Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Desarrollo de una aplicación adaptativa en C# para la generación automática de diagramas de Gantt

Autor: Sergio David Martín Medina

Tutor: Víctor Fernández-Viagas Escudero

Dpto. Organización Industrial y Gestión de Empresas I
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Desarrollo de una aplicación adaptativa en C# para la generación automática de diagramas de Gantt

Autor:
Sergio David Martín Medina

Tutor:
Víctor Fernández-Viagas Escudero
Profesor Contratado Doctor

Dpto. Organización Industrial y Gestión de Empresas I
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado: Desarrollo de una aplicación adaptativa en C# para la generación automática de diagramas de Gantt

Autor: Sergio David Martín Medina
Tutor: Víctor Fernández-Viagas Escudero

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

A Víctor, a quien agradezco su ayuda y su implicación en este proyecto.

A mi familia y amigos, por haberme hecho más amenos estos últimos 4 años y sin los cuales no sería quién soy ahora.

A María, por estar ahí siempre, acompañándome tanto en los buenos como en los malos momentos.

Sergio David Martín Medina
Sevilla, 2020

Resumen

En cualquier entorno productivo o sanitario existe una gran cantidad elementos que han de ser planificados y organizados en el tiempo. El objetivo de este trabajo de fin de grado es desarrollar una aplicación informática en el lenguaje C#, que permita representar gráficamente el resultado de la programación de la producción mediante un diagrama de Gantt interactivo. Asimismo, se pretende dotar al usuario con las herramientas necesarias para realizar cambios en la programación propuesta, con su consiguiente modificación en el diagrama de Gantt. Con el fin de facilitar esta tarea, se va a presentar toda la información disponible mediante paneles informativos, los cuales proporcionen al usuario los datos esenciales de cada elemento productivo.

Abstract

In any productive or sanitary environment there is a great amount of elements that have to be planned and organized in time. The objective of this final thesis is to develop a computer application in the C# language, that will be able to represent graphically the production scheduling through an interactive Gantt chart. Also, it is intended to provide the user with the necessary tools to make changes in the proposed schedule, with its consequent modification in the Gantt chart. In order to facilitate this task, all the available information will be presented through information panels, which will provide the user with the essential data of each productive element.

Índice Abreviado

Índice

1 Introducción

La gestión eficiente de los recursos disponibles es clave para conseguir un rendimiento óptimo en un ambiente quirúrgico, proporcionando así servicios médicos con un coste razonable, manteniendo la satisfacción del paciente [Dios et al., 2015]. Es por este motivo que por parte de la comunidad investigadora de la Escuela Técnica Superior de Ingeniería (E.T.S.I.) se han sumado esfuerzos para aplicar técnicas de organización y control de producción al entorno médico con el fin de mejorar los servicios sanitarios. Este fenómeno puede verse reflejado en diversos artículos científicos como [Molina-Pariente et al., 2015a], [Molina-Pariente et al., 2015b] o [Dios et al., 2015], los cuales abarcan temáticas como metodologías de resolución del problema de planificación y organización de quirófanos con tiempos de cirugía dependientes del cirujano, aplicación de metodología heurística para la planificación/programación de quirófanos, o el desarrollo de un sistema de ayuda a la toma de decisiones para la planificación de los quirófanos por parte del director de cada unidad quirúrgica, respectivamente.

Estos proyectos se han realizado con el fin de mejorar los servicios sanitarios prestados por el Hospital Virgen del Rocío (Sevilla, España), buscando la optimización en la gestión de los recursos disponibles, centrándose en el área de quirófanos. Se trata de un problema complejo en el que se debe tener en consideración la disponibilidad de los distintos quirófanos, entre los que podría haber alguno especializado, el personal clínico, dentro del cual destacan los cirujanos y los pacientes, entre otras restricciones [Dios et al., 2015]. Además, dada la variabilidad inherente de las intervenciones quirúrgicas, la monitorización de los quirófanos y la aplicación de medidas correctivas (debido a ajustes de última hora provocados por ausencias de los pacientes o cancelaciones) son necesarias.

1.1 Descripción del entorno del problema

Este trabajo de fin de grado pretende seguir contribuyendo a la mejora de los servicios del Hospital Virgen del Rocío, al igual que los artículos ya mencionados, mediante el desarrollo de una aplicación informática. Este hospital es uno de los más grandes de España y cuenta con 1400 camas, 50 quirófanos y realiza más de 60,000 intervenciones quirúrgicas al año [Dios et al., 2015]. Los quirófanos están distribuidos en distintas unidades quirúrgicas, las cuales se centran en un tipo de intervención concreta, normalmente por ramas clínicas.

El director a cargo de cada unidad quirúrgica debe realizar la planificación/programación de las intervenciones, esto incluye la estimación de los tiempos de cada intervención, su ordenación en el tiempo y la asignación de los cirujanos a cada operación. Esta compleja tarea es fundamental para la correcta gestión de los recursos y su consiguiente incremento en el rendimiento quirúrgico.

Aunque, como se detallará a lo largo del presente documento, la aplicación desarrollada tiene un carácter genérico, se presentará con una parametrización ajustada a los sectores productivos y sanitarios, con especial atención a la programación de pacientes en quirófanos. Así, la mayoría de los ejemplos analizados en el documento y restricciones consideradas, surgen de la Unidad de Cirugía Cardiovascular y Grandes Quemados. Por este motivo, se va a resumir el proceso de planificación que se sigue dicha unidad quirúrgica, pasando por cada una de sus fases, comentando los inconvenientes que pueden surgir en cada una de ellas:

1. **Asignación del cirujano al paciente.**

   Lo primero que se realiza previo a una intervención es una consulta con el paciente, son una serie de pruebas que garanticen la necesidad de dicha intervención. El encargado de pasar esa consulta suele ser un cirujano por lo que por sencillez y conveniencia del paciente se suele fijar el cirujano al paciente que se atendió en consulta. No obstante, las intervenciones son ejecutadas por un equipo quirúrgico, usualmente conformado por más de un cirujano, por lo que la asignación del resto del equipo sí es tarea del director. Las principales complicaciones a la hora de hacer esta asignación es la de cuadrar los horarios de todos los cirujanos que intervienen en cada operación y gestionar a los especialistas.

2. **Estimación de los tiempos de operación.**

   Una vez determinados los cirujanos y la tipología de la intervención, se ha de realizar una estimación del tiempo que va a ser necesario para su finalización. Para realizar esta estimación el equipo de planificación se basa principalmente en la experiencia. Esto se debe a que la naturaleza de las intervenciones es poco determinista dada la cantidad de variables existentes, por lo que la experiencia es imprescindible para realizar una aproximación aceptable.

   Dentro de las múltiples variables que afectan al tiempo de operación se encuentra el tipo de intervención, el cual es conocido, por lo que es posible emplear una regresión histórica para calcular el tiempo medio de operaciones similares en el pasado. El historial clínico del paciente también es de gran importancia, ya que hay intervenciones que pueden verse alteradas en función de la edad o sexo del paciente o de patologías previas que este presente. Por último, el cirujano que realice la intervención influye en la duración de la misma, ya que el número de veces que haya realizado esa operación en concreto, su experiencia como cirujano, o la propia destreza del mismo van a modificar el tiempo estimado considerando únicamente el tipo de operación y el historial del paciente.

   El equipo médico disponible en el quirófano también puede influir en la duración de la operación pero normalmente suelen estar equipados similarmente o, si tienen diferente equipo, es porque se realizan operaciones de distinta índole en su interior [Molina-Pariente et al., 2015a]. Destacar que, dentro del tiempo de intervención, se incluyen los cambios entre equipos de cirujanos, en el caso de ser una intervención muy larga, o el tiempo de preparación del quirófano.

3. **Planificación de las intervenciones.**

   La planificación de las intervenciones consta de dos actividades claramente diferenciadas, la primera de ellas es la ordenación de las mismas en el tiempo y la segunda la asignación de cada una de ellas a un quirófano. Estas pueden entrar en conflicto, ya que el orden que optimice el número de operaciones que se llevan a cabo a lo largo del tiempo puede no ser

compatible con la disponibilidad de los quirófanos o los cirujanos.

Siendo el caso de los quirófanos especialmente crítico, ya que ciertas operaciones necesitan un equipamiento específico que solamente está disponible en salas especializadas. Dada la cantidad de variables presentes se dispone de una aplicación que organice las operaciones en base a algoritmos de optimización y, a pesar de esto, realizar una planificación con más de dos semanas de anticipación es una tarea casi imposible.

4. **Representación de la planificación en un diagrama.**

   Por último, se debe representar la programación obtenida en un diagrama que sea fácilmente comprensible por la plantilla, para que se identifiquen y monitoricen las operaciones que se deben de estar realizando en cada momento, así como los cirujanos involucrados. Esta representación se suele realizar mediante una aplicación interna del hospital.

Una vez explicado el proceso de planificación de los quirófanos, se pueden apreciar numerosas similitudes con el entorno industrial productivo. Si bien es cierto que el número de operaciones realizadas en un hospital, especialmente si se trata de cirugía mayor, es notablemente reducido en comparación con una fábrica de productos en serie, como puedan ser en la industria alimentaria o la automovilística, los principios de programación son perfectamente aplicables.

En este caso se tiene a los pacientes, que en el entorno productivo serían considerados como "trabajos", los quirófanos, que serían análogos a las máquinas de producción y por último los cirujanos, los cuales cumplirían el papel de operarios. En el caso de los cirujanos, estos toman un papel de mayor relevancia que los operarios en un entorno industrial típico, ya que son un recurso que es capaz de operar en cualquier máquina, pero que puede no ser especialista en todas las operaciones. Además en este caso los pacientes solo han de ser procesados una vez en una máquina cualquiera, salvo excepciones por falta de equipamiento.

Por este motivo se puede encontrar una gran similitud con el entorno aeronáutico, especialmente en las últimas etapas de fabricación, cuando las operaciones son las de montaje y pruebas de la aeronave en la FAL[1] o la PRE-FAL[2]. Estas etapas se caracterizan por el reducido número de operaciones que se realizan (hablando a grandes rasgos) y el elevado tiempo que estas requieren, similar al de las intervenciones quirúrgicas. Además, el personal de la industria aeroespacial suele estar altamente especializado, al igual que ocurre con los cirujanos en cada una de las ramas clínicas. También existe una analogía de los quirófanos con las máquinas de control numérico de varios ejes, ya que ambos se caracterizan por la polivalencia de las operaciones que pueden llevar a cabo.

1.2 Objetivos y requisitos de diseño

Aunque las posibilidades de mejora en el proceso de planificación anteriormente descrito son amplias, es imposible abarcarlas todas. Este trabajo de fin de grado pretende proseguir con la tónica de llevar herramientas de la planificación de la producción al entorno sanitario, focalizándose en la última etapa de la planificación, la representación de la misma en un diagrama. En concreto, se busca desarrollar una aplicación informática que realice dicha representación mediante un diagrama de Gantt interactivo, el cual permita, mediante diversas acciones por parte del usuario, obtener

[1] *Final Assembly Line* (línea final de ensamblado), por sus siglas en inglés, es la planta de fabricación donde se realiza la integración de los grandes conjuntos de la aeronave con el fuselaje así como las pruebas de línea de vuelo.

[2] Planta de fabricación previa a la FAL en la cual se realiza el montaje y pruebas de los grandes conjuntos de una aeronave como puedan ser los estabilizadores, alas, fuselaje, etc.

datos de las distintas operaciones, los cirujanos, pacientes o quirófanos, además de realizar modificaciones en la programación propuesta. El lenguaje de programación empleado será C#, utilizando su herramienta integrada Windows Forms en la IDE[3] *Visual Studio Comunity*.

Esta aplicación busca solventar ciertos problemas que surgen a la hora de obtener el diagrama, algunos de los cuales tienen su raíz en la herramienta que se está empleando en la actualidad. El principal de ellos es el elevado grado de sofisticación de aplicación actual, ya que se vuelve difícil de manejar para el equipo de planificación del hospital, esto es debido a que pretende integrar la parte de planificación con la de representación del diagrama. Otro de los problemas que surge es que, una vez obtenidos los datos de la programación por parte de la herramienta de optimización, la modificación de los mismos es compleja, por lo que, en el caso de que una operación se adelante o se cancele por motivos ajenos al quirófano, como una ausencia por parte del paciente, por ejemplo, no se posible sustituirlo por otro de una forma cómoda.

Es cierto que existen varias opciones de aplicaciones en el mercado que permiten al usuario realizar un diagrama de Gantt en base a unos datos de planificación de la producción, pero están demasiado enfocadas al entorno industrial con difícil aplicabilidad en el ambiente quirúrgico. Además, el producir una herramienta propia permite realizar modificaciones y mejoras al software en un futuro, así como unos niveles de personalización imposibles de conseguir con una aplicación comercial.

Una vez aclarados estos conceptos y los problemas que se pretenden solventar, se pueden extraer los requisitos de diseño de la aplicación. La aplicación debe cumplir con las siguientes especificaciones, todas ellos presentes en la figura 1.1:

- Introducción de los datos mediante la lectura de un archivo de texto.
- Inclusión de una zona que muestre una lista con los trabajos no asignados (Zona A).
- Inclusión de una zona que represente el diagrama de Gantt de los datos introducidos (Zona B).
- Inclusión de una panel de información que muestre datos de los distintos elementos del diagrama al hacer clic sobre ellos (Zona C).
- Inclusión de un panel de información genérico que muestre datos generales (o indicadores de rendimiento) de la solución actual (Zona D).
- Mecánica de modificar la solución propuesta modificando el orden de la planificación o los quirófanos asignados.

[3] *Entorno de Desarrollo Integrado*, por sus siglas en inglés.

Figura 1.1 Esquema de las distintas zonas de la aplicación, *elaboración propia*.

Para realizar cambios en la programación se implementará una mecánica *drag and drop* que permita desplazar las operaciones a realizar de un quirófano a otro o eliminarlas de la programación, pero sin alterar el tiempo de operación dentro del mismo quirófano. En la figura 1.1 existen zonas adicionales a las exigidas como requisitos de diseño, esto es debido a que durante el desarrollo de la herramienta se han ido incluyendo funcionalidades que puedan ser de utilidad, como un zoom o la posibilidad de guardar las modificaciones realizadas en un archivo diferente.

Remarcar que el objetivo de esta aplicación no es del de optimizar la programación ni planificar las intervenciones, sino el de proporcionar un soporte gráfico sencillo que permita la modificación de los datos proporcionados externamente. La posición de la aplicación en la cadena de planificación/programación es la del diagrama de la figura 1.2.

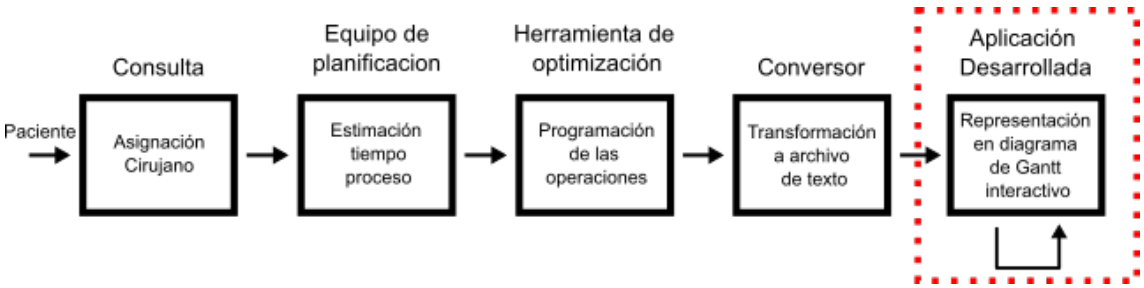

Figura 1.2 Esquema del proceso de planificación de las intervenciones quirúrgicas, *elaboración propia*.

Como se puede observar en la figura 1.2, los datos del optimizador se tendrán que transformar a un archivo de texto legible por la aplicación empleando un conversor, el cual tampoco se desarrollará en el marco de este trabajo de fin de grado.

Junto a los objetivos que ha de cumplir la aplicación, se han marcado una serie de pautas para que el desarrollo de la misma pueda ser continuado en un futuro. Se va a intentar en todo momento

respetar las convenciones del lenguaje C# con el fin de facilitar la comprensión del código. Además, se va a tratar de sacar el máximo provecho a la capacidad de emplear POO[4] que ofrece el lenguaje para minimizar el número total de líneas de código.

Asimismo, esta ultima cualidad permitiría expandir la aplicación, ya que la reutilización del código es notablemente más sencilla. También se va a buscar la optimización del tiempo de ejecución del programa ya que, al tratarse de una aplicación que maneja gráficos, si se ejecuta en ordenadores con peor procesador se podría apreciar *lag* en la pantalla.

Por último, aunque la necesidad de esta aplicación surge dentro de un grupo de investigación en el entorno sanitario, se va a emplear una terminología industrial con el fin de producir un producto genérico que pueda ser implementado tanto en el entorno sanitario como en la industria convencional. Por este motivo se va a emplear durante todo el documento la terminología industrial (trabajo, maquina y operario), en lugar de la médica. Asimismo, en caso de que la producción del archivo de entrada sea lo suficientemente sencilla, esta herramienta se podría incorporar a la docencia en algunas asignaturas adscritas al departamento de Organización Industrial y Gestión de Empresas I de la E.T.S.I.

[4] Programación Orientada a Objetos

2 Marco Teórico

En este capítulo se va a realizar una breve explicación de los fundamentos teóricos en los que se basa la aplicación. De esta manera, se facilita la comprensión de la estructura interna del código, ya que existe correspondencia clara entre las clases empleadas en el código y los elementos de la planificación de la producción.

Se van a abarcar tres conceptos principales, comenzando por unas nociones básicas que servirán para definir los elementos de la organización industrial, los cuales se verán incluidos en la aplicación. Se continuará con una introducción a las metodologías de planificación, que, a pesar de no ser el objetivo de esta aplicación, son fundamentales para tener una visión global de los conceptos anteriormente mencionados. Por último, se explicará con detalle qué es un diagrama de Gantt, sus partes y funciones. En caso de que el lector desee profundizar en la teoría de planificación se recomienda acuda a los libros referentes de la literatura, [Framinan et al., 2014] y [Pinedo, 2012].

2.1 Organización de la producción

La organización de la producción es la rama de la ingeniería que se encarga de distribuir las distintas órdenes de fabricación en el tiempo buscando maximizar la utilización de los recursos de la planta productiva, optimizando así su rendimiento. Dentro de la organización de la producción existen dos elementos principales que conforman todo problema de planificación, estos son:

- **Trabajos**

  Se entiende por trabajo a un objeto el cual debe de ser procesado en una máquina con el fin de proporcionarle un valor añadido. A cada uno de estos procesos se les conoce con el nombre de operación y el tiempo empleado por la máquina en realizarlos se conoce como tiempo de proceso. Un trabajo puede sufrir un número indefinido de operaciones en diferentes máquinas, en función de su tipología o de la estructura de la fábrica, todo esto quedará explicado con posterioridad. Existen diversos tipos de trabajo, como puedan ser la producción de una lata de aluminio desde la materia prima o la fabricación de una aeronave comercial. En el ámbito clínico los pacientes son el equivalente a los trabajos puesto que son los que sufren las "operaciones", tanto en el sentido clínico como en el de organización de la producción.

- **Máquinas**

  Las máquinas son elementos productivos que forman parte de la planta industrial y permiten realizar las distintas operaciones sobre los trabajos, ya sean de transformación o de transporte. La distribución y estructura de las máquinas sobre la fábrica se conoce como

distribución en planta y puede tomar una gran variedad de formas, las cuales se explicarán más adelante. Las máquinas pueden ser desde un simple torno mecánico, hasta una compleja máquina especializada de la industria. Dentro de la planificación orientada al ámbito clínico los quirófanos se consideran las máquinas en las que los pacientes son "procesados".

Tras aclarar los elementos principales hay que explicar la nomenclatura que se emplea típicamente en programación de la producción. Además, se emplea una notación matricial para identificar a los trabajos y las máquinas. La notación más común es:

- **n** : Número de trabajos
- **m** : Número de máquinas
- $\mathbf{p}_{ij}$: Tiempo de proceso del trabajo j en la máquina i
- $\mathbf{C}_{ij}$: Tiempo de finalización del trabajo j en la máquina i
- $\mathbf{S}_{ij}$: Tiempo de comienzo del trabajo j en la máquina i
- $\mathbf{R}_{j}$: Ruta de fabricación del trabajo j

Como puede apreciarse la notación empleada es matricial, lo que sugiere que el código va a tener que emplear esto para almacenar los datos. Cabe destacar que el "operario" no se contempla como elemento productivo, en el caso de los cirujanos estos afectan únicamente al tiempo de proceso de cada trabajo, pero es importante considerarlos para cuadrar los horarios de la plantilla. Si se habla de la planta productora como conjunto existen una serie de variables que sirven para medir su productividad. Estas son las siguientes:

- **Capacidad (K)**

  Número máximo de unidades por hora que puede producir la planta, normalmente determinado por la máquina más lenta, denominado cuello de botella.

- **Tiempo de ciclo**

  Tiempo transcurrido entre dos unidades consecutivas.

- **Tasa de fabricación**

  Número de unidades que se fabrican por unidad de tiempo, es la inversa del tiempo de ciclo.

- **Takt time**

  Tiempo de ciclo con el que se debería de producir para satisfacer la demanda.

Con esto aclarado, faltaría comentar los distintos tipos de distribución de la planta productora, en función del número de máquinas empleadas y el orden de procesado de los trabajos. Ordenadas en función de su complejidad las distribuciones más comunes son las siguientes:

- **Máquina única**

  Este es el caso más sencillo y se cuenta con una única máquina por el cual han de procesarse todos los trabajos. Normalmente esta distribución solamente se encuentra en la teoría.

- **Máquinas paralelas**

  Similar a la distribución en máquina única pero añadiendo otras máquinas en paralelo, el trabajo tiene que procesarse una única vez en cualquiera de ellas. Estas máquinas pueden ser idénticas, lo cual implica tiempos de proceso idénticos, no relacionadas, donde los tiempos de proceso varían con cada máquina, o relacionadas, lo cual implica que el tiempo de proceso depende de la máquina en función de un coeficiente que multiplica el tiempo de proceso de la máquina "ideal".

- **Taller de flujo regular**

  Consta de un conjunto de máquinas en serie, de secuencia prefijada, por las cuales se procesa cada trabajo.

- **Taller de trabajos**

  Consta de un conjunto de máquinas en paralelo, normalmente con propósito diferentes por las cuales se procesa cada trabajo, cada uno con una ruta predeterminada.

- **Taller abierto**

  Similar al taller de trabajos, pero en este caso los trabajos deben procesarse en cada máquina. Sin embargo, la ruta de estos no está fijada, por lo que es el objetivo de la programación.

- **Entorno híbrido**

  El entorno híbrido busca mezclar las distribuciones anteriores uniendo máquinas paralelas con algún taller. De esta manera se tienen estados en lugar de máquinas y, en función del tipo de taller seleccionado se obtiene la ruta de fabricación. En cada uno de esos estados se encuentran máquinas en paralelo, que pueden estar relacionadas o no, o ser idénticas.

Para el caso de la planificación en quirófanos, se tiene una distribución de máquinas en paralelo, las cuales normalmente son idénticas. Aun así se pretende dar flexibilidad en la aplicación para abarcar cualquiera de los casos propuestos.

2.2 Metodologías de programación

En esta sección, se busca analizar las metodologías más comunes de planificación así como las restricciones que se han de tener en cuenta a la hora de planificar. Es cierto que muchas de estas restricciones son triviales, pero no deben pasarse por alto, puesto que al permitir en la aplicación realizar cambios manuales en la programación se puede considerar como una metodología de programación en sí misma, por este motivo hay que prevenir que los posibles errores del usuario e impedir que estos se produzcan, diseñando protecciones en el sistema. Dentro de estas restricciones se encuentran:

- Planificación fuera de horario.
- Procesar varios trabajos simultáneamente en la misma máquina.

La programación fuera de horario significa emplear tiempo en el cual se supone que la maquinaria está parada como útil para la programación. Esta situación es problemática, ya que puede suponer pagar horas extras por parte de la empresa. A pesar de esto, en la aplicación se va a permitir programar fuera de horario (es decir estando el quirófano cerrado) siempre y cuando se encuentre dentro del rango representado en el diagrama de Gantt. Aún así se van a resaltar las zonas fuera de horario para que el propio usuario sepa que esa acción puede suponer un coste adicional para la empresa. No permitir el procesado de varios trabajos simultáneamente en una misma máquina ya que puede darse el caso de que la capacidad de esta sea superior a la unidad. Sin embargo, resulta notablemente más sencillo representar una máquina de mayor tamaño como varias "máquinas" de capacidad unitaria que hacerlo como una única unidad.

Las metodologías de programación buscan resolver un problema de planificación concreto, este consta de unos datos conocidos, en base a los cuales girará toda la planificación. Una vez fijado el problema, los datos que se han de conocer para cada trabajo son:

- $\mathbf{r}_j$: Tiempo de llegada del trabajo j

- $\mathbf{d}_j$: Fecha de entrega del trabajo j
- $\mathbf{w}_j$: Peso del trabajo j

En base a estos datos se pueden obtener las diferentes variables del problema, las cuales se emplean para evaluar la calidad de la solución obtenida. Con estas variables se elaborarán una serie de funciones objetivo de diversa índole en base a las cuales se establecerá qué planificación es la óptima. Las variables del problema derivan de los datos del problema y son las siguientes:

- $\mathbf{F}_j$: Tiempo de flujo del trabajo j (C_j - r_j)
- $\mathbf{L}_j$: Retraso del trabajo j (C_j - d_j)
- $\mathbf{T}_j$: Tardanza del trabajo j : Tiene un valor igual al retraso si este es positivo, en caso contrario vale 0.
- $\mathbf{E}_j$: Adelanto del trabajo j : ídem que le anterior pero con el adelanto.
- $\mathbf{U}_j$: Trabajo tardío : Esta variable toma como valor 0 si el trabajo se adelanta y 1 si se retrasa.

Las funciones objetivo mediante variables propias pueden ser de dos tipos, en función de si se miden acumulativamente o individualmente. A la hora de obtener un resultado óptimo se ha de establecer si lo que se quiere es maximizar la función objetivo o minimizarla, construyendo así un criterio de programación. Las dos tipologías de función son las siguientes:

- **Máximos/mínimos parciales**

  Esta función analiza todos los trabajos programados, y devuelve el valor máximo o el mínimo de la variable evaluada. Un ejemplo de criterio que emplee esta función objetivo puede ser encontrar la programación que logra que el trabajo con el máximo retraso tenga el mínimo posible.

- **Acumulativos**

  Este tipo de función objetivo se aplica al total de la suma de la variable para todos los trabajos. Un posible criterio con este tipo de función puede ser buscar la solución que minimice el número total de trabajos tardíos.

Estas funciones objetivo pueden ser ponderadas en función de lo importante que sea el trabajo de cara a la programación. Para ello se debe multiplicar o dividir el peso de cada trabajo por la variable a medir antes de calcular el valor de la función. Esto es especialmente importante en el caso de tener clientes con los cuales se quiera mantener la relación, por ejemplo; se puede incrementar el peso todos los pedidos de dicho cliente para evitar que se retrasen. En el ámbito quirúrgico los pesos podrían ser empleados para garantizar que se interviene primero a los pacientes de urgencia.

Una vez definidos los criterios, se puede hablar de los algoritmos que se emplean para optimizar en base a ellos. El algoritmo que obtiene la mejor solución según unos criterios se conoce como el algoritmo óptimo. Entrando en las diferentes metodologías de programación, la primera en la que uno puede pensar es en la programación manual.

Este tipo de programación no emplea algoritmo alguno y simplemente asigna los trabajos a sus correspondientes máquinas aleatoriamente o en base a la experiencia, mejorando la solución obtenida por ensayo y error. Aunque es cierto que mediante software se podrían probar las infinitas posibilidades, esto no es eficiente y, en función del tamaño del problema, puede requerir varios días, semanas o incluso años de proceso, siendo inabarcable en entornos reales.

Por este motivo se emplean unos algoritmos básicos que sirven para proporcionar una solución base, la cual se modificará empleando algoritmos más complejos. Estos algoritmos se conocen como las reglas de despacho y las más comunes son:

- **Earliest Due Date**: Asigna primero los trabajos que tienen fecha de entrega más temprana.
- **Shortest Processing Time**: Prioriza los trabajos que tienen un tiempo de proceso más corto.
- **Longest Processing Time**: Prioriza los trabajos que tienen un tiempo de proceso más largo.
- **First In First Out**: Asigna los trabajos por orden de llegada.
- **Last In First Out**: Asigna los trabajos en orden inverso al de llegada (los últimos se procesan primero).
- **Weighted Shortest Processing Time**: Igual que el *Shortest Processing Time* pero aplicado al tiempo de proceso ponderado con el peso del trabajo.

Otra manera de realizar la programación consiste en emplear algoritmos aproximados. Estos consisten en realizar unas serie de iteraciones sistemáticamente, normalmente a una solución ya existente, pero no se garantiza la optimalidad de la nueva solución obtenida. Por último, existe la metodología heurística, que son una serie de algoritmos aproximados que ofrecen mejores resultados pero de gran complejidad. Este tipo de algoritmos es el empleado por el software de planificación del hospital, pero no se va a tratar el tema con mayor profundidad, ya que no es necesario para la comprensión del presente documento.

2.3 Diagramas de Gantt

La representación de la planificación/programación se realiza tradicionalmente en los llamados Diagramas de Gantt, siendo la elaboración de uno de ellos el objetivo principal de la aplicación desarrollada. Aunque siempre representan el tiempo horizontalmente, en cuanto al eje vertical respecta, se puede distinguir entre diagramas orientados a máquina y orientados a trabajos [Framinan et al., 2014]. El primero de estos representa a las máquinas en el eje vertical y los rectángulos indican el tiempo de proceso de cada trabajo y el segundo representa los trabajos en el eje vertical y su procesado en las distintas máquinas en los distintos rectángulos, un ejemplo de estos diagramas se puede ver en la figura 2.1.

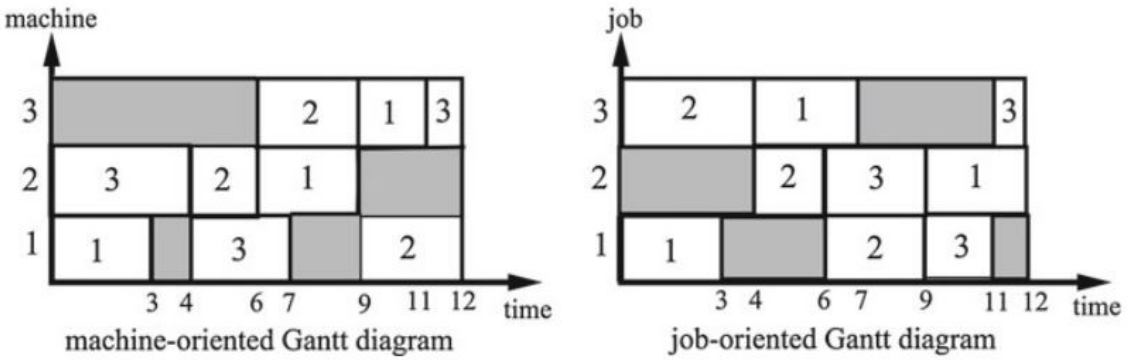

Figura 2.1 Diagramas de Gantt orientado a trabajo y a máquina, [Domschke et al., 1997].

En ambos diagramas de la figura 2.1 se pueden distinguir secuencias de operación, independientemente de si están orientados a máquinas o a trabajos. En el caso de los trabajos, esta secuencia coincidirá con su ruta de fabricación, sin embargo, la secuencia de operaciones de las máquinas

puede emplearse para realizar el diagrama de Gantt cuando la programación se realice en un único día, suponiendo que la programación es continua (inmediatamente después de procesar un trabajo comienza el siguiente). En esta herramienta informática carece de sentido emplear dicha metodología en su aplicación al entrono sanitario, puesto que los trabajos no deben pasar por todas las máquinas o incluso pueden no ser procesados.

Como se puede observar en los diagramas de la figura 2.1, el tiempo se indica únicamente cuando comienza o finaliza cada una de las operaciones. Para esta aplicación se ha decidido emplear un diagrama un poco más elaborado, en el cual el tiempo se encuentra fragmentado por franjas horarias como puede observarse en la figura 2.2. Si se desea obtener con precisión los tiempos de inicio o finalización de una operación, basta con seleccionar el trabajo en cuestión y sus datos aparecerán en un panel informativo.

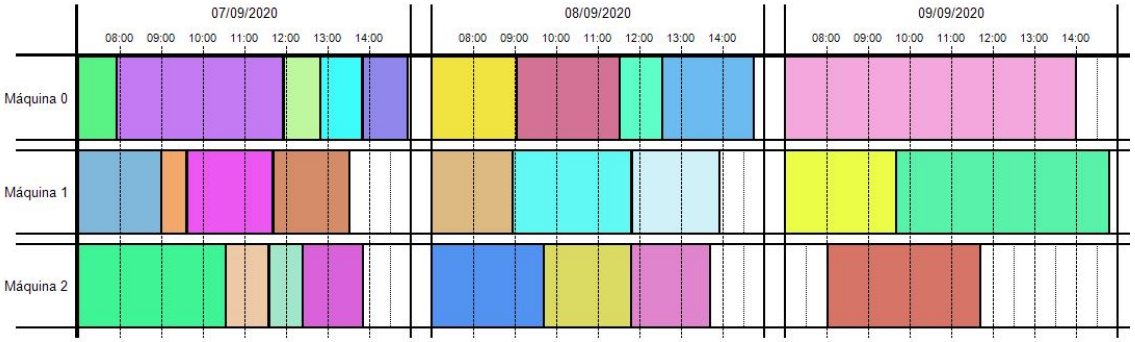

Figura 2.2 Ejemplo del diagrama de Gantt presente en la aplicación, *elaboración propia*.

3 Manual de uso

Como en toda aplicación que se precie, es necesario incluir un manual de usuario que recopile la manera de interactuar con la aplicación, y un resumen de las características de esta. En este capítulo se va a explicar desde cero todas las funcionalidades que ofrece la herramienta para que se le pueda sacar el máximo provecho.

En caso de que se desee conocer la estructura interna del código se recomienda encarecidamente la lectura del capítulo 4, el cual está dedicado a su explicación. También se analiza en ese capítulo la metodología para la elaboración del fichero de entrada, ya que esto no está relacionado con la interfaz de la propia aplicación.

3.1 Guía de la interfaz

Lo principal en una aplicación es comprender qué es posible realizar con cada uno de los controles que conforman la interfaz. En la figura 3.1 se aprecia una captura de pantalla de la aplicación, en la cual aparecen resaltadas todas las zonas de interés. Algunas de estas zonas no permitirán interacción del usuario hasta que se haya cargado un archivo de texto.

En la barra de menús se encuentra el botón de archivo (A), el cual dará opciones para guardar o cargar archivos. Debajo de este botón aparece un control de pestañas (B), el cual permite alternar entre dos listas. La primera de ellas representa los trabajos y en su zona inferior incluye un grupo de botones que permite filtrarlos en función de si estos están programados o no. En la segunda pestaña aparecerán los operarios incluidos en la programación. Cuando se seleccione un trabajo de la lista éste aparecerá resaltado en el diagrama y viceversa, pero si se selecciona un operario se resaltan todos sus trabajos asociados como en la figura 3.2. Además, cuando estos se seleccionen su información aparecerá en el panel de información de elementos (D). El contenido de las listas consiste en los nombres de los elementos que esta contiene, pero, en caso de no haber introducido ninguno, se empleará el nombre predeterminado.

A la derecha de esta lista se encuentra el diagrama de Gantt (C), el cual permite interacción por parte del usuario. Si se mantiene presionado el botón izquierdo del ratón y se arrastra, se puede navegar por el diagrama. A la hora de arrastrar se ha de tener especial cuidado, puesto que, si se comienza la operación en un trabajo seleccionado, se inicia una operación de *drag and drop*, la cual será explicada más adelante. Para seleccionar un trabajo o máquina basta con hacer clic sobre estos en el diagrama, pero no se puede arrastrar ya que esto iniciaría una operación de *scroll* y no se realizaría la selección.

Figura 3.1 Captura de la aplicación desarrollada con todas sus zonas resaltadas, *elaboración propia*.

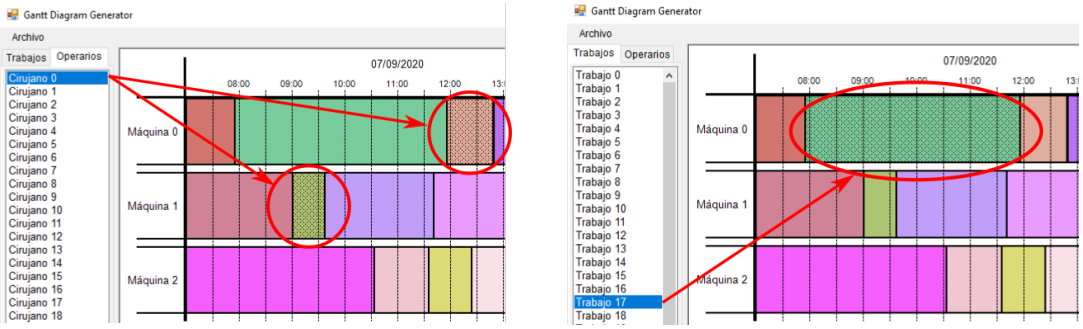

Figura 3.2 Comparación entre los efectos que la selección en la lista de un operario (izquierda) y un trabajo (derecha) tiene en el diagrama principal, *elaboración propia*.

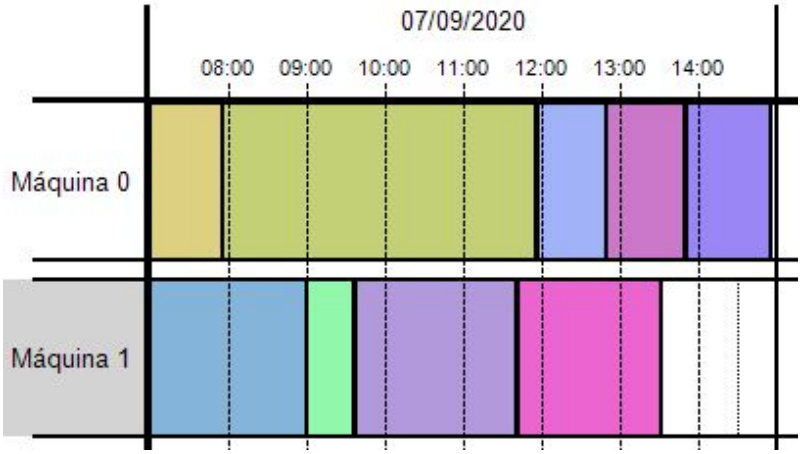

Figura 3.3 Diagrama de Gantt con la máquina 1 seleccionada, *elaboración propia*.

También cabe comentar la interacción de la aplicación con los horarios, ya que es una cualidad que no suele verse en diagramas de Gantt. En la figura 3.4 se pueden ver el rectángulos rayados que re-

presentan los momentos en los que la máquina o el quirófano están parados. Como se puede observar las operaciones se pueden programar fuera de horario siguen siendo visibles detrás de la representación de los mismos, esto indica que la herramienta permite programar manualmente fuera de horario.

Figura 3.4 Captura de la aplicación en la que se representa un diagrama con Horarios, *elaboración propia*.

Cuando cualquier elemento es seleccionado, ya sea en el diagrama o en las listas, toda su información aparece representada en el panel de información (D) ubicado en la zona inferior del mismo. La manera de presentar la información varía en función del tipo de elemento seleccionado. Como se puede observar en la figura 3.5 los paneles informativos tienen elementos comunes, como que aparezca el nombre del elemento, su identificador y su descripción, en caso de que la tenga.

La etiqueta que presenta una mayor cantidad de datos es la de los trabajos, ya que se precisa de ellos para elaborar una planificación correcta. Para cada trabajo se indica su ruta de fabricación, si es rígida o no (taller de trabajos o taller abierto) o si se trata de un modelo de máquinas paralelas. En caso de que el trabajo no esté programado en todas las máquinas se indican las máquinas restantes para facilitar la programación del mismo. A continuación se indican el número de operaciones programadas, en caso de que las tenga, seguida de un listado de dichas operaciones ordenadas cronológicamente. De cada una de estas operaciones quedarán representados en el panel de información sus fechas de inicio y fin, el tiempo de proceso y la máquina asignada. Finalmente, en caso de que tenga operarios asignados, se indicará el número de ellos seguido por el nombre de cada operario.

Las información representada para las máquinas y los operarios más reducida, pero también de utilidad para la programación. De cada máquina se muestra el tiempo total que se ha programado en ellas y el tiempo disponible para la programación, el cual consiste en el tiempo que la máquina se considera abierta, esto permite conocer el grado de utilización de cada máquina. Para el caso de los operarios únicamente se indica el número de trabajos que tienen asignados.

A la derecha del panel de información de los elementos existen dos zonas, en la superior (E) aparece información genérica sobre la programación y, en la inferior (F), una barra que permite

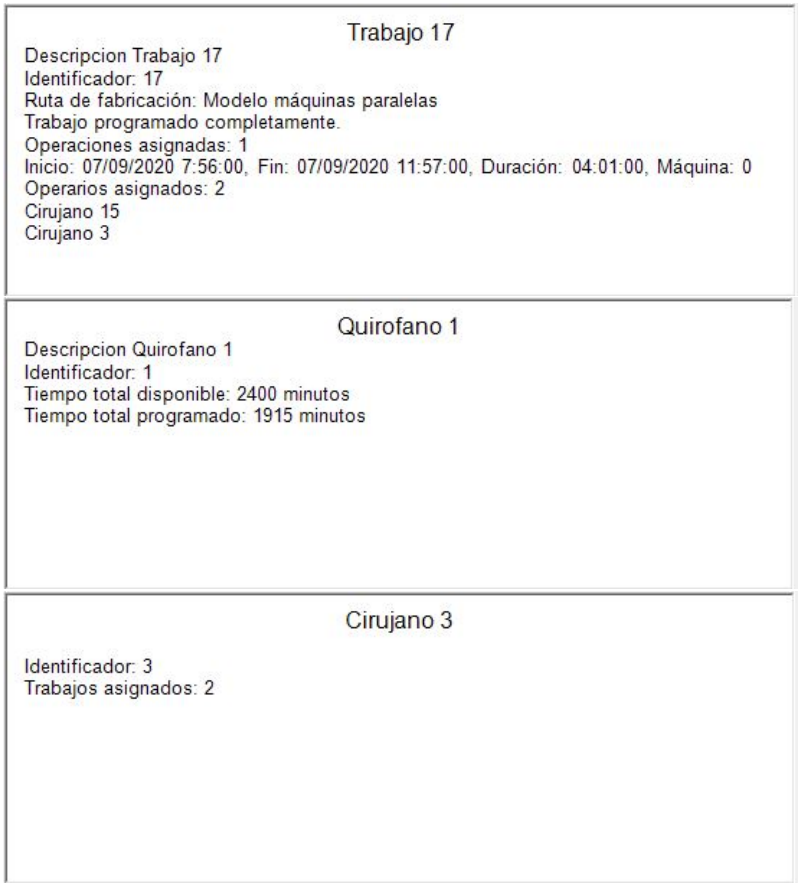

Figura 3.5 Plano detalle del panel de información de los distintos elementos del diagrama en función de cuál de ellos esté seleccionado, *elaboración propia*.

realizar zoom desplazando su marcador. El panel de información genérica permite conocer el número total de pacientes tanto asignados como no asignados. Además, también muestra el tiempo total programado junto al disponible, de esta forma se puede extraer una estimación del aprovechamiento de los recursos. En la figura 3.6 se representa una captura detallada de este panel informativo.

Figura 3.6 Plano detalle del panel de información general de la programación, *elaboración propia*.

Debajo de este panel se encuentra la barra de zoom, la cual tiene un amplio rango de valores, permitiendo así representar desde una semana en el ancho del diagrama hasta únicamente unas horas. Para poder programar con precisión de minutos no es necesario emplear el zoom al máximo, con un valor intermedio debería de ser suficiente. Sin embargo, en caso de que existan trabajos de la mínima duración admitida (1 minuto), se necesita ampliar el diagrama todo lo posible para poder hacer clic en los mismos y modificar su programación. Un ejemplo de este fenómeno puede apreciarse en la figura 3.7, donde, si no existiese el zoom, sería imposible seleccionar la operación

del trabajo 2 en la máquina 1.

Figura 3.7 Capturas de la aplicación que muestran la importancia del zoom para la selección de operaciones, *elaboración propia*.

Por último, cabe mencionar que todas las zonas explicadas hasta ahora, salvo estas dos últimas, disponen de separadores que pueden ser modificados a gusto del usuario. En la figura 3.8 se representan dos posibles configuraciones de la aplicación, una que maximiza el ancho de la lista, para poder leer los nombres completos, y otra que prioriza el tamaño del diagrama, con el fin de facilitar las modificaciones en la programación.

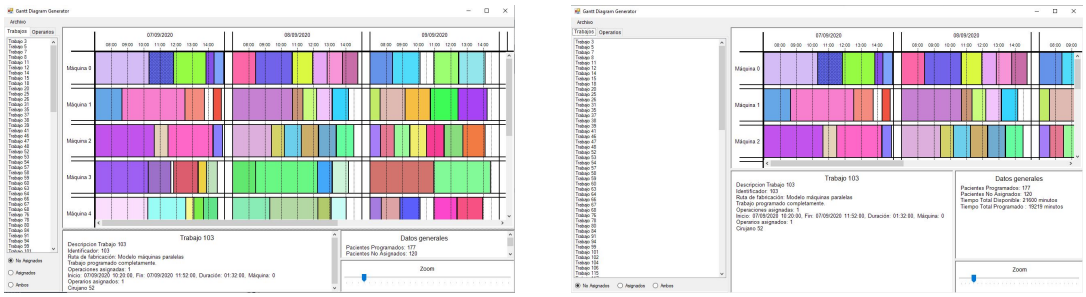

Figura 3.8 Comparación entre dos configuraciones posibles para la aplicación, una que prioriza el tamaño del diagrama (izquierda) y otra que prioriza la lista y los paneles de información (derecha), *elaboración propia*.

3.2 Cargar y guardar archivos

La carga de un archivo es la primera acción que se tiene que realizar al abrir la aplicación. Primero se ha de pulsar en el botón de archivo en la esquina superior izquierda de la pantalla, donde aparecerá un desplegable. Después se debe seleccionar la opción archivo en el desplegable, abriéndose un cuadro de diálogo que permitirá seleccionar el archivo de texto deseado. Comentar que el directorio inicial del cuadro de diálogo, tanto para guardar como para cargar, es la carpeta FicherosEntrada que se encuentra junto a los archivos de la aplicación.

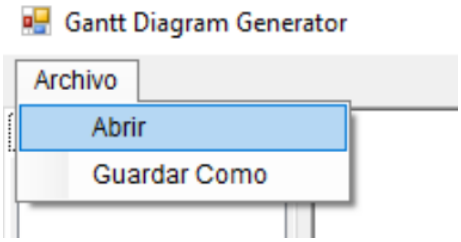

Figura 3.9 Plano detalle de la opción abrir de la aplicación desarrollada, *elaboración propia*.

Si el proceso se ha realizado correctamente deberá de aparecer el diagrama de Gantt ya representado. En caso de error de lectura en el archivo aparecerá un mensaje desplegable diciendo la línea del archivo de texto en la que se ha producido el fallo, pero el código se seguirá ejecutando. Esto da una pista al usuario del motivo del fallo en la lectura. Los errores más comunes son:

- Introducir un espacio o un carácter ilegal donde debería de estar una palabra clave.
- Fallo ortográfico en las palabras clave, las cuales deben coincidir en acentos y mayúsculas o minúsculas.
- La propiedad anterior a la línea del fallo tiene un número de lineas distinto al necesario.
- Inclusión de una propiedad sin desarrollar o de la clase Operario sin emplearla.

Comentar que la aplicación no dispone de protecciones que revisen el archivo de entrada, ya que en un principio la planificación introducida proviene de un software especializado de alta fiabilidad. Por este motivo, al introducir el archivo se permite que dos trabajos se solapen en una máquina o estén fuera de horario. En caso de que se introduzca un archivo elaborado artesanalmente se recomienda revisar que estos fenómenos no se hayan producido.

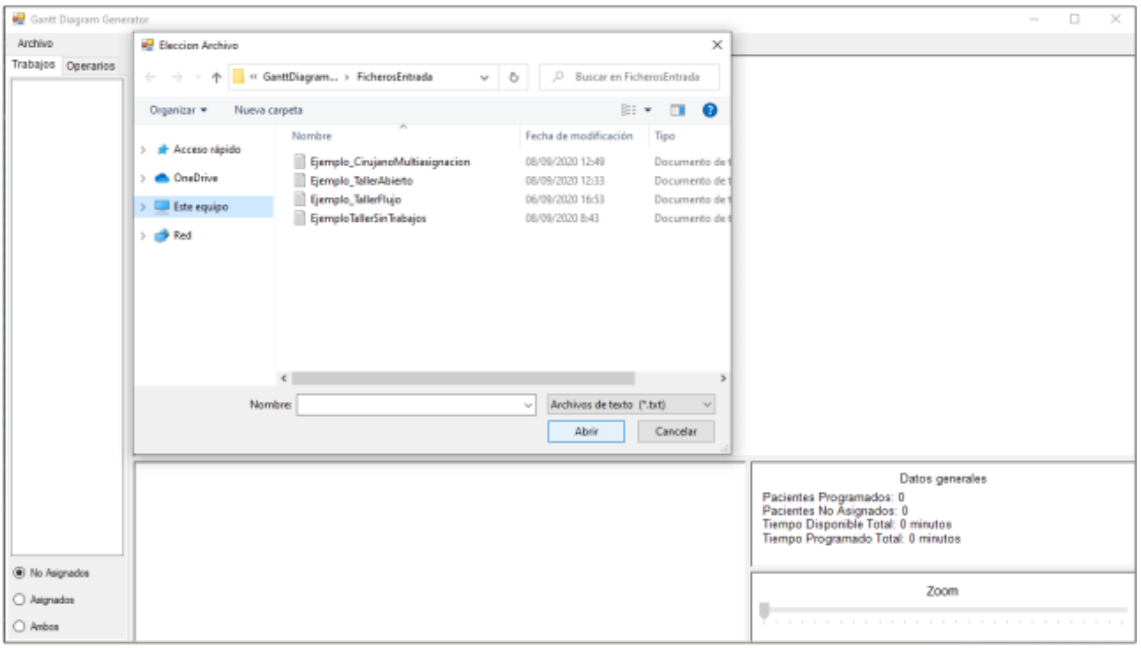

Figura 3.10 Ejemplo de cuadro de diálogo para la selección de archivo en la aplicación desarrollada, *elaboración propia*.

El proceso para guardar el archivo es igual de sencillo, se accede al mismo menú desplegable y se selecciona la opción de guardar como. En el cuadro de diálogo que aparece se debe escribir un nombre para el archivo, si se desea crear uno nuevo, o seleccionar un archivo existente para sobrescribirlo. Una vez realizada esta operación el archivo activo pasa a ser el guardado.

3.3 Modificación de la programación

Por último queda comentar como se han de realizar las acciones *drag and drop* para modificar la planificación. La primera condición que se ha de cumplir es la de que el trabajo esté seleccionado, ya sea para iniciar el *drag and drop* en la lista o en el diagrama. Una vez este está seleccionado, se debe pulsar el botón izquierdo del ratón sobre el trabajo seleccionado y desplazarlo manteniendo el botón presionado, lo cual producirá que un rectángulo con el tamaño y el color del trabajo al cual representa aparezca bajo el cursor y lo persiga. Esto queda representado en la figura 3.11 En el caso de comenzar el *drag and drop* desde la lista, se ha de llevar el ratón hasta la zona del diagrama para que aparezca el rectángulo.

Para listas también existe otra condición, la de que el trabajo no esté programado completamente, para conocer si la cumple o no, se puede mirar en el panel de información o filtrar la lista por trabajos no programados. En caso de que se intente programar un trabajo que ya lo está, aparecerá un mensaje en la pantalla. Destacar que si en la lista se selecciona un operario, puede parecer en el diagrama que el trabajo está seleccionado, cuando solamente aparece resaltado. Para comprobarlo basta con mirar en el panel de información, buscar si hay más de un trabajo resaltado o hacer clic de nuevo en el trabajo para asegurar su selección.

Durante la ejecución se pueden apreciar ligeros cambios en la aplicación, todos ilustrados en la figura 3.12, donde se está modificando el trabajo de color rosa. El cambio más llamativo de ellos es que mientras el ratón este presionado el cursor no podrá salir del rectángulo constituido

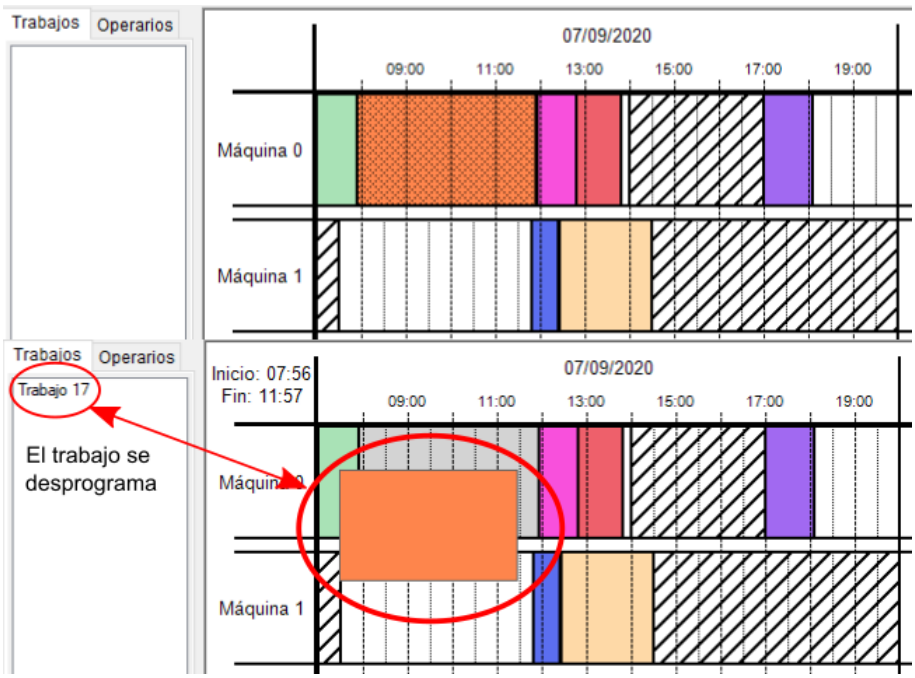

Figura 3.11 Captura que muestra cómo se comienza un evento *drag and drop* en la aplicación con la consiguiente desprogramación del trabajo seleccionado, *elaboración propia*.

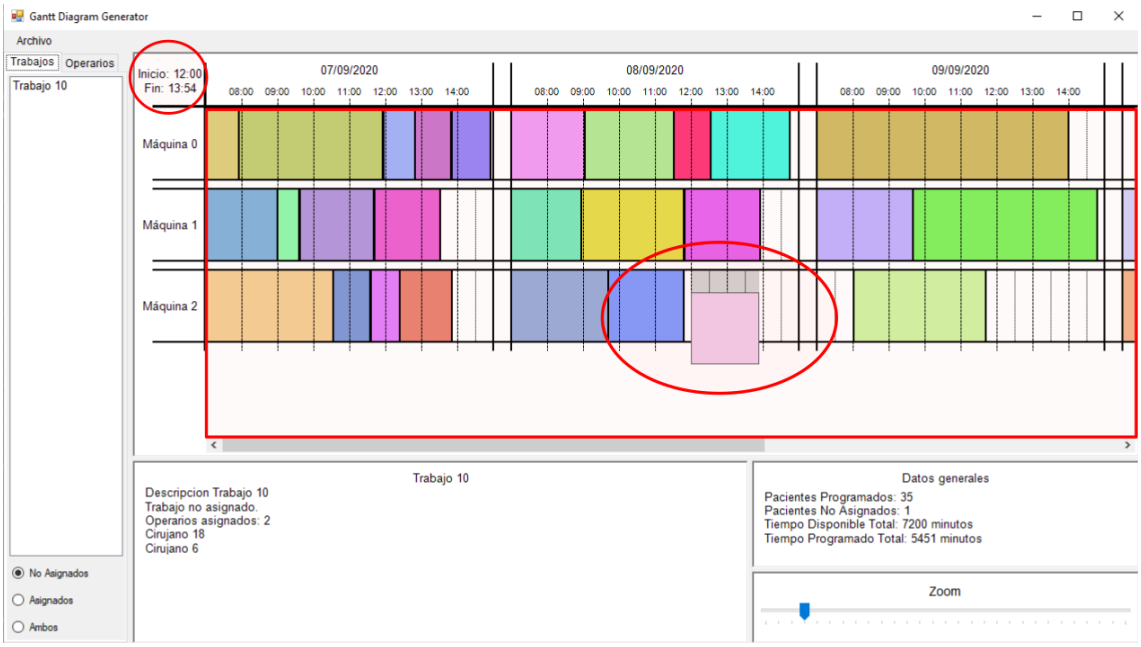

Figura 3.12 Captura de la aplicación durante el evento *drag and drop*, *elaboración propia*.

por el diagrama (rectángulo rojo). Además, en caso de aproximarse a los bordes el diagrama se desplazará en esa dirección, facilitando la colocación del trabajo. Cuando el rectángulo se desplace sobre las diferentes máquinas, irá variando de tamaño en función del tiempo de proceso del trabajo seleccionado en la máquina sobre la que se encuentre. Además, a modo de ayuda, en la esquina superior izquierda del diagrama aparecerán las horas de inicio y de fin del trabajo en la posición en la que se encuentre.

3.3.1 Validación de las nuevas fechas y eliminación de operaciones

Una vez entendidas las modificaciones en la aplicación, es necesario estudiar los criterios que emplea la aplicación para validar la nueva programación. Tras iniciar el *drag and drop*, la operación del trabajo seleccionado se elimina, por lo que aparecería en la lista de trabajos no asignados como puede verse indicado en la figura 3.11. En caso de que el trabajo esté programado parcialmente o se incorpore desde la propia lista, no se produciría ningún cambio en la misma. Cuando se ha hecho esto se puede desplazar el trabajo libremente por el interior del diagrama. Conforme se coloca el trabajo en distintas posiciones, se puede apreciar que aparece un rectángulo sombreado representado en el diagrama, el cual indica las zonas donde se puede colocar el trabajo en cuestión. Además, siempre que este aparezca representado, la ayuda de la esquina superior izquierda también lo hará. En la figura 3.13 aparece el rectángulo sombra que será el destino del trabajo en caso de querer realizar la asignación en ese momento.

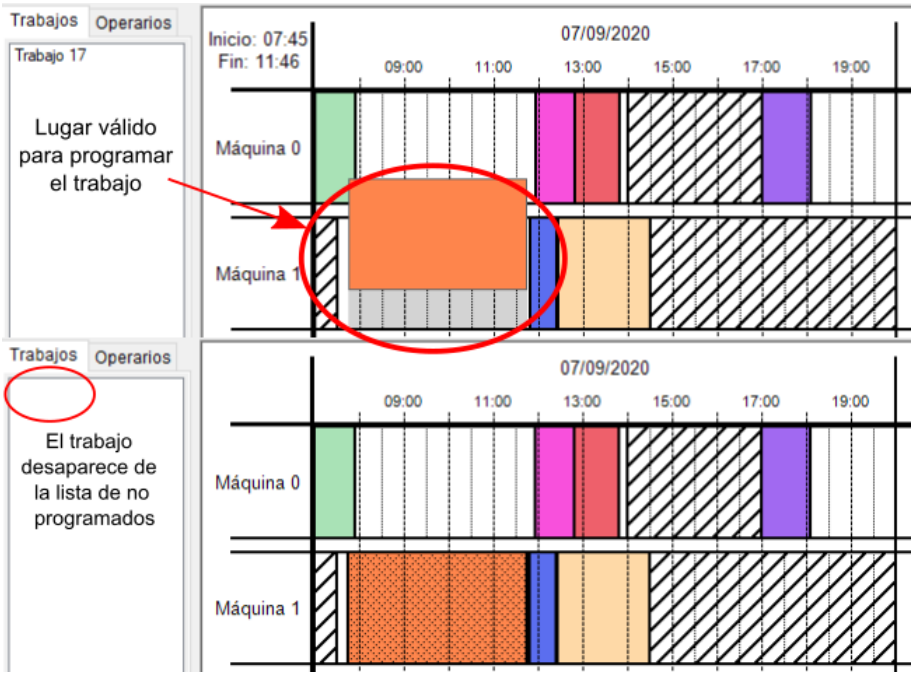

Figura 3.13 Captura que muestra el proceso de asignación de un trabajo una vez se ha iniciado el *drag and drop*, *elaboración propia*.

En caso de que se libere el ratón mientras el rectángulo sombreado no está dibujado, el trabajo quedará eliminado de la programación, como queda ilustrado en la figura 3.14. Aunque la forma de eliminar los trabajos más intuitiva puede ser el desplazar el trabajo hasta la lista, esto no es posible, ya que los bordes del diagrama se emplean para la implementación de su *scroll*. Por este motivo se incluye a continuación una lista de posiciones en las cuales el rectángulo sombra no aparecerá, para que sea más sencilla la eliminación de operaciones:

- El trabajo se está intentando programar entre dos días diferentes.
- El trabajo se está intentando programar en una máquina que ya tiene operaciones programadas en el intervalo temporal deseado.
- El trabajo se está intentando programar en una máquina en la que ya tiene una operación asignada.
- El trabajo se está intentando programar en una máquina, la cual se encuentra en una posición posterior/anterior en su ruta de fabricación, siendo la fecha de este anterior/posterior a la

operación ya programada en la otra máquina. Esta restricción solamente está activa para el caso de taller de trabajos o taller de flujo.

- El trabajo se está intentando programar en un intervalo temporal en el que ya esta programada una operación del mismo en otra máquina. Como es lógico el modelo de máquinas paralelas no esta sujeto a esta restricción.
- El trabajo se encuentra fuera del diagrama, ya sea en la zona inferior o en el margen derecho.

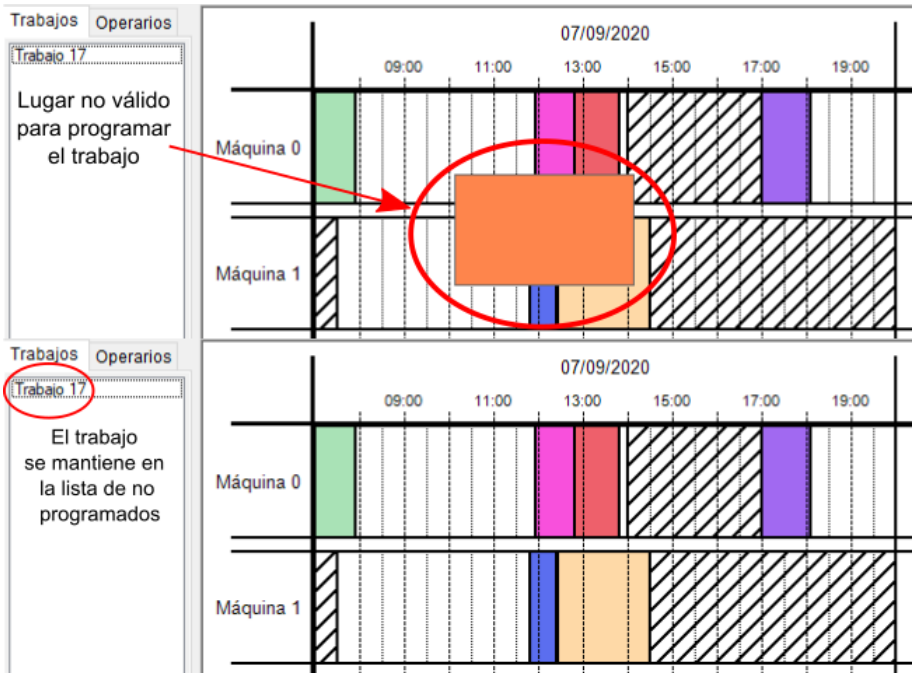

Figura 3.14 Captura que muestra el proceso de eliminación de un trabajo una vez se ha iniciado el *drag and drop*, *elaboración propia*.

3.3.2 Ayudas a la programación

Como se puede deducir de la lista de restricciones, programar dos trabajos consecutivos es una tarea complicada, ya que si se solapan aunque sea por un minuto el rectángulo sombra deja de aparecer. Por este motivo se han incorporado una serie de ayudas para programar los trabajos, las cuales son aplicables tanto para programar un trabajo al inicio o fin del día como continuación de una operación ya programada, ya sea del mismo trabajo en otra máquina o de otros trabajos en la máquina deseada. Este sistema de ayudas se puede ver representado en la figura 3.15 y básicamente consiste en representar el rectángulo sombra en la posición más cercana permitida, la cual se obtiene haciendo un barrido de posiciones dentro de unos límites respecto a la posición actual del trabajo que está siendo programado.

3.4 Ejemplos

En esta sección se pretenden incorporar distintos ejemplos que reflejen las posibilidades de diagramas de Gantt que la aplicación puede representar. No se entrará en detalle en los archivos de entrada puesto que esto se comentará en el capítulo de estructura del código. La aplicación puede representar diagramas para las distribuciones en planta más comunes, como puedan ser máquinas paralelas (figura 3.16), taller de flujo (figura 3.17) o taller abierto (figura 3.18). También se puede representar

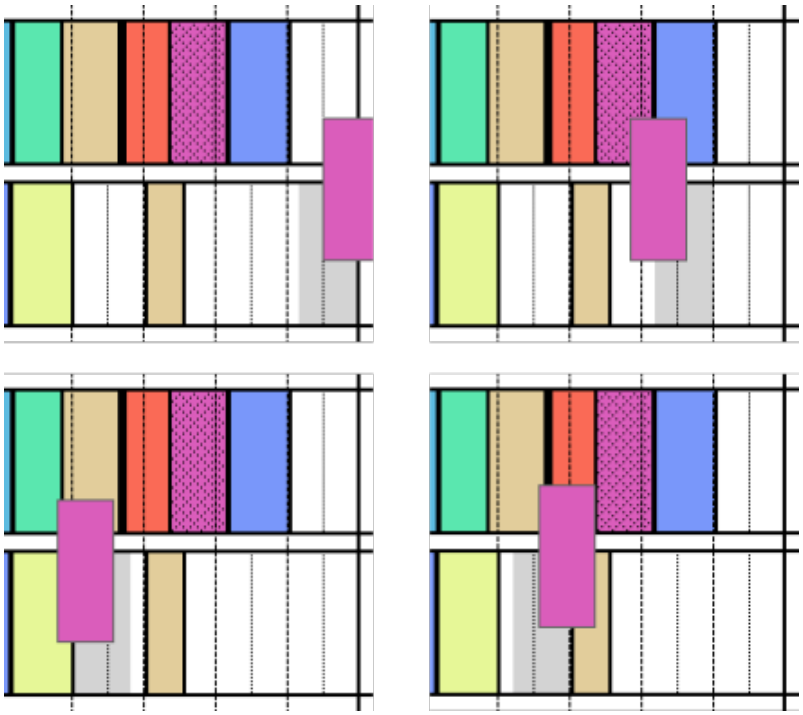

Figura 3.15 Ilustración de las diferentes ayudas a la programación incorporadas en la aplicación, estas son el ajuste al fin/inicio del día (arriba a la izquierda), ajuste con respecto a otras operaciones del mismo trabajo (arriba a la derecha) y ajuste con respecto a otras operaciones (abajo), *elaboración propia*.

un taller de trabajos, pero este es similar al ejemplo de taller abierto (ya que lo único que varía es que la ruta no se puede modificar) y por tanto se ha omitido como ejemplo.

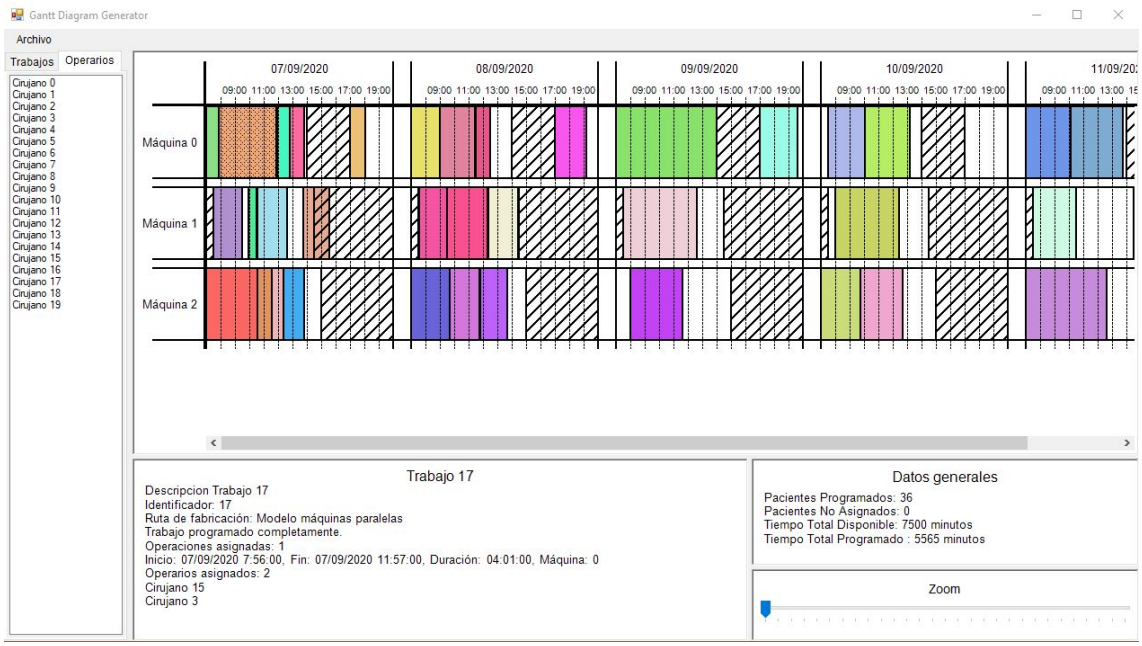

Figura 3.16 Ejemplo de un diagrama de Gantt de un modelo de máquinas paralelas producido por la aplicación desarrollada, *elaboración propia*.

Figura 3.17 Ejemplo de un diagrama de Gantt de un taller de flujo producido por la aplicación desarrollada, *elaboración propia*.

Figura 3.18 Ejemplo de un diagrama de Gantt de un taller abierto producido por la aplicación desarrollada, *elaboración propia*.

4 Estructura del código

Una vez aclarados los conceptos básicos de la programación de la producción y las funcionalidades de la aplicación, se puede proceder con el análisis de las decisiones de diseño tomadas para su desarrollo. A lo largo de este capítulo se expondrá la estructura de clases empleada para el *backend* de la herramienta, así como de los controles personalizados necesarios para la representación de los datos. También se dará una explicación del código empleado para la lectura del archivo de texto así como instrucciones sobre cómo elaborar la estructura del mismo.

Asimismo, se explicarán las ventajas de la solución escogida frente a otras posibilidades existentes y se comentará el motivo detrás de las funcionalidades adicionales incorporadas en la herramienta. Como la extensión del código es considerable, se ha optado por no comentar cada una de las funciones programadas con detalle, pero se dejará indicado una breve descripción del funcionamiento de las más relevantes.

En caso de que se quiera experimentar con el código, o ver en detalle la manera de operar de una función, se anexará todo el código al final del documento. Además, de esta manera, un futuro programador podrá emplear este documento como punto de partida para las labores de mantenimiento de la herramienta o añadir nuevas funcionalidades.

El nivel de las explicaciones que se encontrarán a continuación asume que se conocen tanto el lenguaje C# como la herramienta Windows Forms a nivel usuario. En el caso de estar familiarizado con otros lenguajes POO y desarrollo de interfaces gráficas, la comprensión de los conceptos de este capítulo no debe de ser complicada. En caso de que no sea así, se recomienda una lectura previa de algún manual o tutorial del lenguaje y, en caso de que surja alguna duda de terminología, se acuda a la documentación oficial de C# proporcionada por Microsoft.

4.1 Estructura de clases

Una estructura de clases bien planificada es esencial para reducir el volumen total de líneas de código de cualquier aplicación informática, y en este caso no iba a ser diferente. Por este motivo se ha optado por crear una clase para cada elemento productivo, una para los trabajos, otra para las maquinas y otra para los operarios. De esta manera, cada uno de los objetos de las clases tiene una serie de propiedades como puedan ser su nombre o una breve descripción de sí mismos, en lugar de emplear vectores para almacenar los datos.

Este enfoque se aleja considerablemente del tradicional, en el que se trabaja únicamente con matrices (una con los tiempos de proceso y otra con las fechas de inicio o fin), permitiendo almacenar

una mayor cantidad de información de cada objeto sin la necesidad de depender de su ubicación dentro de una matriz. De esta forma añadir o eliminar trabajos o máquinas en tiempo de ejecución se vuelve considerablemente más sencillo, en el caso de que se implementase esta funcionalidad, y, además, facilita la comprensión del código por parte del lector. Cada una de estas clases dispone de una lista estática en la cual se almacenan todas los objetos de esta clase, así se tiene acceso en todo momento a cada una de las instancias de cada clase, por lo que con un comando se puede obtener toda la información que se desee.

La estructura propuesta pretende ser lo más similar posible a la realidad, ya que en lugar de disponer de una matriz con números, se busca tener una lista de Trabajos, unas entidades de las que se puede obtener todas la información necesaria para la representación del diagrama. Esta información puede ser desde las operaciones que cada Trabajo tiene asignado hasta el color que se va a emplear para representarlo en el gráfico.

Además, existen dos clases contenidas en el interior de las principales, estas son la clase Horario, contenida en la clase Máquina, y Operación, contenida en la clase Trabajo. Se ha decidido diseñarlo de esta forma para reducir el número de líneas de código, ya que en el interior de cada trabajo se almacenan las operaciones que este tiene asignado y, para el caso de las máquinas, se incluye en su interior el horario de cada día. Todas las clases principales implementan una interfaz llamada IDatosIdentificativos, esta interfaz permite aplicar programación genérica a los métodos de lectura de datos y escritura facilitando la reutilización del código. Esta interfaz fuerza la implementación de las siguientes propiedades:

- **Identificador**

  Valor numérico entero (aunque podría ser de tipo *string*) que permite diferenciar inequívocamente a un trabajo del resto. Como valor predeterminado se toma el entero que representa la posición del objeto dentro de la lista de objetos de su misma clase (comenzando por el 0), pero podría emplearse otro criterio, como un identificador único asociado a la orden de producción o un DNI. El principal problema de emplear otro identificador, como el pueda ser el DNI es que puede darse el caso de que al mismo paciente se le realicen dos operaciones diferentes separadas en el tiempo y que se introduzca dos veces en el sistema, dejando de ser el DNI único para ese trabajo.

- **Nombre**

  Cadena de caracteres por la cual se va a referir al objeto de la clase en caso de que el valor no sea el predeterminado. El valor predeterminado es Nombre de clase + identificador, por lo que el primer trabajo tendrá de nombre (Trabajo 0). Este nombre puede ser el nombre real del paciente, del cirujano o, en caso de los quirófanos, el nombre que se le dé en la planta del hospital. Se ha sobrescrito el método ToArray() de estas clases para que devuelva el nombre, de esta manera en las ListBox empleadas aparece escrito el nombre correctamente en caso de que se posea.

- **Descripción**

  Cadena de caracteres que aporta información adicional acerca del objeto en cuestión. A diferencia de las otras dos, esta propiedad no tiene un valor predeterminado y está programada como un recurso que pueden tener los planificadores para hacer aclaraciones sobre los distintos elementos de la planta. Una posible descripción incluiría comentarios del historial clínico del paciente a tener en consideración, remarcaría datos de los cirujanos o dejaría constancia de comentarios sobre los quirófanos.

El hecho de que estos datos se almacenen como propiedades es altamente ventajoso, principalmente por que la inclusión de propiedades nuevas como puedan ser el DNI o número de la Seguridad

Social de los pacientes es trivial. Esto se verá con más detalle en la sección 4.2, ya que es donde se va a estudiar la manera de introducir todos los datos en la aplicación.

Como último apunte del funcionamiento de las clases principales, se va a comentar el mecanismo de inclusión de estos en sus respectivas listas estáticas. En cada constructor de estas clases existe una línea de código que añade la nueva entidad creada a una lista con todas las existentes, ubicándola al final de la lista. Antes de agregarla se le asigna su identificador que equivale al tamaño de la lista previo a la inclusión del nuevo objeto. Para obtener cada uno de los objetos basta con llamar a cualquiera de estos métodos:

- **GetArray()**

  Devuelve un array con todos los objetos de la clase principal en la cual se está aplicando. Por ejemplo, si se escribe Trabajo.GetArray() se obtendría un vector con todos los trabajos incorporados hasta ese momento que no hayan sido eliminados.

- **Get(*int* identificador)**

  Devuelve la objeto de clase con el identificador introducido, si el identificador es igual a la posición de los elementos en la lista, sería equivalente a escribir GetArray()[identificador].

- **Clear()**

  Elimina todos los elementos de la clase en cuestión de su lista integrada.

Como se puede observar obtener los vectores clásicos de la planificación de la producción se antoja bastante sencillo, así como obtener cualquier instancia de la cual se conoce el identificador. Comentar que se devuelve un vector con los objetos de la clase en lugar de la lista en sí misma para evitar que se añadan o eliminen objetos de esta empleando código externo a la clase en cuestión.

Por último, todas estas clases implementan una interfaz llamada ISeleccionable, la cual obliga a implementar la propiedad booleana Seleccionado. Esta propiedad es utilizada para identificar un elemento del resto, ya sea para resaltarlo en el diagrama o para que aparezca su información en el panel de información. Además, en la implementación de esta propiedad, se ha forzado que al seleccionar un elemento el elemento anterior que estaba seleccionado deje de estarlo.

Teniendo la estructura básica clara, se puede entrar en detalle en las clases Trabajo y Máquina, que son notablemente más complejas que la clase Operario, la cual solamente incorpora las cualidades comunes a las tres. Se planteó la posibilidad de asignar un rendimiento a los operarios, pero fue descartada ya que al permitir la asignación de varios operarios a un trabajo, esto carecía de sentido. Esto se debe a que en una operación intervienen varios cirujanos, por lo que en la implementación de la aplicación era necesario habilitar esta funcionalidad.

4.1.1 Trabajo

La clase trabajo es la más extensa de las empleadas ya que es la que más información almacena en su interior. El principal dato que almacena esta clase es el campo de clase estático staticOrigenProgramacion el cual almacena la fecha en la cual se comienza a dibujar el diagrama. Para crear una instancia de esta clase la única variable imprescindible que exige el constructor es la ruta de fabricación. Este dato no se emplea para elaborar el diagrama durante la lectura del archivo pero sí impide que el usuario realice cambios en el diagrama representado que incumplan esta ruta.

La ruta consiste en un vector con los identificadores numéricos de cada una de las máquinas en el orden que han de procesar al trabajo en cuestión. Por ejemplo, si la ruta es (2 ,3 ,1) el trabajo se

procesa primero en la máquina 2, después en la 3 y por último en la 1. Esta manera de introducir de ruta es idónea para modelar los talleres tanto de de flujo como de trabajos, pero deja de lado el resto de modelos de planta. Para poder modelar un taller abierto se permite introducir los números pero con el signo cambiado. Esto indicaría que el trabajo ha de pasar por esas máquinas pero no en ese orden. De este modo la secuencia (-2, -3, -1) indica que el trabajo debe procesarse en las máquinas 2, 3 y 1 pero podría comenzar por la máquina 1 si así lo desease el usuario.

Aún con esto no se puede modelar el caso de interés para el quirófano, el cual consiste en un modelo de máquinas paralelas. Para modelarlo se debe introducir una secuencia de una única máquina con el signo negativo, independientemente del número. Esto no entra en conflicto con la manera de introducir una secuencia para taller abierto, ya que no tiene sentido programar una única máquina e indicar que el orden es indiferente. La ruta de fabricación (-1) indica que el trabajo se procesa una única vez en cualquiera de las máquinas, esto funcionaría igual para cualquier dígito negativo. En caso de que la secuencia introducida contenga signos positivos y negativos el programa automáticamente los considerará todos como negativos. Si la programación asignada al trabajo cumple con su ruta de fabricación existe una propiedad booleana llamada Programado, que devuelve el valor *true*.

Se ha decidido no incluir plantas más complejas como entornos híbridos debido a que complicaban notablemente el código y lo que aportan no compensa al tiempo de desarrollo que se ha de invertir. El objetivo final de esta aplicación es gestionar quirófanos (máquinas paralelas) o a lo sumo servir como herramienta académica, donde los modelos son sencillos para facilitar la comprensión por parte de los alumnos.

En la creación de los trabajos también se permite la introducción de una serie de fechas de inicio o finalización de las operaciones del trabajo en cada una de las máquinas. En caso de que no se desee planificar el trabajo en una de las máquinas la fecha introducida deberá ser anterior a la de origen de programación. La introducción de estas fechas será explicada en la sección 4.2.

Clase Operacion

La clase Operación se se encuentra en el interior de la clase Trabajo, ya que técnicamente no puede existir una operación sin un trabajo asociado. Esta clase permite crear operaciones para un trabajo determinado introduciendo la máquina en la cual se va a procesar el trabajo (de donde se obtendrá el tiempo de proceso) y su fecha de inicio o finalización.

Para evitar que se creen operaciones fuera de la clase Trabajo, la única forma de acceder a sus constructores es mediante el método AddOperacion(*var args*) de la clase trabajo. Este método toma como argumentos los datos necesarios para crear la operación salvo el identificador del trabajo, ya que se obtiene del propio objeto. Además, el método añade automáticamente la operación a una lista interna que posee cada instancia de la clase trabajo, para que sea fácil de identificar y las ordena cronológicamente.

De cada operación se pueden extraer los datos más relevantes de la misma, siendo estos las fechas de inicio y fin, el tiempo de proceso y la máquina en la que se procesa, empleando diferentes métodos. Al tratarse de una clase privada dichos métodos no se pueden emplear fuera de la clase Trabajo, por este motivo para obtener estos datos se han de crear unos métodos intermedios en esta clase. Estos métodos necesitan de argumento el orden de la operación dentro de la lista interna que tiene el trabajo y devuelven el valor cuyo nombre indica, los cuales son:

- GetInicioOperacion(*int*)

- GetFinOperacion(*int*)
- GetTiempoProcesoOperacion(*int*)
- GetIDMaquinaOperacion(*int*)

Adicionalmente existe un método que permite eliminar una operación dado su orden en la lista y un método que devuelve el número de operaciones asignadas a un trabajo, necesario para hacer bucles. Por último, existe un método llamado CheckOperacionValidaTrabajo(*var args*) el cual toma como argumentos una fecha de inicio/fin, un tiempo de proceso y una máquina y devuelve un booleano que es verdadero si la operación se puede programar para ese trabajo y falso en caso contrario. Este método recurre a otros pero estos no son relevantes para el funcionamiento de la aplicación.

4.1.2 Propiedades y métodos

Una vez aclarada la gestión de las operaciones en el interior de la clase trabajo solamente queda comentar el resto de métodos y propiedades que tiene programadas en su interior.

Existen una serie de métodos estáticos que devuelven el número total de trabajos no programados y programados, para representarlos en el panel de información general. Para la gestión de los operarios asignados a cada trabajo se ha incorporado una lista interna. Esta lista contiene todos los operarios asignados al trabajo en cuestión y es diferente para cada uno de los objetos de la clase. También se dispone de un método devuelve la lista como vector y otro que le añade un operario.

Para la representación gráfica de los trabajos existe una propiedad llamada ColorDibujo que devuelve el color con el que debe ser representado el trabajo. En caso de que sea la primera vez que se llama a la propiedad se crea un color nuevo automáticamente y se asigna al trabajo para su futuro uso.

La creación de este color se realiza de forma aleatoria bajo una serie de restricciones. La primera de ellas es que tenga un brillo superior a 0.6 e inferior a 0.95, de esta forma se evitan colores demasiado oscuros o claros para ser identificados en el diagrama. Además, se fuerza a que la saturación de los mismos sea superior a 0.4 para evitar la gama de grises ya que estéticamente es menos agradable.

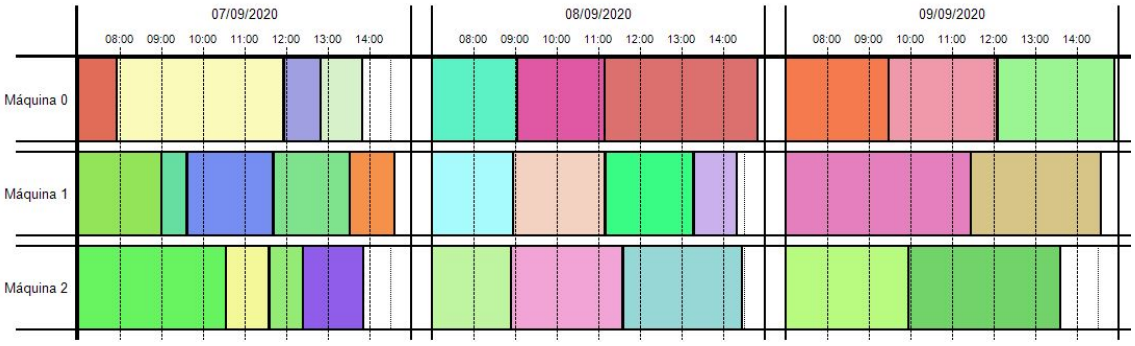

Figura 4.1 Gama de colores representada en un diagrama de Gantt, obtenida empleando el método de generación integrado en la aplicación, *elaboración propia*.

La repetición de colores se evita obligando que los colores se alejen un valor prefijado del resto de colores empleados, esta "distancia" debe cumplirse en cualquiera de los colores RGB. En caso de que se prueben más de 5000 colores y no se encuentre ninguno válido se reduce esa "distancia"

entre los colores. En caso de agotar todos los colores diferentes se notifica al usuario. En la figura 4.1 se puede observar la variada gama de colores que se obtiene con este método.

Por último, existe una propiedad booleana llamada Resaltado que indica si el trabajo ha de ser pintado con un elemento diferenciador al resto. La diferencia entre un trabajo resaltado y otro que no lo está se puede apreciar en la figura 4.2. Esta propiedad es verdadera si el trabajo en cuestión o algún operario de su lista está seleccionado.

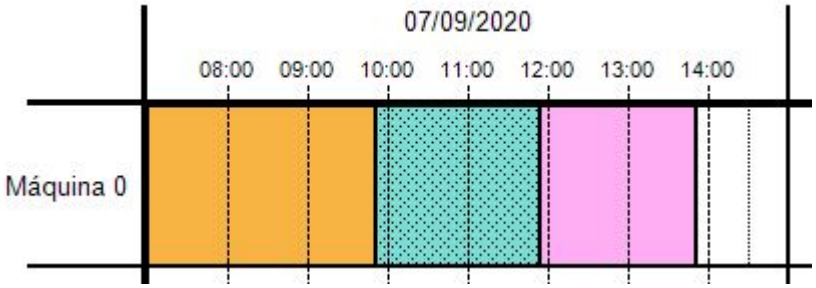

Figura 4.2 Comparación entre la representación gráfica de un trabajo seleccionado (trabajo central) en la aplicación frente al resto, *elaboración propia*.

4.1.3 Maquina

La clase máquina es menos compleja que la clase Trabajo, pero aun así tiene una cierta cantidad de métodos relevantes, especialmente la manera de guardar y gestionar los horarios. Esta clase dispone de un campo estático que establece el número de días en los cuales se va a programar, esto es fundamental ya que determina la longitud que tiene que tener el vector de horarios. Los constructores de esta clase solicitan un vector de enteros el cual representa los tiempos ideales de proceso de cada uno de los trabajos en esa máquina. Estos tiempos de proceso se pueden obtener empleando los métodos GetTiempoProcesoIdeal(*int*) y GetTiempoProcesoIdeal(*int*), los cuales toman como argumento el identificador del trabajo en cuestión, el primero de ellos devuelve los tiempos sin aplicar el rendimiento de la máquina en cuestión y el segundo los aplica. Este rendimiento es una propiedad la cual se introduce mediante el fichero de entrada.

En este constructor se crean los horarios de la máquina, los cuales comienzan con la máquina cerrada todos los días. La clase Horario representa el horario que tiene una máquina en un día concreto el cual dispone de una propiedad booleana con el nombre Abierto, que indica si la máquina está abierta o cerrada, independientemente de las horas de apertura o cierre introducidas.

Para cambiar el horario de un día se emplea el método SetHorario(*var args*), el cual toma de argumentos un entero, que representa al día en el cual se quiere alterar el horario y un vector de elementos TimeSpan que describen las horas de apertura y cierre del horario. La manera de estructurar este vector es la siguiente, se escribe la hora de apertura del quirófano seguido de una hora de cierre tantas veces como segmentos de apertura haya. Por ejemplo el vector (8,5 14 17 20) indica que el quirófano abre de 8 y media a 2 y después tiene un turno de tarde de 5 a 8. Si el vector introducido está vacío, la máquina se cierra automáticamente y se abre en caso contrario.

Existen una serie de métodos en la clase máquina que permiten obtener las aperturas o los cierres de un día concreto y son empleados para dibujar los horarios en el diagrama. Finalmente se han elaborado métodos que obtienen el tiempo programado y el tiempo libre disponible, ya sea para una

máquina o para todas las existentes.

4.2 Método de introducción y extracción de datos

Para la representación del diagrama de Gantt es esencial establecer un método de introducción de datos en la aplicación. Se ha buscado una manera de generar un archivo pueda ser interpretado por una persona además de por el ordenador. Por este motivo se ha optado por elaborar un fichero que emplee palabras clave (*keywords*), que permitan interpretar los números que aparecen en el fichero de texto. De esta manera se tiene un archivo flexible que puede ser ampliado conforme se vaya complicando la aplicación, añadiéndole distintas propiedades a cada uno de los elementos del diagrama.

4.2.1 Archivo de entrada

Primero se va a explicar la estructura general del archivo; ya que el código empleado para su lectura, la función LecturaEntrada(), está altamente condicionado por dicha estructura. El archivo se puede dividir en 4 bloques principales; un bloque de introducción, en el que se encuentran los datos fundamentales del diagrama, seguido de tres bloques dedicados a cada uno de los elementos del diagrama, las máquinas (quirófanos) los operarios (cirujanos) Y los trabajos (pacientes).

Todos estos bloques, salvo la introducción, están precedidos por una palabra clave, Maquina, Operario y Trabajo, a partir de la cual se indica que se va comenzar con la lectura de propiedades de dichos elementos del diagrama. Dentro de cada uno de estos bloques existen palabras que identifican cada una de sus propiedades como puedan ser el nombre o la descripción. Todas estas palabras clave son *'case sensitive'* por lo que se ha de tener cuidado a la hora de escribirlas ya que si estas no coinciden con el patrón producirá un fallo en la lectura. Además, cada uno de los bloques no puede incluir ninguna línea sin texto entre el fin de la lectura de una propiedad y el comienzo de otra, ya que este es el indicador de que se abandona un bloque para entrar en otro.

El bloque de los operarios es opcional por lo que en caso de que esta información no sea relevante se podría omitir. El orden de estos bloques es crucial para la lectura del archivo y ha de ser el siguiente:

1. Introducción
2. Máquinas
3. Operarios
4. Trabajos

Por último comentar que los separadores empleados para dividir las cadenas de números o de texto son el espacio, el cambio de línea y el ampersand (&), pero estos pueden ser modificados en función de los medios que se tengan para la producción del fichero de entrada.

Estructura interna de los bloques

Una vez aclarados estos términos se puede comenzar a realizar un análisis exhaustivo del fichero, comenzando por el bloque de introducción. Este está conformado por dos líneas, en la primera de ellas se introduce la fecha a partir de la cual se va a realizar la programación introduciendo el día, mes y año separados por espacios. En la segunda de ellas se introducen datos relevantes

para la programación, estos son el número total de trabajos, número de máquinas y días totales de la programación (contando el origen de programación dentro de esta cuenta), introducidos como enteros separados por espacios.

Con estos datos ya se puede comenzar con la lectura del resto de bloques, por lo que se pueden dejar una o varias líneas en blanco antes de introducir la palabra clave Maquina, seguida de tantas filas como máquinas haya. En cada una de las filas se encontraran los tiempos de proceso de cada trabajo en cada máquina ordenados por columnas. En caso de que las máquinas sean idénticas dichos tiempos deberán de ser iguales. Después hay que determinar los días que la máquina está operativa, por lo que se ha de introducir el horario de cada máquina para todos los días de la programación. Si estos horarios no fueran introducidos, las máquinas considerarán paradas (quirófanos cerrados) todos los días. Para ello se debe escribir la palabra clave Horario seguida de tantas filas como máquinas haya. En cada una de estas filas se escribirán los horarios de cada día, comenzando por el origen, separadas entre ellas por el ampersand (&). Estos horarios se escribirán como una cadena de números decimales separados por espacios, estructura ya explicada en la sección dedicada a la clase máquina.

Es muy importante que para los decimales se emplee una coma y que el número de horas de apertura y cierre sean iguales. Si se deja un espacio vacío entre dos ampersand (& &) el quirófano se considerará cerrado ese día. A continuación irían las distintas propiedades, pero la escritura de las mismas se explicará más adelante.

Tras dejar una línea vacía comenzaría el bloque de operarios, en caso de que se quisiera incluir, si se omite no se deberá escribir nada. Para incorporar este bloque se debe introducir la palabra clave operario y en la siguiente línea el número total de operarios. Con esto sería suficiente y se podría continuar con las propiedades.

Por último se realiza la lectura de los trabajos, para ello se escribe la palabra clave Trabajo seguida de la palabra clave RutaFabricacion. A continuación se escriben tantas líneas como trabajos haya y en cada línea la ruta de fabricación tal y como se ha explicado en el apartado clase Trabajo.

Después se debe escribir la palabra FechaFin o FechaInicio, según se quieran introducir las fechas en las cuales finalizan o comienzan a procesarse los trabajos en cada una de las máquinas. Posteriormente se introducen tantas filas como trabajos haya y en cada columna el número entero de minutos desde la fecha origen de programación (considerando las 00:00 de ese día) que marcan la fecha de inicio o fin de proceso de ese trabajo en cada máquina, separadas por espacios. En caso de que ese trabajo no se asigne se escribirá un número negativo, normalmente -1.

Finalmente, en caso de que haya optado por incluir los operarios asignados a cada trabajo se debe de incluir la palabra clave OperarioAsignado seguida de la palabra clave ID, la cual indica que se van a emplear los identificadores de los operarios para asignarlos. Se deben escribir tantas filas como trabajos haya y en cada una de ellas los identificadores de los operarios asignados al trabajo en cuestión separados por espacios. Si un trabajo no tuviese operario asignado se deberá dejar la línea que le corresponda en blanco.

Propiedades

Una vez explicado el grueso del archivo se va a continuar con la forma en la que se ha de introducir cada propiedad y los elementos a los cuales va asignado. Todas estas propiedades se comienzan a leer cuando se escribe su palabra clave y tienen que encontrarse dentro del bloque en el cual van a

ser aplicadas, sin dejar una línea vacía entre el fin de escritura de una propiedad y la palabra clave de la siguiente.

- **Nombre**

  Propiedad aplicable a trabajo, operarios y máquinas puede ser una máquina un trabajo o un operario. Para su lectura se escriben en la línea siguiente a la palabra clave los nombres en cuestión de cada elemento separados por ampersand (&).

- **Descripción**

  Propiedad aplicable a trabajo, operarios y máquinas puede ser una máquina un trabajo o un operario.. Tras la palabra clave se introducen tantas líneas como elementos haya y en cada fila se escribe la descripción.

- **Rendimiento**

  Esta propiedad únicamente aplicable a máquinas y se introduce como valores decimales, empleando la coma como el separador de decimales, divididos por espacios, en la línea que sigue a la palabra clave. El primer valor corresponde a la primera máquina, el segundo a la segunda y así sucesivamente. En caso de no introducirse el valor predeterminado es 1 por lo que en caso de máquinas idénticas todas tendrán los mismos tiempos de proceso.

Estas propiedades son todas opcionales y se pueden incorporar más a gusto del consumidor. Algunas posibilidades son el DNI o el número de la seguridad social.

4.2.2 Ejemplos de fichero de entrada

En esta sección se adjuntan dos ficheros de entrada junto a sus diagramas producidos. Se ha intentado que cada fichero sea lo más breve posible, pero que incluya todos los elementos característicos del fichero de entrada. La única diferencia entre ambos ficheros es que el primero incluye operarios y es de máquinas paralelas mientas que el segundo no incluye operarios y es un taller de flujo.

Ejemplo máquinas paralelas

```
7 9 2020
7 3 3

Maquina
214 220 133 66 126 37 118
214 220 133 66 126 37 118
214 220 133 66 126 37 118
Horario
7 14 17 20 & &7 14 17 20
7,5 14,5 & &7,5 14,5
7 15 & &7 15
Nombre
Quirofano 0&Quirofano 1&Quirofano 2
Descripcion
Descripcion Quirofano 0
Descripcion Quirofano 1
Descripcion Quirofano 2
Rendimiento
1 1 1

Operario
```

```
4
Nombre
Cirujano 0&Cirujano 1&Cirujano 2&Cirujano 3
Descripcion
Descripción Cirujano 0
Descripción Cirujano 1
Descripción Cirujano 2
Descripción Cirujano 3

Trabajo
RutaFabricacion
-1
-1
-1
-1
-1
-1
-1
FechaInicio
-1 -1 420
-1 -1 3300
3300 -1 -1
420 -1 -1
-1 453 -1
-1 599 -1
486 -1 -1
Nombre
Trabajo 0&Trabajo 1&Trabajo 2&Trabajo 3&Trabajo 4&Trabajo 5&Trabajo 6
Descripcion
Descripcion Trabajo 0
Descripcion Trabajo 1
Descripcion Trabajo 2
Descripcion Trabajo 3
Descripcion Trabajo 4
Descripcion Trabajo 5
Descripcion Trabajo 6
OperarioAsignado
ID
0 3
2 3
2
1 2
0

0 2
```

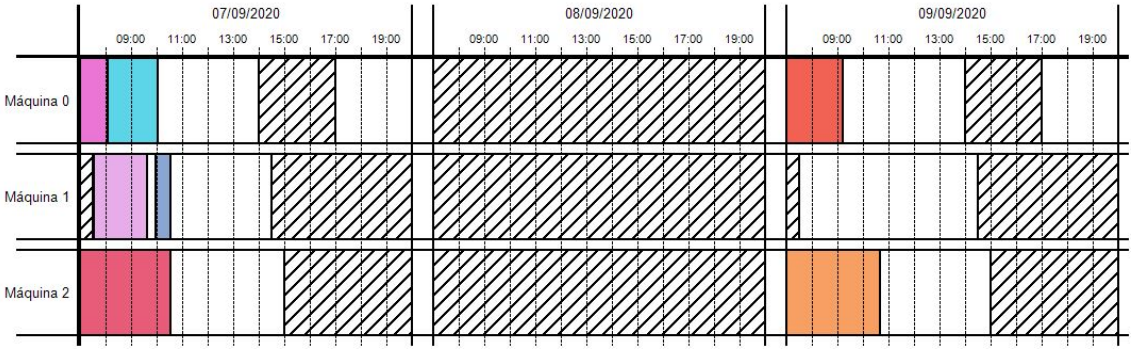

Figura 4.3 Diagrama de Gantt representando un modelo de máquinas paralelas, producido por el fichero de ejemplo correspondiente, *elaboración propia*.

Ejemplo taller de flujo

```
4 9 2020
7 4 1

Maquina
42 7 13 10 39 25 49
23 17 35 28 13 19 45
9 44 42 21 40 18 36
54 25 8 15 30 40 51
Horario
7 15
7 15
7 15
7 15
Nombre
Maquina 0&Maquina 1&Maquina 2&Maquina 3
Descripcion
Descripcion Maquina 0
Descripcion Maquina 1
Descripcion Maquina 2
Descripcion Maquina 3
Rendimiento
1 1 1 1

Trabajo
RutaFabricacion
0 1 2 3
0 1 2 3
0 1 2 3
0 1 2 3
0 1 2 3
0 1 2 3
0 1 2 3
FechaInicio
420 462 485 494
462 485 502 548
```

```
469 502 546 588
482 537 588 609
492 565 609 649
531 578 649 679
556 605 667 719
Nombre
Trabajo 0&Trabajo 1&Trabajo 2&Trabajo 3&Trabajo 4&Trabajo 5&Trabajo 6
Descripcion
Descripcion Trabajo 0
Descripcion Trabajo 1
Descripcion Trabajo 2
Descripcion Trabajo 3
Descripcion Trabajo 4
Descripcion Trabajo 5
Descripcion Trabajo 6
```

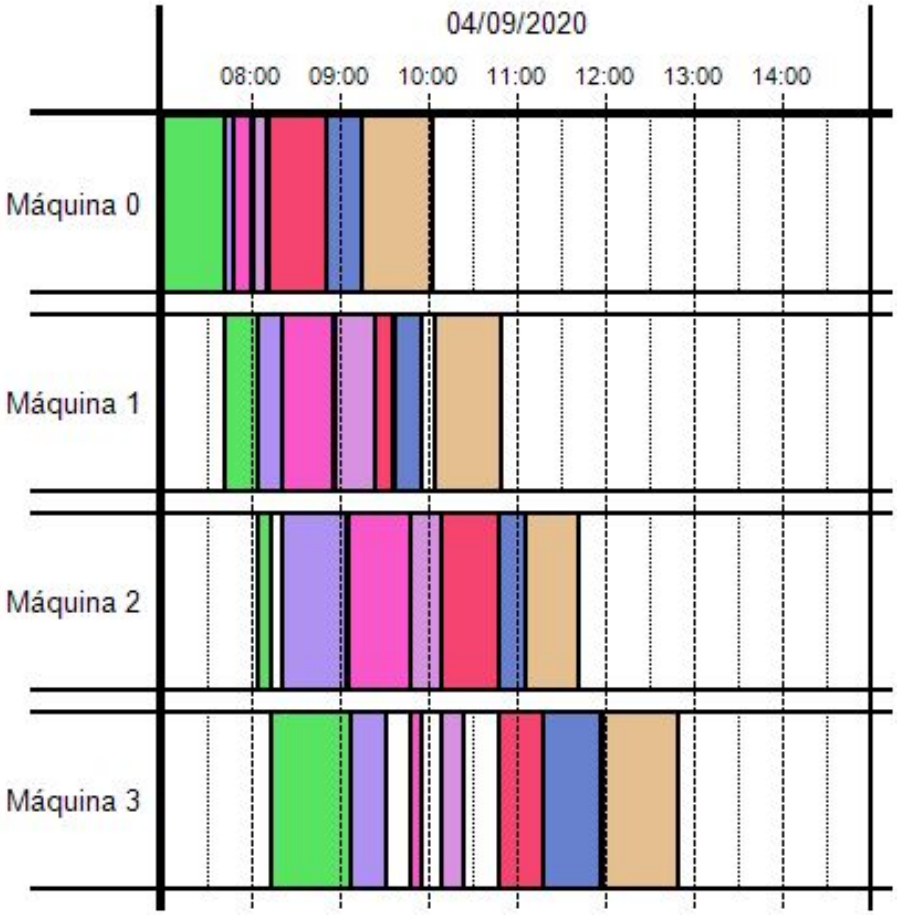

Figura 4.4 Diagrama de Gantt representando un taller de flujo, producido por el fichero de ejemplo correspondiente, *elaboración propia*.

4.2.3 Metodología de lectura y escritura

En este apartado se van a dar unas nociones básicas del funcionamiento de los métodos que permiten la lectura y la escritura del archivo. Ambos necesitan como argumento la ruta que llega hasta el

archivo que debe ser leído o guardado, la cual se obtiene de los cuadros de diálogo.

Con esto introducido se crea un StreamReader o StreamWriter para modificar el archivo. En el caso de la lectura del archivo primero se leen las líneas de los datos generales y después se van leyendo las clases. Más tarde se leen las características generales de cada clase y después una palabra clave seguida de tantas líneas como necesite la propiedad. En caso de que cuando se lea una línea en la que debe de estar una palabra clave no haya salta un mensaje de error con la línea en cuestión.

La escritura es notablemente más sencilla ya que consiste en reproducir el fichero de entrada con los datos que se tienen. Cabe destacar que el método de de escritura incorpora todas las propiedades en el archivo de salida, independientemente de si se han introducido en la entrada o no, devolviendo cadenas de texto vacías en las propiedades de las que no se introdujo valor.

4.3 Controles de la aplicación

Para poder representar información en cualquier aplicación informática desarrollada en Windows Forms es necesaria la implementación de controles de usuario, especialmente si se pretende permitir interacción con el mismo. De este modo los datos almacenados, que han sido introducidos empleando el código interno analizado con anterioridad, pueden presentarse de una manera ordenada, concisa y clara. Desafortunadamente, los controles propuestos en la herramienta Windows Forms son simples, por lo que no basta con emplearlos para que cubran las necesidades de funcionamiento de la aplicación. Por este motivo se han tenido que desarrollar diversos controles complementarios, los cuales tienen una base común llamada ControlGantt, derivada de la clase base Control proporcionada por Windows Forms.

El principal objetivo de la clase es facilitar la sincronización de los controles que hereden de la misma, permitiendo actualizar sus gráficos simultáneamente sin necesidad de emplear código externo a los mismos. La clase ControlGantt dota a todos sus controles heredados de una lista interna llamada controlesEnlazados que agrupa el conjunto de controles con los cuales están enlazados. Un ControlGantt que tiene otros en su lista controlesEnlazados dispone de varios métodos que le permiten actualizar sus gráficos de la manera que se desee. Estos métodos son idénticos a los tradicionales, los cuales se analizarán con detalle junto a la clase DiagramaGantt, pero se les añade la palabra Linked a continuación, uno ejemplo de ellos es el método RefreshLinked().

Para enlazar y desenlazar los controles de la clase ControlGantt se emplean dos métodos estáticos de la misma, UnLink(*var args*) y link(*var args*). Ambos toman como argumentos un array de ControlGantt, el primero desenlaza cada control del array de los controles en los que esté enlazado y el segundo enlaza todos los controles introducidos entre si, llamando a Unlink(*var args*) para cada control antes de realizar los enlaces.

La figura 4.5 representa el diagrama de clases derivadas de ControlGantt, las cuales conforman los controles empleados para la elaboración de la aplicación. Las principales son ListBoxGantt, la LabelGantt y DiagramaGantt, las cuales representan la clase genérica de la lista de elementos de planificación, la clase genérica de paneles de información y el diagrama de Gantt, respectivamente. Un punto positivo de esta implementación es que, en caso de querer elaborar una aplicación similar, se pueden reciclar todos los controles sin la necesidad de implementarlos todos en la nueva herramienta. Por ejemplo, se podría crear una aplicación únicamente con el diagrama y el panel de información o solamente la lista de trabajos y el diagrama.

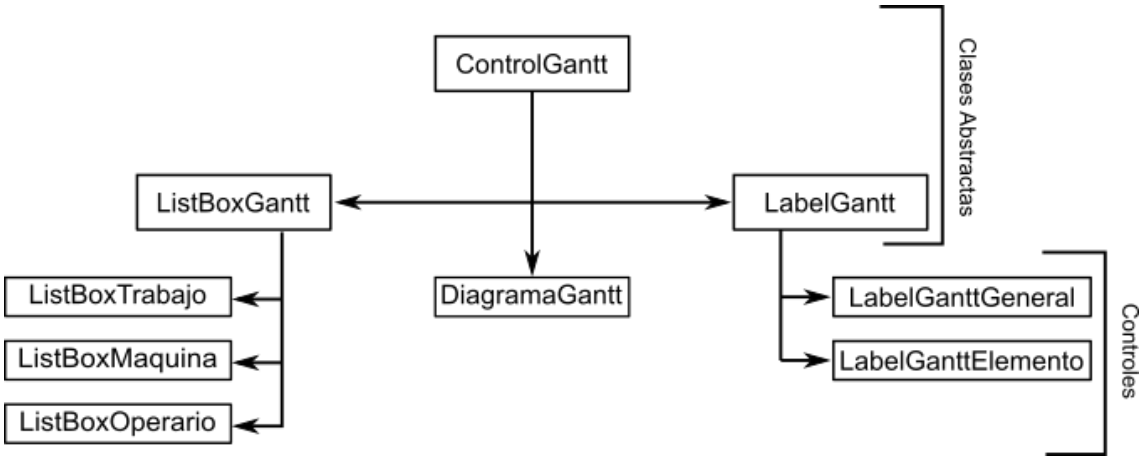

Figura 4.5 Diagrama de clases que derivan de ControlGantt, *elaboración propia*.

4.3.1 Clase DiagramaGantt

La representación del diagrama es la funcionalidad principal de la aplicación, por este motivo es en la clase DiagramaGantt donde se ha invertido una mayor cantidad de horas de trabajo. El objetivo de esta clase es representar un diagrama de Gantt interactivo, salvaguardando posibles problemas de *lag* a la hora de refrescar la pantalla. Existen varias maneras elaborar un diagrama de Gantt con estas características, cada una con sus ventajas e inconvenientes.

Una de ellas consiste en representar los ejes del diagrama de Gantt como un dibujo en un PictureBox, mientras que para los trabajos representados en su interior se emplean paneles creados dinámicamente. Esto facilita la implementación de la mecánica de selección por clic o el *drag and drop*. El principal problema surge al elevar el número de trabajos, ya que esto provoca que exista una cantidad excesiva de paneles activos al mismo tiempo, por lo que el *lag* del programa se vuelve limitante. Por este motivo es recomendable que la representación del diagrama no sea continua y se deba fragmentar por periodos temporales, como puedan ser semanas.

Aunque esta solución ha sido probada con cierto grado de éxito en una aplicación del departamento [Dios et al., 2015], se ha buscado explorar nuevas soluciones que permitan la elaboración de un diagrama continuo sin que se penalice el rendimiento de la herramienta. La principal propuesta es emplear un diagrama dibujado en un PictureBox, que incluya tanto a los ejes del diagrama como la representación de los trabajos, necesitando así un único control. De esta forma se reduce el número total de controles de un valor indefinido a un valor fijo, aunque el tamaño total del diagrama seguirá afectando al rendimiento, ya que el tiempo necesario para representarlo es proporcional a sus dimensiones.

El principal inconveniente de esta propuesta es que implementar la mecánica de clic es ligeramente más complicada que en otras soluciones, ya que no se puede basar en el evento clic de cada Panel sino en el del control principal. No obstante las ventajas que se obtienen compensan estos inconvenientes, ya que se puede realizar *scroll* empleando la mecánica *click and drag* y además se gana en rendimiento.

Implementación del *scroll* y representación del diagrama

A la hora de realizar un diagrama continuo la primera dificultad que surge es la de representar los ejes el diagrama en todo momento mientras el usuario se desplaza por el mismo. Se puede optar por

no incorporar esta característica pero, si el diagrama es excesivamente grande, es probable que los ejes de temporales o de máquinas se escondan dificultando la interpretación del mismo. La solución a este problema no es trivial pero se puede llegar a obtener un resultado de una calidad aceptable dividiendo el diagrama principal en 4 paneles independientes como se indica en la figura 4.6.

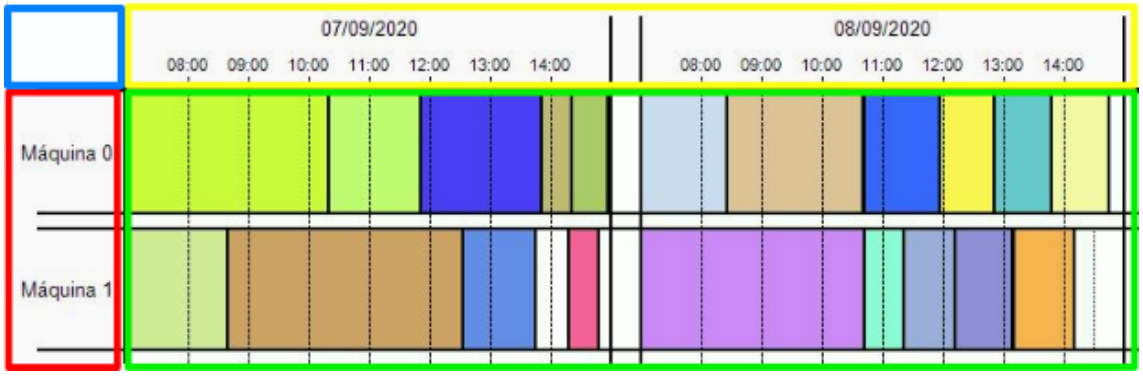

Figura 4.6 División del control DiagramaGantt en los paneles independientes que lo conforman, *elaboración propia* .

Cada uno de estos paneles representará uno de los elementos básicos del diagrama, siendo estos los ejes (rojo y amarillo), la esquina superior (azul), la cual es empleada principalmente para ocultar los paneles al hacer *scroll*, y el diagrama en sí mismo (verde). Todos ellos tienen en su interior incorporados una PictureBox donde se representará el fragmento del diagrama correspondiente, el cual le dará las dimensiones al control. El panel de la esquina y el del diagrama tienen el tamaño justo para aparecer dentro del panel principal, pero los ejes tienen las dimensiones de sus respectivos PictureBox. Esto último ha de ser así para poder acoplar el movimiento de los ejes al AutoScroll del panel del diagrama principal, el cual estará activo y se reajustará su tamaño mínimo para acomodar la PictureBox del diagrama.

Para conseguir que los gráficos del diagrama se mantengan sincronizados se han programado dos métodos, AjustarPosicionesScroll() y AjustarGeometriaPaneles(). El primero sincroniza la posición de los ejes con el diagrama y el segundo reajusta el tamaño de los paneles en caso de que se modifique el tamaño del diagrama, además de llamar a AjustarPosicionesScroll().

Con esta característica pulida e implementada ya se puede proceder a dibujar el diagrama, ya que si el *scroll* no fuese posible no tendría sentido proseguir con el desarrollo del diagrama empleando esta solución. La metodología de representación de los gráficos será analizada al final de esta sección debido a que no se ha empleado un método convencional, como pueda ser dibujar los elementos en el evento Paint, con el fin de mejorar el rendimiento. Al tratarse de un control se ha buscado parametrizar todas las propiedades del diagrama con el fin de facilitar modificaciones del mismo, ya sea en tiempo de ejecución o a sus valores predeterminados. Estas propiedades se pueden ver en la figura 4.7 en la que aparecen los componentes básicos sin incluir ni trabajos ni horarios. A modo de leyenda de la figura 4.7 se va a incluir a continuación un listado con dichas propiedades:

- **A** AltoMaquina
- **B** SeparacionMaquina
- **C** AnchoHora
- **D** SeparacionHora

- **E** AltoEjeDias
- **F** ExcesoSuperiorLineasVerticales
- **G** AnchoEjeMaquinas
- **H** ExcesoIzquierdoLineasHorizontales
- **I** ExcesoInfriorLineasVerticales
- **J** ExcesoDerechoLineasHorizontales
- **K** ExcesoLíneasSecundarias

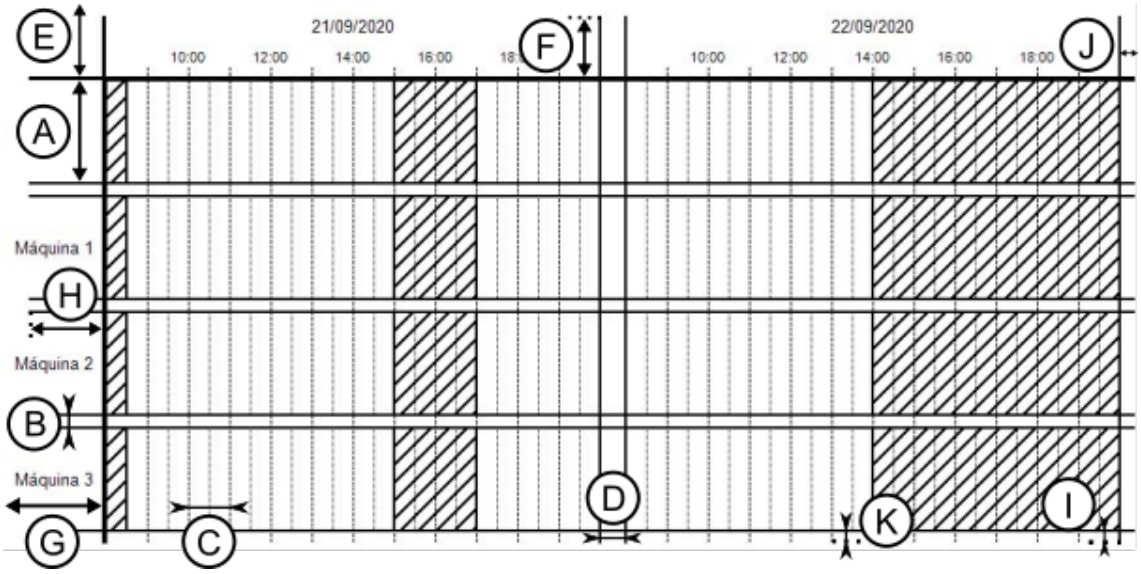

Figura 4.7 Esquema de las diferentes propiedades que modifican el aspecto físico del diagrama de Gantt representado, *elaboración propia*.

De todas estas propiedades las más relevantes son el alto de la máquina y el ancho de la hora, ya que definen en gran medida las dimensiones del diagrama. No se ha seleccionado el ancho del día como parámetro principal debido a que este depende del número de horas por día que se hayan definido, por lo que puede ocurrir que este tamaño sea demasiado pequeño como para representar correctamente el diagrama. A la propiedad AnchoHora solamente se le permite tomar valores que sean múltiplos de 4, esto se debe a que es el valor necesario para que la representación de las líneas secundarias y terciarias sea fiel al número de píxeles existentes. De esta manera no se obtienen fracciones de píxeles al dividir las horas en medias horas o cuartos de hora.

Todas las propiedades que comienzan por exceso indican la longitud de las líneas que sobresalen por cada uno de los extremos del diagrama. En el caso de las líneas secundarias esto sucede tanto por encima como por debajo del diagrama.

Existen una serie de propiedades que también forman parte del diagrama y podrían ser alteradas en tiempo de ejecución en función del diagrama representado si fuera necesario, como el número de días de la programación, la fecha de origen o la hora más temprana y más tardía representadas, pero que son obtenidas directamente de la clases principales en las que están almacenadas. También se puede modificar la fuente y el tamaño de los textos que aparecen en el diagrama.

Dibujar el esqueleto del diagrama una vez definidas las propiedades es trivial, aún así se estima oportuno comentar los métodos DibujaLineasParalelas(*var args*), DibujaSerieStrings(*var args*). El

primero, como su nombre indica, se emplea para dibujar líneas paralelas a lo largo de un eje con un ángulo determinado. El segundo calcula una serie de rectángulos en base aun eje y en el interior de cada uno representa un *string* introducido.

Para dibujar las operaciones de cada trabajo se ha de trabajar únicamente con el panel dedicado a esa función, por lo que la tarea se facilita. El principal problema es ubicar cada operación en un rectángulo dentro del PictureBox del diagrama, esta tarea se consigue empleando un método llamado GetRectanguloDiagrama(*var args*), el cual toma como argumentos un tiempo de proceso, una máquina y una fecha de inicio.

Con estos datos el método devuelve las coordenadas y el tamaño del rectángulo que representaría a la operación introducida en el interior del diagrama. De esta forma se puede representar cada trabajo en el diagrama, ya que bastaría con dibujar ese rectángulo, pero también se puede detectar si la posición del ratón está contenida dentro de dicho rectángulo, lo cual será de utilidad para implementar los eventos clic.

Por último, representar los horarios de las máquinas no es excesivamente complicado, pero se ha de tener claro la manera en la que se organizan los horarios. Se ha programado un método llamado DibujaHorario(*var args*) que dibuja el horario de un día concreto dadas las aperturas, los cierres, el día con respecto al origen y la máquina.

Interacción del usuario

Una vez representado el diagrama el siguiente objetivo ha de ser el de permitir que el propio usuario pueda interactuar con él. Esta clase permite tres formas de interacción que son el de *clic and drag* , la del clic y la de *drag and drop*. La primera habilita el desplazamiento por el diagrama sin necesidad de usar las barras laterales, la segunda es necesaria para seleccionar los trabajos o maquinas en el diagrama y la tercera permite la modificación de la planificación.

Aunque no es estrictamente necesaria la implementación de un *click and drag*, esta característica es muy agradecida, ya que es preferible desplazarse por una imagen o diagrama empleando el ratón en lugar de las barras laterales. Además, una vez la mecánica del *scroll* está conseguida, elaborar un código que permita realizar un *click and drag* es una tarea bastante sencilla. Basta con registrar la posición del ratón cuando se pulsa el botón izquierdo y, si este se desplaza manteniendo el botón presionado, modificar los valores de las ScrollBars del AutoScroll una cantidad igual a la diferencia de posiciones del cursor entre el inicio y el fin. Este proceso se deberá repetir siempre que el botón se mantenga presionado.

Para la implementación del evento clic es necesario emplear el método GetRectanguloDiagrama(*var args*) justo cuando se dispara el evento. Se realiza un bucle de dicho método para todos los trabajos existentes en ese momento y, en el caso de que uno de esos rectángulos contenga la posición del cursor, se cambia el valor de su propiedad seleccionado y se redibuja el diagrama. Empleando la misma técnica se puede identificar si el trabajo que contiene al cursor está seleccionado, por lo que, si en vez de hacer clic se mueve el ratón sobre el trabajo seleccionado, se dispararía un evento de DragEnter en lugar de iniciarse un *click and drag*.

Una vez disparado el evento DragEnter se elimina la operación seleccionada del trabajo y se vuelve a pintar el diagrama. En este momento un panel con el nombre de panelTrabajo es mostrado y desplazado a la posición del cursor, con el tamaño y color de la operación eliminada. Además, se restringe el desplazamiento del ratón al cuadro del diagrama si se mantiene el botón presionado.

Esto permite implementar una mecánica de *scroll* que se produce cuando el cursor roza los bordes del rectángulo del diagrama, que desplaza el dibujo a una velocidad preestablecida en el sentido de la ubicación del cursor. Para conseguirlo emplea el método ScrollDiagrama() durante el evento DragOver el cual se dispara constantemente mientras se desplaza el cursor sobre el diagrama. Durante este evento se realizan también los cálculos para representar el rectángulo de la sombra que permite guiar al usuario, además de llamarse al método de actualizar los gráficos del diagrama para actualizar la posición de dicho rectángulo. Se entrará más en detalle en la gestión de los gráficos dentro del apartado de optimización de los gráficos.

Para calcular la operación que representa el rectángulo sombra se emplean varios métodos que relacionan su posición con la fecha de inicio y con una máquina (obteniendo el tiempo de proceso de la máquina seleccionado). En caso de que no se represente el rectángulo, la operación no se calcularía, esto puede darse si el trabajo se encuentra sobre otro ya planificado o fuera del diagrama. Se ha implementado una ayuda que, cuando la posición del rectángulo sombra no es válida, realiza un barrido de posiciones en ambos sentidos con intervalos de un minuto y vuelve a calcular si es posible representar el rectángulo. De esta forma se facilita al usuario colocar dos trabajos consecutivos, es decir que la fecha de inicio de uno es igual a la de fin del otro o viceversa. Durante este evento también se representan las fechas de inicio y de fin de la operación en la esquina superior izquierda del programa, para facilitar un ajuste preciso de la programación. Por último, en cuanto se levanta el ratón, se dispara el evento DragDrop y, en caso de que el rectángulo sombra esté pintado, añade una operación al trabajo seleccionado con la fecha de inicio y esa fecha de finalización.

Para el caso del evento clic en el eje de las máquinas el principio es idéntico pero empleando una función diferente para identificar la máquina seleccionada. Esta función emplea únicamente la coordenada Y del cursor en lugar de ambas ya que es la única coordenada que define a las máquinas, dentro del panel dedicado a su eje.

Implementación del zoom

Se ha decidido separar el zoom del resto de interacciones ya que para implementarlo se ha empleado un control externo. La barra de Zoom en la esquina inferior derecha de la aplicación toma un valor que, multiplicado por 8, es el introducido como AnchoHora en el diagrama.

Al modificar esta propiedad en tiempo de ejecución el diagrama completo se vuelve a dibujar, permitiendo que el zoom se implemente, para lo cual tambiénse debe de llamar al método AjustarGeometríaPaneles(). Esta funcionalidad incrementa la precisión con la cual se pueden modificar las fechas de las operaciones durante el *drag and drop*, por lo que se ha considerado muy positivo implementarla.

Si se prueba a ampliar o disminuir el zoom se puede apreciar que las líneas discontinuas del interior del diagrama o las de los ejes aumentan o disminuyen en número. Esto se ha conseguido empleando un método que determina las posiciones de cada tipo de línea en función del valor de AnchoHora.

Optimización de los gráficos

En este apartado se va a comentar el proceso seguido para minimizar el retraso de refresco en los gráficos, especialmente en el diagrama. Es de vital importancia reducir el tiempo de ejecución dedicado a calcular los gráficos para que la aplicación se sienta fluida cuando se esté interactuando con ella. Esto es crítico durante el evento DragOver, debido a que este evento se dispara continuamente en la interacción *drag and drop*, por lo que cada vez que se desplaza el ratón se actualizan los gráficos y, si el retraso entre dos disparos de este método es demasiado grande, la imagen se ve

desactualizada y distorsionada.

Para obtener una estimación del tiempo que transcurre entre dos actualizaciones del diagrama se va a ejecutar el código del evento DragOver en modo depuración, de esta forma se puede medir su tiempo de ejecución. Si empleamos la metodología más tradicional, incorporando los gráficos al evento Paint de las PictureBox y refrescando toda la pantalla, se obtiene un tiempo aproximadamente de 100ms. Este resultado es nefasto ya que implica unas 10 actualizaciones por segundo, un número lo suficientemente bajo como para que el ojo humano lo perciba.

La fluidez de los gráficos se obtiene de la tasa de fotogramas por segundo (fps), por lo cual sería ideal alcanzar una tasa similar a la empleada en el cine (24 fps) o en videojuegos poco exigentes gráficamente (30 fps), ya que han demostrado ser suficientes para que una persona no aprecie saltos en la imagen. Esto implicaría que el tiempo de ejecución del código en el evento DragOver ha de ser menor a 42ms, un número bastante alejado del primer dato obtenido. Además, la cifra obtenida en depuración es algo optimista, ya que no incluye el tiempo transcurrido entre dos eventos DragOver, ni el tiempo máximo empleado para calcular el rectángulo sombra, el cual se produce cuando hay muchos trabajos y no se encuentra una posición en el que representarlo. También hay que tener en cuenta que estos tiempos dependen del tamaño del diagrama por lo que cuanto mayor sea más lenta va a ser la ejecución del código. Aún así, a partir de este momento se va a realizar la hipótesis de que el tiempo de proceso es proporcional al tamaño del diagrama, por lo que los valores obtenidos serán analizados cualitativamente comparándolos unos con otros y no cuantitativamente.

Antes de comenzar a optimizar el rendimiento de los gráficos, es fundamental comprender cómo se generan y qué es lo que influye en su tiempo de proceso. La actualización de los gráficos se produce de dos formas diferentes, en función de los métodos empleados. Cada una de ellas tiene sus ventajas e inconvenientes los cuales serán explicados a continuación:

- **Invalidate()** y **Update()**

  Estos dos métodos permiten gestionar a la perfección la actualización de los gráficos de cualquier control. Invalidate() indica que los gráficos del control han de ser actualizados y deja que el sistema decida cuando refrescarlos. Este método es ideal si se realizan varias llamadas consecutivas a él ya que no se ha de volver a pintar el control entero cada vez, sino que las tres actualizaciones entrarán en efecto simultáneamente cuando el sistema lo estime oportuno. Además Invalidate() permite tomar como entrada una región del control para indicar que se debe repintar únicamente esa sección. El método Update() se emplea para indicar al sistema que se han de actualizar las zonas del control que han sido invalidadas.

- **Refresh()**

  Este método produce un efecto similar a llamar a Invalidate() y Update() en rápida sucesión. No es recomendable su uso en eventos que son llamados periódicamente, ya que puede provocar que se refresque la imagen con demasiada frecuencia, ralentizando el resto de funcionalidades del código. Aún así se puede emplear para controles de reducido tamaño y poco contenido gráfico.

Como es lógico se ha optado por emplear Invalidate() sin Update() siempre y cuando sea posible, ya que evita malgastar tiempo actualizando innecesariamente los gráficos, sin embargo, en el evento DragOver es obligatorio usar Update(), ya que el sistema solamente refresca los gráficos cuando el ratón está parado, esto es debido a que cuando eel rtón está en movimiento se dispara los eventos DragOver constantemente. A pesar de esto, emplear Invalidate() ofrece mejores resultados que Refresh(), porque permite actualizar únicamente la zona del rectángulo sombra en lugar de todo el diagrama. Aun así, independientemente de la metodología empleada, siempre se acaba disparando

el evento Paint por lo que reducir su tiempo de proceso es también de vital importancia.

Aislando el evento Paint del diagrama del resto de elementos, se obtiene que son necesarios unos 40ms para su ejecución, este tiempo es excesivo ya que ejecutar este evento conforma únicamente una de las partes de actualizar los gráficos. La mejor forma de acelerar la ejecución consiste en eliminar todos los cálculos innecesarios, para que lo único que se realice en el evento sea dibujar el control. Una forma de lograrlo consiste en almacenar los gráficos en una imagen o mapa de bits (*bitmap* en inglés) y representarlo en lugar de los gráficos.

El primer paso para lograr esta implementación fue medir qué parte del evento Paint es más lenta, resultando ser los códigos que dibujaban los horarios de las máquinas (10ms) y las líneas terciarias (15ms). Si la representación de estos elementos se transcribe a su propio mapa de bits se obtienen unos tiempos de dibujado de 5ms en ambos casos. Esto significa una mejora considerable frente a los resultados anteriores con una reducción del tiempo de ejecución orden del 50 % por lo que todo apunta a que este es el camino correcto. Con el fin de ver si se producía alguna mejora, se decidió mezclar ambos bitmaps en uno solo, el cual también tardaba en representarse 5ms, lo cual sugiere que el tiempo de dibujado de una bitmap de estas características se puede suponer proporcional únicamente a su tamaño. En base a esta hipótesis lo ideal sería agrupar todos los gráficos en un mapa de bits y representarlo, en lugar de pintar cada elemento uno a uno en el evento Paint. Entonces basta con actualizar el mapa de bits tras eliminar la operación del trabajo en el *drag and drop*, y tras cada actualización de los datos que este representa.

El proceso de gestión de gráficos para la modificación de la programación se puede ver detallado en la figura 4.8. Se ha escogido representar este proceso ya que es el más complejo de elaborar. Antes de iniciar el evento DragEnter el bitmap se encuentra actualizado representando el diagrama completo. Una vez se inicia el *drag and drop* el trabajo se elimina y se vuelve a actualizar el bitmap, de esta manera se tiene por un lado el panel que se está desplazando mediante el *drag and drop* y por otro el bitmap con el resto de gráficos del diagrama. De esta forma el bitmap que se representa es siempre el mismo y solamente hay que preocuparse de actualizar la nueva posición del panel. Por este mismo motivo, el rectángulo que se emplea como sombra del diagrama tampoco está incluido en el bitmap y se pinta debajo del mismo. Finalmente, tan prontocomo el *drag and drop* finaliza, se vuelve a actualizar el bitmap con el trabajo en su posición definitiva.

Como era de esperar, tras aplicar estos cambios el rendimiento aumentó considerablemente, consiguiendo obtener un tiempo de ejecución del evento DragOver por debajo de los 40ms. De esta manera se logra que el *lag* resultante de desplazar el trabajo por la pantalla sea apenas apreciable. Sin embargo, esta solución provocó una serie de inconvenientes, ya que el tiempo de creación del bitmap (100ms) es superior al de realizar el evento Paint (40ms), por lo que para cambios continuos los resultados son notablemente peores. Es por este motivo que el dibujado del rectángulo sombra se ha extraído del bitmap, ya que varía continuamente y tarda menos de 1ms en ejecutarse. En la siguiente lista se encuentran los momentos en los empeoró considerablemente el rendimiento:

- Al hacer clic en el diagrama se incrementa el tiempo hasta que el trabajo aparece como seleccionado.
- El zoom tarda demasiado en actualizarse cuando se cambia el valor del ancho de las horas.

Para solucionar este problema se ha incluido una variable booleana que indica si el bitmap está actualizado o no, en caso de estarlo el evento Paint lo emplea para acelerarlo y en caso contrario sigue empleando la metodología más lenta, pero mejor que crear un bitmap nuevo para cambios excesivamente frecuentes. De esta forma cada vez que se cambie el ancho de las horas empleando

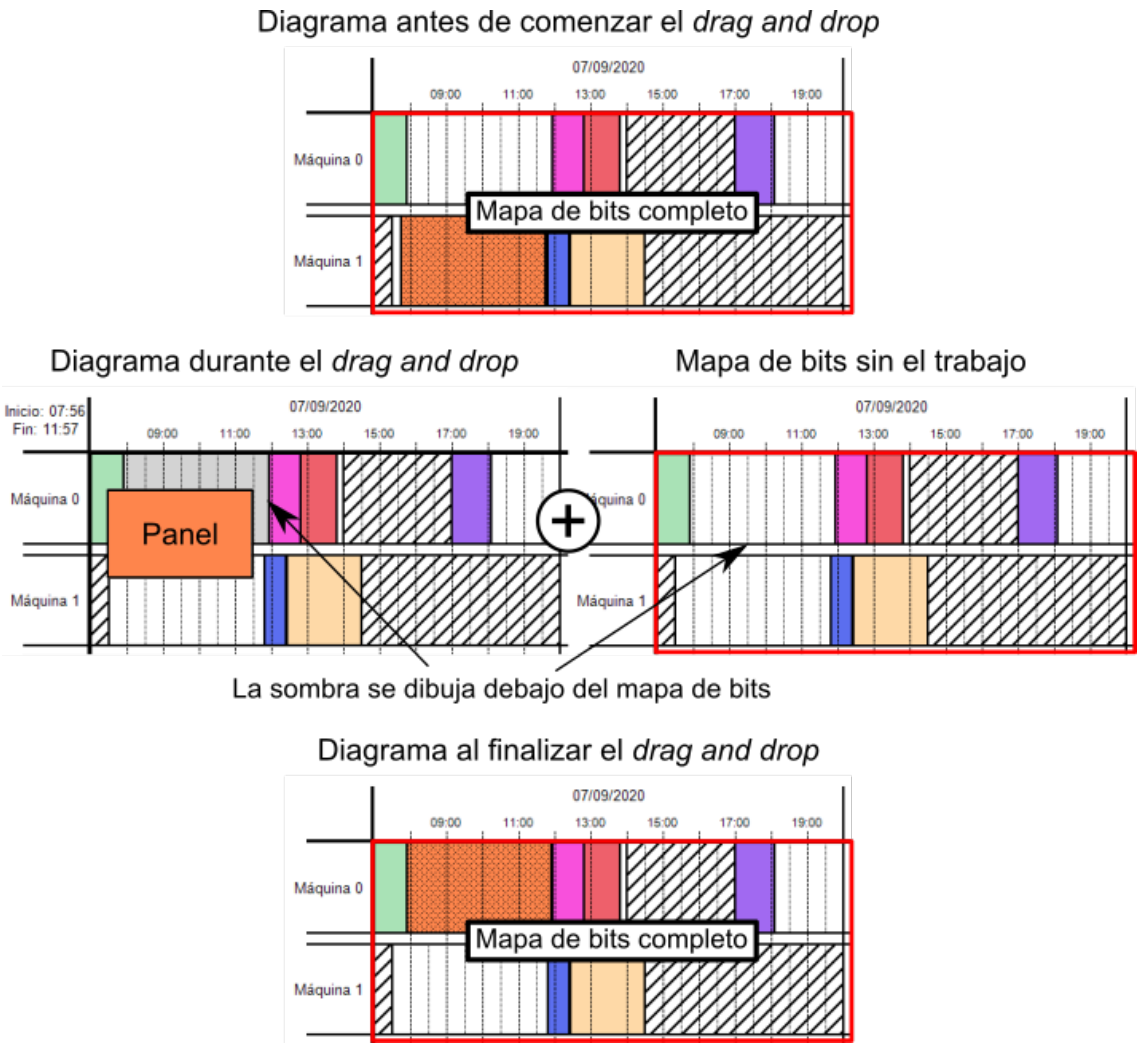

Figura 4.8 Esquema de la elaboración de los gráficos del diagrama para la modificación de la programación de un trabajo mediante el evento *drag and drop*, *elaboración propia*.

el zoom no se creará un bitmap nuevo, sino que se seguirá refrescando a una velocidad razonable. Esto también soluciona el problema al hacer clic en un trabajo, pero provoca que al hacer el *click and drag* sin haber actualizado el diagrama la imagen se siga viendo con retraso.

Existe una solución alternativa que permite mantener el rendimiento también en este caso, esta consiste en actualizar el bitmap, si este no lo está, cuando el que el ratón se desplace sobre el diagrama. Esto no afecta al rendimiento, ya que solamente se actualizará el bitmap la primera vez que entra el ratón en el diagrama, tardando únicamente unos 100ms, en un periodo en el que la aplicación no está ejecutando nada relevante.

Aunque con todas estas medidas aplicadas se obtienen unos resultados de buena calidad, sigue existiendo un problema de *lag* cuando se realiza el *scroll* dentro del evento DragOver al llevar el cursor al borde del diagrama. Esto es debido a que la posición del cursor varía con demasiada frecuencia durante el *scroll* y, en cada una de estas posiciones, se siguen recalculando las posiciones del rectángulo sombra. Además, dado que en estas posiciones lo más común es que el rectángulo sombra no se pueda colocar, el barrido de las posibles posiciones se tiene que hacer al completo, consumiendo un tiempo excesivo, resultando en una tasa de actualización demasiado baja.

Por este motivo, la única solución posible consiste en inhabilitar el método que recalcula el rectángulo sombra durante el desplazamiento del diagrama, reduciendo así el tiempo transcurrido entre actualizaciones de los gráficos. Con esto resuelto los gráficos de la aplicación tienen un calidad lo suficientemente buena como para que no perjudique la experiencia del usuario y le permita realizar cualquier cambio en la planificación sin ninguna dificultad.

4.3.2 Clase ListBoxGantt

Otra clase importante es ListBoxGantt, la cual representa una lista de elementos relevantes para el diagrama de Gantt. Esta es una clase abstracta de la cual heredan las clases ListBoxTrabajo, ListBoxMaquina y ListBoxOperario, en función del elemento del diagrama que busquen representar. En esta aplicación no se emplea el control ListBoxMaquina pero se ha dejado programado en el caso de que se necesitase en un futuro.

Primero, se va explicar el funcionamiento general de la clase base y sus características comunes, para más tarde entrar en detalle en el funcionamiento particular de las clases derivadas. La principal diferencia de este control frente a una ListBox convencional consiste en que la actualización de sus datos se debe realizar mediante el método abstracto ContentUpdate(), aunque para borrarlos se sigue empleando el método Clear(). Este método se debe programar para cada tipo de ListBoxGantt y lo que hace es añadir a cada ListBox el array del elemento correspondiente, borrando antes el contenido de la lista. Además, la clase padre tiene integrado un evento que actualiza los gráficos de todos los controles enlazados al cambiarse el valor de su selección.

A parte de lo ya mencionado, la única clase que difiere del modelo es ListBoxGantt, ya que esta lista incorpora filtro. Este control se conforma por un SplitContainer, que en su parte superior tiene la lista y en la inferior los RadioButtons que indican las opciones de filtrado. El tamaño de la sección inferior se ajusta con el método SetSplitterPosition() y la posición de los RadioButtons se reajusta automáticamente, ya que se encuentran en un FlowLayoutPanel. Para esta clase ha modificado la función ContentUpdate(), de manera que añada los trabajos aplicando el filtro seleccionado en los RadioButtons. Además, en este control siempre se llama a ContentUpdate() antes de actualizar los gráficos.

El control ListBoxGantt también implementa la mecánica *drag and drop* que permite añadir trabajos no asignados al diagrama. Para ello se debe pulsar el botón izquierdo del ratón sobre un trabajo ya seleccionado en la lista que no esté asignado. Una vez hecho esto al mover el ratón se inicia un evento *drag and drop* y, al llevar el cursor hasta el diagrama, ocurrirá lo mismo que cuando se trata de modificar una operación del propio diagrama.

4.3.3 Clase LabelGantt

Para poder representar los datos de los elementos del diagrama es necesario disponer de un control preparado para tal fin. Se ha descartado la opción de emplear la Label predeterminada de Windows Forms debido a que la cantidad de información a representar transcurre en varias líneas con un formato determinado. Aunque el nombre LabelGantt sugiera que esta clase es una Label, este control se trata de un PictureBox que emplea métodos de dibujar *strings* para representar la información. Se ha decidido emplear el nombre LabelGantt porque realmente lo que se está representando es una etiqueta de texto.

Esta clase es abstracta y se heredan de la misma las dos etiquetas de información, LabelGantt-General y LabelGanttElemento, la primera representa datos generales de planificación mientras que la segunda representa datos particulares de cada elemento. El funcionamiento del control base

es bastante sencillo, existe un método abstracto llamado DibujaTextoInformativo(*Graphics lienzo*) que está implementado en el evento Paint del PicutreBox incorporado al control. Este método se encarga de dibujar a información necesaria y además devuelve la altura total del texto representado, la cual se se establece como altura del control, para garantizar que siempre se representa todo el texto.

En el caso de la etiqueta de datos general esto es suficiente, sin embargo para la de los elementos es necesario conocer la clase del elemento seleccionado y, una vez hecho esto, representar los datos en función de la misma. En este control puede ocurrir que, si la etiqueta se comprime demasiado, algunas líneas se salgan de la zona visible por la derecha. Por este motivo es importante fijar el tamaño mínimo al control en el que se encuentre alojado.

Además, para el caso de propiedades que estén conformadas por cadenas de texto muy largas, se ha desarrollado un método con el nombre de DrawStringLines(*var args*) que las divide y las transfiere a una nueva línea. Esto permite que el tamaño mínimo de este control no sea excesivamente grande.

5 Conclusiones

Con la aplicación ya finalizada, es necesario hacer un análisis de la misma, para evaluar si los objetivos planteados han sido alcanzados satisfactoriamente. Asimismo, es recomendable extraer conclusiones del trabajo realizado, para analizar los errores cometidos y evitar repetirlos en un futuro. En este capítulo, también se van a dejar propuestas de mejora y sugerencias de nuevas funcionalidades que se pueden implementar en la aplicación.

La aplicación propuesta cumple los objetivos propuestos con creces, aportando funcionalidades extra a las requeridas por diseño. Se ha obtenido una herramienta que puede implementarse en un entorno productivo genérico y además contiene las funcionalidades necesarias para su utilización en el entorno sanitario. La aplicación no solamente brinda la posibilidad de generar diagramas automáticamente, sino que además permite la modificación de la programación. Esto habilita la posibilidad de incorporar nuevos trabajos en el horizonte o, en el caso del entono quirúrgico, añadir pacientes de urgencia.

La implementación del *drag and drop* tanto en la lista como en el diagrama permiten una modificación intuitiva de la programación, así como la visualización de los trabajos no asignados en el interior de la lista. Asimismo, la incorporación de la mecánica *click and drag* para el desplazamiento en el interior del diagrama es de gran comodidad para el usuario, facilitando la navegación por el mismo.

El hecho de añadir la posibilidad de hacer zoom incrementa notablemente la precisión al programar, esto sumado a las ayudas implementadas en el *drag and drop* permiten al usuario modificar las fechas con exactitud de minutos. La sombra implementada facilita la identificación de los huecos donde los trabajos pueden ser colocados, mostrando al usuario en qué lugares puede liberar el ratón para eliminar la asignación de los trabajos.

La estructura de clases resulta en un código comprensible y altamente reutilizable. Sin embargo, el hecho de que las entidades se almacenen en una lista estática dentro de la clase puede presentar dificultades en el caso de que se intentasen incorporar varios diagramas de Gantt simultáneamente, ya que hay propiedades como Seleccionado que están programadas para funcionar con un único diagrama. Una posible manera de habilitar esto sería almacenar los datos de los trabajos en una lista externa distinta de la estática, aunque implicaría algunas modificaciones en el código. Aun así, emplear varios diagramas de Gantt simultáneamente es extraño puesto que normalmente la planificación se representa solamente en un diagrama, por lo que esto no debería de ser prioritario.

Para la realización de los enlaces entre los controles se ha empleado una metodología que ha proporcionado muy buenos resultados. Al forzar que todos deriven de la clase ControlGantt no

solamente se ha reducido enormemente la cantidad de código necesario, sino que incluir otro tipo de controles en el futuro se podrá realizar de una manera más sencilla. Si se hubiese programado las actualizaciones del resto de controles manualmente, el código no sería reutilizable, ya que los enlaces en una aplicación diferente no funcionarían.

El resultado que se ha obtenido de la representación del diagrama empleando dibujos en lugar de paneles ha superado las expectativas. Si bien es cierto que no se ha podido realizar una prueba de un número significativo de trabajos (del orden de miles), el rendimiento obtenido con esta metodología es mejor que el de la metodología tradicional [Dios et al., 2015]. Además, la escalabilidad de este modelo es clara, por lo que incrementar el tamaño no penalizaría en exceso el rendimiento. No obstante, en caso de que el *lag* fuese excesivo, se podrían emplear distintas técnicas a los gráficos para reducir sus tiempos de ejecución. Un ejemplo podría ser el fragmentar el bitmap del diagrama en diversos trozos para que solamente se actualicen las partes necesarias.

Aunque la forma en la que se ha programado la clase DiagramaGantt ofrece buenos resultados, de cara al futuro sería aconsejable modificar parte de su estructura, principalmente el hecho de que se usen varios paneles. Con la implementación de la metodología de los gráficos en bitmaps, es posible prescindir de estos paneles secundarios y representar el diagrama como fragmentos de un bitmap más grande. En la figura 5.1 se puede apreciar un esquema del funcionamiento del nuevo control, en el PictureBox se representarían de forma adyacente los rectángulos destacados del interior del bitmap completo del diagrama. El ajuste de los *scroll* podría hacerse empleando ScrollBars en lugar de la propiedad AutoScroll, o adaptar la propiedad AutoScroll del panel a la representación de los mapas de bits.

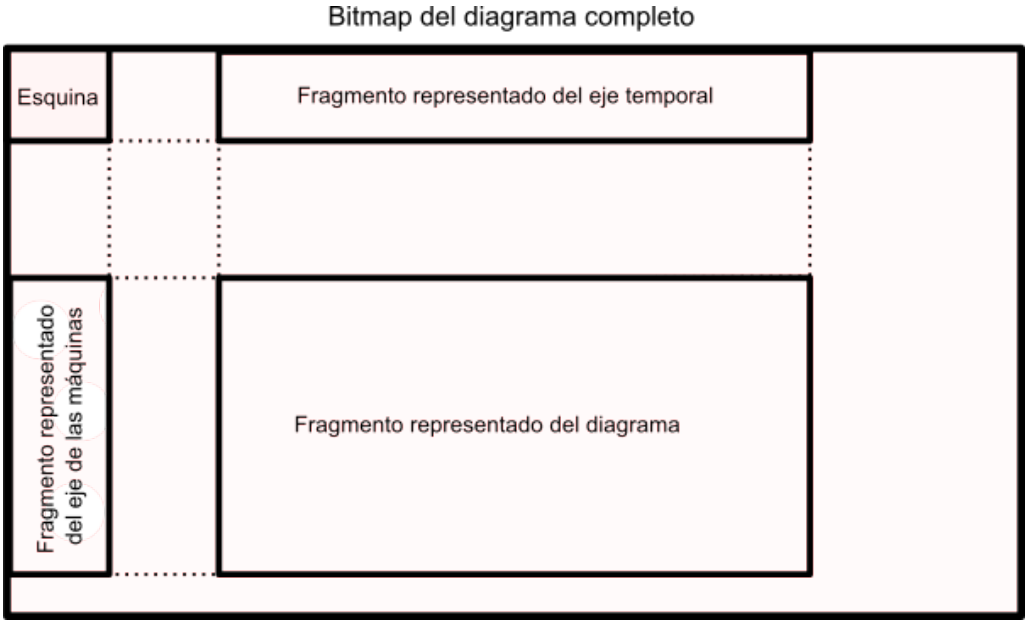

Figura 5.1 Esquema de como podría emplearse un único bitmap y un PictureBox para elaborar el control DiagramaGantt.

Si bien no se sabe a ciencia cierta si esto mejoraría el rendimiento, es muy probable que suceda, especialmente para diagramas grandes, ya que se evitaría la necesidad de tener paneles de gran tamaño, además de reducir el número de controles. Asimismo, dejarían de existir problemas de desincronización entre los ejes y el diagrama puesto que pertenecerían al mismo control.

Continuando con las debilidades del código, la principal carencia que este tiene actualmente es la ausencia de protecciones frente a excepciones. Durante el desarrollo de la aplicación es cierto que han surgido pocos fallos debido a esto, ya que la interacción del usuario con la aplicación es mínima y está muy controlada. Sin embargo, la lectura del archivo de texto permite introducir cualquier dato por lo que se pueden producir todo tipo de excepciones dentro de ese entorno.

El detector de fallos en la lectura que tiene la aplicación actualmente es bastante básico, por lo que sería una buena idea mejorarlo incluyendo también las protecciones frente a excepciones. También se debería de incluir con esto la protección frente a la introducción de fallos en la programación (que dos trabajos estén solapados, por ejemplo) ya que, aunque en este caso se ha confiado en el software, de cara a la elaboración del fichero de forma manual es recomendable implementar alguna protección frente a fallos. Esto sería de total prioridad, sobre todo si se pretende emplear esta aplicación en el ámbito académico, ya que un gran número de alumnos necesitaría producir su propio fichero.

5.1 Posibles mejoras

En esta sección, se van a recopilar distintas sugerencias de mejora para la aplicación, estos consejos no buscan analizar posibles cambios en el código, ya que esto se ha realizado con anterioridad en este capítulo, sino dar ideas para la incorporación de nuevas funcionalidades.

La primera mejora sería la de incorporar un desplegable que aparezca cuando se intente cerrar la aplicación sin haber guardado diciendo si se quieren guardar los cambios. Esto evitaría posibles errores que implicarían la pérdida de todo el progreso y los cambios realizados. Además, se puede incluir un botón de "guardar" junto a "guardar como" que permita sobrescribir el archivo activo, no sin antes mostrar un desplegable que consulte al usuario si desea sobrescribir el archivo. En esta barra de menús también se podría colocar un botón que permita abrir un archivo reciente, y así se evitaría buscarlo en el directorio.

Otra posible incorporación a la aplicación es un apartado de preferencias en el menú. Al pulsarlo aparecería un desplegable en el que se puedan modificar en tiempo de ejecución las propiedades estéticas del diagrama. Los cambios se podrían almacenar en un archivo de preferencias de usuario para que se mantengan al volver a correr el programa.

Incluir un filtro más complejo en las listas puede ser una funcionalidad muy práctica en el caso de tener muchos trabajos. Se podrían filtrar los trabajos en función de las máquinas en las que son procesados, los operarios que los procesan o el día en que son procesados. Otra consideración interesante sería añadir la posibilidad de hacer clic en un hueco sin programar del diagrama y obtener el tiempo libre disponible en ese espacio. Esto facilitaría la selección de los trabajos para incluirlos en los huecos. Por este motivo también sería atractivo incorporar un visor de los tiempos de proceso del trabajo seleccionado en cada máquina.

También sería de gran ayuda incorporar un sistema que impida al usuario programar dos trabajos que tengan asignado al mismo operario en el mismo intervalo temporal, esta restricción es idéntica al hecho de que un trabajo no puede tener dos operaciones asignadas en el mismo intervalo temporal. Se podrían resaltar todos los trabajos de ese operario en el diagrama durante el *drag and drop*, ya sea mediante colores o dibujos, para que sea más sencillo identificar los lugares en los cuales se puede programar el trabajo seleccionado.

Por último, una funcionalidad muy práctica sería la de permitir añadir y eliminar trabajos desde la propia aplicación. De esta manera se podría editar plenamente el diagrama, y, para el caso de los quirófanos, se podrían añadir pacientes de urgencia. Además, la aplicación facilitaría la creación del archivo añadiendo trabajos a un fichero que únicamente tiene máquinas.

Apéndice A
Código empleado

A.1 Ventana principal de la aplicación

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace GanttDiagramGenerator
{
    static partial class Program
    {
        /// <summary>
        /// Punto de entrada principal para la aplicación.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new VentanaPrincipal());
        }
    }
}
```

Código A.1 Código empleado para inicializar el programa.

```
using System;

namespace GanttDiagramGenerator
{
    partial class VentanaPrincipal
    {
        /// <summary>
        /// Variable del diseñador necesaria.
        /// </summary>
        private System.ComponentModel.IContainer components = null;
```

```
        /// <summary>
        /// Limpiar los recursos que se estén usando.
        /// </summary>
        /// <param name="disposing">true si los recursos administrados se
            deben desechar; false en caso contrario.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Código generado por el Diseñador de Windows Forms

        /// <summary>
        /// Método necesario para admitir el Diseñador. No se puede modificar
        /// el contenido de este método con el editor de código.
        /// </summary>
        private void InitializeComponent()
        {
            this.dialogEleccionArchivo = new
                System.Windows.Forms.OpenFileDialog();
            this.menuPrincipal = new System.Windows.Forms.MenuStrip();
            this.archivoToolStripMenuItem = new
                System.Windows.Forms.ToolStripMenuItem();
            this.abrirToolStripMenuItem = new
                System.Windows.Forms.ToolStripMenuItem();
            this.guardarComoToolStripMenuItem = new
                System.Windows.Forms.ToolStripMenuItem();
            this.dialogGuardarArchivo = new
                System.Windows.Forms.SaveFileDialog();
            this.scVerticalPrincipal = new
                System.Windows.Forms.SplitContainer();
            this.tcListBox = new System.Windows.Forms.TabControl();
            this.pageTrabajos = new System.Windows.Forms.TabPage();
            this.listBoxTrabajosPrincipal = new
                GanttDiagramGenerator.ListBoxTrabajos();
            this.pageOperarios = new System.Windows.Forms.TabPage();
            this.listBoxOperariosPrincipal = new
                GanttDiagramGenerator.ListBoxOperarios();
            this.scHorizontalPrincipal = new
                System.Windows.Forms.SplitContainer();
            this.diagramaPrincipal = new GanttDiagramGenerator.DiagramaGantt();
            this.scInferior = new System.Windows.Forms.SplitContainer();
            this.InfoBoxElementos = new
                GanttDiagramGenerator.LabelGanttElementos();
            this.scEsquina = new System.Windows.Forms.SplitContainer();
            this.InfoBoxGeneral = new GanttDiagramGenerator.LabelGanttGeneral();
            this.ZoomBar = new System.Windows.Forms.TrackBar();
            this.lblZoom = new System.Windows.Forms.Label();
            this.menuPrincipal.SuspendLayout();
            ((System.ComponentModel.ISupportInitialize)(
                this.scVerticalPrincipal)).BeginInit();
            this.scVerticalPrincipal.Panel1.SuspendLayout();
```

```
            this.scVerticalPrincipal.Panel2.SuspendLayout();
            this.scVerticalPrincipal.SuspendLayout();
            this.tcListBox.SuspendLayout();
            this.pageTrabajos.SuspendLayout();
            this.pageOperarios.SuspendLayout();
            ((System.ComponentModel.ISupportInitialize)(
                this.scHorizontalPrincipal)).BeginInit();
            this.scHorizontalPrincipal.Panel1.SuspendLayout();
            this.scHorizontalPrincipal.Panel2.SuspendLayout();
            this.scHorizontalPrincipal.SuspendLayout();
            ((System.ComponentModel.ISupportInitialize)(
                this.scInferior)).BeginInit();
            this.scInferior.Panel1.SuspendLayout();
            this.scInferior.Panel2.SuspendLayout();
            this.scInferior.SuspendLayout();
            ((System.ComponentModel.ISupportInitialize)
                (this.scEsquina)).BeginInit();
            this.scEsquina.Panel1.SuspendLayout();
            this.scEsquina.Panel2.SuspendLayout();
            this.scEsquina.SuspendLayout();
            ((System.ComponentModel.ISupportInitialize)(this.ZoomBar)).BeginInit();
            this.SuspendLayout();
            //
            // dialogEleccionArchivo
            //
            this.dialogEleccionArchivo.Filter = "Archivos de texto
                (*.txt)|*.txt";
            this.dialogEleccionArchivo.Title = "Eleccion Archivo";
            this.dialogEleccionArchivo.FileOk += new
                System.ComponentModel.CancelEventHandler(
                this.dialogEleccionArchivo_FileOk);
            //
            // menuPrincipal
            //
            this.menuPrincipal.BackColor = System.Drawing.SystemColors.Control;
            this.menuPrincipal.Font = new System.Drawing.Font("Arial", 9F);
            this.menuPrincipal.Items.AddRange(new
                System.Windows.Forms.ToolStripItem[] {
            this.archivoToolStripMenuItem});
            this.menuPrincipal.LayoutStyle =
                System.Windows.Forms.ToolStripLayoutStyle.HorizontalStackWithOverflow;
            this.menuPrincipal.Location = new System.Drawing.Point(0, 0);
            this.menuPrincipal.Name = "menuPrincipal";
            this.menuPrincipal.Size = new System.Drawing.Size(1264, 24);
            this.menuPrincipal.TabIndex = 2;
            this.menuPrincipal.Text = "Menú pRINCIPAL";
            //
            // archivoToolStripMenuItem
            //
            this.archivoToolStripMenuItem.DropDownItems.AddRange(new
                System.Windows.Forms.ToolStripItem[] {
            this.abrirToolStripMenuItem,
            this.guardarComoToolStripMenuItem});
            this.archivoToolStripMenuItem.Name = "archivoToolStripMenuItem";
            this.archivoToolStripMenuItem.Size = new System.Drawing.Size(58,
                20);
```

```
            this.archivoToolStripMenuItem.Text = "Archivo";
            //
            // abrirToolStripMenuItem
            //
            this.abrirToolStripMenuItem.Name = "abrirToolStripMenuItem";
            this.abrirToolStripMenuItem.Size = new System.Drawing.Size(156, 22);
            this.abrirToolStripMenuItem.Text = "Abrir";
            this.abrirToolStripMenuItem.Click += new
                System.EventHandler(this.abrirToolStripMenuItem_Click);
            //
            // guardarComoToolStripMenuItem
            //
            this.guardarComoToolStripMenuItem.Name =
                "guardarComoToolStripMenuItem";
            this.guardarComoToolStripMenuItem.Size = new
                System.Drawing.Size(156, 22);
            this.guardarComoToolStripMenuItem.Text = "Guardar Como";
            this.guardarComoToolStripMenuItem.Click += new
                System.EventHandler(this.guardarComoToolStripMenuItem_Click);
            //
            // dialogGuardarArchivo
            //
            this.dialogGuardarArchivo.Filter = "Archivos de texto
                (*.txt)|*.txt";
            this.dialogGuardarArchivo.Title = "Guardar archivo";
            this.dialogGuardarArchivo.FileOk += new
                System.ComponentModel.CancelEventHandler(
                this.dialogGuardarArchivo_FileOk);
            //
            // scVerticalPrincipal
            //
            this.scVerticalPrincipal.Dock = System.Windows.Forms.DockStyle.Fill;
            this.scVerticalPrincipal.Location = new System.Drawing.Point(0, 24);
            this.scVerticalPrincipal.Name = "scVerticalPrincipal";
            //
            // scVerticalPrincipal.Panel1
            //
            this.scVerticalPrincipal.Panel1.BackColor =
                System.Drawing.SystemColors.Control;
            this.scVerticalPrincipal.Panel1.Controls.Add(this.tcListBox);
            this.scVerticalPrincipal.Panel1MinSize = 140;
            //
            // scVerticalPrincipal.Panel2
            //
            this.scVerticalPrincipal.Panel2.Controls.Add(
                this.scHorizontalPrincipal);
            this.scVerticalPrincipal.Panel2MinSize = 870;
            this.scVerticalPrincipal.Size = new System.Drawing.Size(1264, 657);
            this.scVerticalPrincipal.SplitterDistance = 140;
            this.scVerticalPrincipal.TabIndex = 3;
            this.scVerticalPrincipal.SplitterMoved += new
                System.Windows.Forms.SplitterEventHandler(
                this.scVerticalPrincipal_SplitterMoved);
            //
            // tcListBox
            //
```

```
            this.tcListBox.Controls.Add(this.pageTrabajos);
            this.tcListBox.Controls.Add(this.pageOperarios);
            this.tcListBox.Dock = System.Windows.Forms.DockStyle.Fill;
            this.tcListBox.Font = new System.Drawing.Font("Arial", 9F);
            this.tcListBox.Location = new System.Drawing.Point(0, 0);
            this.tcListBox.Name = "tcListBox";
            this.tcListBox.SelectedIndex = 0;
            this.tcListBox.Size = new System.Drawing.Size(140, 657);
            this.tcListBox.TabIndex = 0;
            //
            // pageTrabajos
            //
            this.pageTrabajos.BackColor = System.Drawing.SystemColors.Control;
            this.pageTrabajos.Controls.Add(this.listBoxTrabajosPrincipal);
            this.pageTrabajos.Location = new System.Drawing.Point(4, 24);
            this.pageTrabajos.Name = "pageTrabajos";
            this.pageTrabajos.Padding = new System.Windows.Forms.Padding(3);
            this.pageTrabajos.Size = new System.Drawing.Size(132, 629);
            this.pageTrabajos.TabIndex = 0;
            this.pageTrabajos.Text = "Trabajos";
            //
            // listBoxTrabajosPrincipal
            //
            this.listBoxTrabajosPrincipal.AllowDrop = true;
            this.listBoxTrabajosPrincipal.Dock =
                System.Windows.Forms.DockStyle.Fill;
            this.listBoxTrabajosPrincipal.Location = new
                System.Drawing.Point(3, 3);
            this.listBoxTrabajosPrincipal.Name = "listBoxTrabajosPrincipal";
            this.listBoxTrabajosPrincipal.Size = new System.Drawing.Size(126,
                623);
            this.listBoxTrabajosPrincipal.TabIndex = 0;
            this.listBoxTrabajosPrincipal.Text = "listBoxTrabajosPrincipal";
            //
            // pageOperarios
            //
            this.pageOperarios.BackColor = System.Drawing.SystemColors.Control;
            this.pageOperarios.Controls.Add(this.listBoxOperariosPrincipal);
            this.pageOperarios.Location = new System.Drawing.Point(4, 24);
            this.pageOperarios.Name = "pageOperarios";
            this.pageOperarios.Padding = new System.Windows.Forms.Padding(3);
            this.pageOperarios.Size = new System.Drawing.Size(132, 629);
            this.pageOperarios.TabIndex = 1;
            this.pageOperarios.Text = "Operarios";
            //
            // listBoxOperariosPrincipal
            //
            this.listBoxOperariosPrincipal.Dock =
                System.Windows.Forms.DockStyle.Fill;
            this.listBoxOperariosPrincipal.Location = new
                System.Drawing.Point(3, 3);
            this.listBoxOperariosPrincipal.Name = "listBoxOperariosPrincipal";
            this.listBoxOperariosPrincipal.Size = new System.Drawing.Size(126,
                623);
            this.listBoxOperariosPrincipal.TabIndex = 0;
            this.listBoxOperariosPrincipal.Text = "listBoxOperariosPrincipal";
```

```
            //
            // scHorizontalPrincipal
            //
            this.scHorizontalPrincipal.BorderStyle =
                System.Windows.Forms.BorderStyle.Fixed3D;
            this.scHorizontalPrincipal.Dock =
                System.Windows.Forms.DockStyle.Fill;
            this.scHorizontalPrincipal.Location = new System.Drawing.Point(0,
                0);
            this.scHorizontalPrincipal.Name = "scHorizontalPrincipal";
            this.scHorizontalPrincipal.Orientation =
                System.Windows.Forms.Orientation.Horizontal;
            //
            // scHorizontalPrincipal.Panel1
            //
            this.scHorizontalPrincipal.Panel1.BackColor =
                System.Drawing.Color.White;
            this.scHorizontalPrincipal.Panel1.Controls.Add(
                this.diagramaPrincipal);
            this.scHorizontalPrincipal.Panel1MinSize = 350;
            //
            // scHorizontalPrincipal.Panel2
            //
            this.scHorizontalPrincipal.Panel2.Controls.Add(this.scInferior);
            this.scHorizontalPrincipal.Panel2MinSize = 150;
            this.scHorizontalPrincipal.Size = new System.Drawing.Size(1120,
                657);
            this.scHorizontalPrincipal.SplitterDistance = 450;
            this.scHorizontalPrincipal.TabIndex = 1;
            this.scHorizontalPrincipal.SplitterMoved += new
                System.Windows.Forms.SplitterEventHandler(
                this.scHorizontalPrincipal_SplitterMoved);
            //
            // diagramaPrincipal
            //
            this.diagramaPrincipal.AltoEjeDias = 60;
            this.diagramaPrincipal.AltoMaquina = 80;
            this.diagramaPrincipal.AnchoEjeMaquinas = 80;
            this.diagramaPrincipal.AnchoHora = 40;
            this.diagramaPrincipal.Dock = System.Windows.Forms.DockStyle.Fill;
            this.diagramaPrincipal.ExcesoDerechoLineasHorizontales = 10;
            this.diagramaPrincipal.ExcesoInferiorLineasVerticales = 10;
            this.diagramaPrincipal.ExcesoIzquierdoLineasHorizontales = 60;
            this.diagramaPrincipal.ExcesoLineasSecundarias = 10;
            this.diagramaPrincipal.ExcesoSuperiorLineasVerticales = 50;
            this.diagramaPrincipal.Font = new System.Drawing.Font("Arial", 10F,
                System.Drawing.FontStyle.Regular,
                System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.diagramaPrincipal.Location = new System.Drawing.Point(0, 0);
            this.diagramaPrincipal.Name = "diagramaPrincipal";
            this.diagramaPrincipal.SeparacionDia = 20;
            this.diagramaPrincipal.SeparacionMaquina = 10;
            this.diagramaPrincipal.Size = new System.Drawing.Size(1116, 446);
            this.diagramaPrincipal.TabIndex = 0;
            this.diagramaPrincipal.Text = "diagramaPrincipal";
            this.diagramaPrincipal.Visible = false;
```

```
            //
            // scInferior
            //
            this.scInferior.BorderStyle =
                System.Windows.Forms.BorderStyle.Fixed3D;
            this.scInferior.Dock = System.Windows.Forms.DockStyle.Fill;
            this.scInferior.IsSplitterFixed = true;
            this.scInferior.Location = new System.Drawing.Point(0, 0);
            this.scInferior.MinimumSize = new System.Drawing.Size(800, 100);
            this.scInferior.Name = "scInferior";
            //
            // scInferior.Panel1
            //
            this.scInferior.Panel1.AutoScroll = true;
            this.scInferior.Panel1.BackColor = System.Drawing.Color.White;
            this.scInferior.Panel1.Controls.Add(this.InfoBoxElementos);
            this.scInferior.Panel1MinSize = 550;
            //
            // scInferior.Panel2
            //
            this.scInferior.Panel2.Controls.Add(this.scEsquina);
            this.scInferior.Panel2MinSize = 200;
            this.scInferior.Size = new System.Drawing.Size(1120, 203);
            this.scInferior.SplitterDistance = 685;
            this.scInferior.TabIndex = 2;
            //
            // InfoBoxElementos
            //
            this.InfoBoxElementos.CausesValidation = false;
            this.InfoBoxElementos.Dock = System.Windows.Forms.DockStyle.Top;
            this.InfoBoxElementos.Font = new System.Drawing.Font("Arial", 10F,
                System.Drawing.FontStyle.Regular,
                System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.InfoBoxElementos.Location = new System.Drawing.Point(0, 0);
            this.InfoBoxElementos.Margenes = new System.Drawing.Point(10, 10);
            this.InfoBoxElementos.Name = "InfoBoxElementos";
            this.InfoBoxElementos.Size = new System.Drawing.Size(681, 15);
            this.InfoBoxElementos.TabIndex = 0;
            this.InfoBoxElementos.Text = "InfoBoxElementos";
            //
            // scEsquina
            //
            this.scEsquina.BorderStyle =
                System.Windows.Forms.BorderStyle.Fixed3D;
            this.scEsquina.Dock = System.Windows.Forms.DockStyle.Fill;
            this.scEsquina.FixedPanel = System.Windows.Forms.FixedPanel.Panel2;
            this.scEsquina.IsSplitterFixed = true;
            this.scEsquina.Location = new System.Drawing.Point(0, 0);
            this.scEsquina.Name = "scEsquina";
            this.scEsquina.Orientation =
                System.Windows.Forms.Orientation.Horizontal;
            //
            // scEsquina.Panel1
            //
            this.scEsquina.Panel1.AutoScroll = true;
            this.scEsquina.Panel1.BackColor = System.Drawing.Color.White;
```

```
            this.scEsquina.Panel1.Controls.Add(this.InfoBoxGeneral);
            this.scEsquina.Panel1MinSize = 20;
            //
            // scEsquina.Panel2
            //
            this.scEsquina.Panel2.BackColor = System.Drawing.Color.White;
            this.scEsquina.Panel2.Controls.Add(this.ZoomBar);
            this.scEsquina.Panel2.Controls.Add(this.lblZoom);
            this.scEsquina.Panel2MinSize = 80;
            this.scEsquina.Size = new System.Drawing.Size(431, 203);
            this.scEsquina.SplitterDistance = 119;
            this.scEsquina.TabIndex = 0;
            //
            // InfoBoxGeneral
            //
            this.InfoBoxGeneral.Dock = System.Windows.Forms.DockStyle.Top;
            this.InfoBoxGeneral.Font = new System.Drawing.Font("Arial", 10F,
                System.Drawing.FontStyle.Regular,
                System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.InfoBoxGeneral.Location = new System.Drawing.Point(0, 0);
            this.InfoBoxGeneral.Margenes = new System.Drawing.Point(10, 10);
            this.InfoBoxGeneral.Name = "InfoBoxGeneral";
            this.InfoBoxGeneral.Size = new System.Drawing.Size(427, 100);
            this.InfoBoxGeneral.TabIndex = 0;
            this.InfoBoxGeneral.Text = "InfoBoxGeneral";
            //
            // ZoomBar
            //
            this.ZoomBar.BackColor = System.Drawing.Color.White;
            this.ZoomBar.Dock = System.Windows.Forms.DockStyle.Bottom;
            this.ZoomBar.Enabled = false;
            this.ZoomBar.Location = new System.Drawing.Point(0, 31);
            this.ZoomBar.Maximum = 30;
            this.ZoomBar.Minimum = 2;
            this.ZoomBar.Name = "ZoomBar";
            this.ZoomBar.Size = new System.Drawing.Size(427, 45);
            this.ZoomBar.TabIndex = 2;
            this.ZoomBar.Value = 2;
            this.ZoomBar.ValueChanged += new
                System.EventHandler(this.ZoomBar_ValueChanged);
            //
            // lblZoom
            //
            this.lblZoom.AutoSize = true;
            this.lblZoom.Font = new System.Drawing.Font("Arial", 10F);
            this.lblZoom.Location = new System.Drawing.Point(188, 12);
            this.lblZoom.Name = "lblZoom";
            this.lblZoom.Size = new System.Drawing.Size(43, 16);
            this.lblZoom.TabIndex = 3;
            this.lblZoom.Text = "Zoom";
            this.lblZoom.TextAlign =
                System.Drawing.ContentAlignment.MiddleCenter;
            //
            // VentanaPrincipal
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
```

```
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(1264, 681);
            this.Controls.Add(this.scVerticalPrincipal);
            this.Controls.Add(this.menuPrincipal);
            this.DoubleBuffered = true;
            this.Name = "VentanaPrincipal";
            this.StartPosition =
                System.Windows.Forms.FormStartPosition.CenterScreen;
            this.Text = "Gantt Diagram Generator";
            this.ResizeEnd += new
                System.EventHandler(this.VentanaPrincipal_Resize);
            this.Resize += new
                System.EventHandler(this.VentanaPrincipal_Resize);
            this.menuPrincipal.ResumeLayout(false);
            this.menuPrincipal.PerformLayout();
            this.scVerticalPrincipal.Panel1.ResumeLayout(false);
            this.scVerticalPrincipal.Panel2.ResumeLayout(false);
            ((System.ComponentModel.ISupportInitialize)(
                this.scVerticalPrincipal)).EndInit();
            this.scVerticalPrincipal.ResumeLayout(false);
            this.tcListBox.ResumeLayout(false);
            this.pageTrabajos.ResumeLayout(false);
            this.pageOperarios.ResumeLayout(false);
            this.scHorizontalPrincipal.Panel1.ResumeLayout(false);
            this.scHorizontalPrincipal.Panel2.ResumeLayout(false);
            ((System.ComponentModel.ISupportInitialize)(
                this.scHorizontalPrincipal)).EndInit();
            this.scHorizontalPrincipal.ResumeLayout(false);
            this.scInferior.Panel1.ResumeLayout(false);
            this.scInferior.Panel2.ResumeLayout(false);
            ((System.ComponentModel.ISupportInitialize)(this.scInferior)).EndInit();
            this.scInferior.ResumeLayout(false);
            this.scEsquina.Panel1.ResumeLayout(false);
            this.scEsquina.Panel2.ResumeLayout(false);
            this.scEsquina.Panel2.PerformLayout();
            ((System.ComponentModel.ISupportInitialize)(this.scEsquina)).EndInit();
            this.scEsquina.ResumeLayout(false);
            ((System.ComponentModel.ISupportInitialize)(this.ZoomBar)).EndInit();
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.OpenFileDialog dialogEleccionArchivo;
        private System.Windows.Forms.MenuStrip menuPrincipal;
        private System.Windows.Forms.ToolStripMenuItem
            archivoToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem abrirToolStripMenuItem;
        private System.Windows.Forms.SplitContainer scVerticalPrincipal;
        private System.Windows.Forms.SplitContainer scHorizontalPrincipal;
        private System.Windows.Forms.TabControl tcListBox;
        private System.Windows.Forms.TabPage pageTrabajos;
        private System.Windows.Forms.TabPage pageOperarios;
        private System.Windows.Forms.TrackBar ZoomBar;
```

```
        private System.Windows.Forms.SplitContainer scInferior;
        private System.Windows.Forms.SplitContainer scEsquina;
        private System.Windows.Forms.Label lblZoom;
        private System.Windows.Forms.ToolStripMenuItem
            guardarComoToolStripMenuItem;
        private System.Windows.Forms.SaveFileDialog dialogGuardarArchivo;
        private LabelGanttGeneral InfoBoxGeneral;
        private ListBoxTrabajos listBoxTrabajosPrincipal;
        private ListBoxOperarios listBoxOperariosPrincipal;
        private LabelGanttElementos InfoBoxElementos;
        private DiagramaGantt diagramaPrincipal;
    }
}
```

Código A.2 Código producido por el diseñador de Visual Studio para inicializar la clase ventana principal.

```
using System;
using System.IO;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace GanttDiagramGenerator
{
    public partial class VentanaPrincipal : Form
    {
        private string rutaArchivoActivo;

        public VentanaPrincipal()
        {
            InitializeComponent();

            dialogEleccionArchivo.InitialDirectory =
                Directory.GetParent(Environment.CurrentDirectory).Parent.FullName
                + @"\FicherosEntrada";
            dialogGuardarArchivo.InitialDirectory =
                Directory.GetParent(Environment.CurrentDirectory).Parent.FullName
                + @"\FicherosEntrada";

            listBoxTrabajosPrincipal.SetSplitterPosition();

            ControlGantt.Link(new ControlGantt[] { diagramaPrincipal,
                listBoxTrabajosPrincipal, listBoxOperariosPrincipal,
                InfoBoxElementos, InfoBoxGeneral});
            ReubicalblZoom();
            ReubicalblDatosGenerales();
        }

        private void ResetDatos()
```

```
        {
            Trabajo.Clear();
            Maquina.Clear();
            Operario.Clear();
        }

        private void abrirToolStripMenuItem_Click(object sender, EventArgs e)
        {
            dialogEleccionArchivo.ShowDialog();
        }

        private void guardarComoToolStripMenuItem_Click(object sender,
            EventArgs e)
        {
            if (rutaArchivoActivo != null) dialogGuardarArchivo.ShowDialog();
        }

        private void dialogEleccionArchivo_FileOk(object sender,
            CancelEventArgs e)
        {
            if (rutaArchivoActivo == null)
            {

                rutaArchivoActivo = dialogEleccionArchivo.FileName;
                Program.LecturaEntrada(rutaArchivoActivo);
                ZoomBar.Value = diagramaPrincipal.AnchoHora / 8;
                ZoomBar.Enabled = true;
                diagramaPrincipal.Visible = true;
            }
            else
            {
                ResetDatos();
                rutaArchivoActivo = dialogEleccionArchivo.FileName;
                Program.LecturaEntrada(rutaArchivoActivo);
                InfoBoxElementos.Invalidate(true);
                diagramaPrincipal.DesactualizarBitmap();
            }

            diagramaPrincipal.AjustarGeometriaPaneles();
            diagramaPrincipal.Refresh();
            listBoxOperariosPrincipal.ContentUpdate();
            listBoxTrabajosPrincipal.ContentUpdate();
            InfoBoxGeneral.Invalidate(true);

        }

        private void scHorizontalPrincipal_SplitterMoved(object sender,
            SplitterEventArgs e)
        {
            if (rutaArchivoActivo != null )
                diagramaPrincipal.AjustarGeometriaPaneles();
        }

        private void ZoomBar_ValueChanged(object sender, EventArgs e)
        {
            diagramaPrincipal.AnchoHora = ZoomBar.Value * 8;
```

```
        }

        private void ReubicalblZoom()
        {
            int anchoContenedor = scEsquina.Width;
            lblZoom.Location = new Point(anchoContenedor / 2 - lblZoom.Width /
                2, lblZoom.Location.Y);
        }

        private void ReubicalblDatosGenerales()
        {
            int anchoContenedor = scEsquina.Width;
        }

        private void scVerticalPrincipal_SplitterMoved(object sender,
            SplitterEventArgs e)
        {
            if (rutaArchivoActivo != null)
                diagramaPrincipal.AjustarGeometriaPaneles();
            listBoxTrabajosPrincipal.SetSplitterPosition();
            ReubicalblZoom();
            ReubicalblDatosGenerales();
        }

        private void dialogGuardarArchivo_FileOk(object sender,
            CancelEventArgs e)
        {
            if (dialogEleccionArchivo.FileName != "" && rutaArchivoActivo !=
                null)
            {
                rutaArchivoActivo = dialogGuardarArchivo.FileName;
                Program.EscrituraSalida(rutaArchivoActivo);
            }
        }

        private void VentanaPrincipal_Resize(object sender, EventArgs e)
        {
            diagramaPrincipal.AjustarGeometriaPaneles();
        }
    }
}
```

Código A.3 Código empleado para los EventHandlers de los controles de la aplicación (excluyedo a los heredados de ControlGantt), así como la inicialización de la ventana principal.

A.2 Clase Trabajo

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.ComponentModel;
using System.Data.Common;
using System.Linq;
using System.Text;
using System.Diagnostics.Contracts;
using System.Windows.Forms;

namespace GanttDiagramGenerator
{
    public partial class Trabajo : IDatosIdentificativos, ISeleccionable
    {
        public Trabajo( int[] rutaFabricacion)
        {
            identificador = staticListaTrabajos.Count;
            staticListaTrabajos.Add(this);

            this.rutaFabricacion = rutaFabricacion;
        }
        public Trabajo(DateTime[] fechasInicio, int[] rutaFabricacion) :
            this(rutaFabricacion)
        {
            for (var i = 0; i < fechasInicio.Length; i++)
            {
                if (DateTime.Compare(fechasInicio[i], StaticOrigenProgramacion)
                    >= 0) AddOperacion(fechasInicio[i], i);
            }
        }
        public Trabajo(int[] rutaFabricacion, DateTime[] fechasFin) :
            this(rutaFabricacion)
        {
            for (var i = 0; i < fechasFin.Length; i++)
            {
                if (DateTime.Compare(fechasFin[i], StaticOrigenProgramacion) >=
                    0) AddOperacion(i, fechasFin[i]);
            }
        }

        public override string ToString()
        {
            if (Nombre != null) return Nombre;
            else return $"Trabajo {identificador}";
        }

        public string GetInfoString()
        {
            string mensaje = "Trabajo " + identificador;
            if (Nombre != null) mensaje += "\n" + "Nombre: " + Nombre;
            int i = 1;
            foreach (Operacion elemento in listaOperacionesPlanificadas)
            {
```

```
                if(elemento != null)
                {
                    mensaje +=$"\nOperacion {i}\n{elemento}";
                    i++;
                }

            }
            foreach (Operario elemento in listaOperariosAsignados)
            {
                if (elemento != null)
                {
                    mensaje += $"\n{elemento}";
                }

            }
            if (Descripcion != null) mensaje += "\n" + "Descripcion: " +
                Descripcion +"\n";
            return mensaje;
        }

        public string[] ToStringArray()
        {
            List<string> lineasArray = new List<string>();
            lineasArray.Add($"Trabajo {identificador}");
            if (Nombre != null) lineasArray.Add($"Nombre: {Nombre}");
            for (var i = 0; i < listaOperacionesPlanificadas.Count; i++)
            {
                lineasArray.Add($"Operacion {i}");
                lineasArray.Add($"Maquina asignada:
                    {GetIDMaquinaOperacion(i)}");
                lineasArray.Add($"Fecha de inicio: {GetInicioOperacion(i)} ,
                    Fecha de finalización: {GetFinOperacion(i)} , Tiempo de
                    proceso: {GetTiempoProcesoOperacion(i)}");
            }
            if (Descripcion != null) lineasArray.Add($"Descripción:
                {Descripcion}");
            return lineasArray.ToArray();
        }

        public void SetIdentificador(int identificador) => this.identificador
            = identificador;

        public string Descripcion { get; set; }

        public string Nombre { get; set; }

        public int Identificador { get => identificador; }
        private int identificador;

        private static List<Trabajo> staticListaTrabajos = new List<Trabajo>();

        public Color ColorDibujo
        { get
            {
                if (colorDibujo == Color.Black)
                {
```

```
                AsignarColor();
            }
            return colorDibujo;
        }
    }

    private Color colorDibujo = Color.Black;

    private void AsignarColor()
    {
        bool esUnico;
        bool esAdecuado;
        Color colorAleatorio;
        Random rndSeed = new Random();
        Random rndRed = new Random(rndSeed.Next(1000000));
        Random rndBlue = new Random(rndSeed.Next(100000));
        Random rndGreen = new Random(rndSeed.Next(100000));
        int i = 0;
        do
        {
            esUnico = true;
            colorAleatorio = Color.FromArgb(rndRed.Next(256),
                rndGreen.Next(256), rndBlue.Next(256));
            if (colorAleatorio.GetBrightness() > 0.60 &&
                colorAleatorio.GetBrightness() < 0.95 &&
                colorAleatorio.GetSaturation() > 0.4)
            {
                esAdecuado = true;
                foreach (Trabajo elemento in Trabajo.GetArray())
                {
                    if (colorAleatorio.R < elemento.colorDibujo.R +
                        diferenciaMinimaColor &&
                        colorAleatorio.R > elemento.colorDibujo.R -
                            diferenciaMinimaColor &&
                        colorAleatorio.G < elemento.colorDibujo.G +
                            diferenciaMinimaColor &&
                        colorAleatorio.G > elemento.colorDibujo.G -
                            diferenciaMinimaColor &&
                        colorAleatorio.B < elemento.colorDibujo.B +
                            diferenciaMinimaColor &&
                        colorAleatorio.B > elemento.colorDibujo.B -
                            diferenciaMinimaColor)
                    {
                        esUnico = false;
                    }
                }
            }
            else
            {
                esAdecuado = false;
            }

            if (i>5000)
            {
                DiferenciaMinimaColor = diferenciaMinimaColor - 1;
            }
```

```
            i++;
        } while (!esUnico || !esAdecuado);
        colorDibujo = colorAleatorio;

    }

    static int DiferenciaMinimaColor
    {
        set
        {
            if (value <= 0)
            {
                diferenciaMinimaColor = 0;
                MessageBox.Show("Se han agotado todos los colores posibles");
            }
        }
    }
    static int diferenciaMinimaColor = 20;

    public bool Seleccionado
    {
        get { return seleccionado; }
        set
        {
            if (value)
            {
            foreach (Trabajo elemento in Trabajo.GetArray())
                elemento.Seleccionado = false;
            foreach (Maquina elemento in Maquina.GetArray())
                elemento.Seleccionado = false;
            foreach (Operario elemento in Operario.GetArray())
                elemento.Seleccionado = false;
            }
            seleccionado = value;
        }
    }


    private bool seleccionado = false;

    public static Trabajo GetTrabajoSeleccionado()
    {
        foreach (Trabajo elemento in staticListaTrabajos)
        {
            if (elemento.Seleccionado) return elemento;
        }

        return null;
    }

    public bool Resaltado
    {
        get
        {
            if (seleccionado) return true;
            else
```

```
            {
                foreach (Operario elemento in listaOperariosAsignados)
                {
                    if (elemento.Seleccionado) return true;
                }
            }
            return false;
        }
    }

    public static int GetTrabajosNoProgramados()
    {
        int trabajosNoProgramados = 0;
        foreach (Trabajo elemento in staticListaTrabajos)
        {
            if (!elemento.Programado) trabajosNoProgramados++;
        }
        return trabajosNoProgramados;
    }

    public static int GetTrabajosProgramados()
    {
        return staticListaTrabajos.Count - GetTrabajosNoProgramados();
    }

    public bool Programado
    {
        get
        {
            if (listaOperacionesPlanificadas.Count ==
                RutaFabricacion.Length) return true;
            return false;
        }
    }

    public int[] RutaFabricacion { get => rutaFabricacion; }

    private int[] rutaFabricacion;

    public int[] RutaFabricacionProgramada
    {
        get
        {
            List<int> rutaFabricacion = new List<int>();
            foreach (Operacion elemento in listaOperacionesPlanificadas)
            {
                rutaFabricacion.Add(elemento.GetIdentificadorMaquinaAsignada());
            }
            return rutaFabricacion.ToArray();
        }
    }

    public bool SecuenciaRigida
    {
        get
        {
```

```
                bool rigidez = true;
                foreach (int elemento in RutaFabricacion) if (elemento < 0)
                    rigidez = false;

                if (!rigidez)
                {
                    bool error = false;
                    for(var i = 0; i < RutaFabricacion.Length; i++)
                    {
                        if (RutaFabricacion[i] > 0)
                        {
                            RutaFabricacion[i] = -RutaFabricacion[i];
                            error = true;
                        }
                    }
                    if (error) MessageBox.Show($"Error en la secuencia deseada
                        del trabajo {identificador}," +
                                $" se establece como secuencia flexible");
                }
                return rigidez;
            }
        }

        public static Trabajo Get(int identificador) =>
            staticListaTrabajos[identificador];
        public static Trabajo[] GetArray() => staticListaTrabajos.ToArray();

        public static void Clear() => Trabajo.staticListaTrabajos.Clear();

        public static DateTime StaticOrigenProgramacion { set; get; }

        public Operario[] GetOperarios() => listaOperariosAsignados.ToArray();

        private List<Operario> listaOperariosAsignados = new List<Operario>();
        public void AddOperario(int identificadorOperario) =>
            listaOperariosAsignados.Add(Operario.Get(identificadorOperario));
    }
}
```

Código A.4 Código empleado para elaborar la parte principal de la clase Trabajo.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace GanttDiagramGenerator
{
    partial class Trabajo
    {
        public void AddOperacion(DateTime fechaInicio, int
            identificadorMaquinaAsignada)
        {
            listaOperacionesPlanificadas.Add(new Operacion(identificador,
                fechaInicio, identificadorMaquinaAsignada));
```

```
            listaOperacionesPlanificadas.Sort((opA, opB) =>
                opA.GetFechaInicio().CompareTo(opB.GetFechaInicio()));
        }

        public void AddOperacion(int identificadorMaquinaAsignada, DateTime
            fechaFin)
        {
            DateTime fechaInicio =
                fechaFin.Subtract(Maquina.Get(identificadorMaquinaAsignada).GetTiempoProceso(identifi
            listaOperacionesPlanificadas.Add(new Operacion(identificador,
                fechaInicio, identificadorMaquinaAsignada));
            listaOperacionesPlanificadas.Sort((opA, opB) =>
                opA.GetFechaInicio().CompareTo(opB.GetFechaInicio()));
        }

        private static bool CheckOperacionValida(DateTime fechaInicio,
            TimeSpan tiempoProceso, int identificadorMaquina)
        {
            DateTime fechaFin = fechaInicio.Add(tiempoProceso);
            if (fechaInicio.Hour <
                Math.Floor(Maquina.GetAperturaMasTemprana())) return false;
            if (fechaInicio.Hour + 1 >
                Math.Ceiling(Maquina.GetCierreMasTardio()) &&
                fechaInicio.Minute > 0) return false;
            if (fechaInicio.Hour > Math.Ceiling(Maquina.GetCierreMasTardio()))
                return false;
            if (fechaFin.Hour < Math.Floor(Maquina.GetAperturaMasTemprana()))
                return false;
            if (fechaFin.Hour + 1 > Math.Ceiling(Maquina.GetCierreMasTardio())
                && fechaFin.Minute > 0) return false;
            if (fechaFin.Hour > Math.Ceiling(Maquina.GetCierreMasTardio()) )
                return false;
            if (fechaFin.Day != fechaInicio.Day && (fechaFin.Minute != 0 ||
                fechaFin.Hour != 0)) return false;

            DateTime[] arrayFechasInicio =
                GetFechasInicioMaquina(identificadorMaquina);
            DateTime[] arrayFechasFin =
                GetFechasFinMaquina(identificadorMaquina);

            for (var i = 0; i < arrayFechasInicio.Length; i ++)
            {
                if (fechaFin > arrayFechasInicio[i] && fechaInicio <
                    arrayFechasFin[i]) return false;
            }
            return true;
        }

        private static bool CheckOperacionValida(TimeSpan tiempoProceso,
            DateTime fechaFin, int identificadorMaquina)
        {
            return CheckOperacionValida(fechaFin.Subtract(tiempoProceso),
                tiempoProceso, identificadorMaquina);
        }
```

```
        public bool CheckOperacionValidaTrabajo(DateTime fechaInicio, TimeSpan
            tiempoProceso, int identificadorMaquina)
        {
            if (Programado) return false;

            DateTime fechaFin = fechaInicio.Add(tiempoProceso);

            if (fechaInicio < StaticOrigenProgramacion) return false;
            if (fechaFin > StaticOrigenProgramacion.AddDays(
                Maquina.StaticNumeroDiasProgramacion)) return false;

            foreach (Operacion elemento in listaOperacionesPlanificadas)
            {
                if (identificadorMaquina ==
                    elemento.GetIdentificadorMaquinaAsignada()) return false;

                if (SecuenciaRigida)
                {
                    int ordenOperacionRevisada =
                        GetOrdenMaquina(identificadorMaquina);
                    int ordenOperacionComparada =
                        GetOrdenMaquina(elemento.GetIdentificadorMaquinaAsignada());

                    if (fechaFin <= elemento.GetFechaFin() &&
                        ordenOperacionRevisada > ordenOperacionComparada)
                    {
                        return false;
                    }
                    else if (fechaInicio >= elemento.GetFechaInicio() &&
                        ordenOperacionRevisada < ordenOperacionComparada)
                    {
                        return false;
                    }
                    else if (fechaFin > elemento.GetFechaInicio() && fechaInicio
                        < elemento.GetFechaFin())
                    {
                        return false;
                    }

                }
                else if (rutaFabricacion.Length > 1)
                {
                    List<int> maquinasRecorridas = new List<int>();
                    bool maquinaPendiente = false;

                    foreach (int id in rutaFabricacion)
                        maquinasRecorridas.Add(id * -1);

                    foreach (int id in maquinasRecorridas)
                    {
                        if (id == identificadorMaquina) maquinaPendiente = true;
                    }

                    if (!maquinaPendiente) return false;
```

```
            if (fechaFin > elemento.GetFechaInicio() && fechaInicio <
                elemento.GetFechaFin()) return false;
        }
    }
    return CheckOperacionValida(fechaInicio, tiempoProceso,
        identificadorMaquina);
}

public bool CheckOperacionValidaTrabajo(TimeSpan tiempoProceso,
    DateTime fechaFin, int identificadorMaquina)
{
    return
        CheckOperacionValidaTrabajo(fechaFin.Subtract(tiempoProceso),
        tiempoProceso, identificadorMaquina);
}

private int GetOrdenMaquina(int identificadorMaquina)
{
    for (var i = 0; i < rutaFabricacion.Length; i++)
    {
        if (rutaFabricacion[i] == identificadorMaquina) return i;
    }

    return -1;
}

public static DateTime[] GetFechasInicioMaquina(int
    identificadorMaquina)
{
    List<DateTime> listaFechasInicio = new List<DateTime>();
    foreach (Trabajo elemento in staticListaTrabajos)
    {
        Operacion[] operacionesAsignadas =
            elemento.GetOperacionesMaquina(identificadorMaquina);
        foreach (Operacion op in operacionesAsignadas)
            listaFechasInicio.Add(op.GetFechaInicio());
    }
    return listaFechasInicio.ToArray();
}

public static DateTime[] GetFechasFinMaquina(int identificadorMaquina)
{
    List<DateTime> listaFechasFin = new List<DateTime>();
    foreach (Trabajo elemento in staticListaTrabajos)
    {
        Operacion[] operacionesAsignadas =
            elemento.GetOperacionesMaquina(identificadorMaquina);
        foreach(Operacion op in operacionesAsignadas)
            listaFechasFin.Add(op.GetFechaFin());
    }
    return listaFechasFin.ToArray();
}

private Operacion[] GetOperacionesMaquina(int identificadorMaquina)
{
    List<Operacion> listaOperaciones = new List<Operacion>();
```

```
            for (var i = 0; i < GetNumeroOperacionesAsignadas(); i++)
            {
                if (GetIDMaquinaOperacion(i) == identificadorMaquina)
                {
                    listaOperaciones.Add(listaOperacionesPlanificadas[i]);
                }
            }
            return listaOperaciones.ToArray();
        }

        public DateTime GetInicioOperacion(int ordenOperacion) =>
            listaOperacionesPlanificadas[ordenOperacion].GetFechaInicio();
        public DateTime GetFinOperacion(int ordenOperacion) =>
            listaOperacionesPlanificadas[ordenOperacion].GetFechaFin();

        public TimeSpan GetTiempoProcesoOperacion(int ordenOperacion) =>
            listaOperacionesPlanificadas[ordenOperacion].GetTiempoProceso();
        public int GetIDMaquinaOperacion(int ordenOperacion) =>
            listaOperacionesPlanificadas[
            ordenOperacion].GetIdentificadorMaquinaAsignada();
        public int GetNumeroOperacionesAsignadas() =>
            listaOperacionesPlanificadas.Count;
        public void BorraOperacion(int indiceOperacion)
        {
            if (indiceOperacion >= 0)
                listaOperacionesPlanificadas.RemoveAt(indiceOperacion);
        }

        private List<Operacion> listaOperacionesPlanificadas = new
            List<Operacion>();

        private class Operacion
        {
            public Operacion(int identificadorTrabajoAsociado, DateTime
                fechaInicio, int identificadorMaquinaAsignada)
            {
                this.identificadorMaquinaAsignada= identificadorMaquinaAsignada;
                this.fechaInicio = fechaInicio;
                tiempoProceso =
                    Maquina.Get(identificadorMaquinaAsignada).GetTiempoProceso(
                    identificadorTrabajoAsociado);
                fechaFin = fechaInicio.Add(tiempoProceso);
            }

            public Operacion(int identificadorTrabajoAsociado, int
                identificadorMaquinaAsignada, DateTime fechaFin)
            {
                this.identificadorMaquinaAsignada =
                    identificadorMaquinaAsignada;
                this.fechaFin = fechaFin;
                tiempoProceso =
                    Maquina.Get(identificadorMaquinaAsignada).GetTiempoProceso(
                    identificadorTrabajoAsociado);
                fechaInicio = fechaFin.Subtract(tiempoProceso);
            }
```

```
            public override string ToString()
            {
                string mensaje = $"Máquina asignada:
                    {identificadorMaquinaAsignada} \n" ;
                mensaje += $"Fecha de inicio: {fechaInicio} , Fecha de
                    finalización: {fechaFin} , Tiempo de proceso:
                    {tiempoProceso}";

                return mensaje;
            }

            public DateTime GetFechaInicio() => fechaInicio;

            public DateTime GetFechaFin() => fechaFin;

            public TimeSpan GetTiempoProceso() => tiempoProceso;

            public int GetIdentificadorMaquinaAsignada() =>
                identificadorMaquinaAsignada;

            private DateTime fechaFin;

            private DateTime fechaInicio;

            private TimeSpan tiempoProceso;

            private int identificadorMaquinaAsignada;

            private int identificadorTrabajoAsociado;
        }

    }
}
```

Código A.5 Código empleado para elaborar todo lo relacionado con operaciones de la clase Trabajo.

A.3 Clase Maquina

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Dynamic;
using System.Linq;
using System.Text;

namespace GanttDiagramGenerator
{
    public partial class Maquina : IDatosIdentificativos, ISeleccionable,
        IElementoProductivo
    {
        public Maquina(TimeSpan[] tiemposProceso)
        {
            this.tiemposProcesoIdeales = new List<TimeSpan>(tiemposProceso);
            identificador = staticListaMaquinas.Count;
            staticListaMaquinas.Add(this);
            for (var i = 0; i < StaticNumeroDiasProgramacion ; i++)
                listaHorarios.Add(new Horario(false));
        }

        public override string ToString()
        {
            string mensaje = "Máquina " + identificador;
            if (Nombre != null) mensaje += "\n" + "Nombre: " + Nombre;
            mensaje += "\n" + "Tiempos de proceso:";
            foreach (TimeSpan elemento in TiemposProceso) mensaje += " " +
                elemento.TotalMinutes.ToString("0.#");
            if (Rendimiento != 1) mensaje += "\n" + "Rendimiento: " +
                Rendimiento.ToString("0.000");
            if (Descripcion != null) mensaje += "\n" + "Descripcion: " +
                Descripcion +"\n";

            return mensaje;
        }

        public string Descripcion { get; set; }

        public string Nombre { get; set; }

        public double Rendimiento { get => rendimiento; set => rendimiento =
            value; }
        private double rendimiento = 1;

        public bool Seleccionado
        {
            get { return seleccionado; }
            set
            {
                if (value)
                {
                    foreach (Trabajo elemento in Trabajo.GetArray())
                        elemento.Seleccionado = false;
```

```
                foreach (Maquina elemento in Maquina.GetArray())
                    elemento.Seleccionado = false;
                foreach (Operario elemento in Operario.GetArray())
                    elemento.Seleccionado = false;
            }
            seleccionado = value;
        }
    }
    private bool seleccionado = false;

    private List<TimeSpan> tiemposProcesoIdeales;
    private List<TimeSpan> TiemposProceso { get =>
        AplicaRendimiento(tiemposProcesoIdeales, Rendimiento);}

    private List<TimeSpan> AplicaRendimiento(List<TimeSpan>
        tiempoProcesoIdeal, double Rendimiento)
    {
        List<TimeSpan> tiemposProceso = new List<TimeSpan>();
        foreach (TimeSpan elemento in tiempoProcesoIdeal)
        {
            tiemposProceso.Add(new TimeSpan(0,
                (int)Math.Ceiling(Rendimiento * elemento.TotalMinutes), 0));
        }

        return tiemposProceso;
    }

    public TimeSpan GetTiempoProcesoIdeal(int identificadorTrabajo) =>
        tiemposProcesoIdeales[identificadorTrabajo];
    public TimeSpan GetTiempoProceso(int identificadorTrabajo) =>
        AplicaRendimiento(tiemposProcesoIdeales,
        Rendimiento)[identificadorTrabajo];

    public static Maquina Get(int identificador) =>
        staticListaMaquinas[identificador];
    public static double GetTiempoTotalDisponible()
    {
        double tiempoTotal = 0;
        foreach (Maquina elemento in staticListaMaquinas)
        {
            tiempoTotal += elemento.GetTiempoDisponible();
        }
        return tiempoTotal;
    }

    public static double GetTiempoTotalProgramado()
    {
        double tiempoTotal = 0;
        foreach (Maquina elemento in staticListaMaquinas)
        {
            tiempoTotal += elemento.GetTiempoProgramado();
        }
        return tiempoTotal;
    }

    public static Maquina[] GetArray() => staticListaMaquinas.ToArray();
```

```
        public static void Clear() => staticListaMaquinas.Clear();
        public int Identificador { get { return identificador; } }
        private int identificador;

        public static int StaticNumeroDiasProgramacion { get; set;}

        private static List<Maquina> staticListaMaquinas = new List<Maquina>();
    }
}
```

Código A.6 Código empleado para elaborar la parte principal de la clase Maquina.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GanttDiagramGenerator
{
    public partial class Maquina
    {
        private List<Horario> listaHorarios = new List<Horario>();

        public double GetTiempoDisponible()
        {
            double tiempoAbierto = 0;
            foreach (Horario elemento in listaHorarios)
            {
                if (elemento.Abierto)
                {
                    TimeSpan[] aperturas = elemento.GetAperturas();
                    TimeSpan[] cierres = elemento.GetCierres();

                    for (var i = 0; i < aperturas.Length; i++)
                    {
                        tiempoAbierto += (cierres[i] -
                            aperturas[i]).TotalMinutes;
                    }
                }
            }
            return tiempoAbierto;
        }

        public double GetTiempoProgramado()
        {
            double tiempoOcupado = 0;
            foreach (Trabajo elemento in Trabajo.GetArray())
            {
                for (var i = 0; i < elemento.GetNumeroOperacionesAsignadas();
                    i++)
                {
                    if (elemento.GetIDMaquinaOperacion(i) == identificador)
                    {
```

```
                tiempoOcupado +=
                    elemento.GetTiempoProcesoOperacion(i).TotalMinutes;
            }
        }
    }
    return tiempoOcupado;
}
public TimeSpan[] GetAperturas(int diasDesdeOrigen)
{
    if (listaHorarios[diasDesdeOrigen].Abierto)
    {
        return listaHorarios[diasDesdeOrigen].GetAperturas();
    }
    else
    {
        return new TimeSpan[0];
    }
}

public TimeSpan[] GetCierres(int diasDesdeOrigen)
{
    if (listaHorarios[diasDesdeOrigen].Abierto)
    {
        return listaHorarios[diasDesdeOrigen].GetCierres();
    }
    else
    {
        return new TimeSpan[0];
    }
}

public static double GetAperturaMasTemprana()
{
    double horaMasTemprana = 24;
    foreach (Maquina elemento in Maquina.staticListaMaquinas)
    {
        for (var i = 0; i < StaticNumeroDiasProgramacion; i++)
        {
            if (elemento.GetHorario(i).Abierto)
            {
                horaMasTemprana = Math.Min(horaMasTemprana,
                    elemento.GetAperturas(i)[0].TotalHours);
            }
        }
    }
    return horaMasTemprana;
}

public static double GetCierreMasTardio()
{
    double horaMasTardia = 0;
    foreach (Maquina elemento in Maquina.staticListaMaquinas)
    {
        for (var i = 0; i < StaticNumeroDiasProgramacion; i++)
        {
            if (elemento.GetHorario(i).Abierto)
```

```
                    {
                        horaMasTardia = Math.Max(horaMasTardia,
                            elemento.GetCierres(i)[
                            elemento.GetCierres(i).GetUpperBound(0)].TotalHours);
                    }
                }
            }
            return horaMasTardia;
        }

        public void SetHorario(int diasDesdeOrigen, TimeSpan[] listaHoras)
        {
            listaHorarios[diasDesdeOrigen].SetHorario(listaHoras);
            if (listaHoras.Length == 0) listaHorarios[diasDesdeOrigen].Abierto
                = false;
            else listaHorarios[diasDesdeOrigen].Abierto = true;
        }
        private Horario GetHorario(int diasDesdeorigen) =>
            listaHorarios[diasDesdeorigen];

        private class Horario
        {
            public Horario(bool abierto)
            {
                Abierto = abierto;
            }
            public void SetHorario(TimeSpan[] listaHoras)
            {
                aperturas.Clear();
                cierres.Clear();

                for (var i = 0; i < listaHoras.Length; i++)
                {
                    if (i % 2 == 0) aperturas.Add(listaHoras[i]);
                    else cierres.Add(listaHoras[i]);
                }
            }
            public TimeSpan[] GetAperturas() => aperturas.ToArray();
            public TimeSpan[] GetCierres() => cierres.ToArray();

            private List<TimeSpan> aperturas = new List<TimeSpan>();

            private List<TimeSpan> cierres = new List<TimeSpan>();
            public bool Abierto { get; set; }
        }
    }
}
```

Código A.7 Código empleado para elaborar todo lo relacionado con horarios de la clase Maquina.

A.4 Clase Operario

```
using System;
using System.Collections.Generic;
using System.Text;

namespace GanttDiagramGenerator
{
    public class Operario : IDatosIdentificativos, ISeleccionable
    {
        public Operario()
        {
            identificador = staticListaOperarios.Count;
            staticListaOperarios.Add(this);
        }

        public override string ToString()
        {
            if (Nombre != null) return Nombre;
            else return "Operario " + identificador;
        }

        public int GetTrabajosAsignados()
        {
            int trabajosAsignados = 0;
            foreach(Trabajo elemento in Trabajo.GetArray())
            {
                foreach (Operario op in elemento.GetOperarios())
                {
                    if (identificador == op.identificador) trabajosAsignados++;
                }
            }

            return trabajosAsignados;
        }

        public static Operario Get(int identificador) =>
            staticListaOperarios[identificador];

        public static Operario[] GetArray() => staticListaOperarios.ToArray();

        public string Descripcion { get; set; }

        public string Nombre { get; set; }

        public int Identificador { get { return identificador; } }
        private int identificador;

        public bool Seleccionado
        {
            get { return seleccionado; }
            set
            {
                if (value)
                {
```

```
                    foreach (Trabajo elemento in Trabajo.GetArray())
                        elemento.Seleccionado = false;
                    foreach (Maquina elemento in Maquina.GetArray())
                        elemento.Seleccionado = false;
                    foreach (Operario elemento in Operario.GetArray())
                        elemento.Seleccionado = false;
                }
                seleccionado = value;
            }
        }
        private bool seleccionado = false;

        private static List<Operario> staticListaOperarios = new
            List<Operario>();

        public static void Clear() => staticListaOperarios.Clear();
    }
}
```

Código A.8 Código empleado para elaborar la clase Operario.

A.5 Interfaces implementadas

```
using System;
using System.Collections.Generic;
using System.Text;

namespace GanttDiagramGenerator
{
    public interface IDatosIdentificativos
    {
        string Descripcion { set; get; }
        string Nombre { set; get; }
        int Identificador { get; }
    }


    public interface IElementoProductivo
    {
        double Rendimiento { set; get; }
    }

    public interface ISeleccionable
    {
        bool Seleccionado { set; get; }
    }
}
```

Código A.9 Código que define las interfaces que han de implementar las clases Trabajo, Maquina y Operario.

A.6 Lectura y escritura de los ficheros

```
using System;
using System.Collections.Generic;
using System.Diagnostics.CodeAnalysis;
using System.Diagnostics.Contracts;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Text;
using System.Windows.Forms;

namespace GanttDiagramGenerator
{
    static partial class Program
    {
        public static void LecturaEntrada(string rutaArchivoActivo)
        {
            StreamReader lectorArchivo = new StreamReader(rutaArchivoActivo,
                Encoding.GetEncoding("iso-8859-1"));
            string linea;
            string[] arrayLinea;
            int contador = 0;

            arrayLinea = lectorArchivo.ReadLine().Trim().Split(' ');
            contador++;
            Trabajo.StaticOrigenProgramacion = new
                DateTime(Int32.Parse(arrayLinea[2]),
                Int32.Parse(arrayLinea[1]), Int32.Parse(arrayLinea[0]));

            arrayLinea = lectorArchivo.ReadLine().Trim().Split( );
            contador++;
            int numeroTrabajos = int.Parse(arrayLinea[0]);
            int numeroMaquinas = int.Parse(arrayLinea[1]);
            Maquina.StaticNumeroDiasProgramacion = int.Parse(arrayLinea[2]);

            do
            {
                linea = lectorArchivo.ReadLine();
                if (linea != null) linea = linea.Trim();
                contador++;
                switch (linea)
                {
                    case "Maquina":
                        for (var i = 0; i < numeroMaquinas; i++)
                        {
                            arrayLinea = lectorArchivo.ReadLine().Trim().Split('
                                ');
                            contador++;
                            TimeSpan[] tiemposProceso =
                                Array.ConvertAll(arrayLinea, a => new TimeSpan(0,
                                int.Parse(a), 0));
                            _ = new Maquina(tiemposProceso);
                        }
```

```
                    do
                    {
                        linea = lectorArchivo.ReadLine().Trim();
                        contador++;
                        switch (linea)
                        {
                            case "Horario":
                                foreach (Maquina elemento in
                                    Maquina.GetArray())
                                {
                                    arrayLinea =
                                        lectorArchivo.ReadLine().Trim().Split('&');
                                    contador++;
                                    int diasDesdeOrigen = 0;
                                    foreach (string stringHoras in arrayLinea)
                                    {
                                        bool existeHorario = true;
                                        foreach (string fragmento in
                                            stringHoras.Trim().Split(' '))
                                        {
                                            if (fragmento == "") existeHorario
                                                = false;
                                        }
                                        if (existeHorario)
                                        {
                                            TimeSpan[] arrayHoras =
                                                Array.ConvertAll(stringHoras.Trim(
                                                ).Split(' '),
                                               s => TimeSpan.FromHours(
                                                   double.Parse(s,
                                                   NumberStyles.AllowDecimalPoint)));
                                            elemento.SetHorario(diasDesdeOrigen,
                                                arrayHoras);
                                        }
                                        diasDesdeOrigen++;
                                    }
                                }
                                break;

                            case "Nombre":
                                arrayLinea =
                                    lectorArchivo.ReadLine().Trim().Split('&');
                                for (var i = 0; i < arrayLinea.Length; i++)
                                    arrayLinea[i] = arrayLinea[i].Trim();
                                ActualizaNombres(Maquina.GetArray(),
                                    arrayLinea);
                                contador++;
                                break;

                            case "Descripcion":
                                arrayLinea = new string[numeroMaquinas];
                                for (var i = 0; i < numeroMaquinas; i++)
                                {
                                    arrayLinea[i] = lectorArchivo.ReadLine();
                                    contador++;
                                }
```

```
                                ActualizaDescripciones(Maquina.GetArray(),
                                    arrayLinea);
                                break;

                            case "Rendimiento":
                                arrayLinea =
                                    lectorArchivo.ReadLine().Trim().Split('
                                    ');
                                double[] arrayRendimientos =
                                    Array.ConvertAll(arrayLinea, a =>
                                    double.Parse(a,
                                    NumberStyles.AllowDecimalPoint));
                                ActualizaRendimientos(Maquina.GetArray(),
                                    arrayRendimientos);
                                contador++;
                                break;

                            case "":
                                break;

                            default:
                                MessageBox.Show($"Error en la línea
                                    {contador}.");
                                break;
                        }
                    } while (linea != "" && linea != null);
                    break;

                case "Operario":
                    int numeroOperarios =
                        int.Parse(lectorArchivo.ReadLine().Trim());
                    contador++;
                    for (var i = 0; i < numeroOperarios; i++) _ = new
                        Operario();
                    do
                    {
                        linea = lectorArchivo.ReadLine().Trim();
                        contador++;
                        switch (linea)
                        {
                            case "Nombre":
                                arrayLinea =
                                    lectorArchivo.ReadLine().Trim().Split('&');
                                for (var i = 0; i < arrayLinea.Length; i++)
                                    arrayLinea[i] = arrayLinea[i].Trim();
                                ActualizaNombres(Operario.GetArray(),
                                    arrayLinea);
                                contador++;
                                break;

                            case "Descripcion":
                                arrayLinea = new string[numeroOperarios];
                                for (var i = 0; i < numeroOperarios; i++)
                                {
                                    arrayLinea[i] = lectorArchivo.ReadLine();
                                    contador++;
```

```
                        }
                        ActualizaDescripciones(Operario.GetArray(),
                            arrayLinea);
                        break;

                    case "":
                        break;

                    default:
                        MessageBox.Show($"Error en la línea
                            {contador}.");
                        break;
                }
            } while (linea != "" && linea != null);
            break;

        case "Trabajo":
            linea = lectorArchivo.ReadLine().Trim();
            contador++;
            List<int[]> secuenciaMaquinas = new List<int[]>();
            if (linea == "RutaFabricacion")
            {
                for (var i = 0; i < numeroTrabajos; i++)
                {
                    arrayLinea =
                        lectorArchivo.ReadLine().Trim().Split(' ');
                    secuenciaMaquinas.Add(Array.ConvertAll(arrayLinea,
                        int.Parse));
                    contador++;
                }
                linea = lectorArchivo.ReadLine().Trim();
                contador++;
            }
            else
            {
                for (var i = 0; i < numeroTrabajos; i++)
                    secuenciaMaquinas.Add(new int[] { -1 });
            }

            switch (linea)
            {
                case "FechaFin":
                    for (var i = 0; i < numeroTrabajos; i++)
                    {
                        arrayLinea =
                            lectorArchivo.ReadLine().Trim().Split('
                            ');
                        contador++;
                        TimeSpan[] tiempoDesdeOrigen =
                            Array.ConvertAll(arrayLinea, a => new
                            TimeSpan(0, int.Parse(a), 0));
                        DateTime[] fechasFin =
                            Array.ConvertAll(tiempoDesdeOrigen, a =>
                            Trabajo.StaticOrigenProgramacion.Add(a));
                        _ = new Trabajo(secuenciaMaquinas[i],
                            fechasFin);
```

```
                    }
                    break;
                case "FechaInicio":
                    for (var i = 0; i < numeroTrabajos; i++)
                    {
                        arrayLinea =
                            lectorArchivo.ReadLine().Trim().Split('
                            ');
                        contador++;
                        TimeSpan[] tiempoDesdeOrigen =
                            Array.ConvertAll(arrayLinea, a => new
                            TimeSpan(0, int.Parse(a), 0));
                        DateTime[] fechasInicio =
                            Array.ConvertAll(tiempoDesdeOrigen, a =>
                            Trabajo.StaticOrigenProgramacion.Add(a));
                        _ = new Trabajo(fechasInicio,
                            secuenciaMaquinas[i]);
                    }
                    break;

                case "":
                    break;

                default:
                    MessageBox.Show($"Error en la línea {contador}.");
                 break;
            }
            do
            {
                linea = lectorArchivo.ReadLine();
                if (linea != null) linea = linea.Trim();
                contador++;
                switch (linea)
                {
                    case "Nombre":
                        arrayLinea =
                            lectorArchivo.ReadLine().Trim().Split('&');
                        for (var i = 0; i < arrayLinea.Length; i++)
                            arrayLinea[i] = arrayLinea[i].Trim();
                        ActualizaNombres(Trabajo.GetArray(),
                            arrayLinea);
                        contador++;
                        break;

                    case "Descripcion":
                        arrayLinea = new string[numeroTrabajos];
                        for (var i = 0; i < numeroTrabajos; i++)
                        {
                            arrayLinea[i] =
                                lectorArchivo.ReadLine().Trim();
                            contador++;
                        }
                        ActualizaDescripciones(Trabajo.GetArray(),
                            arrayLinea);
                        break;
```

```
                        case "OperarioAsignado":

                            linea = lectorArchivo.ReadLine().Trim();
                            contador++;
                            switch (linea)
                            {
                                case "ID":
                                    for (var i = 0; i <
                                        Trabajo.GetArray().Length; i++)
                                    {
                                        arrayLinea =
                                            lectorArchivo.ReadLine().Trim(
                                            ).Split(' ');
                                        contador++;
                                        foreach (string elemento in
                                            arrayLinea)
                                        {
                                            if (elemento != "")
                                                Trabajo.Get(
                                                i).AddOperario(int.Parse(
                                                elemento));
                                        }
                                    }
                                    break;

                                default:
                                    break;
                            }

                            break;

                        case "":
                            break;

                        case null:
                            break;

                        default:
                            MessageBox.Show($"Error en la línea
                                {contador}.");
                            break;
                    }
                } while (linea != "" && linea != null);
                break;

            case "":
                break;

            case null:
                break;

            default:
                MessageBox.Show($"Error en la línea {contador}.");
                break;
        }
    } while (linea != null);
```

```
            lectorArchivo.Close();
        }
        static void ActualizaNombres<T>(T[] arrayObjetos, string[]
            nuevosNombres) where T : IDatosIdentificativos
        {
            for (var i = 0; i < arrayObjetos.Length; i++)
                arrayObjetos[i].Nombre = nuevosNombres[i];
        }
        static void ActualizaDescripciones<T>(T[] arrayObjetos, string[]
            nuevasDescripciones) where T : IDatosIdentificativos
        {
            for(var i = 0; i < arrayObjetos.Length; i++)
                arrayObjetos[i].Descripcion = nuevasDescripciones[i];
        }
        static void ActualizaRendimientos<T>(T[] arrayObjetos, double[]
            nuevosRendimientos) where T : IElementoProductivo
        {
            for (var i = 0; i < arrayObjetos.Length; i++)
                arrayObjetos[i].Rendimiento = nuevosRendimientos[i];
        }
    }
}
```

Código A.10 Código empleado para leer el fichero de entrada.

```
using System;
using System.Collections.Generic;
using System.Diagnostics.CodeAnalysis;
using System.Diagnostics.Contracts;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Text;
using System.Windows.Forms;

namespace GanttDiagramGenerator
{
    static partial class Program
    {
        public static void EscrituraSalida(string rutaArchivo)
        {
            StreamWriter escritorArchivo = new StreamWriter(rutaArchivo, false,
                Encoding.GetEncoding("iso-8859-1"));

            DateTime origenProgramacion = Trabajo.StaticOrigenProgramacion;

            escritorArchivo.WriteLine($"{origenProgramacion.Day}
                {origenProgramacion.Month} {origenProgramacion.Year}");
            escritorArchivo.WriteLine($"{Trabajo.GetArray().Length}
                {Maquina.GetArray().Length}
                {Maquina.StaticNumeroDiasProgramacion}");

            escritorArchivo.WriteLine();
```

```
            escritorArchivo.WriteLine("Maquina");
            foreach (Maquina elemento in Maquina.GetArray())
            {
                for (var i = 0; i < Trabajo.GetArray().Length; i++)
                {
                    TimeSpan tiempoProceso = elemento.GetTiempoProceso(i);
                    if (i != 0) escritorArchivo.Write(' ');
                    escritorArchivo.Write(
                        $"{(int)Math.Round(tiempoProceso.TotalMinutes)}");
                }
                escritorArchivo.Write(escritorArchivo.NewLine);
            }

            escritorArchivo.WriteLine("Horario");
            foreach (Maquina elemento in Maquina.GetArray())
            {
                for (var i = 0; i < Maquina.StaticNumeroDiasProgramacion; i++)
                {
                    TimeSpan[] aperturas = elemento.GetAperturas(i);
                    TimeSpan[] cierres = elemento.GetCierres(i);

                    if (i != 0) escritorArchivo.Write('&');

                    if (aperturas.Length == 0 || cierres.Length == 0)
                    {
                        escritorArchivo.Write(' ');
                    }
                    else
                    {
                        for (var j = 0; j < aperturas.Length; j++)
                        {
                            escritorArchivo.Write($"{aperturas[j].TotalHours}
                                {cierres[j].TotalHours} ");
                        }
                    }
                }
                escritorArchivo.Write(escritorArchivo.NewLine);
            }
            EscribeNombres(Maquina.GetArray(), escritorArchivo);
            EscribeDescripciones(Maquina.GetArray(), escritorArchivo);
            escritorArchivo.WriteLine("Rendimiento");
            for (var i = 0; i < Maquina.GetArray().Length; i++)
            {
                if (i != 0) escritorArchivo.Write(' ');
                escritorArchivo.Write(Maquina.GetArray()[i].Rendimiento.ToString());
            }
            if(Maquina.GetArray().Length != 0)
                escritorArchivo.Write(escritorArchivo.NewLine);


            escritorArchivo.WriteLine();
            if (Operario.GetArray().Length != 0)
            {
                escritorArchivo.WriteLine("Operario");
                escritorArchivo.WriteLine(Operario.GetArray().Length);
                EscribeNombres(Operario.GetArray(), escritorArchivo);
```

```
            EscribeDescripciones(Operario.GetArray(), escritorArchivo);
            escritorArchivo.WriteLine();
        }


        escritorArchivo.WriteLine("Trabajo");
        escritorArchivo.WriteLine("RutaFabricacion");
        foreach (Trabajo elemento in Trabajo.GetArray())
        {
            for (var i = 0; i < elemento.RutaFabricacion.Length; i++)
            {
                if (i != 0) escritorArchivo.Write(' ');
                escritorArchivo.Write(elemento.RutaFabricacion[i]);
            }
            escritorArchivo.Write(escritorArchivo.NewLine);
        }
        escritorArchivo.WriteLine("FechaInicio");
        foreach (Trabajo elemento in Trabajo.GetArray())
        {
            for (var i = 0; i < Maquina.GetArray().Length; i++)
            {
                if (i != 0) escritorArchivo.Write(' ');
                string textoEscrito = "-1";
                if (elemento.GetNumeroOperacionesAsignadas() != 0)
                {
                    for (var j = 0; j <
                        elemento.GetNumeroOperacionesAsignadas(); j++)
                    {
                        if (elemento.GetIDMaquinaOperacion(j) == i)
                        {
                            DateTime fechaInicio =
                                elemento.GetInicioOperacion(j);
                            TimeSpan tiempoDesdeOrigen =
                                fechaInicio.Subtract(
                                Trabajo.StaticOrigenProgramacion);
                            textoEscrito =
                                $"{(int)Math.Round(tiempoDesdeOrigen.TotalMinutes)}";
                        }
                    }
                }
                escritorArchivo.Write(textoEscrito);
            }
            escritorArchivo.Write(escritorArchivo.NewLine);
        }
        EscribeNombres(Trabajo.GetArray(), escritorArchivo);
        EscribeDescripciones(Trabajo.GetArray(), escritorArchivo);
        if (Operario.GetArray().Length != 0)
        {
            escritorArchivo.WriteLine("OperarioAsignado");
            escritorArchivo.WriteLine("ID");
            foreach (Trabajo elemento in Trabajo.GetArray())
            {
                for (var i = 0; i < elemento.GetOperarios().Length; i++)
                {
                    if (i != 0) escritorArchivo.Write(' ');
```

```
                    escritorArchivo.Write(
                        elemento.GetOperarios()[i].Identificador);
                }
                escritorArchivo.Write(escritorArchivo.NewLine);
            }
        }

        escritorArchivo.Close();
    }

    static void EscribeNombres<T>(T[] arrayObjetos, StreamWriter
        escritorArchivo) where T : IDatosIdentificativos
    {
        escritorArchivo.WriteLine("Nombre");
        for (var i = 0; i < arrayObjetos.Length; i++)
        {
            if (i != 0) escritorArchivo.Write('&');
            escritorArchivo.Write(arrayObjetos[i].Nombre);
        }
        if (arrayObjetos.Length != 0)
            escritorArchivo.Write(escritorArchivo.NewLine);
    }

    static void EscribeDescripciones<T>(T[] arrayObjetos, StreamWriter
        escritorArchivo) where T : IDatosIdentificativos
    {
        escritorArchivo.WriteLine("Descripcion");
        for (var i = 0; i < arrayObjetos.Length; i++)
        {
            if (i != 0) escritorArchivo.Write(escritorArchivo.NewLine); ;
            escritorArchivo.Write(arrayObjetos[i].Descripcion);
        }
        if (arrayObjetos.Length != 0)
            escritorArchivo.Write(escritorArchivo.NewLine);
    }
  }
}
```

Código A.11 Código empleado para elaborar el fichero de salida.

A.7 Clase abstracta ControlGantt

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Runtime.Remoting.Messaging;
using System.Security.Policy;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace GanttDiagramGenerator
{
    abstract public class ControlGantt : Control
    {
        public ControlGantt() : base()
        {

        }

        protected List<ControlGantt> controlesEnlazados = new
            List<ControlGantt>();

        public static void Link(ControlGantt[] listaControles)
        {
            UnLink(listaControles);

            foreach (ControlGantt elemento in listaControles)
            {
                elemento.controlesEnlazados.AddRange(listaControles);
                elemento.controlesEnlazados.Remove(elemento);
            }
        }

        public static void UnLink(ControlGantt[] listaControles)
        {
            foreach (ControlGantt elemento in listaControles)
            {
                elemento.controlesEnlazados.ForEach(c =>
                    c.controlesEnlazados.Remove(elemento));
                elemento.controlesEnlazados.Clear();
            }
        }

        protected void RefreshLinked()
        {
            DesactualizaBitmapDiagramasEnlazados();
            controlesEnlazados.ForEach(c => c.Refresh());
        }
        protected void InvalidateLinked()
        {
            DesactualizaBitmapDiagramasEnlazados();
            controlesEnlazados.ForEach(c => c.Invalidate(true));
        }
```

```
        protected void UpdateLinked()
        {
            DesactualizaBitmapDiagramasEnlazados();
            controlesEnlazados.ForEach(c => c.Update());
        }

        protected void DesactualizaBitmapDiagramasEnlazados()
        {
            foreach (ControlGantt elemento in controlesEnlazados)
            {
                if (elemento is DiagramaGantt)
                {
                    DiagramaGantt diagramaEnlazado = elemento as DiagramaGantt;
                    diagramaEnlazado.DesactualizarBitmap();
                }
            }
        }

        protected void ActualizaListBoxEnlazadas()
        {
            foreach (ControlGantt elemento in controlesEnlazados)
            {
                if (elemento is ListBoxTrabajos)
                {
                    ListBoxTrabajos listBoxEnlazada = elemento as
                        ListBoxTrabajos;
                    listBoxEnlazada.ContentUpdate();
                }
                else if (elemento is ListBoxMaquinas)
                {
                    ListBoxMaquinas listBoxEnlazada = elemento as
                        ListBoxMaquinas;
                    listBoxEnlazada.ContentUpdate();
                }
                else if (elemento is ListBoxOperarios)
                {
                    ListBoxOperarios listBoxEnlazada = elemento as
                        ListBoxOperarios;
                    listBoxEnlazada.ContentUpdate();
                }
            }
        }
    }
}
```

Código A.12 Código empleado para elaborar la clase abstracta ControlGantt.

A.7.1 Control DiagramaGantt

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Runtime.Remoting.Messaging;
```

```
using System.Security.Policy;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace GanttDiagramGenerator
{
    public partial class DiagramaGantt : ControlGantt
    {
        public DiagramaGantt() : base()
        {
            Inicializar();
        }

        public void Inicializar()
        {
            InicializarElementosGraficos();
            InicializarPictureBoxes();
            InicializarPaneles();
            AjustarGeometriaPaneles();

            Controls.Add(ejeMaquina);
            Controls.Add(ejeDias);
            Controls.Add(esquinaSuperior);
            Controls.Add(diagrama);
        }

        private void InicializarPictureBoxes()
        {
            dibujoEsquinaSuperior.Dock = DockStyle.Fill;
            dibujoEsquinaSuperior.MouseDown += new
                MouseEventHandler(ActualizaDiagrama_MouseMove);
            dibujoEsquinaSuperior.Paint += new
                PaintEventHandler(EsquinaSuperior_Paint);
            dibujoEsquinaSuperior.MouseDown += new
                MouseEventHandler(DragScroll_MouseDown);
            dibujoEsquinaSuperior.MouseMove += new
                MouseEventHandler(DragScroll_MouseMove);


            dibujoEjeDias.Dock = DockStyle.Fill;
            dibujoEjeDias.Size = ejeDias.Size;
            dibujoEjeDias.MouseMove += new
                MouseEventHandler(ActualizaDiagrama_MouseMove);
            dibujoEjeDias.Paint += new PaintEventHandler(EjeDias_Paint);
            dibujoEjeDias.MouseDown += new
                MouseEventHandler(DragScroll_MouseDown);
            dibujoEjeDias.MouseMove += new
                MouseEventHandler(DragScroll_MouseMove);

            dibujoDiagrama.Dock = DockStyle.Fill;
            dibujoDiagrama.Size = diagrama.Size;
            dibujoDiagrama.MouseMove += new
                MouseEventHandler(ActualizaDiagrama_MouseMove);
            dibujoDiagrama.MouseMove += new
                MouseEventHandler(Diagrama_MouseMove);
```

```
            dibujoDiagrama.Paint += new PaintEventHandler(Diagrama_Paint);
            dibujoDiagrama.MouseDown += new
                MouseEventHandler(DragScroll_MouseDown);
            dibujoDiagrama.AllowDrop = true;
            dibujoDiagrama.DragEnter += new
                DragEventHandler(DragTrabajo_DragEnter);
            dibujoDiagrama.DragOver += new
                DragEventHandler(DragTrabajo_DragOver);
            dibujoDiagrama.DragDrop += new
                DragEventHandler(DragTrabajo_DragDrop);
            dibujoDiagrama.Click += new EventHandler(Diagrama_Click);

            panelTrabajo.Visible = false;
            dibujoDiagrama.Controls.Add(panelTrabajo);

            dibujoEjeMaquina.Dock = DockStyle.Fill;
            dibujoEjeMaquina.Size = ejeMaquina.Size;
            dibujoEjeMaquina.MouseMove += new
                MouseEventHandler(ActualizaDiagrama_MouseMove);
            dibujoEjeMaquina.Paint += new PaintEventHandler(EjeMaquina_Paint);
            dibujoEjeMaquina.MouseDown += new
                MouseEventHandler(DragScroll_MouseDown);
            dibujoEjeMaquina.MouseMove += new
                MouseEventHandler(DragScroll_MouseMove);
            dibujoEjeMaquina.Click += new EventHandler(EjeMaquina_Click);
        }

        private void InicializarPaneles()
        {
            esquinaSuperior.Visible = true;
            esquinaSuperior.Controls.Add(dibujoEsquinaSuperior);

            ejeDias.Visible = true;
            ejeDias.Controls.Add(dibujoEjeDias);

            diagrama.Visible = true;
            diagrama.Controls.Add(dibujoDiagrama);
            diagrama.AutoScroll = true;
            diagrama.AutoScrollMargin = margenAutoscroll;
            diagrama.Scroll += new ScrollEventHandler(Diagrama_Scroll);
            diagrama.MouseWheel += new MouseEventHandler(Diagrama_OnMouseWheel);

            ejeMaquina.Visible = true;
            ejeMaquina.Controls.Add(dibujoEjeMaquina);

            panelTrabajo.BorderStyle = BorderStyle.FixedSingle;
        }

        public void AjustarGeometriaPaneles()
        {
            esquinaSuperior.Location = new Point(0, 0);
            esquinaSuperior.Size = new Size(AnchoEjeMaquinas, AltoEjeDias);

            diagrama.Location = new Point(AnchoEjeMaquinas, AltoEjeDias);
            diagrama.Size = new Size(Width - AnchoEjeMaquinas - Margin.Left,
                Height - AltoEjeDias - Margin.Bottom);
```

```
            diagrama.AutoScrollMinSize = new Size(AnchoDiagrama +
                ExcesoDerechoLineasHorizontales, AltoDiagrama +
                ExcesoInferiorLineasVerticales);

            ejeDias.Location = new Point(AnchoEjeMaquinas, 0);
            ejeDias.Size = new Size(Width + AnchoDiagrama + AnchoEjeMaquinas +
                ExcesoDerechoLineasHorizontales + margenAutoscroll.Width,
                AltoEjeDias);

            ejeMaquina.Location = new Point(0, AltoEjeDias);
            ejeMaquina.Size = new Size(AnchoEjeMaquinas, Height + AltoDiagrama
                + ExcesoInferiorLineasVerticales + margenAutoscroll.Height);

            AjustarPosicionesScroll();
        }

        public void AjustarPosicionesScroll()
        {
            ejeMaquina.SetBounds(0, AltoEjeDias -
                diagrama.VerticalScroll.Value, ejeMaquina.Width,
                ejeMaquina.Height);
            ejeDias.SetBounds(AnchoEjeMaquinas -
                diagrama.HorizontalScroll.Value, 0, ejeDias.Width,
                ejeDias.Height);
            dibujoDiagrama.BringToFront();
            esquinaSuperior.BringToFront();
        }

        public void SetBitmapDiagrama()
        {
            bmpDiagrama = new Bitmap(AnchoDiagrama +
                ExcesoDerechoLineasHorizontales, AltoDiagrama +
                ExcesoInferiorLineasVerticales);
            Graphics lienzo = Graphics.FromImage(bmpDiagrama);
            DibujarDiagrama(lienzo);
            lienzo.Dispose();

            bitmapActualizado = true;
        }
        public void DesactualizarBitmap() => bitmapActualizado = false;

        public Rectangle GetRectanguloDiagrama(DateTime fechaInicio, TimeSpan
            tiempoProceso, int identificadorMaquina)
        {
            TimeSpan tiempoDesdeOrigen =
                fechaInicio.Subtract(FechaInicioProgramacion);
            TimeSpan tiempoDesdeApertura = tiempoDesdeOrigen -
                TimeSpan.FromDays(tiempoDesdeOrigen.Days) -
                TimeSpan.FromHours(HoraInicioDia);
            int xInicio = (int)(tiempoDesdeOrigen.Days * (SeparacionDia +
                AnchoDias) + tiempoDesdeApertura.TotalHours * AnchoHora);
            int ancho = (int)(tiempoProceso.TotalHours * AnchoHora);
            int yInicio = identificadorMaquina * (SeparacionMaquina +
                AltoMaquina);
            int alto = AltoMaquina;
            return new Rectangle(xInicio, yInicio, ancho, alto);
```

```
        }

        public bool TrabajoContienePosicion(Trabajo trabajoComprobado, Point
            posicionLocal)
        {
            for (var i = 0; i <
                trabajoComprobado.GetNumeroOperacionesAsignadas(); i++)
            {
                Rectangle rectanguloDiagrama =
                    GetRectanguloDiagrama(trabajoComprobado.GetInicioOperacion(i),
                    trabajoComprobado.GetTiempoProcesoOperacion(i),
                        trabajoComprobado.GetIDMaquinaOperacion(i));

                if (rectanguloDiagrama.Contains(posicionLocal)) return true;
            }
            return false;
        }

        private void SetRectanguloSombra()
        {
            dibujoDiagrama.Invalidate(rectanguloSombra);
            rectanguloSombra = CalculaRectanguloSombra();
            dibujoDiagrama.Invalidate(rectanguloSombra);
        }

        private Rectangle CalculaRectanguloSombra()
        {
            Point posicionRectangulo = new Point(panelTrabajo.Location.X,
                panelTrabajo.Location.Y +
                (int)Math.Round(panelTrabajo.Height/2.0));
            Maquina maquinaOperacion = GetMaquinaPosicion(posicionRectangulo.Y);
            if (maquinaOperacion != null)
            {
                TimeSpan tiempoProceso =
                    GetTiempoProcesoPosicion(posicionRectangulo,
                    Trabajo.GetTrabajoSeleccionado());
                int diferenciaMinutos = 1;
                int i = 1;
                DateTime fechaInicio =
                    GetFechaInicioPosicion(posicionRectangulo);
                do
                {
                    if (Trabajo.GetTrabajoSeleccionado(
                        ).CheckOperacionValidaTrabajo(fechaInicio,
                        tiempoProceso, maquinaOperacion.Identificador))
                    {
                        fechaInicioRectanguloSombra = fechaInicio;
                        identificadorMaquinaRectanguloSombra =
                            maquinaOperacion.Identificador;
                        Rectangle rectanguloSombra =
                            GetRectanguloDiagrama(fechaInicio, tiempoProceso,
                            maquinaOperacion.Identificador);

                        if (rectanguloSombra.X > AnchoDiagrama) return
                            Rectangle.Empty;
                        else return rectanguloSombra;
```

```
                }
                fechaInicio = fechaInicio.AddMinutes(diferenciaMinutos);
                if (fechaInicio.Hour == HoraFinDia)
                {
                    fechaInicio = new DateTime(fechaInicio.Year,
                        fechaInicio.Month, fechaInicio.Day +
                        1,HoraInicioDia, 0, 0);
                }
                diferenciaMinutos = -1 * Math.Sign(diferenciaMinutos) *
                    (Math.Abs(diferenciaMinutos) + 1);

            } while (diferenciaMinutos <= 60);
        }
        return Rectangle.Empty;
    }

    private DateTime GetFechaInicioPosicion(Point posicionDiagrama)
    {
        int diaDiagrama = posicionDiagrama.X / (AnchoDias + SeparacionDia);
        double horaDiagrama = (double)(posicionDiagrama.X - diaDiagrama *
            (AnchoDias + SeparacionDia)) / AnchoHora + HoraInicioDia;
        DateTime fechaInicio = FechaInicioProgramacion.AddDays(diaDiagrama);
        fechaInicio = fechaInicio.AddHours(horaDiagrama);
        DateTime fechaInicioTruncada = new DateTime(fechaInicio.Year,
            fechaInicio.Month, fechaInicio.Day, fechaInicio.Hour,
            fechaInicio.Minute,0);
        return fechaInicioTruncada;
    }

    private TimeSpan GetTiempoProcesoPosicion(Point posicionDiagrama,
        Trabajo trabajoSeleccionado)
    {
        Maquina maquinaOperacion = GetMaquinaPosicion(posicionDiagrama.Y);
        if (maquinaOperacion != null)
        {
            return maquinaOperacion.GetTiempoProceso(
                trabajoSeleccionado.Identificador);
        }
        else
        {
            return TimeSpan.FromHours(-1);
        }
    }

    private Maquina GetMaquinaPosicion(int alturaDiagrama)
    {
        for (var i = 0; i < NumeroMaquinas; i++)
        {
            if (alturaDiagrama > (i * (separacionMaquina + AltoMaquina)) &&
              alturaDiagrama < (i * (separacionMaquina + AltoMaquina)) +
                  AltoMaquina)
            {
                return Maquina.Get(i);
            }
        }
        return null;
```

```
        }

        private Rectangle CreaRectanguloCentradoDiagrama(int margenInterno)
        {
            int xRectangulo =
                diagrama.PointToScreen(diagrama.ClientRectangle.Location).X +
                margenInterno;
            int yRectangulo =
                diagrama.PointToScreen(diagrama.ClientRectangle.Location).Y +
                margenInterno;
            int widthRectangulo = diagrama.ClientRectangle.Width - 2 *
                margenInterno;
            int heightRectangulo = diagrama.ClientRectangle.Height - 2 *
                margenInterno;
            return new Rectangle(xRectangulo, yRectangulo, widthRectangulo,
                heightRectangulo);
        }

        private void ScrollDiagrama (int margenScroll, int valorScroll)
        {
            int xCursor = diagrama.PointToClient(Cursor.Position).X;
            int yCursor = diagrama.PointToClient(Cursor.Position).Y;
            int xAdd = 0;
            int yAdd = 0;

            if (xCursor <= margenScroll)
            {
                xAdd = -valorScroll;
            }
            else if (xCursor >= diagrama.ClientRectangle.Location.X +
                diagrama.ClientRectangle.Width - margenScroll)
            {
                xAdd = valorScroll;
            }

            if (yCursor <= margenScroll)
            {
                yAdd = -valorScroll;
            }
            else if (yCursor >= diagrama.ClientRectangle.Location.Y +
                diagrama.ClientRectangle.Height - margenScroll)
            {
                yAdd = valorScroll;
            }
            diagrama.AutoScrollPosition = new
                Point(diagrama.HorizontalScroll.Value + xAdd,
                diagrama.VerticalScroll.Value + yAdd);
            AjustarPosicionesScroll();
        }

        private double[] DeterminaPosicionLineasTerciarias()
        {
            List<double> listaHoras = new List<double>();
            if (AnchoHora >= limiteCuartoHora)
            {
```

```
            for (var i = 0; i <= HorasXDia * 4; i++) listaHoras.Add(i *
                anchoHora / 4.0);
        }
        else if (AnchoHora >= limiteMediaHora)
        {
            for (var i = 0; i <= HorasXDia * 2; i++) listaHoras.Add(i *
                anchoHora / 2.0);
        }
        else if (AnchoHora >= limiteHora)
        {
            for (var i = 0; i <= HorasXDia; i++) listaHoras.Add(i *
                anchoHora);
        }
        return listaHoras.ToArray();
    }

    private double[] DeterminaPosicionLineasSecundarias()
    {
        List<double> listaHoras = new List<double>();

        if (AnchoHora >= limiteCuartoHora)
        {
            for (var i = 1; i < HorasXDia * 2; i++) listaHoras.Add(i *
                anchoHora / 2.0);
        }
        else if (AnchoHora >= limiteHora)
        {
            for (var i = 1; i < HorasXDia; i++) listaHoras.Add(i *
                anchoHora);
        }

        return listaHoras.ToArray();
    }



    private void MostrarPanelTrabajo(Rectangle rectanguloPanel, Color
        colorPanel)
    {
        panelTrabajo.BackColor = colorPanel;
        panelTrabajo.Size = rectanguloPanel.Size;
        panelTrabajo.Location = rectanguloPanel.Location;
        panelTrabajo.Visible = true;
        diferenciaPosicion = panelTrabajo.PointToClient(Cursor.Position);
    }

    private void MostrarPanelTrabajo(Color colorPanel)
    {
        Point posicionCursor = diagrama.PointToClient(Cursor.Position);
        panelTrabajo.BackColor = colorPanel;
        panelTrabajo.Size = new Size (AnchoHora,AltoMaquina);

        Maquina maquinaAsignada;
        int i = 0;

        do
```

```
            {
                maquinaAsignada = GetMaquinaPosicion(panelTrabajo.Location.Y +
                    (int)Math.Round(panelTrabajo.Height / 2.0) - i);
                i++;
            } while (maquinaAsignada == null);

            TimeSpan tiempoProceso = maquinaAsignada.GetTiempoProceso(
                Trabajo.GetTrabajoSeleccionado().Identificador);
            panelTrabajo.Width = (int)Math.Round(tiempoProceso.TotalHours *
                AnchoHora);

            panelTrabajo.Visible = true;
            diferenciaPosicion = new Point(panelTrabajo.Width / 2,
                panelTrabajo.Height / 2);
        }

        private int NumeroMaquinas { get => Maquina.GetArray().Length; }

        private int NumeroDias { get => Maquina.StaticNumeroDiasProgramacion; }

        public DateTime FechaInicioProgramacion { get =>
            Trabajo.StaticOrigenProgramacion; }

        public DateTime FechaFinProgramacion { get =>
            FechaInicioProgramacion.AddDays(NumeroDias); }

        public int AltoDiagrama { get => AltoMaquina * NumeroMaquinas +
            SeparacionMaquina * (NumeroMaquinas - 1); }
        public int AnchoDiagrama { get => AnchoDias * NumeroDias +
            SeparacionDia * (NumeroDias - 1); }

        public int AltoEjeDias { get => altoEjeDias; set => altoEjeDias =
            value; }
        private int altoEjeDias = 60;

        public int AnchoEjeMaquinas { get => anchoEjeMaquinas; set =>
            anchoEjeMaquinas = value; }
        private int anchoEjeMaquinas = 80;

        public int SeparacionDia { get => separacionDia; set => separacionDia
            = value; }
        private int separacionDia = 20;

        public int SeparacionMaquina { get => separacionMaquina; set =>
            separacionMaquina = value; }
        private int separacionMaquina = 10;

        public int AnchoHora
        {
            get
            {
                return anchoHora;
            }
            set
            {
                bitmapActualizado = false;
```

```
            if (value % 4 != 0) anchoHora = value / 4 * 4;
            else anchoHora = value;

            AjustarGeometriaPaneles();
            Invalidate(true);
        }
    }

    private Panel esquinaSuperior = new Panel();

    private Panel diagrama = new Panel();

    private Panel ejeMaquina = new Panel();

    private Panel ejeDias = new Panel();

    private Panel panelTrabajo = new Panel();

    private Point diferenciaPosicion;

    private PictureBox dibujoEjeDias = new PictureBox();

    private PictureBox dibujoEjeMaquina = new PictureBox();

    private PictureBox dibujoDiagrama = new PictureBox();

    private PictureBox dibujoEsquinaSuperior = new PictureBox();

    private Bitmap bmpDiagrama;


    private int anchoHora = 40;

    private int limiteHora = 10;

    private int limiteMediaHora = 30;

    private int limiteCuartoHora = 60;

    public int AnchoDias { get => anchoHora * HorasXDia; }

    public int AltoMaquina { get => altoMaquina; set => altoMaquina =
        value; }
    private int altoMaquina = 80;


    public int ExcesoSuperiorLineasVerticales { get =>
        excesoSuperiorLineasVerticales; set =>
        excesoSuperiorLineasVerticales = value; }
    private int excesoSuperiorLineasVerticales = 50;

    public int ExcesoInferiorLineasVerticales { get =>
        excesoInferiorLineasVerticales; set =>
        excesoInferiorLineasVerticales = value; }
```

```
        private int excesoInferiorLineasVerticales = 10;

        public int ExcesoIzquierdoLineasHorizontales { get =>
            excesoIzquierdoLineasHorizontales; set =>
            excesoIzquierdoLineasHorizontales = value; }
        private int excesoIzquierdoLineasHorizontales = 60;

        public int ExcesoDerechoLineasHorizontales { get =>
            excesoDerechoLineasHorizontales; set =>
            excesoDerechoLineasHorizontales = value; }
        private int excesoDerechoLineasHorizontales = 10;


        public int ExcesoLineasSecundarias { get => excesoLineasSecundarias;
            set => excesoLineasSecundarias = value; }
        private int excesoLineasSecundarias = 10;

        private Size margenAutoscroll = new Size(20, 20);

        public int HorasXDia { get => HoraFinDia - HoraInicioDia; }

        public int HoraInicioDia { get =>
            (int)Math.Floor(Maquina.GetAperturaMasTemprana()); }

        public int HoraFinDia { get =>
            (int)Math.Ceiling(Maquina.GetCierreMasTardio()); }

        public int SeparacionHoras = 20;

        private bool dragDropCheck = false;

        private bool resetSombraCheck = false;

        private bool bitmapActualizado = false;

        private Point posicionInicialScroll;

        private Point posicionInicialCursor;

        private Rectangle rectanguloSombra;

        private DateTime fechaInicioRectanguloSombra;

        private int identificadorMaquinaRectanguloSombra;

        public override Color BackColor
        {
            get => base.BackColor;

            set
            {
                base.BackColor = value;
                ejeMaquina.BackColor = value;
                esquinaSuperior.BackColor = value;
                ejeDias.BackColor = value;
                diagrama.BackColor = value;
```

```
            }
        }
    }
}
```

Código A.13 Código empleado para elaborar la parte principal del control heredado DiagramaGantt.

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace GanttDiagramGenerator
{
    public partial class DiagramaGantt : ControlGantt
    {
        private void ActualizaDiagrama_MouseMove(object sender, MouseEventArgs
            e)
        {
            if (!bitmapActualizado) SetBitmapDiagrama();
        }

        public void Diagrama_Paint(object sender, PaintEventArgs e)
        {
            if (dragDropCheck) e.Graphics.FillRectangle(new
                SolidBrush(colorSombra), rectanguloSombra);

            if (bitmapActualizado) e.Graphics.DrawImageUnscaled(bmpDiagrama, 0,
                0);
            else DibujarDiagrama(e.Graphics);
        }

        public void EjeDias_Paint(object sender, PaintEventArgs e)
        {
            int InicioLineasPrincipales = AltoEjeDias -
                ExcesoSuperiorLineasVerticales;

            e.Graphics.DrawLine(penBordes, new Point(0, AltoEjeDias), new
                Point(AnchoDiagrama + ExcesoDerechoLineasHorizontales,
                AltoEjeDias));
            e.Graphics.DrawLine(penLineasPrincipales, new Point(AnchoDiagrama ,
                InicioLineasPrincipales), new Point(AnchoDiagrama,
                AltoEjeDias));
            DibujaLineasParalelas(e.Graphics, new Point(AnchoDias,
                InicioLineasPrincipales), 0, NumeroDias - 1, AnchoHora *
                HorasXDia + SeparacionDia, AltoEjeDias, -Math.PI / 2,
                penLineasPrincipales);
            DibujaLineasParalelas(e.Graphics, new Point(AnchoDias +
                SeparacionDia, InicioLineasPrincipales), 0, NumeroDias - 1,
                AnchoHora * HorasXDia + SeparacionDia, AltoEjeDias, -Math.PI /
                2, penLineasPrincipales);
```

```
        string[] stringFechas = new string[NumeroDias];
        for (var i = 0; i < NumeroDias; i++) stringFechas[i] =
            FechaInicioProgramacion.AddDays(i).ToShortDateString();
        DibujaSerieStrings(e.Graphics, new Point(0, AltoEjeDias -
            ExcesoSuperiorLineasVerticales), 0, new Size(AnchoDias,
            AltoEjeDias), SeparacionDia, stringFechas, brEjesPrincipales,
            sfEjeDias, FuenteEjes);

        double[] posicionLineas = DeterminaPosicionLineasSecundarias();
        double separacionLineas;
        if (posicionLineas.Length > 2) separacionLineas = posicionLineas[1]
            - posicionLineas[0];
        else separacionLineas = 0;

        for (var i = 0; i < posicionLineas.Length; i++)
        {
            string textoHora = (TimeSpan.FromHours(HoraInicioDia) +
                TimeSpan.FromHours(posicionLineas[i] /
                AnchoHora)).ToString(@"hh\:mm");
            string[] arrayHoras = Enumerable.Repeat(textoHora,
                NumeroDias).ToArray();
            int anchoString = (int)Math.Ceiling(e.Graphics.MeasureString(
                TimeSpan.FromHours(0).ToString(@"hh\:mm"),
                FuenteTitulosHoras).Width);

            if (anchoString < separacionLineas * 0.85 || i % 2 == 1)
            {
                DibujaSerieStrings(e.Graphics, new
                    Point((int)Math.Round(posicionLineas[i]) - anchoString /
                    2, AltoEjeDias - excesoLineasSecundarias -
                    FuenteEjes.Height), 0, new Size(anchoString,
                    FuenteEjes.Height), SeparacionDia + AnchoDias -
                    anchoString, arrayHoras, brEjesPrincipales,
                    sfEjeMaquina, FuenteTitulosHoras);
            }

            DibujaLineasParalelas(e.Graphics, new
                Point((int)Math.Round(posicionLineas[i]), AltoEjeDias -
                excesoLineasSecundarias), 0, NumeroDias, AnchoHora *
                HorasXDia + SeparacionDia, excesoLineasSecundarias,
                -Math.PI / 2, penLineasSecundarias);
        }
    }

    public void EjeMaquina_Paint(object sender, PaintEventArgs e)
    {
        foreach (Maquina elemento in Maquina.GetArray())
        {
            if (elemento.Seleccionado)
            {
                int yRectangulo = elemento.Identificador * (AltoMaquina +
                    SeparacionMaquina);
                Rectangle rectangulo = new Rectangle(0, yRectangulo,
                    AnchoEjeMaquinas, AltoMaquina);
```

```
                e.Graphics.FillRectangle(new SolidBrush(Color.LightGray),
                    rectangulo);
            }
        }

        int InicioLineasPrincipales = AnchoEjeMaquinas -
            ExcesoIzquierdoLineasHorizontales;

        e.Graphics.DrawLine(penBordes, new Point(AnchoEjeMaquinas, 0), new
            Point(AnchoEjeMaquinas, AltoDiagrama +
            ExcesoInferiorLineasVerticales));
        e.Graphics.DrawLine(penLineasPrincipales, new
            Point(InicioLineasPrincipales, AltoDiagrama), new
            Point(AnchoEjeMaquinas, AltoDiagrama));
        DibujaLineasParalelas(e.Graphics, new
            Point(InicioLineasPrincipales, AltoMaquina), -Math.PI / 2,
            NumeroMaquinas - 1, AltoMaquina + SeparacionMaquina,
            AnchoEjeMaquinas, 0, penLineasPrincipales);
        DibujaLineasParalelas(e.Graphics, new
            Point(InicioLineasPrincipales, AltoMaquina +
            SeparacionMaquina), -Math.PI / 2, NumeroMaquinas - 1,
            AltoMaquina + SeparacionMaquina, AnchoEjeMaquinas, 0,
            penLineasPrincipales);

        string[] stringMaquinas = new string[NumeroMaquinas];
        for (var i = 0; i < NumeroMaquinas; i++) stringMaquinas[i] =
            $"Máquina {i}";
        DibujaSerieStrings(e.Graphics, new Point(0, 0), -Math.PI / 2, new
            Size(AnchoEjeMaquinas, AltoMaquina), SeparacionMaquina,
            stringMaquinas, brEjesPrincipales, sfEjeMaquina, FuenteEjes);
    }

    public void EsquinaSuperior_Paint(object sender, PaintEventArgs e)
    {
        e.Graphics.DrawLine(penBordes, new Point(AnchoEjeMaquinas,
            AltoEjeDias - ExcesoSuperiorLineasVerticales), new
            Point(AnchoEjeMaquinas, AltoEjeDias));
        e.Graphics.DrawLine(penBordes, new Point(AnchoEjeMaquinas -
            ExcesoIzquierdoLineasHorizontales, AltoEjeDias), new
            Point(AnchoEjeMaquinas, AltoEjeDias));
        if (dragDropCheck)
        {
            if (rectanguloSombra != Rectangle.Empty)
            {
                DateTime fechaInicio = fechaInicioRectanguloSombra;
                TimeSpan tiempoProceso = GetTiempoProcesoPosicion(new
                    Point(rectanguloSombra.Location.X,
                    rectanguloSombra.Location.Y + rectanguloSombra.Height /
                        2), Trabajo.GetTrabajoSeleccionado());

                TimeSpan horaFechaInicio = new TimeSpan(fechaInicio.Hour,
                    fechaInicio.Minute, 0);
                TimeSpan horaFechaFin = horaFechaInicio + tiempoProceso;
                string textoHoraInicio = "Inicio: "+
                    horaFechaInicio.ToString(@"hh\:mm");
```

```
                string textoHoraFin = "Fin: " +
                    horaFechaFin.ToString(@"hh\:mm");
                StringFormat sfDraw = new StringFormat();
                sfDraw.Alignment = StringAlignment.Center;
                sfDraw.LineAlignment = StringAlignment.Center;
                e.Graphics.DrawString(textoHoraInicio, FuenteEjes, new
                    SolidBrush(Color.Black), new Rectangle(0, AltoEjeDias /
                    2 - FuenteEjes.Height, AnchoEjeMaquinas,
                    FuenteEjes.Height), sfDraw);
                e.Graphics.DrawString(textoHoraFin, FuenteEjes, new
                    SolidBrush(Color.Black), new Rectangle(0, AltoEjeDias /
                    2 , AnchoEjeMaquinas, FuenteEjes.Height), sfDraw);
            }
        }
    }

    public void Diagrama_Scroll(object sender, ScrollEventArgs e)
    {
        if (e.ScrollOrientation == ScrollOrientation.HorizontalScroll)
        {
            ejeDias.SetBounds(AnchoEjeMaquinas -
                diagrama.HorizontalScroll.Value, 0, ejeDias.Width,
                ejeDias.Height);
        }

        if (e.ScrollOrientation == ScrollOrientation.VerticalScroll)
        {
            ejeMaquina.SetBounds(0, AltoEjeDias -
                diagrama.VerticalScroll.Value, ejeMaquina.Width,
                ejeMaquina.Height);
        }

        esquinaSuperior.BringToFront();
    }

    public void Diagrama_OnMouseWheel(object sender, MouseEventArgs e)
    {
        AjustarPosicionesScroll();
        esquinaSuperior.BringToFront();
    }

    private void DragScroll_MouseDown(object sender, MouseEventArgs e)
    {
        posicionInicialCursor = Cursor.Position;
        posicionInicialScroll = Cursor.Position;
    }

    private void DragScroll_MouseMove(object sender, MouseEventArgs e)
    {
        Size tamañoRectangulo = SystemInformation.DragSize;
        Point posicionRectangulo = Point.Subtract(posicionInicialCursor,
            new Size(tamañoRectangulo.Width / 2, tamañoRectangulo.Height /
                2));
        Rectangle rectanguloLimite = new Rectangle(posicionRectangulo,
            tamañoRectangulo);
```

```
            if (!rectanguloLimite.Contains(Cursor.Position) && e.Button ==
                MouseButtons.Left)
            {
                int sumaH = posicionInicialScroll.X - Cursor.Position.X;
                int sumaV = posicionInicialScroll.Y - Cursor.Position.Y;
                diagrama.AutoScrollPosition = new
                    Point(diagrama.HorizontalScroll.Value + sumaH,
                    diagrama.VerticalScroll.Value + sumaV);
                AjustarPosicionesScroll();
                posicionInicialScroll = Cursor.Position;
            }
        }

        private void Diagrama_Click(object sender, EventArgs e)
        {
            Size tamañoRectangulo = SystemInformation.DragSize;
            Point posicionRectangulo = Point.Subtract(posicionInicialCursor,
                new Size(tamañoRectangulo.Width / 2, tamañoRectangulo.Height /
                    2));
            Rectangle rectanguloLimite = new Rectangle(posicionRectangulo,
                tamañoRectangulo);

            if (rectanguloLimite.Contains(Cursor.Position))
            {
                int xDiagramaCursor = diagrama.PointToClient(Cursor.Position).X
                    + diagrama.HorizontalScroll.Value;
                int yDiagramaCursor = diagrama.PointToClient(Cursor.Position).Y
                    + diagrama.VerticalScroll.Value;
                Point posicionDiagramaCursor = new Point(xDiagramaCursor,
                    yDiagramaCursor);

                foreach (Trabajo elemento in Trabajo.GetArray())
                {
                    if (TrabajoContienePosicion(elemento,
                        posicionDiagramaCursor))
                    {
                        elemento.Seleccionado = true;

                        SetBitmapDiagrama();
                        dibujoDiagrama.Invalidate();
                        dibujoEjeMaquina.Invalidate();
                        InvalidateLinked();
                    }
                }
            }
        }

        private void EjeMaquina_Click(object sender, EventArgs e)
        {
            Size tamañoRectangulo = SystemInformation.DragSize;
            Point posicionRectangulo = Point.Subtract(posicionInicialCursor,
                new Size(tamañoRectangulo.Width / 2, tamañoRectangulo.Height /
                    2));
            Rectangle rectanguloLimite = new Rectangle(posicionRectangulo,
                tamañoRectangulo);
```

```
            if (rectanguloLimite.Contains(Cursor.Position))
            {
                int yDiagramaCursor = diagrama.PointToClient(Cursor.Position).Y
                    + diagrama.VerticalScroll.Value;
                Maquina maquinaSeleccionada =
                    GetMaquinaPosicion(yDiagramaCursor);
                if (maquinaSeleccionada != null)
                {
                    maquinaSeleccionada.Seleccionado = true;

                    SetBitmapDiagrama();
                    Invalidate(true);
                    RefreshLinked();
                }
            }
        }

        private void Diagrama_MouseMove(object sender, MouseEventArgs e)
        {
            Size tamañoRectangulo = SystemInformation.DragSize;
            Point posicionRectangulo = Point.Subtract(posicionInicialCursor,
                new Size(tamañoRectangulo.Width / 2, tamañoRectangulo.Height /
                    2));
            Rectangle rectanguloLimite = new Rectangle(posicionRectangulo,
                tamañoRectangulo);

            if (!rectanguloLimite.Contains(Cursor.Position) && e.Button ==
                MouseButtons.Left)
            {
                int xDiagramaCursor =
                    diagrama.PointToClient(posicionInicialCursor).X +
                    diagrama.HorizontalScroll.Value;
                int yDiagramaCursor =
                    diagrama.PointToClient(posicionInicialCursor).Y +
                    diagrama.VerticalScroll.Value;
                Point posicionDiagramaCursor = new Point(xDiagramaCursor,
                    yDiagramaCursor);
                Trabajo trabajoSeleccionado = Trabajo.GetTrabajoSeleccionado();

                if (trabajoSeleccionado != null &&
                    TrabajoContienePosicion(trabajoSeleccionado,
                    posicionDiagramaCursor))
                {
                    int operacionSeleccionada = -1;
                    Rectangle rectanguloOperacionSeleccionada = new Rectangle(0,
                        0, 0, 0);
                    for (var i = 0; i <
                        trabajoSeleccionado.GetNumeroOperacionesAsignadas(); i++)
                    {

                        Rectangle rectanguloDiagrama = GetRectanguloDiagrama(
                            trabajoSeleccionado.GetInicioOperacion(i),
                            trabajoSeleccionado.GetTiempoProcesoOperacion(i),
                                trabajoSeleccionado.GetIDMaquinaOperacion(i));

                        if (rectanguloDiagrama.Contains(posicionDiagramaCursor))
```

```
                    {
                        operacionSeleccionada = i;
                        rectanguloOperacionSeleccionada = rectanguloDiagrama;
                    }
                }

                MostrarPanelTrabajo(rectanguloOperacionSeleccionada,
                    Trabajo.GetTrabajoSeleccionado().ColorDibujo);

                panelTrabajo.Location = new
                    Point(dibujoDiagrama.PointToClient(Cursor.Position).X -
                    diferenciaPosicion.X,
                    dibujoDiagrama.PointToClient(Cursor.Position).Y -
                        diferenciaPosicion.Y);
                trabajoSeleccionado.BorraOperacion(operacionSeleccionada);
                RefreshLinked();
                dibujoDiagrama.DoDragDrop(diagrama, DragDropEffects.All);
            }
            else
            {
                posicionInicialCursor = new Point(-1, -1);
                if (!rectanguloLimite.Contains(Cursor.Position) && e.Button
                    == MouseButtons.Left)
                {
                    int sumaH = posicionInicialScroll.X - Cursor.Position.X;
                    int sumaV = posicionInicialScroll.Y - Cursor.Position.Y;
                    diagrama.AutoScrollPosition = new
                        Point(diagrama.HorizontalScroll.Value + sumaH,
                        diagrama.VerticalScroll.Value + sumaV);
                    AjustarPosicionesScroll();
                    posicionInicialScroll = Cursor.Position;
                }
            }
        }
    }

    private void DragTrabajo_DragEnter(object sender, DragEventArgs e)
    {
        SetBitmapDiagrama();
        if(!panelTrabajo.Visible)
            MostrarPanelTrabajo(Trabajo.GetTrabajoSeleccionado().ColorDibujo);
        e.Effect = DragDropEffects.All;
        Cursor.Clip = CreaRectanguloCentradoDiagrama(1);
        dragDropCheck = true;

        rectanguloSombra = Rectangle.Empty;
        diagrama.Invalidate();
        esquinaSuperior.Invalidate();
    }

    private void DragTrabajo_DragDrop(object sender, DragEventArgs e)
    {
        panelTrabajo.Location = new
            Point(dibujoDiagrama.PointToClient(Cursor.Position).X -
            diferenciaPosicion.X,
```

```
            dibujoDiagrama.PointToClient(Cursor.Position).Y -
                diferenciaPosicion.Y);
            dragDropCheck = false;
            panelTrabajo.Visible = false;

            Cursor.Clip = Rectangle.Empty;

            if (rectanguloSombra != Rectangle.Empty)
            {
                Trabajo.GetTrabajoSeleccionado().AddOperacion(fechaInicioRectanguloSombra,
                    identificadorMaquinaRectanguloSombra);
                ActualizaListBoxEnlazadas();
            }
            RefreshLinked();
            SetBitmapDiagrama();
            diagrama.Refresh();
            esquinaSuperior.Refresh();
        }

        private void DragTrabajo_DragOver(object sender, DragEventArgs e)
        {
            int margenRectangulo = 10;
            Rectangle rectanguloInterno =
                CreaRectanguloCentradoDiagrama(margenRectangulo);

            panelTrabajo.Location = new
                Point(dibujoDiagrama.PointToClient(Cursor.Position).X -
                diferenciaPosicion.X,
                dibujoDiagrama.PointToClient(Cursor.Position).Y -
                    diferenciaPosicion.Y);

            if (rectanguloInterno.Contains(Cursor.Position))
            {
                SetRectanguloSombra();

                if (rectanguloSombra != Rectangle.Empty)
                {
                    panelTrabajo.Width = rectanguloSombra.Width;
                }
                else
                {
                    Maquina maquinaAsignada =
                        GetMaquinaPosicion(panelTrabajo.Location.Y +
                        (int)Math.Round(panelTrabajo.Height / 2.0));
                    if (maquinaAsignada != null)
                    {
                        TimeSpan tiempoProceso =
                            maquinaAsignada.GetTiempoProceso(Trabajo.GetTrabajoSeleccionado().Identifi
                        panelTrabajo.Width =
                            (int)Math.Round(tiempoProceso.TotalHours *
                            AnchoHora);
                    }
                }

                esquinaSuperior.Refresh();
                resetSombraCheck = false;
```

```
            }
            else
            {
                if(dibujoDiagrama.PointToClient(Cursor.Position).X <=
                    margenRectangulo * 2)
                {
                    SetRectanguloSombra();
                }
                else if (!resetSombraCheck)
                {
                    dibujoDiagrama.Invalidate(rectanguloSombra);
                    rectanguloSombra = Rectangle.Empty;
                    esquinaSuperior.Refresh();
                    resetSombraCheck = true;
                }

                if (posicionInicialCursor == Cursor.Position)
                    ScrollDiagrama(margenRectangulo, AnchoHora / 2);
                posicionInicialCursor = Cursor.Position;
            }
            dibujoDiagrama.Update();
        }
    }
}
```

Código A.14 Código empleado para elaborar los EventHandlers del control heredado DiagramaGantt.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;
using System.Runtime.CompilerServices;

namespace GanttDiagramGenerator
{
    public partial class DiagramaGantt : ControlGantt
    {
        public static void DibujaLineasParalelas(Graphics lienzo, Point
            origenEje, double anguloEje, int cantidadLineas, int
            separacionLineas, int longitudLineas, double anguloLineas, Pen
            drawPen)
        {
            for (var i = 0; i < cantidadLineas; i++)
            {
                Point inicioLinea = new Point((int)(origenEje.X + i *
                    separacionLineas * Math.Cos(anguloEje)),
                    (int)(origenEje.Y - i * separacionLineas *
                        Math.Sin(anguloEje)));
                Point finLinea = new Point((int)(inicioLinea.X + longitudLineas
                    * Math.Cos(anguloLineas)),
```

```
                (int)(inicioLinea.Y - longitudLineas *
                    Math.Sin(anguloLineas)));
            lienzo.DrawLine(drawPen, inicioLinea, finLinea);
        }
    }

    public static void DibujaSerieStrings(Graphics lienzo, Point
        origenEje, double anguloEje, Size tamañoRectangulo, int
        separacionRectangulos, string[] drawText, Brush drawBrush,
        StringFormat drawStringFormat, Font drawFont)
    {
        for (var i = 0; i < drawText.Length; i++)
        {
            Point posicionRecangulo = new Point((int)(origenEje.X +
                (separacionRectangulos + tamañoRectangulo.Width) *
                Math.Cos(anguloEje) * i),
               (int)(origenEje.Y - (separacionRectangulos +
                  tamañoRectangulo.Height) * Math.Sin(anguloEje) * i));

            RectangleF drawRectangle = new Rectangle(posicionRecangulo,
                tamañoRectangulo);
            lienzo.DrawString(drawText[i], drawFont, drawBrush,
                drawRectangle, drawStringFormat);
        }
    }

    public static void DibujaRectanguloRayado(Graphics lienzo, Rectangle
        rectangulo, int espaciadoLineas, double anguloLineas, Pen
        borderPen, Pen innerPen, Color backColor)
    {
        lienzo.FillRectangle(new SolidBrush(backColor), rectangulo);
        lienzo.DrawRectangle(borderPen, rectangulo);
        Point inicioLinea = new Point(rectangulo.X, rectangulo.Y);
        Point finLinea = inicioLinea;
        int yAdd = (int)(espaciadoLineas / Math.Cos(anguloLineas));
        int xAdd = (int)(espaciadoLineas / Math.Sin(anguloLineas));
        do
        {
            if (inicioLinea.Y <= (int)(rectangulo.Y + rectangulo.Width *
                Math.Tan(anguloLineas)) && inicioLinea.X == rectangulo.X)
            {
                finLinea = new Point((int)(rectangulo.X + (inicioLinea.Y -
                    rectangulo.Y) / Math.Tan(anguloLineas)), rectangulo.Y);
            }
            else if (inicioLinea.Y > (int)(rectangulo.Y + rectangulo.Width
                * Math.Tan(anguloLineas)) && inicioLinea.X == rectangulo.X)
            {
                finLinea = new Point(rectangulo.X + rectangulo.Width,
                    (int)(inicioLinea.Y - rectangulo.Width *
                    Math.Tan(anguloLineas)));
            }
            else if (inicioLinea.Y == rectangulo.Y + rectangulo.Height &&
                inicioLinea.X <= rectangulo.X + rectangulo.Width -
                rectangulo.Height / Math.Tan(anguloLineas))
            {
```

```
                finLinea = new Point((int)(inicioLinea.X + rectangulo.Height
                    / Math.Tan(anguloLineas)), rectangulo.Y);
            }
            else if (inicioLinea.Y == rectangulo.Y + rectangulo.Height &&
                inicioLinea.X > rectangulo.X + rectangulo.Width -
                rectangulo.Height / Math.Tan(anguloLineas))
            {
                finLinea = new Point(rectangulo.X + rectangulo.Width,
                    (int)(inicioLinea.Y - (rectangulo.X + rectangulo.Width -
                    inicioLinea.X) * Math.Tan(anguloLineas)));
            }

            lienzo.DrawLine(innerPen, inicioLinea, finLinea);

            if (inicioLinea.Y + yAdd < rectangulo.Y + rectangulo.Height)
            {
                inicioLinea = new Point(inicioLinea.X, inicioLinea.Y + yAdd);
            }
            else if (inicioLinea.Y != (rectangulo.Y + rectangulo.Height))
            {
                inicioLinea = new Point((int)(inicioLinea.X + xAdd -
                    (rectangulo.Y + rectangulo.Height - inicioLinea.Y) /
                    Math.Tan(anguloLineas)),
                    rectangulo.Y + rectangulo.Height);
            }
            else
            {
                inicioLinea = new Point(inicioLinea.X + xAdd, rectangulo.Y +
                    rectangulo.Height);
            }

        } while (inicioLinea.X < (rectangulo.X + rectangulo.Width) ||
            inicioLinea.Y < (rectangulo.Y + rectangulo.Height));
    }

    private void DibujaOperacion(Graphics lienzo, Rectangle rectangulo,
        string textoInterior, Color colorFondo)
    {
        lienzo.FillRectangle(new SolidBrush(colorFondo), rectangulo);
        lienzo.DrawRectangle(penLineasPrincipales, rectangulo);
    }

    private void DibujaOperacionSeleccionada(Graphics lienzo, Rectangle
        rectangulo, string textoInterior, Color colorFondo)
    {
        lienzo.FillRectangle(new System.Drawing.Drawing2D.HatchBrush(
            System.Drawing.Drawing2D.HatchStyle.DottedDiamond,
            Color.Black, colorFondo), rectangulo);
        lienzo.DrawRectangle(penLineasPrincipales, rectangulo);
    }

    private void DibujaOperacionesTrabajo(Graphics lienzo, Trabajo
        trabajoDibujado)
    {
        for (var i = 0; i <
            trabajoDibujado.GetNumeroOperacionesAsignadas(); i++)
```

```
            {
                Rectangle rectanguloDibujo =
                    GetRectanguloDiagrama(trabajoDibujado.GetInicioOperacion(i),
                    trabajoDibujado.GetTiempoProcesoOperacion(i),
                        trabajoDibujado.GetIDMaquinaOperacion(i));
                if (trabajoDibujado.Resaltado)
                    DibujaOperacionSeleccionada(lienzo, rectanguloDibujo, " ",
                    trabajoDibujado.ColorDibujo);

                else DibujaOperacion(lienzo, rectanguloDibujo, " ",
                    trabajoDibujado.ColorDibujo);
            }
        }

        private void DibujaHorario(Graphics lienzo, TimeSpan[] arrayAperturas,
            TimeSpan[] arrayCierres, int dia, int numeroMaquina)
        {
            Color colorFondo = Color.Transparent;
            int xDiaHorario = (AnchoDias + SeparacionDia) * dia;
            int yHorario = numeroMaquina * (SeparacionMaquina + AltoMaquina);

            if (arrayAperturas.Length != 0 || arrayCierres.Length != 0)
            {
                int anchoHorario = (int)(AnchoHora *
                    (arrayAperturas[0].TotalHours - HoraInicioDia));
                int xHorario = xDiaHorario;

                Rectangle rectanguloSombreado = new Rectangle(xHorario,
                    yHorario, anchoHorario, altoMaquina);
                DibujaRectanguloRayado(lienzo, rectanguloSombreado, 10, Math.PI
                    / 4, penLineasPrincipales, penLineasPrincipales,
                    colorFondo);

                anchoHorario = AnchoDias - (int)(AnchoHora *
                    (arrayCierres[arrayCierres.Length - 1].TotalHours -
                    HoraInicioDia));
                xHorario = xDiaHorario + (int)(AnchoHora *
                    (arrayCierres[arrayCierres.Length - 1].TotalHours -
                    HoraInicioDia));

                rectanguloSombreado = new Rectangle(xHorario, yHorario,
                    anchoHorario, altoMaquina);
                DibujaRectanguloRayado(lienzo, rectanguloSombreado, 10, Math.PI
                    / 4, penLineasPrincipales, penLineasPrincipales,
                    colorFondo);

                for (var i = 1; i < arrayAperturas.Length; i++)
                {
                    anchoHorario = (int)(AnchoHora *
                        (arrayAperturas[i].TotalHours - arrayCierres[i -
                        1].TotalHours));
                    xHorario = xDiaHorario + (int)(AnchoHora * (arrayCierres[i -
                        1].TotalHours - HoraInicioDia));
                    rectanguloSombreado = new Rectangle(xHorario, yHorario,
                        anchoHorario, altoMaquina);
```

```
                DibujaRectanguloRayado(lienzo, rectanguloSombreado, 10,
                    Math.PI / 4, penLineasPrincipales, penLineasPrincipales,
                    colorFondo);
            }
        }
        else
        {
            Rectangle rectanguloCierre = new Rectangle(xDiaHorario,
                yHorario, AnchoDias, AltoMaquina);
            DibujaRectanguloRayado(lienzo, rectanguloCierre, 10, Math.PI /
                4, penLineasPrincipales, penLineasPrincipales, colorFondo);
        }
    }

    private void DibujaHorariosMaquina(Graphics lienzo, Maquina
        maquinaSeleccionada)
    {
        for (var i = 0; i < NumeroDias; i++)
        {
            DibujaHorario(lienzo, maquinaSeleccionada.GetAperturas(i),
                maquinaSeleccionada.GetCierres(i),
                i, maquinaSeleccionada.Identificador);
        }
    }

    public void DibujarDiagrama(Graphics lienzo)
    {
        foreach (double elemento in DeterminaPosicionLineasTerciarias())
        {
            for (var i = 0; i < Maquina.GetArray().Length; i++)
            {
                DibujaLineasParalelas(lienzo, new
                    Point((int)Math.Round(elemento), i * (AltoMaquina +
                    SeparacionMaquina)), 0, NumeroDias, AnchoHora *
                    HorasXDia + SeparacionDia, AltoMaquina, -Math.PI / 2,
                    penLineasTerciarias);
            }
        }

        foreach (Trabajo elemento in Trabajo.GetArray())
            DibujaOperacionesTrabajo(lienzo, elemento);

        int LongitudLineasHorizontalesPrincipales = AnchoDiagrama +
            ExcesoDerechoLineasHorizontales;
        int LongitudLineasVerticalesPrincipales = AltoDiagrama +
            ExcesoInferiorLineasVerticales;

        lienzo.DrawLine(penLineasPrincipales, new Point(0, 0), new Point(0,
            LongitudLineasVerticalesPrincipales));
        lienzo.DrawLine(penLineasPrincipales, new Point(AnchoDiagrama, 0),
            new Point(AnchoDiagrama, LongitudLineasVerticalesPrincipales));
        DibujaLineasParalelas(lienzo, new Point(AnchoDias, 0), 0,
            NumeroDias - 1, AnchoHora * HorasXDia + SeparacionDia,
            LongitudLineasVerticalesPrincipales, -Math.PI / 2,
            penLineasPrincipales);
```

```
            DibujaLineasParalelas(lienzo, new Point(AnchoDias + SeparacionDia,
                0), 0, NumeroDias - 1, AnchoHora * HorasXDia + SeparacionDia,
                LongitudLineasVerticalesPrincipales, -Math.PI / 2,
                penLineasPrincipales);
            lienzo.DrawLine(penLineasPrincipales, new Point(0, 0), new
                Point(LongitudLineasHorizontalesPrincipales, 0));
            lienzo.DrawLine(penLineasPrincipales, new Point(0, AltoDiagrama),
                new Point(LongitudLineasHorizontalesPrincipales, AltoDiagrama));
            DibujaLineasParalelas(lienzo, new Point(0, AltoMaquina), -Math.PI /
                2, NumeroMaquinas - 1, AltoMaquina + SeparacionMaquina,
                LongitudLineasHorizontalesPrincipales, 0, penLineasPrincipales);
            DibujaLineasParalelas(lienzo, new Point(0, AltoMaquina +
                SeparacionMaquina), -Math.PI / 2, NumeroMaquinas - 1,
                AltoMaquina + SeparacionMaquina,
                LongitudLineasHorizontalesPrincipales, 0, penLineasPrincipales);

            foreach (double elemento in DeterminaPosicionLineasSecundarias())
            {
                DibujaLineasParalelas(lienzo, new
                    Point((int)Math.Round(elemento), 0), 0, NumeroDias,
                    AnchoHora * HorasXDia + SeparacionDia, AltoDiagrama +
                    ExcesoLineasSecundarias, -Math.PI / 2,
                    penLineasSecundarias);
            }

            foreach (Maquina elemento in Maquina.GetArray())
                DibujaHorariosMaquina(lienzo, elemento);
        }

        public void InicializarElementosGraficos()
        {
            sfEjeMaquina.Alignment = StringAlignment.Center;
            sfEjeMaquina.LineAlignment = StringAlignment.Center;
            sfEjeDias.Alignment = StringAlignment.Center;
            penLineasSecundarias.DashStyle = DashStyle.Dash;
            penLineasTerciarias.DashStyle = DashStyle.Dot;
        }

        private SolidBrush brEjesPrincipales = new SolidBrush(Color.Black);

        private Color colorSombra = Color.LightGray;
        private Font FuenteEjes { get => new Font(Font.Name,
            tamañoFuenteEjes); }

        private float tamañoFuenteEjes = 10;

        private Font FuenteTitulosHoras { get => new Font(Font.Name,
            tamañoFuenteTitulosHoras); }

        private float tamañoFuenteTitulosHoras = 8;

        private StringFormat sfEjeMaquina = new StringFormat();

        private StringFormat sfEjeDias = new StringFormat();

        private Pen penLineasPrincipales = new Pen(Color.Black, 2);
```

```
        private Pen penBordes = new Pen(Color.Black, 6);

        private Pen penLineasSecundarias = new Pen(Color.Black, 1);

        private Pen penLineasTerciarias = new Pen(Color.Black, 1);
    }
}
```

Código A.15 Código empleado para elaborar los elementos gráficos del control heredado DiagramaGantt.

A.7.2 Clase abstracta ListBoxGantt y controles heredados

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GanttDiagramGenerator
{
    public abstract class ListBoxGantt<T> : ControlGantt where T :
        IDatosIdentificativos , ISeleccionable
    {
        public ListBoxGantt()
        {
            InicializarMainListBox();
        }

        protected void InicializarMainListBox()
        {
            mainListBox.Dock = DockStyle.Fill;
            mainListBox.HorizontalScrollbar = true;
            mainListBox.IntegralHeight = false;
            mainListBox.BackColor = Color.White;
            mainListBox.FormattingEnabled = true;
            mainListBox.ItemHeight = 16;
            mainListBox.BorderStyle = BorderStyle.Fixed3D;
            mainListBox.Font = Font;
            mainListBox.SelectedValueChanged += new
                EventHandler(Seleccion_SelectedValueChanged);
        }

        private void Seleccion_SelectedValueChanged(object sender, EventArgs e)
        {
            T elementoSeleccionado = (T)mainListBox.SelectedItem;
            if (elementoSeleccionado != null)
            {
                elementoSeleccionado.Seleccionado = true;
                InvalidateLinked();
            }
```

```
        }
        public abstract void ContentUpdate();

        public void Clear() => mainListBox.Items.Clear();

        protected ListBox mainListBox = new ListBox();
    }

    public class ListBoxTrabajos : ListBoxGantt<Trabajo>
    {
        public ListBoxTrabajos() :base()
        {
            Inicializar();
            Controls.Add(scFondo);
        }

        protected void Inicializar()
        {
            scFondo.IsSplitterFixed = true;
            scFondo.Orientation = Orientation.Horizontal;
            scFondo.Dock = DockStyle.Fill;
            scFondo.BackColor = SystemColors.Control;

            scFondo.Panel1.BackColor = BackColor;
            scFondo.Panel1.Controls.Add(mainListBox);

            scFondo.Panel2.BackColor = SystemColors.Control;
            scFondo.Panel2.Controls.Add(flpBotones);

            flpBotones.Dock = DockStyle.Fill;

            btnNoAsignados.Font = Font;
            btnNoAsignados.Text = "No Asignados";
            btnNoAsignados.CheckedChanged += new
                EventHandler(radioButton_CheckedChanged);
            btnNoAsignados.Checked = true;

            btnAsignados.Font = Font;
            btnAsignados.Text = "Asignados";
            btnAsignados.CheckedChanged += new
                EventHandler(radioButton_CheckedChanged);


            btnTrabajos.Font = Font;
            btnTrabajos.Text = "Ambos";
            btnTrabajos.CheckedChanged += new
                EventHandler(radioButton_CheckedChanged);


            flpBotones.Controls.Add(btnNoAsignados);
            flpBotones.Controls.Add(btnAsignados);
            flpBotones.Controls.Add(btnTrabajos);
            Graphics gfx_flpBotones = flpBotones.CreateGraphics();
            btnNoAsignados.Width =
                gfx_flpBotones.MeasureString(btnNoAsignados.Text,
                btnNoAsignados.Font).ToSize().Width + 25;
```

```
            btnAsignados.Width =
                gfx_flpBotones.MeasureString(btnAsignados.Text,
                btnAsignados.Font).ToSize().Width + 25;
            btnTrabajos.Width = gfx_flpBotones.MeasureString(btnTrabajos.Text,
                btnTrabajos.Font).ToSize().Width + 25;
            gfx_flpBotones.Dispose();
            SetSplitterPosition();

            mainListBox.MouseDown += new MouseEventHandler(MouseDown_ListBox);
            mainListBox.MouseMove += new MouseEventHandler(MouseMove_ListBox);
        }

        public override void ContentUpdate()
        {
            Clear();
            if (btnTrabajos.Checked)
            {
                mainListBox.Items.AddRange(Trabajo.GetArray());
            }
            else if (btnAsignados.Checked)
            {
                foreach (Trabajo elemento in Trabajo.GetArray())
                {
                    if (elemento.Programado) mainListBox.Items.Add(elemento);
                }
            }
            else if (btnNoAsignados.Checked)
            {
                foreach (Trabajo elemento in Trabajo.GetArray())
                {
                    if (!elemento.Programado) mainListBox.Items.Add(elemento);
                }
            }
        }

        public void SetSplitterPosition()
        {
            if (scFondo.Width > btnNoAsignados.Width + btnAsignados.Width +
                btnTrabajos.Width + 17)
            {
                scFondo.SplitterDistance = scFondo.Height - 30;
            }
            else if (scFondo.Width > btnAsignados.Width + btnTrabajos.Width +
                11)
            {
                scFondo.SplitterDistance = scFondo.Height - 60;
            }
            else
            {
                scFondo.SplitterDistance = scFondo.Height - 90;
            }
        }

        public override void Refresh()
        {
            ContentUpdate();
```

```
        base.Refresh();
    }

    private SplitContainer scFondo = new SplitContainer();
    private FlowLayoutPanel flpBotones = new FlowLayoutPanel();
    private RadioButton btnTrabajos = new RadioButton();
    private RadioButton btnAsignados = new RadioButton();
    private RadioButton btnNoAsignados = new RadioButton();

    bool mouseDownCheck = false;
    Point posicionInicialCursor;

    void radioButton_CheckedChanged(object sender, EventArgs e)
    {
        RadioButton rb = sender as RadioButton;
        if (rb.Checked) ContentUpdate();
    }

    void MouseDown_ListBox(object sender, MouseEventArgs e)
    {
        mouseDownCheck = true;
        posicionInicialCursor = Cursor.Position;
    }

    void MouseMove_ListBox(object sender, MouseEventArgs e)
    {
        if (mouseDownCheck == true)
        {
            Size tamañoRectangulo = SystemInformation.DragSize;
            Point posicionRectangulo = Point.Subtract(posicionInicialCursor,
                new Size(tamañoRectangulo.Width / 2, tamañoRectangulo.Height
                    / 2));
            Rectangle rectanguloLimite = new Rectangle(posicionRectangulo,
                tamañoRectangulo);

            if (mainListBox.SelectedItem != null &&
                !rectanguloLimite.Contains(Cursor.Position) && e.Button ==
                MouseButtons.Left)
            {
                Trabajo elementoSeleccionado = mainListBox.SelectedItem as
                    Trabajo;
                if (elementoSeleccionado.Seleccionado)
                {
                    if (elementoSeleccionado.Programado)
                    {
                        MessageBox.Show("El trabajo ya está programado.");
                    }
                    else
                    {
                        InvalidateLinked();
                        mainListBox.DoDragDrop(mainListBox,
                            DragDropEffects.All);
                    }
                }
            }
        }
```

```
        }
    }

    public class ListBoxMaquinas : ListBoxGantt<Maquina>
    {
        public ListBoxMaquinas() : base()
        {
            Controls.Add(mainListBox);
        }

        public override void ContentUpdate()
        {
            Clear();
            mainListBox.Items.AddRange(Maquina.GetArray());
        }
    }

    public class ListBoxOperarios : ListBoxGantt<Operario>
    {
        public ListBoxOperarios() : base()
        {
            Controls.Add(mainListBox);
        }

        public override void ContentUpdate()
        {
            Clear();
            mainListBox.Items.AddRange(Operario.GetArray());
        }
    }
}
```

Código A.16 Código empleado para elaborar la clase abstracta heredada ListBoxGantt y sus tres controles heredados, LsitBoxTrabajo, ListBoxMaquina y ListBoxOperario.

A.7.3 Clase abstracta LabelGantt y controles heredados

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Runtime.Remoting.Messaging;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace GanttDiagramGenerator
{
    abstract public class LabelGantt : ControlGantt
    {
        public LabelGantt() : base()
        {
            Inicializar();
        }
```

```
        private void Inicializar()
        {
            dibujoLabel.Dock = DockStyle.Fill;
            dibujoLabel.Paint += new PaintEventHandler(dibujoLabel_Paint);
            Controls.Add(dibujoLabel);
        }

        protected abstract int DibujaTextoInformativo(Graphics lienzo);

        private void dibujoLabel_Paint(object sender, PaintEventArgs e)
        {
            Height = DibujaTextoInformativo(e.Graphics) + 5;
        }

        protected float DrawStringLines(Graphics lienzo, PointF origen, int
            anchoLineas, string texto, Font drawFont, Brush drawBrush)
        {
            float yDibujo = origen.Y;
            if (lienzo.MeasureString(texto, drawFont).Width > anchoLineas)
            {
                string[] textoTroceado = texto.Split(' ');
                string linea = "";
                foreach (string elemento in textoTroceado)
                {
                    if (lienzo.MeasureString(linea + " " + elemento,
                        drawFont).Width > anchoLineas)
                    {
                        lienzo.DrawString(linea, drawFont, drawBrush, origen.X,
                            yDibujo);
                        linea = elemento;
                        yDibujo += drawFont.GetHeight(lienzo);
                    }
                    else
                    {
                        linea += " " + elemento;
                    }
                }
                lienzo.DrawString(linea, drawFont, drawBrush, origen.X,
                    yDibujo);
                yDibujo += drawFont.GetHeight(lienzo);
            }
            else
            {
                lienzo.DrawString(texto, drawFont, drawBrush, origen.X,
                    yDibujo);
                yDibujo += drawFont.GetHeight(lienzo);
            }
            return yDibujo;
        }

        protected Font fontTitulo
        {
            get
            {
                Font fuente = new Font(Font.Name, Font.SizeInPoints + 2);
                return fuente;
```

```
            }
        }

        public Point Margenes { set => margenes = value; get => margenes; }
        protected Point margenes = new Point(10, 10);

        protected int Ancho { get => Width - margenes.X * 2; }

        protected Brush brushForeColor { get => new SolidBrush(ForeColor); }

        protected PictureBox dibujoLabel = new PictureBox();
    }

    public class LabelGanttElementos : LabelGantt
    {
        public LabelGanttElementos() :base()
        {

        }

        protected override int DibujaTextoInformativo(Graphics lienzo)
        {
            foreach (Trabajo elemento in Trabajo.GetArray())
            {
                if (elemento.Seleccionado) return DibujaTextoElemento(lienzo,
                    elemento);
            }

            foreach (Maquina elemento in Maquina.GetArray())
            {
                if (elemento.Seleccionado) return DibujaTextoElemento(lienzo,
                    elemento);
            }

            foreach (Operario elemento in Operario.GetArray())
            {
                if (elemento.Seleccionado) return DibujaTextoElemento(lienzo,
                    elemento);
            }

            return 10;
        }

        private int DibujaTextoElemento(Graphics lienzo, Trabajo
            trabajoSeleccionado)
        {
            float yLinea = margenes.Y;
            float xLinea = margenes.X;
            yLinea = DibujaTitulo<Trabajo>(lienzo, new PointF(xLinea, yLinea),
                trabajoSeleccionado);

            int[] rutaFabricacion = trabajoSeleccionado.RutaFabricacion;
            string stringRutafabricacion = "";

            for(var i = 0; i < rutaFabricacion.Length; i++)
            {
```

```
            if (i != 0) stringRutafabricacion += ", ";
            if (rutaFabricacion[i] > 0) stringRutafabricacion +=
                rutaFabricacion[i];
            else stringRutafabricacion += -1 * rutaFabricacion[i];
        }

        if (trabajoSeleccionado.SecuenciaRigida)
        {
            yLinea = DrawStringLines(lienzo, new PointF(xLinea, yLinea),
                Ancho,
                $"Ruta de fabricación: Fija, ({stringRutafabricacion})",
                    Font, brushForeColor);
        }
        else if (rutaFabricacion.Length > 1)
        {
            yLinea = DrawStringLines(lienzo, new PointF(xLinea, yLinea),
                Ancho,
                $"Ruta de fabricación: Flexible, ({stringRutafabricacion})",
                    Font, brushForeColor);
        }
        else
        {
            lienzo.DrawString($"Ruta de fabricación: Modelo máquinas
                paralelas", Font, brushForeColor, xLinea, yLinea);
            yLinea += Font.GetHeight(lienzo);
        }

        if (trabajoSeleccionado.GetNumeroOperacionesAsignadas() > 0)
        {
            if (trabajoSeleccionado.Programado)
            {
                lienzo.DrawString("Trabajo programado completamente.", Font,
                    brushForeColor, xLinea, yLinea);
                yLinea += Font.GetHeight(lienzo);
            }
            else
            {
                int[] rutaProgramada =
                    trabajoSeleccionado.RutaFabricacionProgramada;
                List<int> maquinasFaltantes = new List<int>();
                foreach (int elemento in rutaFabricacion)
                {
                    bool identico = false;
                    foreach (int idMaquina in rutaProgramada) if (elemento
                        == idMaquina) identico = true;
                            if (!identico) maquinasFaltantes.Add(elemento);
                }

                string stringMaquinasFaltantes = "";

                for (var i = 0; i < maquinasFaltantes.Count; i++)
                {
                    if (i != 0) stringMaquinasFaltantes += ", ";
                    stringMaquinasFaltantes += maquinasFaltantes[i];
                }
```

```
                    if (maquinasFaltantes.Count > 1)
                    {
                        yLinea = DrawStringLines(lienzo, new PointF(xLinea,
                            yLinea), Ancho,
                            $"Trabajo no programado completamente, faltan las
                                máquinas ({stringMaquinasFaltantes})",
                            Font, brushForeColor);
                    }
                    else
                    {
                        yLinea = DrawStringLines(lienzo, new PointF(xLinea,
                            yLinea), Ancho,
                            $"Trabajo no programado completamente, falta la
                                máquina {stringMaquinasFaltantes}",
                            Font, brushForeColor);
                    }
                }

                lienzo.DrawString($"Operaciones asignadas:
                    {trabajoSeleccionado.GetNumeroOperacionesAsignadas()}",
                    Font, brushForeColor, xLinea, yLinea);
                yLinea += Font.GetHeight(lienzo);
                for (var i = 0; i <
                    trabajoSeleccionado.GetNumeroOperacionesAsignadas(); i++)
                {
                    string mensajeOperacion;
                    mensajeOperacion = $"Inicio:
                        {trabajoSeleccionado.GetInicioOperacion(i)}, " +
                        $"Fin: {trabajoSeleccionado.GetFinOperacion(i)}," +
                        $" Duración:
                            {trabajoSeleccionado.GetTiempoProcesoOperacion(i)},"
                            +
                        $" Máquina:
                            {trabajoSeleccionado.GetIDMaquinaOperacion(i)}";

                    lienzo.DrawString(mensajeOperacion, Font, brushForeColor,
                        xLinea, yLinea);
                    yLinea += Font.GetHeight(lienzo);
                }
            }
            else
            {
                lienzo.DrawString("Trabajo no programado.", Font,
                    brushForeColor, xLinea, yLinea);
                yLinea += Font.GetHeight(lienzo);
            }

            if (trabajoSeleccionado.GetOperarios().Length > 0)
            {
                lienzo.DrawString($"Operarios asignados:
                    {trabajoSeleccionado.GetOperarios().Length}", Font,
                    brushForeColor, xLinea, yLinea);
                yLinea += Font.GetHeight(lienzo);
                for (var i = 0; i < trabajoSeleccionado.GetOperarios().Length;
                    i++)
                {
```

```
                lienzo.DrawString(
                    trabajoSeleccionado.GetOperarios()[i].Nombre, Font,
                    brushForeColor, xLinea, yLinea);
                yLinea += Font.GetHeight(lienzo);
            }
        }
        else
        {
            lienzo.DrawString("Sin operarios asignados.", Font,
                brushForeColor, xLinea, yLinea);
            yLinea += Font.GetHeight(lienzo);
        }

        return (int)Math.Ceiling(yLinea);
    }

    int DibujaTextoElemento(Graphics lienzo, Maquina maquinaSeleccionada)
    {
        float yLinea = margenes.Y;
        float xLinea = margenes.X;
        yLinea = DibujaTitulo<Maquina>(lienzo, new PointF(xLinea, yLinea),
            maquinaSeleccionada);

        lienzo.DrawString($"Tiempo total disponible:
            {maquinaSeleccionada.GetTiempoDisponible()} minutos", Font,
            brushForeColor, xLinea, yLinea);
        yLinea += Font.GetHeight(lienzo);
        lienzo.DrawString($"Tiempo total programado:
            {maquinaSeleccionada.GetTiempoProgramado()} minutos", Font,
            brushForeColor, xLinea, yLinea);
        yLinea += Font.GetHeight(lienzo);

        return (int)Math.Ceiling(yLinea);
    }

    int DibujaTextoElemento(Graphics lienzo, Operario operarioSeleccionado)
    {
        float yLinea = margenes.Y;
        float xLinea = margenes.X;
        yLinea = DibujaTitulo<Operario>(lienzo, new PointF(xLinea, yLinea),
            operarioSeleccionado);

        int numeroTrabajosAsignados =
            operarioSeleccionado.GetTrabajosAsignados();
        if (numeroTrabajosAsignados > 0)
        {
            lienzo.DrawString($"Trabajos asignados:
                {numeroTrabajosAsignados}", Font, brushForeColor, xLinea,
                yLinea);
        }
        else
        {
            lienzo.DrawString("Sin trabajos asignados.", Font,
                brushForeColor, xLinea, yLinea);
        }
        yLinea += Font.GetHeight(lienzo);
```

```
            return (int)Math.Ceiling(yLinea);
        }


        private float DibujaTitulo<T>(Graphics lienzo, PointF origen, T
            elementoDibujado) where T : IDatosIdentificativos
        {
            float xLinea = origen.X;
            float yLinea = origen.Y;
            StringFormat sf = new StringFormat();
            sf.Alignment = StringAlignment.Center;
            sf.LineAlignment = StringAlignment.Center;
            if (elementoDibujado.Nombre != null)
            {
                lienzo.DrawString(elementoDibujado.Nombre, fontTitulo,
                    brushForeColor, new RectangleF(xLinea, yLinea, Ancho,
                    Font.GetHeight(lienzo)), sf);
                yLinea += Font.GetHeight(lienzo);
            }
            else
            {
                lienzo.DrawString($"Trabajo {elementoDibujado.Identificador}",
                    Font, brushForeColor, xLinea, yLinea);
                yLinea += Font.GetHeight(lienzo);
            }

            if (elementoDibujado.Descripcion != null)
            {
                yLinea = DrawStringLines(lienzo, new PointF(xLinea, yLinea),
                    Ancho, elementoDibujado.Descripcion, Font, brushForeColor);
            }

            lienzo.DrawString($"Identificador:
                {elementoDibujado.Identificador}", Font, brushForeColor,
                xLinea, yLinea);
            yLinea += Font.GetHeight(lienzo);

            return yLinea;
        }
    }

    public class LabelGanttGeneral : LabelGantt
    {
        public LabelGanttGeneral() : base()
        {

        }

        protected override int DibujaTextoInformativo(Graphics lienzo)
        {
            float xLinea = margenes.X;
            float yLinea = margenes.Y;
            StringFormat sf = new StringFormat();
            sf.Alignment = StringAlignment.Center;
            sf.LineAlignment = StringAlignment.Center;
```

```
            lienzo.DrawString("Datos generales", fontTitulo, brushForeColor,
                new RectangleF(xLinea, yLinea, Ancho,
                fontTitulo.GetHeight(lienzo)), sf);
            yLinea += fontTitulo.GetHeight(lienzo);

            yLinea += 5;

            sf.Alignment = StringAlignment.Near;
            sf.LineAlignment = StringAlignment.Center;
            lienzo.DrawString($"Pacientes Programados:
                {Trabajo.GetTrabajosProgramados()}", Font, brushForeColor, new
                RectangleF(xLinea, yLinea, Ancho, Font.GetHeight(lienzo)), sf);
            yLinea += Font.GetHeight(lienzo);
            lienzo.DrawString($"Pacientes No Asignados:
                {Trabajo.GetTrabajosNoProgramados()}", Font, brushForeColor,
                new RectangleF(xLinea, yLinea, Ancho, Font.GetHeight(lienzo)),
                sf);
            yLinea += Font.GetHeight(lienzo);
            lienzo.DrawString($"Tiempo Total Disponible:
                {Maquina.GetTiempoTotalDisponible()} minutos", Font,
                brushForeColor, new RectangleF(xLinea, yLinea, Ancho,
                Font.GetHeight(lienzo)), sf);
            yLinea += Font.GetHeight(lienzo);
            lienzo.DrawString($"Tiempo Total Programado :
                {Maquina.GetTiempoTotalProgramado()} minutos", Font,
                brushForeColor, new RectangleF(xLinea, yLinea, Ancho,
                Font.GetHeight(lienzo)), sf);
            yLinea += Font.GetHeight(lienzo);

            return (int)Math.Ceiling(yLinea);
        }
    }
}
```

Código A.17 Código empleado para elaborar la clase abstracta heredada LabelGantt y sus tres controles heredados, LabelGanttElementos y LabelGanttGeneral.

Índice de Figuras

Índice de Códigos

Bibliografía

[Dios et al., 2015] Dios, M., Molina-Pariente, J. M., Fernandez-Viagas, V., Andrade-Pineda, J. L., and Framinan, J. M. (2015). A decision support system for operating room scheduling. *Computers & Industrial Engineering*, 88:430–443.

[Domschke et al., 1997] Domschke, W., Scholl, A., and Voß, S. (1997). *Produktionsplanung*, volume 2.

[Framinan et al., 2014] Framinan, J. M., Leisten, R., and García, R. R. (2014). *Manufacturing scheduling systems*. Springer.

[Molina-Pariente et al., 2015a] Molina-Pariente, J. M., Fernandez-Viagas, V., and Framinan, J. M. (2015a). Integrated operating room planning and scheduling problem with assistant surgeon dependent surgery durations. *Computers & Industrial Engineering*, 82:8–20.

[Molina-Pariente et al., 2015b] Molina-Pariente, J. M., Hans, E. W., Framinan, J. M., and Gomez-Cia, T. (2015b). New heuristics for planning operating rooms. *Computers & Industrial Engineering*, 90:429–443.

[Pinedo, 2012] Pinedo, M. (2012). *Scheduling*, volume 29. Springer.