CONVERSOR DE REGLAS DE PRIORIDAD A ÁRBOLES DE EXPRESIÓN

Trabajo de Fin de Grado en Ingeniería Informática del Software

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

Autor: Jorge Iturrioz del Vigo

Tutores: Francisco Javier Gil Gala y María Rita Sierra Sánchez

Índice de contenidos

Índice de figuras

Resumen

Las reglas de prioridad son un tipo de algoritmo frecuentemente utilizado en la resolución de problemas de optimización, especialmente, en aquellos problemas donde el tiempo de resolución es limitado. Normalmente, una regla de prioridad es un algoritmo voraz que se compone de dos componentes clave: un constructor de soluciones y una función de prioridad. En este paradigma, la función de prioridad es usada para guiar al algoritmo voraz de forma que se asigna una prioridad a cada una de las posibles opciones a elegir. El algoritmo voraz escogería en cada paso la opción más prioritaria hasta terminar de construir la solución.

La realización de este proyecto viene motivada por la línea de investigación dedicada al aprendizaje de reglas de prioridad para scheduling en tiempo real [1] [2] del grupo en *Investigación en Optimización y Scheduling Inteligentes* (**iScOp**) de la Universidad de Oviedo. El objetivo principal de este proyecto es diseñar e implementar una herramienta que permita la representación gráfica de reglas de prioridad y facilite la creación de nuevas reglas.

La herramienta de conversión, resultado del proyecto, será utilizada por usuarios especializados, concretamente investigadores del grupo iScOp. La herramienta les permitirá no sólo representar reglas de prioridad previamente calculadas de manera automática, con alguna estrategia inteligente, sino también modificar reglas existentes o crear nuevas reglas. Además, permitirá la ejecución de las reglas en un determinado contexto suministrado por el usuario.

Palabras clave

Antlr.

CSV.

Inorden.

Java.

Montículo binario.

PDF.

Regla de prioridad.

Resultado.

Visualización de reglas.

Capítulo 1. Memoria del proyecto

1.1 Hojas de identificación

Título del proyecto: Conversor de reglas de prioridad en Árboles de Expresión.

Código identificador: ConversorRegPriArbolExp / CRPAE

Información relativa al cliente:

Los tutores del proyecto en cuestión son María Rita Sierra Sánchez ([redacted]@uniovi.es) y Francisco Javier Gil Gala ([redacted]@uniovi.es), pertenecientes ambos a la Escuela de Ingeniería Informática de la Universidad de Oviedo, situada en Campus de Los Catalanes, Calle Valdés Salas, Oviedo, Asturias, con código postal 33007 y teléfono (+34) 985 10 27 96.

Información relativa al suministrador del proyecto:

El encargado de la realización del proyecto es el alumno Jorge Iturrioz del Vigo, también perteneciente a la Escuela de Ingeniería Informática de la Universidad de Oviedo, con correo de contacto [redacted]@uniovi.es

Resumen:

El trabajo con diferentes tipos de reglas de prioridad y la manipulación de estas, así como de los resultados obtenidos, genera gran cantidad de información objeto de estudio. Es por esto por lo que, se ha solicitado una herramienta que permita manejar y visualizar estas reglas, así como calcular el valor de estas, de manera rápida y accesible a los usuarios.

Duración estimada: 39 días.

Coste estimado: 7.952,12 €.

1.2 Introducción

Los investigadores del grupo de *Investigación en Optimización y Scheduling Inteligentes* (**iScOp**) de la Universidad de Oviedo, dedicados al cálculo automático de reglas de prioridad para distintos problemas de optimización y satisfacción de restricciones, se enfrentan a menudo a la ardua tarea de representar de forma gráfica las reglas calculadas por las diferentes estrategias inteligentes que emplean en sus investigaciones.

La herramienta desarrollada deberá cubrir sus necesidades en este sentido y satisfacer las funcionalidades establecidas para este trabajo. Estas necesidades consisten en la

representación gráfica de reglas nuevas, de reglas previamente calculadas, así como la modificación y almacenamiento de todas estas reglas y la ejecución de estas en el contexto de un problema proporcionado por el usuario.

1.3 Motivación

La elaboración del proyecto viene dada por el grupo de investigación de la Universidad de Oviedo *Investigación en Optimización y Scheduling Inteligentes* (**iScOp**). Este grupo aborda habitualmente la resolución de problemas de optimización combinatoria y satisfacción de restricciones mediante técnicas inteligentes, como la búsqueda en espacios de estados, algoritmos voraces, metaheurísticas, programación con restricciones o hiperheurísticos.

Los hiperheurísicos permiten resolver problemas mediante el cálculo de buenos heurísticos o reglas de prioridad [3]. Entre las estrategias más utilizadas para este fin se encuentra la Programación Genética (GP) [1] [2] [4] [5] [6] un tipo de algoritmo evolutivo donde los cromosomas se representan mediante árboles, una representación natural de las reglas de prioridad, al ser estas habitualmente definidas como expresiones aritméticas.

El grupo iScOp, en una de sus líneas de investigación, aborda el cálculo mediante GP de reglas de prioridad para distintos problemas, entre ellos, el problema se secuenciamiento de una máquina con capacidad variable o el problema del viajante de comercio [1] [2] [6] [7]. Dada la cantidad de reglas que calculan estas estrategias, una herramienta como la planteada en este trabajo facilitaría mucho a los investigadores de iScOp la manipulación y representación gráfica de las reglas calculadas para cualquier tipo de problema.

1.4 Objeto

Este proyecto tiene como objetivo desarrollar una herramienta software capaz de procesar diferentes tipos de reglas de prioridad introducidas por el usuario, obteniéndose como resultado la forma gráfica de estas, así como un resultado numérico que evalúa la calidad de cada una. Por ello, los objetivos a lograr con el proyecto son los siguientes:

- Desarrollo de aplicación que permita, en base a diferentes ficheros introducidos por el usuario (concretamente, el contexto de un problema), la gestión de las reglas introducidas por este mismo.
- Visualización de las reglas, así como del resultado de estas, que permita clasificarlas según su prioridad.
- Desarrollo de una herramienta que disponga de un historial que permita almacenar y ver todas las ejecuciones de las reglas realizadas hasta el momento.

1.5 Normas y referencias

1.5.1 Disposiciones y normas aplicadas

- *Norma UNE 157801* [8]
  - Estándar relativo a la estructura que ha de seguir un documento relativo a la elaboración de sistemas informáticos, en el cual se basa este documento.
- *Java Look and Feel Guidelines* [9]
  - Estándares de diseño de interfaces gráficas en Java, que incluyen, entre otras cosas, disposición de elementos, prácticas de usabilidad recomendadas o uso de iconos.
- *IEE 830-1998* [10]
  - *Estándar que trata sobre prácticas recomendadas a la hora de elaborar requisitos de software, utilizado en el formato de especificaciones de requisitos del proyecto.*

1.5.2 Bibliografía

[1] F. J. Gil-Gala, M. Sierra, C. Mencía y R. Varela, «Genetic programming with local search to evolve priority rules for shceduling jobs on a machine with time-varying capacity,» *Swarm and Evolutionary Computation,* 2021.

[2] F. J. Gil-Gala, C. Mencía, M. Sierra y R. Varela, «Evolving priority rules for on-line scheduling of jobs on a single machine with variable capacity overtime,» *Applied Soft Computing 85: 105782,* 2019.

[3] E. Burke, M. Hyde, G. Kendall, E. Ozcan y J. Woodward, «A Classification of Hyper-Heuristic Approaches: Revisited. In: Gendreau M., Potvin JY.,» *(eds) Handbook of Metaheuristics. International Series in Operations Research & Management Science 272,* pp. 453-477, 2019.

[4] M. Đurasević, D. Jakobović y K. Knežević, «Adaptative scheduling on unrelated machines with genetic programming.,» *Applied Soft Computing 48,* pp. 419-430, 2016.

[5] F. Zhang, Y. Mei, S. Nguyen, K. Tan y M. Zhang, «Multitask Genetic Programming-Based Generative Hyperheuristics: A case Study in Dynamic Scheduling,» *IEEE Transactions on Cybernetics, 1(3),* pp. 1-14, 2021.

[6] F. Gil-Gala, M. Đurasević, M. Sierra y R. Varela, «Building Heuristics and Ensembles for the Travel Salesman Problem. In: Ferrández Vicente, J.M., Álvarez-Sánchez, J.R., de la Paz López, F., Adeli, H. (eds) Bio-inspired Systems and Applications: from Robotics to Ambient Intelligence,» *IWINAC 2022. Lecture Notes in Computer Science. Springer, Cham. https://doi.org/10.1007/978-3-031-06527-9_13,* vol. 13259.

[7] F. Gil-Gala, R. Varela y M. Sierra, «Cálculo de reglas de prioridad para problemas de scheduling con hiperheurísticos,» Tesis doctoral, 2021.

[8] UNE, «UNE 157801:2007,» 2007. [En línea]. Available: https://www.une.org/encuentra-tu-norma/busca-tu-norma/norma/?c=N0039577. [Último acceso: 15 Junio 2022].

[9] Sun Mycrosystems Inc., Java Look and Feel Design Guidelines, Addison-Wesley, 2001.

[10] IEEE, «IEEE 830-1998,» 1998. [En línea]. Available: https://standards.ieee.org/standard/830-1998.html. [Último acceso: 16 Junio 2022].

[11] Project Management Institute, A guide to the Project Management Body of Knowledge (PMBOK guide), Newton Square, PA: Project Management Institute, 2017.

[12] E. Hart y K. Sim, «A hyper-heuristic ensemble method for static job-shop scheduling,» *Evolutionary Computation 24(4),* pp. 609-635, 2016.

[13] Wikipedia, «P versus NP Problem,» [En línea]. Available: https://en.wikipedia.org/wiki/P_versus_NP_problem#/media/File:P_np_np-complete_np-hard.svg. [Último acceso: 2 Julio 2022].

[14] A. Hernández-Arauzo, J. Puente, R. Varela y J. Sedano, «Electric vehicle charging under power and balance constraints as dynamic scheduling,» *Computers & Industrial Engineering, 85,* pp. 306-315, 2015.

[15] P. McMenemy, N. Veerapen, J. Adair y G. Ochoa, «Rigorous performance analysis of state-of-the-art tsp heuristic solvers. In: Liefooghe, A., Paquete, L. (eds.) Evolutionary Computation in Combinatorial Optimization. pp. 99{114. Springer International Publishing, Cham,» 2019.

[16] Refactoring Guru, «Visitor Behavioral Pattern,» [En línea]. Available: https://refactoring.guru/design-patterns/visitor. [Último acceso: 2 Julio 2022].

[17] E. Dijkstra, «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Shunting_yard_algorithm. [Último acceso: 20 Junio 2022].

1.5.3 Métodos, herramientas y modelos

En el ámbito de la ingeniería del software, existen diferentes metodologías de trabajo, entre las cuales se encuentran las metodologías iterativas e incrementales, pensadas principalmente para ser usadas durante toda la duración del proyecto, abarcando desde la documentación hasta el desarrollo de este.

Para este proyecto se ha decidido seguir una metodología de trabajo similar, de forma que se siga un proceso de desarrollo incremental e iterativo. Concretamente, para este proyecto se ha buscado priorizar la funcionalidad a desarrollar de forma que se obtenga un producto con valor para el usuario, y sobre esto ir añadiendo de forma iterativa funcionalidad.

Para realizar esta labor, se han utilizado las siguientes herramientas:

- **Microsoft Word:** Para redactar el presente documento.
- **Microsoft Excel:** Para el acceso a ficheros con formato ".csv"
- **Microsoft Project:** Para la planificación temporal del proyecto y seguimiento de este.
- **OneDrive:** Para almacenar en la nube elementos tales como documentación y demás ficheros útiles.
- **Microsoft Teams / Outlook:** Para establecer reuniones en las cuales hablar sobre el estado del proyecto, avances o cualquier comentario relativo entre el desarrollador y el cliente.
- **Eclipse Java:** Entorno de desarrollo (IDE) utilizado en la codificación del proyecto.
- **GitHub:** Herramienta de control de versiones donde se ha almacenado el repositorio correspondiente al proyecto.
- **EclEmma:** Plugin utilizado junto a Eclipse para obtener la cobertura de código del programa, en función de las pruebas unitarias realizadas.
- **Antlr:** Librería utilizada para parsear las reglas introducidas por el usuario, generando las estructuras necesarias para el resto de la aplicación.

1.5.4 Control de calidad

Como medida empleada para asegurar la calidad del proyecto, se ha procurado mantener una comunicación con los tutores relativa a cualquier duda/progreso realizado en el proyecto. Se tiene en cuenta que las aportaciones al desarrollo y a la propia documentación sean relevantes y de calidad, informando de los avances realizados y del estado actual, para obtener retroalimentación y ver, en base a esta, el estado general del proyecto.

1.6 Requisitos iniciales

El producto desarrollado en el proyecto se ha de ajustar a las necesidades de los clientes, enumeradas a continuación:

1. Manipulación de las reglas de prioridad.
2. Conversión a montículo binario y viceversa.
3. Representación gráfica en forma de montículo binario.
4. Ejecución de las reglas de prioridad.

Para ello, se tiene que permitir las operaciones enumeradas a continuación:

- Representación textual de reglas de prioridad.
- Representación gráfica de las reglas de prioridad.
- Carga de ficheros que contengan las reglas de prioridad.
- Generación de ficheros con las reglas de prioridad.
- Generación de ficheros que contengan la representación gráfica de las reglas de prioridad.

De forma paralela a la herramienta, para que esta sea de utilidad, se requiere que el usuario suministre un archivo ejecutable que especifique como se van a ejecutar las reglas, así como un fichero de configuración que especifique el tipo de reglas a emplear. Estos ficheros han de ser compatibles con la funcionalidad de la herramienta elaborada.

La funcionalidad completa del programa se puede consultar en el apartado, Capítulo 3. Análisis

1.7 Solución propuesta

La herramienta proporcionada ha de permitir tanto cargar y guardar reglas, como representarlas y modificarlas, así como ejecutarlas. Para ello, las reglas se podrán importar desde un fichero ".csv" o bien podrán ser escritas por el usuario en los campos editables de la herramienta. También se hará uso de selectores de ficheros que permitan indicar la ubicación en dónde almacenar los resultados obtenidos, tanto los relacionados con la representación gráfica, como los resultados de las reglas.

También, se proporcionará la opción de administrar un fichero de configuración válido para que dictamine como han de ejecutarse las reglas escritas/administradas por este mismo.

Para la representación de los elementos, se opta por representarlos mediante un gráfico elaborado en función de la posición que ocupan los elementos de una regla, de forma su representación gráfica coincida con su representación en forma de montículo binario, es decir, que se pueda convertir de una a otra de forma inequívoca.

Para ejecutar una regla, el usuario debe tener una regla y seleccionar un archivo de instancia, luego ha de suministrar un archivo ejecutable con extensión ".jar", para lo cual se hará uso de la funcionalidad que dispone el lenguaje de programación Java, que permite simular una consola de comandos (como las vistas en MS-DOS) para ejecutar el archivo, con los parámetros necesarios, y obtener el resultado de aplicar la regla a la instancia seleccionada previamente.

1.8 Alcance del proyecto

Siendo el tiempo el factor más influyente en el desarrollo del proyecto, se ha decidido acotar en la medida de lo posible algunas ideas iniciales que se tenían sobre la funcionalidad a incluir de la herramienta, así como la forma de implementarse. Teniendo esto en cuenta, el alcance del proyecto puede resumirse en los siguientes puntos:

- Herramienta gráfica que permita la manipulación de las reglas, así como su representación en forma gráfica y su posterior procesado.
- Representación de reglas de formato "inorden" a montículo y viceversa.
- Carga de reglas desde fichero, integrado en la herramienta gráfica.

1.9 Definiciones y abreviaturas

- **Montículo binario:** Estructura de datos donde la información se encuentra ordenada, de forma que un elemento en una posición "x" va a tener sus hijos izquierdo y derecho respectivamente, en las posiciones $[2x + 1,\ 2x + 2]$ respectivamente. En caso de que un elemento en una posición "x" sea una función unaria, el hijo en la posición $2x + 2$ será siempre "null".
- **Regla:** Conjunto de símbolos aritméticos que puedan ser representados y ejecutados por el archivo con extensión ".jar" suministrado por el usuario. Un ejemplo de una regla podría ser la cadena: $((-p) + d)$, donde los elementos han de ser definidos previamente en un fichero de configuración, también suministrado por el usuario.
- **Java:** Lenguaje de programación que utiliza una máquina virtual para funcionar en cualquier equipo, independientemente de su sistema operativo.
- **Inorden:** Formato empleado comúnmente para representar operaciones matemáticas, donde, primero se tiene en cuenta el hijo derecho de un elemento padre, seguido de este y por último el hijo derecho. Un ejemplo de este formato puede ser cualquier expresión matemática sencilla como $((-p) + d)$, donde el padre de ambos elementos es el operador "+", seguido de su hijo izquierdo "-p" y su hijo derecho "d". A su vez, este hijo izquierdo tendría como padre el operador "-" y como hijo izquierdo "p", siendo inexistente en este caso un hijo derecho.

1.10 Análisis de riesgos

1.10.1 Identificación y categorización de los riesgos

A continuación, se desglosan los riesgos que se han encontrado, así como su categoría y una breve descripción de estos:

ID	Nombre del Riesgo	Breve Descripción	Categoría
1	Limitaciones de iText a la hora de guardar imágenes en cierta resolución	Las limitaciones de memoria en Java no permiten manejar el número de ficheros deseado o con la longitud requerida.	Software
2	Librerías iText quedan obsoletas	Las librerías utilizadas para el almacenamiento de PDF dejan de ser mantenidas o actualizadas.	Software
3	Tiempo insuficiente para desarrollar el proyecto	Los objetivos o el alcance definidos son demasiado para el tiempo disponible.	Planificación
4	Fallo en vías de comunicación con tutores del proyecto	Las vías de comunicación utilizadas dejan de estar disponibles (Outlook, Microsoft Teams, etc.)	Software
5	Pérdida de progreso	Se pierde parte del progreso realizado en el proyecto, bien por corrupción de datos, pérdida de la versión más reciente, etc.	Personal
6	Definición errónea de los requisitos del proyecto	Los requisitos definidos para el proyecto no se ajustan a las necesidades o el enfoque del proyecto de los tutores.	Software

Figura 1. Tabla de riesgos encontrados

1.10.2 Análisis del impacto y probabilidad de los riesgos

En este análisis, se define una escala de impacto que podrían suponer los riesgos en caso de ocurrir para el proyecto. Con estos valores, obtenemos la matriz de probabilidad-impacto mostrada a continuación, basada en la matriz de riesgos planteada en [11]:

Probabilidad	**Muy Alta**	***0,90***	0,05	0,14	0,27	0,50	0,81
	Alta	***0,70***	0,04	0,11	0,21	0,39	0,63
	Media	***0,50***	0,03	0,08	0,15	0,28	0,45
	Baja	***0,30***	0,02	0,05	0,09	0,17	0,27
	Muy Baja	***0,10***	0,01	0,02	0,03	0,06	0,09
			0,05	***0,15***	***0,30***	***0,55***	***0,90***
			Inapreciable	**Bajo**	**Medio**	**Alto**	**Crítico**
			0,05	***0,15***	***0,30***	***0,55***	***0,90***
			Impacto				

Figura 2. Matriz de impacto de riesgos

La matriz mostrada anteriormente no es la original de la cita bibliográfica, debido a que en la original se muestra una versión homóloga representando una matriz de oportunidades, no aplicable a este proyecto debido a que no hay oportunidades visibles.

Siguiendo esta matriz y teniendo en cuenta los riesgos encontrados, se muestra a continuación la clasificación y valoración de impacto para cada riesgo mostrado anteriormente:

ID	Nombre del Riesgo	Probabilidad	Impacto				Impacto
			Presup.	Planific.	Alcance	Calidad	
1	Limitaciones de iText a la hora de guardar imágenes en cierta resolución	Baja	Medio	Alto	Alto	Medio	0,17
2	Librerías iText quedan obsoletas	Baja	Medio	Bajo	Inapreciable	Medio	0,09
3	Tiempo insuficiente para desarrollar el proyecto	Baja	Alto	Medio	Alto	Bajo	0,17
4	Fallo en vías de comunicación con tutores del proyecto	Muy Baja	Bajo	Medio	Bajo	Bajo	0,03
5	Pérdida de progreso	Muy Baja	Medio	Crítico	Crítico	Medio	0,09
6	Mal ajuste de los requisitos del proyecto	Media	Medio	Crítico	Alto	Medio	0,45

Figura 3. Impacto de riesgos

1.10.3 Estrategia y respuesta hacia los riesgos

De forma análoga a la representación de los riesgos, tras analizar el impacto de los mismos se aportan las siguientes medidas a tomar en caso de que sucedan.

ID	Nombre del Riesgo	Respuesta al Riesgo	Estrategia
1	Limitaciones de iText a la hora de guardar imágenes en cierta resolución	Se utiliza otra tecnología para la generación de archivos en formato PDF, como por ejemplo “PdfBox”	Asumir el riesgo
2	Librerías iText quedan obsoletas	Emplear la última versión estable que no tenga vulnerabilidades/errores críticos conocidos. En caso de no ser así, usar otra tecnología	Asumir el riesgo
3	Tiempo insuficiente para desarrollar el proyecto	Esta situación debe ser evitada en la medida de lo posible. Los objetivos del proyecto han de mantenerse acotados en función del progreso, con el objetivo de tener un alcance viable	Eliminar el riesgo
4	Fallo en vías de comunicación con tutores del proyecto	En caso de darse, han de establecerse vías alternas de comunicación o esperar a que vuelvan a estar operativas	Mitigar el riesgo
5	Pérdida de progreso	Uso de mecanismos de control de versiones de forma constante para evitar la posible pérdida de progreso o necesidad de retroceder a una versión anterior del proyecto	Eliminar el riesgo
6	Definición errónea de los requisitos del proyecto	Comprobar de forma reiterada los requisitos del proyecto junto a los tutores, así como hacer seguimiento del desarrollo para verificar que los requisitos se ajustan a sus necesidades, aparte de ser correctos	Eliminar el riesgo

Figura 4. Respuesta a los riesgos

1.11 Organización, gestión y planificación del proyecto

1.11.1 Responsables del proyecto

Los principales responsables del proyecto son los tutores asignados para la supervisión y seguimiento del mismo, así como el autor que lo realiza. Además, se puede considerar a la Universidad de Oviedo como entidad responsable y principal motivadora de la creación del proyecto, en calidad de Trabajo de Fin de Grado, así como los miembros del

grupo de *Investigación en Optimización y Scheduling Inteligentes* (iScOp) de la Universidad de Oviedo, que constituyen los usuarios finales.

1.11.2 Organización del proyecto

La organización del proyecto se basa en la planificación temporal, y en el progreso del proyecto respecto a esta. Se ha intentado mantener tanto los objetivos como el alcance del proyecto lo más acotados y cerrados posible, teniendo en cuenta posibles cambios en el alcance que fuesen viables (como funcionalidad adicional), siempre basándose en el avance hasta el momento de proponerse el cambio respecto a la planificación temporal.

1.11.3 Metodología utilizada

La metodología de trabajo empleada no ha sido ninguna metodología clásica que se siga en proyectos de gran calibre, sino que se ha basado en la supervisión continua del avance realizado por parte de los tutores, así como la dirección de estos sobre que tareas son más prioritarias temporalmente.

Además de esto, se ha procurado seguir un desarrollo incremental, manteniendo el contacto continuo con los tutores, preguntando por retroalimentación a los mismos sobre qué aspectos se han de tener en cuenta, así como aspectos a mejorar.

Aunque el desarrollo no sea como tal totalmente iterativo, sí que se ha organizado la realización del trabajo en diferentes hitos de la herramienta que constituye el proyecto.

1.11.4 Planificación Temporal

Nombre de tarea	Duración	Comienzo	Fin
Trabajo Fin de Grado	**39 días**	**vie 13/05/22**	**mié 06/07/22**
Desarrollo de la herramienta	**39 días**	**vie 13/05/22**	**mié 06/07/22**
Manejo de ficheros	**18 días**	**vie 13/05/22**	**mar 07/06/22**
Carga de CSV	1 día	vie 13/05/22	vie 13/05/22
Generación de CSV	1 día	lun 16/05/22	lun 16/05/22
Generación de imágenes	15 días	mar 17/05/22	lun 06/06/22
Generación de PDF	2 días	lun 06/06/22	mar 07/06/22
Manejo de reglas	**20 días**	**mié 25/05/22**	**mar 21/06/22**
Elaboración gramática	6 días	mié 25/05/22	mié 01/06/22

Nombre de tarea	Duración	Comienzo	Fin
Parser regla a montículo	3 días	jue 02/06/22	lun 06/06/22
Parser montículo a regla	3 días	mar 07/06/22	jue 09/06/22
Manipulación de reglas	2 días	vie 10/06/22	lun 13/06/22
Ejecución de reglas	6 días	mar 14/06/22	mar 21/06/22
Interfaz gráfica de usuario	**9 días**	**mié 08/06/22**	**lun 20/06/22**
Ventana principal	**9 días**	**mié 08/06/22**	**lun 20/06/22**
Panel representación gráfica	9 días	mié 08/06/22	lun 20/06/22
Panel de botones inferior	1 día	mié 08/06/22	mié 08/06/22
Barra de menú	1 día	mié 08/06/22	mié 08/06/22
Elaboración de la ayuda	2 días	mié 08/06/22	jue 09/06/22
Ventana historial	**3 días**	**jue 09/06/22**	**lun 13/06/22**
Elaboración de histórico	3 días	jue 09/06/22	lun 13/06/22
Panel de botones inferior	1 día	jue 09/06/22	jue 09/06/22
Pruebas	**19 días**	**vie 10/06/22**	**mié 06/07/22**
Pruebas unitarias	12 días	vie 10/06/22	lun 27/06/22
Pruebas de sistema	5 días	mar 28/06/22	lun 04/07/22
Pruebas de usabilidad	2 días	mar 05/07/22	mié 06/07/22
Documentación	**17 días**	**vie 13/05/22**	**lun 06/06/22**
Memoria	**11 días**	**mié 18/05/22**	**mié 01/06/22**
Hojas de identificación	1 día	mié 18/05/22	mié 18/05/22
Introducción	1 día	jue 19/05/22	jue 19/05/22
Motivación	1 día	jue 19/05/22	jue 19/05/22
Objeto	1 día	vie 20/05/22	vie 20/05/22
Normas y referencias	1 día	vie 20/05/22	vie 20/05/22
Requisitos iniciales	1 día	vie 20/05/22	vie 20/05/22
Solución propuesta	1 día	lun 23/05/22	lun 23/05/22
Alcance del proyecto	1 día	lun 23/05/22	lun 23/05/22
Definiciones y abreviaturas	1 día	mar 24/05/22	mar 24/05/22
Análisis de riesgos	1 día	mar 24/05/22	mar 24/05/22

Nombre de tarea	Duración	Comienzo	Fin
Organización, gestión y planificación del proyecto	4 días	mié 25/05/22	lun 30/05/22
Presupuesto	2 días	mar 31/05/22	mié 01/06/22
Aspectos teóricos	**3 días**	**jue 02/06/22**	**lun 06/06/22**
Optimización combinatoria y satisfacción de restricciones	3 días	jue 02/06/22	lun 06/06/22
Métodos de resolución	3 días	jue 02/06/22	lun 06/06/22
Reglas de prioridad	2 días	vie 03/06/22	lun 06/06/22
Problemas utilizados como casos de uso	3 días	jue 02/06/22	lun 06/06/22
Complejidad	1 día	lun 06/06/22	lun 06/06/22
Análisis	**4 días**	**vie 13/05/22**	**mié 18/05/22**
Definición del sistema	1 día	vie 13/05/22	vie 13/05/22
Requisitos del sistema	1 día	vie 13/05/22	vie 13/05/22
Identificación de subsistemas	1 día	lun 16/05/22	lun 16/05/22
Descripción de las clases	1 día	lun 16/05/22	lun 16/05/22
Casos de uso y escenarios	1 día	lun 16/05/22	lun 16/05/22
Análisis interfaces de usuario	1 día	mar 17/05/22	mar 17/05/22
Especificación plan de pruebas	2 días	mar 17/05/22	mié 18/05/22
Diseño del sistema	**7 días**	**jue 19/05/22**	**vie 27/05/22**
Arquitectura del sistema	2 días	jue 19/05/22	vie 20/05/22
Diseño de la interfaz gráfica	2 días	lun 23/05/22	mar 24/05/22
Especificación técnica de las pruebas	3 días	mié 25/05/22	vie 27/05/22
Implementación del sistema	**3 días**	**jue 26/05/22**	**lun 30/05/22**
Herramientas usadas para el desarrollo	1 día	jue 26/05/22	jue 26/05/22
Estudio de alternativas	1 día	vie 27/05/22	vie 27/05/22
Problemas en el desarrollo	1 día	lun 30/05/22	lun 30/05/22
Manuales del sistema	**4 días**	**mié 01/06/22**	**lun 06/06/22**
Manual de ejecución	2 días	mié 01/06/22	jue 02/06/22
Manual de desarrollador	2 días	mié 01/06/22	jue 02/06/22
Manual de usuario	2 días	vie 03/06/22	lun 06/06/22
Ampliaciones y trabajo futuro	2 días	vie 03/06/22	lun 06/06/22
Anexos	1 día	lun 06/06/22	lun 06/06/22

Figura 5. Planificación temporal del proyecto

1.12 Presupuesto

1.12.1 Presupuesto de costes

El presupuesto está organizado en tres partidas diferentes: Desarrollo de Software, Documentación y Otros costes (que incluye costes indirectos).

- Desarrollo de Software:

Descripción	Cantidad	Unidades	Precio	Subtotal
Desarrollo de la herramienta				
Manejo de Ficheros	58	horas	12,50 €	725,00 €
Manejo de Reglas	66	horas	12,50 €	825,00 €
Interfaz gráfica de usuario	36	horas	12,50 €	450,00 €
Pruebas	48	horas	12,50 €	600,00 €
Documentación	54	horas	12,50 €	675,00 €
Total				3275,00 €

Figura 6. Costes desarrollo

- Documentación:

Descripción	Cantidad	Unidades	Precio	Subtotal
Documentación				
Manual de ejecución	12	horas	12,50 €	150,00 €
Manual de usuario	20	horas	12,50 €	250,00 €
Total				400,00 €

Figura 7. Costes documentación

- Otros costes:

Descripción	Cantidad	Unidades	Precio	Subtotal (3)	Subtotal (2)
Costes indirectos					
Servicios empresa					255,00 €
Agua	1,5	meses	20,00 €	30,00 €	
Electricidad	1,5	meses	100,00 €	150,00 €	
Teléfono e Internet	1,5	meses	50,00 €	75,00 €	
Herramientas Software					1.270,00 €
Microsoft Project Pro 2019	1	licencia	1.030,00 €	1.030,00 €	
Microsoft Office 365 Premium	1	licencia	240,00 €	240,00 €	
Otros					100,00 €
Reuniones con clientes	8	horas	12,50 €		
Total					1.625 €

Figura 8. Costes indirectos

- Presupuesto final de costes:

PRESUPUESTO DE COSTES		
Cod	**Partida**	**Total**
1	Desarrollo de la herramienta	3.275,00 €
2	Documentación	400,00 €
3	Otros costes	1.625,00 €
	Total	5.300,00 €

Figura 9. Presupuesto final de costes

1.12.2 Presupuesto de cliente

El presupuesto de cliente tiene incluido un beneficio del 12% distribuido entre las partidas de desarrollo y de documentación (incluyendo gastos de los otros costes relacionados ocasionados durante el proyecto), dando un total de 636 € a añadir en cada una de estas partidas. Así mismo, en el total también se calculará un IVA del 21%.

Partida	**Ítem**	**Producto**	**Coste**
1		**Desarrollo de la herramienta**	**3.911,00 €**
	1	Manejo de Ficheros	1.303,67 €
	2	Manejo de Reglas	1.303,67 €
	3	Interfaz gráfica de usuario	1.303,67 €
2		**Formación**	**2661,00 €**
	1	Elaboración de los manuales de ejecución	1.330,50 €
	2	Elaboración de los manuales de usuario	1.330,50 €
		Total (Sin IVA)	6.572,00 €
		Total (IVA 21%)	7.952,12 €

Figura 10. Presupuesto de cliente

Capítulo 2. Aspectos teóricos

En este capítulo se introducirán algunos aspectos teóricos a los que se hace referencia a lo largo de la documentación del trabajo. En concreto, se tratarán aspectos relativos a lo que son los problemas de optimización combinatoria, las reglas de prioridad, qué métodos permiten calcularlas de manera automática, cómo se pueden representar y los problemas empleados para los que se calculan reglas representadas por la herramienta.

2.1 Problemas de optimización combinatoria y de satisfacción de restricciones

Los problemas de scheduling aparecen con bastante frecuencia en la vida real en numerosos entornos productivos y de servicios. Son problemas difíciles (frecuentemente pertenecientes a las clases de complejidad NP-completo y NP-duro) que requieren organizar la ejecución en el tiempo de una serie de tareas que compiten por el uso de un conjunto finito de recursos y que están sujetas a un conjunto de restricciones impuestas por factores como pueden ser las características físicas del entorno, la normativa laboral, etc. Lo que se pretende a la hora de resolver estos problemas es optimizar uno o varios objetivos, relacionados normalmente con el coste, el beneficio o el tiempo de ejecución.

Estos problemas son de naturaleza combinatoria, es decir, hay que elegir una opción entre un conjunto exponencialmente grande de combinaciones posibles. Por ello, los problemas de esta naturaleza precisan de algoritmos de búsqueda inteligentes para encontrar soluciones aceptables en un tiempo razonable. Así, en la literatura se pueden encontrar soluciones a estos problemas basadas en prácticamente todas las metaheurísticas conocidas y en particular en los algoritmos de búsqueda heurística propios de áreas como la Investigación Operativa y la Inteligencia Artificial.

2.2 Métodos de resolución

Uno de los métodos aproximados más utilizados en la literatura para la resolución de este tipo de problemas son las reglas de prioridad, debido a su fácil implementación y a su baja complejidad temporal. Las reglas de prioridad pueden ser empleadas para modificar el criterio de elección, en distintos algoritmos, a la hora de construir soluciones heurísticas.

En muchas situaciones de la vida cotidiana el dominio de estos problemas puede cambiar en tiempo real (a la vez que se va resolviendo). Por este motivo se emplean de forma regular algoritmos voraces guiados por reglas de prioridad. Estas reglas son, por lo general, definidas en base al conocimiento de los expertos sobre el dominio de un

problema dado, sobre el que se busca solución. Son fáciles de entender por los humanos, pero por lo general, son incapaces de capturar características no triviales sobre el dominio del problema no evidentes para el experto. Por tanto, el cálculo automático de reglas de prioridad se posiciona como un método alternativo. Para este fin se diseñan y desarrollan técnicas híper-heurísticas [3] en particular Programación Genética, recientemente empleada en algunos contextos [1] [2] [4] [5] [6] [12], entre los que se encuentran los problemas empleados como caso de uso de la herramienta: el problema de secuenciamiento de una Máquina con Capacidad Variable [1] [2] y el problema del Viajante de Comercio [6].

La Híper-heurística, es una heurística particular en la que el proceso de diseño de métodos heurísticos se automatiza para resolver problemas computacionales de búsqueda difíciles, es decir son "un método de búsqueda o mecanismo de aprendizaje para seleccionar o generar heurísticas para resolver problemas computacionales de búsqueda" [3]. De acuerdo con la naturaleza del espacio de búsqueda en el que se utiliza el modelo, la heurística se puede seleccionar a partir de heurísticas ya existentes o ser generada a partir de las componentes de las que ya existen, lo que se puede hacer mediante un método como la Programación Genética.

La Programación Genética es una especialización de los algoritmos genéticos donde cada individuo es un heurístico (o regla de prioridad), por lo que la búsqueda se realiza en un espacio de heurísticos en lugar de en un espacio de soluciones. Puede considerarse una técnica de aprendizaje automático usada para optimizar una población de heurísticos o reglas de prioridad. Dado que la mayoría de heurísticos pueden definirse mediante expresiones aritméticas, representables mediante árboles, los individuos de la población se representan de también mediante árboles (representados como montículos binarios), pues esta representación facilita la evaluación de los mismos.

2.3 Reglas de prioridad

Las reglas de prioridad son heurísticos que permiten tomar decisiones para seleccionar la siguiente tarea a planificar en el proceso de construcción de una solución, también denominada planificación. Son posiblemente uno de los métodos más usados para resolver problemas de scheduling, debido a su fácil implementación y a su baja complejidad temporal. Permiten establecer una prioridad para cada una de las tareas aún no planificadas, lo que permite a la estrategia empleada para construir una solución elegir entre ellas.

Generalmente una regla de prioridad se expresa mediante una función de prioridad que calcula una prioridad para cada tarea aún no planificada, y necesita de un mecanismo para construir planificaciones, normalmente un algoritmo voraz (greedy en inglés), en el que se emplean dichas reglas como proceso de discriminación entre las tareas, en función de la prioridad asignada por la regla. Los algoritmos voraces son algoritmos de búsqueda que, en cada paso de su ejecución, entre las distintas posibilidades a elegir

(por ejemplo, las siguientes tareas a planificar) discrimina según la prioridad que les otorga un heurístico o regla de prioridad.

Entre las ventajas de las reglas de prioridad se encuentran su rápida velocidad de ejecución en comparación con otros algoritmos, su sencilla implementación ya mencionada, la amplia gama de entornos dinámicos en los que se puede aplicar y su adaptación a cambios en las condiciones de un entorno de planificación. Su principal desventaja es que no garantizan la solución óptima, pues es un mecanismo que toma decisiones locales y no tiene en cuenta todo el problema.

2.4 Complejidad

La complejidad se puede definir como el tiempo, así como los recursos que se utilizan durante la resolución de un problema. Hay diferentes tipos de complejidades:

- **P (Polynomial):** Los problemas con este tipo de complejidad pueden ser resueltos de forma determinista en un tiempo polinomial.
- **NP (Nondeterministic polynomial time):** Este tipo de complejidad, como indica su nombre, aborda situaciones que se pueden resolver en un tiempo polinomial de forma no determinista. A su vez, esta clasificación se puede dividir en dos grandes grupos:
    - **NP-Duro (Nondeterministic polynomial time hardness):** Los problemas que poseen esta complejidad pueden reducirse a problemas del dominio **NP.** En otras palabras, una solución aplicable a un problema NP-Duro puede ser utilizada para resolver un problema **NP** en tiempo polinomial, en caso de encontrar un algoritmo que resuelva el problema **NP-Duro** en tiempo polinomial.
    - **NP-Completo:** Este tipo de problemas son aquellos que entran dentro de la categoría de **NP,** que a su vez son lo más difícil de resolver dentro de la categoría **NP.**

Cabe destacar que, hasta la fecha, existe un problema sin resolver conocido como “P vs NP”, que plantea si $P = NP$ o si $P \neq NP$. En otras palabras, plantea si existen problemas cuya solución se pueda validar en tiempo polinomial puedan ser resueltas en tiempo polinomial.

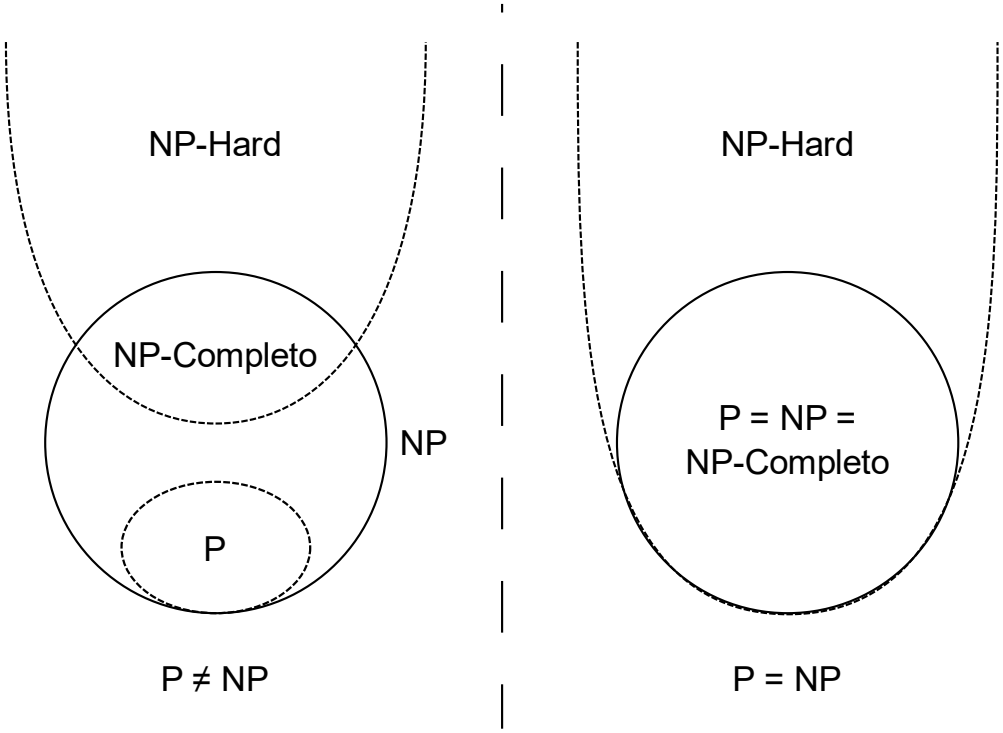

Figura 11. Diagrama de Euler con las familias de problemas P, NP, NP-Completo y NP-Duro [13].

En la imagen anterior se puede apreciar la relación entre los diferentes tipos de complejidades, planteando si $P = NP$ o si $P \neq NP$.

2.5 Problemas utilizados como casos de uso

2.5.1 Problema de secuenciamiento en una máquina con capacidad variable $(1, Cap(t)||\sum Ti)$

El problema de secuenciamiento de una máquina con capacidad variable consiste en planificar una serie de trabajos en una máquina, con el objetivo de minimizar la función objetivo del tiempo de retardo total (denominada tardiness). La característica singular de este problema es que la máquina tiene capacidad para procesar varios trabajos al mismo tiempo, y que esta capacidad es variable a lo largo del tiempo.

Este problema fue introducido en [14] el contexto de la planificación de los tiempos de recarga de flotas de vehículos eléctricos (EVCSP), y se denota con $(1, Cap(t)||\sum Ti)$. Resolver un prolema EVCSP requiere resolver a lo largo del tiempo múltiples instancias del problema $(1, Cap(t)||\sum Ti)$.

Dada la elevada complejidad computacional del problema ECVSP y la necesidad de respuesta en tiempo real este problema ha de ser resuelto en línea aplicando reglas de prioridad, estas reglas permiten decidir el siguiente trabajo a planificar (en el contexto

de los coches eléctricos, el siguiente vehículo a cargar), para ello, asignan una prioridad a cada trabajo, siendo el siguiente trabajo procesado, en un momento dado, aquel con mayor prioridad entre los disponibles.

Existen diversas reglas clásicas para la resolución de este problema, como la **EDD** (Earliest due date, da la mayor prioridad a la tarea con fecha de vencimiento, duedate, más temprano), **SPT** (shortest processing time, da la mayor prioridad a la tarea con menor tiempo de procesamiento), **ATC** (Apparent Tardiness Cost, regla más elaborada que establece la prioridad considerando más información del domino del problema que el resto, como puede ser el promedio de las duraciones, el tiempo más temprano en el que hay capacidad para que una tarea se pueda procesar, la duración de la tarea, su due date, etc. Se trata de una regla más compleja que presente un buen funcionamiento).

A pesar de poder diseñar reglas de manera manual, los métodos automáticos pueden considerar características del dominio del problema que no son evidentes para los expertos humanos, haciendo interesante abordar el cálculo de reglas de manera automática, empleando estrategias inteligentes como pueden ser los híper-heurísticos, pues la búsqueda debe realizarse en un espacio de heurísticos en lugar de en un espacio de soluciones. Por lo tanto, dado que las reglas de prioridad son expresiones aritméticas, que se representan de manera natural mediante árboles, es frecuente el uso de la Programación Genética para el cálculo automático de reglas de prioridad, como se ha realizado para este problema en [1] [2].

El problema $(1, Cap(t)||\sum T_i)$ se define de manera formal del siguiente modo: dado un conjunto de n trabajos $\{1, \dots, n\}$, todos disponibles en el instante $t = 0$, éstos deben ser planificados en una máquina cuya capacidad varía a lo largo del tiempo, siendo $Cap(t) \geq 0, t \geq 0$, la capacidad de la máquina en el intervalo $[t, t + 1)$. El trabajo i tiene una duración de p_i y un tiempo de vencimiento (due date) d_i. El objetivo es asignar tiempo de inicio $st_i, 1 \leq i \leq n$ a los trabajos en la máquina, de manera que se satisfagan las siguientes restricciones:

i. En cada instante de tiempo $t \geq 0$ el número de trabajos que son procesados en paralelo en la máquina, $X(t)$, no puede exceder la capacidad de dicha máquina:

$$X(t) \leq Cap(t).$$

ii. El procesamiento de los trabajos en la máquina no puede ser interrumpido:

$$C_i = st_i + p_i.$$

Donde C_i es el tiempo de fin del trabajo i.

La función objetivo, el tiempo de retardo total (Total Tardiness):

$$\sum_{i=1}^{n} \max(0, C_i - d_i)$$

debe ser minimizada.

Un caso particular de este problema es cuando la capacidad de la máquina permanece constante a lo largo del tiempo. Este es el problema de máquinas paralelas idénticas, que es NP-duro y se denota como $(P||\sum T_i)$. Se deduce, por tanto, que el problema $(1, Cap(t)||\sum T_i)$ es también NP-duro.

A continuación, se muestra una imagen que contiene el árbol de expresión y el montículo binario en el que se almacena la regla para la regla: $\min(-p_j, y(\propto) - d_j)$.

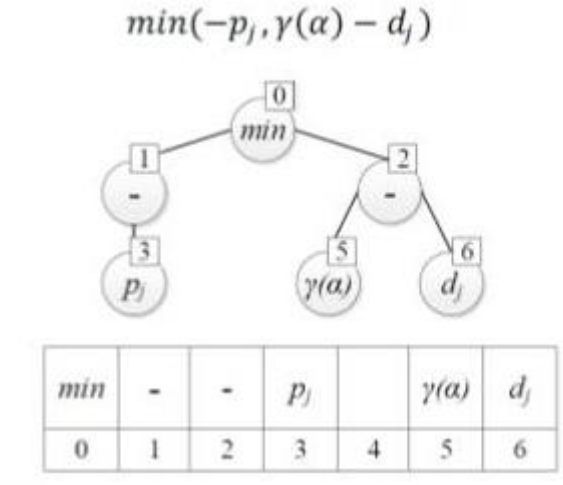

Figura 12. Árbol de expresión y representación del montículo para una regla del problema $(1, Cap(t)||\sum T_i)$ *[7]*

En el caso de la regla ATC (Apparent Tardiness Cost):

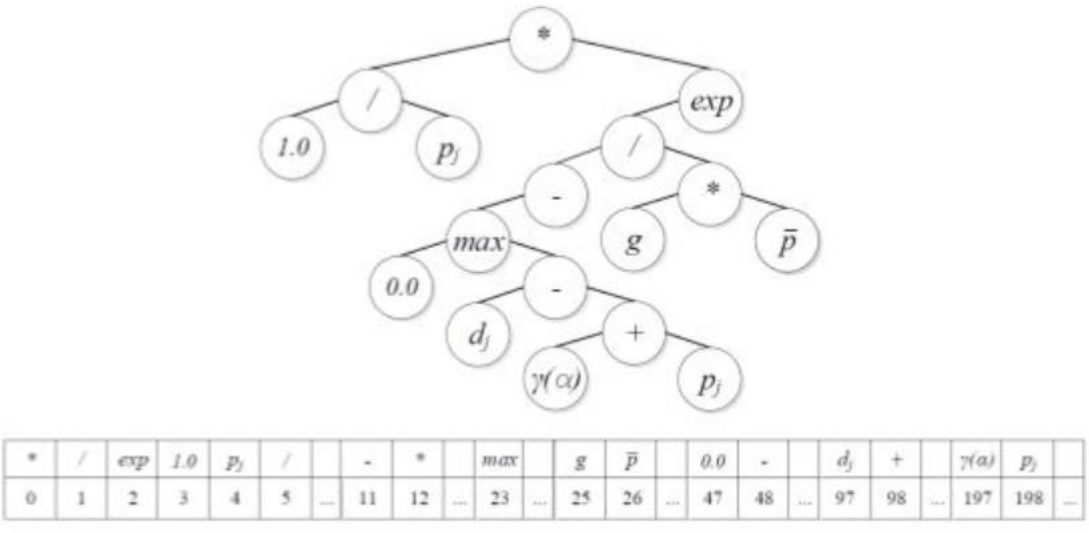

Figura 13. Árbol de expresión y representación del montículo para la regla ATC para $(1, Cap(t)||\sum T_i)$ *[7]*

2.5.2 Problema del Viajante del comercio

El problema del viajante de comercio es un problema de optimización combinatoria clásico, al que se suele conocer por sus siglas en inglés como TSP (Travelling Salesman Problem). Fue descrito por primera vez en 1857 por W.R. Hamilton y Thomas Kirkman en uno de sus trabajos en los que se considera un puzle realizado sobre un dodecaedro, y representado mediante un grafo, siendo el objetivo hallar un ciclo hamiltoniano que recorriera todos sus vértices.

Puede ser enunciado del siguiente modo: "Dada una lista de ciudades, y las distancias entre cada par de ellas, ¿cuál es la ruta más corta que permite visitar todas las ciudades una sola vez finalizando en la ciudad de partida?".

El problema del TSP es uno de los problemas de optimización más estudiados en la literatura a lo largo de la historia, especialmente en el campo de la Ciencia de la Computación y particularmente de la Inteligencia Artificial. Se trata de un problema NP-duro que cuenta con multitud de aplicaciones en el sector productivo y logístico. Para enunciar el problema de manera formal es necesario introducir la siguiente terminología:

Sea un grafo $G = (V, A, D)$ donde V es el conjunto de vértices, A el de aristas y $D = (d_{ij})$ la matriz de costes, es decir d_{ij} es el coste o distancia de la arista que une a los vértices i y j.

Se entiende como camino a una sucesión de aristas $(a_1, a_2, \dots, a_k)$ donde el vértice final de cada arista coincide con el inicial de la siguiente.

- Un camino es simple si no utiliza el mismo vértice más de una vez.
- Un ciclo es un camino $(a_1, a_2, \dots, a_k)$ en el que el vértice inicial es el mismo que el final.
- Un ciclo es simple si el camino que lo define también lo es.
- Un subtour es un ciclo simple que no visita todos los vértices del grafo.
- Un tour o ciclo hamiltoniano es un ciclo simple que visita todos los vértices del grafo.

Así pues, el Problema del Viajante consiste en determinar un tour con coste mínimo. Es decir, la función objetivo del TSP es minimizar el coste del tour, de forma que la suma de las distancias del tour sea lo más pequeña posible.

A lo largo de la historia se han propuesto diferentes algoritmos para resolver este problema, como, por ejemplo, algoritmos genéticos o el conocido heurístico Lin-Kernighan [15] sin embargo, ninguno de ellos en la actualidad es eficiente para resolver instancias de gran tamaño en entornos dinámicos. En estas situaciones, un algoritmo voraz guiado por una regla de prioridad es la mejor alternativa para la resolución de este problema. Una regla de prioridad para este problema es una expresión aritmética que

asigna una prioridad a cada ciudad aún no visitada. Esta prioridad se calcula teniendo en cuenta información sobre el dominio del problema, por ejemplo, la regla heurística NN sólo tiene en cuenta la distancia entre dos ciudades d_{ij}: si *i* es la ciudad actual la prioridad de visitar desde *i* la ciudad *j* es dada por $1/d_{ij}$. Una buena regla de prioridad es capaz de proporcionar buenas soluciones para un buen número de instancias, aunque no para todas. Al mismo tiempo, diferentes reglas pueden producir diferentes soluciones para diferentes instancias. Por esta razón, es frecuente que una única regla no sea robusta para todas las instancias de un conjunto, y pueda ser adecuado explotar conjuntos de reglas que permitan tomar decisiones agregando los valores individuales de varias reglas en lugar de sólo una única regla, esta aproximación ha sido abordada recientemente en [6], donde se construyen conjuntos de reglas para abordar el problema TSP.

Capítulo 3. Análisis

3.1 Definición de alcance del sistema

El objetivo principal del proyecto es el desarrollo de una herramienta software gráfica, orientada a entornos de escritorio, motivado por la necesidad de automatizar el proceso de carga de reglas de prioridad previamente calculados por Programación Genética. Para ello, se usará el lenguaje de programación Java debido a la versatilidad que posee de poder ejecutarse en cualquier equipo independientemente de su sistema operativo.

Las principales operaciones que ha de realizar la aplicación son las siguientes:

- Representación gráfica de las reglas de prioridad.
- Conversión de las reglas de prioridad a montículo binario y viceversa.
- Manipulación de las reglas de prioridad.
- Carga de archivos con extensión ".csv" que contengan un conjunto de reglas a almacenar en el programa.
- Almacenamiento de las reglas en formato ".csv" que incluya el resultado de las mismas.
- Almacenamiento de las reglas en formato árbol, contenidas en ficheros con formato ".pdf".
- Ejecución de las reglas.

El objetivo principal de la aplicación consiste en dar apoyo a usuarios expertos del dominio en el que se basa la herramienta, permitiendo agilizar las tareas relacionadas con reglas de prioridad que se manejen.

Las descripciones descritas con anterioridad a modo de resumen son las características principales de la herramienta. Existen otras funcionalidades, que a pesar de no haber sido implementadas por estar fue del alcance establecido en el proyecto, serán detallas en la sección de ampliaciones.

3.2 Requisitos del sistema

3.2.1 Requisitos funcionales

3.2.1.1 Descripción de la aplicación

ID	Descripción
RA_1	La aplicación permitirá al usuario cargar las reglas de prioridad desde ficheros.
RA_1.1	Las modificaciones pertinentes serán las siguientes.
RA_1.1.1	Carga de fichero con extensión ".csv".
RA_1.1.1.1	Dicho archivo constará de 1 a N expresiones.
RA_1.1.1.1.1	Una expresión estará compuesta por:
RA_1.1.1.1.1.1	Regla aritmética. Ej: (5+3)
RA_1.1.1.1.1.2	Montículo binario. Ej: + 5 3
RA_1.1.1.1.1.2.1	Estos a su vez podrán ser elementos "null" (para indicar la ausencia de un símbolo).
RA_1.1.1.1.1.3	Resultado numérico (decimal), en caso de existir.
RA_1.1.1.1.2	Se mostrará un error en caso de que la regla no sea válida.
RA_1.1.2	Cambiar los elementos de la regla (símbolos de la regla)
RA_1.1.2.1	Los cambios se harán en formato texto, escribiendo como se conformará la nueva regla.
RA_1.1.2.1.1	Una vez efectuado el cambio, se podrá representar el árbol binario que codifica la regla.
RA_1.2	Las reglas se compondrán de los siguientes símbolos, permitiendo la combinación de estos:
RA_1.2.1	Terminales (compuestos por variables del dominio y constantes numéricas).
RA_1.2.2	Operadores aritméticos, que podrán ser:
RA_1.2.2.1	Unarios. Solo un símbolo como argumento.
RA_1.2.2.2	Binarios. Con dos símbolos como argumentos.

Figura 14. Requisitos de la aplicación general

3.2.1.2 Manipulación de las reglas de prioridad

ID	Descripción
RM_1	La aplicación dispondrá de un histórico de las operaciones realizadas.
RM_1.1	Dicho histórico contendrá las expresiones generadas por el usuario.
RM_1.2	El histórico también contendrá las expresiones que se hayan cargado desde un fichero externo con formato ".csv".
RM_2	Las operaciones a realizar con los elementos del histórico son:
RM_2.1	Carga de la regla para posterior manipulación. (Como en **RA_1.1**).
RM_2.2	Eliminación de la expresión del histórico.
RM_2.3	Almacenamiento de la expresión.
RM_2.3.1	Almacenada, en formato ".csv" siguiendo el formato definido en **RA_1.1.1.1.1.**
RM_2.3.2	Guardada, a modo de representación visual del árbol binario en formato ".pdf".
RM_3	La aplicación permitirá transformar reglas en árboles binarios y viceversa.
RM_3.1	La representación de las reglas se hará siguiendo la estructura de un montículo binario.
RM_3.1.1	Esta representación a su vez podrá ser:
RM_3.1.1.1	Modificada, como se menciona en **RA_1.1.2.**
RM_3.1.1.2	Ejecutada, mostrando a su vez el resultado de la regla ejecutada, en caso de existir.

Figura 15. Requisitos sobre manipulación de reglas

3.2.1.3 Resolución de las instancias

ID	Descripción
RR_1	El usuario administrará un archivo con extensión ".jar" que dictaminará la forma en la que se han de ejecutar las reglas.
RR_1.1	Este fichero será un ejecutable que reciba por parámetro la regla de prioridad y la instancia del problema a resolver.
RR_1.1.2	El resultado de la operación (número decimal) se mostrará al usuario.
RR_1.1.3	En caso de error, se notificará al usuario.
RR_2	El usuario administrará un fichero ".conf" con los operadores y terminales de las reglas definidos.
RR_2.1	El formato del fichero será el visto en los anexos.
RR_2.2	En caso de ejecutar una regla con un operador no definido, el sistema mostrará un error.

Figura 16. Requisitos resolución de instancias

3.2.2 Requisitos no funcionales

ID	Descripción
RNF_1	La aplicación se implementará en el lenguaje Java.
RNF_1.1	La versión utilizada será Java 8.
RNF_1.2	Las interfaces gráficas se implementarán con Swing.
RNF_2	El sistema de ayuda de la aplicación se hará con HTML.
RNF_2.1	Se implementará utilizando la librería JavaHelp.
RNF_3	La conversión a formato ".pdf" se hará mediante iText
RNF_4	Las dimensiones donde se representará la regla serán las de una hoja A4 en horizontal.

Figura 17. Requisitos no funcionales

3.3 Identificación de los subsistemas

3.3.1 Descripción de los subsistemas

La estructura principal de la aplicación se ha dividido en diferentes capas, resultando en la siguiente jerarquía de paquetes:

- **gui:** El paquete en el que se encuentra la interfaz gráfica de usuario. Se trata de la capa principal y más externa del sistema. En esta capa, el usuario interactúa con las ventanas gráficas del programa, hechas mediante la herramienta Swing de Java. Dentro de este paquete, aparte de las ventanas utilizadas en la aplicación, se encuentra una clase encargada de cambiar el comportamiento por defecto de una tabla, haciendo que no se pueda modificar los valores de ninguna fila o de ninguna columna, cosa que no interesa para este programa en cuestión.
- **logic:** Este es el paquete correspondiente a la lógica de negocio. Dentro de este paquete se encuentran las clases encargadas de la conversión de reglas de un formato a otro. En este paquete a su vez, se encuentran los siguientes paquetes:
    - **ast:** contiene una única clase, que se encarga de definir un árbol de sintaxis abstracto que permite desglosar los diferentes elementos generados por la herramienta Antlr.
    - **visitor:** contiene clases necesarias para implementar el patrón de mismo nombre [16] con el objetivo de determinar qué hacer con los elementos recogidos por la herramienta Antlr.
- **parser:** En este paquete se encuentran todas las clases generadas mediante la herramienta Antlr, así como la gramática necesaria para la generación de estas.
- **util:** Contiene clases cuyo propósito consiste en proporcionar utilidad al resto de clases, concretamente, realizar operaciones de carga/guardado de ficheros y redireccionar salidas por consola (necesario para el ejecutable que el usuario suministra).

3.3.2 Descripción de inferfaces entre subsistemas

Debido a la separación en paquetes y tener la lógica de negocio separada de la interfaz gráfica, es necesario especificar en que puntos una capa se comunica con otra. Las clases de la capa “gui” a su vez llaman únicamente a la clase “ExpressionParser” del paquete “logic”, que es la que se encarga de la conversión de las reglas a montículos y viceversa. Por último, las clases del paquete “gui” también llaman a la clase del paquete “útil” “FileUtil”, encargada de guardar/cargar los ficheros necesarios para que la aplicación funcione.

3.4 Descripción de las clases

3.4.1 Paquete GUI

Nombre
MainApplication
Descripción
Representa la ventana principal de la interfaz gráfica.
Responsabilidades
Representar gráficamente la ventana principal. Invocar funcionalidad desde la ventana principal.

Figura 18. Descripción MainApplication

Nombre
HistoricalWindow
Descripción
Representa el historial de la aplicación.
Responsabilidades
Representar el historial de la aplicación. Invocar funcionalidad desde la ventana del historial.

Figura 19. Descripción HistoricalWindow

Nombre
NonEditableModel
Descripción
Redefine funcionalidad de tabla.
Responsabilidades
Redefinir funcionalidad de una JTable para hacer que los valores de sus filas no sean modificables.

Figura 20. Descripción NonEditableModel

Nombre
TreeView
Descripción
Muestra la representación gráfica de las reglas introducidas por el usuario.
Responsabilidades
Representar gráficamente las reglas introducidas por el usuario.

Figura 21. Descripción TreeView

3.4.2 Paquete logic

Nombre
ExpressionParser
Descripción
Parser de reglas de formato inorden a montículo y viceversa.
Responsabilidades
Parser de reglas de formato inorden a montículo y viceversa.

Figura 22. Descripción ExpressionParser

Nombre
ParsedExpression
Descripción
Representa el resultado de convertir una regla a formato montículo.
Responsabilidades
Representa el resultado de convertir una regla a formato de montículo.

Figura 23. Descripción ParsedExpression

Nombre
Rule
Descripción
Representa la información que contiene una regla.
Responsabilidades
Representar la información que contiene una regla, a modo de DTO (Data Transfer Object).

Figura 24. Descripción Rule

3.4.2.1 Paquete ast

Nombre
MyExpression
Descripción
Representa la estructura de los elementos que encuentra Antlr.
Responsabilidades
Representar el Abstract Syntax Tree (AST) generado por la herramienta Antlr. De esta clase se obtiene a su vez mediante el visitor la expresión ya procesada y se almacena en ParsedExpression.

Figura 25. Descripción MyExpression

3.4.2.2 Paquete visitor

Nombre
Visitor
Descripción
Interfaz que define como han de actuar el resto de Visitor.
Responsabilidades
Interfaz que define los métodos que tendrá un Visitor.

Figura 26. Descripción Visitor

Nombre
ParserVisitor
Descripción
Se encarga de visitar cada elemento MyExpression generado por Antlr.
Responsabilidades
Se encarga de calcular la posición correspondiente a un montículo binario para cada elemento MyExpression generado por Antlr.

Figura 27. Descripción ParserVisitor

3.4.3 Paquete parser

Nombre
TFGLexer
Descripción
Clase autogenerada por Antlr que define los tokens a utilizar.
Responsabilidades
Define los tokens a utilizar a la hora de procesar la expresión que compone el AST.

Figura 28. Descripción TFGLexer

Nombre
TFGParser
Descripción
Clase autogenerada por Antlr que procesa los tokens que recibe.
Responsabilidades
Procesa los tokens que componen el AST.

Figura 29. Descripción TFGParser

Nombre
TFG.g4
Descripción
Archivo de texto que contiene la gramática y tokens necesarios para la generación de TFGLexer y TFGParser.
Responsabilidades
Estructurar la gramática necesaria para reconocer cadenas de texto.

Figura 30. Descripción TFG.g4

3.4.4 Paquete util

Nombre
FileUtil
Descripción
Clase encargada del manejo de operaciones entrada/salida, relacionadas a ficheros.
Responsabilidades
Manejar ficheros de entrada/salida.

Figura 31. Descripción FileUtil

Nombre
CustomPrintStream
Descripción
Redefine las salidas por consola del programa.
Responsabilidades
Redefine la salida por consola de Antlr para procesar en caso de error y redirige la salida por consola del ejecutable ".jar" suministrado por el usuario para poder leerlo en el programa.

Figura 32. Descripción CustomPrintStream

3.5 Casos de uso y escenarios

3.5.1 Caso de uso I: Carga de fichero de configuración

Carga de un fichero de configuración	
Precondiciones	La aplicación debe haberse ejecutado.
Postcondiciones	Los datos del fichero de configuración son recogidos por el sistema.
Actores	Iniciado por el usuario y finalizado por el sistema.
Descripción	El usuario del sistema introduce un fichero con extensión ".conf" que dictamina los elementos que han de existir al momento de **ejecutar** una regla. 1. El usuario deberá hacer click en la opción de menú "Archivo". 2. El usuario deberá presionar, dentro de este menú la opción "Cargar fichero .conf". 3. El usuario deberá seleccionar el directorio donde se encuentre el fichero especificado. 4. El usuario deberá hacer click en la opción "Abrir" una vez seleccionado el fichero en concreto.
Escenarios alternativos	• El usuario selecciona un archivo con la extensión correspondiente pero que no contiene información relevante. El sistema no recogerá información necesaria para el procesamiento de las reglas.
Comentarios	El sistema no mostrará error en este punto en caso de proporcionarse un fichero con extensión válida pero que no contenga información relevante. Este aviso se verá reflejado al momento de ejecutar la regla en cuestión.

Figura 33. Caso de uso I: Carga de fichero de configuración

3.5.2 Caso de uso II: Representación de una regla (Campo Regla)

Representación de una regla (Campo Regla)	
Precondiciones	La aplicación debe haberse ejecutado y ha de haberse cargado un fichero de configuración.
Postcondiciones	El campo "Montículo" se rellena acorde a las especificaciones del fichero de configuración y del campo "Regla".
Actores	Iniciado por el usuario y finalizado por el sistema.
Descripción	El usuario introduce una regla a procesar en el campo "Regla" y obtiene su representación en forma de "Montículo". 1. El usuario introduce una regla válida. 2. El campo "Montículo" se rellena en función del campo "Regla" y de los elementos del fichero de configuración suministrado. 3. Se mostrará en pantalla, a su vez, la representación en forma gráfica que tendría la regla introducida.
Escenarios alternativos	• El usuario no selecciona un fichero de configuración con información relevante. El sistema tratará de representar la regla en función de las reglas establecidas mediante la gramática creada con la herramienta Antlr, pudiendo generar error en la representación al no especificarse la especificación que debería tener. • El usuario introduce una regla que no es válida desde el punto de vista de la gramática generada por Antlr. El sistema mostrará un error notificando de esto al usuario. • El usuario no introduce ningún dato y trata de realizar una representación. El sistema notificará al usuario mediante un error.
Comentarios	El sistema no mostrará error en este punto en caso de proporcionarse no haber información relevante. El aviso se verá reflejado al momento de tratar de ejecutar una regla.

Figura 34. Caso de uso II: Representación de una regla (Campo Regla)

3.5.3 Caso de uso III: Representación de una regla (Campo Montículo)

Representación de una regla (Campo Montículo)	
Precondiciones	Haber representado anteriormente una regla como se específica en el caso de uso anterior.
Postcondiciones	El sistema deberá rellenar el campo "Regla" en función del campo "Montículo"
Actores	Iniciado por el usuario y finalizado por el sistema.
Descripción	El usuario del sistema hará click en la opción "Representar" con el campo "Montículo" no vacío para obtener la representación del campo "Regla" 1. El usuario hace click en la opción "Representar". 2. El sistema rellenará el campo "Regla" con la información proporcionada.
Escenarios alternativos	• No hay ningún dato en ninguno de los campos "Regla" o "Montículo". El sistema mostrará un error notificando al usuario.
Comentarios	N/A

Figura 35. Caso de uso III: Representación de una regla (Campo Montículo)

3.5.4 Caso de uso IV: Almacenamiento de imagen

Almacenamiento de imagen	
Precondiciones	Haber representado anteriormente una regla
Postcondiciones	El sistema deberá generar un archivo con extensión ".pdf" que contenga una imagen mostrando la representación gráfica de la regla.
Actores	Iniciado por el usuario y finalizado por el sistema.
Descripción	El usuario hará click en la opción "Guardar Imagen" con una representación válida en pantalla 1. El usuario presiona la opción para guardar la imagen. 2. El usuario selecciona la ubicación, así como el nombre del fichero a guardar
Escenarios alternativos	• No hay imagen mostrada en pantalla. El sistema mostrará un error notificando al usuario. • Alguno de los campos "Regla" o "Montículo" se encuentra vacío a pesar de haber una representación válida. El sistema mostrará un error notificando al usuario. • No se genera archivo porque el usuario cancela la acción.
Comentarios	N/A

Figura 36. Caso de uso IV: Almacenamiento de imagen

3.5.5 Caso de uso V: Almacenamiento de CSV

Almacenamiento de CSV	
Precondiciones	Haber representado anteriormente una regla
Postcondiciones	El sistema deberá generar un archivo con extensión ".csv" que contenga una regla, con los valores de los campos "Regla" y "Montículo" como mínimo y, los valores de "Instancia" y "Resultado" en caso de haberse ejecutado la regla.
Actores	Iniciado por el usuario y finalizado por el sistema.
Descripción	El usuario hará click en la opción "Guardar CSV" con una regla válida en la aplicación 1. El usuario hará click en la opción "Guardar CSV". 2. El usuario seleccionará la ubicación, así como el nombre del fichero a guardar.
Escenarios alternativos	• No está la información de los campos mínimos "Regla" y "Montículo" rellena, el sistema mostrará un error notificando al usuario. • No se genera archivo porque el usuario cancela la acción o no introduce nombre del archivo a guardar.
Comentarios	N/A

Figura 37. Caso de uso V: Almacenamiento de CSV

3.5.6 Caso de uso VI: Ejecución de una regla

Ejecución de una regla	
Precondiciones	Haber cargado un fichero de configuración válido, haber representado anteriormente una regla, que sus elementos se encuentren en el fichero de configuración.
Postcondiciones	El sistema deberá mostrar en pantalla, así como rellenar el campo "Resultado" con la operación realizada en caso de ser exitosa. El sistema también tendrá que incluir la operación en el histórico de la aplicación.
Actores	Iniciado por el usuario y finalizado por el sistema.
Descripción	El usuario hará click en la opción "Ejecutar" con una regla válida en la aplicación 1. El usuario presionará la opción "Ejecutar". 2. El usuario introducirá la ubicación de un ejecutable con extensión ".jar" que dictamine como ha de calcularse el resultado de la regla. 3. El usuario introducirá la ubicación de un fichero con extensión ".txt" que dictamine la instancia sobre la cual se quiere calcular la regla. 4. El sistema devolverá el resultado obtenido.
Escenarios alternativos	• No está la información de los campos mínimos "Regla" y "Montículo" rellena, el sistema mostrará un error notificando al usuario. • Algún elemento descrito en la regla no se encuentra descrito en el fichero de configuración. El sistema mostrará un error. • No se calcula el resultado de la operación porque el usuario cancela la acción. Se mostrará un error relativo a la causa por el que no se ha podido completar una acción en concreto.
Comentarios	N/A

Figura 38. Caso de uso VI: Ejecución de una regla

3.5.7 Caso de uso VII: Guardar PDF desde historial

Guardar PDF desde historial	
Precondiciones	Historial no vacío
Postcondiciones	El sistema deberá generar un archivo con extensión ".pdf" por cada elemento que sea representable que se encuentre en el histórico.
Actores	Iniciado por el usuario y finalizado por el sistema.
Descripción	El usuario seleccionará la opción de "Guardar PDF" con elementos activos en el historial. 1. El usuario presiona la opción "Guardar PDF". 2. El usuario selecciona nombre base de los archivos, así como ubicación donde guardarlos.
Escenarios alternativos	• El historial se encuentra vacío, se mostrará un error notificando al usuario. • Alguna regla tiene una representación no válida. Se mostrará un error indicando la fila cuya regla no es válida al usuario. • No se genera archivo porque el usuario cancela la acción.
Comentarios	N/A

Figura 39. Caso de uso VII: Guardar PDF desde historia

3.5.8 Caso de uso VIII: Guardar CSV desde historial

Guardar CSV desde historial	
Precondiciones	Historial no vacío
Postcondiciones	El sistema deberá generar un único archivo con extensión ".csv" por cada elemento que se encuentre en el histórico.
Actores	Iniciado por el usuario y finalizado por el sistema.
Descripción	El usuario seleccionará la opción de "Guardar CSV" con elementos activos en el historial. 1. El usuario presiona la opción "Guardar CSV". 2. El usuario selecciona nombre del archivo, así como ubicación donde guardarlo.
Escenarios alternativos	• El historial se encuentra vacío, se mostrará un error notificando al usuario. • No se genera archivo porque el usuario cancela la acción.
Comentarios	N/A

Figura 40. Caso de uso VIII: Guardar CSV desde historial

3.5.9 Caso de uso IX: Carga de CSV

Cargar CSV	
Precondiciones	Haber ejecutado la aplicación
Postcondiciones	El sistema deberá cargar las reglas que se encuentren en el fichero ".csv" suministrado
Actores	Iniciado por el usuario y finalizado por el sistema.
Descripción	El usuario seleccionará la opción de "Cargar CSV" desde el historial. 1. El usuario presionará la opción de "Cargar CSV". 2. El usuario seleccionará un fichero con extensión ".csv" 3. El contenido del fichero se mostrará por pantalla al usuario.
Escenarios alternativos	• Se cancela la acción de cargar fichero ".csv"
Comentarios	N/A

Figura 41. Carga de CSV

3.6 Análisis de interfaces de usuario

La aplicación dispondrá de una ventana principal, dividida en paneles para asegurar que se pueda ajustar a cualquier tamaño de pantalla, cuadrando el tamaño de los elementos en función de su disposición en pantalla.

A continuación, se muestra el diseño de la ventana principal.

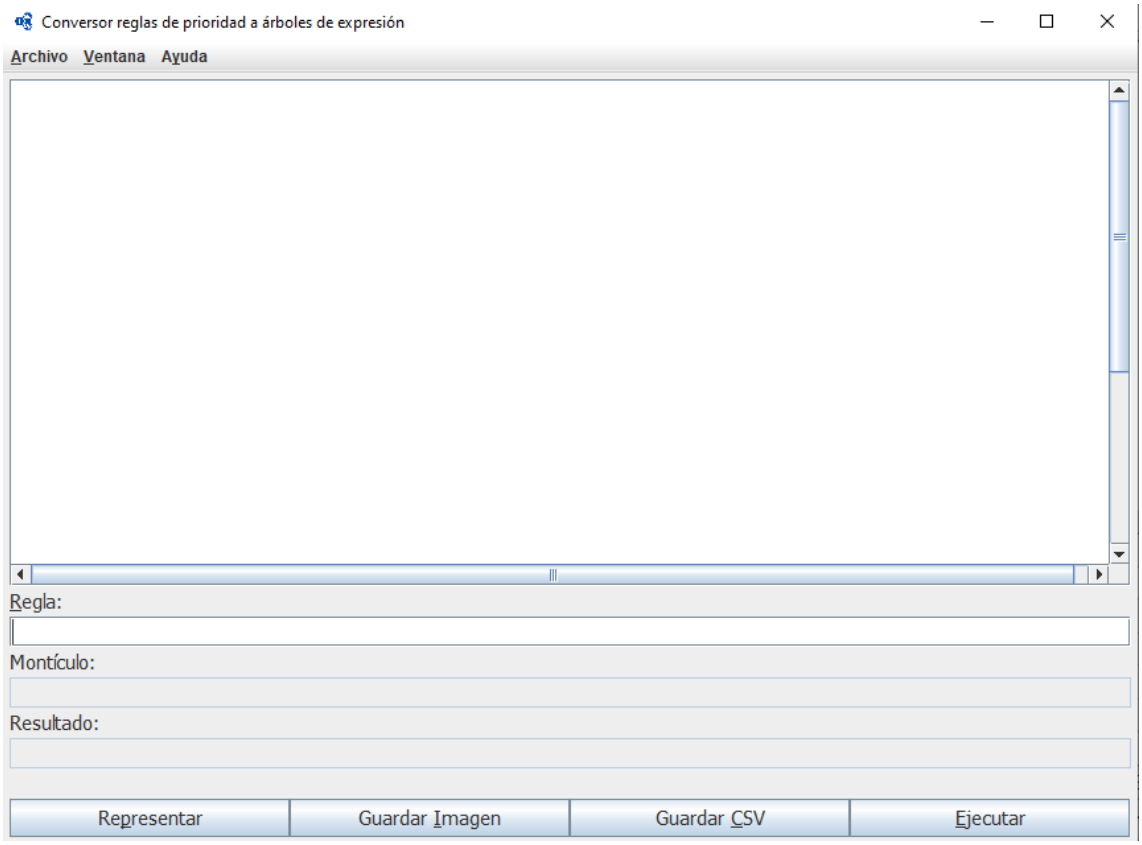

Figura 42. Diseño ventana principal

Lo primero a hacer por el usuario sería abrir la opción de menú y presionar la opción "Cargar fichero .conf". Esto es recomendable hacerlo al inicio para poder generar de forma correcta la representación de "Montículo" a "Regla", en caso de borrar este último campo por error, para representar de forma correcta las expresiones binarias que no sigan un formato inorden. Una vez hecho esto, al usuario le saldrá un selector de ficheros del siguiente aspecto.

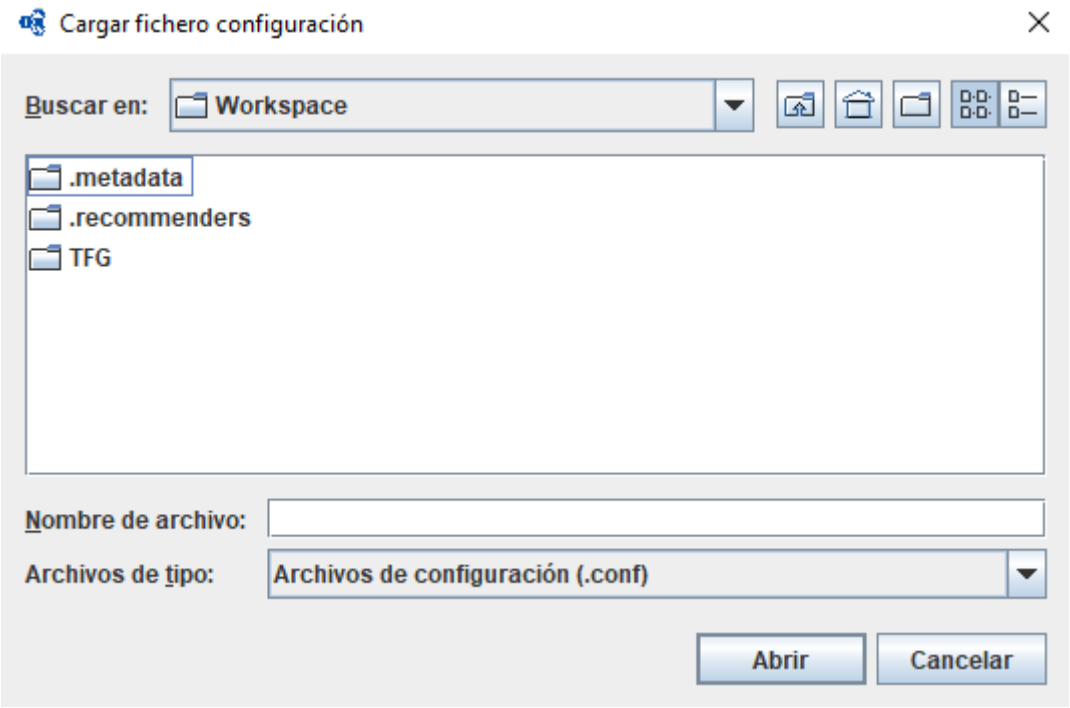

Figura 43. Selector de fichero ".conf"

Una vez cargado el fichero de configuración, puede procederse a la representación de elementos de forma gráfica. Para ello, se introducirá en el campo “Regla” la expresión a representar. Una vez hecho esto, se hará click en la opción “Representar”. Un ejemplo de cómo resultaría la operación se puede ver a continuación.

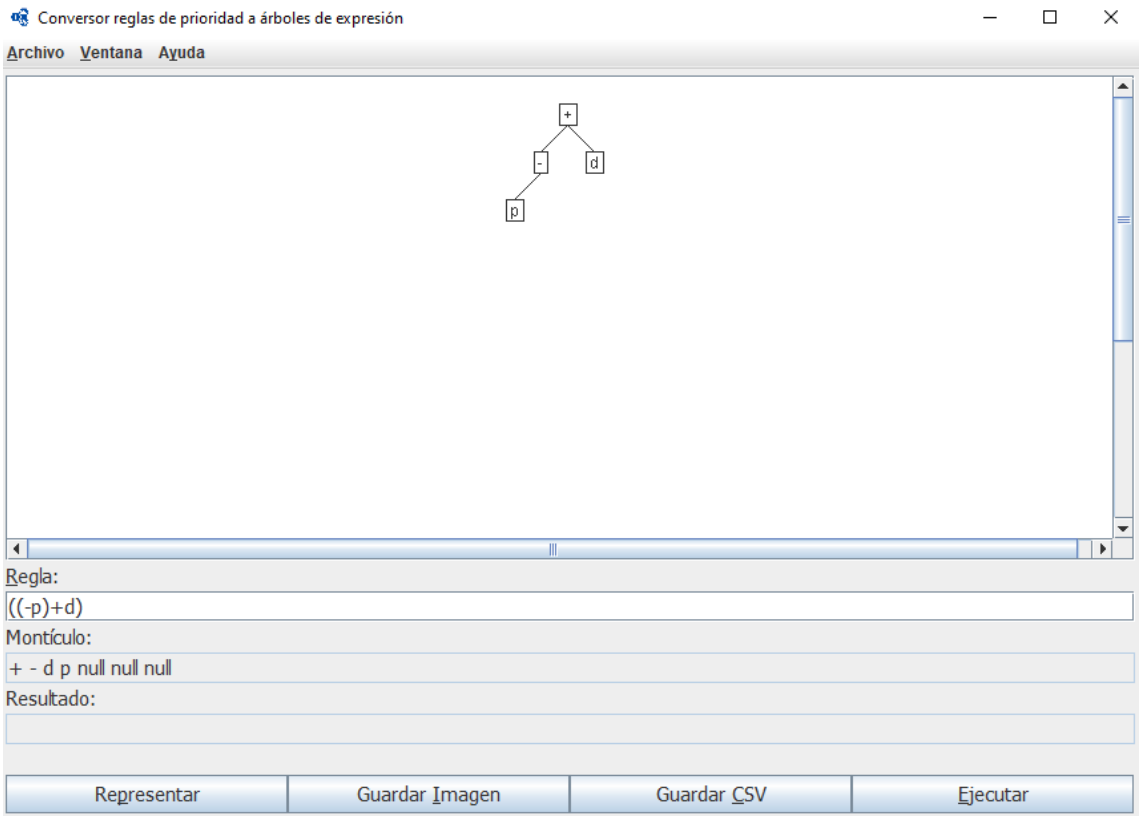

Figura 44. Aplicación tras representar una regla

Por último, para dejar constancia de todas las interfaces relevantes con las que puede interactuar el usuario, se muestra la vista que tiene el historial de la aplicación, mostrado tras pulsar en la opción de menú “Ventana”, en la única opción que dispone “Mostrar histórico”.

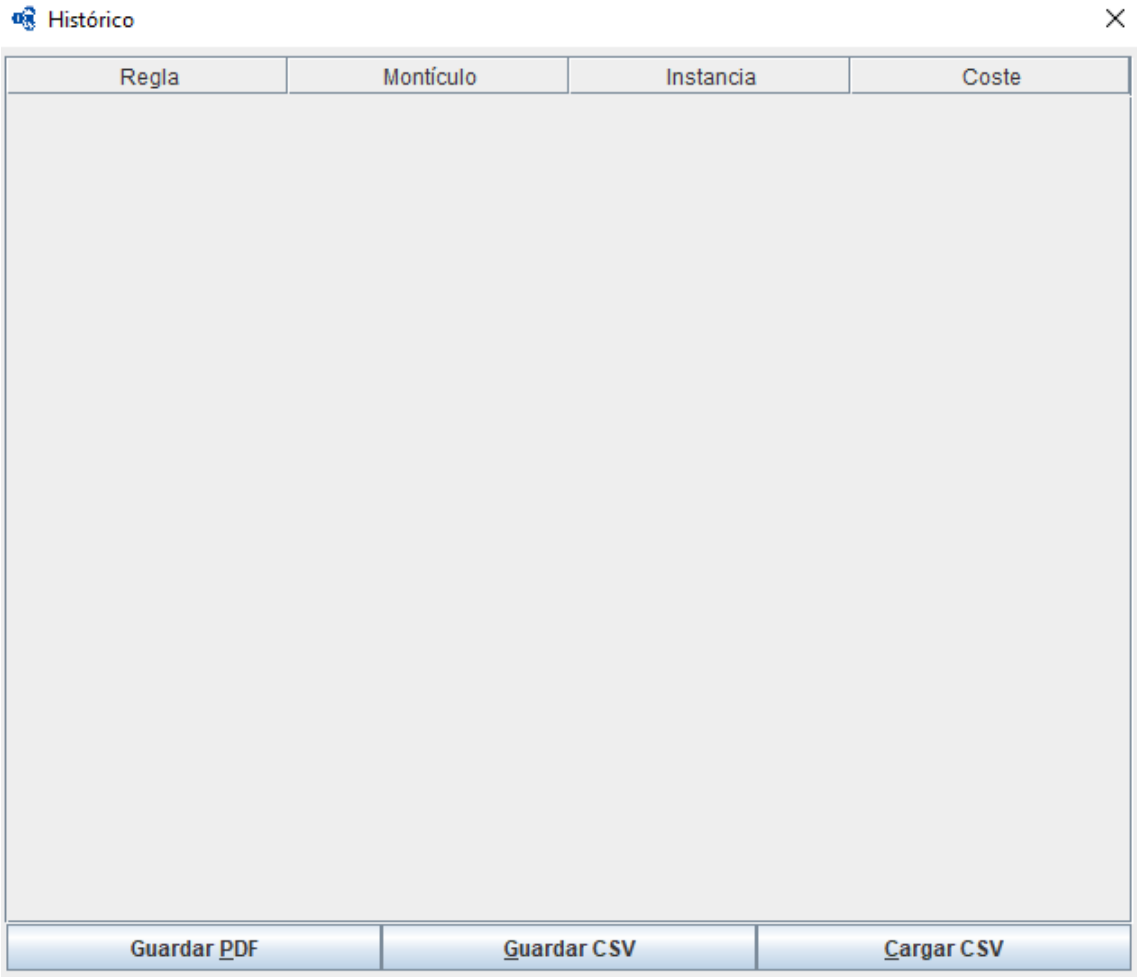

Figura 45. Histórico de la aplicación

3.7 Especificación del plan de pruebas

3.7.1 Pruebas unitarias

Este tipo de pruebas consiste en probar de forma independiente componentes o funcionalidad del sistema en la medida en que esto sea posible. En este proyecto, teniendo en cuenta que la mayoría de la lógica de negocio viene suministrada por el propio usuario (archivo “.jar” que dictamina como han de ejecutarse las reglas), se han utilizado pruebas para los componentes que se encuentran en el paquete “logic” descrito en el apartado 3.3.1 Descripción de los subsistemas

3.7.2 Pruebas de sistema

Este tipo de pruebas son pruebas de caja negra. Esto significa que se prueba únicamente el comportamiento del sistema, sin tener en cuenta su estructura interna, basándose en que el comportamiento sea acorde a lo que espera un usuario, considerando la especificación del sistema.

El motivo principal para realizar este tipo de pruebas es comprobar que el sistema cumple con los requisitos funcionales especificados y, por tanto, hace lo que se espera y no hace lo que no se espera.

Caso de uso I: Carga de un fichero de configuración		
ID	**Prueba**	**Resultado Esperado**
1.1	El usuario selecciona un fichero de configuración válido.	El sistema carga el contenido del fichero de configuración.
1.2	El usuario selecciona un fichero con la extensión correcta, pero contenido no válido.	El sistema no carga el contenido del fichero de configuración y queda en el estado inicial.
1.3	El usuario cancela la operación.	El sistema debe finalizar la operación sin guardar ningún dato y regresar al estado anterior.

Figura 46. Prueba de sistema caso de uso I

Caso de uso II: Representación de una regla (Campo Regla)		
ID	**Prueba**	**Resultado Esperado**
2.1	El usuario trata de representar reglas con los campos “Regla” y “Montículo” vacíos.	El sistema notifica al usuario indicando que los campos están vacíos.
2.2	El usuario introduce una regla no válida y procede a representarla.	El sistema notifica al usuario de que la regla introducida no es válida.
2.3	El usuario introduce una regla válida y procede a representarla.	El sistema procesa la regla y rellena el campo “Montículo”, así como muestra la representación gráfica de la regla.

Figura 47. Prueba de sistema caso de uso II

Caso de uso II: Representación de una regla (Campo Montículo)		
ID	**Prueba**	**Resultado Esperado**
3.1	El usuario trata de representar reglas con los campos “Regla” y “Montículo” vacíos.	El sistema notifica al usuario indicando que los campos están vacíos.
3.2	El usuario introduce una regla no válida y procede a representarla.	El sistema notifica al usuario de que la regla introducida no es válida.
3.3	El usuario introduce una regla válida y procede a representarla.	El sistema procesa la regla y rellena el campo “Montículo”, así como muestra la representación gráfica de la regla.
3.4	El usuario procede a eliminar el campo “Regla”, dejando el campo “Montículo”.	El sistema procesa la regla y rellena el campo faltante, así como muestra nuevamente la representación.
3.5	El usuario procede a cambiar el campo “Regla” no modificando el campo “Montículo”, siendo este último un campo válido.	El sistema procesa la regla desde el campo “Montículo”, sobrescribiendo el campo “Regla” con un valor correcto calculado a partir del montículo.

Figura 48. Prueba de sistema caso de uso III

Caso de uso IV: Almacenamiento de imagen		
ID	**Prueba**	**Resultado Esperado**
4.1	El usuario trata de guardar una imagen bajo la opción “Guardar Imagen” sin ningún valor en los campos “Regla” y “Montículo”.	El sistema lanza un error indicando que los campos necesarios para la generación de imagen están vacíos, por tanto, no se puede continuar la operación.
4.2	El usuario, una vez representada una regla válida, procede a guardar la imagen, seleccionando nombre y ruta donde almacenar el fichero.	El sistema guarda de forma satisfactoria el archivo “.pdf”, junto con la representación gráfica de la regla en el fichero.
4.3	El usuario, una vez representada una regla válida, proceder a guardar la imagen, pero cancela la operación.	El sistema vuelve al punto anterior a intentar guardar la imagen.

Figura 49. Prueba de sistema caso de uso IV

Caso de uso V: Almacenamiento de CSV		
ID	**Prueba**	**Resultado Esperado**
5.1	El usuario trata de guardar una regla bajo la opción “Guardar CSV” sin ningún valor en los campos “Regla” y “Montículo”.	El sistema lanza un error indicando que los campos necesarios para la generación de CSV están vacíos, por tanto, no se puede continuar la operación.
5.2	El usuario, con una regla no válida introducida, trata de guardar esta como un fichero con extensión “.csv”.	El sistema mostrará un error debido a que un campo necesario para la generación del fichero (montículo) está vacío.
5.3	El usuario, una vez representada una regla válida, procede a guardar la imagen, seleccionando nombre y ruta donde almacenar el fichero.	El sistema guarda de forma satisfactoria el archivo “.csv”, conteniendo el montículo y la regla como tal.
5.4	El usuario, una vez representada una regla válida, procede a ejecutar la regla y esta se procesa de forma satisfactoria. A continuación, procede a guardarla.	El sistema guarda de forma satisfactoria el archivo “.csv”, conteniendo el montículo y la regla como tal, además de los valores adicionales indicando la instancia y el resultado de esa regla para dicha instancia.
5.5	El usuario, con una regla válida representada y los campos necesarios rellenos, procede a guardar la regla, pero cancela la acción.	El sistema vuelve al punto anterior a intentar guardar el fichero con extensión “.csv”.

Figura 50. Prueba de sistema caso de uso V

Caso de uso VI: Ejecución de una regla		
ID	**Prueba**	**Resultado Esperado**
6.1	El usuario trata de ejecutar una regla sin ningún valor en los campos “Regla” y “Montículo”.	El sistema lanza un error indicando que los campos necesarios para la generación de ejecución de una regla están vacíos.
6.2	El usuario, con una regla inválida, con solo uno de los campos rellenos, trata de ejecutarla.	El sistema mostrará un error debido a que un campo necesario para la ejecución de una regla está vacío (no se pudo rellenar debido a que la regla no es válida).
6.3	El usuario, con una regla inválida pero una regla válida en uno de los campos trata de ejecutarla.	El sistema autocompleta el valor que contiene el valor inválido en función del campo válido, sobrescribiendo el primero, continuando con el proceso de ejecución.
6.4	El usuario, con una regla válida, decide cancelar la operación al momento de seleccionar un fichero “.jar” necesario para la ejecución.	El sistema notificará de que se ha cancelado la operación, así como el motivo al usuario.
6.5	El usuario, con una regla válida pero que contiene valores no definidos en el fichero de configuración, trata de ejecutarla.	El sistema notifica al usuario, indicando que hay valores introducidos en su regla que no existen en el fichero de configuración.
6.6	El usuario, con una regla válida, decide cancelar la operación al momento de elegir una instancia en la que ejecutar la regla.	El sistema notifica al usuario, indicando que no se ha seleccionado una instancia para aplicar la regla.
6.7	El usuario, con una regla válida, carga todos los archivos necesarios, cargando una instancia válida, pero la regla, a pesar de ser válida, no es aplicable para dicha instancia.	El sistema notifica al usuario indicando que ha habido algún tipo de problema a la hora de calcular el resultado numérico con los datos proporcionados.
6.8	El usuario, con una regla válida y aplicable a una instancia determinada, administra todos los ficheros necesarios para la ejecución de la regla en cuestión.	El sistema notifica al usuario del resultado numérico obtenido en la operación. A su vez, añade la operación al historial de operaciones y actualiza el campo “Resultado” en la interfaz gráfica.

Figura 51. Prueba de sistema caso de uso VI

Caso de uso VII: Guardar PDF desde historial		
ID	**Prueba**	**Resultado Esperado**
7.1	El usuario trata de generar archivos PDF con los elementos del historial, estando este vacío.	El sistema muestra un mensaje de error, indicando que el historial está vacío y no se puede realizar ninguna operación de almacenamiento.
7.2	El usuario trata de generar archivos PDF con los elementos del historial, habiendo elementos no válidos.	El sistema mostrará un mensaje de error para cada fila que se haya tratado de generar un PDF, indicando la fila de la regla no válida en cuestión.
7.3	El usuario trata de generar archivos PDF con elementos del historial, pero cancela la operación.	El sistema vuelve al estado anterior a tratar de guardar los ficheros PDF.

Figura 52. Prueba de sistema caso de uso VII

Caso de uso VIII: Guardar CSV desde historial		
ID	**Prueba**	**Resultado Esperado**
7.1	El usuario trata de generar un archivo CSV con los elementos del historial, estando este vacío.	El sistema muestra un mensaje de error, indicando que el historial está vacío y no se puede realizar ninguna operación de almacenamiento.
7.2	El usuario trata de generar un archivo CSV con los elementos del historial, habiendo elementos no válidos.	El sistema guardará la información línea a línea de los elementos que se encuentren en el historial, independientemente si estos son válidos o no.
7.3	El usuario trata de generar el archivo CSV con elementos del historial, pero cancela la operación.	El sistema vuelve al estado anterior a tratar de generar el fichero CSV con los elementos del historial.

Figura 53. Prueba de sistema caso de uso VIII

Caso de uso IX: Cargar CSV		
ID	**Prueba**	**Resultado Esperado**
7.1	El usuario trata de cargar un archivo que debería contener reglas, pero está vacío.	El sistema opera con normalidad, mostrando el historial vacío sin ningún cambio.
7.2	El usuario trata de cargar un archivo que contiene reglas, pero estas no son válidas.	El sistema opera con normalidad, mostrando los elementos en el historial, independientemente si estos son válidos o no.
7.3	El usuario trata de cargar un archivo con reglas, todas ellas válidas.	El sistema muestra el conjunto de reglas en el historial, independientemente si son válidas o no.

Figura 54. Prueba de sistema caso de uso IX

3.7.3 Pruebas de usabilidad

Mediante estas pruebas lo que se busca es el grado de satisfacción de los clientes con la aplicación, cuán fácil es de usar y conocer en qué medida se ajusta a sus necesidades. En el caso de este proyecto, los usuarios de la herramienta serán los miembros del grupo de investigación **iScOp***.*

Para ello, se dispone de este cuestionario inicial a rellenar por los usuarios finales de la aplicación

Cuestionario Inicial	
Edad del usuario	
Sexo del usuario	
Minusvalías	
Puesto de trabajo	
Estudios	
Experiencia en campo de la Informática	
Experiencia trabajando con Java	
Interés en la herramienta	

Figura 55. Cuestionario inicial pruebas usabilidad

Una vez completado el formulario, el usuario dispondrá de un margen de tiempo de 20 minutos para usar la herramienta, a fin de familiarizarse con la aplicación. Después de este tiempo, el usuario deberá completar los siguientes campos con una valoración numérica del 1 al 10, siendo la nota mínima y representando el mayor grado de disconformidad el 1, y siendo la nota máxima representando el mayor grado de satisfacción, el 10.

Valoración final	
La interfaz es intuitiva	
La interfaz es accesible	
El lenguaje de la interfaz es claro y conciso	
Las operaciones que se pueden realizar están bien definidas y diferenciadas	
Los elementos de la interfaz están bien distribuidos	
La ayuda es útil	
Los mensajes mostrados en pantalla son claros y concisos	
Utilidad	
Mejoraría el rendimiento en el trabajo	
Ayudaría a completar las tareas relacionas más rápidamente	
Facilitaría el manejo de información en el trabajo	
El sistema es, en general, útil en su trabajo	
Otros aspectos	
La aplicación funciona en un lapso aceptable	
Facilidad de uso	
Facilidad de aprendizaje	
El sistema es, en general, útil en su trabajo	

Figura 56. Cuestionario final sobre el sistema

Capítulo 4. Diseño del sistema

En este capítulo se detalla de la arquitectura del sistema, la estructura de las clases, y el diseño de las interfaces gráficas del sistema. A continuación, se exponen los elementos citados, además de concretar los resultados obtenidos en las pruebas realizadas en el sistema.

4.1 Arquitectura del sistema

4.1.1 Diagrama de paquetes

La figura 57 muestra la estructura final de los paquetes en los que se ha dividido la herramienta desarrollada. Cabe destacar que es una ilustración en la que no se tienen en cuenta las clases que se incluyen en cada paquete o las clases que pertenezcan a un paquete general que a su vez tenga subpaquetes.

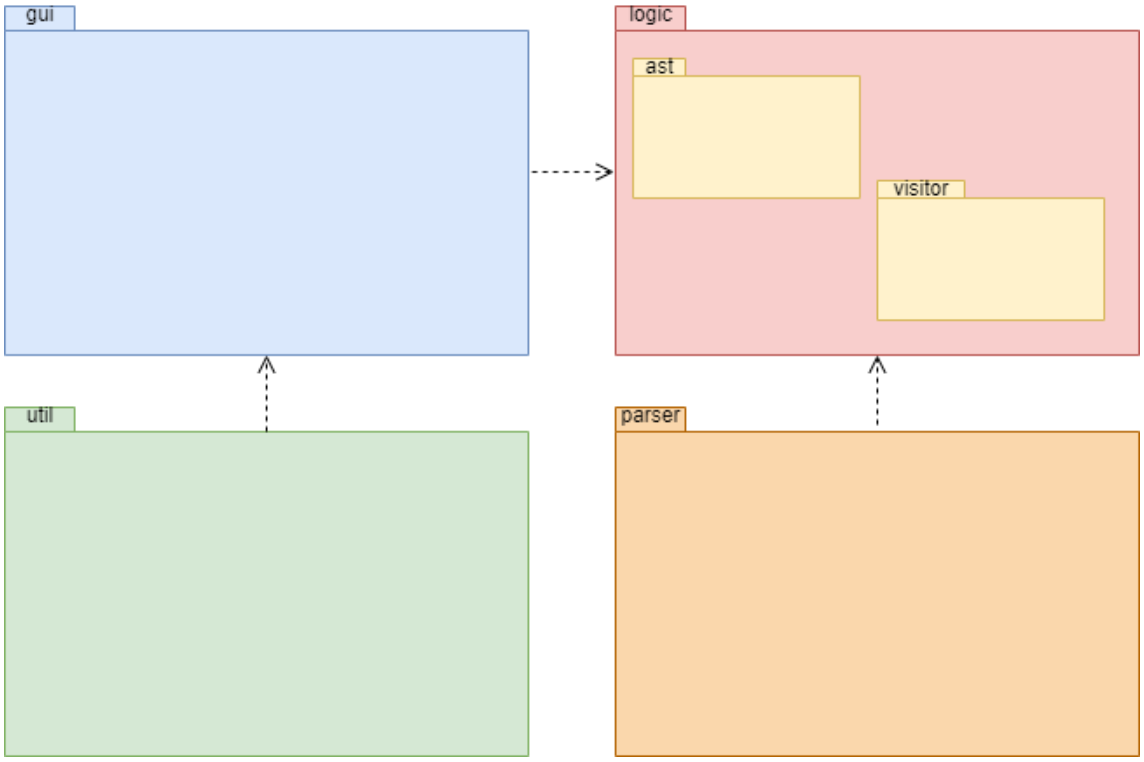

Figura 57. Modelo de caja negra de la estructura de paquetes del proyecto

El contenido de cada paquete se muestra a continuación:

- **gui**: Se trata del paquete en el que se encuentra la interfaz gráfica de la herramienta desarrollada. Contiene las clases que representan los componentes visuales, así como componentes que redefinen elementos visuales básicos del paquete Swing de Java. Este paquete a su vez llama a los elementos del paquete "logic" necesarios para aplicar la lógica de negocio y mostrarla en los elementos gráficos.
- **logic**: Se trata del paquete que contiene la lógica de negocio principal de la aplicación. A su vez, este paquete se divide en los siguientes subpaquetes
    - **ast**: Se trata de un paquete que contiene una única clase "MyExpression" que sirve para representar el AST (Abstract Syntax Tree) recogido por la herramienta Antlr.
    - **visitor**: Se trata de un paquete que contiene tanto una interfaz que determina la funcionalidad base que ha de tener un visitor [16], así como la implementación del visitor necesario para recoger los elementos obtenidos por Antlr y procesarlos como un montículo.
- **util**: Paquete que contiene clases relativas a la manipulación de archivos de entrada y salida, así como una clase que redefine el comportamiento de la entrada/salida por consola de Java
- **parser**: Paquete que contiene los elementos necesarios para el funcionamiento de la herramienta Antlr. Este paquete contiene clases autogeneradas por la herramienta, así como la gramática base sobre la cual se han generado estas últimas.

4.1.2 Diagrama de componentes

El diagrama de componentes de la aplicación es, en gran parte, similar al diagrama de paquetes, debido a que la estructuración en paquetes del proyecto se ha basado en los componentes descritos con anterioridad.

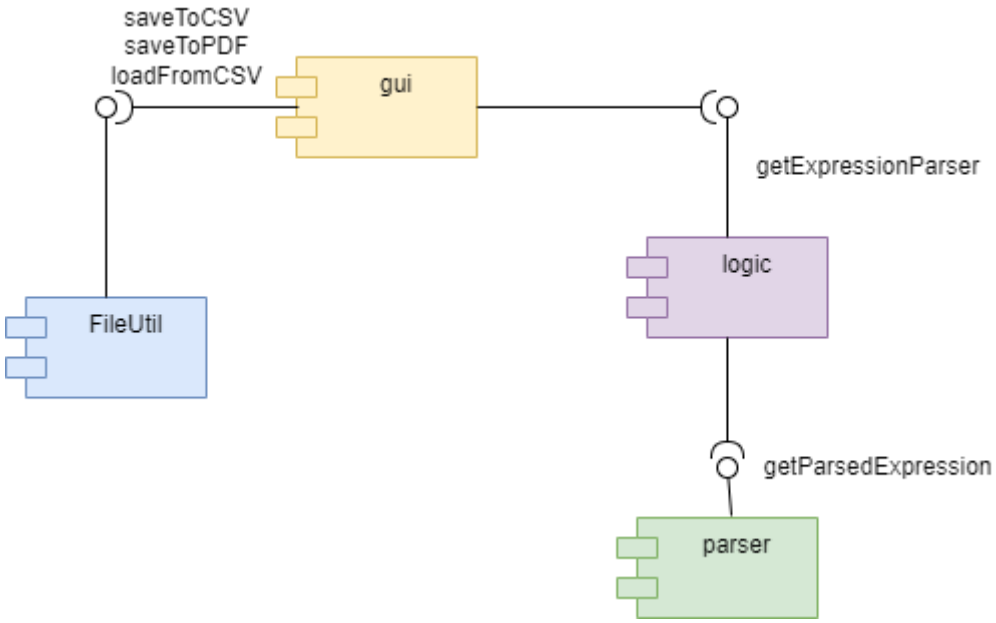

Figura 58. Diagrama de componentes de la aplicación

En la figura anterior, se puede ver la similitud que hay con el diagrama de paquetes de la aplicación. El componente representado como “FileUtil” no es otro que la clase del paquete “util” que se encarga de la carga/descarga de ficheros. En la imagen, también se aprecia la comunicación entre los componentes representados como “gui”, “logic” y “parser”, que tienen su análogo en forma de paquete.

4.1.3 Diagrama de despliegue

A continuación, se muestra el diagrama de despliegue de la herramienta, centrado en un entorno de sobremesa, independientemente de su sistema operativo. Esto se debe principalmente a que la herramienta está desarrollada con Java, lo que le permite ser ejecutada en cualquier plataforma siempre y cuando se cuente con el entorno Java para su ejecución

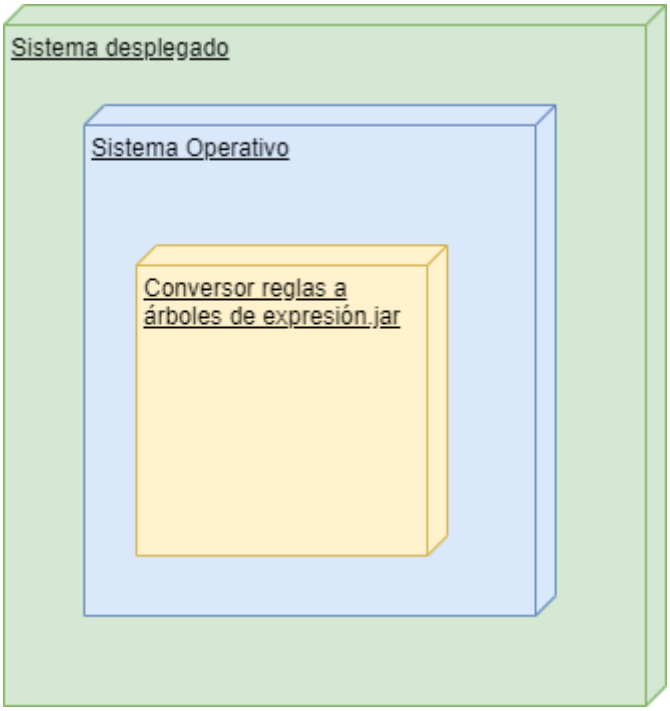

Figura 59. Diagrama de despliegue de la aplicación

4.1.4 Diagrama de interacción y estados

En este apartado se abordarán los diagramas de estados de los elementos más representativos de la aplicación, siendo estos la representación y la ejecución de reglas.

4.1.4.1 Diagrama de estados representación de una regla

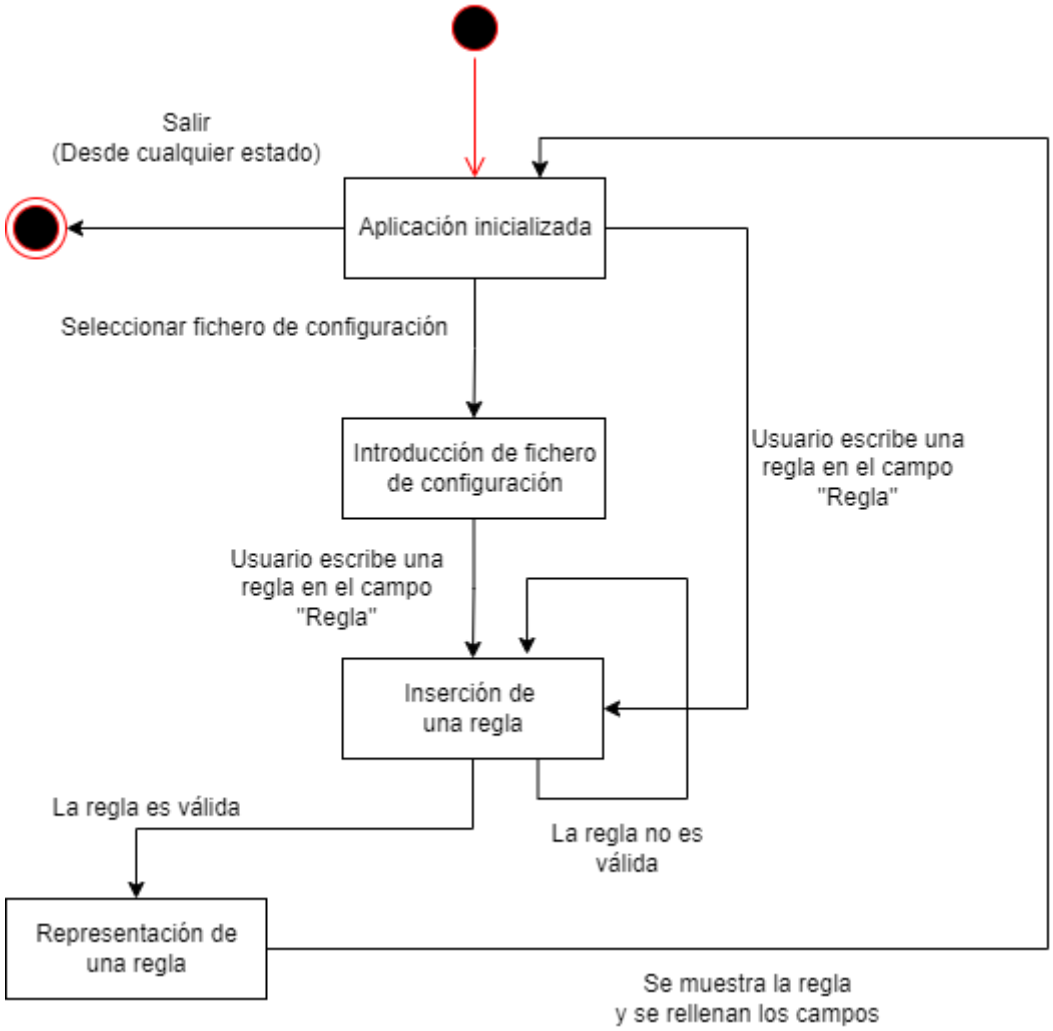

Figura 60. Diagrama de estados de representación de una regla

4.1.4.2 Diagrama de estados ejecución de una regla

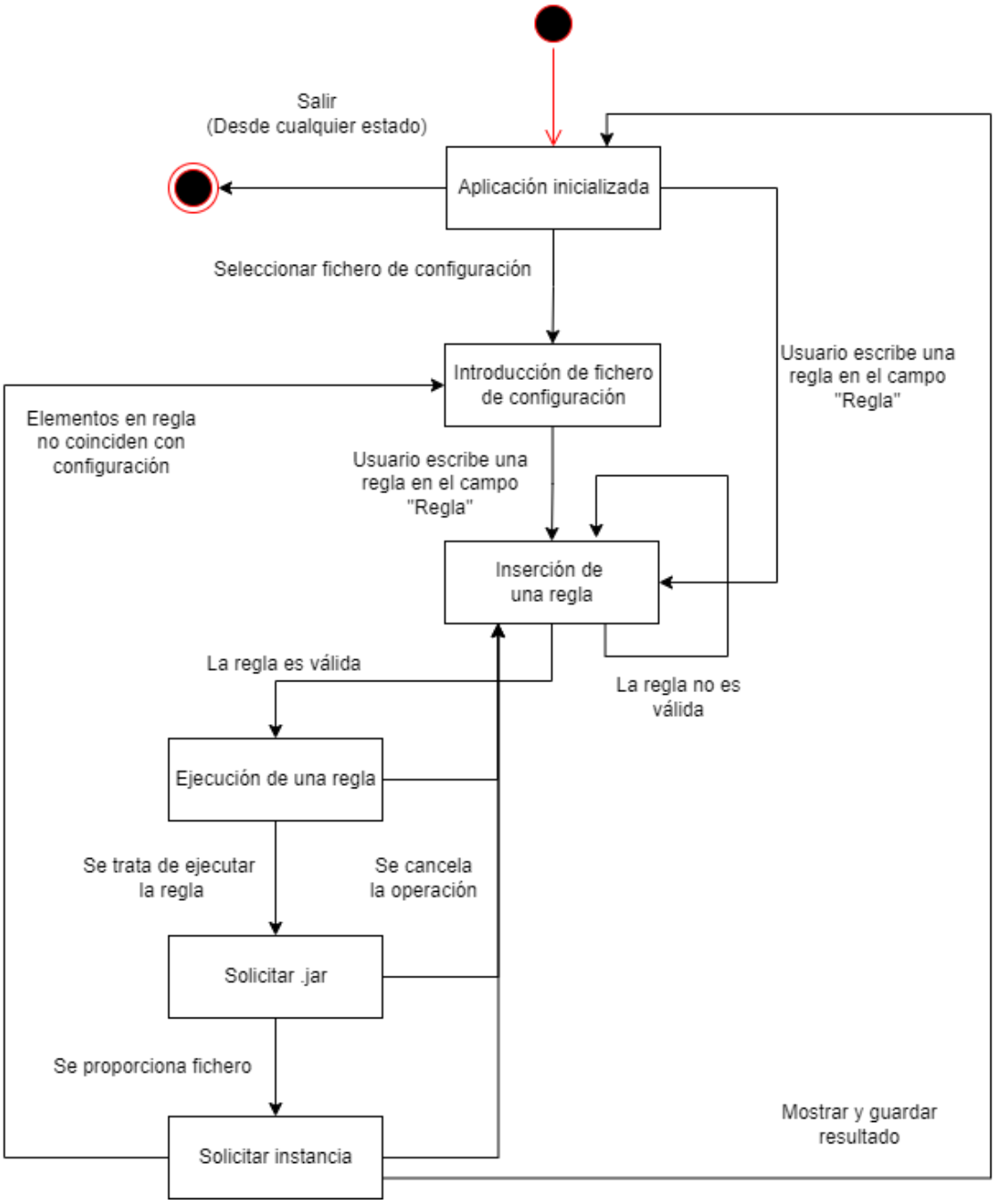

Figura 61. Diagrama de estados de ejecución de una regla

4.1.5 Diagrama de clases

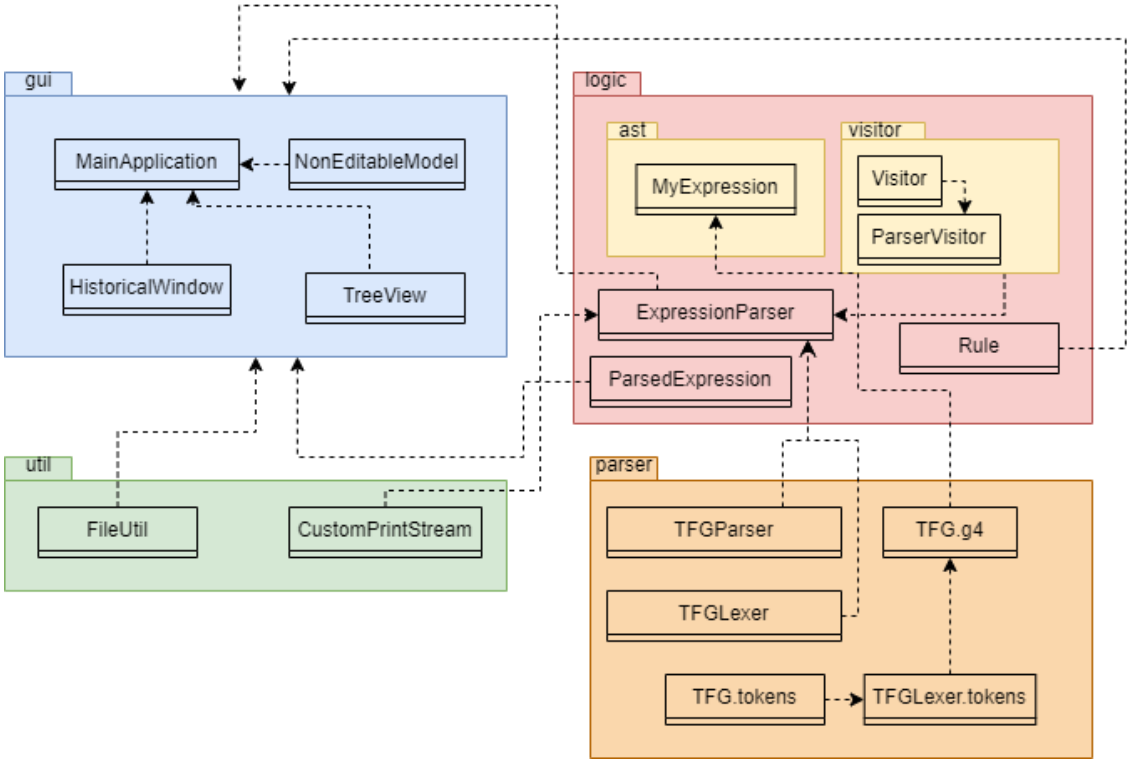

Figura 62. Diagrama de clases resumido del sistema

4.2 Diseño de la interfaz gráfica

En esta sección se especifica el diseño de las interfaces empleadas por la herramienta, que a su vez se han hecho en base a la funcionalidad descrita por los mismos.

4.2.1 Ventana principal

La ventana principal es la primera que se visualiza al iniciar la aplicación, y sobre la que recae la gran parte de funcionalidad de la misma.

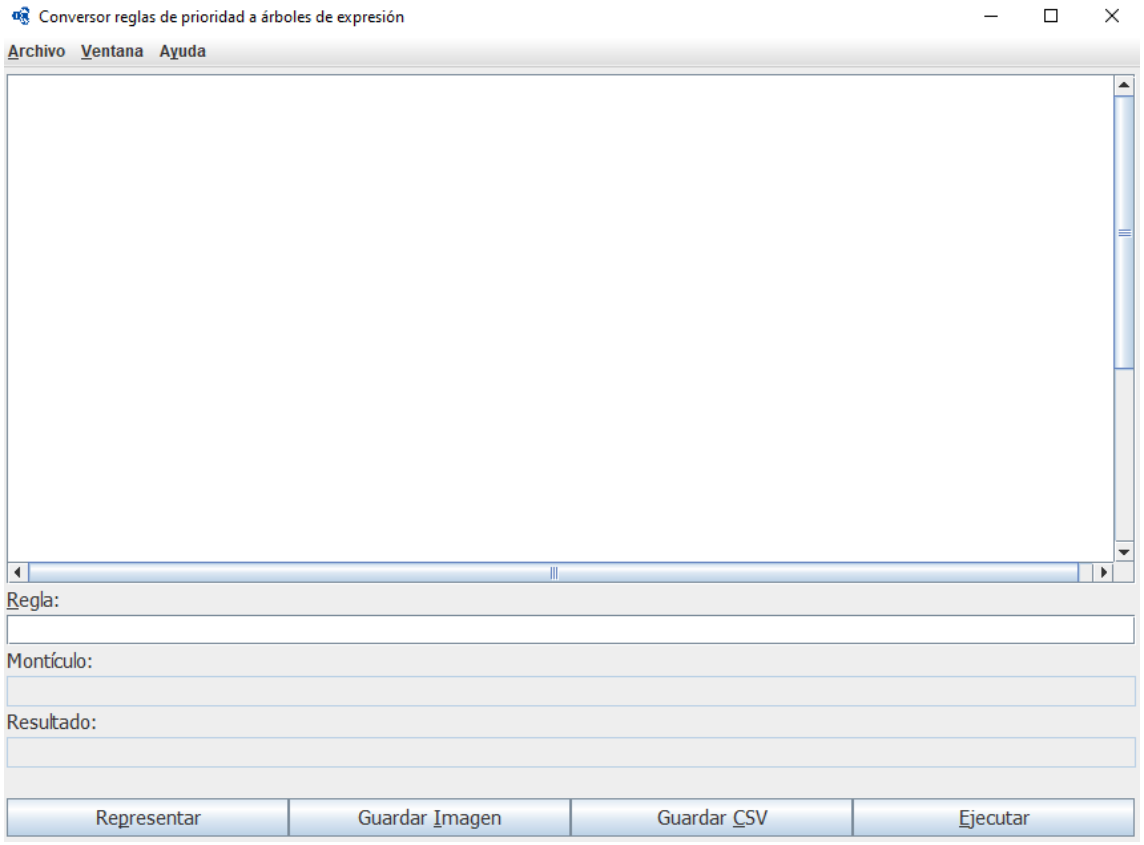

Figura 63. Ventana principal de la aplicación

En el panel de fondo blanco se encuentra la zona en la que se va a mostrar el árbol con la regla que se quiera representar en un momento dado. Cabe destacar que las dimensiones de este panel son las mismas que las de una hoja A4 en horizontal, debido a que la imagen a guardar se va a guardar en un fichero PDF con estas dimensiones, como se menciona en **RNF_4**. Debajo de este panel tenemos otro panel en el que se encuentran los campos de texto para “Regla”, “Montículo” y “Resultado”. El único campo editable en este caso será el primero citado, debido a que el resto de los campos se autogenera en base a este, a la hora de representar y ejecutarse, en caso de ser una regla válida. A su vez, tenemos el último panel en el que se encuentran todos los botones

que representan la funcionalidad de la ventana principal. Por último, tenemos en la parte del menú de arriba diferentes campos, entre los cuales se encuentra el primero, necesario para la correcta ejecución de las reglas y el segundo, donde se puede ver el historial de la aplicación.

4.2.2 Ventana historial

Esta ventana será únicamente accesible desde la ventana principal, al presionar la opción de menú bajo el campo de "Ventana" llamada "Mostrar histórico".

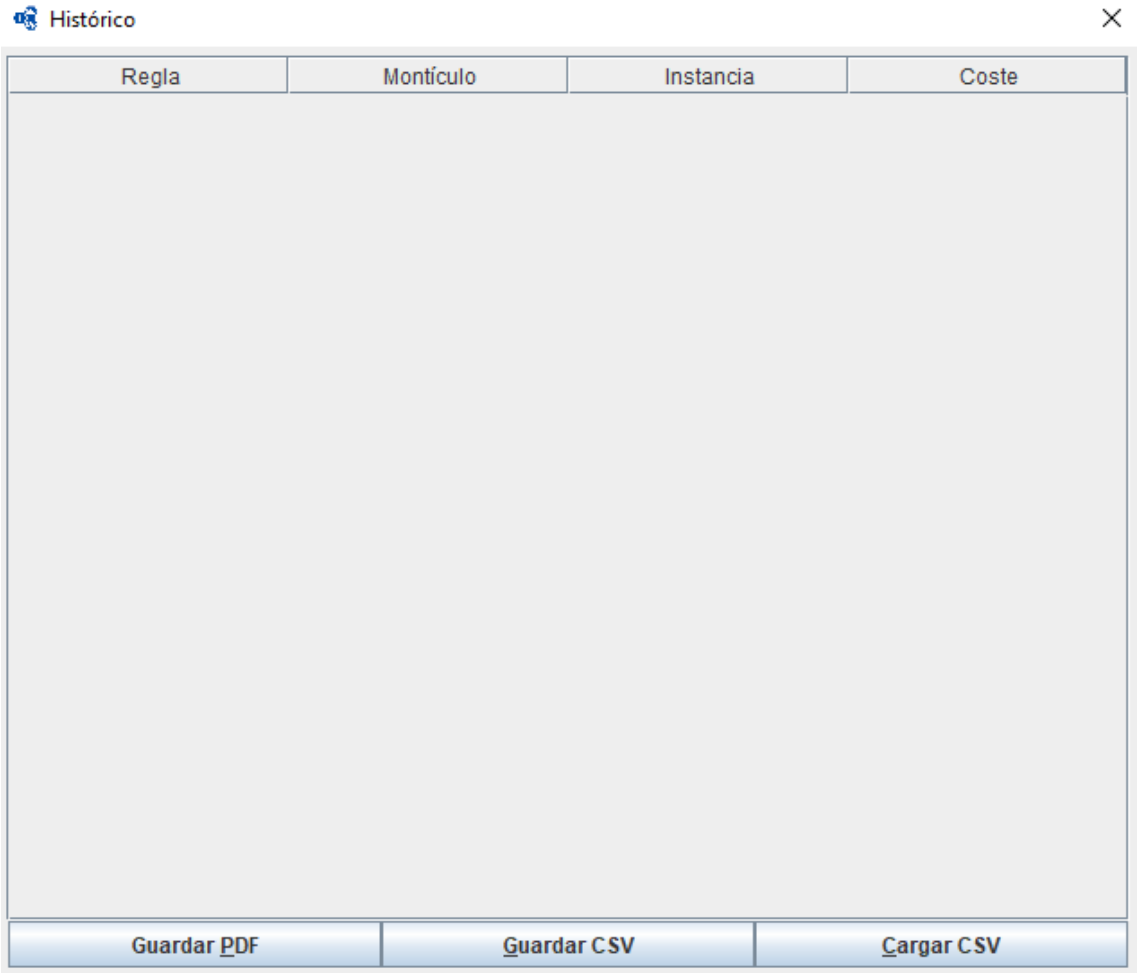

Figura 64. Ventana del historial de la aplicación

Esta ventana posee un panel principal en el cual se carga una tabla, inicializada sin elementos, que a su vez permite que sea desplazable hacia abajo, en caso de ser necesario (al igual que los scroll mostrados en la ventana principal). Por último, en la parte de abajo se encuentra un panel que contiene los botones mostrados en la imagen, cuya funcionalidad puede inferirse en base al nombre de los botones.

4.3 Especificación técnica de las pruebas

En este apartado se definirá en concreto los valores utilizados para las pruebas descritas en el apartado 3.7.2 Pruebas de sistema

4.3.1 Pruebas unitarias

Como se indica en la especificación del plan de pruebas, parte de la lógica viene dada por los ficheros suministrados por el usuario, de modo que las pruebas a realizar se componen de:

- Conversión de formato regla a montículo y viceversa.
- Carga, creación y almacenamiento de ficheros y su contenido.

Para la parte de conversión y carga, se ha tenido en cuenta los siguientes datos, pertenecientes al TSP:

- Regla: Con valor
    - "((Din*((Math.min(((Dc-Dcn)-Dcn),Dc))+(Math.min(((Math.min((Din-Dcn),Dc))-Dcn),Din))))-(Dc-(Dcn-(Math.min(Dcn,(Dc-(Math.max(0.6,0))))))))"
- Montículo: Con valor
    - "- * - Din + Dc - null null Math.min Math.min null null Dcn Math.min null null null null - Dc - Din null null null null null null Dcn - null null null null null null null null - Dcn null null Math.min Dcn null null null null null null null null null null null null null null null null Dc Math.max null null null null null null null null null null null null null null null null Dc Dcn null null null null null null - Dc null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null 0.6 0 null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null Din Dcn null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null null ";
- Fichero de configuración: test.conf. El contenido del mismo se encuentra en la imagen a continuación:

```
Problema del TSP
binary      +, -, /, *
binaryInverso   Math.max, Math.min
unary       Math.pow, Math.sqrt, -
terminals   Dc, Dcn, Din
constants   0, 0.1, 0.2, 0.6
```

Figura 65. Fichero test.conf

Para la parte de creación y almacenamiento de ficheros y contenido, no se parte de ningún dato inicial, puesto que son estos pasos los que generan ficheros.

4.3.1.1 Pruebas ExpressionParser

A continuación, se hará una descripción de las pruebas unitarias desarrolladas en función de los datos suministrados arriba.

Nombre del test / Datos	**Descripción de la prueba**	**Resultado esperado**
testParseFromRuleToHeap/ *Regla, Montículo*	Se crea un parser que convierta la entrada regla a una salida que coincida con montículo	Se comprueba que los resultados de los test retornen el mismo resultado.
testParseFromRuleToHeap/ *"9-", Montículo*	Se crea un parser que trate de convertir la entrada "9-" (inválida) a un montículo	Se comprueba que el resultado del test sigue siendo el valor de montículo, ya que conserva el último valor de una regla válida
testParseFromHeapToRule/ *Montículo, Regla*	Se crea un parser que trate de convertir la entrada montículo al elemento regla	No se realiza adecuadamente la conversión, debido a que falta la inserción de un fichero ".conf"
testParseFromHeapToRule/ *Montículo, Regla, test.conf*	Se provee del fichero ".conf" citado en la Figura 50 y se realiza de nuevo la prueba	Se comprueba que los resultados de los test devuelvan el mismo resultado

Figura 66. Test unitarios ExpressionParser

4.3.1.2 Pruebas FileUtil

Nombre del test / Datos	Descripción de la prueba	Resultado esperado
testSaveToCSV / *Regla, Montículo*	Se trata de guardar un archivo CSV que contenga los datos proporcionados. Se toma que el archivo no exista en un principio y que se cree un archivo después de la ejecución	Se comprueba que se genera un fichero con formato CSV con los datos introducidos.
testSaveToPDF / Imagen	Se trata de guardar un archivo PDF que contenga una imagen. Para facilitar la labor, en vez de obtener desde el esta prueba unitaria una imagen correspondiente a una regla de la aplicación, se usará un placeholder a modo de imagen	Se comprueba que antes de ejecutar la prueba, no exista el fichero PDF a generar. Después de la ejecución de la prueba, se comprueba que exista un fichero PDF.
testLoadFromCSV/ *Montículo, Regla*	Se llama a un fichero con extensión CSV con los contenidos de *Montículo y Regla*. Se comprueba que el contenido del fichero, una vez leído, corresponde con los valores de *Montículo y Regla*	Se comprueba que los resultados de los tests coinciden. En este caso, los resultados esperados coinciden con el contenido del fichero.

Figura 67. Test unitarios FileUtil

Una vez realizadas las pruebas y con la herramienta integrada EclEmma, obtenemos la siguiente cobertura de código, que nos indica la cantidad de líneas de código que se han llegado a ejecutar en las pruebas unitarias.

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
TFG	36,6 %	1.931	3.340	5.271
src/main/java	34,3 %	1.742	3.331	5.073
gui	0,0 %	0	2.780	2.780
parser	72,3 %	877	336	1.213
TFGParser.java	67,3 %	658	320	978
TFGLexer.java	93,2 %	219	16	235
util	65,7 %	318	166	484
FileUtil.java	75,8 %	307	98	405
CustomPrintStream.java	13,9 %	11	68	79
logic	89,4 %	415	49	464
ExpresionParser.java	90,0 %	296	33	329
Rule.java	82,9 %	58	12	70
ParsedExpression.java	93,8 %	61	4	65
logic.ast	100,0 %	49	0	49
MyExpression.java	100,0 %	49	0	49
logic.visitor	100,0 %	83	0	83
ParserVisitor.java	100,0 %	83	0	83
src/test/java	95,5 %	189	9	198
logic	95,5 %	189	9	198
FileUtilTest.java	95,5 %	126	6	132
AllTests.java	0,0 %	0	3	3
ExpresionParserTest.java	100,0 %	63	0	63

Figura 68. Cobertura de código del proyecto

Cabe mencionar que a pesar de tener una cobertura completa de un **36.6%**, la cobertura sobre el paquete en el que se han centrado las pruebas alcanza un **89.4%.** Estos porcentajes se deben a que solo se han realizado pruebas relacionadas con la lógica de la aplicación, así como los elementos auxiliares que utilizan las clases del paquete “logic” siendo estas clases las que se encuentran dentro del paquete “util”. Es por eso por lo que son los paquetes con más cobertura.

4.3.2 Pruebas de sistema

Las pruebas por realizar ya se han expuesto en el apartado 3.7.2 Pruebas de sistema

Estas se ejecutarán en la misma máquina en la que se ha desarrollado el proyecto. Las características de la máquina a ejecutar dichas pruebas son las siguientes:

- **Sistema Operativo:** Windows 10 Pro, Versión 10.0.19043.1766.
- **Java:** JDK 8.0.144.
- **RAM:** 8 GB.
- **Procesador:** i7-7700 3.6 GHz.

4.3.3 Pruebas de usabilidad

Los cuestionarios que se han descrito en el apartado 3.7.3 Pruebas de usabilidad

Dicho esto, los resultados de las pruebas han sido los siguientes:

Cuestionario Inicial	
Edad del usuario	28
Género del usuario	Varón
Minusvalías	No
Puesto de trabajo	Analista de sistemas
Estudios	Doctor
Experiencia en campo de la Informática	Graduado en ingeniería informática
Experiencia trabajando con Java	Alto
Interés en la herramienta	Alto

Figura 69. Cuestionario inicial usuario 1

Cuestionario Inicial	
Edad del usuario	45
Género del usuario	Mujer
Minusvalías	No
Puesto de trabajo	Profesora Titular de Universidad
Estudios	Doctora por la Universidad de Oviedo
Experiencia en campo de la Informática	Ingeniera Informática
Experiencia trabajando con Java	Medio
Interés en la herramienta	Alto

Figura 70. Cuestionario inicial usuario 2

Valoración final	
La interfaz es intuitiva	10
La interfaz es accesible	10
El lenguaje de la interfaz es claro y conciso	10
Las operaciones que se pueden realizar están bien definidas y diferenciadas	10
Los elementos de la interfaz están bien distribuidos	10
La ayuda es útil	10
Los mensajes mostrados en pantalla son claros y concisos	10
Utilidad	
Mejoraría el rendimiento en el trabajo	10
Ayudaría a completar las tareas relacionas más rápidamente	10
Facilitaría el manejo de información en el trabajo	10
El sistema es, en general, útil en su trabajo	10
Otros aspectos	
La aplicación funciona en un lapso aceptable	10
Facilidad de uso	10
Facilidad de aprendizaje	10
El sistema es, en general, útil en su trabajo	10

Figura 71. Cuestionario final usuario 1

Valoración final	
La interfaz es intuitiva	10
La interfaz es accesible	10
El lenguaje de la interfaz es claro y conciso	10
Las operaciones que se pueden realizar están bien definidas y diferenciadas	10
Los elementos de la interfaz están bien distribuidos	10
La ayuda es útil	9
Los mensajes mostrados en pantalla son claros y concisos	10
Utilidad	
Mejoraría el rendimiento en el trabajo	10
Ayudaría a completar las tareas relacionas más rápidamente	10
Facilitaría el manejo de información en el trabajo	10
El sistema es, en general, útil en su trabajo	10
Otros aspectos	
La aplicación funciona en un lapso aceptable	10
Facilidad de uso	10
Facilidad de aprendizaje	10
El sistema es, en general, útil en su trabajo	10

Figura 72. Cuestionario final usuario 2

4.3.3.1 Problemas encontrados y sugerencias

En este apartado, como indica su nombre, se exponen los diferentes problemas que se han encontrado, así como diferentes sugerencias que se han tenido en cuenta de cara a mejorar la herramienta:

- Solo se pueden cargar instancias del TSP porque la extensión termina en ".tsp" y no en ."txt".

La solución que se ha aportado es poder incluir ambos tipos de extensión. En caso de querer saber cómo modificar la cantidad de extensiones permitidas, ver la sección 6.2 Manual de desarrollador

- Cada vez que se quiere ejecutar una regla, hay que navegar dos veces por el explorador de archivos ya que por defecto aparece en “Mis Documentos”.

Se ha cambiado de forma que, en caso de haberse interactuado con algún selector de ficheros y se haya escogido de forma satisfactoria alguno, se cambia la ruta de los siguientes selectores de ficheros, relacionados con la carga de archivos, a la ubicación donde se encontraba el último archivo cargado por el usuario.

Capítulo 5. Implementación del sistema

En este capítulo se hablará sobre las diferentes herramientas software utilizadas para elaborar la herramienta, así como los problemas más significativos encontrados durante el desarrollo, así como alguna alternativa que se ha planteado.

5.1 Herramientas usadas para el desarrollo

5.1.1 Eclipse Java

Eclipse es un entorno de desarrollo que dispone de un sistema de extensiones o plugins, que además permite codificar aplicaciones en diferentes lenguajes, a pesar de que el lenguaje más popular usado en este entorno es Java.

5.1.2 Git

Git es un sistema de control de versión, con el que se ha realizado el desarrollo completo de este proyecto. Los repositorios creados para el proyecto mediante el uso de Git han sido alojados en la plataforma GitHub.

5.1.3 Antlr

Antlr (ANother Tool for Language Recognition) es una herramienta que permite leer, procesar e incluso ejecutar texto con una estructura definida. Se ha utilizado en el proyecto a la hora de procesar las entradas del usuario, a partir de las cuales se genera un AST, el cual servirá para generar un montículo binario.

5.1.4 iText

iText es la librería más sonada para la visualización de PDF en Java. El core es libre y de código abierto y cuenta con funcionalidad que abarca desde la construcción de ficheros PDF desde Java hasta renderizado de ficheros PDF en una aplicación entre otras opciones. Se ha utilizado en el programa para la generación de forma programática de archivos PDF que contengan imágenes, siendo estas últimas representaciones visuales de las reglas de prioridad introducidas por el usuario.

5.1.4 EclEmma

Plugin gratuito disponible para el lenguaje de programación Java desde el entorno de desarrollo Eclipse que permite la obtención de cobertura de un proyecto ejecutando los tests unitarios de este, de forma rápida y sencilla, estando la opción disponible entre los distintos tipos de ejecución que ofrece el propio entorno de desarrollo de base.

5.2 Estudio de alternativas

A la hora de plantear la funcionalidad de procesar las reglas de un formato inorden a montículo y viceversa, fue necesario plantear diferentes alternativas en cuanto a que dirección tomar para abordar el problema. A continuación, se muestran algunas de las opciones barajadas pero que finalmente fueron descartadas en favor de utilizar Antlr.

5.2.1 Hacer parser manual

Al principio se planteó realizar un parser desde cero a mano utilizando el algoritmo "Shunting Yard" [17], pero este no era compatible debido a que el algoritmo solo contempla elementos escritos en notación infija (elementos inorden), mientras que, los elementos que pueden recibirse como parámetro de parte del usuario, a pesar de representarse como un montículo al igual que los elementos binarios normales, no siguen este tipo de notación, por tanto quedó totalmente descartado.

5.2.2 Utilizar herramientas para representación gráfica

Se planteó investigar si existe algún tipo de herramienta que permita representar elementos de forma gráfica de manera más sencilla en vez de hacerlo desde cero. Desafortunadamente, no se encontró nada que fuese de provecho para el proyecto en el tiempo invertido y, teniendo en cuenta que la representación gráfica por parte del usuario no era relevante para este proyecto, se ha descartado el seguir investigando herramientas que desempeñen esta labor.

5.3 Problemas en el desarrollo

Durante el desarrollo de la herramienta se han encontrado diferentes problemas a abordar. En este apartado se hablará de forma somera de los más relevantes.

Los problemas vienen, sobre todo, relacionados a la representación gráfica de una regla, debido a que esta puede llegar a ser superior al margen de la pantalla donde se va a representar la imagen, al igual que puede ser superior a los márgenes establecidos en la aplicación para guardar en un PDF. Esto en parte se debe a la forma en la que se calculan las posiciones a la hora de pintar cada elemento de forma individual, que por cada nivel a representar la distancia entre elementos crece a gran velocidad. Se ha determinado que en la aplicación solo se van a representar reglas de hasta 8 niveles como máximo, por lo que los valores utilizados para el cálculo de las posiciones no deben ser ajustados.

Otro problema, también relacionado a la representación, es la separación entre elementos, debido a que, si un elemento ocupa mucho más que otro elemento en su mismo nivel, el primero puede llegar a solapar al segundo. Esto no es un problema en principio, debido a que los elementos individuales en cuestión no son muy grandes, por lo que permite margen de maniobra para la correcta representación de los elementos.

Capítulo 6. Manuales del sistema

6.1 Manual de ejecución

El proyecto se ha desarrollado y compilado en el lenguaje Java, de modo que se puede ejecutar en cualquier dispositivo que lo soporte. Hay distintas formas de ejecutar la herramienta.

- Ejecución directa (Doble click en el ejecutable).
- Línea de comandos.

Para ejecutar la aplicación directamente, basta con tener Java instalado en el equipo y hacer doble click sobre el ejecutable. En caso de no abrirse adecuadamente, también puede seleccionarse la opción de abrir con un programa externo, y seleccionando Java, tal y como se muestra en el ejemplo a continuación.

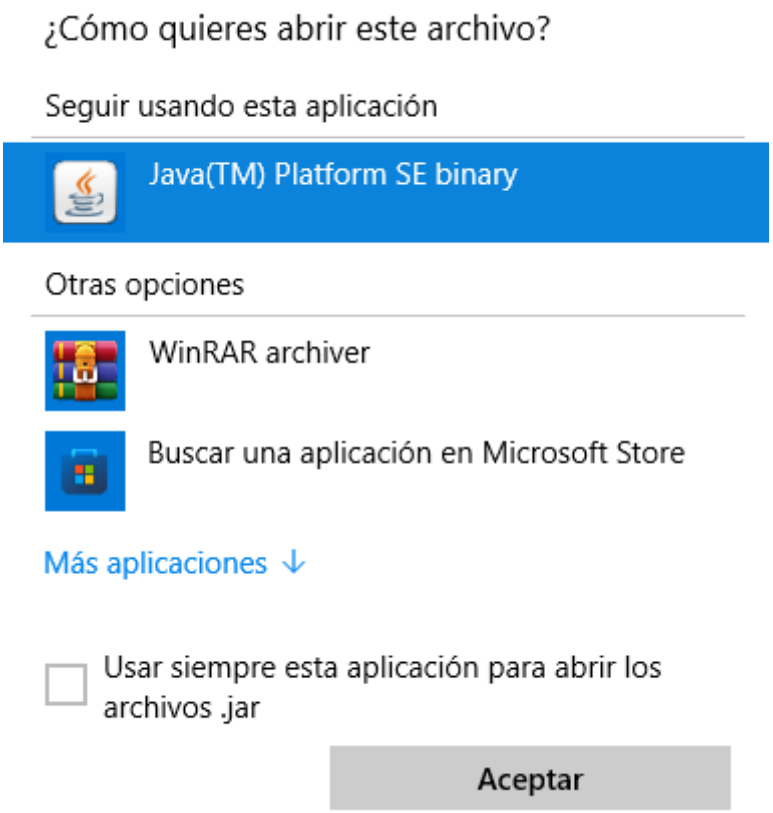

Figura 73. Abrir archivo con Java

Para su ejecución mediante línea de comandos, es necesario tener Java configurado en la variable CLASSPATH, o bien suministrar la ubicación donde se tiene instalado java e introducir lo siguiente:

java -jar conversor.jar

En caso de no tener la variable java definida en el CLASSPATH, cambiar el primer parámetro por la ubicación donde se tiene instalado Java. El último parámetro representa el nombre del ejecutable en cuestión. Este ejecutable también tendrá que encontrarse en el directorio desde el cual se esté utilizando la consola de comandos, o bien poner la ruta completa en la que se encuentra el archivo.

6.2 Manual de desarrollador

En este apartado se detallará el uso a nivel de desarrollador y los cambios pertinentes que hay que realizar en caso de que se quiera modificar la representación de las reglas de prioridad en el programa.

Para modificar la gramática con la que se generan los montículos, es necesario modificar el archivo “TFG.g4” que se encuentra en el paquete “parser”.

```
grammar TFG;

@header{

import java.util.*;
import logic.ast.*;

}

expression returns [MyExpression ast]:

            '('expression')' { $ast = $expression.ast; }
            | '-' ex1 = expression { $ast = new MyExpression("-", $ex1.ast, null); }
            | invFun {$ast = $invFun.ast; }
            | ex1 = expression o= ('^'|'**') ex2 = expression { $ast = new MyExpression($o.text, $ex1.ast, $ex2.ast); }
            | ex1 = expression o= ('*'|'/'|'%') ex2 =  expression { $ast = new MyExpression($o.text, $ex1.ast, $ex2.ast); }
            | ex1 = expression o= ('+'|'-') ex2 =  expression { $ast = new MyExpression($o.text, $ex1.ast, $ex2.ast); }
            | var = VARIABLE { $ast = new MyExpression($var.text, null, null); }
        ;


invFun returns [MyExpression ast]:
            var = VARIABLE '(' ex1 = expression ',' ex2 = expression')' { $ast = new MyExpression($var.text, $ex1.ast, $ex2.ast); }
          | var = VARIABLE '(' ex1 = expression')' { $ast = new MyExpression($var.text, $ex1.ast, null); }
        ;

fragment

INT: [0-9]+ ;

VARIABLE: (([a-zA-Z]|'_'|'.')+ ('()')? | (([a-zA-Z]|'_'|'.')+ ('['(([a-zA-Z]|'_'|'.') | INT)*']')?) | (INT ('.' INT)?))+ ;

WHITESPACE: [\r\n\t ] -> skip ;
```

Figura 74. Gramática de la aplicación

En caso de querer modificar el orden en el que se ejecutan las reglas, se han de cambiar de orden las líneas que se encuentran debajo de “expression”, poniendo en la parte superior la que se desee que tenga más preferencia. También en caso de querer cambiar la precedencia de 2 operadores que compartan la misma línea, se cambia el orden en el que aparecen dentro de la misma.

Una vez modificada la gramática, es necesario autogenerar el resto de clases que se encuentran en el paquete “parser”. Para ello, introducir (desde el directorio en el que se encuentra el proyecto):

java -jar lib/antlr-4.7-complete.jar src/main/java/parser/TFG.g4 -package parser -o src/main/java/parser -no-listener

El primer parámetro es la llamada a la librería de Antlr. El segundo parámetro es la ubicación del fichero que contiene la gramática a utilizar por la herramienta. El tercer parámetro indica el lugar donde se van a almacenar las clases autogeneradas, en caso de no existir dicho paquete. El cuarto parámetro es el nombre del paquete donde guardar las clases. Por último, el quinto parámetro indica que no se generen clases con el patrón "Visitor", puesto que se hará uso de clases propias para los fines del programa.

Para cambiar la representación gráfica, la clase a modificar es la que se encuentra en el paquete "gui" bajo el nombre "TreeView". Esta clase es la que se encarga de representarse en la interfaz gráfica.

Si se quiere ver a fondo que parámetros son en cuestión, se describen en el apartado 5.3 Problemas en el desarrollo

```
private HashMap<Integer, Rectangle> nodesPosition = null;
private HashMap<Integer, Dimension> subtreeSizes = null;
private static final int YOFFSET = 0, PARENT2CHILD = 20, CHILD2CHILD = 20;
private Dimension empty = new Dimension(0, 0);
private FontMetrics fm = null;
```

Figura 75. Parámetros modificables para representación gráfica

En concreto los parámetros a modificar son los encontrados en la tercera línea, representando cada uno lo siguiente

- **YOFFSET:** Determina la posición relativa al centro de una imagen desde el eje de ordenadas. Se recomienda dejar sin modificar.
- **PARENT2CHILD:** Determina la distancia entre un elemento padre y sus hijos. En caso de querer representar los elementos con menos separación disminuir el valor, en caso contrario aumentarlo.
- **CHILD2CHILD:** De forma análoga al anterior, determina la distancia entre hijos. En caso de querer representar los elementos con menos separación disminuir el valor, en caso contrario aumentarlo.

Si por otra parte se quieren modificar las extensiones para la carga de instancias utilizadas en la aplicación (actualmente trabaja con instancias con extensiones ".tsp" y ".txt"), la clase a modificar será la clase del paquete "gui" llamada "MainApplication", concretamente el método llamado "loadInstance()"

```
private File loadInstance() {

    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setDialogTitle("Cargar instancia");
    fileChooser.setMultiSelectionEnabled(false);
    fileChooser.setAcceptAllFileFilterUsed(false);
    fileChooser.setFileFilter(
            new FileNameExtensionFilter("Archivo .txt que represente una instancia (.tsp, .txt)", "txt", "tsp"));
```

Figura 76. Método loadInstance. Contiene los filtros utilizados para las instancias de la aplicación

Para añadir más filtros, simplemente añadir el nombre que tendrá la extensión a incluir, a partir del primer parámetro. Si se quiere también, se puede modificar este primer parámetro para indicar al usuario los formatos de fichero que acepta y hacer la interfaz más intuitiva.

6.3 Manual de usuario

La herramienta realizada es una aplicación que permite a los usuarios la modificación de reglas de prioridad, así como su representación y ejecución, además de guardar los resultados de las diferentes opciones que permite la aplicación en ficheros.

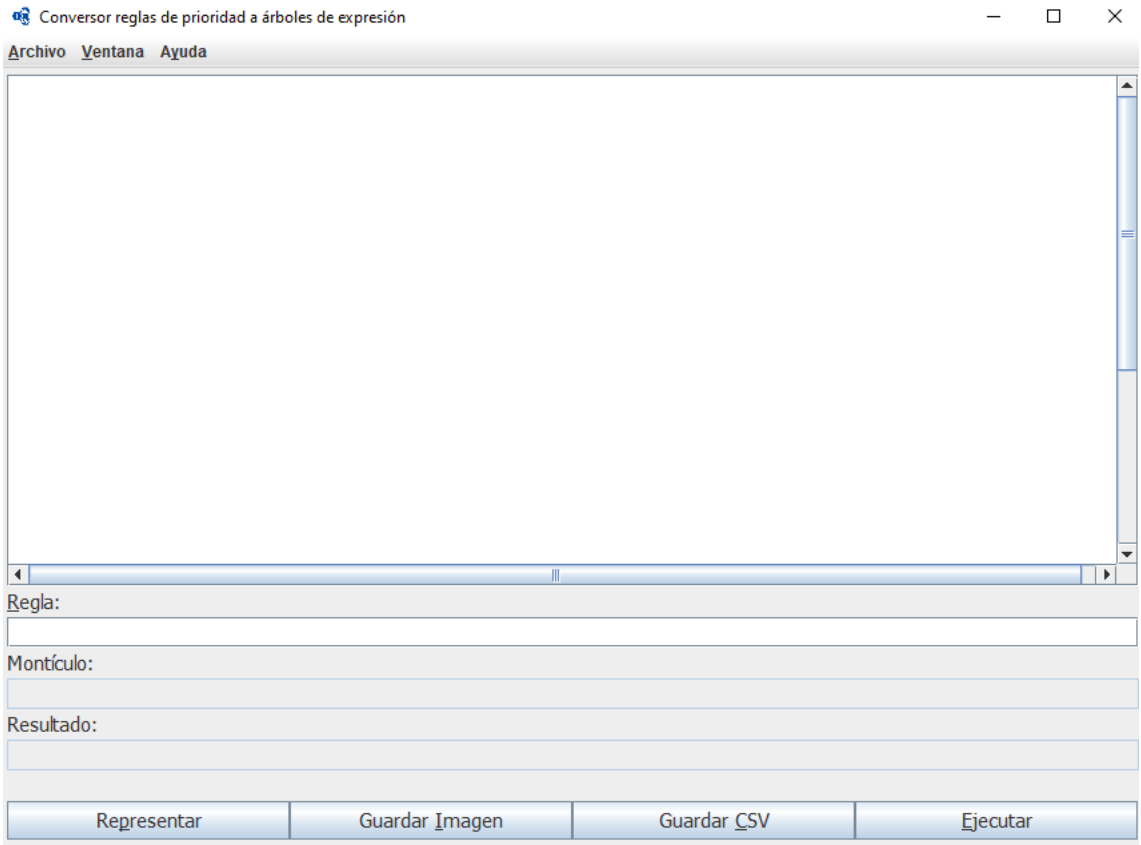

Figura 77. Ventana principal de la aplicación

La herramienta dispone de la siguiente funcionalidad:

- Representación gráfica de reglas de prioridad.
- Modificación de reglas de prioridad.
- Almacenamiento de la representación de las reglas de prioridad, en formato PDF.
- Almacenamiento de las reglas de prioridad y sus elementos, en formato CSV.
- Carga de reglas de prioridad desde CSV.
- Ejecución de reglas de prioridad.
- Visualización de elementos en un histórico.

La explicación sobre la funcionalidad de la aplicación también se encuentra integrada en la propia aplicación, a través de un sistema de ayuda, accesible desde la opción de menú mostrada en la parte superior, bajo el nombre de “Ayuda”.

6.3.1 Representación de reglas de prioridad

Para poder representar reglas de prioridad, lo primero es introducir la regla en el único campo disponible, siendo este el campo de “Regla”, mostrado en la imagen superior.

En caso de no introducirse ningún elemento y tratar de realizar una representación, el aviso a recibir será el siguiente.

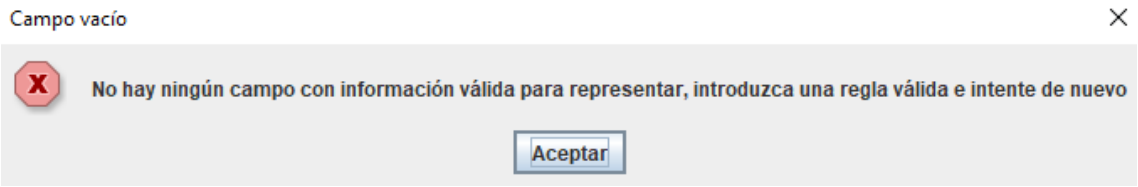

Figura 78. Error mostrado al no haber regla representable

En caso de introducir una regla que no se considere válida, entendiéndose válida a toda regla que siga la notación infija (al igual que las expresiones matemáticas) junto a funciones. Por ejemplo, la expresión matemática $(2 + 2)$ es válida, puesto que posee elementos terminales, en este caso los valores numéricos “2” y un operador que dictamina la operación a realizar, en este caso el operador “+”. Por otra parte, la expresión $(2+)$ no sería válida, ya que el operador “+” empleado precisa de dos elementos para poder ser considerada una expresión válida. Dicho esto, pongamos a prueba el ejemplo. Nos mostrará el siguiente error.

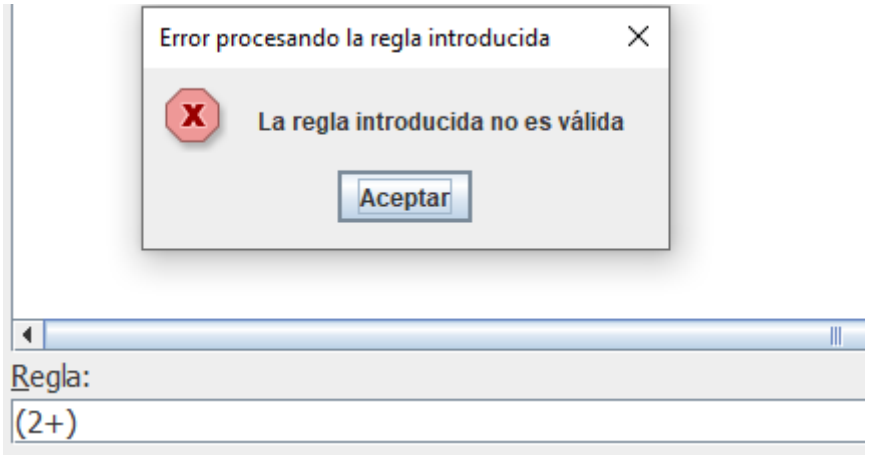

Figura 79. Error por insertar regla no válida

Sin embargo, al introducirse una regla válida, obtenemos los siguientes resultados.

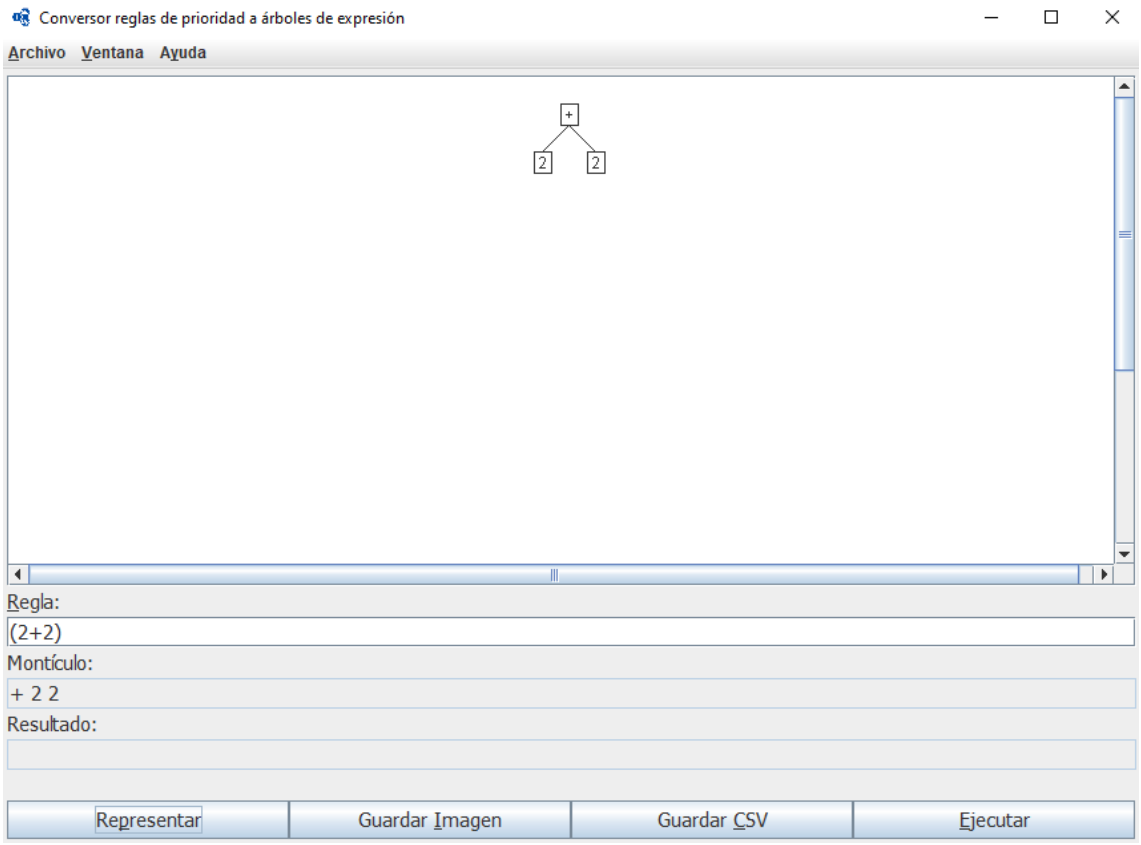

Figura 80. Representación gráfica de una regla

En la imagen se puede apreciar cómo se representa gráficamente la regla, además de autocompletarse el campo "Montículo", mostrando una estructura donde un elemento en una posición "x" tiene como hijos izquierdo y derecho respectivamente (en caso de existir) a los elementos en las posiciones: $[2 * x + 1,\ 2 * x + 2\ ]$ respectivamente, con $x = 0$ inicialmente.

No obstante, también se pueden representar funciones matemáticas en la herramienta. El único inconveniente que esto presenta es que, si no se ha cargado previamente un fichero de configuración, la representación de estas será la siguiente.

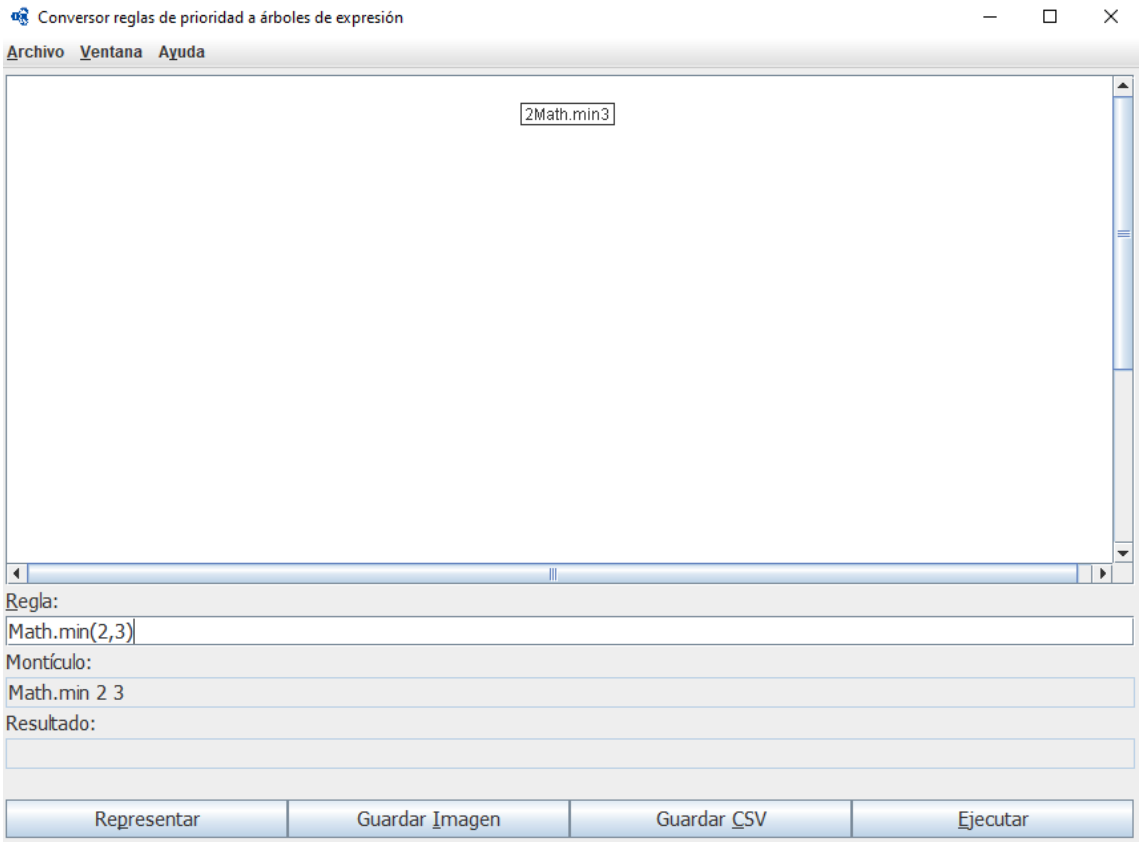

Figura 81. Representación errónea de regla con funciones

Para solventar esto, es necesario administrar un fichero de configuración que especifique los elementos que han de existir en la regla a la hora de ejecutarla (que no de representarla) indicando los elementos que no siguen la notación infija (véase las funciones que aceptan dos parámetros y las que solo aceptan un parámetro). Para el ejemplo mostrado, se introduce el siguiente fichero de configuración.

```
Problema de una máquina
binary       +, -, /, *
binaryInverso   Math.max, Math.min
unary        Math.pow, Math.sqrt, -
terminals    p, d, pmedia
constants    2, 3
```

Figura 82. Fichero configuración manual de usuario

Una vez añadido el fichero, vemos que la representación es la siguiente.

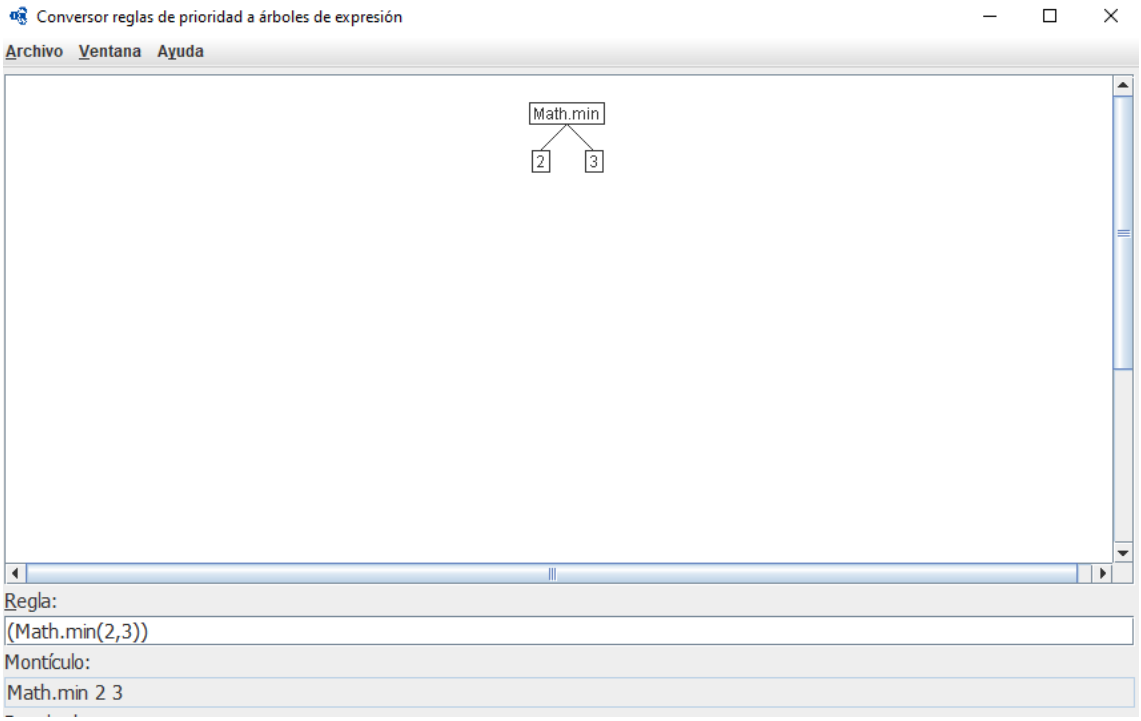

Figura 83. Representación correcta de regla con funciones

6.3.2 Modificación de reglas de prioridad

Partiendo de la regla de la funcionalidad anterior, para modificar una regla, tan solo hay que cambiar el elemento que se quiera sustituir por otro nuevo, cumpliendo siempre las especificaciones para hacer que la regla siga siendo válida. En caso de modificar la regla y hacer que esta no sea válida, en caso de haber una regla en el campo “Montículo”, esta sobrescribirá al valor a introducir en el campo “Regla”.

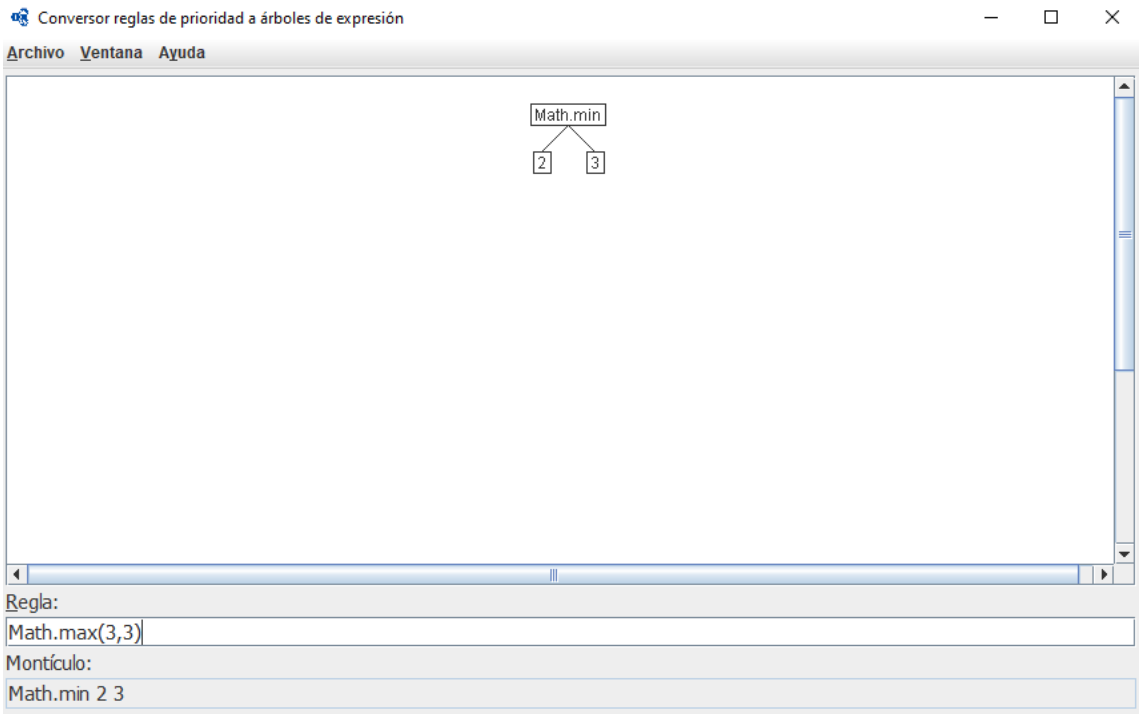

Figura 84. Regla válida antes de ser modificada

Como se puede apreciar, los campos en "Regla" y "Montículo" no coinciden, pero al ser el valor del primer campo válido, sobrescribirá al segundo campo y mostrará la representación correspondiente. Para ello, pulsamos la opción "Representar".

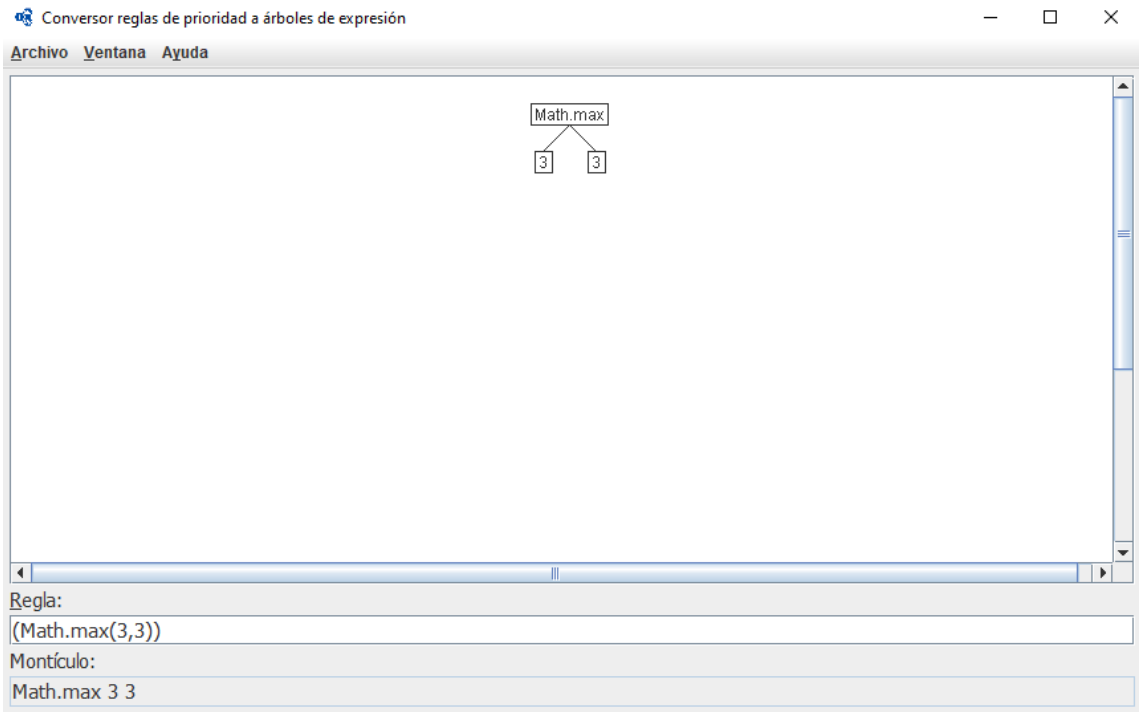

Figura 85. Representación de una regla modificada

Como se puede ver en la imagen, la regla ha cambiado, al igual que la representación y los valores de los campos "Regla" y "Montículo", que ahora coinciden.

6.3.3 Almacenamiento de representación de reglas, formato PDF

Una vez se tenga una representación válida en pantalla, se podrá guardar esta misma en un fichero PDF que contenga la representación. Para ello, hacer click en la opción "Guardar Imagen". Una vez hecho click en la misma, en caso de tener alguno de los campos vacíos debido a que el usuario lo ha borrado, o bien porque no se ha representado nada previamente, se mostrará el siguiente mensaje.

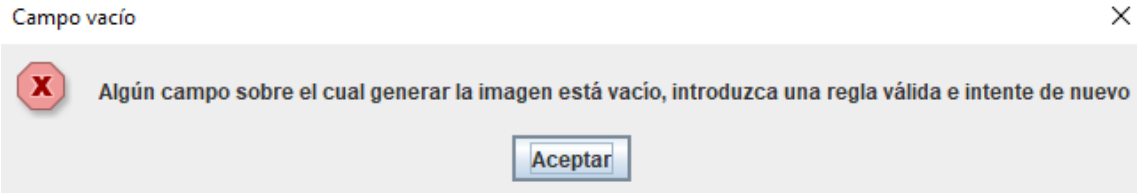

Figura 86. Error por campo vacío al almacenar

En caso de tener ambos campos con valores, se abrirá el siguiente menú en el cual se puede seleccionar la ubicación, así como el nombre del archivo a guardar.

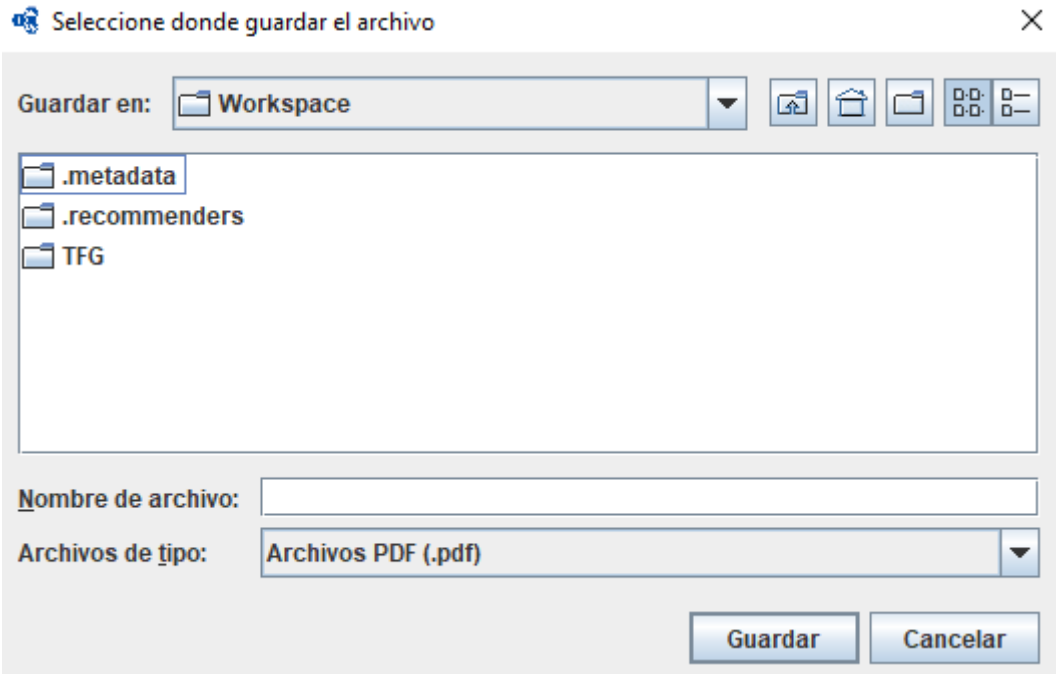

Figura 87. Menú donde guardar representación

Una vez seleccionado el lugar donde guardar la imagen, así como la ubicación, comprobamos el contenido del fichero generado. En este caso ilustrativo, se guardará en una carpeta llamada "Tutorial" bajo el nombre "test".

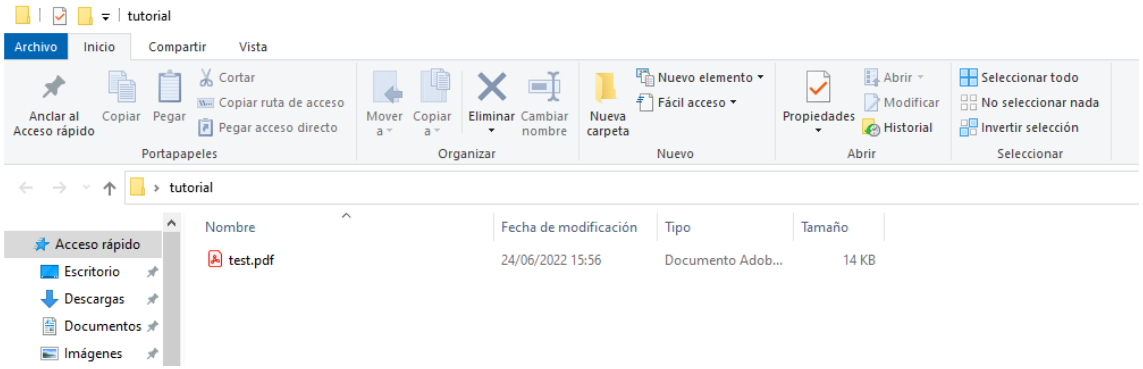

Figura 88. Almacenamiento de imagen en el sistema

Una vez abierto el fichero, vemos que el contenido del fichero es el siguiente:

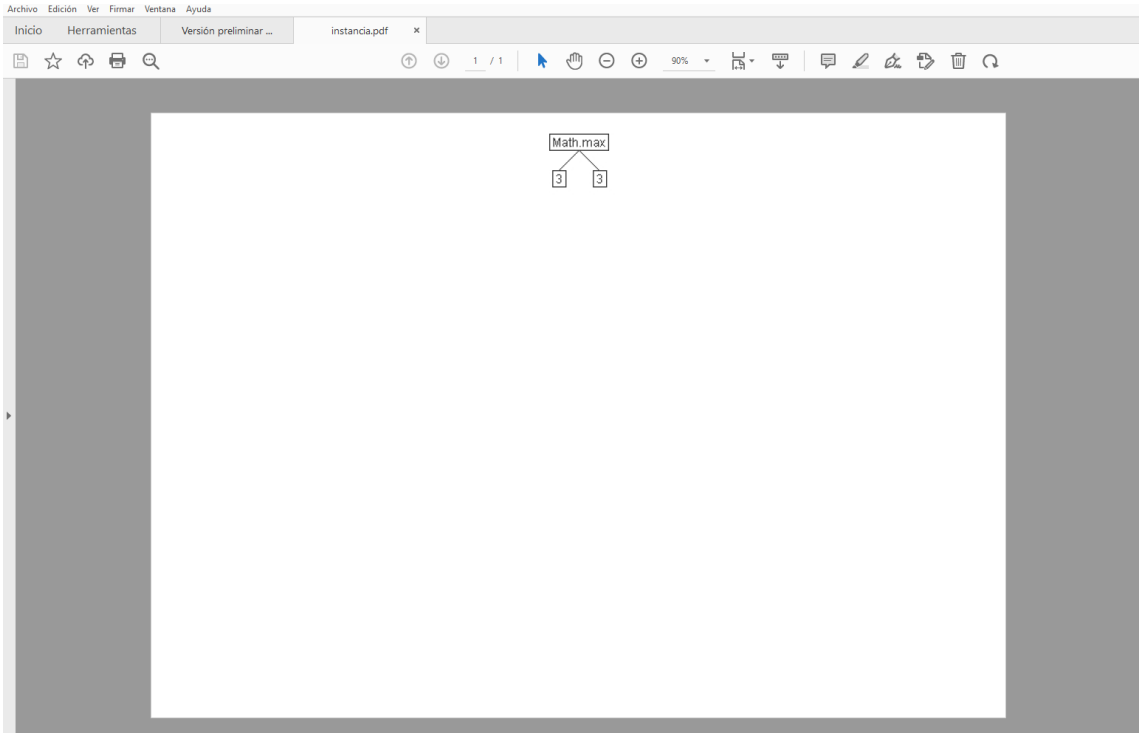

Figura 89. Representación en formato PDF

6.3.4 Almacenamiento de reglas de prioridad, formato CSV

Si se trata de guardar un CSV con ambos campos vacíos (nada más inicializar la aplicación) se mostrará el siguiente error

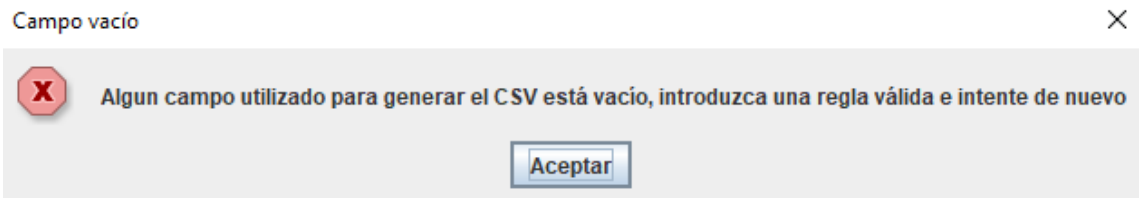

Figura 90. Error al generar CSV

En caso de tener una regla válida pero no representada previamente, al tener un campo vacío mostrará el mismo mensaje.

Una vez representada la regla, se podrá guardar mediante el botón “Guardar CSV”. Una vez presionado el botón, surgirá un menú parecido al de “Guardar Imagen” indicando lo siguiente.

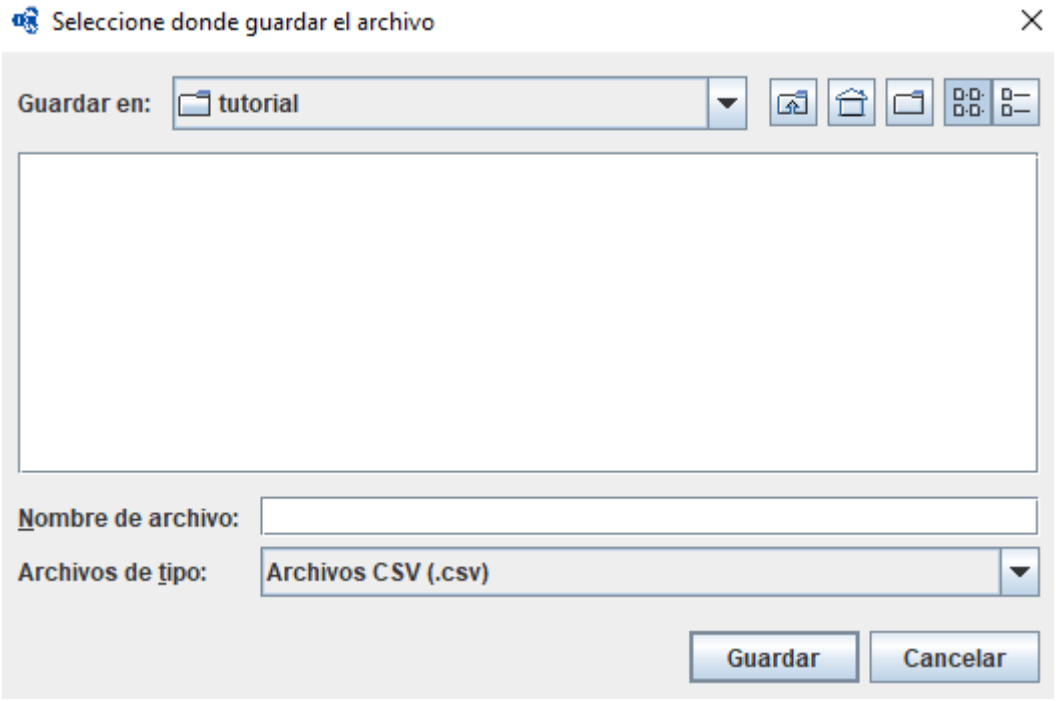

Figura 91. Menú para seleccionar donde guardar CSV

La ubicación donde se va a guardar la regla mostrada en los casos anteriores será un directorio llamado “tutorial” bajo el nombre test. Una vez guardado, procedemos a ver el archivo que se genera.

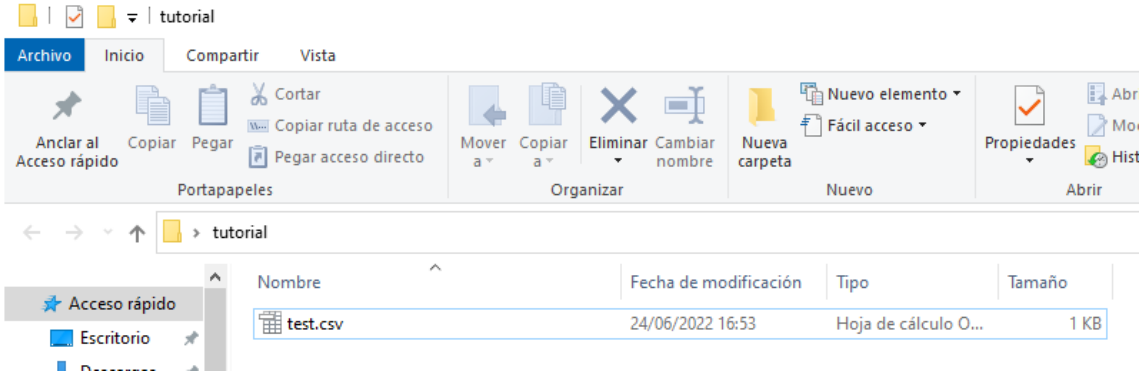

Figura 92. Archivo CSV generado.

Si abrimos el contenido del archivo, podemos ver que tiene el resultado esperado, bajo una cabecera.

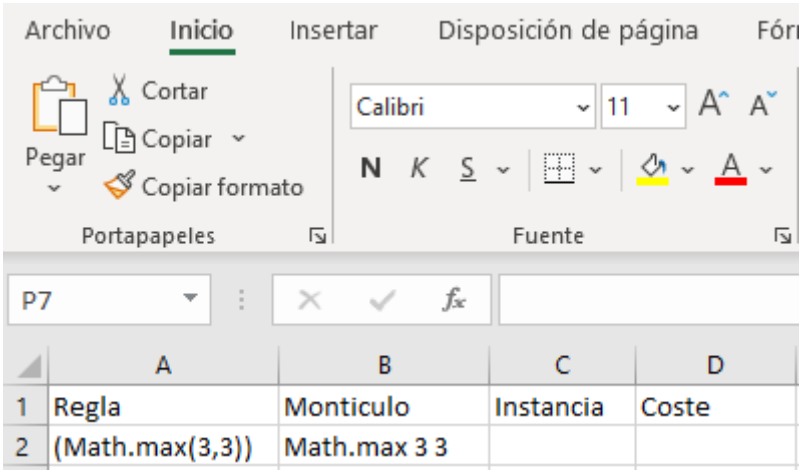

Figura 93. Visualización en Excel del CSV generado

6.3.5 Carga de reglas de prioridad desde CSV

Para poder cargar las reglas contenidas en un fichero CSV, hay que abrir el historial de la aplicación, bajo la opción de menú “Ventana”. Una vez seleccionada dicha opción, se desplegará un menú con un único elemento que permitirá mostrar el historial. Al hacer click en el elemento, obtenemos la siguiente vista.

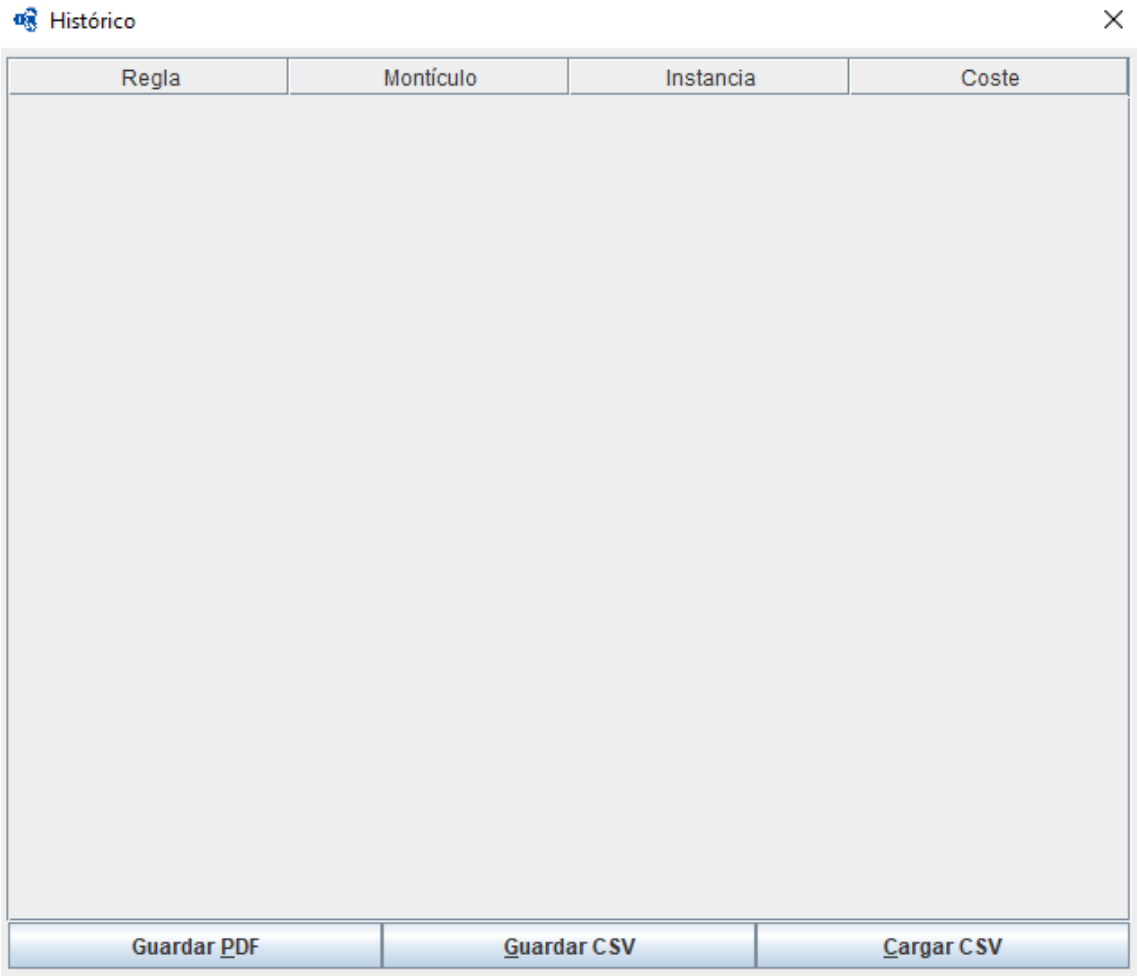

Figura 94. Vista del historial de la aplicación

Para cargar un fichero, damos click a la opción “Cargar CSV”. Al hacer click, saldrá una ventana que preguntará, de forma análoga a ejemplos vistos anteriormente, por un fichero con extensión válida. A diferencia de los anteriores menús, este necesita que se seleccione el archivo a cargar no siendo suficiente seleccionar el directorio donde se encuentra el fichero.

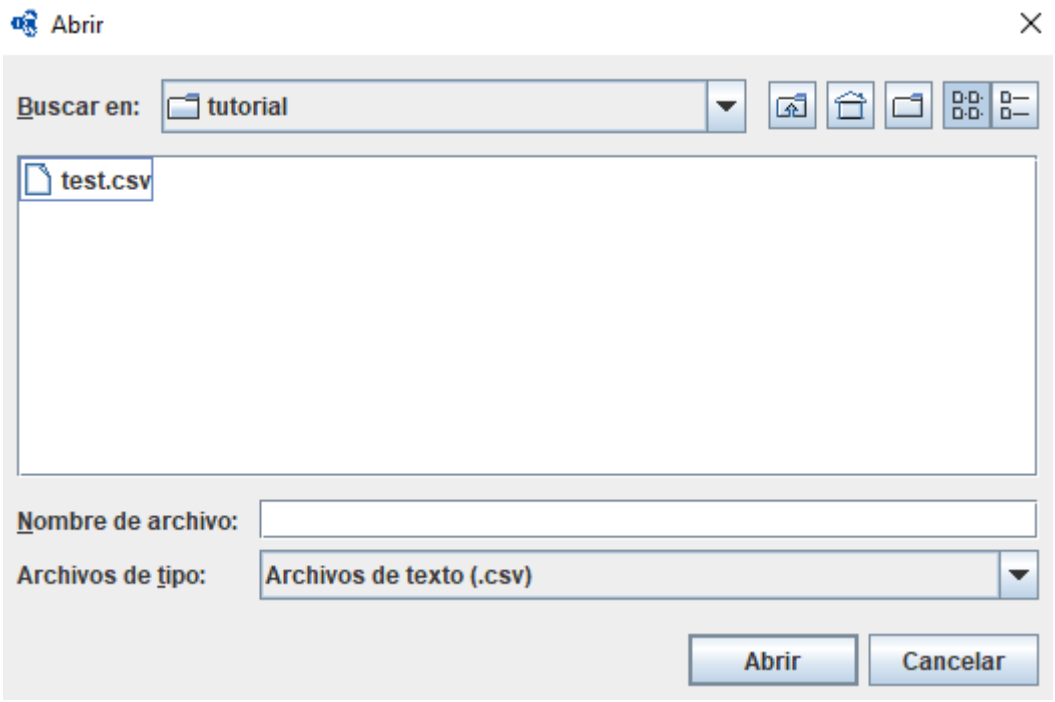

Figura 95. Menú mostrado al cargar CSV

Una vez seleccionado el fichero, damos a la opción abrir y vemos los cambios realizados.

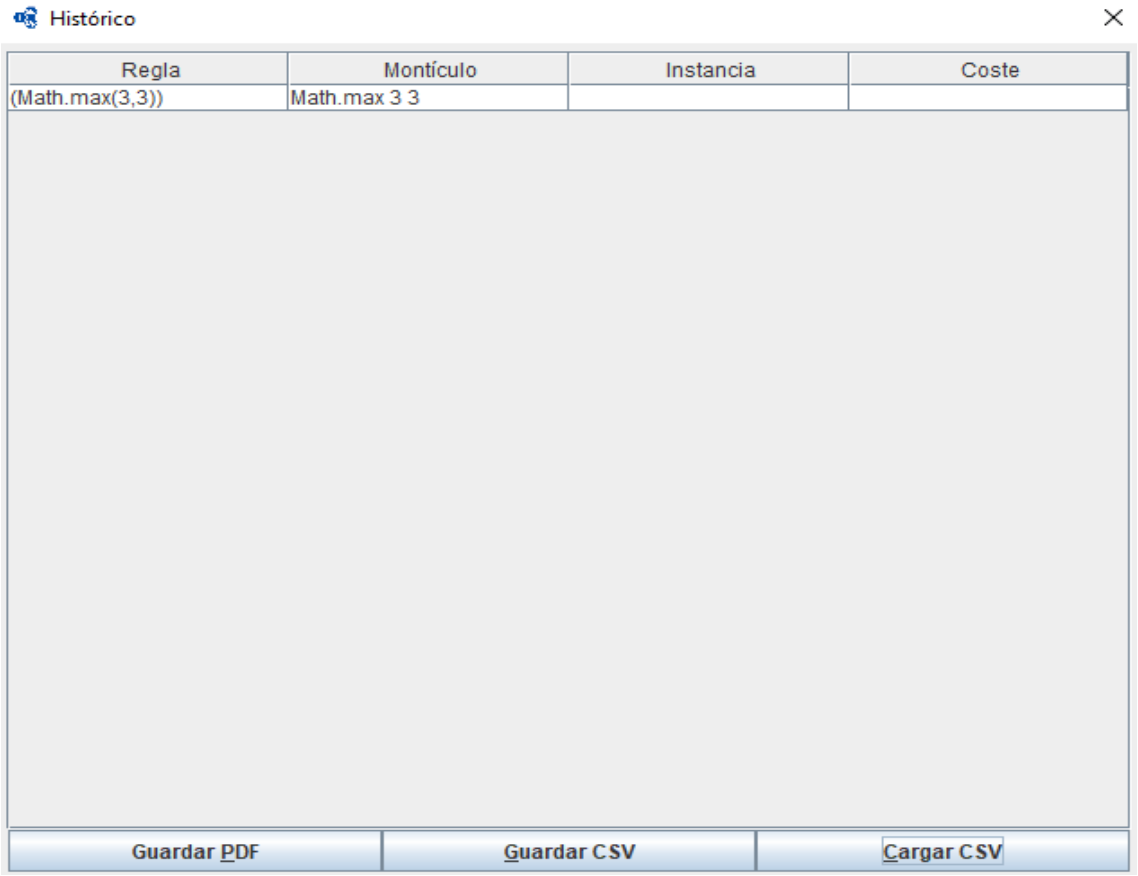

Figura 96. Historial de la aplicación tras cargar un CSV con reglas

Una vez haya elementos en el histórico, estos se pueden eliminar del mismo o bien modificar mediante los elementos de la pantalla principal. Para eliminarlo del histórico, se selecciona la fila que se desea eliminar. A continuación, se presiona la tecla “Supr” o “Del”.

Para proceder a la modificación, se selecciona la fila que se desee modificar y se hará doble click izquierdo. Una vez hecho esto, se podrá ver que los elementos “Regla”, “Montículo” y “Resultado” (en caso de existir), se rellenarán conforme a los valores de la fila del historial.

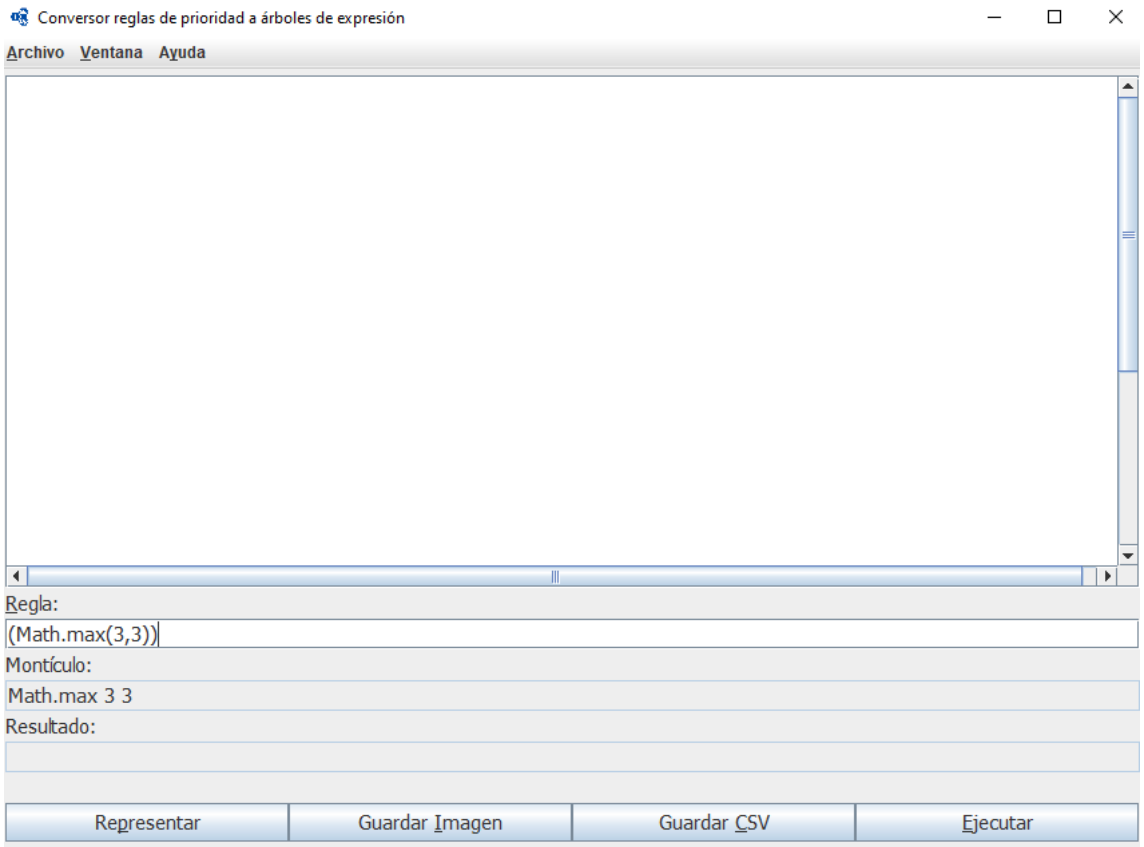

Figura 97. Ventana principal tras seleccionar regla del historial a modificar

A diferencia de una ejecución normal donde solo es posible rellenar el campo “Montículo” si se representa una regla, si se carga desde el historial y esta posee dicho campo, se rellenará sin proceder a la previa representación.

6.3.6 Ejecución de reglas de prioridad

Para ejecutar una regla, primero han de introducirse los campos necesarios para ello. Para este ejemplo se introducirá una regla y una instancia determinadas, siendo la regla “-instancia.d[tarea]”.

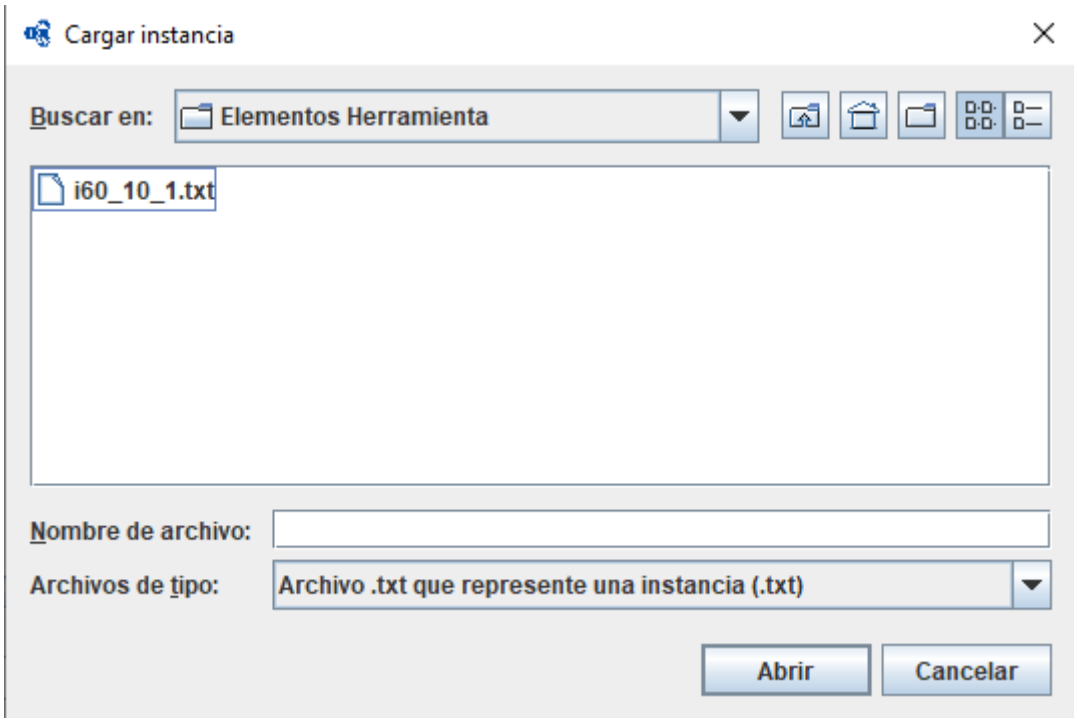

Figura 98. Ventana emergente mostrada al introducir instancia

En caso de no suministrarse ningún archivo, debido a que se cancele la operación, se mostrará el siguiente error.

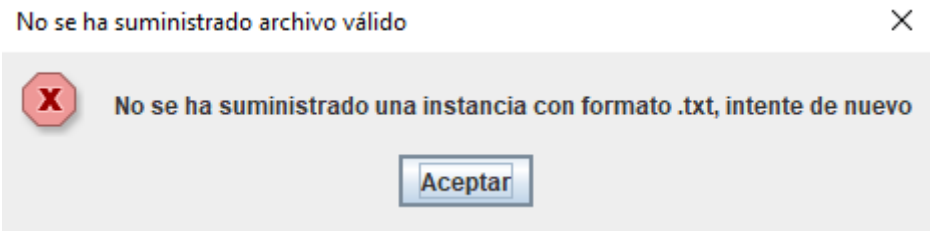

Figura 99. Error al no suministrar fichero de instancias

Una vez cargado, procedemos a cargar un fichero de configuración que incluya los elementos necesarios para la ejecución de la regla, quedando algo así:

```
Problema de una máquina
binaryInverso       Math.min, Math.max
unary   -
binary      +, -, /, *
terminals   p, d, pmedia, tarea, instancia
```

Figura 100. Contenido test.conf usado para ejemplo ilustrativo

En caso de que en la regla se intente utilizar algún elemento inexistente en el archivo donde se definen los elementos permitidos en la regla, el sistema indicará el siguiente error.

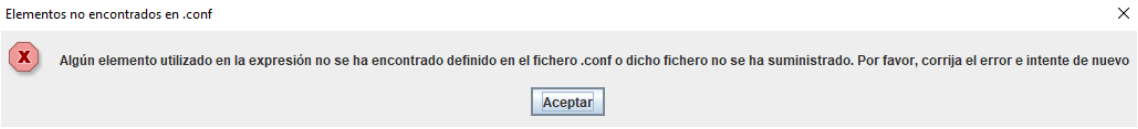

Figura 101. Error elementos no encontrados en configuración

También será posible ver el siguiente error, en caso de que se cancele la operación a la hora de pedir un archivo con extensión ".jar" y que este no se suministre.

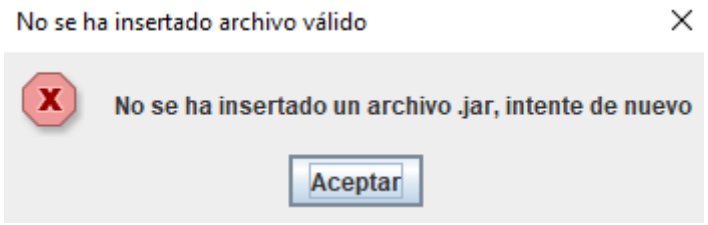

Figura 102. Error por no insertar archivo .jar

En caso de que se quiera introducir una regla inválida y se trate de ejecutar, también se mostrará un error, de forma análoga a como se mostraba a la hora de representar.

Figura 103. Error en representación de regla. Campo vacío

En caso de cancelar la operación a la hora de solicitar una instancia, el sistema mostrará el siguiente error.

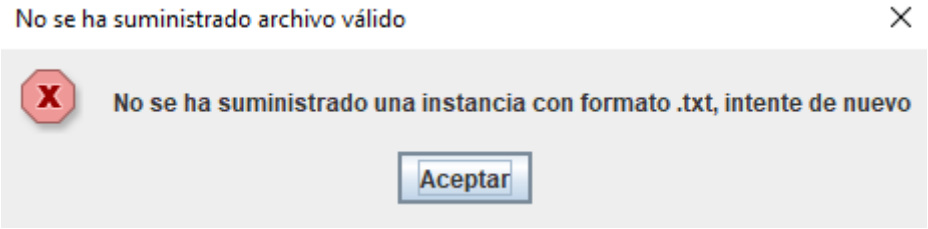

Figura 104. Error al suministrar instancia

Otro error que puede ocurrir es que la regla sea válida, se haya cargado una instancia y el usuario haya suministrado un archivo con extensión ".jar", pero a la hora de ejecutar la regla, debido a su configuración, no se pueda obtener un resultado, de forma que se obtiene el siguiente mensaje.

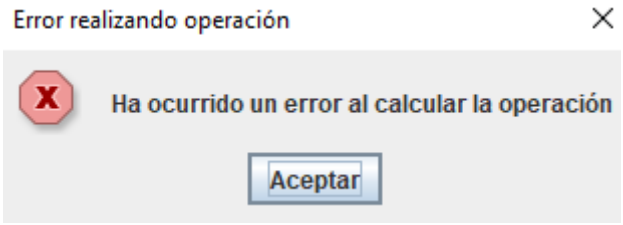

Figura 105. Error al calcular resultado a la hora de ejecutar instancia

En cualquier otro caso donde no ocurra ninguno de los errores anteriores, el sistema mostrará el resultado de la operación con los parámetros definidos por el usuario, reflejandose en la interfaz de la siguiente manera.

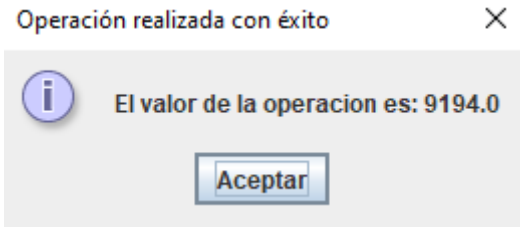

Figura 106. Ejecución realizada con éxito

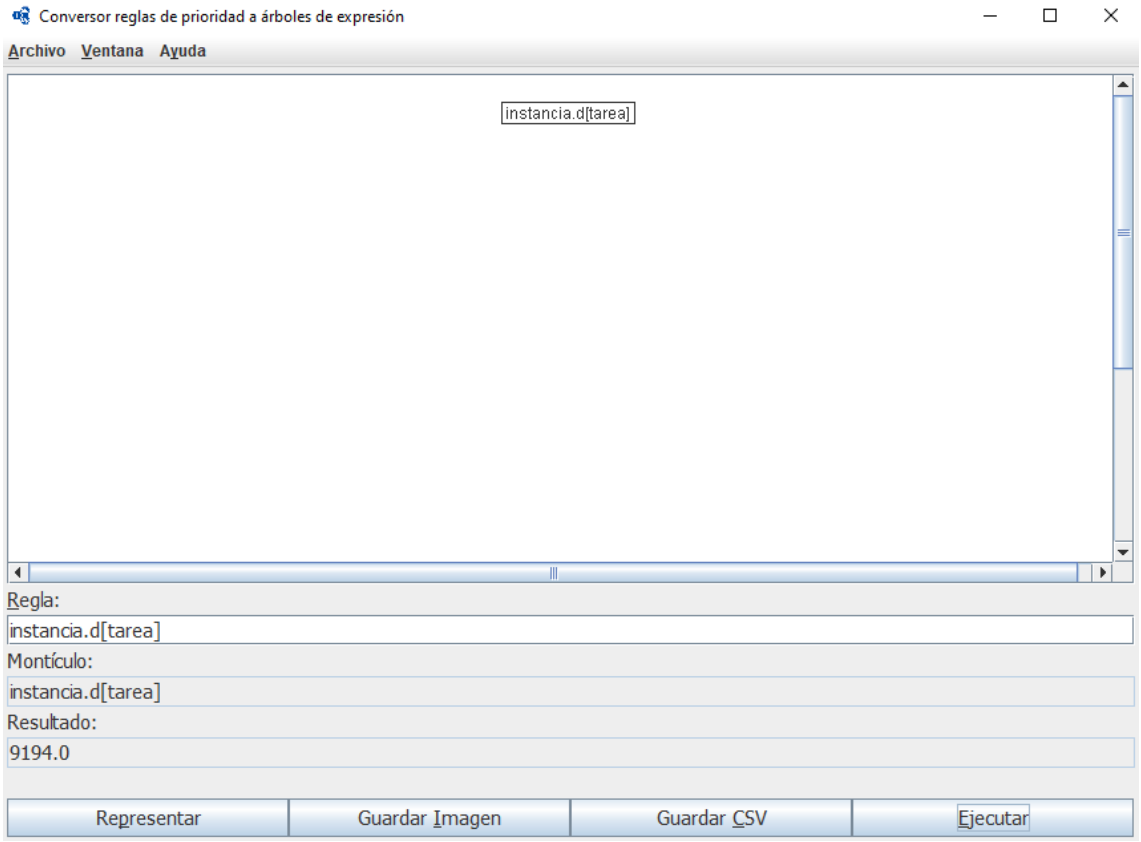

Figura 107. Ventana principal tras ejecutar una regla con éxito

6.3.7 Guardar imagen (PDF) desde historial

De igual forma que se permiten almacenar las imágenes en la ventana principal, el historial también ofrece la misma posibilidad con los elementos de esta ventana. El proceso es totalmente análogo al mostrado en la ventana principal, disponible en 6.3.3 Almacenamiento de representación de reglas, formato PDF

La única diferencia importante, es que desde esta ventana se muestra el siguiente error si es que se intenta generar PDF al tener el historial vacío.

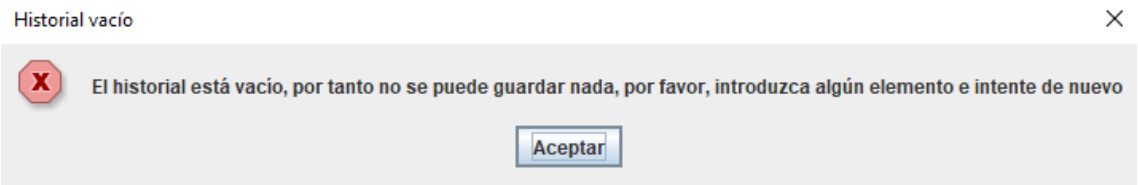

Figura 108. Error al hacer operación de almacenamiento historial

Por otro lado, también se puede dar la situación de tener una regla no válida cargada en el historial debido a un error del usuario al momento de escribirla. Si este es el caso, al tratar de guardar como imagen, mostrará un error relatando la fila errónea y guardando únicamente los elementos válidos. El error se muestra a continuación.

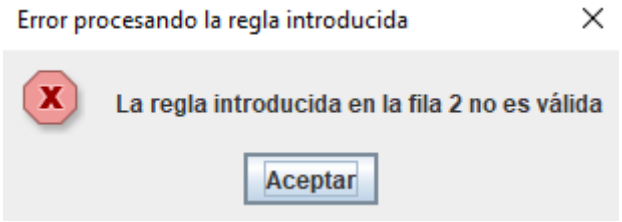

Figura 109. Error al generar PDF desde historial en fila determinada

6.3.8 Guardar CSV desde historial

De igual forma que se hace desde la ventana principal, desde esta ventana también existe la posibilidad de generar archivos CSV. La principal diferencia es que desde esta opción, se generará un único archivo con extensión ".csv" que contenga todas las reglas que constituyen el histórico, así como los valores de todas las celdas que lo componen. El proceso de almacenamiento es análogo al visto en la ventana principal, disponible en 6.3.4 Almacenamiento de reglas de prioridad, formato CSV

La única diferencia importante, es que desde esta ventana se muestra el siguiente error si es que se intenta generar un CSV al tener el historial vacío.

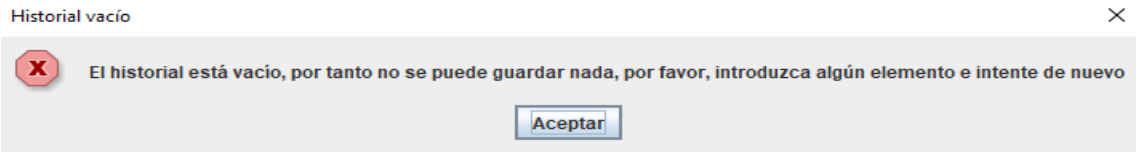

Figura 110. Error al generar CSV con historial vacío

Capítulo 7. Conclusiones

El objetivo principal de este proyecto era desarrollar una herramienta software que ayudase a los usuarios finales a la hora de representar de reglas de prioridad, su modificación y su ejecución.

Cabe destacar que a pesar de tener una serie de objetivos que se han planteado inicialmente y que no se han cumplido (debido principalmente al tiempo de desarrollo) podría ser más amigable para nuevos usuarios y hacer algo más cómodo su uso a usuarios expertos.

Otro punto importante que destacar durante el desarrollo del proyecto ha sido el uso de Java, que, si bien tiene sus limitaciones a la hora de realizar ciertas operaciones en comparación a otros lenguajes que hacen lo mismo de forma más eficiente, la versatilidad que proporciona este lenguaje que, a su vez, permite que se pueda ejecutar en cualquier sistema operativo que pueda generar una máquina virtual para su ejecución, no es para nada menospreciable.

En conclusión, la herramienta cumple los objetivos especificados para el proyecto. Se ha desarrollado una aplicación que facilita la manipulación de reglas de prioridad, así como su visualización y mejora en el desempeño de los usuarios de este nuevo sistema, que, además, es compatible en cualquier sistema operativo, facilitando el mantenimiento entre sistemas de forma drástica.

A pesar de haber desarrollado una herramienta que cumpla las especificaciones, aún cabe la posibilidad de ampliaciones de cara a mejorar la herramienta.

Capítulo 8. Ampliaciones y trabajo futuro

La herramienta desarrollada descrita a lo largo de este proyecto se ha diseñado para satisfacer una serie de necesidades relativas al manejo de reglas de prioridad y su representación gráfica. A pesar de que la aplicación cumple todos los objetivos planteados para este proyecto y cubre los requisitos de funcionalidad especificados para el mismo, es posible proponer una serie de mejoras y ampliaciones, no tenidas en cuenta debido a limitaciones temporales y de alcance del proyecto. Entre ellas:

- **Gramática configurable por el usuario:** La idea original era hacer una plantilla configurable por el usuario y que, al momento de ejecutar la herramienta, esta autogenerase las clases necesarias para la conversión a montículo de un conjunto de reglas, permitiendo que la gramática fuese flexible y que, tras cada ejecución, en caso de cambiarse algunos de los elementos del paquete “parser”, se generasen nuevas clases de forma transparente al usuario mediante la herramienta Antlr. Actualmente, se dispone de una gramática general que trata de abarcar la mayor cantidad de elementos posible, por lo que, si el usuario decidiese utilizar algún tipo de regla que no fuese contemplada en este esquema, tendría que modificar la gramática con la que se han generado las clases encargadas de procesar las reglas del proyecto, ejecutar la herramienta para generar las clases y compilar el proyecto. Esta mejora se traduciría en que el usuario sería capaz de crear, modificar o de ampliar la gramática a su gusto.
- **Mejora en la representación de reglas:** Actualmente, la representación de las reglas está definida en una de las clases del proyecto, pensadas de forma que permita visualizar cualquier regla contemplada sin salirse de un margen que comprende las dimensiones de un folio tamaño A4 en horizontal. Lo óptimo sería crear la ventana donde se representan los elementos con una dimensión acorde a lo que se va a representar, en vez de establecer el tamaño mínimo y máximo al de las dimensiones ya mencionadas. Para más información relativa a esto, consultar el apartado 6.2 Manual de desarrollador
- **Representación gráfica de una regla de forma manual:** En un principio, se planteó que los usuarios del sistema pudiesen aparte de generar representaciones gráficas tal como se hace ahora (de forma automática y transparente al usuario, en función de los datos introducidos), se pudiera pintar/representar/modificar elementos seleccionando estos, o bien pintando nuevos nodos, así como dibujando aristas que interconectasen a los nodos. Debido a las limitaciones temporales y que este aspecto no era muy relevante para los usuarios finales del sistema, se ha dejado de lado.
- **Internacionalización de la aplicación:** Ofrecer la posibilidad de cambiar de idioma la aplicación. En un principio no se planteó esta posibilidad debido a que los usuarios finales de la aplicación serían únicamente hispanohablantes, por lo que no se ha planteado hacer desde un inicio soporte a futuros nuevos idiomas.

- **Modificar valores utilizados para representación gráfica:** Permitir al usuario que esté utilizando la aplicación modificar los valores mostrados en 6.2 Manual de desarrollador

  Estos valores son fijos debido a que van intrínsicamente ligados con la forma en la que se representan actualmente las reglas, tal y como se comenta en el punto anterior **Mejora en la representación de reglas**, por lo que no es recomendable permitir la modificación de estos valores sin cerciorarse de que se haya cambiado la forma en la que se representan gráficamente los elementos.

Capítulo 9. Anexos

9.1 Formato de fichero de configuración

A continuación, se procede a mostrar los elementos que componen un fichero de configuración, explicando cuales son opcionales y cuáles no.

Un fichero de configuración está compuesto por los siguientes elementos, de los cuales solo ha de existir el elemento “terminals” siendo el resto opcionales.

- **Binary:** Indica los operadores que serán de tipo binario, es decir, que tengan dos y solo dos elementos, siempre. Ej. “+”,”-“,”*”.
- **BinaryInverso:** De forma análoga al anterior, se referirá a los operadores que tengan dos y solo dos elementos. La diferencia principal respecto al primer grupo es que la representación en notación infija de estos elementos es diferente de la del resto de elementos, por lo que es necesario crear esta separación a la hora de convertir internamente de montículo a regla. La característica principal de este tipo de operadores es que todas son funciones. Ej: Math.min, Math.max.
- **Unary:** Representa los operadores/funciones que únicamente acepten uno y solo un elemento. Ej: “-“ unario (no confundir con el “-“ binario), Math.sqrt, etc.
- **Terminals:** Representan los que vendrían a ser los nodos hoja del árbol de expresión.
- **Constants:** Representan valores numéricos que pueden ser utilizados en la regla.

Adicionalmente, se puede incluir más de una configuración por fichero, por ejemplo.

```
Problema de una máquina
binary          +, -, /, *
binaryInverso   Math.max, Math.min
unary           Math.pow, Math.sqrt, -
terminals       p, d, pmedia
constants       0.0, 0.1, 0.2, 1.0

Problema del TSP
binary          +, -, /, *
binaryInverso   Math.max, Math.min
unary           Math.pow, Math.sqrt, -
terminals       distance, centroid
constants       0.0, 0.1, 0.2, 1.0
```

Figura 111. Ejemplo archivo de configuración con TSP y OMSP respectivamente

Donde cada fila diferente corresponde a un tipo de problema diferente, siendo los de la imagen el OMSP y el TSP respectivamente. Esto es transparente a la hora de ejecutar la aplicación y solo tiene relevancia en el “.jar” suministrado por el usuario.

9.2 Formato de fichero de instancia OMSP

Este fichero consta de los siguientes elementos en orden:

- **[NINT: numeroEntero]:** Determina el número de intervalos de capacidad para una instancia. A su vez, está dividido en 3 elementos, **[ComienzoIntervalo, FinIntervalo, CapacidadIntervalo]** siendo el valor máximo de **FinIntervalo** 300000, que se considerará como infinito.
- **[NOP: numeroEntero]:** Representa el número de tareas. Los elementos que componen cada tarea son 3. **[índiceTarea, DuraciónTarea, DueDateTarea],** representando cada una el índice, la duración y el tiempo de validez de la tarea (a modo de caducidad). En el fichero de instancia habrá tantas filas como "numeroEntero" a partir del número de elementos marcado por NINT.

Un ejemplo de instancia se muestra a continuación:

NOP: 60

NINT: 15

0 39 4

39 67 5

67 100 6

…

476 300000 2

1 80 404

2 96 521

3 43 227

4 98 338

5 74 101

…

58 81 384

59 34 278

60 56 327

9.3 Formato de fichero de instancia TSP

Este fichero consta de los siguientes elementos en orden:

- **NAME:** Determina el nombre de la instancia.
- **COMMENT:** Incluye una breve descripción del problema que se describe en la instancia.
- **TYPE:** Determina el tipo de problema que representa la instancia, en este caso TSP.
- **DIMENSION:** Número que indica la cantidad de elementos que conforman la instancia.
- **EDGE_WEIGHT_TYPE:** Determina el tipo de valor que han de seguir las distancias entre nodos. En este caso, EUC_2D (distancia euclídea bidimensional).
- **NODE_COORD_SECTION:** Lista de elementos determina mediante tres elementos los siguientes valores en orden:
  - **Índice**
  - **Coordenada X**
  - **Coordenada Y**
- **EOF:** Determina el fin del documento.

Un ejemplo de instancia parcial de este tipo se muestra a continuación:

NAME: a280

COMMENT: drilling problem (Ludwig)

TYPE: TSP

DIMENSION: 280

EDGE_WEIGHT_TYPE: EUC_2D

NODE_COORD_SECTION

1 288 149

2 288 129

3 270 133

4 256 141

5 256 157

...

279 260 129

280 280 133

EOF