Carpeta de Desarrollo

Belén Beltran
belubeltran@gmail.com

Pablo Ariel Rodriguez
rodriguez.pabloariel@gmail.com

Federico Martin Rossi
federicomrossi@gmail.com

2do. Cuatrimestre 2014
75.52 Taller de Programación II
Facultad de Ingeniería, Universidad de Buenos Aires

Contenido

Capítulo 1

Introducción

1.1 Descripción general

El presente documento se centra en el desarrollo de *Graferator*, una aplicación de computadora cuyo objetivo es llevar a cabo el estudio de la teoría de grafos con el fin de ser utilizado en la enseñanza y el aprendizaje de la respectiva temática.

Recordando que un grafo es un conjunto conformado por vértices o nodos unidos por enlaces conocidos como aristas o arcos, el software permitirá de manera gráfica la confección y manipulación de estos a través de una interfaz de usuario intuitiva y simple.

Los usuarios podrán hacer uso de dos modalidades de ejecución, a saber:

- **Modo edición**: permite editar el grafo agregando nuevos vértices y aristas.
- **Modo aprendizaje**: resolución paso a paso con avance a solicitud del aprendiz (al siguiente paso o al resultado final).
- **Modo autoevaluación**: resolución paso a paso con elaboración guiada del resultado de cada paso a cargo del aprendiz. Confirmación de la corrección del resultado con opción de reintentar en caso de que fuere incorrecto, o ver el resultado correcto.

1.2 Funcionalidad

Graferactor constará de las funcionalidades básicas que se requieren para el armado de un grafo. Inicialmente se le permitirá al usuario la creación de un *Grafo Orientado* o un *Grafo No Orientado*.

Además, para cada una de estas opciones se deberá elegir si se desea comenzar a partir de un grafo vacío o si se prefiere generar un grafo aleatorio, especificando la cantidad de vértices y aristas a tomar en cuenta.

De esta manera, la aplicación permitirá:

- el alta o baja de vértices,
- el alta o baja de aristas o arcos,
- el borrado del grafo,
- la ponderación de arcos.

1.3 Algoritmos

Una vez armado el grafo deseado, Graferator permitirá a los usuarios aplicar sobre este una serie de algoritmos que ayudarán y aportarán al estudio de sus propiedades.

Como se ha adelantado previamente, cada algoritmo podrá ser ejecutado en dos modos distintos (modo aprendizaje y modo autoevaluación). Así, los usuarios no solamente comprenderán el grafo, sino que también adquirirán conocimientos acerca de cómo funcionan los algoritmos aplicables a estos.

De esta manera, habiendo confeccionado el grafo, se podrán efectuar los siguientes algoritmos:

- Recorrido en profundidad,
- Recorrido en anchura,
- Prueba de aciclidad.

Si el grafo es orientado, además será posible ejecutar los siguientes:

- Recorrido topológico en anchura y en profundidad (si es acícilico),
- Obtención de la cerradura transitiva,
- Obtención de componentes fuertemente conexas.

Si se trata de un grafo orientado con aristas ponderadas:

- Algoritmo de caminos mínimos de Dijkstra con un mismo origen (requiriéndose que todas las ponderaciones no sean negativas),
- Algoritmo de caminos mínimos de Floyd entre todos los pares de nodos,
- Algoritmo de flujos máximos de Ford-Fulkerson.

Por último, si el grafo es no orientado:

- Algoritmo de árbol de expansión de coste mínimo.

1.4 Ejecución de algoritmos

Dispuesto el grafo sobre la zona de diseño y elegido el algoritmo junto con el modo en el cual se ejecutará el mismo, el usuario deberá dar comienzo a la simulación. Para esto la interfaz contará con una serie de controles con los cuales se indicará el inicio o el fin de la ejecución, así como también controles con los que se podrá desplazar a lo largo de esta.

A medida que se corre el algoritmo se podrá visualizar paso a paso la ejecución de este sobre el grafo mediante colores que facilitarán la comprensión del avance en el mismo.

En el momento en que se realiza la elección del algoritmo, habrá una ventana interna en la que se visualizará el pseudocódigo respectivo. En la ejecución se resaltarán una a una las líneas de las sentencias de manera tal que los usuarios realicen un seguimiento detallado de todo el proceso, siendo esto de complemento y apoyo en el aprendizaje del funcionamiento de los distintos algoritmos. A esto se le incluye el conteo del número de sentencias ejecutadas. De esta manera los usuarios tendrán la información necesaria para realizar distintas tareas de análisis, tales como el cálculo de la complejidad algorítmica, refactorización de los algoritmos, entre otros.

Cada paso podrá ser deshecho dado que no solo existirán controles para realizar el avance, sino también para llevar a cabo el retroceso en cualquier instancia del proceso.

Capítulo 2

Herramientas de Desarrollo

2.1 Lenguaje y entorno de programación

En la presente sección se detallarán aquellas herramientas elegidas para llevar a cabo el desarrollo de la aplicación.

2.1.1 Lenguaje Java

Se ha optado como lenguaje principal de desarrollo el *Lenguaje Java* versión 1.8. Java es un lenguaje de programación de propósito general,concurrente y orientado a objetos, que fue diseñado específicamente paratener tan pocas dependencias de implementación como fuera posible.

Se ha optado por la utilización de este lenguaje dada la necesidad de queGraferator sea ejecutado al menos en los sistemas operativos MicrosoftWindows y aquellos basados en GNU/Linux. El motivo es que Java ha sido intencionalmente concebido para permitir que los desarrolladores de aplicaciones escriban programas una única vez y lo ejecuten en cualquier dispositivo (esto se conoce en inglés como WORA o Write Once, Run Anywhere. Es decir, el código que es ejecutado en una plataforma no tiene que ser recompilado nuevamente para poder correr en otra distinta.

Para el desarrollo de esta aplicación se debe contar con la *Java Development Kit 8* la cual se puede descargar de http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

2.1.2 Eclipse IDE

Si bien sería posible programar en Java mediante un editor de texto simple, hemos optado por la utilización de un IDE el cual, en nuestro caso será *Eclipse*. Este nos proporciona un ambiente mas ameno para no solo llevar a cabo el desarrollo en sí sino que consigo incluye herramientas de debug, pruebas (JUnit), controladores de versiones, entre otras.

El mismo se puede descargar de https://eclipse.org/downloads/

2.1.3 WindowBuilder (GUI)

Para confeccionar la interfaz gráfica se utilizará un plug-in de Eclipse conocido como *WindowBuilder*. Este nos permite desarrollar de forma rápida y cómoda la GUI (interfaz gráfica de usuario) de las aplicaciones Java por medio de una paleta de componentes variados que deben ser arrastrados y soltados sobre un área de diseño. El plug-in se encargará de generar

el código necesario para su funcionamiento desligándonos de esta tarea muchas veces engorrosa.

2.1.4 Maven

Para mejorar la administración del presente proyecto de software, se ha decidido incorporar el framewor conocido con el nombre de *Maven*. Con administración nos referimos a la gestión el ciclo de vida desde la creación del proyecto hasta la generación del binario que se distribuirá con el proyecto.

Maven es una herramienta que automatiza el proceso de construcción de un proyecto en el lenguaje Java, simplificando enormemente la realización de tareas como borrar los archivos .class, compilar, generar la documentación de javadoc, el jar, generar documentación web, entre otras cosas.

Maven, con comandos simples, crea una estructura de directorios para el proyecto con un sitio para los fuentes, los iconos, ficheros de configuración y datos, etc. Además, nos permite manejar de manera óptima el uso de librerías externas ya que nos permite indicarle qué archivos JARs externos hemos de necesitar y este se ocupará de buscarlos en internet y descargarlos para posteriormente ser incluidos por nosotros.

Provee un conjunto de estándares de construcción, un modelo de repositorio de artefactos y un motor de software que administra y describe los proyectos. Por ejemplo, teniendo en cuenta la naturaleza interdependiente de proyectos open source, Maven permite normalizar ubicaciones para los archivos fuente, documentación y archivos binarios, a fin de proveer una plantilla común para la documentación de proyecto y recuperar dependencias de proyecto de un repositorio compartido, de tal forma que el proceso de construcción consuma menos tiempo y sea mucho más transparente.

2.2 Controlador de versiones

Para la gestión de los diversos archivos y cambios que puedan surgir sobre estos a lo largo del desarrollo, hemos elegido utilizar el controlador de versiones *Git*.

Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente.

Por su parte, hemos optado por utilizar los servicios de Github.com, la cual es una plataforma online de desarrollo colaborativo de software para alojar proyectos utilizando Git. El uso de estos proporcionan varias herramientas útiles que mejoran en gran medida el trabajo en equipo.

2.3 Librerías utilizadas

Para el desarrollo de la aplicación se utilizaron las siguientes librerías:

- **JGraphT**: Es una librería que permite la representación teórica de un grafo.
- **JGraphX**: Es una librería que permite la representación gráfica de un grafo.
- **Log4J**: Es una librería que permite loggear fácilmente información y errores en un archivo.

2.4 Herramientas para documentación

Por último, para las tareas de documentación y confección de manuales y carpetas hemos decidido utilizar la herramienta *LaTeX*.

Latex es un sistema de composición de textos, orientado a la creación de documentos escritos que presenten una alta calidad tipográfica. Por sus características y posibilidades, es usado de forma especialmente intensa en la generación de artículos y libros científicos que incluyen, entre otros elementos, expresiones matemáticas. Además, es muy utilizado para la composición de artículos académicos, tesis y libros técnicos, dado que la calidad tipográfica de los documentos realizados es comparable a la de una editorial científica de primera línea.

La realización de los diagramas que componen los documentos se realizarán utilizando la herramienta online *Draw.io*, la cual es comparable con el software Microsoft Visio para la confección de planos y diagramas técnicos.

Finalmente, para el diseño de Mockups de la interfaz gráfica de Graferator, se utilizará la herramienta online *Moqups.com*.

Capítulo 3

Diseño de la solución

3.1 Arquitectura

En el primer capítulo de este documento hemos detallado la funcionalidad con la que contara Graferator. Allí hemos podido ver que será necesaria una interfaz gráfica la cual será la mediadora entre el usuario y la aplicación.

Para contar con un entorno visual va a ser necesario que exista un motor o modelo que lo respalde. Esto es, un módulo que se encarge de manejar aquellos aspectos abstractos e internos que permitirán que el software funcione correctamente. Dado esto es que se cree conveniente utilizar como base en el desarrollo de la aplicación el *patrón de arquitectura Model-View-Controller* o mayormente conocido por sus siglas MVC.

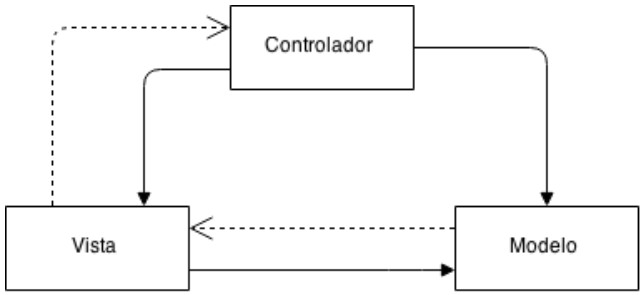

Imagen 3.1: *Diagrama del patrón de arquitectura MVC*

El patrón MVC hará posible separar los datos y la lógica de negocio de la interfaz de usuario (también conocida por las siglas GUI) y el módulo encargado de gestionar los eventos y las comunicaciones. Para esto, dicho patrón propone la construcción de tres módulos descentralizados: *Modelo*, *Vista* y *Controlador*. De esta manera, se establecen por un lado los componentes que hacen a la representación de la información y por otro lado se constituyen componentes para la interacción con el usuario. En la *Imagen 3.1* se muestra un diagrama representativo de este patrón.

La totalidad de la aplicación será desarrollada en base a dicha arquitectura en gran medida por los grandes beneficios que significan poseer el modelo y la lógica separados de todo lo restante. Por esta razón, en los siguientes apartados se profundizará sobre cada módulo y se describirán los componentes que lo conforman como así también la forma en la que trabajan entre sí y se comunican con el exterior para lograr los objetivos planteados por el alcance del software.

3.2 Modelo

Como ya se ha adelantado, el *Modelo* es la representación de la información con la cual nuestro sistema operará. Por lo tanto, gestionará todos los accesos a los datos, tanto consultas como actualizaciones. Mediante solicitudes de la *Vista*, este módulo le enviará la información requerida para ser mostrada y visualizada por el usuario. Estas peticiones llegarán al modelo a través del módulo *Controlador*.

3.2.1 Componentes básicos

Primeramente, encontraremos que el modelo necesitará representar a los constituyentes básicos de los grafos. En la *Imagen 3.2* se puede observar un primer diagrama de clases UML en donde se encuentran las clases *Vertice* y *Arista*. Ambas implementarán a la interfaz *Selectable*, quien simplemente establece que tanto los objetos Arista como Vertice son seleccionables. A su vez, Arista hereda de clase que define a objetos con peso, de manera tal que las aristas podrán ser ponderadas cuando así lo requieran.

Por otro lado, la clase Vertice también implementa las interfaces *Serializable* y *Comparable*, las cuales le dan caracteristicas para poder ser serializados y poder ser comparados respectivamente.

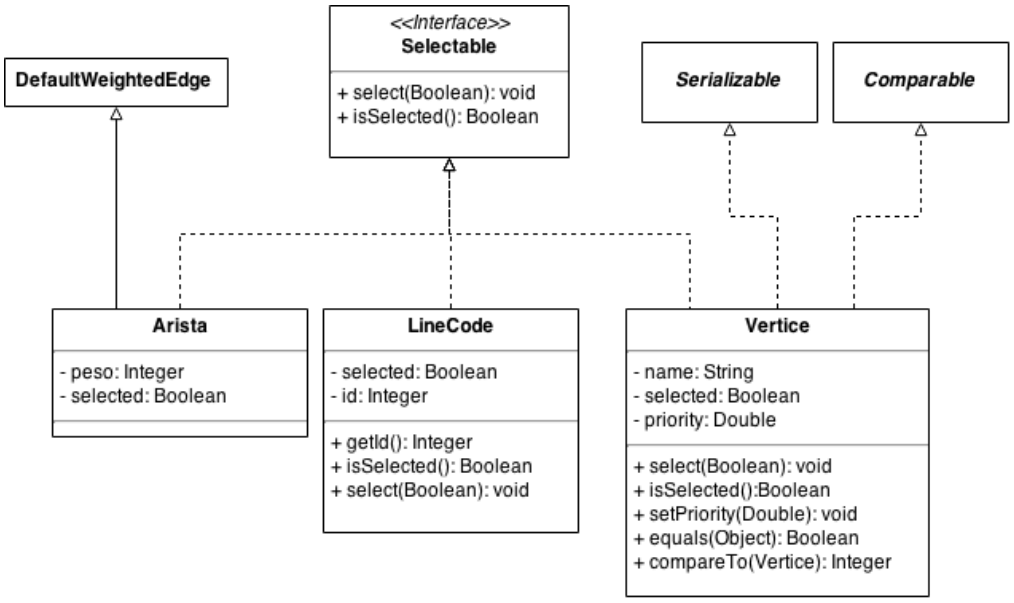

Imagen 3.2: *Diagrama de clases de los componentes básicos del modelo*

3.2.2 Representación del Grafo

Conformadas las componentes básicas, necesitamos una entidad que modele al grafo. Esta clase la denominaremos *GraphModel*. Esta hará uso de un conjunto de objetos Vertice relacionados entre sí por objetos del tipo Arista. En la *Imagen 3.3* se puede observar el diagrama de clases en donde se muestra como está conformado GraphModel. Esta clase se relaciona con objetos que implementan la interfaz *Executable*, quien será descrita mas

adelante. Además se puede observar que se relaciona con la clase *Resultado*, la cual se encarga de modelar el resultado de aplicar un algoritmo al grafo.

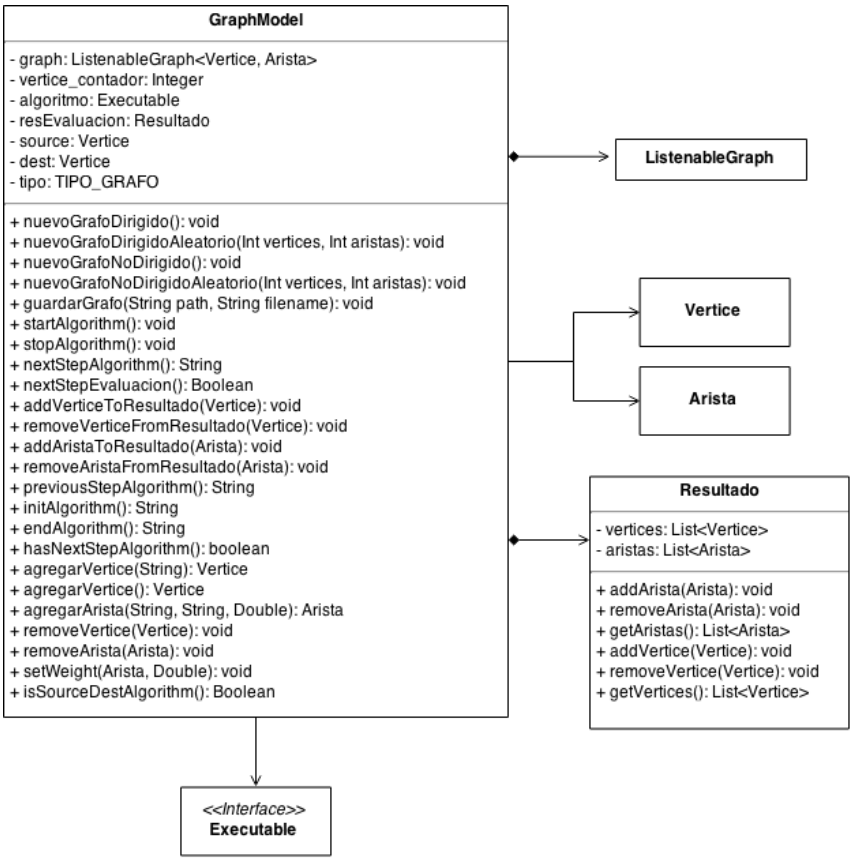

Imagen 3.3: *Diagrama de clases de la representación de los grafos.*

3.2.3 Representación de los Algoritmos

Cada algoritmo mencionado en el Capítulo 1 será representado por una clase del mismo nombre. Todas estas entidades son concretas y heredarán de la clase abstracta *GraphAlgorithm* (*Imagen 3.4*). De esta manera, se hará uso del concepto de polimorfismo utilizado en programación orientada a objetos (OOP) con el fin de no tener un modelo atado a un número fijo de algoritmos, sino que por el contrario, será posible extender la cantidad existente a aplicar sobre los grafos. Por lo tanto, tendremos un modelo altamente escalable.

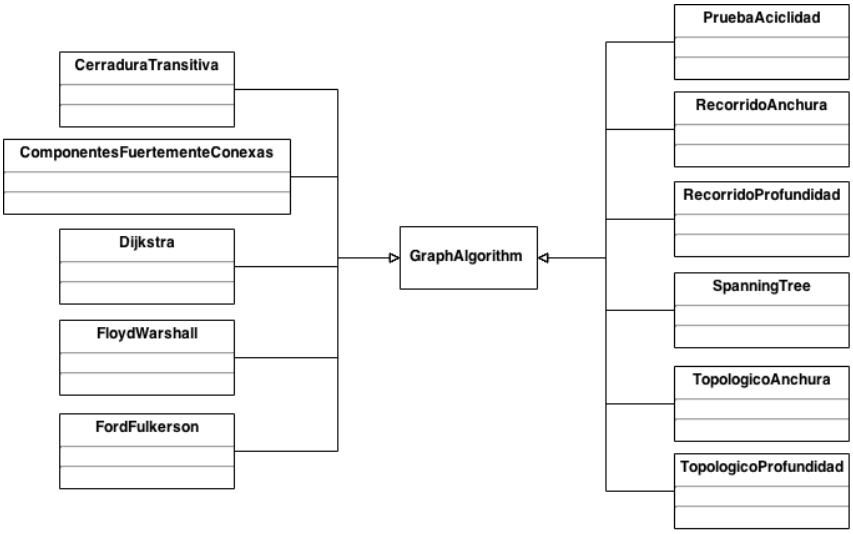

Imagen 3.4: *Diagrama de clases con jerarquía de algoritmos.*

La clase abstracta GraphAlgorithm implementa a su vez las interfaces *Algorithm*, *Documentable* y *Executable*.

La interfaz Algorithm simplemente permite obtener de aquellas clases que la implementen la URL del archivo en donde se encuentra el pseudocodigo del algoritmo en sí.

Documentable por su parte declara cómo obtener el título y la descripción del algoritmo.

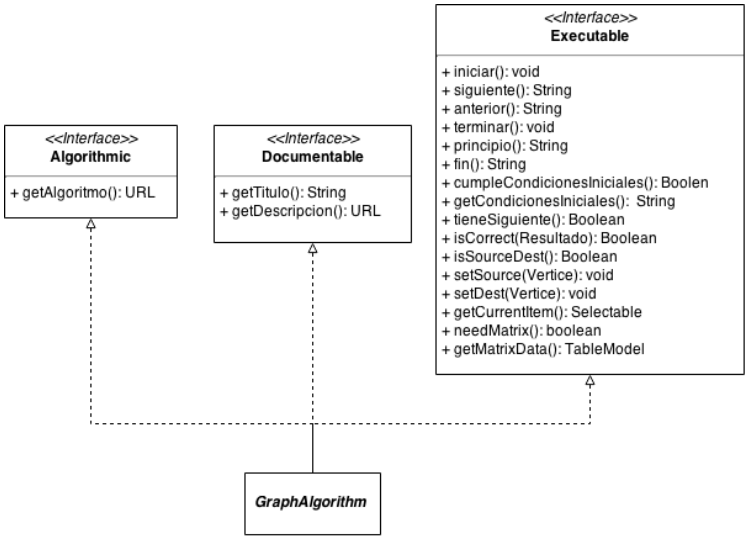

Imagen 3.5: *GraphAlgorithm y las interfaces que implementa.*

Por último, la interfaz Executable establece los métodos a través de los cuales se podrá ejecutar el algoritmo. Este fuerza a declarar y definir los métodos que permiten iniciar y detener la ejecución, como así también avanzar y retroceder en cualquier punto de la corrida.

Además declara mensajes especiales para comunicarse con los algoritmos, como por ejemplo, *isCorrect()*, al cual se le deberá pasar como parámetro un objeto del tipo Resultado y el algoritmo responderá si, en el paso actual, el resultado concuerda con este último. En la *Imagen 3.5* se muestra el diagrama correspondiente a lo recién descrito.

3.3 Vista

3.3.1 Mockups

Para el desarrollo de la vista se ha utilizado, como ya se ha mencionado en el Capítulo 2, el diseñador de GUIs de Java *WindowBuilder*. Nos limitaremos aquí a mostrar los distintos mockups que definirán la interfaz gráfica de Graferator.

En la *Imagen 3.6* se puede observar la pantalla inicial que verán los usuarios al iniciar la aplicación. Inicialmente no se abrirá ninguna ventana. Se contará con un menú superior donde se encontrarán las opciones de inicialización. Allí se podrá crear un grafo orientado o no orientado vacío, como así también se encontrará la opción de crear un grafo aleatorio de estos mismos tipos. En este mismo menú se encontrarán las opciones de guardar y abrir, cuya funcionalidad se orienta a poder persistir el grafo creado por los usuarios para abrirlo y utilizarlo en un futuro.

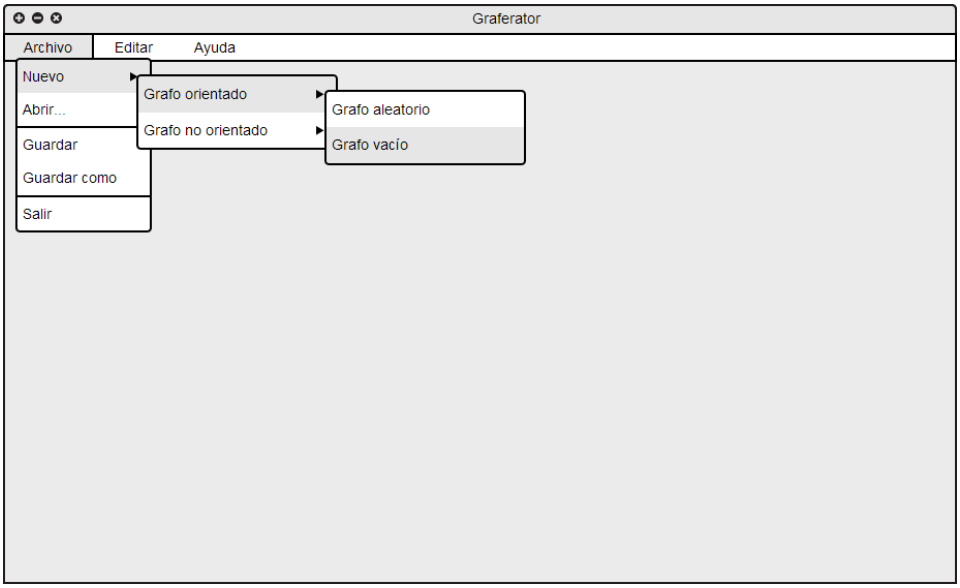

Imagen 3.6: *Mockup de la ventana inicial de Graferator*

Una vez elegido el tipo de grafo con el que se desea trabajar, se habilitarán las distintas ventanas internas que conforman el entorno del software (*Imagen 3.7*). Este último estará conformado por un panel izquierdo cuya zona superior contendrá un listado de los algoritmos disponibles dispuestos con radio buttons forzando a que solo pueda ser elegido uno a la vez. En la zona inferior se podrá elegir el modo con el que se ejecutará el algoritmo (modo aprendizaje, modo autoevaluación y modo edición).

A la derecha se encontrará un panel con dos pestañas: *Información* y *Algoritmo*. En la pestaña Información se podrán visualizar datos generales referentes al algoritmo activo (historia, particularidades, etcétera). En cambio, en la pestaña Algoritmo se encontrará el pseudocódigo correspndiente con La particularidad de que al iniciar la ejecución y avanzar o retroceder, los usuarios tendrán a disposición la línea del pseudocódigo activa resaltada con color. Esto permitirá un seguimiento mas ilustrativo de todo el proceso de análisis y estudio.

En la parte superior central, justo debajo de la barra de menú, se encontrará la barra de control. Esta barra contendrá los controles necesarios para llevar a cabo la ejecución de los algoritmos, es decir, tendrá los botones de *Iniciar*, *Avanzar*, *Retroceder*, *Ir al final* y *Volver al inicio*.

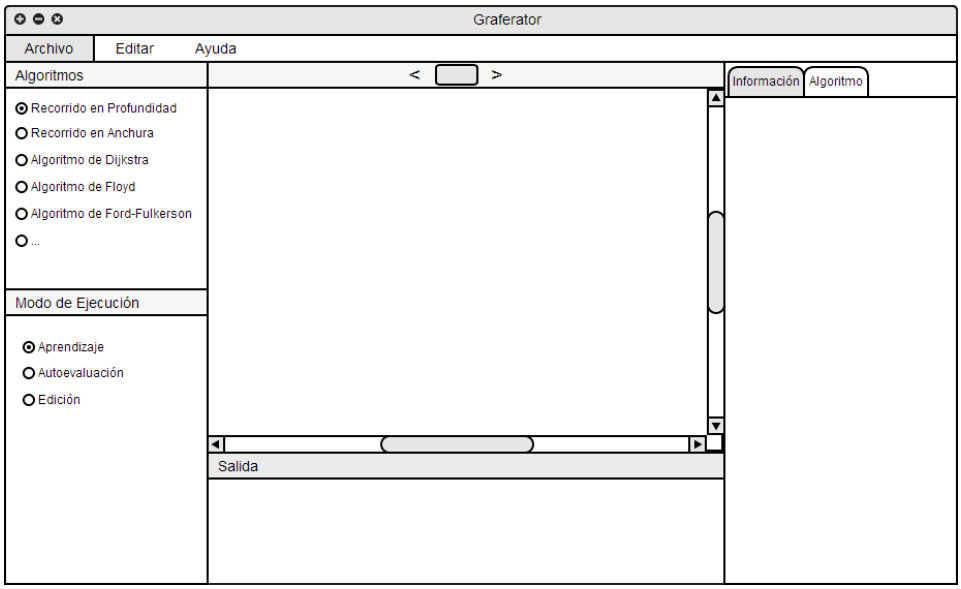

Imagen 3.7: *Mockup de la distribución de ventanas internas sobre el entorno.*

Se poseerá además, en la parte inferior, una ventana de salida en donde se podrán leer los resultados o mensajes provenientes de la ejecución y de los propios algoritmos. Allí se visualizarán los resultados finales de ejecución con formatos legibles y podrán ser fácilmente copiados para su utilización en medios externos a Graferator.

Por último, ubicado en el centro de la ventana central se encuentra la zona de trabajo principal. Allí los usuarios deberán confeccionar el grafo deseado. Para esto tendrán que hacer uso del menú editar de la barra superior, en donde encontrarán las opciones para agregar nuevos vértices y aristas. Luego podrán posicionar estos sobre dicha área para su posterior ejecución.

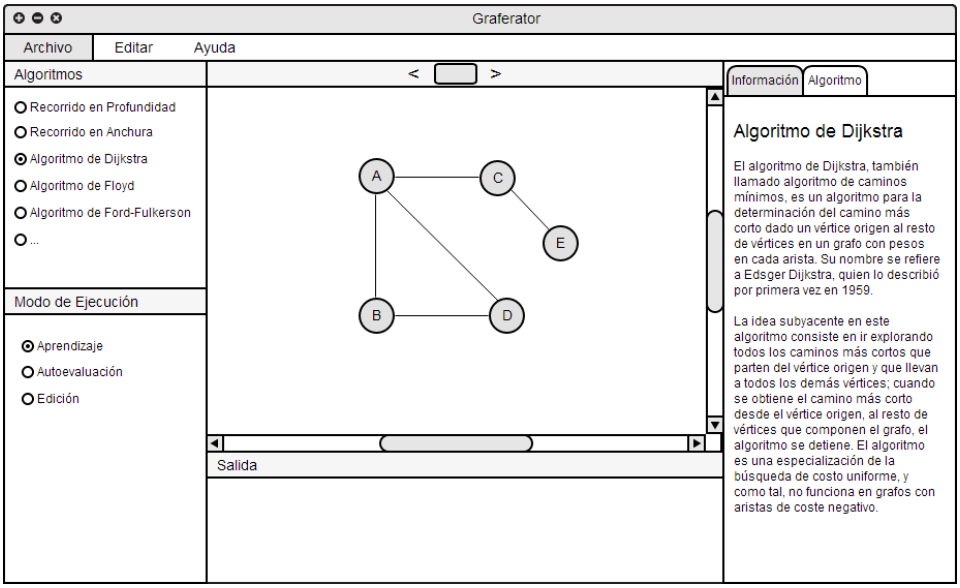

Imagen 3.8: *Mockup del área de trabajo en donde se diseñarán los grafos.*

3.3.2 Diagramas de Clases

En cuanto se refiere al diseño de la vista en sí podemos destacar las siguientes clases importantes:

- **Main**: Contiene los componentes generales.
- **GraphView**: Contiene la representación gráfica del grafo.
- **PanelAlgoritmos**: Contiene los botones de selección de algoritmos.
- **PanelInformación**: Contiene la información de los algoritmos.
- **PanelModo**: Contiene los botones de selección de modo.
- **PanelPseudocódigo**: Contiene el pseudocódigo de los algoritmos.
- **PapersMenu**: Contiene el menú de la aplicación.
- **PapersToolbar**: Contiene los botones de Inicio/Fin, siguiente y atrás de la ejecución de los algoritmos.

Las relaciones entre las mismas se detallan en la *Imagen 3.9*.

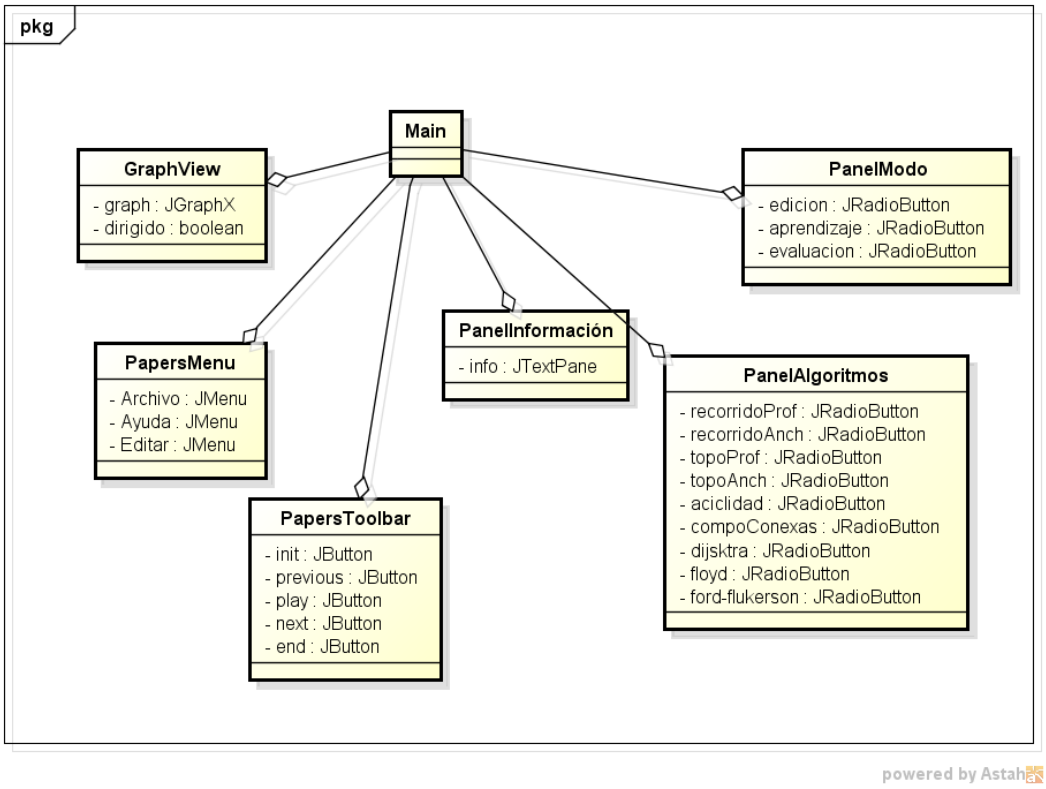

Imagen 3.9: *Diagrama de clases de la Vista.*

3.4 Controlador

El módulo Controlador contiene las clases que gestionan las acciones del usuario (selección de algoritmos, modo, creación de aristas, etc.) y es el encargado de reflejar dichas acciones tanto en modelo como en la vista.

3.4.1 Listeners del Menú

Son las clases que contienen la gestión de las acciones de los diferentes menús. Las relaciones entre las mismas se detallan en la *Imagen 3.10*.

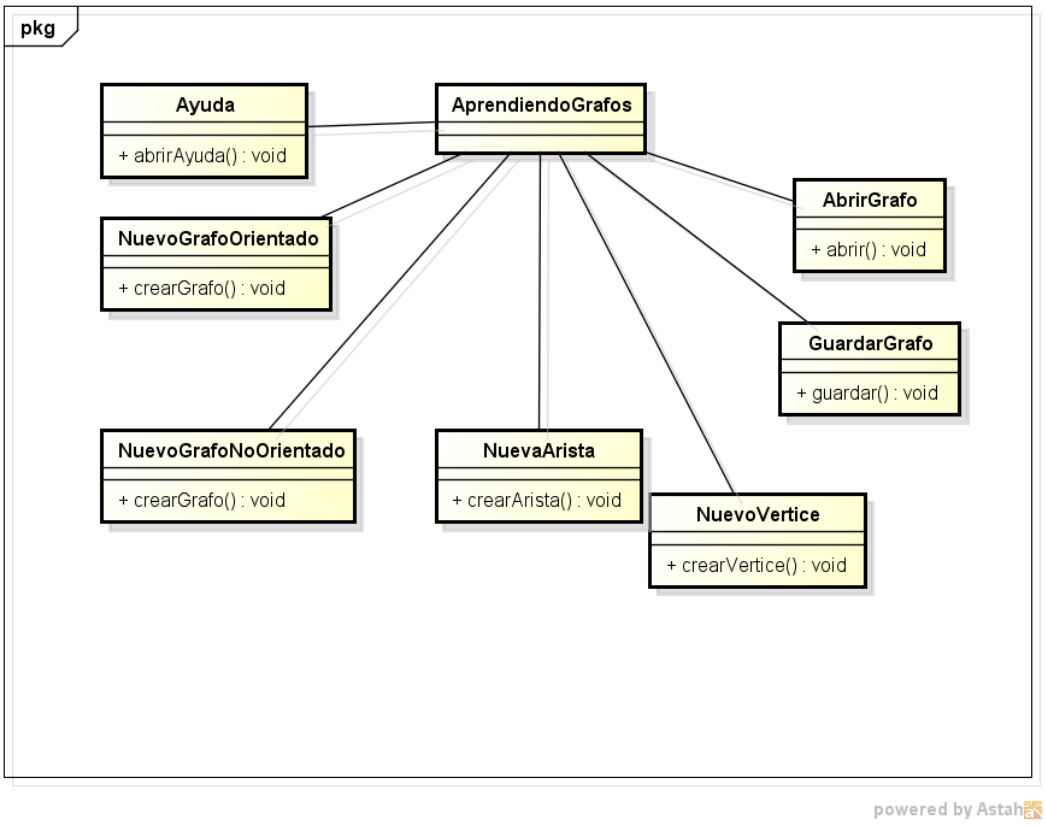

Imagen 3.10: *Diagrama de clases de los listeners del Menú.*

3.4.2 Listeners del Mouse

Son las clases que contienen la gestión de las acciones realizadas con el Mouse (selección de vértices, creación de vértices y aristas, etc.). Las relaciones entre las mismas se detallan en la *Imagen 3.11*.

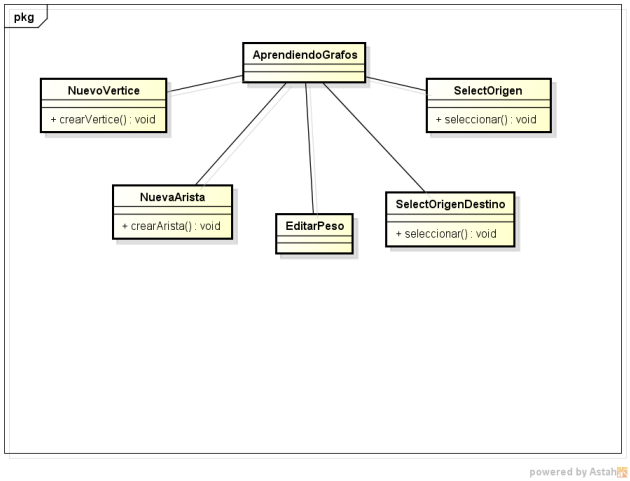

Imagen 3.11: *Diagrama de clases de los listeners del mouse.*

3.4.3 Listeners de ejecución

Son las clases que contienen la gestión de las acciones que conciernen a la ejecución de los algoritmos (siguiente, atrás, inicio/fin, etc.). Las relaciones entre las mismas se detallan en la *Imagen 3.12*.

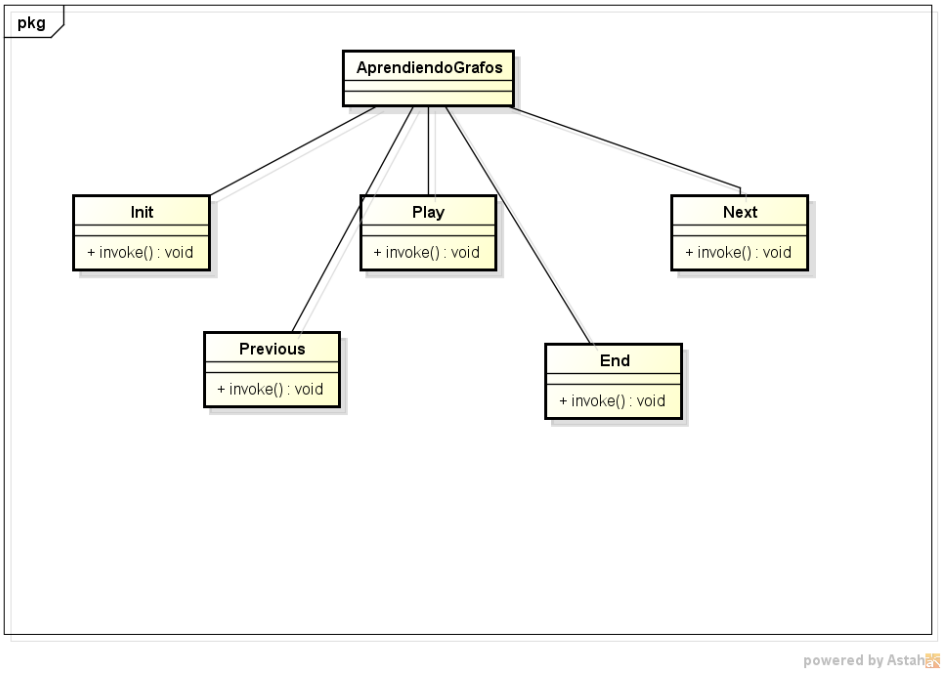

Imagen 3.12: *Diagrama de clases de los listeners de ejecución*