Departamento de informática

PROYECTO FIN DE GRADO

SIMULACIÓN DEL MOVIMIENTO DE TRENES PARA MINIMIZAR EL CONSUMO ENERGÉTICO Y OPTIMIZACIÓN PARA PLATAFORMAS MULTICORE

Autor: Manuel Muñoz Archidona

Tutor: Rubén Saá Álvarez

Leganés, junio de 2012

Esta página ha sido dejada en blanco de forma intencionada.

Agradecimientos

Me gustaría mencionar en primer lugar a mi tutor del proyecto. Gracias Rubén por la paciencia que has tenido conmigo, por todo lo que me has enseñado durante el último año, y lo que me has ayudado para que el proyecto saliera adelante lo mejor posible.

Gracias a mi familia, por apoyar siempre mis decisiones, y por darme la oportunidad de formarme en la universidad. Nunca olvidaré lo que habéis hecho por mí y mi futuro.

A mis compañeros de clase, en especial a Alberto, Gabriel y Álvaro. Con los que he compartido experiencias y horas de trabajo durante estos cuatro años. Sin olvidarme de mis compañeros de departamento. En todo este tiempo he aprendido mucho gracias a vosotros, a vuestros consejos y a vuestras enseñanzas. Siempre os tendré en consideración.

A mis amigos Patricia, Rafael y Roberto. Tanto por la ayuda ofrecida como por los ánimos que me han dado en todo momento.

Resumen

Los avances en computación de los últimos 25 años han promovido la simulación del movimiento y consumo de trenes en circulaciones. El presente proyecto contempla la creación de un algoritmo que simula el movimiento de un tren bajo una infraestructura ferroviaria real. La ejecución de un algoritmo de este estilo supone una gran carga computacional que plantea la necesidad de usar computación en paralelo. En este documento se describen los aspectos que han envuelto la creación del proyecto.

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1

Introducción y objetivos

Capítulo 1: Introducción y objetivos

1.1 Introducción

La aparición del tren fue producto de la revolución industrial originada en Inglaterra durante los siglos XVIII y XIX. Desde ese momento pasó a tener un papel crucial en los transportes tanto de mercancías como de personas. Su presencia en los diferentes países suponía una gran ventaja en cuestiones de industrialización y, por tanto, ha tenido una gran influencia en el desarrollo de muchas sociedades por todo el mundo.

Hasta el día de hoy, los trenes han estado sujetos a los avances tecnológicos. El fin ha sido siempre mejorar las prestaciones, es decir, proporcionar un medio transporte que cumpla las expectativas de los usuarios, y que haga uso de una infraestructura de calidad para obtener los máximos beneficios para la empresa propietaria y sus clientes directos.

Las primeras locomotoras eran impulsadas por vapor de agua. Esta forma de tracción predominó hasta mediados del siglo XX. Las locomotoras a vapor eran máquinas sencillas y adaptables a una gran variedad de combustibles, pero su escasa eficiencia y el considerable trabajo que provocaba su mantenimiento fomentaron la aparición de nuevas formas de tracción.

A partir de mediados del siglo XX aparecieron las locomotoras impulsadas por motores diesel o eléctricos, que desbancaron a las locomotoras a vapor. En concreto, las locomotoras eléctricas proporcionan una serie de ventajas [1] entre las que destacan un menor impacto sobre el medio ambiente por parte de la locomotora, mejores prestaciones y mayor potencia desarrollada por los motores eléctricos, menor coste de mantenimiento, y menor coste de la energía eléctrica con respecto a las de vapor.

Las compañías de ferrocarril tienden a desplegar líneas aéreas de contacto como mecanismo para suministrar dicha energía eléctrica a las locomotoras [2]. El hecho de ser un sistema aéreo aporta dos ventajas adicionales sobre otros sistemas de suministro localizados a nivel del suelo. Por un lado, existe mayor seguridad en lo referente a contactos accidentales de personas o animales. Por otro lado, tiene menos restricciones de voltaje gracias a la elevación sobre el terreno, por lo que permite usar locomotoras más potentes y tener más tráfico por las vías.

A pesar de que todas estas ventajas animen a las operadoras de ferrocarriles a implantar locomotoras eléctricas, el coste de montar la infraestructura (tendido de la catenaria, subestaciones eléctricas de suministro de energía, sistemas de control) es muy elevado [3]. Por tanto, los administradores de las infraestructuras han de buscar medidas para conseguir que el uso de las mismas sea lo más eficiente posible.

1.2 Motivación

Como ya se ha explicado en el apartado anterior, la implantación de trenes eléctricos supone un importante gasto para los administradores de infraestructuras ferroviarias. Como cualquier empresa, y sobre todo en el panorama económico actual, los

administradores necesitan optimizar sus recursos para obtener el máximo beneficio posible.

Uno de los gastos más importantes, y el que se trata en este trabajo, es debido al consumo eléctrico. Los trenes eléctricos son máquinas que pesan y transportan cientos de toneladas. Para mover toda esa maquinaria es necesario proporcionar una cantidad de energía suficiente y acorde a su tamaño, sin que haya excesos. Además, cuanto mayor sea la velocidad de circulación de los trenes, mayores serán los costes de consumo eléctrico [4]. Por tanto, una buena estrategia de ahorro consiste en la reducción del consumo eléctrico en las circulaciones. Se entenderá por circulación como un trayecto realizado por un tren, a lo largo de parte de la infraestructura ferroviaria, y que cumple unos tiempos predeterminados.

Si se logra planificar y diseñar circulaciones que permitan al tren realizar el mínimo consumo posible, se estará reduciendo al máximo el gasto energético, con el consiguiente ahorro de costes.

Minimizar el consumo eléctrico de un tren está directamente relacionado con obtener el movimiento óptimo del mismo para una circulación dada. Dicho movimiento está influenciado por distintos parámetros a tener en cuenta:

- El trayecto de la circulación. Es decir, las estaciones por las que pasa, los horarios de llegada y salida de las mismas, los tiempos de parada en ellas, y el camino que une una estación con otra.

- Las infraestructuras por las que circula el tren. Es necesario conocer las características de las vías. Para cada punto kilométrico de una vía, presente en el recorrido, se ha de saber su pendiente, radio de curva, velocidad máxima, y si hay un túnel o no.

- El tren utilizado y su material motriz. Estos parámetros determinan la capacidad tractora de tren o el peso que lleva remolcado.

Sin embargo, el elevado número de parámetros a considerar para poder obtener el movimiento óptimo de un tren, hace difícil el hecho de realizar los cálculos a mano. Además, es necesario disponer de datos por cada segundo de la circulación, como pueden ser la velocidad del tren para ese segundo, su posición, el esfuerzo del material motriz, o la potencia que necesita.

En otro orden de cosas, las compañías de ferrocarril han intentado aumentar la productividad y estandarizar sus métodos de trabajo desde hace muchos años. Y los ordenadores han jugado un papel importante en este esfuerzo, ya que permiten automatizar numerosos procesos. Por ello, se plantea la necesidad de implementar un algoritmo para ser ejecutado en un ordenador, que permita calcular de forma automatizada el movimiento óptimo de un tren para una circulación.

Además, el número de circulaciones puede llegar a ser muy elevado, por lo que la carga computacional asociada sería grande. Para tratar este aspecto, se pretende optimizar el algoritmo de forma que consiga explotar la capacidad de los ordenadores actuales de ejecutar tareas concurrentemente. Y es que, en los últimos años, el número

de núcleos en los ordenadores ha aumentado, por lo que diferentes circulaciones podrían ser tratadas en diferentes threads al mismo tiempo, optimizando así el rendimiento general.

Con todo lo anterior, y con la motivación de ahorrar costes, la empresa ADIF ha desarrollado en colaboración con el grupo ARCOS de la Universidad Carlos III de Madrid, una **aplicación para calcular y optimizar el consumo energético de trenes eléctricos**. Como parte de la aplicación se implementará el algoritmo mencionado que hará uso de **datos reales** que se toman de la base de datos de ADIF.

1.3 Objetivos

Con este Trabajo de Fin de Grado se pretenden alcanzar los siguientes objetivos:

- Conocer y comprender los elementos que participan en la circulación y que por tanto afectan al movimiento del tren.
  Antes de desarrollar un algoritmo capaz de calcular el movimiento de un tren de manera fiable, es necesario aprender cómo funciona todo en el mundo real. Es decir, la infraestructura utilizada por la empresa ADIF, el material motriz, las circulaciones y la física que envuelve el movimiento de los trenes.

- Diseñar e implementar un algoritmo capaz de calcular el movimiento óptimo de un tren dada una circulación.
  Una vez ya se conocen todos los aspectos que giran alrededor de una circulación, hay que analizar cómo desarrollar un algoritmo que genere un movimiento óptimo para el tren. Comprendiendo óptimo como aquel movimiento que permita obtener un menor consumo eléctrico.

- Proporcionar un entorno adecuado al usuario para el uso de la aplicación. Por ejemplo, proporcionarle gráficas que permitan analizar mejor los datos devueltos por el algoritmo, o mostrándole mensajes cuando se produce un error.
  De esta forma nos aseguramos un correcto feedback con el usuario, que disfrutará de una mejor experiencia en el uso de la aplicación. Este aspecto es tenido en cuenta desde el análisis y diseño del proyecto, y su evolución se mantiene a lo largo del desarrollo.

- Aprender a utilizar el entorno de desarrollo Microsoft Visual Studio en el lenguaje C#:
  De esta forma se garantiza el uso de todos los recursos que nos proporciona tanto el lenguaje como el entorno. Por tanto, el resultado del desarrollo será de mayor calidad.

- Aplicar los conocimientos adquiridos a lo largo de la carrera para desarrollar un código estructurado, cuyos futuros cambios no supongan una grave penalización en tiempo y dinero para el proyecto del que forma parte.

1.4 Estructura del documento

- **Capítulo 1: Introducción y objetivos**. Capítulo en el que se hace una breve introducción al tema del proyecto, se explican sus objetivos, y la estructura del documento.
- **Capítulo 2**: **Estado del arte.** Capítulo donde se hace referencia a la situación actual de las aplicaciones centradas en la simulación de movimiento para trenes.
- **Capítulo 3: Aspectos técnicos de la infraestructura**. Describe los elementos de la infraestructura ferroviaria que son necesarios para el proyecto.
- **Capítulo 4: Aspectos técnicos de la circulación**. Describe los elementos básicos de una circulación ferroviaria.
- **Capítulo 5: Aspectos técnicos del algoritmo de movimiento**. Analiza y describe el funcionamiento del algoritmo de movimiento.
- **Capítulo 6: Análisis**. Refleja los aspectos que se han de tener en cuenta para el posterior diseño del software. Incluye la definición de los casos de uso y de los requisitos de software
- **Capítulo 7: Diseño**. Detalla las características del proyecto para facilitar su implantación por parte del equipo de trabajo. Define la arquitectura del sistema y las clases del proyecto.
- **Capítulo 8: Pruebas**. Analiza las pruebas realizadas sobre el funcionamiento del programa una vez ha sido creado.
- **Capítulo 9: Presupuesto**. Contiene los gastos que ha supuesto la realización del proyecto. Contempla las fases del desarrollo, los gastos en personal, en recursos tecnológicos y en aplicaciones para estos.
- **Capítulo 10: Conclusiones y líneas futuras**. Tras la finalización del proyecto, se desarrollan una serie de conclusiones y líneas futuras a tener en cuenta.
- **Capítulo 11: Bibliografía**. Muestra la bibliografía usada para la realización del proyecto.

1.5 Glosario de términos y acrónimos

Término	Definición
Algoritmo	"Conjunto preescrito de instrucciones o reglas bien definidas, ordenadas y finitas que permite realizar una actividad mediante pasos sucesivos".
Array	"Colección de datos que puede ser seleccionada por índices".
C#	"Lenguaje de programación orientado a objetos y estandarizado por Microsoft como parte de su plataforma .NET"
Catenaria	"Línea aérea de alimentación que transmite energía eléctrica a las locomotoras u otro material motor".
Feedback	"Feedback o retroalimentación es un mecanismo de control de los sistemas dinámicos por el cual una cierta proporción de la señal de salida se redirige a la entrada, y así regula su comportamiento".
Fichero log	Fichero de registro de datos.
Framework	"Es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, con base a la cual otro proyecto de software puede ser más fácilmente organizado y desarrollado".
Interfaz	"Medio con que el usuario puede comunicarse con una máquina, un equipo, o una computadora".
Interpolación	"Se denomina interpolación a la obtención de nuevos puntos partiendo del conocimiento de un conjunto discreto de puntos".
Multicore	"Microprocesador que combina dos o más procesadores independientes en un solo paquete"
Procesador	"Componente del computador que interpreta las instrucciones contenidas en los programas y procesa los datos"
Software	"Conjunto de componentes lógicos necesarios que hacen posible la realización de tareas específicas, en contraposición a los componentes físicos".
Subestación eléctrica	"Instalación destinada a modificar y establecer los niveles de tensión de una infraestructura eléctrica".

Thread	"Unidad de procesamiento más pequeña que puede ser planificada por un sistema operativo".

Acrónimo	Significado
1 **ADIF**	*Administrador de Infraestructuras Ferroviarias*
2 **ARCOS**	*Grupo de investigación de arquitectura de computadores, comunicaciones y sistemas de la UC3M.*
3 **SSE**	*Streaming SIMD Extensions*. Extensión al grupo de instrucciones MMX para procesadores Pentium III y Pentium IV.
4 **SIMD**	*Single Instruction, Multiple Data*. Técnica empleada para conseguir paralelismo a nivel de datos.
5 **GPS**	*Global Positioning System*. Sistema global de navegación por satélite que permite determinar en todo el mundo la posición de un objeto.
6 **IVA**	*Impuesto sobre el Valor Añadido.*

CAPÍTULO 2

Estado del arte

Capítulo 2: Estado del arte

El uso de herramientas de simulación para ayudar a resolver problemas de ingeniería está ampliamente expandido [5]**¡Error! No se encuentra el origen de la referencia.**. El dominio ferroviario no es una excepción a esta regla, puesto que muchos trabajos de investigación se han llevado a cabo para mejorar la fiabilidad y la calidad de sus servicios. Bae sugiere en [6] el uso de la simulación como una forma efectiva de optimizar la localización de instalaciones eléctricas. Banerjee propone en [7] un modelo de grafos para estudiar la estabilidad y comportamiento del tren en la vía férrea. Yalçinkaya muestra en [8] un framework de simulación para calcular un horario factible para un conjunto de trenes. Un simulador para el diseño de configuraciones óptimas de agujas aéreas en desvíos ferroviarios se recoge en [9].

La simulación ha sido aplicada en múltiples áreas del dominio ferroviario, como la energía, la planificación, la mecánica, etc. El cálculo del movimiento de un tren forma parte de una de esas áreas que aparecen en múltiples artículos de investigación.

Por ejemplo, Goodman [10] describe el movimiento del tren como el cálculo de los perfiles de velocidad y distancia cuando un tren está viajando de un punto a otro de acuerdo a las limitaciones impuestas por el sistema de señalización y las características del material motriz. A medida que avanza el tren por el camino, el movimiento se ve afectado también por las restricciones de la geometría del terreno y las limitaciones de velocidad, haciendo que el cálculo sea dependiente de la posición. Asimismo, propone dos modelos para calcular el movimiento del tren en función del tiempo: un modelo basado en el tiempo, y otro basado en eventos.

En el modelo de simulación basado en el tiempo, el horario del movimiento es dividido en intervalos para que el movimiento de tren sea evaluado para cada intervalo. El comportamiento del tren es actualizado de forma simultánea y el sistema evoluciona constantemente en el tiempo. Conceptualmente, esto es cercano al movimiento de los trenes en la realidad. Este tipo de modelos de simulación son más sencillos de construir. A pesar de su simplicidad, el modelo basado en el tiempo supone normalmente una gran carga computacional puesto que una significante cantidad de información tiene que ser producida para cada actualización del tiempo.

Por otro lado, el modelo basado en eventos, se centra en el progreso del movimiento del tren para una secuencia de eventos predefinidos, como la llegada o salida de las estaciones. Como los eventos están relacionados unos con otros, las interacciones con los trenes dispararán los diferentes eventos. Finalmente se obtiene una secuencia de eventos que determina el progreso de los trenes.

El esfuerzo computacional se reduce de forma substancial puesto que se evita el cálculo de detalles exactos del movimiento del tren entre pares de eventos.

A tenor de que un simulador debe aplicar las leyes de la física al cálculo del movimiento de trenes, todos los simuladores comerciales deben ser muy similares en los cálculos que desarrollan [11]. Las principales diferencias se encuentran en la interfaz del usuario y en los datos de entrada.

En la simulación más básica de todas, se produce un balance de fuerzas. La locomotora aplica una fuerza que supera a la resistencia y acelera el tren. La aceleración se produce hasta que se alcanza una balanza donde las fuerzas de aceleración coinciden con las fuerzas de resistencia al avance, alcanzando el tren una velocidad constante. Esta balanza de velocidad se mantiene hasta que la fuerza motriz es eliminada y reemplazada por otra fuerza de frenado que provoca que el tren pare.

En la mayoría de los simuladores, la velocidad de los trenes podría estar controlada por cuatro parámetros: la velocidad de la vía, la velocidad máxima del tren, las señalizaciones que limitan la velocidad de forma condicional, y la velocidad establecida para la ruta. La limitación de velocidad activa del tren es la menor de las cuatro.

Además, la gran parte de los simuladores hacen uso de los tiempos que tarda el tren para viajar entre dos puntos de la red ferroviaria. También incluyen los factores que tienen que ser considerados para diseñar el material motriz de una simulación dada: la velocidad máxima, el freno, la disponibilidad de la ruta, la resistencia al avance del tren, su longitud, etc.

Otras simulaciones hacen uso de algoritmos genéticos para optimizar los movimientos de los trenes. Además, tal y como se aprecia en el trabajo de Chang [12], tienen en cuenta otros aspectos como la planificación del tren, o la carga de pasajeros esperada. Buscando de esta forma un movimiento óptimo en términos de puntualidad, confort por parte del pasajero, y consumo eléctrico.

Como ya se ha dicho, la simulación del movimiento de trenes eléctricos puede suponer una gran carga computacional. Ding Yong propone en su artículo [13] el uso de computación en paralelo para la simulación del movimiento de varios trenes sobre una vía ferroviaria electrificada. Para ello utiliza el modelo de ejecución SSE, presente en la arquitectura de los procesadores Intel Pentium III, e Intel Pentium IV. SSE permite acelerar las aplicaciones en un solo procesador. Para su correcto uso, el autor rediseñó varios procedimientos para explotar el paralelismo SIMD en el ordenador. Por ejemplo, desenrollando bucles y modificando el uso de los registros del procesador.

La creación de un simulador de carácter general supone una serie de dificultades [14]. Esto se debe a la diferencia de los sistemas ferroviarios de cada país o región. Por tanto, en el panorama actual se pueden elegir varios tipos de simuladores, que hacen uso de cálculos similares, pero que, en su mayoría, están destinados a un sistema ferroviario en concreto.

Como consecuencia de la falta de simuladores de carácter general, la empresa ADIF considera necesario un simulador de movimiento de trenes que le permita obtener el gasto energético de las circulaciones. Este proyecto cubre esa necesidad, proporcionando una herramienta de simulación, para una aplicación definida, que se ajusta a las características de la infraestructura ferroviaria española que es propiedad de ADIF. Por tanto, la herramienta proporcionada hará uso de datos reales procedentes de la base de datos de ADIF.

El algoritmo utilizado para la simulación está diseñado, según la clasificación de Goodman [10], de acuerdo a un modelo basado en el tiempo. También tiene en cuenta las leyes de la física y los aspectos utilizados por la mayoría de los simuladores. Por

tanto, cuando se quieran simular varias circulaciones a la vez, será necesario optimizar la ejecución mediante computación en paralelo. De esta forma, la pesada carga computacional de este modelo de simulación se verá reducida.

CAPÍTULO 3

Aspectos técnicos de la infraestructura

Capítulo 3: Aspectos técnicos de la infraestructura

La infraestructura es clave en el movimiento y consumo energético de los trenes. Cuantas más dificultades tenga el tren para avanzar debido a la infraestructura, mayor será el consumo energético que necesitará. A continuación se describen los elementos de la infraestructura y la información que aportan al algoritmo de movimiento.

3.1 Vía

La vía es la parte principal de la infraestructura sobre la que se desplazan los trenes. En ella convergen el resto de elementos de la infraestructura que afectan al algoritmo. Por tanto, para cada punto kilométrico de la vía, hay asociados una serie de parámetros como la pendiente o la velocidad máxima permitida.

3.2 Estaciones

El camino que el tren ha de seguir en una circulación está establecido por medio de un trayecto. Un trayecto no es más que una lista ordenada de las estaciones a las que ha de ir el tren.

Una estación puede estar conectada a una o varias vías. Por cada una de ellas tendrá un punto kilométrico. Cada estación entonces se localizará en diferentes puntos kilométricos dependiendo de la vía que se esté teniendo en cuenta.

Entonces, para que dos estaciones estén comunicadas tienen que tener al menos una vía en común. De esta forma, las estaciones son necesarias para indicar por donde ha de ir el tren en su movimiento y los tiempos que ha de cumplir.

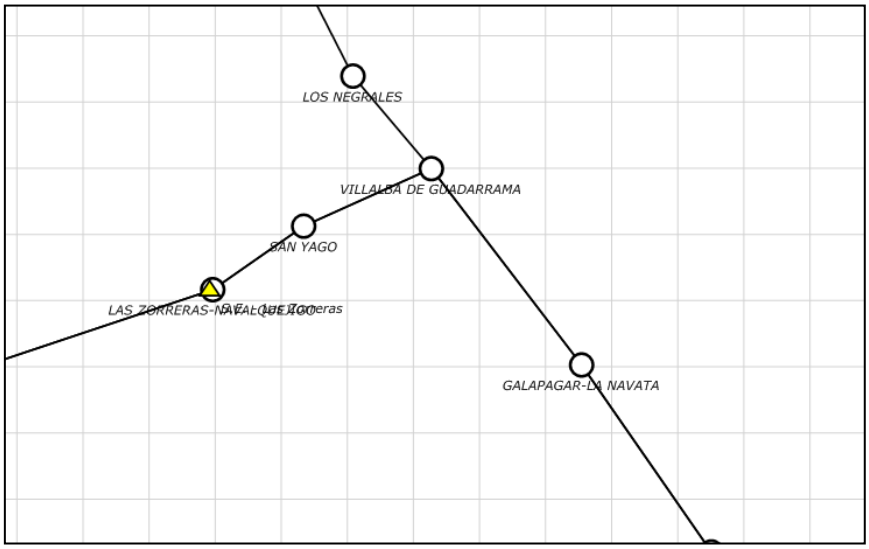


Figura 1. Representación de estaciones unidas por vías en el entorno de la aplicación

En la figura 1 se puede ver cómo se representan las estaciones en la aplicación desarrollada. Esta representación se puede realizar gracias a las coordenadas GPS de cada una de las estaciones. De esta forma se puede ver cómo las estaciones están unidas

por medio de las vías. Es más, se puede ver como por la estación de "Villalba de Guadarrama" pasan dos vías diferentes.

3.3 Perfiles en planta

Los perfiles en planta son las curvas que hay en la vía. Debido a la gran longitud y masa de los trenes, las curvas afectan de forma negativa al movimiento del tren.

Para cada perfil en planta se deben considerar los puntos kilométricos de inicio y fin en la vía. De esta forma, cuando el tren esté situado en un punto kilométrico de una determinada vía, podremos saber si está pasando por una curva.

Además, también se almacena en metros el radio de dicha curvatura. Cuanto mayor es el radio, el perfil en planta se asemeja más a una recta, mientras que cuanto menor es, la curva es más acentuada y el tren adquiere una mayor resistencia en su avance.

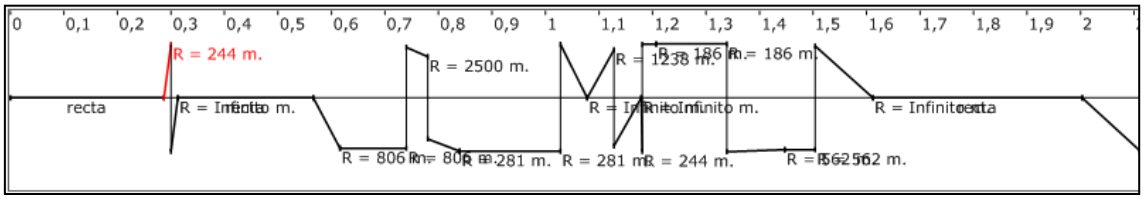


Figura 2. Representación de perfiles en planta en el entorno de la aplicación

En la figura 2 se muestra la representación de los perfiles en planta de una vía en la aplicación desarrollada. En el margen superior se indica el punto kilométrico de la vía en el que se encuentra la curva. Si la curva es hacia la izquierda, se representa por encima del eje central, si va hacia la derecha, se representa por debajo del eje central. Por cada curva además se muestra el radio correspondiente.

3.4 Perfiles en alzado

Los perfiles en alzado nos indican la pendiente que hay a lo largo de una vía. La pendiente afecta notablemente al movimiento del tren debido a su gran masa. Para subir una pendiente, el tren tiene que incrementar la potencia y el esfuerzo motriz. Por tanto, aumenta el gasto energético.

En la aplicación desarrollada, los perfiles vienen dados por sus puntos kilométricos de inicio y fin en la vía, y por el valor de la pendiente en tantos por mil (‰).

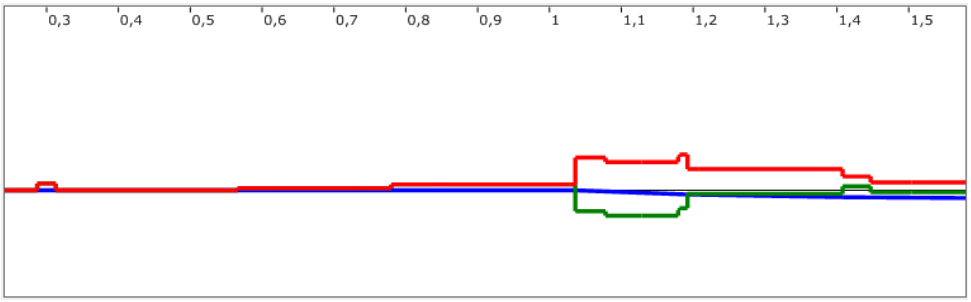


Figura 3. Representación de perfiles en alzado en el entorno de la aplicación

En la figura 3 se muestra la representación de los perfiles en alzado de una vía en la aplicación. La línea verde muestra la pendiente cuando se recorre la vía en sentido kilométrico ascendente, y la línea roja la pendiente cuando se recorre en sentido descendente. Para su representación se ha escalado el valor representado de las pendientes, puesto que al tener valores muy bajos y en tantos por mil, apenas se podrían apreciar visualmente.

3.5 Tramos de velocidad

Las vías poseen unos tramos de velocidad que indican la velocidad máxima a la que un tren puede pasar por ellos. El cambio de un tramo de velocidad a otro puede provocar que el tren cambie a su vez la velocidad que lleva. Consecuentemente el tren acelerará o frenará y, como no se mantiene una velocidad continua, acaba aumentando el consumo.

3.6 Túneles

Las vías pueden atravesar túneles. Atravesar un túnel supone incrementar la resistencia que tiene que vencer el tren al avanzar. Esta resistencia adicional vendrá dada por un coeficiente característico del túnel. Para cada túnel se guarda el punto kilométrico inicial y final en la vía.

CAPÍTULO 4

Aspectos técnicos de la circulación

Capítulo 4: Aspectos técnicos de la circulación

Antes de analizar y diseñar el algoritmo de movimiento de circulaciones, es necesario comprender que son las circulaciones. En el presente capítulo no sólo se describe que es una circulación, sino también a aquellos elementos que forman parte de ella.

4.1 Circulación

Una circulación está formada básicamente por dos elementos: un tren y un trayecto con un horario asociado. En este proyecto se busca que el algoritmo obtenga la forma de recorrer el trayecto de la circulación ahorrando consumo energético y cumpliendo con los tiempos establecidos.

Para cada estación del trayecto habrá una hora de llegada del tren y una hora de salida. La diferencia entre ambos horarios supondrá el tiempo de parada del tren en la estación. En los casos en los que la hora de llegada y la hora de salida coinciden, el tren no para en la estación.

4.2 Trenes

Los trenes son una serie de vagones o vehículos propulsados por un material motriz. Estos vehículos están dotados de unas ruedas que les permiten circular por raíles. Para cada tren se tendrá en cuenta el número de materiales motrices que forman parte de él, el peso remolcado, su velocidad máxima y su tipo de uso (pasajeros o mercancías).

Figura 4. Tren Serie 330 de ADIF [www.treneando.com]

4.2.1 Material Motriz

El material motriz es aquel vehículo del tren que tiene capacidad tractora, pero que no lleva carga. El resto de vehículos serán material remolcado y el algoritmo sólo tendrá en cuenta el peso adicional que proporcionan al tren. A continuación se describen cada uno de los componentes del material motriz que el algoritmo tiene en consideración.

Figura 5: Material Motriz Serie 330 [www.ferropedia.es]

a) Parámetros globales

Para cada material motriz se considerarán los siguientes parámetros globales:

- **Masa**: la masa del material motriz en kg.
- **Coeficiente de adherencia estático**: expresa la oposición en reposo al deslizamiento que ofrecen las superficies de las ruedas del tren y los carriles de la vía.
- **Velocidad Máxima**: la velocidad máxima que puede alcanzar el material motriz en kilómetros por hora.
- **Esfuerzo de arranque**: el esfuerzo que ha de realizar el material motriz para arrancar medido en kg.
- **Número de ejes**: el número de ejes del material motriz.
- **Longitud**: longitud del material motriz.

b) Coeficientes de resistencia al avance

Al estar en movimiento, el material motriz sufre una resistencia al avance. Esta resistencia es provocada por varios factores como el peso del tren o la pendiente que hay en el camino.

Para hallar la resistencia al avance, se usan tres coeficientes A (kN), B ($\frac{kN}{kmh}$), y C ($\frac{kN}{kmh^2}$) que son característicos de cada material motriz. La expresión simplificada de la resistencia al avance es por tanto:

$$Ravance\ (kN) = A + B \cdot V + C \cdot V^2$$

Como se puede ver, la resistencia depende de la velocidad que lleve el tren. Más adelante se desarrolla la expresión para que tenga en consideración el tipo de tren, el paso por túnel, el avance por pendientes, y el radio de las curvas.

c) Curva de esfuerzo

La curva de esfuerzo nos indica el esfuerzo que necesita realizar el motor del tren en función de la velocidad del mismo. Es decir, la fuerza que actúa sobre el tren para alcanzar una determinada velocidad.

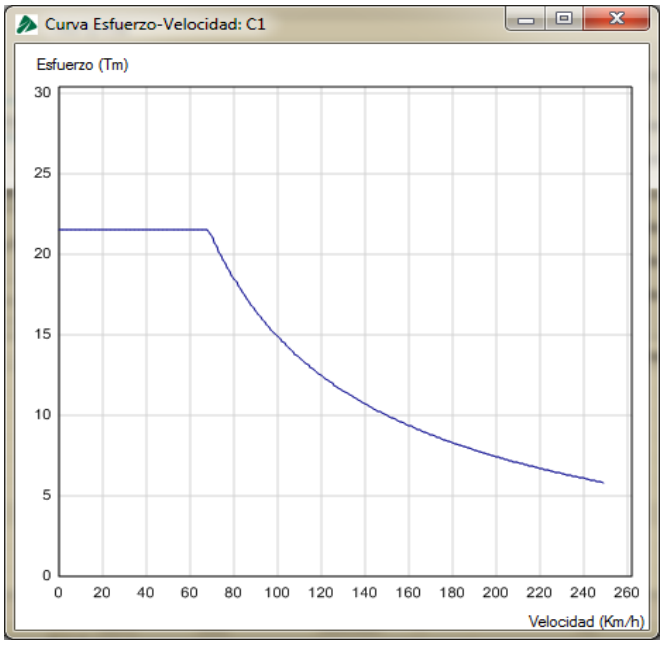


Figura 6. Curva de esfuerzo del automotor TAV S/114

d) Curva de frenado

Al igual que hay una curva de esfuerzo, hay otra de esfuerzo de frenado. Se usará cuando el tren tiene que decelerar para alcanzar una determinada velocidad. Como la curva de esfuerzo, proporciona el valor del esfuerzo de frenado en función de la velocidad del tren.

Si entre los datos proporcionados desde ADIF del material motriz no hay ninguno relacionado con la curva de frenado, se usará con este fin la curva opuesta a la curva de esfuerzo. Es decir, los esfuerzos en la curva de frenado serán los opuestos a los de la curva de esfuerzo.

e) Tipo de material motriz

Se distinguen principalmente dos tipos:

- **Locomotora**: la locomotora es el vehículo con un material motriz que se utiliza para dar tracción a los trenes. Están formados por una única unidad cuya fuerza de tracción permitirá mover el resto de vagones.

Figura 7. Locomotora de la serie 252 [www.ferropedia.es]

- **Automotor**: se trata de un vehículo autopropulsado por un motor eléctrico en nuestro caso. Suelen estar formados por varias unidades semipermanentemente unidas o unidas y articuladas. Aunque también pueden acoplarse varios automotores entre sí.

Figura 8. Automotor de la serie 100 [www.ferropedia.es]

4.3 Trayectos

Para cada circulación hay que definir un trayecto. Un trayecto es una lista de estaciones ordenadas que nos indicará el camino que ha de recorrer el tren en la circulación. Además, para cada tramo entre una estación y otra, habrá que elegir una vía por la que recorrerlo.

En la figura 9 se muestra la representación de una sección del trayecto entre la estación de “El Escorial” y “Madrid-Chamartin”. La línea roja representará el camino que sigue el tren en la circulación.

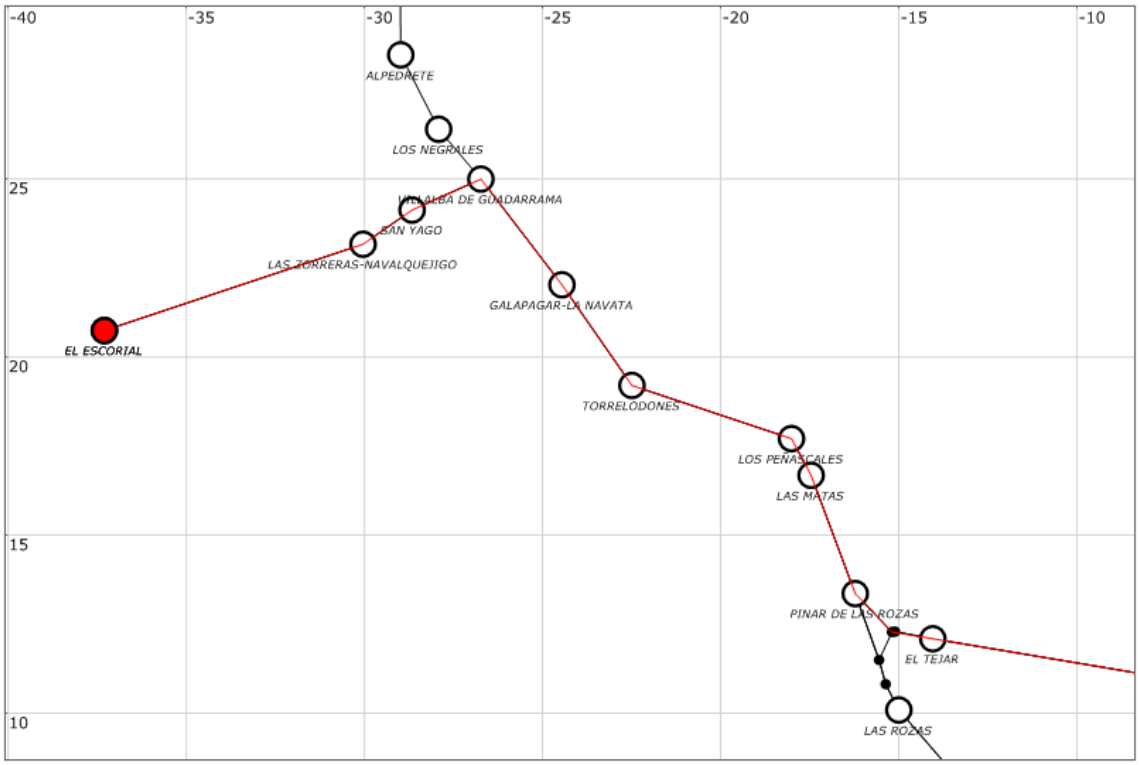


Figura 9. Representación de un trayecto en la aplicación

CAPÍTULO 5

Aspectos técnicos del algoritmo de movimiento

Capítulo 5: Aspectos técnicos del algoritmo de movimiento

Una circulación se considera como una unidad o un tren de composición variable (locomotora mas carga remolcada) con unos horarios de paso y de paradas en las estaciones.

La estimación del movimiento de una circulación sigue un proceso iterativo hasta que se consigue cumplir con las especificaciones de tiempo marcadas, en cada uno de los tramos que se estudian (trayectos entre dos estaciones consecutivas).

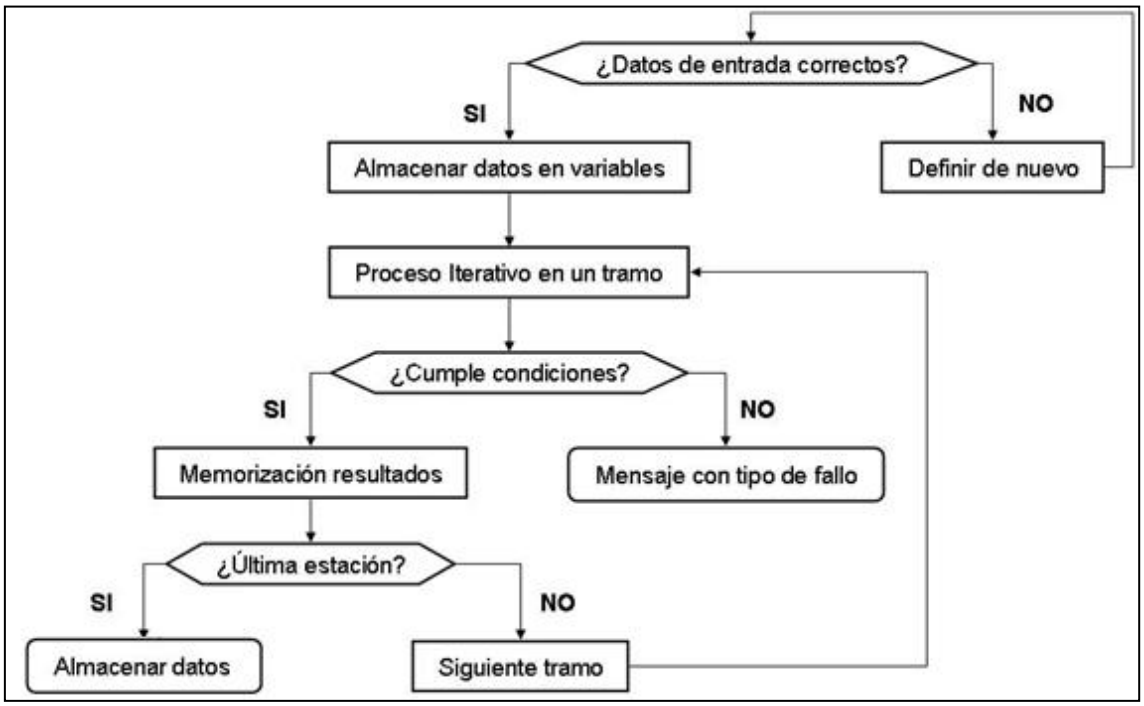


Figura 10. Diagrama de flujo del algoritmo

En la figura 10 se muestra, mediante un diagrama de flujo de algoritmo, los pasos fundamentales en los que se estructura el algoritmo. A continuación se describen estos pasos.

En primer lugar, antes de empezar cualquier cálculo u operación, se comprueba que los datos de entrada son correctos y, si hace falta, sus valores son adaptados a las unidades que necesite el algoritmo. En caso contrario se deben volver a introducir.

Una vez comprobado que son correctos se realizan las operaciones oportunas para calcular los valores que intervienen en el proceso de estimación. La estimación consiste en comprobar si es posible cumplir los horarios con el material móvil y los límites de velocidad especificados.

Las operaciones citadas en el párrafo anterior pueden tener distintos objetivos:

- Almacenar valores en variables (como el valor del peso adicional del material motriz).

- Acceder a elementos de una tabla de la base de datos (como sucede con los datos del material motriz, a través de su nombre).
- Definir otras variables necesarias durante la ejecución del programa. Por ejemplo, se calcula la aceleración media de frenado a partir de las curvas de frenado del material motriz teniendo en cuenta tanto el peso del material motriz como el adicional. Esta aceleración se utiliza durante el proceso de simulación de frenado del tren.

Una vez almacenados los valores necesarios y con la estructura adecuada, comienza el proceso de estimación del movimiento. Si la circulación tiene definidas en total N estaciones (incluidas la inicial y la final), el programa realiza la estimación por tramos en N-1 tramos existentes.

El movimiento del tren se analiza en periodos de un segundo, calculando para cada uno de ellos los siguientes valores:

- Esfuerzo motriz o de frenado
- Rendimiento de tracción y de frenado
- Consumos auxiliares
- Velocidad instantánea
- Pendiente en ‰
- Radio de curvatura en m
- Factor túnel (Vale 1 a cielo abierto, de 1.2 a 2 en túneles)
- Resistencia al avance
- Aceleración
- Distancia recorrida
- Vía en la que se encuentra y su punto kilométrico
- Potencia

Para cada tramo el proceso es el mismo. Se realiza un proceso iterativo hasta conseguir cumplir con los plazos de tiempo marcados en la definición de la circulación, y si se llega a una solución correcta, se almacenan los valores correspondientes. En caso de que durante el proceso iterativo, existan factores que impidan cumplir el horario del tramo, se interrumpe la ejecución del algoritmo y se muestran por pantalla las causas.

Para realizar el proceso iterativo en cada tramo, se conservan los valores de las variables del movimiento en el instante de comienzo del tramo hasta que el proceso iterativo finaliza con éxito. En concreto, lo que se busca es la velocidad uniforme máxima que permita realizar el movimiento con los horarios especificados, considerando el material motriz, el remolcado y el trazado (pendiente, curvas, túneles).

Antes de empezar el proceso se comprueba si es un tramo con parada al final del mismo.

Cuando al final de un tramo hay parada, el algoritmo va calculando en cada instante la distancia que le falta para llegar al final del mismo, y en función de esta se determina si debe empezar a frenar. Cuando está frenando se calcula en cada instante la aceleración **(**negativa**)** en función de la velocidad y la distancia que le falta. Este último cálculo no es necesario si el tren no para al final del tramo.

En ambos casos (con parada o sin parada final) el movimiento de aceleración se realiza de la misma manera. En cada instante del estudio hay que obtener la pendiente, la curvatura y el factor túnel del punto de la vía en la que se encuentra el tren. Además se calcula el esfuerzo de tracción máximo del material motriz a la velocidad actual del tren, interpolando linealmente en los valores disponibles de la curva de esfuerzo máximo del material motriz correspondiente.

5.1 Resistencia al avance

Para cada velocidad se calcula la resistencia al avance. Dependiendo de qué tipo sea la circulación, la expresión obtenida es diferente (de acuerdo a la norma técnica de resistencia al avance de Renfe):

- Unidad:

$$Rav(kN) = A + B{\cdot}Vel + C{\cdot}Tf{\cdot}Vel^2 + \left(Pdte + \frac{800}{Curv}\right){\cdot}\left(\frac{PesoMatMot}{100}\right)$$

- Locomotora con viajeros:

$$Rav(kN) = A + B{\cdot}Vel + C{\cdot}Tf{\cdot}Vel^2 + \left(Pdte + \frac{800}{Curv}\right){\cdot}\left(\frac{PesoMatMot + pesoAdic}{100}\right) +$$

$$+\left(2 + \frac{Vel^2}{4500}\right){\cdot}\left(\frac{pesoAdic}{100}\right)$$

- Locomotora con mercancías

$$Rav(kN) = A + B{\cdot}Vel + C{\cdot}Tf{\cdot}Vel^2 + \left(Pdte + \frac{800}{Curv}\right){\cdot}\left(\frac{PesoMatMot + pesoAdic}{100}\right) +$$

$$+\left(2 + \frac{Vel^2}{1600}\right){\cdot}\left(\frac{pesoAdic}{100}\right)$$

Las variables anteriores tienen las unidades siguientes:

- **Coeficientes de resistencia** al avance del material motriz [15]:

$$A(kN),\quad B{\cdot}\left(\frac{kN}{km/h}\right),\quad C\left(\frac{kN}{(km/h)^2}\right),\ Tf\,(1-2)$$

- **Vel** (velocidad): kilómetro por hora(km/h)
- **Pdte** (pendiente): tanto por mil (‰)
- **Curv** (radio de curvatura): metro (m)
- **PesoMatMot** (masa del material motriz): tonelada (t)
- **PesoAdic** (masa adicional): tonelada (t).

Como se observa en las expresiones anteriores, la resistencia al avance está compuesta por tres grupos de términos.

Un grupo, común a los tres, depende la velocidad que lleve el tren [15]:

$$A + B \cdot Vel + C \cdot Tf \cdot Vel^2$$

Otro grupo representa la resistencia al avance por perfil y radio de curva:

Unidad: $\left(Pdte + \frac{800}{Curv} \right) \cdot \left(\frac{PesoMatMot}{100} \right) (kN)$

Tren: $\left(\left(Pdte + \frac{800}{Curv} \right) \cdot \left(\frac{PesoMatMot + pesoAdic}{100} \right) \right) (kN)$

Y un tercero que representa la resistencia al avance por material remolcado (viajeros/mercancías) [16].

$\left(2 + \frac{Vel^2}{4500} \right) \cdot \left(\frac{pesoAdic}{100} \right) (kN)$ Para tren de **viajeros**.

$\left(2 + \frac{Vel^2}{1600} \right) \cdot \left(\frac{pesoAdic}{100} \right) (kN)$ Para tren de **mercancías**.

5.2 Velocidad máxima

Las condiciones que debe cumplir el movimiento son los tiempos de llegada a la siguiente estación y la velocidad máxima permitida en cada tramo. La velocidad máxima será la menor de las tres siguientes:

1. La máxima del material motriz.
2. La máxima del conjunto formado por el material motriz más el material remolcado.
3. La máxima del trayecto.

Para la primera iteración se toma este valor y en función de los resultados obtenidos se admite como válida si el tiempo de llegada es inferior al programado en menos de 20 segundos. En caso contrario se va reduciendo hasta que se cumpla la condición anterior.

5.3 Aceleración de frenado media

Antes de comenzar el movimiento se calcula la aceleración de frenado media de la velocidad máxima del conjunto [17]. Para determinar la **aceleración de frenado media** de la circulación (Acelfm) se utiliza la velocidad máxima del conjunto, el tiempo de frenado, y la distancia total de tramo. El valor resultante se multiplica por 0.8 para disponer de margen.

El proceso para calcular la aceleración media es el siguiente:

Primero hallamos el tiempo de frenado para el instante actual. Para ello se obtiene el valor del esfuerzo de frenado. A partir del valor del esfuerzo de frenado y el peso del tren, podemos obtener la aceleración de frenado para esa velocidad:

$$AcelFren\ \left(\frac{m}{s^2}\right) = \frac{valorEsfFren}{pesoMatMovil + pesoAdicional}$$

Gracias a esto, y al tratarse de un movimiento uniformemente decelerado, se consigue la distancia avanzada en ese intervalo de tiempo (1 segundo):

$$Sparcial\ (s) = velControl \cdot t + \frac{1}{2}\ acelFren \cdot\ t^2\ (t = 1)$$

Por tanto, para obtener el valor de la velocidad tras haber frenado durante un segundo, tan sólo hay que hacer el siguiente cálculo:

$$velControl\ \left(\frac{m}{s}\right) = velControl + acelFren$$

Estos tres pasos se repetirán hasta que la velocidad sea igual a cero. De esta forma, sumando el número de iteraciones y las distancias avanzadas en cada intervalo, podemos obtener el tiempo y distancia de frenado.

A continuación hallamos dos aceleraciones medias de frenado:

- Teniendo en cuenta la velocidad máxima del conjunto

$$AcelMediaFrenUno\ \left(\frac{m}{s^2}\right) = \frac{vmaxConj}{tiempoFrenado}$$

- Teniendo en cuenta la distancia total del tramo

$$AcelMediaFrenDos\left(\frac{m}{s^2}\right) = \frac{(2 \cdot STotal)}{tiempoFrenado^2}$$

La aceleración media de frenado final será la menor de las dos.

5.4 Cálculo del esfuerzo

En cada instante hasta llegar a la estación final del tramo se puede dar diferentes situaciones y en cuestión de las mismas, se acelera, se mantiene la velocidad constante o se frena, hasta completar el recorrido. Las situaciones posibles son las siguientes:

1. Que se encuentre en un instante donde la velocidad es menor que la uniforme correspondiente a la iteración. En tal caso hay que acelerar, siendo la aceleración la correspondiente al esfuerzo acelerador máximo. Tal esfuerzo es la diferencia entre el esfuerzo motriz máximo menos la resistencia al avance.

$$Esfm(kN) - Rav(kN) = (PesoMatMot(t) + pesoAdic(t)) \cdot Acel(\frac{m}{s^2})$$

2. Que la velocidad en el instante sea la uniforme correspondiente a la iteración y la resistencia al avance sea menor que el esfuerzo máximo de tracción a dicha velocidad. En tal caso se circula con movimiento uniforme (a velocidad constante).

3. Cuando la velocidad es la uniforme correspondiente a la iteración, pero la resistencia al avance es mayor que el esfuerzo máximo de tracción, se frena con una aceleración correspondiente a la diferencia entre ambos valores aplicada a la masa total del tren.

$$Esfm(kN) - Rav(kN) = (PesoMatMot(t) + pesoAdic(t)) \cdot Acel(\frac{m}{s^2})$$

4. En el caso de que se encuentre en un instante donde la velocidad es mayor que la máxima, se frena utilizando la capacidad de frenado del tren, es decir el esfuerzo máximo de frenado (Esff (kN) negativo) obtenido de su curva de frenado y también la resistencia al avance correspondiente.

$$Esff(kN) - Rav(kN) = (PesoMatMot(t) + pesoAdic(t)) \cdot Acel(\frac{m}{s^2})$$

5. En el caso de que exista parada al final del tramo se determina en cada instante si tiene que comenzar a frenar para llegar parado a la estación final del mismo. Se comienza a frenar cuando la distancia hasta el final del tramo es menor que el siguiente valor:

$$Kmfrenada = \frac{-Vel^2}{Acelfm} + 0.5 \cdot \frac{Vel^2}{Acelfm}$$

Si se encuentra en la zona de frenado se actualiza la aceleración (negativa) en función de la velocidad y del espacio disponible hasta el final del tramo.

$$Acel(\frac{m}{s^2}) = -\frac{Vel^2}{2 \cdot s}$$

En cualquiera de las situaciones anteriores se calculan los valores para el segundo siguiente de la siguiente forma:

Con la aceleración se calcula cual es la velocidad un segundo después [17].

$$Acel\left(\frac{m}{s^2}\right) = \frac{Esff\ (kN) - Rav(kN)}{pesoMatMovil + pesoAdicional}$$

$$VelFinal\ \left(\frac{m}{s}\right) = Vel + Acel\ \cdot t\ (t = 1s)$$

Además, se calcula la potencia mediante la siguiente expresión [17]:

$$Potencia\ (kW) = \frac{Esf\ (kN) \cdot Vel\ (\frac{m}{s})}{rendimientoMotor} + potAuxMatMovil\ (kW)$$

El valor rendimientoMotor es un porcentaje que hace referencia al rendimiento mecánico de motor eléctrico y que permite obtener la potencia nominal eléctrica del mismo.

Con la posición en la que se encontraba la circulación, la velocidad y la aceleración, se actualizan la posición y la velocidad un segundo después. El algoritmo ha de tener en cuenta si hay cambio de vía, si se llega al final del tramo, y si se circula según kilómetros crecientes o decrecientes. En cada instante se archivan:

- Situación (vía y km)
- Velocidad
- Aceleración
- Potencia
- Esfuerzo

5.5 Obtención de resultados

Una vez que se llega a la estación, se comprueba el tiempo de llegada. Como ya se ha indicado, la primera iteración se realiza en las condiciones más rápidas posibles, es decir, con la máxima velocidad permitida. Si con esta velocidad se llega antes del tiempo marcado, se debe ir más despacio y la velocidad debe ser menor. Si llega en hora (dentro de la tolerancia) la velocidad es la buscada, y si llega tarde no es posible realizar ese recorrido.

Cuando llega antes se realizará una nueva iteración con las condiciones iniciales del tramo pero tomando como velocidad máxima una menor. Esto se repite hasta que llega en hora.

El proceso se realiza en todos los tramos hasta llegar a la estación final. Existe una limitación del número de iteraciones. Cuando se alcanza el número máximo de iteraciones se debe fundamentalmente a que el tren sale de la estación inicial del tramo a una velocidad tal que aún frenando al máximo no puede cumplir el horario (es necesario disminuirle el tiempo disponible).

5.6 Vía virtual

Para simplificar el uso del algoritmo y poder mostrar correctamente gráficas con los datos de la circulación, se crea una vía virtual. La vía virtual no es más que un mapeo de las vías por las que pasa el tren en una circulación, tal y como se puede ver en la figura 11:

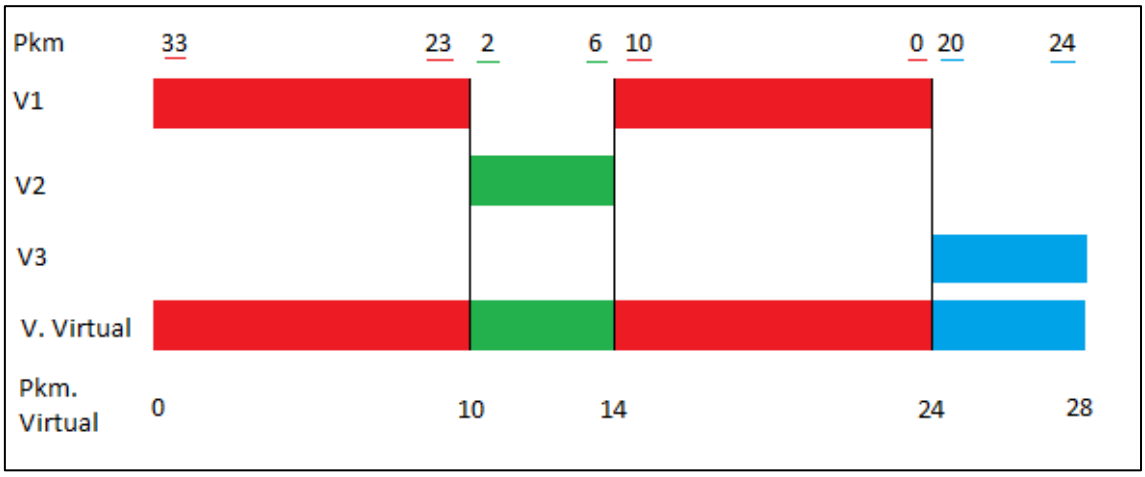


Figura 11. Esquema de vía virtual

La vía virtual nos permite, por un lado, ejecutar el algoritmo como si en la circulación solo hubiera una vía y, por otro, devolver los datos en función de los valores de vía y punto kilométrico reales.

CAPÍTULO 6

Análisis

Capítulo 6: Análisis

El principal objetivo de este capítulo es reflejar los aspectos que debemos tener en cuenta para el posterior diseño del software.

Su alcance comprende la definición de los casos de uso y de los requisitos de software. El análisis no abarcará toda la aplicación, sino que sólo tendrá en cuenta los elementos necesarios para el algoritmo de movimiento.

6.1 Casos de uso

Los casos de uso permiten definir una serie de interacciones entre el sistema y los **actores**, que en este caso serán **siempre usuarios**. Por tanto, el campo de actor será eliminado de los casos de uso.

Para cada caso de uso se usa la siguiente plantilla:

Caso de uso:	**Identificador:**
Objetivo:	
Precondiciones:	
Post-condiciones:	
Escenario:	
Escenario alternativo:	

Tabla 1. Plantilla de caso de uso

- **Caso de uso:** en este campo aparece el nombre único del caso de uso. Se trata de un nombre descriptivo y que permite localizar el caso de uso dentro del diagrama.
- **Identificador:** se trata de un identificador único del caso de uso. Viene dado por la expresión "CU-XX", donde XX es un número entre 01 y 99.
- **Objetivo:** explica el objetivo que persigue el caso de uso.
- **Precondiciones:** indica las condiciones que se han de cumplir para que se lleve a cabo la interacción entre la aplicación y el actor.
- **Post-condiciones:** muestra las condiciones del sistema tras la interacción entre la aplicación y el actor.
- **Escenario:** describe los pasos que ha de realizar el actor para cumplir la interacción descrita en el caso de uso.
- **Escenario alternativo:** si hay más formas de describir los pasos del actor, serán explicados en este apartado.

6.1.1 Diagrama de casos de uso

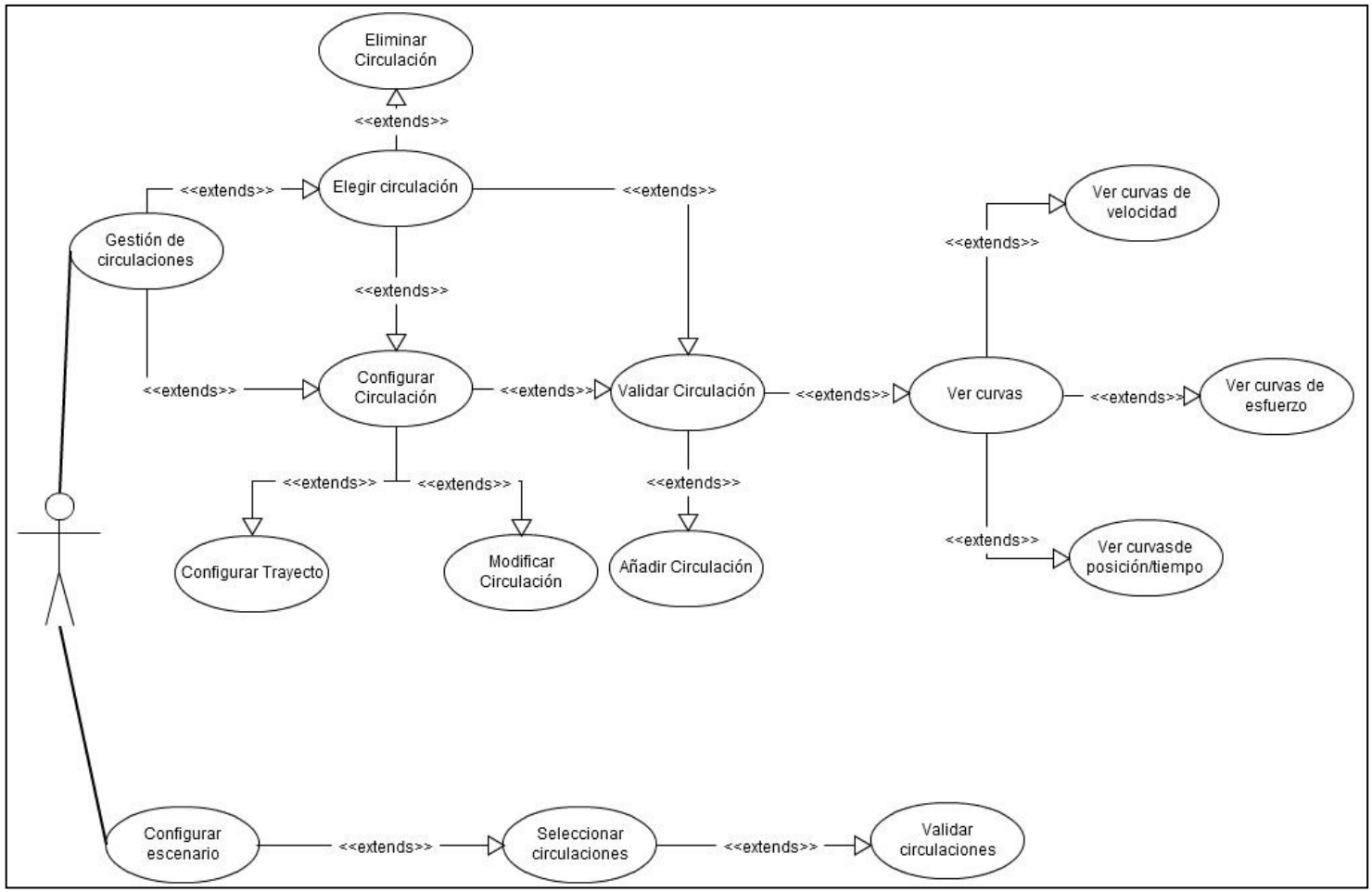


Figura 12. Diagrama de casos de uso

6.1.2 Descripción textual de los casos de uso

- **Casos de uso de Gestión de circulaciones**

Caso de uso: Elegir circulación	**Identificador:** CU-01
Objetivo: Seleccionar una circulación de entre una lista para visualizar sus datos o validar su movimiento con el algoritmo.	
Precondiciones: - Debe haberse creado o cargado un proyecto en la aplicación. - Debe haberse creado en el proyecto una circulación con anterioridad. - El usuario debe acceder a la interfaz de gestión de circulaciones.	
Post-condiciones: - El usuario puede ver información de la circulación seleccionada. - El usuario puede ejecutar el algoritmo de movimiento pasándole como entrada la circulación seleccionada.	
Escenario: - El usuario accede a un proyecto creado o crea uno nuevo. - El usuario accede a la interfaz de gestión de circulaciones. - El usuario selecciona una circulación de entre varias que aparecen en una lista.	
Escenario alternativo: No hay escenario alternativo.	

Tabla 2. Caso de uso CU-01

Caso de uso: Configurar circulación	**Identificador:** CU-02
Objetivo: Configurar los parámetros de una circulación.	
Precondiciones: - Debe haberse creado o cargado un proyecto en la aplicación. - Debe haber trayectos creados anteriormente en el proyecto. - Debe haber trenes creados anteriormente en el proyecto. - El usuario debe acceder a la interfaz de gestión de circulaciones.	
Post-condiciones: - Los parámetros de la circulación habrán cambiado. - Cambia la información que aparece en la interfaz de gestión de circulaciones.	
Escenario: - El usuario accede a un proyecto creado o crea uno nuevo. - El usuario accede a la interfaz de gestión de circulaciones. - El usuario realiza alguna de las siguientes acciones: • El usuario rellena el campo vacío asociado al nombre de la circulación. • El usuario selecciona un trayecto de entre varios que aparecen en una lista. • El usuario selecciona un tren de entre varios que aparecen en una lista.	
Escenario alternativo: - El usuario accede a un proyecto creado o crea uno nuevo.	

- El usuario accede a la interfaz de gestión de circulaciones. - El usuario selecciona una circulación de entre varias que aparecen en una lista. - El usuario realiza alguna de las siguientes acciones: • El usuario modifica el campo asociado al nombre de la circulación. • El usuario selecciona un trayecto, diferente al actual de la circulación, de entre varios que aparecen en una lista. • El usuario selecciona un tren, diferente al actual de la circulación, de entre varios que aparecen en una lista.

Tabla 3. Caso de uso CU-02

Caso de uso: Configurar trayecto	**Identificador:** CU-03
Objetivo: Configurar los parámetros de un trayecto.	
Precondiciones: - Debe haberse creado o cargado un proyecto en la aplicación. - El usuario debe acceder ala interfaz de gestión de circulaciones. - Debe haber un trayecto asociado a la circulación.	
Post-condiciones: - Se mostrarán en la interfaz de gestión de circulaciones los parámetros modificados de los trayectos.	
Escenario: - El usuario accede a un proyecto creado o crea uno nuevo. - El usuario accede a la interfaz de gestión de circulaciones. - El usuario selecciona un trayecto para la circulación a definir. - El usuario realiza alguna de las siguientes acciones: • El usuario añade o modifica el horario de una estación. • El usuario añade o modifica la velocidad de paso de una estación. • El usuario activa o desactiva la opción de parada en una estación.	
Escenario alternativo: - El usuario accede a un proyecto creado o crea uno nuevo. - El usuario accede a la interfaz de gestión de circulaciones. - El usuario selecciona una circulación de entre varias que aparecen en una lista. - El usuario realiza alguna de las siguientes acciones: • El usuario modifica el horario de una estación. • El usuario modifica la velocidad de paso de una estación. • El usuario activa o desactiva la opción de parada en una estación.	

Tabla 4. Caso de uso CU-03

Caso de uso: Validar circulación	**Identificador:** CU-04
Objetivo: Ejecutar el algoritmo de circulación pasándole como entrada una circulación creada o seleccionada por el usuario de entre las existentes.	

Precondiciones: - Debe haberse creado o cargado un proyecto en la aplicación. - El usuario debe acceder a la interfaz de gestión de circulaciones. - Debe haber una circulación creada en el proyecto. - La circulación seleccionada para ser validada debe tener todos los campos rellenados y sin errores de formato.
Post-condiciones: - Si la circulación ha sido validada por el algoritmo, pasará a tener el estado de validada. - El algoritmo ha generado un fichero log con toda la información que genera tras su ejecución. - Si la circulación ha sido validada por el algoritmo, se muestra una nueva interfaz al usuario para que pueda ver los datos resultantes de la ejecución del algoritmo.
Escenario: - El usuario accede a un proyecto creado o crea uno nuevo. - El usuario accede a la interfaz de gestión de circulaciones. - El usuario proporciona un nombre a la circulación. - El usuario selecciona un trayecto para la circulación. - El usuario selecciona un tren para la estación. - El usuario asigna horarios a las estaciones del trayecto. - El usuario asigna paradas a las estaciones del trayecto. - El usuario selecciona las velocidades de paso para las estaciones del trayecto. - El usuario selecciona la opción de validar.
Escenario alternativo: - El usuario accede a un proyecto creado o crea uno nuevo. - El usuario accede a la interfaz de gestión de circulaciones. - El usuario selecciona una circulación de entre varias que aparecen en una lista. - El usuario modifica, si lo desea, los datos de la circulación seleccionada que crea convenientes. - El usuario selecciona la opción de validar.

Tabla 5. Caso de uso CU-04

Caso de uso: Ver curvas de la circulación	**Identificador:** CU-05
Objetivo: Acceder a las gráficas con las curvas resultantes de la ejecución del algoritmo.	
Precondiciones: - Debe haberse creado o cargado un proyecto en la aplicación. - El usuario debe acceder a la interfaz de gestión de circulaciones. - El usuario debe haber definido o seleccionado una circulación. - El algoritmo de movimiento debe haber validado la circulación.	
Post-condiciones: - Se muestra una interfaz al usuario desde la cual acceder a las gráficas de la circulación.	
Escenario: - El usuario accede a un proyecto creado o crea uno nuevo.	

- El usuario accede a la interfaz de gestión de circulaciones. - El usuario define o selecciona una circulación. - El usuario selecciona la opción de validar. - El algoritmo valida la circulación. - Se muestra al usuario una interfaz desde la cual acceder a la visualización de las curvas. - El usuario selecciona la opción de "Ver Curvas".
Escenario alternativo: - El usuario accede a un proyecto creado o crea uno nuevo. - El usuario accede a la interfaz de gestión de circulaciones. - El usuario define o selecciona una circulación. - El usuario selecciona la opción de validar. - El algoritmo no valida la circulación. - Se muestra un mensaje de error al usuario y no puede ver las curvas.

Tabla 6. Caso de uso CU-05

Caso de uso: Ver curvas de velocidad	**Identificador:** CU-06
Objetivo: Ver las curvas de velocidad en función de la posición del tren y del tiempo de la circulación.	
Precondiciones: - Debe haberse creado o cargado un proyecto en la aplicación. - El usuario debe acceder a la interfaz de gestión de circulaciones. - El usuario debe haber seleccionado o definido una circulación. - El algoritmo de movimiento debe haber validado la circulación. - El usuario debe haber seleccionado la opción de ver curvas.	
Post-condiciones: - Aparece en pantalla una gráfica con las curvas de velocidad del tren.	
Escenario: - El usuario accede a un proyecto creado o crea uno nuevo. - El usuario accede a la interfaz de gestión de circulaciones. - El usuario define o selecciona una circulación. - El usuario selecciona la opción de validar. - El algoritmo valida la circulación. - Se muestra al usuario una interfaz desde la cual acceder a la visualización de las curvas. - El usuario selecciona la opción de "Ver Curvas". - El usuario elige entre ver las curvas en función de la posición del tren, o en función del tiempo de la circulación.	
Escenario alternativo: No hay escenario alternativo.	

Tabla 7. Caso de uso CU-06

Caso de uso: Ver curvas de esfuerzo	**Identificador:** CU-07
Objetivo: Ver las curvas de esfuerzo en función de la posición del tren y del tiempo de la circulación.	
Precondiciones:	

- Debe haberse creado o cargado un proyecto en la aplicación. - El usuario debe acceder a la interfaz de gestión de circulaciones. - El usuario debe haber seleccionado una circulación. - El algoritmo de movimiento debe haber validado la circulación. - El usuario debe haber seleccionado la opción de ver curvas.
Post-condiciones: - Aparece en pantalla una gráfica con las curvas de esfuerzo del tren.
Escenario: - El usuario accede a un proyecto creado o crea uno nuevo. - El usuario accede a la interfaz de gestión de circulaciones. - El usuario define o selecciona una circulación. - El usuario selecciona la opción de validar. - El algoritmo valida la circulación. - Se muestra al usuario una interfaz desde la cual acceder a la visualización de las curvas. - El usuario selecciona la opción de "Ver Curvas". - El usuario selecciona una de las dos pestañas asociadas a las curvas de esfuerzo.
Escenario alternativo: No hay escenario alternativo.

Tabla 8. Caso de uso CU-07

Caso de uso: Ver curvas de posición/tiempo	**Identificador:** CU-08
Objetivo: Ver las curvas de posición del tren en función del tiempo de circulación y viceversa.	
Precondiciones: - Debe haberse creado o cargado un proyecto en la aplicación. - El usuario debe acceder a la interfaz de gestión de circulaciones. - El usuario debe haber seleccionado una circulación. - El algoritmo de movimiento debe haber validado la circulación. - El usuario debe haber seleccionado la opción de ver curvas.	
Post-condiciones: - Aparece en pantalla una gráfica con las curvas de posición/tiempo del tren.	
Escenario: - El usuario accede a un proyecto creado o crea uno nuevo. - El usuario accede a la interfaz de gestión de circulaciones. - El usuario define o selecciona una circulación. - El usuario selecciona la opción de validar. - El algoritmo valida la circulación. - Se muestra al usuario una interfaz desde la cual acceder a la visualización de las curvas. - El usuario selecciona la opción de "Ver Curvas". - El usuario elige entre ver la curva de posición en función del tiempo, o ver la curva de tiempo en función de la posición.	
Escenario alternativo: No hay escenario alternativo.	

Tabla 9. Caso de uso CU-08

- **Casos de uso de Edición BBDD**

Caso de uso: Añadir circulación	**Identificador:** CU-09
Objetivo: Añadir una circulación a la base de datos del proyecto	
Precondiciones: - Debe haberse creado o cargado un proyecto en la aplicación. - El usuario debe haber accedido a la interfaz de circulación. - El usuario debe haber rellenado correctamente todos los campos de la circulación.	
Post-condiciones: - Se ha añadido una circulación a la base de datos del proyecto.	
Escenario: - El usuario accede a un proyecto creado o crea uno nuevo. - El usuario accede a la interfaz de gestión de circulaciones. - El usuario rellena correctamente todos los campos de la circulación. - El usuario selecciona la opción de añadir una circulación a la base de datos.	
Escenario alternativo: - El usuario accede a un proyecto creado o crea uno nuevo. - El usuario accede a la interfaz de gestión de circulaciones. - El usuario rellena de forma incorrecta todos los campos de la circulación. - El usuario selecciona la opción de añadir una circulación a la base de datos. - Se muestra un mensaje de error al usuario y la circulación no es añadida.	

Tabla 10. Caso de uso CU-09

Caso de uso: Modificar circulación	**Identificador:** CU-10
Objetivo: Modificar los datos de una circulación guardada en la base de datos del proyecto.	
Precondiciones: - Debe haberse creado o cargado un proyecto en la aplicación. - El usuario debe haber accedido a la interfaz de circulación. - Debe haber creada al menos una circulación en el proyecto. - El usuario debe haber seleccionado una circulación.	
Post-condiciones: - La circulación queda modificada en la base de datos.	
Escenario: - El usuario accede a un proyecto creado o crea uno nuevo. - El usuario accede a la interfaz de gestión de circulaciones. - El usuario selecciona una circulación. - El usuario modifica correctamente algún campo de la circulación. - El usuario selecciona la opción de modificar una circulación de la base de datos.	
Escenario alternativo: - El usuario accede a un proyecto creado o crea uno nuevo. - El usuario accede a la interfaz de gestión de circulaciones.	

- El usuario selecciona una circulación. - El usuario rellena de forma incorrecta algún campo de la circulación. - El usuario pulsa selecciona la opción de modificar una circulación de la base de datos. - Se muestra un mensaje de error al usuario y la circulación no es modificada.

Tabla 11. Caso de uso CU-10

Caso de uso: Eliminar circulación	**Identificador:** CU-11
Objetivo: Borrar los datos de una circulación guardada en la base de datos del proyecto.	
Precondiciones: - Debe haberse creado o cargado un proyecto en la aplicación. - El usuario debe haber accedido a la interfaz de circulación. - Debe haber creada al menos una circulación en el proyecto. - El usuario debe haber seleccionado una circulación.	
Post-condiciones: - La circulación queda eliminada de la base de datos.	
Escenario: - El usuario accede a un proyecto creado o crea uno nuevo. - El usuario accede a la interfaz de gestión de circulaciones. - El usuario selecciona una circulación. - El usuario modifica correctamente todos los campos de la circulación. - El usuario selecciona la opción de eliminar una circulación de la base de datos.	
Escenario alternativo: No hay escenario alternativo.	

Tabla 12. Caso de uso CU-11

- **Casos de uso de Definir escenario**

Caso de uso: Seleccionar circulaciones	**Identificador:** CU-12
Objetivo: Seleccionar una o varias circulaciones para su validación posterior.	
Precondiciones: - Debe haberse creado o cargado un proyecto en la aplicación. - El usuario debe haber accedido a la interfaz de definición de escenarios del proyecto. - Debe haber creada al menos una circulación en el proyecto.	
Post-condiciones: - Se muestra una serie de circulaciones seleccionadas.	
Escenario: - El usuario accede a un proyecto creado o crea uno nuevo. - El usuario accede a la interfaz de definición de escenarios del proyecto. - El usuario selecciona una serie de circulaciones.	
Escenario alternativo: No hay escenario alternativo.	

Tabla 13. Caso de uso CU-12

Caso de uso: Validar circulaciones	**Identificador:** CU-13
Objetivo: Validar una o varias circulaciones seleccionadas previamente.	
Precondiciones: - Debe haberse creado o cargado un proyecto en la aplicación. - El usuario debe haber accedido a la interfaz de definición de escenarios del proyecto. - Debe haber creada al menos una circulación en el proyecto. - Debe haber al menos una circulación seleccionada en la interfaz de definición de escenarios del proyecto.	
Post-condiciones: - Si la circulación ha sido validada por el algoritmo, pasará a tener el estado de validada. - El algoritmo ha generado, para cada circulación, un fichero log con toda la información de que genera tras su ejecución. - Se muestra un mensaje con el número de circulaciones validadas.	
Escenario: - El usuario accede a un proyecto creado o crea uno nuevo. - El usuario accede a la interfaz de definición de escenarios del proyecto. - El usuario selecciona una serie de circulaciones. - El usuario selecciona la opción de validar.	
Escenario alternativo: No hay escenario alternativo.	

Tabla 14. Caso de uso CU-13

6.2 Requisitos software

En este apartado se hará una descripción completa del comportamiento del sistema que se va a desarrollar. Usaremos los siguientes tipos de requisitos:

- **Requisitos funcionales**: establecen los comportamientos del sistema. Derivan de los casos de uso.
- **Requisitos de operación**: indican cómo trabaja el sistema internamente a la hora de realizar las diversas tareas.
- **Requisitos de comprobación**: muestran las necesidades que se han de cumplir, o las comprobaciones necesarias que se deben realizar para poder hacer frente a ciertas tareas.

Para este proyecto, se considerarán sólo aquellos requisitos que afectan de manera más directa al algoritmo de movimiento.

Al definir cada requisito se seguirá el formato de la siguiente plantilla:

Identificador			
Descripción			
Prioridad	☐ Alta	☐ Media	☐ Baja
Necesidad	☐ Alta	☐ Media	☐ Baja
Verificabilidad	☐ Alta	☐ Media	☐ Baja
Estabilidad			

Tabla 15. Plantilla de requisito

- **Descripción:** en este apartado se explica en qué consiste el requisito**.**
- **Identificador:** identifica de forma única al requisito. Dependiendo del tipo de requisito su formato variará:
    - **Requisitos funcionales**: su identificador será RF-XX, donde XX es un número entre 00 y 99.
    - **Requisitos de operación**: su identificador será RO-XX, donde XX es un número entre 00 y 99.
    - **Requisitos de comprobación**: su identificador será RC-XX, donde XX es un número entre 00 y 99.
- **Prioridad:** indica en qué parte de la construcción del sistema se incluirá la funcionalidad.
    - **Alta**: el requisito ha de tener preferencia a la hora de ser llevado a cabo.
    - **Media**: el requisito es importante, pero no es el más prioritario.
    - **Baja**: el requisito no es prioritario, será implementado cuando los demás ya estén cumplidos.
- **Necesidad**: hace referencia al interés de los usuarios en que la aplicación lo cumpla. Refleja hasta qué punto estarían dispuestos a eliminarlo de la aplicación.

  - **Alta**: el requisito debe ser cumplido.
  - **Media**: el requisito no es obligatorio, pero su cumplimiento mejoraría la calidad de la aplicación.
  - **Baja**: el requisito puede ser omitido sin problema alguno.

- **Verificabilidad**: indica la facilidad de comprobar que un requisito ha sido incorporado en el diseño.

- **Estabilidad**: indica si el requisito puede cambiar a lo largo del ciclo de vida del proyecto o si permanece estable.

6.2.1 Requisitos funcionales

RF-01			
Descripción	El sistema mostrará al usuario una lista con todas las circulaciones guardadas en la base de datos del proyecto.		
Prioridad	☑ Alta	☐ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Durante toda la vida del sistema.		

Tabla 16. Requisito RF-01

RF-02			
Descripción	El usuario podrá seleccionar una circulación de la lista de circulaciones mostradas por el sistema.		
Prioridad	☑ Alta	☐ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Durante toda la vida del sistema.		

Tabla 17. Requisito RF-02

RF-03			
Descripción	Cuando el usuario selecciona una circulación, el sistema le mostrará por pantalla la siguiente información: • Nombre de la circulación. • Trayecto de la circulación. • Tren de la circulación. • Horarios de la circulación. • Paradas de la circulación. • Velocidades máximas de los tramos de la circulación.		
Prioridad	☑Alta	☐ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja

Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Variable en cuanto a los campos que se muestran.		

Tabla 18. Requisito RF-03

RF-04			
Descripción	El usuario podrá definir el nombre de una circulación elegida.		
Prioridad	☑Alta	☐ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Durante toda la vida del sistema.		

Tabla 19. Requisito RF-04

RF-05			
Descripción	El sistema mostrará al usuario una lista de trayectos para elegir.		
Prioridad	☑Alta	☐ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Durante toda la vida del sistema.		

Tabla 20. Requisito RF-05

RF-06			
Descripción	El usuario podrá elegir un trayecto para la circulación.		
Prioridad	☑ Alta	☐ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Durante toda la vida del sistema.		

Tabla 21. Requisito RF-06

RF-07			
Descripción	El sistema mostrará al usuario una lista de trenes para elegir.		
Prioridad	☑Alta	☐ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Durante toda la vida del sistema.		

Tabla 22. Requisito RF-07

RF-08			
Descripción	El usuario podrá elegir un tren para la circulación.		
Prioridad	☑ Alta	☐ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Durante toda la vida del sistema.		

Tabla 23. Requisito RF-08

RF-09			
Descripción	El usuario podrá asignar horarios a las estaciones de la circulación.		
Prioridad	☑ Alta	☐ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Durante toda la vida del sistema.		

Tabla 24. Requisito RF-09

RF-10			
Descripción	El sistema notificará al usuario cuando un horario se ha introducido con un formato incorrecto.		
Prioridad	☐ Alta	☑ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Durante toda la vida del sistema.		

Tabla 25. Requisito RF-10

RF-11			
Descripción	El usuario podrá elegir la velocidad de paso por la estación.		
Prioridad	☐ Alta	☑ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Durante toda la vida del sistema.		

Tabla 26. Requisito RF-11

RF-12			
Descripción	El sistema notificará al usuario cuando la velocidad de paso por la estación no sea un número.		
Prioridad	☐ Alta	☐ Media	☑Baja
Necesidad	☑Alta	☐ Media	☐ Baja
Verificabilidad	☑Alta	☐ Media	☐ Baja
Estabilidad	Durante toda la vida del sistema.		

Tabla 27. Requisito RF-12

RF-13			
Descripción	El usuario podrá elegir si una estación del trayecto tiene parada o no.		
Prioridad	☑Alta	☐ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Durante toda la vida del sistema.		

Tabla 28. Requisito RF-13

RF-14			
Descripción	El usuario podrá validar una circulación. Es decir, se ejecutará el algoritmo de movimiento pasándole como parámetro de entrada la circulación que se quiere validar.		
Prioridad	☑ Alta	☐ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Durante toda la vida del sistema.		

Tabla 29. Requisito RF-14

RF-15			
Descripción	El sistema generará un fichero con los datos que devuelve el algoritmo. Ese fichero contiene las siguientes columnas de datos: - Segundo inicial. - Segundo final. - Velocidad inicial. - Velocidad final. - Aceleración. - Vía inicial. - Vía final. - Punto kilométrico inicial. - Punto kilométrico final. - Variación de la posición. - Kilómetros recorridos. - Esfuerzo máximo. - Resistencia al avance. - Esfuerzo. - Potencia. - Pendiente de la vía. - Curvatura de la vía. - Esfuerzo de frenado. Habrá una fila con estos campos para cada segundo de avance del tren en la simulación.		
Prioridad	☑ Alta	☐ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Variable en cuanto a los valores que se devuelven.		

Tabla 30. Requisito RF-15

RF-16			
Descripción	El sistema generará un fichero de registro con todos los datos que genera el algoritmo, incluyendo aquellos que son descartados.		
Prioridad	☐ Alta	☐ Media	☑ Baja

Necesidad	☐ Alta	☑ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Variable tanto en el conjunto de datos que contiene como en su desaparición del proyecto.		

Tabla 31. Requisito RF-16

RF-17			
Descripción	El sistema permite al usuario ver las curvas que representan la información devuelta por el algoritmo.		
Prioridad	☐ Alta	☑ Media	☐ Baja
Necesidad	☐ Alta	☑ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Variable en la forma de representar los datos.		

Tabla 32. Requisito RF-17

RF-18			
Descripción	Tras validar la circulación, el usuario puede ver las curvas de velocidad del tren en función de su posición kilométrica y del tiempo de la circulación.		
Prioridad	☐ Alta	☑ Media	☐ Baja
Necesidad	☐ Alta	☑ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Variable en la forma de representar los datos.		

Tabla 33. Requisito RF-18

RF-19			
Descripción	Tras validar la circulación, el usuario puede ver las curvas de esfuerzo del tren en función de su posición kilométrica y del tiempo de la circulación.		
Prioridad	☐ Alta	☑ Media	☐ Baja
Necesidad	☐ Alta	☑ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Variable en la forma de representar los datos.		

Tabla 34. Requisito RF-19

RF-20			
Descripción	Tras validar la circulación, el usuario puede ver las curvas de posición del tren en función del tiempo de la circulación.		
Prioridad	☐ Alta	☑ Media	☐ Baja
Necesidad	☐ Alta	☑ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Variable en la forma de representar los datos.		

Tabla 35. Requisito RF-20

RF-21			
Descripción	Tras validar la circulación, el usuario puede ver las curvas de tiempo del tren en función de su posición kilométrica.		
Prioridad	☐ Alta	☑Media	☐ Baja
Necesidad	☐ Alta	☑ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Variable en la forma de representar los datos.		

Tabla 36. Requisito RF-21

RF-22			
Descripción	El usuario puede borrar una circulación de la base de datos del proyecto.		
Prioridad	☑ Alta	☐ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Durante toda la vida del sistema.		

Tabla 37. Requisito RF-22

RF-23			
Descripción	El sistema mostrará al usuario una lista de circulaciones para que una o varias puedan ser elegidas.		
Prioridad	☑ Alta	☐ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Durante toda la vida del sistema.		

Tabla 38. Requisito RF-23

RF-24			
Descripción	El usuario podrá elegir una o varias circulaciones para que puedan ser validadas a la vez.		
Prioridad	☑ Alta	☐ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Durante toda la vida del sistema.		

Tabla 39. Requisito RF-24

RF-25			
Descripción	El sistema puede validar varias circulaciones a la vez.		
Prioridad	☑ Alta	☐ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja
Verificabilidad	☑Alta	☐ Media	☐ Baja
Estabilidad	Durante toda la vida del sistema.		

Tabla 40. Requisito RF-25

6.2.2 Requisitos de operación

RO-01			
Descripción	Una circulación sólo puede ser validada si los campos: • Nombre de la circulación. • Trayecto de la circulación. • Tren de la circulación. • Horarios de la circulación. • Paradas de la circulación. • Velocidades máximas de los tramos de la circulación. Están correctamente cumplimentados.		
Prioridad	☑ Alta	☐ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Durante toda la vida del sistema.		

Tabla 41. Requisito R0-01

RO-02			
Descripción	El usuario puede añadir una circulación a la base de datos del proyecto si los siguientes campos están rellenados de forma correcta: • Nombre de la circulación. • Trayecto de la circulación. • Tren de la circulación. • Horarios de la circulación. • Paradas de la circulación. • Velocidades máximas de los tramos de la circulación.		
Prioridad	☑ Alta	☐ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Durante toda la vida del sistema.		

Tabla 42. Requisito R0-02

RO-03			
Descripción	El usuario puede modificar una circulación de la base de datos del proyecto si los siguientes campos están rellenados de forma correcta: • Nombre de la circulación. • Trayecto de la circulación. • Tren de la circulación. • Horarios de la circulación. • Paradas de la circulación. • Velocidades máximas de los tramos de la circulación.		
Prioridad	☑ Alta	☐ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Durante toda la vida del sistema.		

Tabla 43. Requisito R0-03

6.2.3 Requisitos de comprobación

RC-01			
Descripción	Los horarios de las estaciones han de cumplir el formato "dd:hh:mm:ss".		
Prioridad	☑Alta	☐ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Durante toda la vida del sistema.		

Tabla 44. Requisito RC-01

RC-02			
Descripción	La velocidad de paso por la estación sólo puede ser un número.		
Prioridad	☐ Alta	☑ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Durante toda la vida del sistema.		

Tabla 45. Requisito RC-02

RC-03			
Descripción	La aplicación debe ser implementada en lenguaje de programación C#.		
Prioridad	☐ Alta	☑ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja

Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Durante toda la vida del sistema.		

Tabla 46. Requisito RC-03

RC-04			
Descripción	La aplicación hará uso de datos reales de infraestructura y circulaciones, procedentes de la base de datos de ADIF.		
Prioridad	☐ Alta	☑ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Durante toda la vida del sistema.		

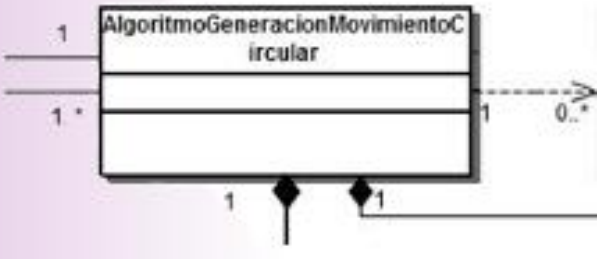
Tabla 47. Requisito RC-04

RC-05			
Descripción	La validación de varias circulaciones a la vez debe ser realizada mediante computación en paralelo, no de manera secuencial.		
Prioridad	☐ Alta	☑ Media	☐ Baja
Necesidad	☑ Alta	☐ Media	☐ Baja
Verificabilidad	☑ Alta	☐ Media	☐ Baja
Estabilidad	Durante toda la vida del sistema.		

Tabla 48. Requisito RC-05

CAPÍTULO 7

Diseño



Capítulo 7: Diseño

El propósito de este capítulo es detallar con la mayor precisión posible las características del sistema para facilitar su desarrollo por parte del equipo de trabajo. En él se detallará todo lo necesario para desarrollar el sistema en el aspecto lógico (modelo de clases, modelo de datos, etc.).

Se hará hincapié en el diseño de los módulos asociados a la validación y creación de circulaciones, pues el proyecto actual forma parte de una aplicación mayor. Por tanto, todo módulo de la aplicación que no esté relacionado con las circulaciones no será tenido en cuenta.

Por último se detallan las clases y la base de datos del proyecto.

7.1 Definición de la arquitectura del sistema

En este apartado se va a explicar cómo va a ser la arquitectura del sistema a desarrollar, es decir, qué tipo de software se realizará para alcanzar los diferentes objetivos.

Se ha optado por una arquitectura Modelo Vista Contolador (MVC) acorde al siguiente esquema:

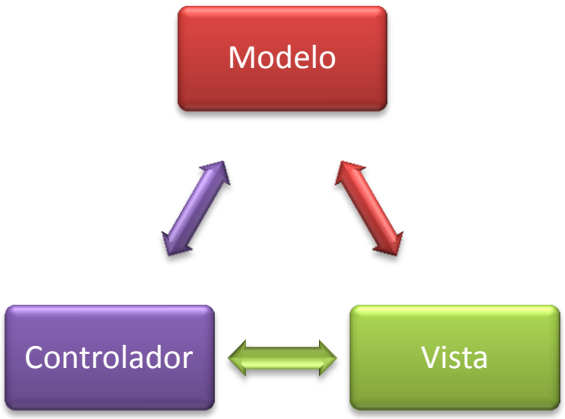


Figura 13. Modelo vista controlador

- **Modelo**: es la representación de los objetos del dominio o estructuras de datos con los que el sistema trabaja.
- **Vista**: presenta el modelo de manera que el usuario pueda interactuar con él. Normalmente se suele representar por medio de la interfaz de usuario.
- **Controlador**: traduce los datos de entrada del usuario en operaciones para el modelo. Como resultado de su ejecución ocurren cambios en el modelo y en la vista.

7.2 Identificación de subsistemas de diseño

El diseño de la parte del sistema relacionada con el proyecto se ha dividido en tres subsistemas organizados como se muestra en la figura 14:

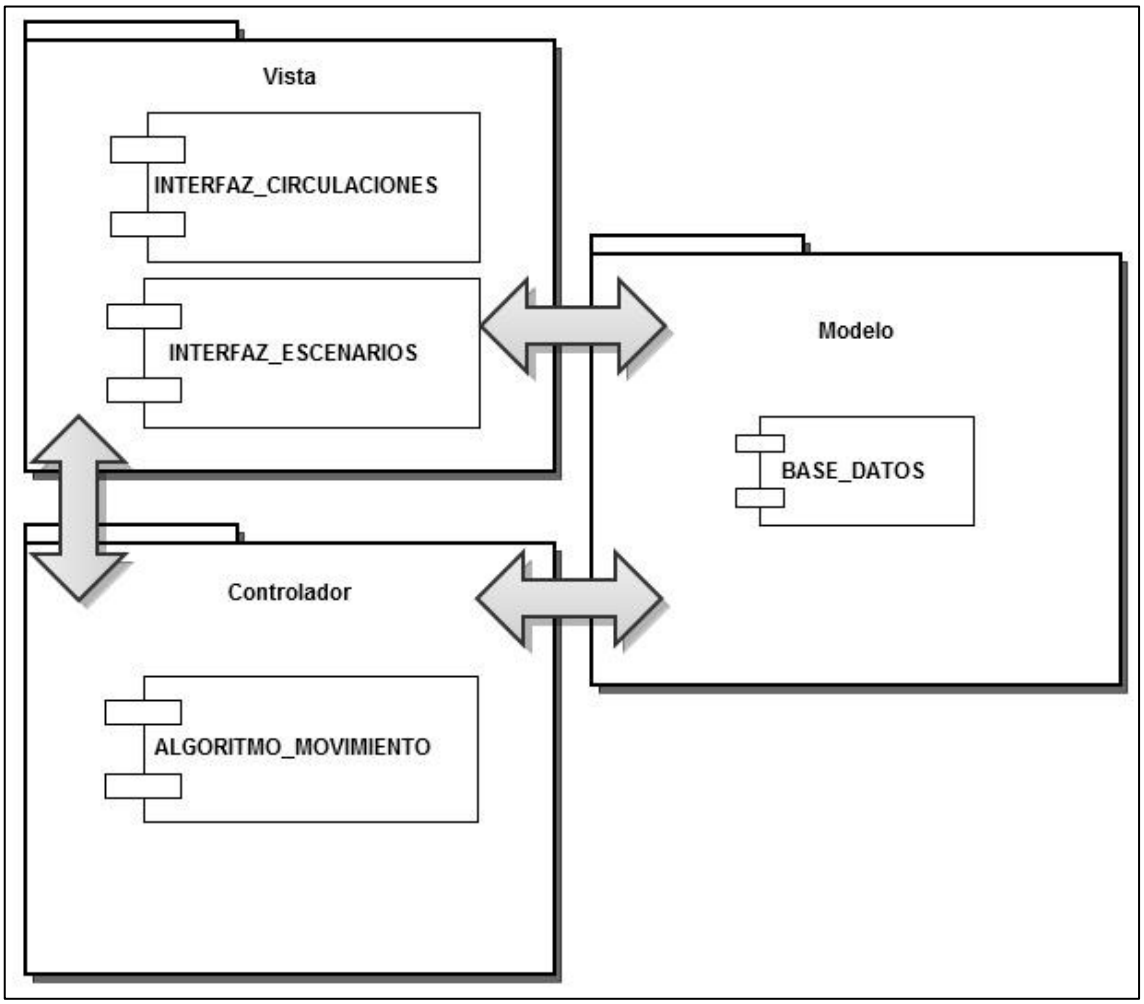


Figura 14. Diagrama de componentes

- **Vista**: está formado por los módulos INTERFAZ_CIRCULACIONES e INTERFAZ_ESCENARIOS. Como su nombre indica, se trata de las interfaces encargadas de recoger los datos del usuario para pasárselos al controlador y así poder validar las circulaciones con el algoritmo de movimiento. Una vez se han procesado los datos por parte del controlador, el subsistema vista recibirá la respuesta y la mostrará al usuario.
- **Controlador**: procesa la información que recibe de la vista y del modelo. El módulo ALGORITMO_MOVIMIENTO se encargará de convertir los datos de la vista y del modelo de forma que sean útiles para el algoritmo de movimiento. Luego procesará todos estos datos proporcionados produciendo otros de salida para el subsistema de vista.
- **Modelo**: permite la obtención de la información guardada en la base de datos. Además ofrece los métodos necesarios para su modificación.

7.3 Diseño detallado

7.3.1 Diagrama de clases

El diagrama de clases que se muestra a continuación describe la estructura del proyecto en la aplicación. Muestra sus clases, atributos y las relaciones que hay entre ellos.

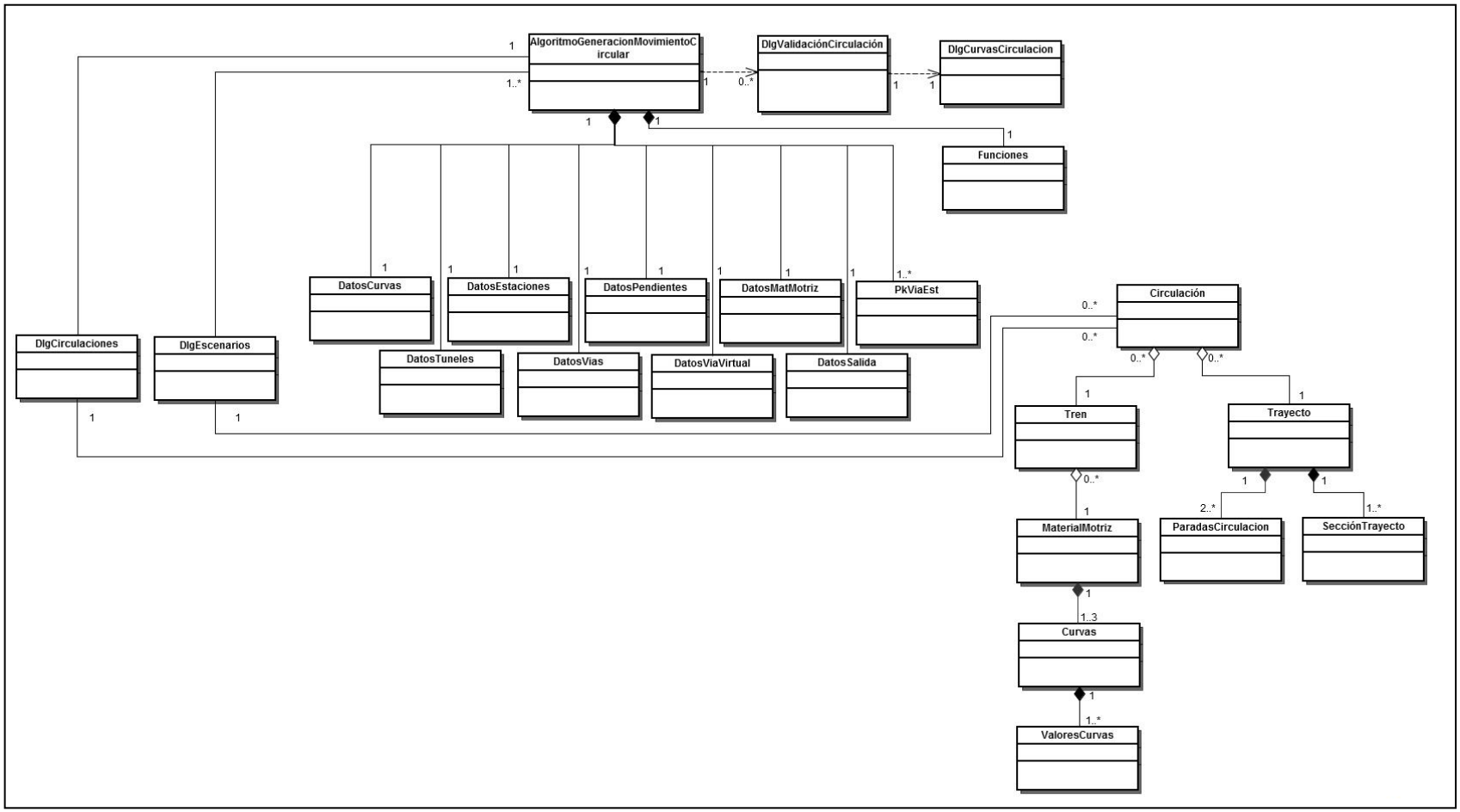


Figura 15. Diagrama de clases

7.3.2 Descripción detallada de las clases

Clase DlgCirculaciones

DlgCirculaciones
-buttonAdd: ToolStripButton **-buttonSalir:** ToolStripButton **-buttonDel:** ToolStripButton **-buttonMod:** ToolStripButton **-buttonValidarCirculacion:** Button **-dataGridViewParadas:** DataGridView **-toolStripProgressBar:** ToolStripProgressBar **-backgroundWorkerValidacion:** BackgroundWorker **-comboBoxTren:** ComboBox **-comboBoxTrayecto:** ComboBox **-textBoxNombre:** TextBox **-listView_codigo:** ListView **-proyecto:** Proyecto **-circulacionValidada**: bool **-horaCirculacionValidad:** salida **-salida**: List<DatosSalida>
-listView_codigo_selectedIndexChanged(entrada sender: object, entrada e: EventArgs): void **-comboBoxTrayecto_SelectedIndexChanged**(entrada sender: object, entrada e: EventArgs): void **-textBoxNombre_TextChanged**(entrada sender: object, entrada e: EventArgs): void **-comboBoxTren_SelectedIndexChanged**(entrada sender: object, entrada e: EventArgs): void **-comboBoxTipoCirculacion_SelectedIndexChanged**(entrada sender: object, entrada e: EventArgs): void **-RefrescarListView()**: void −**RefrescarComboTrenes()**: void **-RefrescarComboTrayectos()**: void **-RellenarDataGridView**(entrada trayecto: TrayectoRow): void **-RellenarDataGridView**(entrada circulacion: CirculacionRow): void **-RellenarDataGridView()**: void **-RefrescarLabelValidacion**(entrada validado: bool , entrada horaValidacion: DateTime): void **-buttonAdd_Click**(entrada sender: object, entrada e: EventArgs): void **-buttonDel_Click**(entrada sender: object, entrada e: EventArgs): void **-EliminacionSeleccionado**(): void **-buttonMod_Click**(entrada sender: object, entrada e: EventArgs): void **-buttonAtras_Click**(entrada sender: object, entrada e: EventArgs): void **-buttonSalir_Click**(entrada sender: object, entrada e: EventArgs): void **-CerrandoFormulario**(entrada sender: object, entrada e: EventArgs): void **-Saliendo()**: void

-**VerificarCamposFormulario**(): bool
-**VerificarCamposBasicos**(): bool
-**VerificarTablaHoras**(): bool
-**VerificarTablaParadas**(): bool
-**VerificarCirculacionValidada()**: bool
-**dataGridViewParadas_CellValidated**(entrada sender: object, entrada e: DataGridViewCellEventArgs): void
-**dataGridViewParadas_CellValueChanged**(entrada sender: object, entrada e: DataGridViewParadasEventArgs): void
-**InmovilizarFormulario**(entrada inmovilizar: bool): void
-**buttonValidarCirculacion_Click**(entrada sender: object, entrada e: EventArgs): void
-**FilaIsComplete**(entrada fila: DataGridViewRow): bool
-**FilaIsVacia**(entrada fila: DataGridViewRow): bool
-**backgroundWorkerValidacion_DoWork**(entrada sender: object, entrada e: DoWorkEventArgs): void
-**backgroundWorkerValidacion_ProgressChanged**(entrada sender: object, entrada e: ProgressChangedEventArgs): void
-**backgroundWorkerValidacion_RunWorkerCompleted**(entrada sender: object, entrada e: RunWorkerCompletedEventArgs): void

Figura 16. Clase DlgCirculaciones

- **Nombre**: DlgCirculaciones.
- **Subsistema**: Interfaz_Circulaciones.
- **Descripción**: se trata de una clase sobre la que se implementa una interfaz de usuario. Desde la interfaz el usuario podrá administrar y validar las circulaciones. Con este fin, presenta tres botones que permiten añadir, modificar y borrar una circulación.

  Mostrará una lista de circulaciones por medio de un elemento listView. De esta forma se podrá elegir una circulación de la lista para que sus datos sean cargados en la interfaz.

  Gracias a dos elementos comboBox, se podrá visualizar varios trenes y trayectos y elegir uno de cada. En un campo de texto el usuario podrá introducir o ver el nombre de una circulación.

  La información del trayecto y horarios de la circulación se visualizará en un objeto del tipo dataGridView, que es una tabla que permite modificar sus contenidos.

  Por último, un botón nos permitirá validar la circulación escogida. Es decir, pasarle los datos al algoritmo de movimiento para su ejecución. La ejecución se realizará de forma asíncrona gracias al uso de un componente BackgroundWorker. El componente BackgroundWorker lanza un subproceso distinto al subproceso de la interfaz de usuario. Así esta no se bloqueará mientras el algoritmo es ejecutado. Por último, el progreso de la ejecución podrá ser visualizado por medio de una barra de progreso.

Clase DlgEscenarios

DlgEscenarios
-checkedListBoxCirculaciones: CheckedListBox **-buttonCalcularCancelarEscenario:** Button -**proyecto**: Proyecto -**instanteInicial:** DateTime -**instanteFinal:** DateTime -**mutexThreads:** Mutex -**eventoHuecoNuevoThread:** AutoResetEvent -**numThreadsSinTerminar**: int -**numeroMaximoHilos**: int -**backgroundWorkerValidacionCirculaciones**: BackgroundWorker
-**refrescarCirculaciones():** void -**checkBoxCirculaciones_CheckedChanged**(entrada sender: object, entrada e: DoWorkEventArgs): void -**CalcularEscenario_Click**(entrada sender: object, entrada e: EventArgs): void -**backgroundWorkerValidacionCirculaciones_DoWorkParalelo**(entrada sender: object, entrada e: ProgressChangedEventArgs): void -**backgroundWorkerValidacionCirculaciones_ProgressChanged**(entrada sender: object, entrada e: RunWorkerCompletedEventArgs): void -**backgroundWorkerValidacionCirculaciones_RunWorkerCompleted**(entrada sender: object, entrada e: EventArgs): void

Figura 17. Clase DlgEscenarios

- **Nombre**: DlgEscenarios.
- **Subsistema**: Interfaz_Escenarios.
- **Descripción**: DlgEscenarios es una interfaz de usuario destinada, en el caso del presente proyecto, a la validación de varias circulaciones a la vez.

  Presenta el elemento checkedListBoxCirculaciones sobre el que se mostrará un listado con todas las circulaciones presentes en la base de datos. En ese listado se podrán seleccionar las circulaciones que se pueden validar.

  La validación de varias circulaciones a la vez debe estar optimizada para sistemas multicore. Como en DlgCirculaciones, se usa un elemento BackgroundWorker, para evitar el bloqueo de la interfaz. Desde el BackgroundWorker se lanzará la validación de las circulaciones. Pero antes, el código cuenta el número de procesadores que tiene el ordenador. Ese será el máximo de hilos posibles que lanzará la aplicación. Cada hilo tendrá asociado la ejecución de una circulación. Por tanto, en un procesador con cuatro núcleos, se podrán ejecutar hasta cuatro circulaciones a la vez.

Clase AlgoritmoGeneraciónMovimientoCircular

AlgoritmoGeneraciónMovimientoCircular
-**_datosReales**: PkViaEst -**_listaVias**: List<DatosVias> -**_listaViaVirtual**: List<DatosViaVirtual> -**_structCurvas**: List<DatosCurvas> -**_structEstaciones**: DatosEstaciones -**_structEstacionesVirtual**: DatosEstaciones -**_structMatMotriz**: DatosMatMotriz -**_structPendientes**: DatosPendientes -**_structSalidaMovimiento**: DatosSalida -**_structSalidaMovimientoDefinitivo**: DatosSalida -**_structTuneles**: DatosTuneles -**_structVelocLim**: DatosVelocLim -**_func**: Funciones
-**CalcularTiempoFrenado** (entrada Stotal: ref double, entrada velControl: double) : double -**CalcularAcelMediaFren** (entrada Stotal: double, entrada tiempoFrenado: double):double -**CalculoLongitudEntreEstaciones**(entrada estK: int): double -**CheckValoresTunelPendCurvaCorrectos**(entrada salidaError: ref String, entrada coefTuelCorrecto: bool, entrada pendienteCorrecta: bool, entrada curvaCorrecta: bool, entrada CurvatWh: double, entrada PdteWh: double, entrada TunelWh: double): Boolean. -**FuncionAlgoritmo**(entrada nombreProtecto: String, entrada directorioLogs: String): String. -**GuardarLog**(entrada salidaError: ref string, entrada estK: ref int, entrada numItera: double, entrada segFWh: double, entrada horaMinEnSeg: double, entrada horaMaxEnSeg: double, entrada posKmFinalWh: double): void -**LlegaAntes**(entrada segRealesEntreEstac: ref double, entrada factor: ref double, entrada VmaxPermitidaWh: ref double, entrada estK: int, entrada segFWh: double, entrada horaMinEnSeg: double, entrada factorReducirMax: double, entrada segInicTramo: double, entrada horaSalidaEstF: double): void -**LlegaTarde**(entrada segRealesEntreEstac: ref double, entrada factor: ref double, entrada VmaxPermitidaWh: ref double, entrada horaSalidaEstF: double[], entrada segFWh: double, entrada segInicTramo: double, entrada factorAumentarMin: double): void. -**MostrarInfoFalloPrimeraIteracion**(entrada salidaError: ref String, entrada estK: ref int, entrada VmaxPermitidaWh: double, entrada segFWh: double, entrada horaMinEnSeg: double, entrada horaMaxEnSeg: double, entrada posKmFinalWh: double) -**MostrarInfoMaxIteracionesPermitidas**(entrada salidaError: ref String, entrada estK: ref int, entrada necesidadHolgura: bool, entrada VmaxPermitidaWh: double, entrada segFWh:double, entrada horaMinEnseg: double, entrada horaMaxEnSeg: double, entrada posKmFinalWh: double): bool -**NumMaxPasosAlcanzado**(entrada salidaError: ref String, entrada estK: ref int,

entrada numItera:double, entrada VmaxPermitidaWh: double, entrada posKmFinalWh: double, entrada segFWh: double, entrada horaMinEnSeg: double, entrada horaMaxEnSeg: double): Boolean
-**ObtencionDatosViaMatMotriz**(entrada PdteWh: ref double, entrada pendienteCorrecta: ref bool, entrada CurvatWh: ref double, entrada curvaCorrecta: ref bool, entrada TunelWh: ref double, entrada coefTunelCorrecto: ref bool, entrada VmaxPermitidaWh: ref double, entrada EsfMaxCurvaWh: ref double, entrada EsfFrenadoWh: ref double, entrada velocMaxim: ref double, entrada maxPendiente: double, entrada minCurva: double, entrada maxCoefTunel: double, entrada VmaxPermitidaWh: double, entrada V0KmhWh: double): String
-**ObtenerEstacionesVirtuales**(entrada structEstaciones: DatosEstaciones): DatosEstaciones
-**ObtenerPkViaEstRea**(entrada viaVirtual: List<DatosViaVirtual>, entrada pkVirtual: double): PkViaEst
-O**btenerResistAvance**(entrada V0KmhWh:double, entrada PdteWh: double, entrada CurvatWh: double, entrada TunelWh: double)
-**ObtenerViaVirtual**(entrada structEstaciones: DatosEstaciones): List<DatosViaVirtual>
-**PrimerCasoEsfuerzo**(entrada AcMs2Wh: ref double, entrada EsfuerzoWh: ref double, entrada potMaxPerm: ref double, entrada potenciaKWWh: ref double, entrada V1mpsWh: ref double, entrada ResistAvanceWh: double, entrada EsfMaxWh: double, entrada potMaxPermitida: double, entrada V0mpsWh: double): void
-**SegundoCasoEsfuerzo**(entrada AcMs2Wh: ref double, entrada EsfuerzoWh: ref double, entrada potMaxPerm: ref double, entrada potenciaKWWh: ref double, entrada V1mpsWh: ref double, entrada estK: int, entrada velocMaxima: double, entrada V0mpsWh: double, entrada errorV:double, entrada longitudKm: double, entrada kmFrenadaWh: double, entrada kmRecorridosWh: double, entrada ResistAvanceWh: double, entrada EsfMaxWh: double, entrada potMaxPermitida: double): void
-**TercerCasoEsfuerzo**(entrada AcMs2Wh: ref double, entrada EsfuerzoWh: ref double, entrada potenciaKWWh: ref double, entrada V1mpsWh: ref double, entrada EsfFrenadoWh: ref double, entrada V0mpsWh: double, entrada velocMaxima: double, entrada ResistAvanceWh: double): void
-**CuartoCasoEsfuerzo**(entrada zonaFrenadoWh: Boolean, entrada AcMs2Wh: ref double, entrada V1mpsWh: ref double, entrada EsfuerzoWh: ref double, entrada potenciaKWWh: ref double, entrada estK: int, entrada kmRecorridosWh: double, entrada kmFrenadaWh: double, entrada longitudKm: double, entrada V0mpsWh: double, entrada ResistAvanceWh: double, entrada EsfFrenadoWh: double): void
-**VariacionPosicion**(entrada VarPosWh: ref double, entrada posKmFinalWh: ref double, entrada kmRecorridosWh: ref double, entrada continuaMismoTramoWh: ref bool, entrada estK: int, entrada V0mpsWh: double, entrada AcMs2Wh: double, entrada posKmInicialWh: double, entrada longitudKm: double).

Figura 18. Clase AlgoritmoGeneraciónMovimientoCircular

- **Nombre**: AlgoritmoGeneraciónMovimientoCircular.
- **Subsistema**: Alg_Movimiento.

- **Descripción**: esta clase contiene el algoritmo de movimiento. Su funcionamiento está descrito en el capítulo de aspectos técnicos del algoritmo de movimiento.

Clase DatosCurvas

DatosCurvas
-i: int **-CURV**: double[,]
+**GetCURV**():double[,] +**SetTamanyoCURV**(entrada tamanyo:int): void +**AddCurva**(entrada idCurva: double, entrada idVia: double, entrada pkInicial: double, entrada radio: double): void +**LiberarVectores**(): void

Figura 19. Clase DatosCurvas

- **Nombre**: DatosCurvas.
- **Subsistema**: Alg_Movimiento.
- **Descripción**: esta clase contiene un array de dos dimensiones. En el array se guardan los datos de las curvas de la infraestructura para su uso por parte del algoritmo. Además, en ella están presentes los métodos que se usan para la gestión de estos datos. Gracias a los métodos se pueden añadir nuevas curvas al array, o borrar los datos de las curvas para liberar la memoria principal del ordenador.

Clase DatosTúneles

DatosTuneles
-i:int -**TUNELES**: double[,]
+**GetTUNELES**(): double[,] +**SetTamanyoTUNELES**(entrada tamanyo:int):void +**AddTunel**(entrada idTunel: double, entrada idVia: double, entrada pkInicial: double, entrada coefic: double): void +**LiberarVectores**(): void

Figura 20. Clase DatosTuneles

- **Nombre**: DatosTuneles.
- **Subsistema**: Alg_Movimiento.
- **Descripción**: esta clase contiene un array de dos dimensiones donde se guardan los datos de los túneles de la infraestructura para su uso por parte del algoritmo. Además, en ella están presentes los métodos que se utilizan para la gestión de estos datos. Gracias a los métodos se pueden añadir nuevos túneles al array, o borrar los datos de los túneles para liberar la memoria principal del ordenador

Clase DatosEstaciones

DatosEstaciones
-**i**: int -**tamanyo**: int -**numIteracionesPermitidas**: double -**nombServ**: string -**vmaxConj**: double -**tipoConjunto**: double -**pesoAdic**: double -**vectorEstaciones**: string[] -**posKmEstacK**:string[] -**vectorViaEst**: double[] -**numEst**: double -**nombEstInic**: string -**nombEstFin**: string -**vectorVmaxTrazado**: double[] -**vectorSentidos**: double[] -**vectorHoraPasada**: double[,] -**vectorHoraPasadaSeg**: double[] -**vectorParaSiNo**: double[] -**tParada**: double[] -**vectorHoraCalculada**: double[,] -**velocidadInicial**: double -**velocidadFinal**: double
+**getTamanyo()**: int +**getNumIteracionesPermitidas()**: double +**getNombServ()**: string +**getVmaxConj()**: double +**getTipoConjunto()**: double +**getPesoAdic()**: double +**getVectorEstaciones()**: string[] +**getPosKmEstacK()**:string[] +**getVectorViaEst()**: double[] +**getNumEst()**: double +**getNombEstInic()**: string +**getNombEstFin()**: string +**getVectorVmaxTrazado()**: double[] +**getVectorSentidos()**: double[] +**getVectorHoraPasada()**: double[,] +**getVectorHoraPasadaSeg()**: double[] +**getVectorParaSiNo()**: double[] +**getTParada()**: double[] +**getVectorHoraCalculada()**: double[,] +**getVelocidadInicial()**: double +**getVelocidadFinal()**: double +**EstacionesGeneral (**entrada numIteracionesPermitidas: double, entrada nombServ: string, entrada vmaxConj: double, entrada tipoConjunto: double, entrada pesoAdic: double, entrada velocidadInicial: double, entrada velocidadFinal:

double**):**void +**SetTamanyo(**entrada tamanyoAux: int): void +**AddEstacion(**entrada varVectorEstaciones: string, entrada varPosKmEstacK1: double, entrada varVectorViaEst: double, entrada varVectorVmaxTrazado: double, entrada varVectorSentidos: double, entrada varVectorHoraPasada: double, entrada varVectorHoraPasadaMin: double, entrada varVectorHoraPasadaSseg: double, entrada varVectorHoraPasadaSeg: double, entrada varVectorParaSiNo: double, entrada varTparada: double, entrada varVectorHoraCalculadaHora: double, entrada varVectorHoraCalculadaMin: double, entrada varVectorHoraCalculadaSeg: double**):**void +**RellenarVectorHoraPasada()**: void +**RellenarNombEstInicEstFin**(): void +**LiberarVectores()**: void

Figura 21. Clase DatosEstaciones

- **Nombre**: DatosEstaciones.
- **Subsistema**: Alg_Movimiento.
- **Descripción**: esta clase contiene información general de la circulación y de su trayecto. Para cada estación o parada de la circulación se almacena su vía, punto kilométrico, velocidad máxima de paso por la estación, si tiene una parada, y el tiempo de parada. Además, en ella están presentes los métodos que se usan para la gestión de estos datos.

Clase DatosVías

DatosVias
-idVia: int **-curvasVia**: double[,] **-tunelVia:** double[,] -**pendVia**: double[,] **-velocLimVia:** double[,]
+getIdVia(): int **+getCurvasVia()**: double[,] **+getTunelVia():** double[,] **+getPendVia()**: double[,] **+getVelocLimVia():** double[,] +**DatosVias(**entrada idVia: int, entrada curvas: double[,], entrada tuneles: double[,], entrada pendientes: double[,], entrada velocLim: double[,] **)**: void **+LiberarVectores()**: void

Figura 22. Clase DatosVias

- **Nombre**: DatosVias.
- **Subsistema**: Alg_Movimiento.
- **Descripción**: esta clase contiene un array de dos dimensiones donde se guardan los datos de las vías de la infraestructura para su uso por parte del algoritmo.

Además, en ella están presentes los métodos que se utilizan para la gestión de estos datos. Gracias a los métodos se pueden añadir nuevas vías al array, o borrar los datos de las vías para liberar la memoria principal del ordenador

Clase DatosPendientes

DatosPendientes
-i:int **-PEND**: double[,]
+**GetPEND**(): double[,] +**SetTamanyoPEND**(entrada tamanyo:int):void +**AddPendiente**(entrada idPendiente: double, entrada idVia: double, entrada pkInicial: double, entrada pendiente: double): void +**LiberarVectores**(): void

Figura 23. Clase DatosPendientes

- **Nombre**: DatosPendientes.
- **Subsistema**: Alg_Movimiento.
- **Descripción**: esta clase contiene un array de dos dimensiones donde se guardan los datos de las pendientes de la infraestructura para su uso por parte del algoritmo. Además, en ella están presentes los métodos que se utilizan para la gestión de estos datos. Gracias a los métodos se pueden añadir nuevas pendientes al array, o borrar los datos de las pendientes para liberar la memoria principal del ordenador

Clase DatosVíaVirtual

DatosViaVirtual
-pkInicioVirtual: double **-pkFinVirtual**: double **-pkInicioReal**: double **-pkFinReal**: double **-idViaReal**: double
+**getPkInicioVirtual()**: double **+getPkFinVirtual()**: double +**getPkInicioReal()**: double **+getPkFinReal()**: double **+getIdViaReal()**: double +**DatosViaVirtual**(entrada pkInicioVirtual: double, entrada pkFinVirtual: double, entrada pkInicioReal: double, entrada pkFinReal: double, entrada idViaReal: double): void **+LiberarVectores()**: void

Figura 24. Clase DatosViaVirtual

- **Nombre**: DatosViaVirtual.

- **Subsistema**: Alg_Movimiento.
- **Descripción**: esta clase sirve para almacenar la vía virtual, explicada en el capítulo de aspectos técnicos del algoritmo, y para pasársela como parámetro al mismo. Permite la creación de tramos de la vía virtual, así como su borrado general para liberar espacio en la memoria principal del ordenador.

Clase PkViaEst

PkViaEst
-pk: double **-via:** double **-estK:** int **-sentido:** int
+getPk(): double **+getVia():** double **+getEstK():** int **+getSentido():** int **+PkViaEst(**entrada pk: double, entrada via: double, entrada estK: int, entrada sentido: int**):**void

Figura 25. Clase PkViaEst

- **Nombre**: PkViaEst.
- **Subsistema**: Alg_Movimiento.
- **Descripción**: a la hora de usar el algoritmo, se usará la clase PkViaEst para guardar, una vez obtenidos de la vía virtual, los valores reales de punto kilométrico, última estación, y sentido de la circulación.

Clase DatosMatMotriz

DatosMatMotriz
-i**: int** **-j: int** **-nomMatMovil:** string **-tipoMatMovil:** double **-pesoMatMovil:** double **-vmaxMatMovilKmh:** double **-AMatMovil:** double **-BMatMovil:** double **-CMatMovil:** double **-rtoMotorMatMovil:** double **-rtoFrenMatMovil:** double **-potAuxMatMovil:** double **-numLocomotoras:** int **-esfMotMatMotriz**: double[,] **-esfFrenMotMatMotriz:** double[,] **-curvaFrenadoDefecto:** double[,]

+**getNomMatMovil():** string
+**getTipoMatMovil():** double
+**getPesoMatMovil():** double
+**getVmaxMatMovilKmh():** double
+**getAMatMovil():** double
+**getBMatMovil():** double
+**getCMatMovil():** double
+**getRtoMotorMatMovil():** double
+**getRtoFrenMatMovil():** double
+**getPotAuxMatMovil():** double
+**getNumLocomotoras():** int
+**getEsfMotMatMotriz()**: double[,]
+**getEsfFrenMotMatMotriz():** double[,]
+**getCurvaFrenadoDefecto():** double[,]
+**SetMatMotriz(**entrada nombMatMotriz: string, entrada tipoMatMovil: double, entrada pesoMatMovil: double, entrada vmaxMatMovilKmh: double, entrada AMatMovil: double, entrada BMatMovil: double, entrada CMatMovil: double, entrada rtoMotorMatMovil: double, entrada rtoFrenMatMovil: double, entrada potAuxMatMovil: double, entrada numLocomotoras: int**)**:void
+**SetTamanyoMot**(entrada tamanyo:int): void
+**SetTamanyoFren**(entrada tamanyo: int): void
+**AddEsfMotMatMotriz**(entrada velocidad: double, entrada esfuerzo: double, entrada numLocomotoras: int): void
+**AddEsfFrenMotMatMotriz**(entrada velocidad: double, entrada esfuerzo: double, entrada numLocomotoras: int): void
+**RellenarCurvaFrenadoDefecto**(): void
+**LiberarVectores():** void

Figura 26. Clase DatosMatMotriz

- **Nombre**: DatosMatMotriz.
- **Subsistema**: Alg_Movimiento.
- **Descripción**: esta clase contiene información del material motriz del tren. En ella están presentes los métodos que se usan para la gestión de estos datos. Estos métodos permiten definir las curvas de esfuerzo motriz, las curvas de esfuerzo de frenado, y los aspectos característicos del material motriz. También se podrá borrar los datos para liberar carga de la memoria principal del ordenador.

Clase DatosSalida

DatosSalida
-nombreCirculacion: string **-segInic:** double **-segF:** double **-V0mps:** double **-V1mps:** double **-AcMs2:** double **-PosViaInicial:** string **-PosViaFinal:** string **-PosKmInicial:** double **-PosKmFinal:** double **-VarPos:** double **-KmRecorridos:** double **-EsfMax:** double **-ResistAvance:** double **-Esfuerzo:** double **-PotenciaKW:** double **-Pdte:** double **-Curvat:** double **-EsfFrenado:** double
+getNombreCirculacion(): string +**setNombreCirculacion(**entrada nombre: string): void **+getSegInic():** double **+getSegF():** double **+getV0mps():** double **+getV1mps():** double **+getAcMs2():** double **+getPosViaInicial():** string **+getPosViaFinal():** string **+getPosKmInicial():** double **+getPosKmFinal():** double +**getVarPos():** double +**getKmRecorridos():** double **+getEsfMax():** double **+getResistAvance():** double **+getEsfuerzo():** double **+getPotenciaKW():** double **+getPdte():** double +**getCurvat():** double +**getEsfFrenado():** double **+DatosSalida(**entrada segInic: double, entrada segF: double, entrada V0mps: double, entrada V1mps: double, entrada AcMs2: double, entrada posViaInicial: string, entrada posViaFinal: string, entrada posKmInicial: double, entrada posKmFinal: double, entrada VarPos: double, entrada kmRecorridos: double, entrada EsfMax: double, entrada ResistAvance: double, entrada Esfuerzo: double, entrada potenciaKW: double, entrada Pdte: double, entrada Curvat: double, entrada EsfFrenado: double**)**: void

Figura 27. Clase DatosSalida

- **Nombre**: DatosSalida.
- **Subsistema**: Alg_Movimiento.
- **Descripción**: esta clase es usada por el algoritmo de movimiento para ir guardando los datos de salida que se van generando en cada iteración.

Clase DlgValidaciónCirculación

DlgValidaciónCirculación
-button_Aceptar: Button **-button_verCurvas**: Button **-button_Cancelar**: Button **-structEstaciones:** DatosEstaciones -**DatosSalida:** DatosSalida
+button_verCurvas_click(entrada sender: object, entrada e: EventArgs): void **+button_verAceptar_click(**entrada sender: object, entrada e: EventArgs): void **+button_verCancelar_click(**entrada sender: object, entrada e: EventArgs): void

Figura 28. DlgValidacionCirculacion

- **Nombre**: DlgValidaciónCirculación.
- **Subsistema**: Interfaz_Circulaciones.
- **Descripción**: se trata de una interfaz de usuario que se muestra tras validar una circulación desde DlgCirculaciones. Da tres opciones al usuario, aceptar la validación de la circulación, cancelarla, y ver las curvas de esfuerzo, velocidad, posición y tiempo resultantes de la ejecución. Si la circulación no ha podido ser validada, en vez de mostrarse la interfaz de esta clase, se presentará al usuario un mensaje de error.

Clase DlgCurvasCirculación

DlgCurvasCirculación
-tabVelocidadEsfuerzo: TabControl **-tabPageVelEsf:** TabPage **-tabPageVel:** TabPage **-tabPageEsf:** TabPage **-tabPageVelSec:** TabPage **-tabPageEsfSec:** TabPage **-tabPagePosTemp:** TabPage **-tabPageTempPos:** TabPage **-plotterVel1:** Plotter **-plotterEsf1:** Plotter **-ploterVel2:** Plotter **-ploterEsf2:** Plotter **-plotterVelSec:** Plotter

-plotterEsfSec: Plotter **-plotterPosTiempo:** Plotter **-plotterTiempoPos:** Plotter **-listaVelocidad:** List<double> **-listaEsfuerzo:** List<double> **-listaPkm:** List<double> **-listaSegundos:** List<double> **-listaEstacionesPkm:** List<double> **-cruzEstacionVel1:** List<double> **-cruzEstacionEsf1:** List<double> **-cruzEstacionVel2:** List<double> **-cruzEstacionEsf2:** List<double> **-cruzEstacionVelSec:** List<double> **-cruzEstacionSecVel:** List<double> **-cruzEstacionPosSec:** List<double> **-cruzEstacionSecPos:** List<double> **-mayorVelocidad:** double **-menorVelocidad:** double **-mayorEsfuerzo:** double **-menorEsfuerzo:** double
+DlgCurvasCirculacion(entrada listaVelocidad: List<double>, entrada listaEsfuerzo: List<double>, entrada listaPkm: List<double>, entrada listaSegundos: List<double>, entrada listaEstacionesPkm: List<double>**):** void **+contarCifras**(entrada diferencia: double): int **+configurarGrid**(entrada diferencia: double, entrada contadorCifras: int): double **+obtenerInferior**(entrada inferior: double, entrada contadorCifras: int): double **+obtenerSuperior**(entrada superior: double, entrada contadorCifras: int): double **+fObtenerValorCruz**(entrada valorRef: double, entrada listaRef: List<double>, entrada valores: List<double>): double **+obtenerCrucesPlotter**(entrada listaEstacionesPkm: List<double>, entrada listaKm: List<double>, entrada listaValores: List<double>, entrada kmOSeg: int): DataPlotter.PointD[] **+buttonSalir_click**(entrada sender: object, entrada e: EventArgs): void

Figura 29. DlgCurvasCirculacion

- **Nombre**: DlgCurvasCirculación.
- **Subsistema**: Interfaz_Circulaciones.
- **Descripción**: es una interfaz de usuario que contiene varios plotters o gráficas desde los cuales comprobar los resultados obtenidos al validar una circulación.

  Su diseño consta de un sistema de pestañas que permite elegir al usuario entre una curva de resultados u otra.

  Proporciona métodos para configurar y rellenar las gráficas, de forma que los datos se muestren de la forma más entendible para el usuario.

Clase Funciones

Funciones
#**fpendiente(**entrada posKm: double, entrada sent: double, entrada matrizPend: double[,]**):**double #**fcurvatura(**entrada posKm: double, entrada sent: double, entrada matrizCurv: double[,]**):**double #**ftuneles(**entrada posKm: double, entrada sent: double, entrada matrizTunel: double[,]**):**double #**fvelocLim(**entrada posKm: double, entrada sent: double, entrada matrizLimVel: double[,]**):**double #**fInterpolacion(**entrada Vkmh: double, entrada x1: double, entrada x2: double, entrada y1: double, entrada y2: double**):**double #**fValorEsfMot(**entrada Vkmh: double, entrada vectorEsfMot: double[,]**):**double #**fValorEsfFrenado(**entrada Vkmh: double, entrada vectorEsfFren: double[,]**):**double #**fNuevaHora(**entrada horaIn: double, entrada minIn: double, entrada segIn: double, entrada segAdic: double**):** double[] #**fRestarTiempo(**entrada horaIn: double, entrada minIn: double, entrada segIn: double, entrada segAdic: double**):** double[] #**fDifTiempoSegundos(**entrada horasCero: double, entrada minutosCero: double, entrada segCero: double, entrada horas: double, entrada minutos: double, entrada seg: double**):**double #**fHorasSegundos(**entrada horas: double, entrada minutos: double, entrada segundos: double**):**double #**fObtenerVias(**entrada curvas: DatosCurvas, entrada pendientes: DatosPendientes, entrada tuneles: DatosTuneles, entrada velocLim: DatosVelocLim**):**List<DatosVias>

Figura 30. Clase Funciones

- **Nombre**: Funciones.
- **Subsistema**: Alg_Movimiento.
- **Descripción**: esta clase presenta una serie de métodos que sirven de utilidad al algoritmo. Estos métodos permiten obtener información de la infraestructura a partir de un instante determinado de la circulación. Por ejemplo, a partir de la posición kilométrica del tren o su sentido.

  También hay métodos para realizar cálculos con horas e interpolar valores entre dos puntos.

Clase Circulación

Circulación
+**IDCirculacion:** int **+IDTren:** int **+IDTrayecto:** int

+Nombre: string **+TipoCirculacion:** int **+VelocidadInicial:** double **+VelocidadFinal:** double **+HoraValidacion:** DateTime
+GetCirculaciones(): CirculacionRow[] **+GetCirculacionRowByNombre(**entrada nombre: string**):** CirculacionRow +**GetCirculacionRowByIdCirculación(**entrada idCirculacion: int): CirculacionRow **+GetCirculacionExisteByIdCirculacion(**entrada idCirculacion: int): bool +**EliminarCirculacion():** void **+ModificarCirculacion(**entrada tren: TrenRow, entrada trayecto: TrayectoRow, entrada nombre: string, entrada tipoCirculacion: int, entrada velIni: double, entrada velFin: double, entrada horaValidacion: DateTime**)**: void **+GetFirstIntervaloCirculacionRow()**: IntervalosCirculacionRow **+GetParadasCirculacionByCirculacionOrderByOrden**(entrada circulación: CirculacionRow): ParadasCirculacionRow[]

Figura 31. Clase circulación

- **Nombre**: Circulacion.
- **Subsistema**: Base_Datos.
- **Descripción**: esta clase permite la gestión de la tabla de circulaciones de la base de datos. Proporciona métodos para obtener circulaciones y paradas, de acuerdo a diferentes criterios, y otros para modificar circulaciones o eliminarlas.

Clase Tren

Tren
+IDTren: int **+IDInventario:** string **+Nombre:** string **+IDMaterialMotriz:**int **+NumTracciones:** int **+Longitud:** double **+PesoRemolcado:** double **+ConsumoAux:** double **+VelocidadMax:** double **+TipoCirculacion:** int **+TipoUso:** int
+GetTrenRowByIdInventario(entrada idInventario: string**)**: TrenRow **+GetTrenRowsByNombre**(entrada nombre: string):TrenRow[] **+GetTrenExisteByIdInventario**(entrada idInventario: string) Boolean **+EliminarTren**(): void **+ModificarTren**(entrada idInventario: string, entrada nombre: string, entrada material: MaterialMotrizRow, entrada numTracciones: int, entrada longitud: double, entrada pesoRem: double, entrada consumoAux: double, entrada velMax: double, entrada tipoCirculacion: int, entrada tipoUso: int): void

Figura 32. Clase Tren

- **Nombre**: Tren.
- **Subsistema**: Base_Datos.
- **Descripción**: esta clase permite la gestión de la tabla de trenes de la base de datos. Proporciona métodos para modificar, eliminar, u obtener trenes de acuerdo a diferentes criterios.

Clase MaterialMotriz

MaterialMotriz
+IDMaterialMotriz: int **+Codigo:** string **+Nombre:** string **+TipoMaterialMotriz:** int **+VelMaxima:** double **+TensionMinima:** double **+TensionMaxima:** double **+CoeficienteA:** double **+CoeficienteB:** double **+CoeficienteC:** double **+Masa:** double **+NumEjes:** int +**Longitud:** double **+EsfuerzoArranque:**double **+CoefAdherenciaEstatico:** double
+CrearMaterialMotriz(entrada codigo: string, entrada nombre: string**):** MaterialMotrizRow

Figura 33. Clase MaterialMotriz

- **Nombre**: MaterialMotriz.
- **Subsistema**: Base_Datos.
- **Descripción**: esta clase permite la gestión de la tabla de MaterialMotriz de la base de datos. Proporciona métodos para modificar, eliminar, u obtener materiales motrices de acuerdo a diferentes criterios.

Clase Curvas

Curvas
+IDCurva: int **+IDMaterialMotriz:** int **+TipoCurva:** string **+NumCurva:** int **+TipoDefinicion:** string
+**CrearMaterialMotriz**(entrada código:string, entrada nombre: string, entrada

tipoMaterialMotriz: int, entrada velMaxima: double, entrada tensionMinima: double, entrada tensionMaxima: double, entrada coefA: double, entrada coefB: double, entrada coefC: double, entrada masa: double, entrada numEjes: int, entrada longitud: double, entrada esfArranque: double, entrada coefAdhEstatico: double): MaterialMotrizRow +**GetMaterialesMotriz**(): MaterialMotrizRow[] +**GetMaterialMotrizByCodigo**(entrada codigo: string): MaterialMotrizRow[] **+GetMaterialMotrizByNombre**(entrada nombre: string): MaterialMotrizRow[] **+GetMaterialMotrizExisteByCodigo(**entrada codigo: string): Boolean +**EliminarMaterialMotriz**(): void **+ModificarMatMotriz**(entrada código:string, entrada nombre: string, entrada tipoMaterialMotriz: int, entrada velMaxima: double, entrada tensionMinima: double, entrada tensionMaxima: double, entrada coefA: double, entrada coefB: double, entrada coefC: double, entrada masa: double, entrada numEjes: int, entrada longitud: double, entrada esfArranque: double, entrada coefAdhEstatico: double): void

Figura 34. Clase Curvas

- **Nombre**: Curvas.
- **Subsistema**: Base_Datos.
- **Descripción**: esta clase permite la gestión de la tabla de curvas de esfuerzo, propias de los materiales motrices, perteneciente a la base de datos. Proporciona métodos para modificar, eliminar y obtener curvas de acuerdo a diferentes criterios.

Clase ValoresCurva

ValoresCurva
+IDValoresCurva:int **+IDCurva:** int **+X:** double **+Y:** double
+CrearCurva(entrada matMotriz: MaterialMotrizRow, entrada tipoCurva: string, entrada numCurva: int, entrada tipoDefinicion: string**)**: CurvaRow **+GetCurvas()**: CurvaRow[] **+GetCurvaByMatMotrizOrderByIDCurva**(entrada matMotriz: MaterialMotrizRow): CurvaRow[] **+EliminarCurva**(): void **+ModificarCurva**(entrada matMotriz: MaterialMotrizRow, entrada tipoCurva: string, entrada numCurva: int, entrada tipoDefinicion: string**)**: void

Figura 35. Clase ValoresCurva

- **Nombre**: ValoresCurva.
- **Subsistema**: Base_Datos.

- **Descripción**: esta clase permite la gestión de la tabla de ValoresCurva, propios de las curvas de esfuerzo, perteneciente a la base de datos. Proporciona métodos para modificar, eliminar y obtener curvas de acuerdo a diferentes criterios.

Clase Trayecto

Trayecto
+IDTrayecto: int **+IDInventario:** string **+Nombre:** string **+Descripcion:** string
+GetTrayectoRowByIdInventario(entrada idInventario: string): TrayectoRow +**GetTrayectoRowsByNombre**(entrada nombre: string):TrayectoRow +**GetTrayectoExisteByIdInventario**(entrada idInventario: Boolean): Boolean +**EliminarTrayecto()**: void +**ModificarTrayecto(**entrada idInventario: string, entrada nombre: string, entrada descripcion: string**)**: void

Figura 36. Clase Trayecto

- **Nombre**: Trayecto.
- **Subsistema**: Base_Datos.
- **Descripción**: esta clase permite la gestión de la tabla de Trayectos de la base de datos. Proporciona métodos para modificar, eliminar, u obtener trayectos de acuerdo a diferentes criterios.

Clase SeccionTrayecto

SeccionTrayecto
+IDSeccionTrayecto:int **+IDTrayecto:** int **+Orden:** int **+IDEstacionInicial:** int **+IDEstacionFinal:** int **+IDVia:** int **+Distancia:** double **+OrdenPkCreciente:** boolean
+GetSeccionesTrayectoByTrayectoOrderByOrden(entrada trayecto: TrayectoRow): SeccionTrayectoRow[] **+GetSeccionTrayectoExisteByIdInventario**(entrada inventario: String): Boolean +**CrearSeccionTrayecto**(entrada IDTrayecto: int, entrada Orden: int, entrada IDEstacionInicial: int, entrada IDEstacionFinal: int, entrada IDVia: int, entrada Distancia: double, entrada OrdenPkCreciente: Boolean): void +**ModificarSeccionTrayecto**(entrada IDTrayecto: int, entrada Orden: int, entrada IDEstacionInicial: int, entrada IDEstacionFinal: int, entrada IDVia: int, entrada Distancia: double, entrada OrdenPkCreciente: Boolean): void **+EliminarSeccionTrayecto()**: void

+**EliminarSeccionTrayectoConReordenamiento()**: void

Figura 37. Clase SeccionTrayecto

- **Nombre**: SeccionTrayecto.
- **Subsistema**: Base_Datos.
- **Descripción**: esta clase permite la gestión de la tabla de SeccionTrayecto de la base de datos. Proporciona métodos para modificar, eliminar, u obtener secciones de trayecto de acuerdo a diferentes criterios.

Clase ParadasCirculacion

ParadasCirculacion
+**IDParadasCirculacion:**int +**IDCirculacion:** int +**IDEstacion:** int +**Orden:** int +**Parada:** boolean +**HoraLlegada:** TimeSpan +**HoraSalida:** TimeSpan +**TiempoParada:** TimeSpan +**Velocidad:** double
+**GetParadasCirculacionByCirculacionOrderByOrden(**entrada circulación: CirculacionRow): ParadasCirculacionRow[] +**CrearParadaCirculacion**(entrada idCirculacion: int, entrada idEstacion: int, entrada orden: int, entrada parada: boolean, entrada horaLlegada: TimeSpan, entrada horaSalida: TimeSpan, entrada TiempoParada: TimeSpan, entrada Velocidad: double): ParadasCirculacion +**ModificarParadaCirculacion**(entrada idCirculacion: int, entrada idEstacion: int, entrada orden: int, entrada parada: boolean, entrada horaLlegada: TimeSpan, entrada horaSalida: TimeSpan, entrada TiempoParada: TimeSpan, entrada Velocidad: double): void +**EliminarParadaCirculacion**(): void

Figura 38. ParadasCirculación

- **Nombre**: ParadasCirculacion
- **Subsistema**: Base_Datos.
- **Descripción**: esta clase permite la gestión de la tabla de ParadasCirculacion de la base de datos. Proporciona métodos para modificar, eliminar, u obtener paradas de circulación de acuerdo a diferentes criterios.

7.3.3 Base de datos

A continuación se presenta el diseño de la base de datos. Incluiremos las tablas que han tenido que ser creadas para el presente proyecto. Por tanto, todas las no relacionadas directamente con el algoritmo de movimiento, no son incluidas.

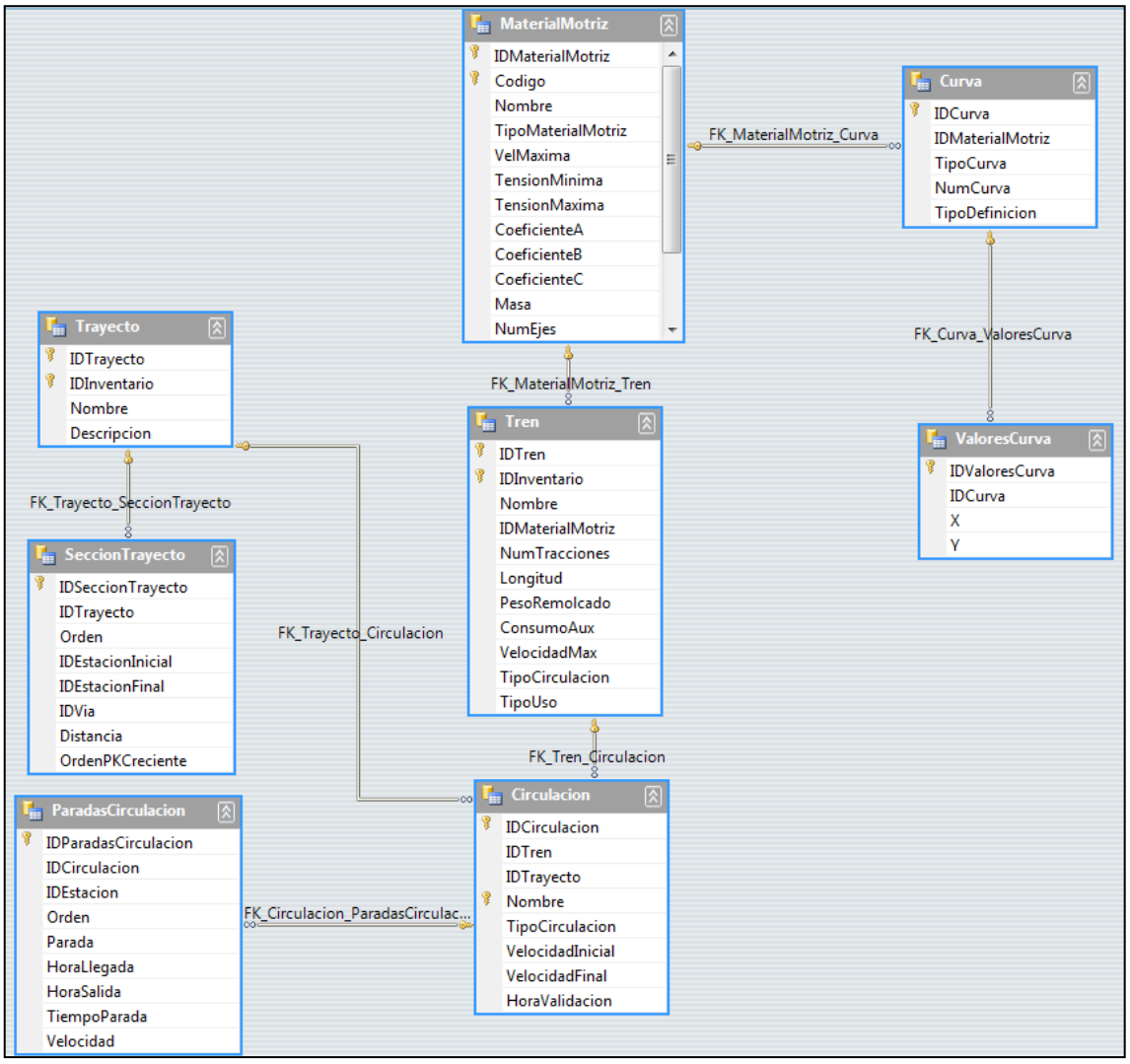


Figura 39. Esquema de la base de datos

Cada circulación tendrá un tren, un trayecto, y una o varias paradas. A su vez, un tren, un trayecto o una parada de circulación, podrán estar asociados a cero o varias circulaciones.

Cada tren estará asociado a un material motriz y un material motriz podrá estar asociado a cero o mucho trenes. Cada material motriz posee una o varias curvas de esfuerzo, las cuales tienen uno o varios valores de curva.

CAPÍTULO 8

Pruebas

Capítulo 8: Pruebas

Una vez finalizados el análisis, diseño e implementación, se procede a realizar una serie de pruebas para verificar el correcto funcionamiento del programa.

En el presente apartado se llevarán a cabo las siguientes pruebas:

- Verificación de una circulación por parte de algoritmo.
- Verificación de varias circulaciones por parte del algoritmo.
- Tiempo de ejecución del algoritmo de movimiento en plataformas multicore.

Para las pruebas se usarán datos reales de trenes, infraestructuras y circulaciones, proporcionados por ADIF. Con este fin se ha creado un proyecto en la aplicación con una infraestructura que se centra en la zona norte de Madrid. En este proyecto, llamado "Caso de estudio MADRID-VILLALBA", hay 182 circulaciones que se corresponden con otras existentes.

8.1 Verificación de una circulación

La primera prueba a realizar se trata de una circulación con código en el programa "EL-MAD0". Si el algoritmo funciona de forma correcta, deberá dar por válida la circulación, puesto que contiene información fiel a la realidad. A continuación se muestra un resumen de dicha información:

Estación	Parada	Hora Llegada	Hora Salida	Tiempo de parada
EL ESCORIAL	True	6:31:00	6:32:00	0:01:00
LAS ZORRERAS-NAVALQUEJIGO	False	6:38:00	6:38:00	0:00:00
SAN YAGO	False	6:40:00	6:40:00	0:00:00
VILLALBA DE GUADARRAMA	True	6:44:00	6:45:00	0:01:00
GALAPAGAR-LA NAVATA	False	6:49:00	6:49:00	0:00:00
TORRELODONES	False	6:53:00	6:53:00	0:00:00
LAS MATAS	False	6:57:00	6:57:00	0:00:00
PINAR DE LAS ROZAS	False	7:01:00	7:01:00	0:00:00
BIF. P. PIO	False	7:04:00	7:04:00	0:00:00
EL TEJAR	False	7:05:00	7:05:00	0:00:00
PITIS	False	7:15:00	7:15:00	0:00:00
RAMON Y CAJAL	False	7:19:00	7:19:00	0:00:00
MADRID-CHAMARTIN	True	7:23:00	7:23:00	0:00:00

Tabla 49. Trayecto de la circulación "EL-MAD0"

Esta circulación es recorrida por un tren de pasajeros, con velocidad máxima de 140 kilómetros por hora y cuyo material motriz es un "U T 450 M4RM".

Figura 40: Serie 450 de ADIF (www.ferropedia.es)

Procedemos a validar la circulación con el algoritmo y obtenemos el siguiente mensaje:

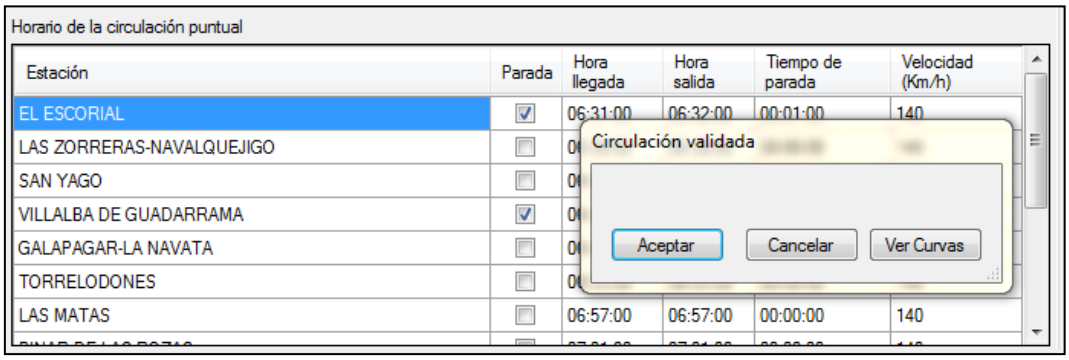


Figura 41. Resultado de circulación validada

Esto quiere decir que el algoritmo de generación de movimiento ha dado por buena la circulación. Con lo cual el programa da la posibilidad al usuario de ver las curvas de esfuerzo, velocidad, posición y tiempo.

Las gráficas que contienen las curvas muestran una cruz por cada estación que hay en la circulación. Para comprobar hasta cierto punto la fiabilidad de los cálculos realizados por el algoritmo hay que tener en consideración los siguientes puntos:

- Cuando el tren arranca, el esfuerzo será siempre el máximo que pueda realizar su material motriz.
- Cuando el tren frena para mantener su velocidad o para reducirla, el esfuerzo se irá reduciendo hasta obtener valores negativos.
- Cuando el tren deba acelerar para aumentar o mantener su velocidad, el esfuerzo se incrementará.
- Al ver las curvas en función de la posición: en el caso de haber una estación con parada, se mantendrá el valor de la velocidad y del esfuerzo a cero, mientras que el valor de la posición se mantendrá (el tren está parado).

A continuación se va analizar los resultados devueltos por el algoritmo y mostrados en las curvas.

Gráficas de velocidad:

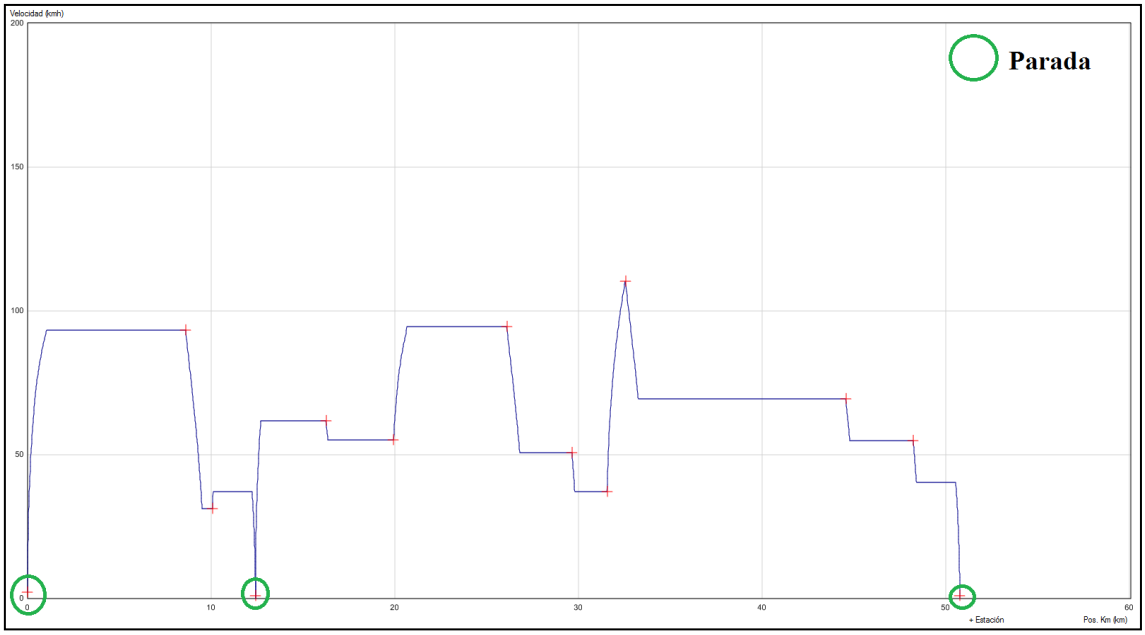


Figura 42. Gráfica velocidad/posición

En la figura 42 se puede ver la velocidad en km/h (eje x) en función de la posición en km (eje y). Se puede ver como en cada parada la posición mantiene su valor y la velocidad se mantiene a cero. Además, se comprueba que al salir de una estación el tren incrementa su velocidad hasta alcanzar aquella que le permite llegar a tiempo a su destino en el tiempo establecido. Igualmente, se ve que cuando el tren llega a la zona de frenado calculada por el algoritmo, este empieza a frenar para llegar con una velocidad igual a cero a la estación con parada.

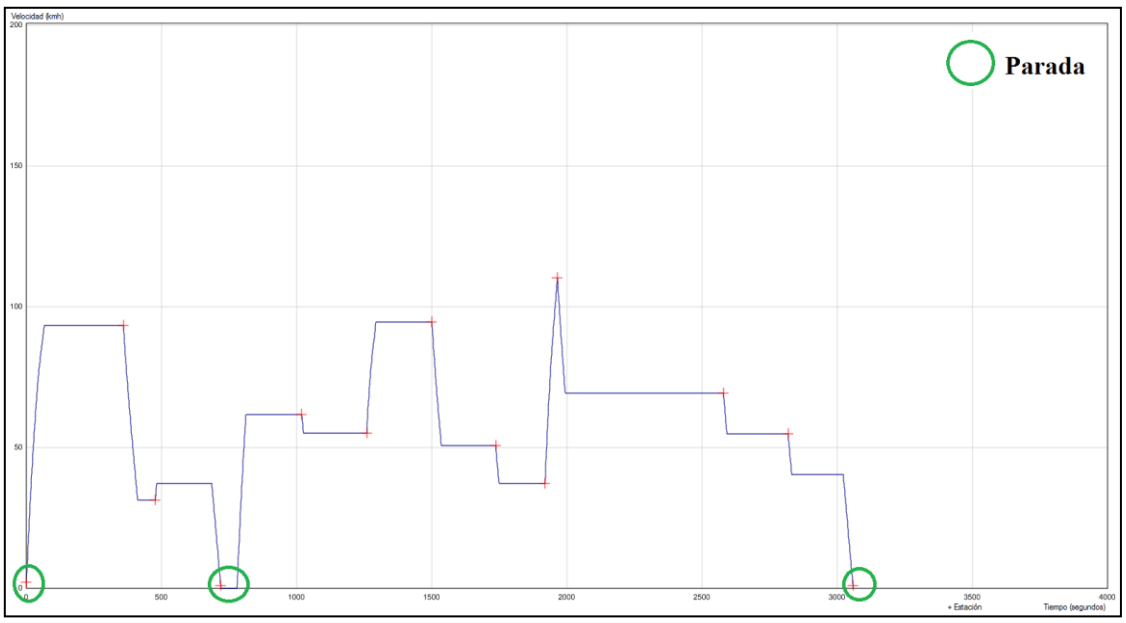


Figura 43. Gráfica velocidad/tiempo

En el caso de representar la velocidad en km/h (eje y) en función del tiempo en segundos (eje x). Se aprecia como en la cuarta estación, que tiene una parada de 60 segundos, la velocidad se mantiene a cero durante ese tiempo.

Gráficas de esfuerzo:

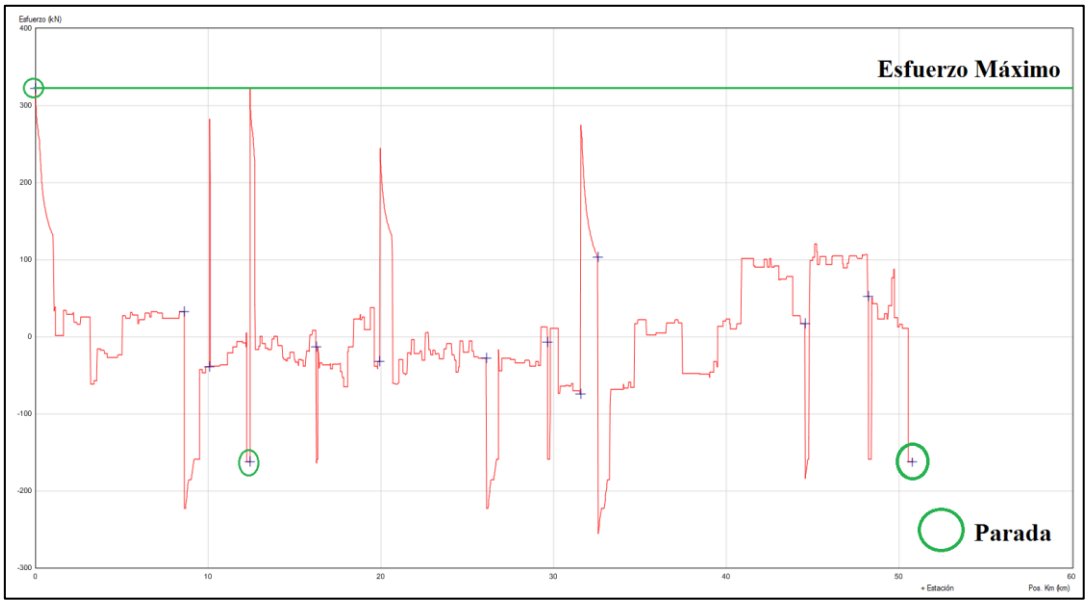


Figura 44. Gráfica esfuerzo/posición

En la figura 44 se muestra el esfuerzo en kN (eje y) en función de la posición en km (eje x). Se puede ver cómo el tren sale de las paradas ejerciendo un esfuerzo máximo. Ese esfuerzo máximo coincide con el del material motriz (322106,4 kN). Además, se puede ver cómo el esfuerzo varía tras cada estación para aumentar o disminuir la velocidad. Las pequeñas fluctuaciones en la curva se deben a las dificultades que se puede encontrar el tren en el trayecto (curvas, pendientes, limitaciones de velocidad, etc.) y que provocan cambios en el esfuerzo para que el tren pueda mantener la velocidad constante.

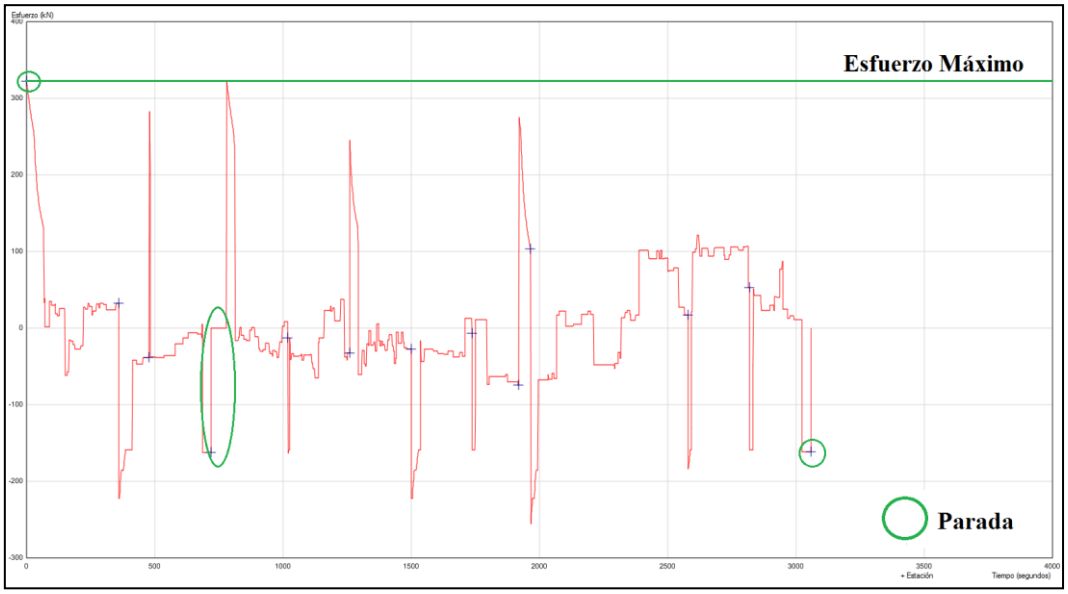


Figura 45. Gráfica de esfuerzo/tiempo

Ahora se muestra el esfuerzo en kN (eje y) en función del tipo en segundos (eje x). A diferencia de la anterior, se aprecia como en la cuarta estación, con parada de un minuto, el esfuerzo se mantiene a cero durante ese tiempo.

Gráficas de posición y tiempo:

Al ser casi iguales, sólo se muestra la gráfica con la curva de posición en segundos (eje y) en función del tiempo en segundos (eje x).

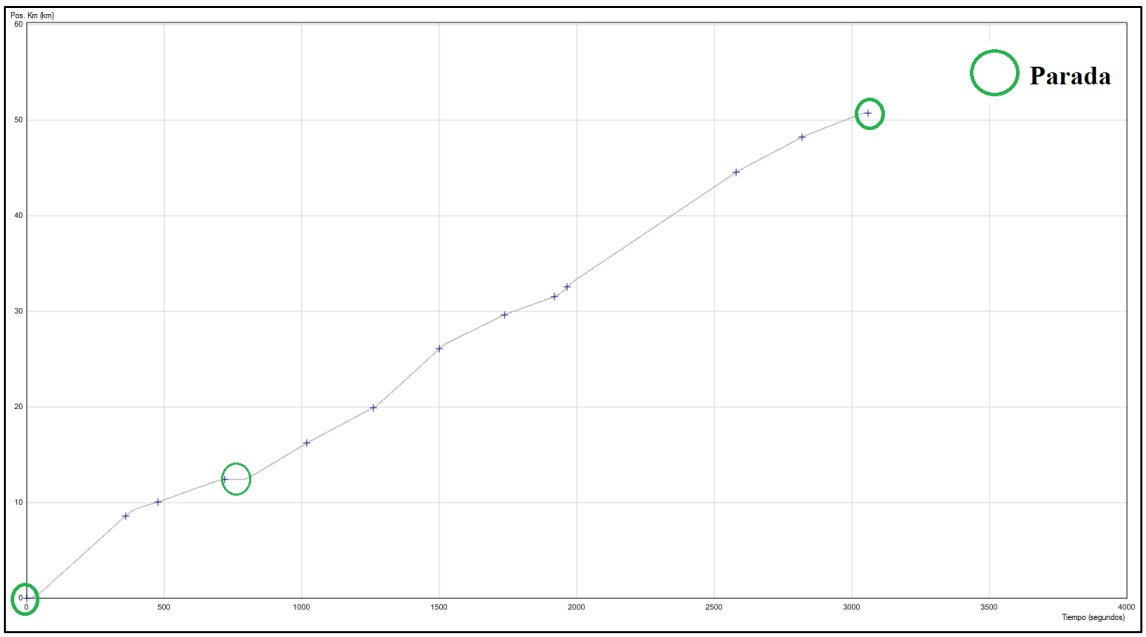


Figura 46. Gráfica posición/tiempo

A medida que la curva de la gráfica tiene mayor pendiente, mayor será la velocidad que tiene el tren para ese momento. Asimismo, en la cuarta estación, se puede comprobar cómo se mantiene la posición durante el tiempo de la parada (60 segundos).

Una vez vistas las curvas, se va a probar a realizar un cambio en la circulación de la prueba anterior. Se modificarán las horas de llegada y salida de una estación. Se reducirán, para que el tren no tenga tiempo a llegar. De esta forma, se muestra como el algoritmo detecta circulaciones con horarios imposibles de realizar por el tren.

El cambio se realizará sobre la estación de “El Tejar”. En vez de llegar a las 7:05:00, se ha cambiado para que llegue a las 7:04:40. Ahora los tiempos de la circulación no son realistas y se obtiene el siguiente mensaje:

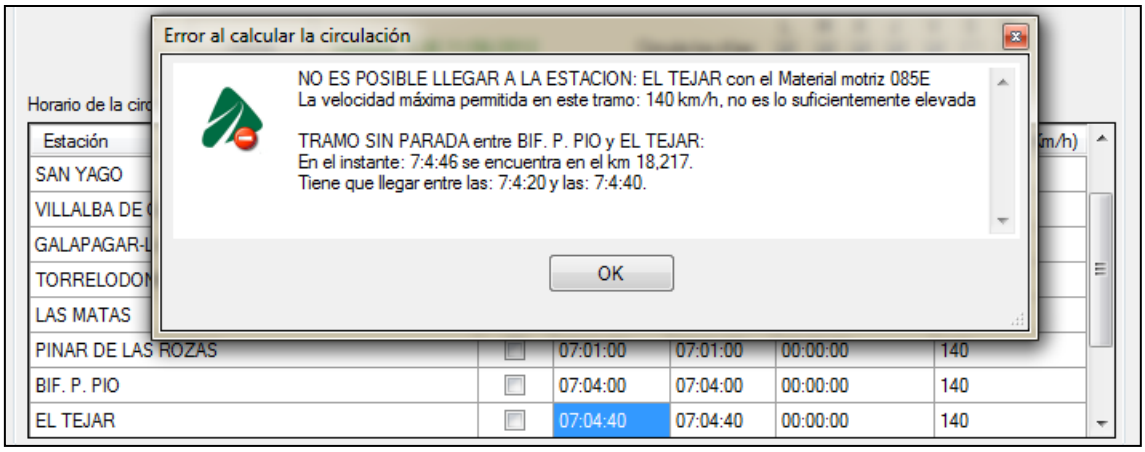


Figura 47. Mensaje de error al validar la circulación

El cual dice que, para esos tiempos, el tren no puede llegar ni aunque fuese a la velocidad máxima posible.

8.2 Verificación de varias circulaciones

Ahora se va a hacer una prueba en la que se intenta validar las 182 circulaciones del proyecto desde la interfaz de gestión de escenarios.

El primer resultado que obtenemos al validar las 182 circulaciones es el siguiente:

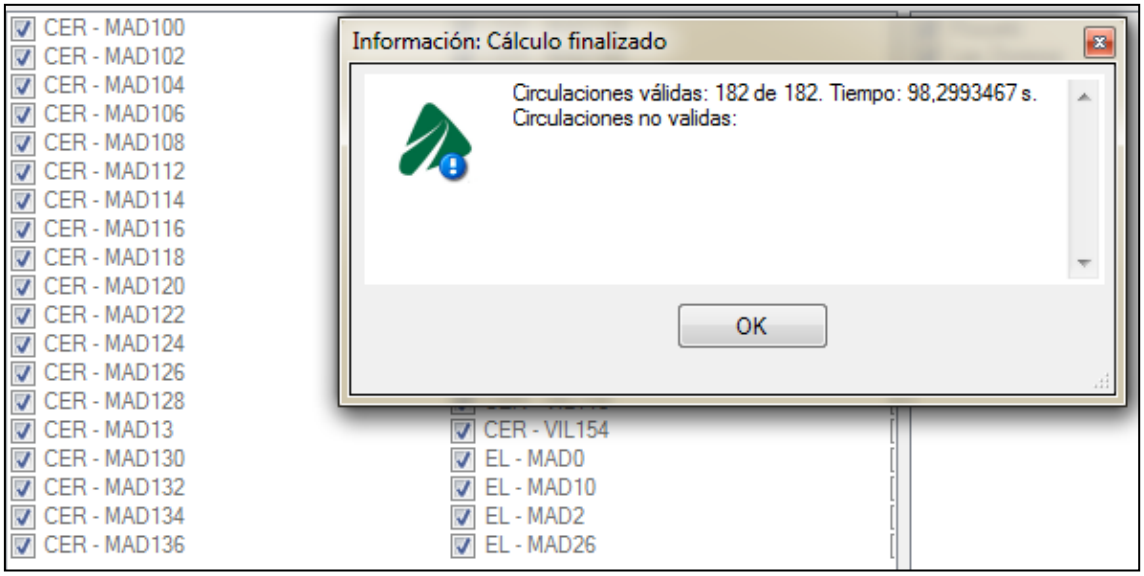


Figura 48. Verificación de varias circulaciones

Es decir, todas las circulaciones se han validado. Con lo que el algoritmo ha generado un movimiento realista para todas ellas. En cambio, si se vuelve a modificar la circulación con nombre “EL-MAD0”, para que falle, se muestra este mensaje:

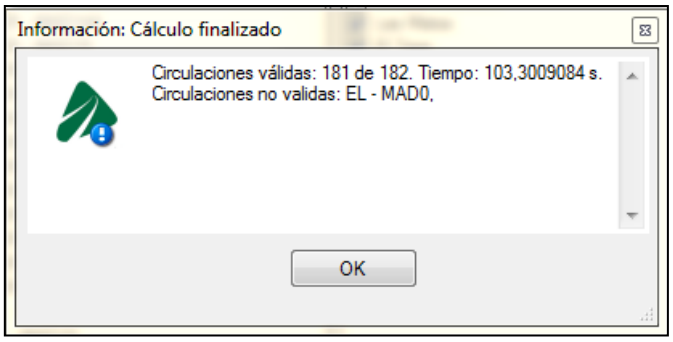


Figura 49: Verificación de varias circulaciones (con error)

El mensaje nos confirma que, de entre las 182 circulaciones, hay una cuyo trayecto y horario no pueden ser cumplidos por el tren. Además, el programa identifica a la circulación que falla, que se corresponde con la que se ha cambiado con anterioridad.

8.3 Análisis de tiempos en plataformas multicore

Se ha lanzado la verificación de las 182 circulaciones del proyecto en cuatro máquinas diferentes con procesador multicore. Primero se definen las cuatro máquinas y luego se analizan los resultados de tiempos obtenidos. La ejecución de la verificación se realizará de dos maneras: secuencial y paralelo.

En secuencial se irán validando las circulaciones una a una. En cambio, en el modo paralelo se validan varias circulaciones a la vez, tantas como núcleos tenga el procesador.

8.3.1 Definición de los equipos

a) Equipo 1

Procesador: Intel Core 2 Duo Processor E6300.
Frecuencia: 1.86 GHz.
Caché: 2 MB.
Número de núcleos: 2 núcleos.
Número de hilos: 2 núcleos.

Capacidad memoria RAM: 2.50 GB.
Sistema Operativo: Windows 7 Professional 64 bits

b) Equipo 2

Procesador: Intel Pentium Processor E5300.
Frecuencia: 2.60 GHz.
Caché: 2 MB.
Número de núcleos: 2 núcleos.
Número de hilos: 2 hilos.

Capacidad memoria RAM: 4 GB.

Sistema Operativo: Windows 7 Home Premium 64 bits

c) Equipo 3

Procesador: Intel I3 370 M.
Frecuencia: 2.40 GHz.
Caché: 3 MB.
Número de núcleos: 2 núcleos.
Número de hilos: 4 hilos.

Capacidad memoria RAM: 4 GB.
Sistema Operativo: Windows 7 Home Premium 64 bits

d) Equipo 4

Procesador: Intel Core i5-760.
Frecuencia: 2.8 GHz.
Caché: 8 MB.
Número de núcleos: 4 núcleos.
Número de hilos: 4 hilos.

Capacidad memoria RAM: 4 GB.
Sistema Operativo: Windows 7 Professional 64 bits

8.3.2 Análisis de los resultados

Para cada equipo se obtienen los siguientes tiempos de ejecución:

Equipo	Secuencial	Paralelo	Speedup	Ahorro de tiempo
A	105.538	73.259	1.44	30.6 %
B	97.05	69.972	1.39	27.9%
C	63.539	40.032	1.59	37%
D	42.073	19.172	2.19	54.4%

Tabla 50. Tiempos de ejecución de la validación de 182 circulaciones

Con los tiempos de ejecución se obtiene la figura 50:

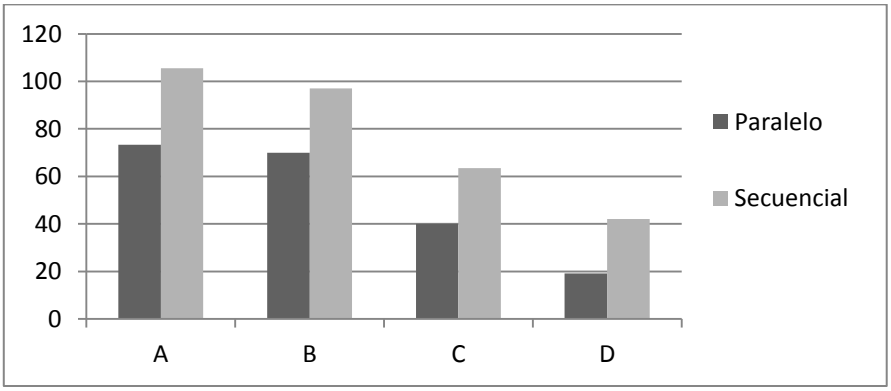


Figura 50. Gráfica de tiempos de ejecución de la validación de 182 circulaciones

Gracias a la gráfica se puede ver como la optimización del algoritmo para este tipo de sistemas cumple con su propósito. En cualquier procesador con varios núcleos, el tiempo de ejecución en modo paralelo es menor que en secuencial.

Cuanto mayor es el número de núcleos del procesador, mayor es la diferencia entre los dos tipos de ejecución. Por ejemplo, en el ordenador con el procesador I5-760 llega a reducir a la mitad los tiempos de espera.

Estos resultados se ven confirmados por el porcentaje de ahorro de tiempo. En los procesadores con dos núcleos, se ahorra entorno a un 30% ejecutando la validación en paralelo. Excepto en el caso del Intel I3, que con una arquitectura más avanzada, permite mejorar los tiempos. Mientras que, en el ordenador con procesador Intel I5, el porcentaje de ahorro se dispara hasta el 54.432 %.

Por tanto, a mayor número de núcleos en el procesador, mayor es el ahorro de tiempo con respecto a la ejecución de la aplicación en modo secuencial.

En computación en paralelo, el speedup [18] se refiere a cuánto de rápido es un algoritmo paralelo con respecto a su correspondiente algoritmo secuencial. Se define mediante la siguiente fórmula:

$$S_p = \frac{T_1}{T_p}$$

T_1 es el tiempo de ejecución del algoritmo secuencial, mientras que T_p, es el tiempo de ejecución del algoritmo paralelo con p procesadores. Un valor de speedup ideal se obtiene cuando su valor se acerca al del número de procesadores, puesto que al ejecutar un algoritmo con el doble de procesadores también se dobla la velocidad.

En este caso, el speedup está lejos de ser el ideal, pero en la gráfica se puede ver como hay una buena escalabilidad a medida que aumenta el número de procesadores.

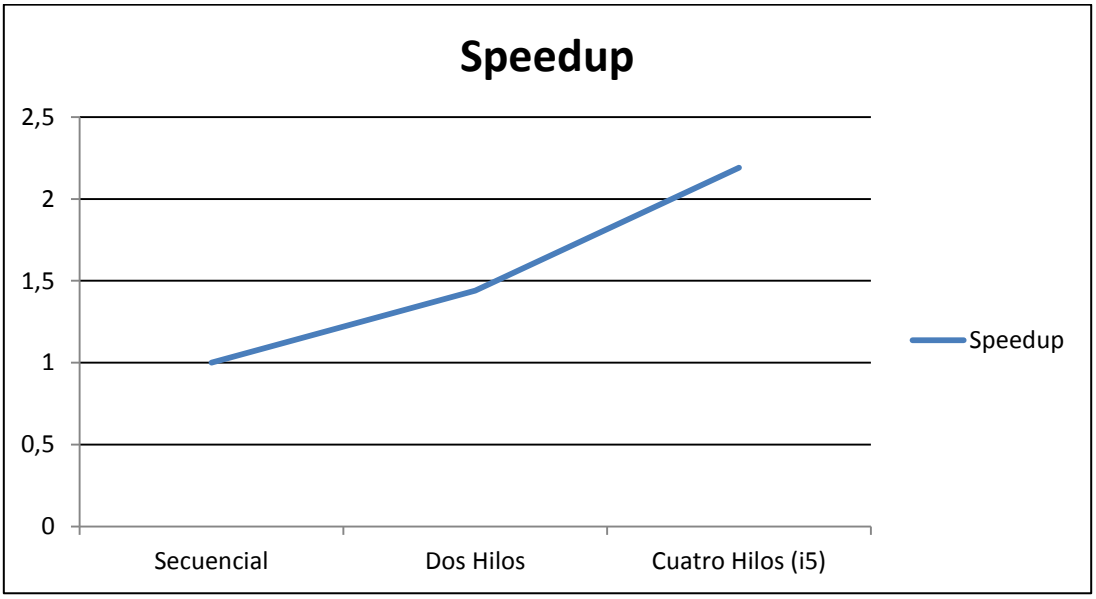


Figura 51. Gráfica de speedup

CAPÍTULO 9

Presupuesto

Capítulo 9: Presupuesto

El presupuesto contiene los gastos que ha provocado la realización del proyecto. Contempla las fases del desarrollo, los gastos en personal y los gastos en recursos tecnológicos como ordenadores y aplicaciones para los mismos.

9.1 Fases del desarrollo

A continuación se muestran las diferentes fases que se han llevado a cabo para la realización del proyecto. La duración de las fases se contabiliza en días hábiles.

Fase	Duración	Descripción
Estudio previo de la infraestructura.	5 días.	Estudio de todos los elementos que envuelven a una circulación. Desde las infraestructuras a los trenes usados por la compañía ADIF.
Estudio previo del problema.	15 días.	Familiarización con el movimiento de un tren. Comprender como los elementos de la circulación afectan al consumo y velocidad del tren. Conocer cómo es el movimiento de los trenes en las circulaciones reales y hallar la manera de conseguir un movimiento óptimo.
Estudio previo del entorno de Visual Studio.	7 días.	Aprendizaje en el uso del entorno de desarrollo Visual Studio para programar en lenguaje C#. Análisis de códigos ya creados con anterioridad, modificación de los mismos y lectura del API online proporcionado por Microsoft.
Análisis.	3 días.	Desarrollar los casos de uso y requisitos que el programa debe cumplir.
Diseño.	7 días.	Estudio de la arquitectura de la aplicación donde reside el proyecto. Definición de los subsistemas de diseño y de las diferentes clases que forman la aplicación.
Implementación.	31 días.	Programación en lenguaje C# de las clases definidas en la fase de diseño.
Pruebas.	17 días.	Corrección de los errores producidos durante la fase de implementación y optimización del funcionamiento de la aplicación.
Documentación.	30 días.	Redacción del presente documento.

Tabla 51: Duración de las fases del desarrollo del proyecto

A continuación se muestra el diagrama de Gantt asociado a las fases del proyecto.

Nombre	Duración	Inicio	Fin
Estudio previo de la infraestructura	5d	06/01/2012	12/01/2012
Estudio previo del problema	15d	13/01/2012	02/02/2012
Estudio previo del entorno de Visual Studio	7d	03/02/2012	13/02/2012
Análisis	3d	14/02/2012	16/02/2012
Diseño	7d	17/02/2012	27/02/2012
Implementación	31d	28/02/2012	10/04/2012
Pruebas	17d	11/04/2012	03/05/2012
Documentación	30d	04/05/2012	14/06/2012

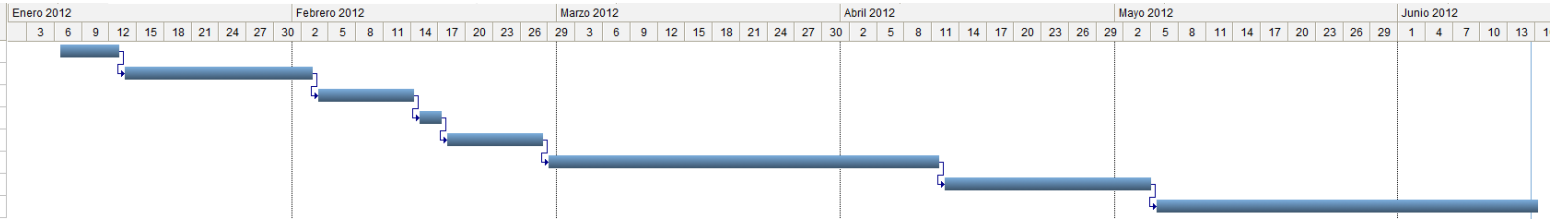


Figura 52. Diagrama de Gantt

9.2 Recursos humanos

Están formados por las personas encargadas de la realización del proyecto. De acuerdo al cargo, recibirán más o menos honorarios.

De manera específica, dos son las personas que han llevado a cabo el proyecto. Las identificamos como "Ingeniero junior" e "Ingeniero senior". El ingeniero junior ha estado presente en todas las fases del desarrollo, mientras que el ingeniero senior ha estado presente en todas menos en las fases de pruebas y documentación.

Podemos obtener los costes que suponen sus salarios brutos de la siguiente forma:

Categoría	Número de hombres	Dedicación (días)	Coste hombre/día	Coste (Euros)
Ingeniero junior	1	115	65 €	7.475 €
Ingeniero senior	1	68	135 €	8.775 €
			Total:	16.250 €

Tabla 52. Costes de personal

9.3 Recursos tecnológicos

A continuación se muestran los costes debido a los equipos usados:

Descripción	Coste	% Uso dedicado al proyecto	Dedicación (meses)	Periodo de depreciación (meses)	Coste imputable
Ordenador Ingeniero Junior	650 €	100	6	60	65 €
Ordenador Ingeniero Senior	750 €	100	4	14	214,29 €
				Total:	279,29 €

Tabla 53. Costes de equipos

El valor del coste imputable ha sido obtenido a través de la fórmula de cálculo de la amortización:

$$\frac{A}{B} \cdot C \cdot D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto

También hay que tener en cuenta los gastos ocasionados por la compra de productos necesarios para el desarrollo del proyecto. Estos costes directos del proyecto son mostrados a continuación:

Descripción	Empresa	Unidades	Precio	Costes
Windows 7 Professional	Microsoft	2	162 €	324 €
Office 2010 Hogar y Estudiantes	Microsoft	1	139 €	139 €
Visual Studio 2008 Professional	Microsoft	2	499 €	998 €
			Total:	1.461 €

Tabla 54. Costes directos de software

9.4 Resumen de costes

El presupuesto final del proyecto es el siguiente:

Presupuesto Costes Totales	
Personal	16.250 €
Equipos	279,29 €
Costes directos del software	1.461 €
Subtotal	**17.990,29 €**

18 % I.V.A 3.238,25 €

PRESUPUESTO FINAL	**21.228,54 €**

Tabla 55. Presupuesto Final

El presupuesto final del proyecto asciende a la cantidad de 21.228,54 € (veintiún mil doscientos veintiocho con cincuenta y cuatro euros)

Leganés a 14 de junio de 2012

Fdo. Manuel Muñoz Archidona

CAPÍTULO 10

Conclusiones y líneas futuras

Capítulo 10: Conclusiones y líneas futuras

10.1 Conclusiones

Una vez finalizado el proyecto, se puede asegurar que se han cumplido todos los objetivos que se pretendían conseguir.

- Se han analizado, descrito y comprendido todos los elementos que participan en una circulación y que afectan al movimiento del tren.
- Se ha diseñado e implementado un algoritmo capaz de calcular el movimiento óptimo de un tren para una circulación. Además, se han realizado diversas pruebas para comprobar su correcto funcionamiento.

  Las pruebas se han realizado con datos de circulaciones reales, para que sus trayectos sean verificados por el algoritmo. Asimismo, se han modificado otras circulaciones para que el algoritmo las descarte al presentar unas características que las hacían inviables en la vida real. Ante todas estas pruebas, el algoritmo se ha comportado de forma correcta.

  Igualmente, se ha llevado a cabo un análisis sobre los tiempos de ejecución del algoritmo para la validación de varias circulaciones. Este análisis se ha producido sobre varios equipos con procesadores multicore. La conclusión que obtenemos de él es que, cuantos más núcleos tiene el procesador, menor es el tiempo que tarda en ejecutarse. De esta forma se ha demostrado que el algoritmo está correctamente optimizado para este tipo de procesadores.
- El proceso de análisis, diseño e implementación ha dado lugar a un entorno adecuado para el uso de la aplicación. El usuario obtiene por parte de la aplicación respuestas a sus decisiones. Puede visualizar los resultados de la ejecución del algoritmo y obtiene mensajes que le muestran información sobre el resultado de los diferentes procesos y los errores cometidos. Por último, la aplicación proporciona al usuario una interfaz intuitiva, que implica un corto período de aprendizaje.
- Para poder implantar correctamente el proyecto sobre la aplicación desarrollada, se ha aprendido a utilizar el entorno de desarrollo Microsoft Visual Studio bajo lenguaje C#. El aprendizaje se produjo antes y durante el desarrollo del proyecto. Los conocimientos adquiridos han permitido implementar el proyecto de forma eficiente y, a pesar de la complejidad computacional de un algoritmo iterativo de este tipo, se ha conseguido que los tiempos de ejecución sean lo más pequeños posible.
- Tanto la arquitectura, como los subsistemas definidos en el diseño, permiten que los futuros cambios que sean necesarios introducir en el proyecto no supongan una grave penalización en tiempo y dinero en el presupuesto.

10.2 Líneas futuras

El proyecto ha sido diseñado para facilitar la incorporación de nuevas funcionalidades a la aplicación.

Una de esas posibles funcionalidades sería la de controlar las colisiones que se pudieran producir entre los trenes. Al compartir los trenes infraestructuras reales, es necesario comprobar que el movimiento generado por el algoritmo no provoca un choque de los mismos.

Por tanto, una buena solución sería crear un algoritmo que leyera los datos de varias circulaciones, producidos por el algoritmo de movimiento, y los comparara hasta encontrar un punto kilométrico donde dos trenes coincidan en un mismo instante de tiempo.

Otra buena incorporación consistiría en poder mostrar las gráficas de varias circulaciones a la vez. De esta forma, el usuario podría comparar dos circulaciones similares para ver con más claridad cual le proporciona mejores prestaciones para un ahorro de consumo energético.

Además, sería recomendable considerar otras alternativas de paralelización del algoritmo de movimiento. Hasta el momento se paralelizan las validaciones a nivel de circulación, es decir, se pueden realizar validaciones de varias circulaciones a la vez. Sin embargo, se podría añadir un grado de paralelismo más fino relativo a una sola circulación. Consistiría en fragmentar de forma lógica una circulación en tramos que discurren entre estaciones con parada. Como los trenes parten de cero en cada uno de estos tramos, no hay dependencias entre ellos, por lo que, en vez de lanzar un hilo por circulación, se podría lanzar uno por cada tramo de esta circulación. Por tanto, si todos los tramos son válidos, la circulación también lo será.

Cabría también la posibilidad de paralelizar una circulación en tramos que discurren entre estaciones sin parada. Pero esta opción resulta inviable, puesto que se necesita saber la velocidad a la que el tren llega a la estación para saber la velocidad de salida de la misma.

Asimismo, dado que la principal motivación de este proyecto es el ahorro del consumo energético, el algoritmo diseñado sirve como base para una nueva extensión de la aplicación encargada de calcular ese consumo energético.

CAPÍTULO 11

Bibliografía

Capítulo 11: Bibliografía

[1] Clemow, C. J. (1972). Planning for railway electrification. In *Proceedings of the IEE. , vol. 119*(4), 431–440.

[2] Ross, B. A. (1971). A Survey of Western European AC Electrified Railway Supply Substation and Catenary System Techniques and Standards. In *IEEE Transactions on Industry and General Applications., vol. IGA-7*(5), 666–672.

[3] Crompton, O. J., Wallace, G. A. (1953). Economic aspects of overhead equipment for d.c. railway electrification In *Proceedings of the IEE - Part I: General. , vol. 100*(124), 133–145.

[4] De Rus, G., Inglada, V. Análisis coste-beneficio del tren de alta velocidad en Espana. *Revista de Economía Aplicada 1* (1993) 27–48.

[5] Maceri, F., Casciati, F. Modelling and simulation of advanced problems and smart systems in civil engineering. *Simulation Modelling Practice and Theory 11* (5–6) (2003) 313.

[6] Bae, C.H. A simulation study of installation locations and capacity of regenerative absorption inverters in DC 1500 V electric railways system. *Simulation Modelling Practice and Theory 17* (5) (2009) 829–838.

[7] Banerjee, N., Saha, A.K., Karmakar, R., Bhattacharyya R. Bond graph modeling of a railway truck on curved track. *Simulation Modelling Practice and Theory 17* (1) (2009) 22–34

[8] Yalçinkaya, O., Bayhan, G.M. A feasible timetable generator simulation modelling framework for train scheduling problem. *Simulation Modelling Practice and Theory 20* (1) (2012) 124–141.

[9] Gomez C., Saa, R., Garcia, A., Garcia-Carballeira, F., Carretero, J. A model to obtain optimal designs of railway overhead knuckle junctions using simulation, *Simulation Modelling Practice and Theory 26* (2012) 16-31.

[10] Goodman C.J., Siu, L.K., Ho, T.K. A review of simulation models for railway systems. *Developments in Mass Transit Systems, 1998*, p. 80-85.

[11] Martin, P. Train Performance and Simulation. *IET Digest*, Volume 2010, Issue 131517, p. 191-206.

[12] Chang, C.S., Sim, S.S. Optimising train movements through coast control using genetic algorithms. *Electric Power Applications*, Volume 144 (Enero 1997), p. 65-73.

[13] Yong, D., Fang-Ming, Z., Yun, B., Tin-kin, H., Yu-fai, F. "Parallel Computing for Multi-train Movement Simulation on Electrified Railway," icic, vol. 4, pp.280-283, 2009 *Second International Conference on Information and Computing Science*, 2009

[14] Ho, T.K., Mao, B.H., Yuan, Z.Z., Liu, Z.Z., Fung, Y.F. Computer simulation and modeling in railway applications, *Computer Physics Communications*, Volume 143, Issue 1, 1 February 2002, Pages 1-10

[15] Alberto, G.A., Dinámica de los trenes en alta velocidad.

[16] Manual de instrucciones del programa SimulaFFCC.

[17] Tipler, P.A. Física para la ciencia y la tecnología. Volumen 1. Cuarta edición.

[18] http://en.wikipedia.org/wiki/Speedup (Accedido el 02/06/12).

Esta página ha sido dejada en blanco de forma intencionada.