oriented Patterns* http://www.oodesign.com/, última visita 30/06/2015.

[12] Borja Vázquez-Barreiros, Manuel Mucientes, Manuel Lama: *ProDiGen: Mining complete, precise and minimal structure process models with a genetic algorithm.* Inf. Sci. 294: 315-333 (2015).

[13] *The Standish Group Report: CHAOS* http://www.projectsmart.co.uk/docs/chaos-report.pdf, última visita 30/06/2015.

[14] D. J. Cook and L. B. Holder. *Substructure Discovery Using Minimum Description Length and Background Knowledge.* Journal of Artificial Intelligence Research, 1(1):231–255, 1994.

[15] Brooke, J. (1996) *SUS: a "quick and dirtyüsability scale.* In P. W. Jordan, B. Thomas, B. A. Weerdmeester & A. L. McClelland (eds.) Usability Evaluation in Industry. London

[16] Borja Vázquez Barreiros. *Un algoritmo genético para la minería de flujos de trabajo en educación*. Trabajo de Fin de Grado. Escuela Técnica Superior de Ingeniería. Universidad de Santiago de Compostela, 2011.

[17] *Vitae: Estudio salarial sector TIC Galicia 2014-2015* www.vitaedigital.com/download/NDQx/o_-543_f_585.pdf, última visita 30/06/2015.

[18] WfMC. *Workflow Management Coalition Terminology Glossary*. Technical Report WFMC-TC-1011, Workflow Management Coalition, Feb. 1999. Document Status Issue 3.0