UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Proyecto de Fin de Carrera de Ingeniero Informático

HERRAMIENTA PARA LA REGULACIÓN DE LA CIRCULACIÓN DE TRENES

MIGUEL ARRANZ PASCUAL

Dirigido por: **ALFONSO URQUÍA MORALEDA**

Curso: CURSO 2013/2014 Presentación Enero 2014

HERRAMIENTA PARA LA REGULACIÓN

DE LA

CIRCULACIÓN DE TRENES

Proyecto de Fin de Carrera de modalidad oferta específica

Realizado por: **MIGUEL ARRANZ PASCUAL**

Dirigido por: **ALFONSO URQUÍA MORALEDA**

Tribunal calificador:

Presidente: D./D[a]. .....

(firma)

Secretario: D./D[a]. .....

(firma)

Vocal: D./D[a]. .....

(firma)

Fecha de lectura y defensa: .....

Calificación: .....

Resumen

El ser humano se ha ayudado siempre de representaciones esquemáticas de la realidad con la finalidad de interpretarla y entenderla mejor, transmitiendo esta información a sus semejantes.

Con el desarrollo del ferrocarril se fueron creando centros de mando para la regulación de la circulación de los trenes, desarrollándose a la vez herramientas de trabajo para esta función.

Una herramienta de este tipo es la Malla Gráfica de Trenes, una representación de la posición teórica o real de los trenes en el tiempo dentro de un área de infraestructura ferroviaria determinada. Estas mallas gráficas se realizaban en un principio sobre papel y actualmente con medios informáticos.

La utilidad de esta herramienta es presentar al operador humano una información que le ayude a determinar con su experiencia el estado de la situación y tomar las decisiones adecuadas para optimizar la regularidad en la circulación de los trenes.

El objetivo de este proyecto es desarrollar una aplicación informática que dibuje y represente la Malla Gráfica de Trenes y que además de la información necesaria para esta representación, guarde el conocimiento que usa el experto humano, de forma que la propia aplicación sea también capaz de interpretar las situaciones de la realidad de la misma forma que lo hace el usuario.

Una vez diseñada y programada la aplicación objetivo de este proyecto, podemos considerar que se han conseguido cuando menos los siguientes resultados:

- Se ha desarrollado una estructura propia de clases de datos en C++ para administrar el conocimiento necesario para las necesidades y objetivos de la aplicación.

- La aplicación genera por si misma los horarios de trenes a partir de unos datos mínimos de entrada, lo que facilita en posibles desarrollos ampliados posteriores, modificar los datos de un tren en uno o varios puntos y generar un nuevo horario o analizar y comparar varios horarios diferentes.

- Se ha implementado una representación gráfica de la malla de trenes con un aspecto final similar a otras herramientas profesionales al uso, pero realizado con objetos gráficos y lenguaje de programación distintos.

- Se ha comprobado que la aplicación identifica las situaciones de trenes en el tiempo incompatibles con la infraestructura, que posteriormente requieren la adopción de decisiones para su solución.

Si la herramienta cumple éstos objetivos es un paso para el posterior desarrollo de simuladores en el campo profesional de la regulación de la circulación de trenes, que ante situaciones teóricas que se les presenten, las identifiquen y muestren el resultado de las medidas de respuesta posibles a adoptar, con el objetivo de minimizar retrasos o solventar situaciones de incidencias de forma eficaz.

Lista de Palabras Clave

Malla Gráfica de Trenes

Infraestructura Ferroviaria

Estación

Bifurcación

ByPass

Agujas

Punto de Paso

Trayecto

Vía

Tren

Horario

Circulación

Abstract

Human beings have always used schematic representations of reality in order to interpret it and understand it better, transmitting this information to their peers.

With development of the railway system, manager centers were created for the control of train traffic, also developing tools for this function.

An example of this kind of tool is the Graphic Train Net, a representation of the situation (real or theoretical) of the trains in time and beyond a specific railway area. These nets were originally made in paper. However, nowadays computer methods are used.

The usefulness of this tool is based on its capability of giving information to the human operator which helps him to determine, according to his experience, the state of the situation and make the most accurate decisions to optimize the regularity of the train traffic.

The aim of this project is to develop a computer application which not only draws, represents and gives the information of the Graphic Train Net, but also saves the knowledge used by the human expert. With this procedure, the application itself is able to read into the situation in the same way the user would.

Once this application is designed and programmed, we can conclude that at least the following results have been achieved:

- A specific structure for C++ data has been designed in order to manage the necessary knowledge for the aims and needs of the application.
- The application generates on its own the train schedules starting from minimum input data, which makes it easier to modify the data of a train or different points and generate a new schedule, or analize and compare several different schedules, in possible later developments of the application.
- A graphic representation of the train net has been implemented, with a similar aspect to other professional usage tools, but made with different programming language and graphic objects.

- It has been proved that the application identifies in time the trains situations which are incompatibles with the infrastructure that they later require making decisions for their solution.

If this tool meets these objectives, it would be the basis for further development of simulators in the professional field of train traffic control. These simulators, facing theoretical situations, could identify them and show the result of possible response measures to take, in order to minimize delays or solve incident situations in an efficient way.

Keywords

Graphic Train Net

Railway infrastructure

Train Station

Bifurcation

ByPass

Needles

Waypoint

Course

Railway

Train

Train Schedule

Train Traffic

Índice

Lista de Figuras

Lista de Ficheros de Código Incluidos

Capítulo

1

Introducción, Objetivos y Estructura

1.1 Introducción

Las mallas de trenes son una representación del mundo real que sirve para visualizar mentalmente un estado momentáneo de la realidad, interpretarlo y así actuar influyendo en la forma deseada sobre ese mundo real.

La realidad es una infraestructura ferroviaria construida para que sobre la misma se desplacen los trenes desde su origen hasta su destino, empleando para ello un tiempo determinado. Puesto que esta infraestructura enlaza diferentes puntos geográficos de un país, a lo largo del tiempo tenemos en cada instante diferentes trenes en distintas posiciones geográficas y por tanto interactuando entre sí.

Con las mallas de trenes se pretende representar esta realidad en un gráfico de dos dimensiones, en el eje de abscisas se representa el tiempo, en el de ordenadas diferentes puntos de la infraestructura ferroviaria, los espacios entre estos puntos representan tramos de la infraestructura que los enlazan. En el gráfico se dibujan segmentos que unen puntos del mismo formando líneas, cada línea representa el movimiento de un tren por los diferentes puntos y tramos de la infraestructura a lo largo del tiempo.

Esta representación se ha utilizado en el ferrocarril desde que el desarrollo del mismo fue aumentando el número de trenes y haciendo más compleja la regulación del tráfico ferroviario.

En principio estas mallas gráficas se representaban sobre papel, usándose dos tipos, una teórica que se dibujaba con los trenes y sus movimientos previstos y otra real que se iba dibujando con las informaciones que el personal de servicio en las estaciones y puntos de paso iba transmitiendo sobre las horas de paso de cada tren.

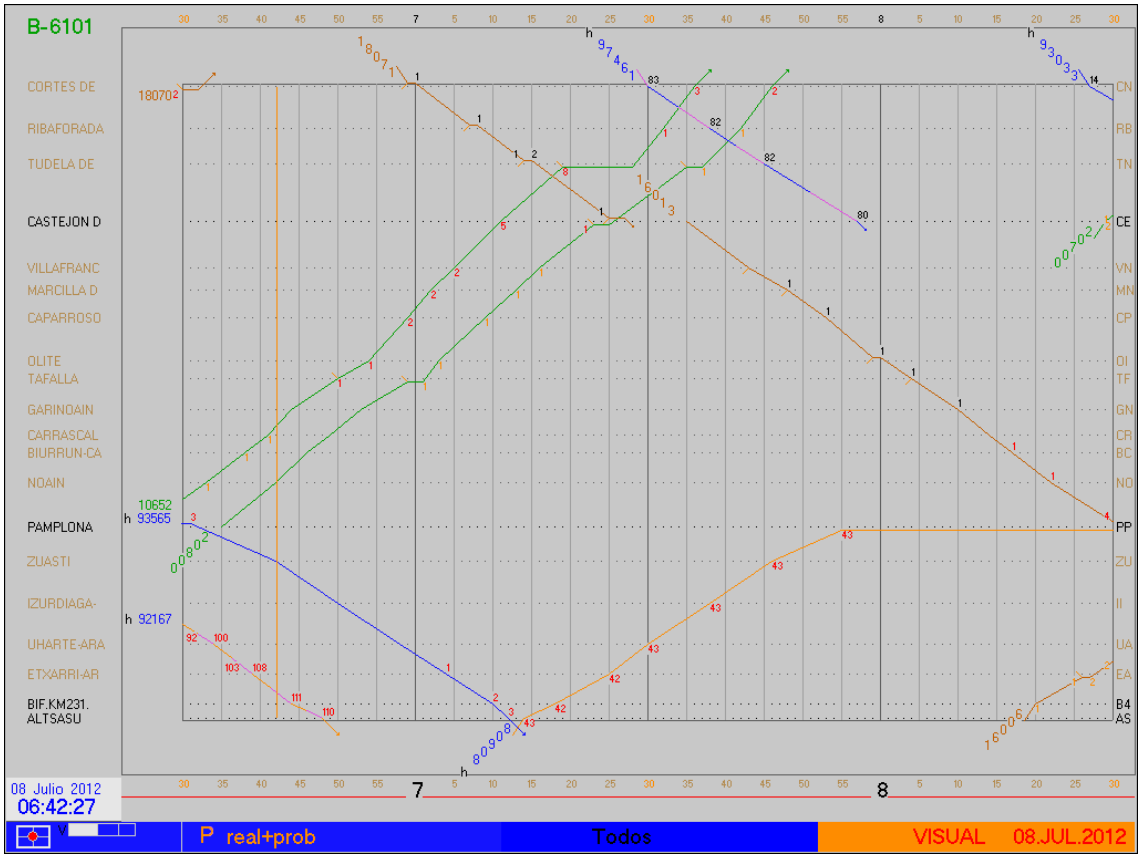

Fig. 1.1 Malla de Trenes

(En la Figura 1.1 se muestra una malla de trenes real, banda con los trayectos superiores de vía doble e inferiores de vía única)

Con el desarrollo de los ordenadores y la informática se crearon las aplicaciones que dibujan en una pantalla los gráficos teóricos y los reales según se van introduciendo los datos con las informaciones de paso de trenes recibidas. Mas recientemente se han introducido en la infraestructura elementos que captan automáticamente el paso de los trenes y transmiten éstos a la base de datos correspondiente.

Con estos gráficos y el conocimiento que el usuario tiene de la realidad, de la infraestructura, los trenes y los procedimientos de seguridad, se detectan los movimientos y situaciones incompatibles y se adoptan las decisiones procedentes para optimizar la regularidad de los trenes con las restricciones que la capacidad de la infraestructura impone.

1.2 Objetivos

El objetivo de este proyecto es crear una aplicación que, a partir de unos datos de entrada de infraestructura y trenes, represente las mallas gráficas descritas y además la propia aplicación sea capaz de detectar algunas de las incidencias e incompatibilidades en los movimientos previstos de los trenes en esa infraestructura: las mismas incompatibilidades que debe detectar un usuario con el conocimiento que tiene de la realidad que la malla gráfica representa.

Por tanto hay un primer objetivo, que es que la aplicación dibuje y represente una malla gráfica de trenes característica, pero además se considera muy importante el proceso de análisis y diseño.

Analizar la realidad de la infraestructura y los trenes, para extraer los atributos y características necesarios para esta representación y principalmente extraer el conocimiento. Si la aplicación guarda este conocimiento podrá detectar por si misma, lo que observa el usuario y además abre la puerta a un simulador, es decir en futuros desarrollos, la propia aplicación, de acuerdo con unos criterios especificados puede proponer soluciones, aplicarlas y optimizar resultados.

A continuación se describen los resultados tangibles que deben obtenerse. Se pretende con este proyecto diseñar e implementar una herramienta de software, programada en lenguaje C++, que tenga la funcionalidad siguiente:

- La herramienta admite varios ficheros de entrada. Un grupo de ficheros configura toda la información referente a la infraestructura de la red y otro grupo de ficheros configura la información relativa a la operación prevista de los trenes.

- La herramienta debe comprobar que los ficheros de entrada son sintácticamente correctos, mostrando en su caso los correspondientes mensajes de error.

- Si ambos ficheros son correctos, la herramienta debe comprobar que los recorridos planificados de los trenes son compatibles por ejemplo, que el recorrido está compuesto por una sucesión de PP adyacentes (hay una vía que los comunica) etc. y generará los recorridos horarios de los trenes.

- La herramienta debe dibujar a continuación las mallas previstas de trenes tal como se han descrito en las secciones anteriores.

- La herramienta debe detectar también las incompatibilidades que se pueden producir:

    - En cuanto a las características de las vías adjudicadas a los trenes en sus paradas en un punto de paso determinado.

    - De vías y coincidencias de circulación de los trenes en el tiempo en un trayecto determinado.

El objetivo de este proyecto es representar gráficamente el desplazamiento en el tiempo de los trenes a través de una determinada infraestructura para proporcionar una visión global de estos movimientos. Además se pretende detectar las incongruencias e incompatibilidades en las interacciones entre los trenes y la infraestructura o entre los trenes entre sí en cuanto a sus horarios y ocupación de vías e instalaciones.

1.3 Estructura de la Memoria del Proyecto

Esta memoria se estructura en varios capítulos. A continuación se describe someramente a qué se dedica cada uno de ellos.

El Capítulo 2 describe algunos conceptos relativos a la Regulación de la Circulación de Trenes y un primer análisis de las entidades y conceptos que intervienen en la misma.

El Capítulo 3 describe la especificación de la aplicación que va a realizarse, se explican mas detalladamente los objetivos a conseguir y la interfaz gráfica de usuario necesaria para presentar los resultados.

El Capítulo 4 describe los principales datos necesarios, cómo se deducen algunas de las relaciones entre estos datos y la consistencia y permanencia de datos y relaciones.

El Capítulo 5 describe la malla gráfica de trenes, cómo se construye y presenta en la aplicación.

El Capítulo 6 describe los algoritmos y funciones con los que identifica el sistema cada una de las incompatibilidades descritas en la especificación.

El Capítulo 7 describe el diseño de la aplicación, la organización modular y la descripción más detallada de sus clases y relaciones.

En el Capítulo 8 se hace una relación de las pruebas realizadas en las distintas fases del desarrollo.

En el Capítulo 9 se describen las fases del desarrollo y su secuencia temporal también se hace un análisis aproximado de costes.

En el Capítulo 10 se resumen las conclusiones y los posibles trabajos futuros.

1.4 Estructura del CD del Proyecto

El CD contiene la carpeta raíz denominada PFC_MiguelArranzPascual. Esta carpeta contiene dos archivos

- leame.txt con el resumen del contenido del CD
- PFCMallas EXE : un acceso directo para ejecutar la aplicación.

Además las siguientes carpetas:

BIN : Contiene el código de la aplicación en dos subcarpetas:

- **Archivos de Encabezado**: Que contiene los archivos de extensión .h agrupados en subcarpetas según la distribución modular de la aplicación

- **Archivos de código fuente**: Que contiene los archivos de extensión .cpp agrupados en subcarpetas según la distribución modular de la aplicación.

DOC : Contiene la memoria en formato .pdf, un archivo denominado Anexo A que contiene también el código completo y otros documentos.

DATOS : Contiene una copia de los archivos de texto con los datos de infraestructura y trenes.

PFC_Mallas : Contiene el ejecutable de la aplicación, otros archivos necesarios y los archivos de texto de los datos.

PFC_PruebasMallas : Con el mismo contenido de la anterior con la finalidad de ejecutar la aplicación con variaciones en los datos para efectuar pruebas.

PFCMallas_VC2005 : Contiene el proyecto íntegro para cargar con Visual Studio 2005

PFCMallas_VC2010 : Contiene el proyecto íntegro para cargar con Visual Studio 2010.

Esta estructura de carpetas se muestra en la Figura 1.2

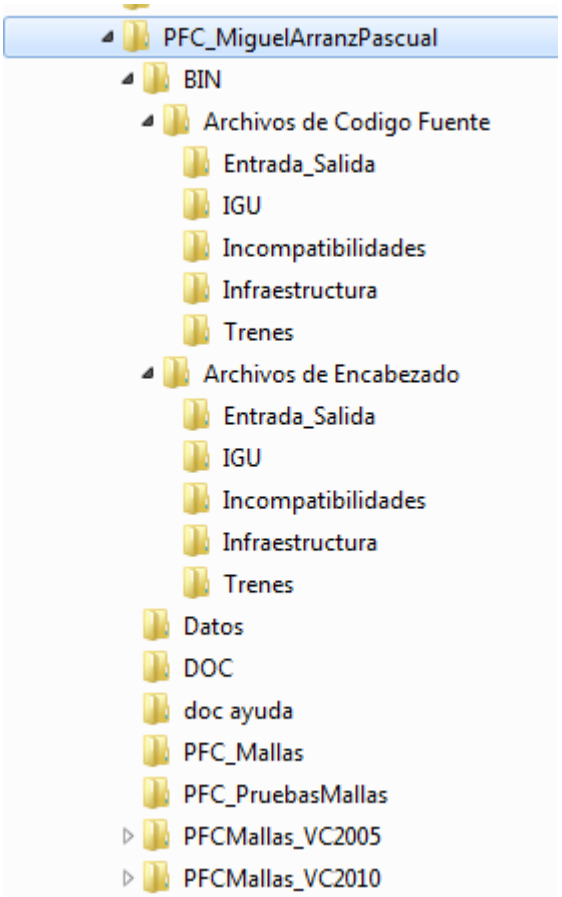

Fig. 1.2 Estructura de carpetas del CD

Capítulo

2

Regulación de la Circulación de Trenes

2.1 Introducción

La gestión y regulación de la circulación de trenes descansa en dos aspectos fundamentales: la seguridad y la regularidad.

La primera se consigue con rigurosos protocolos de actuación sobre los diferentes dispositivos de seguridad (señales, agujas, órdenes, etc.), que garantizan las paradas y movimientos de los trenes de un modo seguro.

Con el desarrollo de las comunicaciones y la informática, cada vez más todas estas instalaciones y dispositivos de seguridad son tele mandados. Los protocolos de seguridad y las incompatibilidades de órdenes de señales, agujas etc…, han sido implementadas en aplicaciones informáticas que garantizan su funcionamiento de un modo absolutamente seguro. El operador de estos sistemas interactúa con los mismos sobre una representación esquemática de la realidad, que le permite comprobar a distancia que los movimientos de agujas vías e indicaciones de señales y otros dispositivos son correctas.

La segunda tiene por objeto conseguir que, además de que los trenes circulen con seguridad, efectúen las paradas intermedias y lleguen a destino en el horario que tienen establecido.

Una herramienta de ayuda principalmente para esta segunda función son las Mallas Gráficas de Trenes o **“Mallas de Trenes”.**

Las “Mallas de Trenes” son una representación de la realidad como tantas otras que utiliza el ser humano en sus actividades para ayudarse, obteniendo una mejor información y visión de esa parte de la realidad que en un momento le interesa para la consecución de sus objetivos.

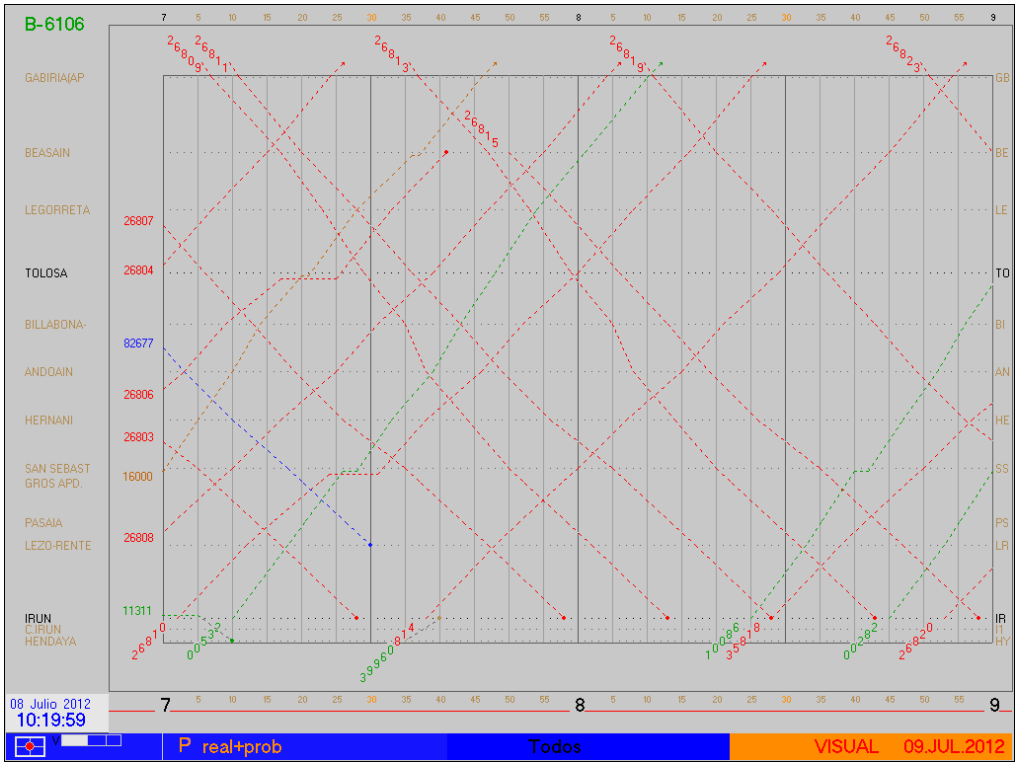

Fig. 2.1 Malla de trenes

(En la Figura 2.1 se muestra una malla prevista de trenes, banda de cercanías, vía doble en todos los trayectos)

Estas mallas gráficas de trenes constituyen una representación de la situación de los trenes en el espacio y en el tiempo.

Las mallas se construyen con los datos de los horarios de paso de cada tren por los diferentes puntos: estaciones, apeaderos, apartaderos, bifurcaciones, etc. Pueden ser horarios previstos o reales.

En el eje de abscisas se representa el tiempo y en el eje de ordenadas los puntos de paso (abreviadamente, Pp). Las líneas de la malla definen el recorrido del tren, como es obvio de mayor o menor inclinación según la velocidad del tren y horizontales en las paradas. En la Figura 2.1 se muestra una malla gráfica prevista.

A la vista de la malla de trenes, el operador humano con sus conocimientos de la infraestructura y los trenes, detecta las incompatibilidades de marcha, planifica los cruces o alcances de trenes para corregir disfunciones y maximizar la regularidad de los trenes.

El operador tiene a la vista un grupo de puntos de paso (banda) y ve como se dispondrían en el tiempo los trenes en esa zona, pero además debe conocer o tener información sobre las características de los Pps., vías con su longitud y situación, los trayectos si son de vía única o doble así como el número de trenes que admiten a la vez, que está determinada por su longitud y los dispositivos de seguridad y señalización de que disponen.

Con esta información, se toman las decisiones sobre puntos de cruce o alcances de trenes. Esto implica paradas que aumentan el tiempo de recorrido de los trenes, es decir retrasos sobre su horario previsto. Además, suponen un desplazamiento de la línea de recorrido de ese tren en esa banda y las siguientes, lo que implican nuevas interacciones con otros trenes en otros puntos y por tanto nuevas decisiones de cruces o alcances.

La labor de los operadores y reguladores de la circulación de trenes es conseguir que los movimientos de trenes se realicen con total seguridad y minimizando los retrasos de los mismos, en situaciones normales o degradadas por incidencias en los trenes o en las instalaciones. Para esta labor, las mallas de trenes, como representación gráfica actualizada de su situación y movimientos es una herramienta imprescindible.

Para tener una visión más clara del tema, vamos a analizar primeramente en líneas generales, cómo se modelizan los dos elementos físicos principales, la **infraestructura** y los **trenes,** así como sus interacciones, apoyando este primer análisis con algunos ejemplos e imágenes.

2.2 La Infraestructura

La red viaria ferroviaria construida sobre una zona geográfica nacional está compuesta por un conjunto de líneas que se cruzan entre sí, determinando con su disposición y accesos de unas a otras el recorrido de los trenes entre diferentes puntos de esa zona geográfica.

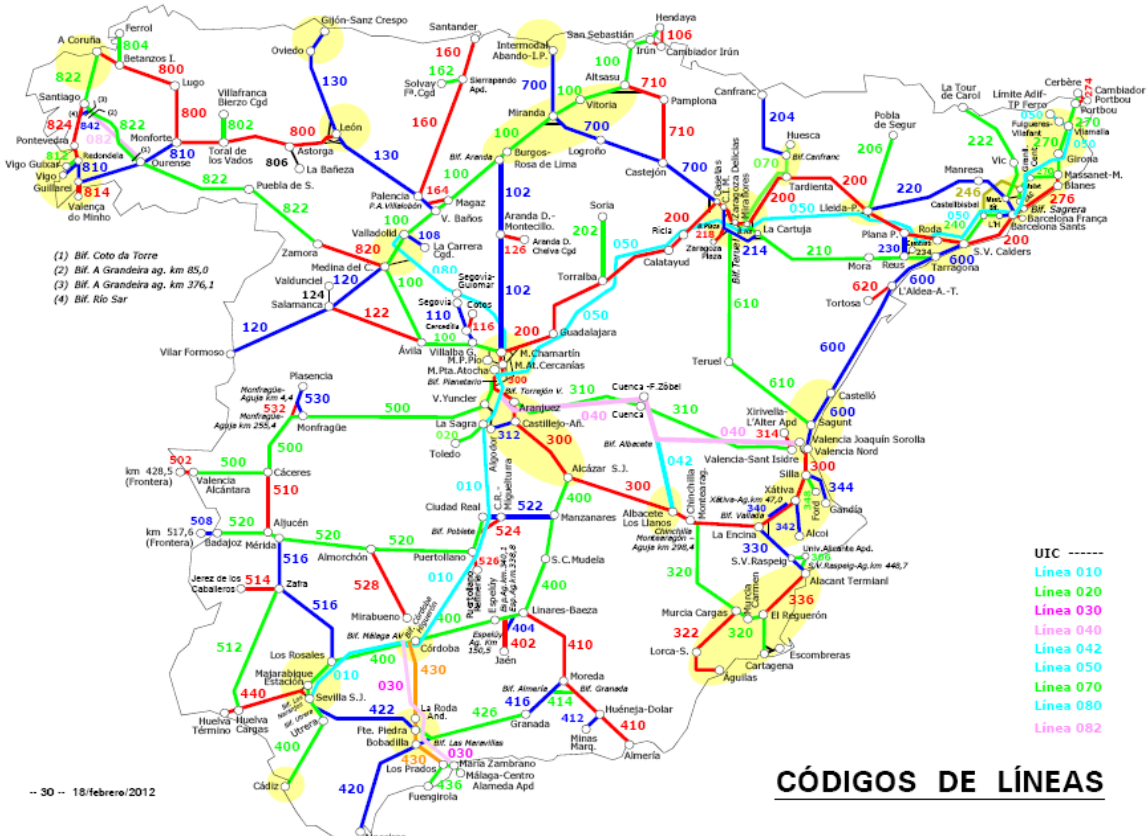

Fig. 2.2 Mapa de líneas de la red

(Mapa de Líneas de la red nacional con sus códigos, estructura de red)

Cada Línea se suele denominar con un nombre indicativo de su origen y destino final (Madrid-Hendaya) y tiene además un identificador propio (un número de tres cifras). En el origen se inicia la kilometración de la misma. Todos sus elementos (estaciones, pasos a nivel, elementos de señalización, túneles, viaductos, etc...) están perfectamente ubicados por su posición en el punto kilométrico (P.K.) correspondiente.

Cada Línea tiene definido un sentido PAR que es en el que circulan los trenes pares circulado obviamente los impares en sentido contrario.

A lo largo de la Línea se ubican además estaciones, cargaderos, apeaderos, apartaderos, bifurcaciones, agujas, cambiadores de ejes, etc.. con características muy diferentes entre sí, pero con una común que son los puntos donde nos interesa registrar la hora de entrada, salida o paso de los trenes, se denominan Estaciones de Paso o Puntos de paso, adoptaremos esta última denominación (Pps).

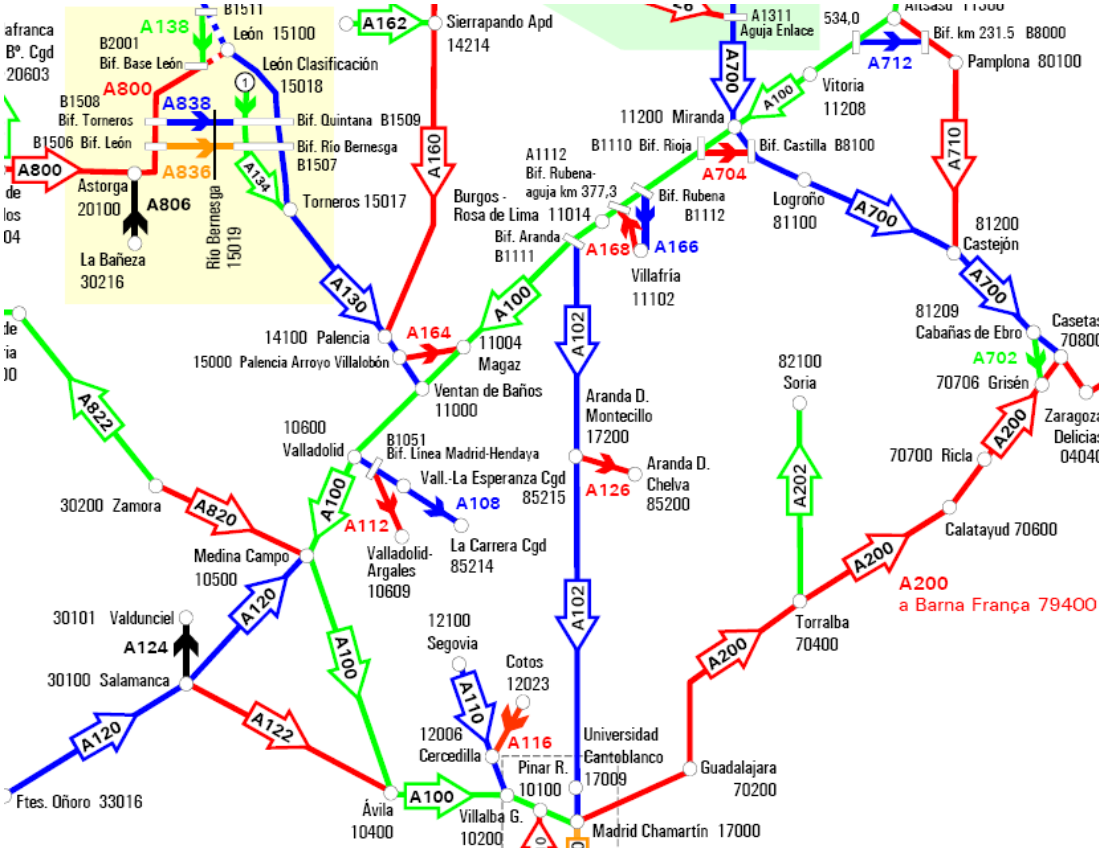

Fig. 2.3 Líneas y sentido de circulación de trenes pares

(Mapa detalle de líneas con su indicación de orientación sentido PAR)

Estos Pps pueden ser origen o destino de una o varias líneas, pueden ser puntos intermedios para una Línea y origen o destino para otra, de este modo la distribución geográfica de las diferentes líneas forma una red.

Ejemplos:

La línea 100: Madrid-Hendaya sentido IMPAR tiene una longitud de 641 Km. y más de 100 Pps.

La línea 704: Bif.Castilla-Bif.Rioja sentido IMPAR tiene una longitud de 2 Km. solo tiene dos PPS. Su origen y su destino, es un ByPass que une la línea 100 con la 700 evitando el paso por Miranda de Ebro.

La red geográficamente se divide en 7 zonas (0,1, ...6) y dentro de cada zona en **bandas** de regulación.

Fig. 2.4 Bandas representación esquemática

(Representación de una zona de regulación con su división en bandas)

Una banda es una parte de la infraestructura que puede visualizarse a la vez en una pantalla y generalmente permite ser controlada por un solo operador. Las bandas se identifican con 4 cifras. La primera es la indicativa de la zona donde se incluye. Cada banda permite visualizar y gestionar un número de entre 18 y 25 puntos de paso de una o varias líneas.

Una lista ordenada de Pps define el recorrido de un tren de origen a destino. Además podemos representar la posición de los trenes en el tiempo, definiendo las horas de entrada y salida de cada tren por sus puntos de paso.

1101 : ATOCHA CERCANÍAS - PINAR ROZAS

18000	**MT**	ATOCHA CERCANIAS
18003	**MD**	MÉNDEZ ÁLVARO
60004	**XC**	STA. CATALINA
18004	**DE**	DELICIAS
18005	**PM**	PIRÁMIDES
10000	**MP**	MADRID P. PIO
10002	**PO**	POZUELO
B1005	**B1**	BIF. CHAMARTÍN
10100	**PR**	PINAR ROZAS

1102 : PINAR ROZAS - CERCEDILLA

10100	**PR**	PINAR ROZAS
10105	**LK**	MATAS CLASIF.
10101	**LT**	LAS MATAS
10103	**TR**	TORRELODONES
10200	**VG**	VILLALBA DE GUAD.
10203	**EE**	EL ESCORIAL
12002	**XT**	ALPEDRETE
12004	**CM**	COLLADO MEDIANO
12005	**LG**	MOLINOS-GUADARRAMA
12006	**CE**	CERCEDILLA

Fig. 2.5 Ejemplo bandas de regulación de las cercanías de Madrid

Hay una serie de atributos de los Pps que son importantes en relación con los movimientos de los trenes por ellos. Por ejemplo el número de vías y las características de las mismas. Algunos ejemplos de esta interacción:

- Para que un tren de viajeros pueda efectuar parada comercial en una vía de un Pp, ésta debe estar dotada de anden.
- Un tren no puede estacionarse en una vía de un Pp si la longitud de la misma no es mayor que la del tren.
- En un momento determinado no pueden estar estacionados en un Pp más trenes que vías de estacionamiento existan en el mismo.
- Etc..

Entre un Pp y el siguiente hay un **Trayecto** de una determinada longitud. Sobre la relación de los trenes y los trayectos hay que establecer algunas consideraciones:

- Un tren que sale de un Pp está en el trayecto hasta que llega al Pp siguiente. El tiempo que tarda está ligado a la longitud del trayecto y la velocidad tipo del tren.
- Estos trayectos pueden tener una o dos vías (hay trayectos de más de dos vías pero no van a considerarse en este caso).
- Cada vía puede admitir uno o más trenes a la vez en el trayecto de la misma paridad (mismo sentido de circulación), pero obviamente en este caso un tren no puede llegar al siguiente Pp hasta que no haya llegado el que le precede.

- Una vía de un trayecto no puede admitir un tren en un sentido hasta que no esté libre de trenes circulando en sentido contrario.

2.3 Los Trenes

Podemos considerar los trenes como los elementos que se desplazan en el tiempo por la infraestructura, de un PP al siguiente por los correspondientes trayectos.

Cada tren tiene un identificador único de 5 cifras o dos letras y 3 cifras, además de otros nombres o alias que lo identifican mejor de cara al usuario.

Tiene un atributo que está relacionado con su velocidad y que según la longitud del trayecto determina el tiempo mínimo que tardará el tren en desplazarse de un Pp al siguiente, tambien la clase (mercancias, viajeros,...) y en su clase la categoria a la que pertenece.

Cada tren tiene un itinerario o recorrido previsto desde un Pp origen en una banda, hasta un Pp de destino en esa banda o en otra. En cada banda tiene una paridad según el acceso a la misma, lo que determina el orden en el que va a recorrer los Pps de esa banda hasta el Pp de salida o destino.

Ademas, un tren puede tener una vía predeterminada de estacionamiento y un tiempo de parada prevista en un Pp.

Otra característica del tren es su composición, es decir su locomotora o automotor, vagones, ejes, longitud, peso total en toneladas, etc..

Hoja de Ruta
MALLAS

adif

Documento válido para el dia: 14 Feb 13

Tren: 91521 T.L.E. Origen: ZARAGOZA-PLAZA Destino: LEON CLASIFICAC.

91521	T.L.E.	ZARAGOZA-PLAZA =>	BIF. RIOJA
91520	T.L.E.	BIF. RIOJA =>	MAGAZ
91521	T.L.E.	MAGAZ =>	LEON CLASIFICAC.

N.Comercial: TLE550M(INTERMODAL) Observac: TLE550M-TTE. EXCEP. 80521

Locomotora	Desde Estación	Hasta Estación	Peso	Tipo
LOC 269-	ZARAGOZA-PLAZA	LEON CLASIFICAC.	1580	100

CAT	Tipo de bloqueo	Sit Klm	Vel max	Dependencia Precaución	ASF	Tc Op	Pef	Horas Sal-Pas	Para Tec	Tmpo Conc	C. Rad
		004.5		ZARAGOZA-PLAZA.....	A			13:29			
		001.4	100	BIF.PZA AG K 1,4.....	A			13:36		7.0	
		000.0	060	BIF. PLAZA.....	A			13:38		2.0	(64)
		000.7	100	CIM-AGUJA KM 0,7.....	A			13:43.3		5.3	
		335.6	050	C.I.M. DE ZARAGO.....	A			13:47.3		4.0	
		330.0		UTEBO (APD).....	A			13:52.3		5.0	
		327.5		CASETAS.....	A			13:54		1.3	
		020.7		LA JOYOSA-TORRES (APD).....	A			13:58		4.0	
		024.8		ALAGON.....	A			14:01		3.0	
		029.9		CABAÑAS DE E.....	A			14:04.3		3.3	(69)
	BAB CTC	033.3		PEDROLA (APD).....	A			14:07		2.3	
		037.5		LUCENI.....	A			14:10		3.0	
		045.4		GALLUR (APD).....	A			14:15.3		5.3	
		055.3		CORTES NAVARRA.....	A			14:22		6.3	
		063.8		BUÑUEL (APD).....	A			14:27.3		5.3	
		067.4	100	RIBAFORADA.....	A			14:30		2.3	
		077.4		TUDELA NAVARRA.....	A			14:36.3		6.3	
		081.1		C. DE R.....	A			14:39.3		3.0	(99)
		089.4		C. DE R.....	A			14:46		6.3	
		093.7		CASTEJON.....	A			14:51		5.0	
		005.3		ALFARO.....	A			14:58	04	7.0	
		014.9		RINCON DE SOTO.....	A			15:09.3		7.3	
		027.8		CALAHORRA.....	A			15:18		8.3	
		040.9		LODOSA.....	A			15:26.3		8.3	
		045.1		FECULAS-NAVARRA.....	A			15:29.3		3.0	
		048.1	080	ALCANADRE.....	A			15:32		2.3	
		054.8		C. DE R.....	A			15:37		5.0	
		060.1	100	ARRUBAL.....	A			15:41		4.0	
	BAU CTC	063.4		AGONCILLO (APD).....	A			15:43.3		2.3	(62)
		067.5	090	RECAJO.....	A			15:47.3		4.0	
		076.4	100	LOGROÑO.....	A			15:54.3		7.0	
		085.3	095	KM. 85,340.....	A			16:01		6.3	
		091.5	080	FUENMAYOR.....	A			16:05	22	4.0	
		095.2	090	KM. 95.2.....	A			16:31.3		4.3	
		097.1		MATAGON (CGD).....	A			16:33		1.3	
		101.8	100	CENICERO.....	A			16:37.3		4.3	
		103.1	090	CENICERO-S ISIDR (APD).....	A			16:38.3		1.0	

Fig. 2.6 Hoja de ruta de un tren

(En la Figura 2.6 se muestra la Hoja de Ruta de un tren elaborada con los datos de su recorrido horario, incluye además informaciones de seguridad de tipos de bloqueo y velocidades máximas en trayectos)

2.4 Resumen de Objetos y Atributos descritos.

De estas descripciones anteriores podemos extraer e identificar las entidades y sus atributos que podrían ser útiles a nuestro proyecto y que describimos a continuación:

Tren (ID :Número de tren, Fecha, Nombre Comercial, Empresa, Producto, Tipo, Situación, Composición, Recorrido, Definición de recorrido)

Situación (Estación, Hora, Retraso, Motivos, E/S)

Composición (Locomotora, Vehiculos, Ejes, Longitud, Toneladas, P-S-B, Observaciones, T.E.)

Recorrido (Cod PP, Nombre PP, Vías – Trayecto, Estación, Hora Llegada, Hora Salida, E/S, Observaciones)

Hora Llegada (Retraso, Motivo)

Hora Salida (Retraso Motivo)

Definición de recorrido (Banda , Origen Banda , Hora Salida)

Infraestructura (Red, Linea, Banda)

Red (Líneas)

Linea (ID, Nombre, P.P.s)

Banda (ID, Nombre, PPs.)

PP (ID, Nombre, Abreviatura, P.K. Situación, Vías, Trayectos)

Via (ID, Longitud, Andén (S/N))

Trayecto (Longitud, Velocidad Max, Capacidad)

2.5 Conclusiones

Esta información está extraída de un análisis inicial desde el punto de vista de usuario de otras aplicaciones ya desarrolladas que manejan estos datos. Estas aplicaciones son de uso y propiedad de la empresa ADIF y se relacionan en el apartado de referencias y bibliografía. Se ha extraído y resumido en función de los objetivos del proyecto, malla de trenes e incompatibilidades.

Esta definición de entidades y atributos no es completa y está sesgada para servir de guía en la definición de los objetos y entidades necesarios en el proyecto. Aunque la aplicación a desarrollar no tiene que ser igual a las descritas, va a representar la misma realidad de una forma similar y por tanto parte de unos datos similares, siendo conveniente además lograr una aplicación con el mayor grado de compatibilidad posible con las aplicaciones mencionadas en el parrafo anterior. Esta compatibilidad es necesaria para poder trabajar por ejemplo en simuladores pero con datos de infraestructura y trenes obtenidos de archivos de texto que han sido a su vez elaborados con consultas sobre las bases de datos que manejan estas aplicaciones.

Capítulo

3

Especificaciones y Requisitos

3.1 Introducción

Nuestro proyecto tiene fundamentalmente dos objetivos:

- **Representar la malla gráfica de trenes**
- **Detectar la incompatibiliades de los trenes en la misma**.

Las especificaciones de estos dos objetivos van a determinar a su vez la sintaxis de los elementos necesarios para ello. En los apartados siguientes se enumeran para cada uno de estos abjetivos, los requisitos especificos sobre qué debe obtenerse y cómo, así como la forma de presentarlos usuario.

3.2 Malla de Trenes

El objetivo es una representación gráfica del recorrido de los trenes en el tiempo.

Se deben poder visualizar los tramos de recorrido de los trenes independientemente por bandas, a selección del usuario.

El tamaño de los objetos graficos debe ser referenciado al de la pantalla, de modo que la malla pueda visualizarse correctamente independientemente del tamaño de la pantalla o de la ventana, aunque normalmente ésta estará maximizada.

Se deben distinguir por colores los trenes en función de la categoría o producto a que pertenecen.

En el eje de abscisas se representa el tiempo, la hora del día en minutos. En el eje de ordenadas se representa un recorrido de Puntos de Paso. Es importante que la separación vertical de estos puntos entre sí sea proporcional a la distancia real que existe entre ellos para que de este modo podamos apreciar la diferente velocidad de los trenes en función de la inclinación de sus líneas en el gráfico.

Los puntos del plano representan así la situación de un tren en un determinado momento. Para cada tren, un punto representa su hora de entrada o salida de un Pp y la unión de cada dos puntos dibuja un segmento y la línea formada por todos los segmentos representa el recorrido del tren. Como ya hemos indicado un mayor o menor ángulo de esta recta con el eje de abcisas nos indica una mayor o menor velocidad de desplazamiento del tren.

El conjunto de líneas de los trenes nos presentan gráficamente con sus aproximaciones, paralelismo o cruces, las interacciones entre las marchas de los diferentes trenes en los puntos de paso o trayectos.

En una malla vamos a representar un conjunto de Puntos de Paso que vamos a denominar Banda, así ya tenemos los tres componentes principales de la Malla: Banda, Horario, Trenes.

Banda: Sucesión de Puntos de Paso y distancias entre ellos o trayectos.

Horario: Sucesión de puntos que representan una hora del día de 0:00 a 23:59.

Trenes : Un tren está definido por su recorrido, es decir, una sucesión de puntos de paso y la hora de entrada y salida de cada uno.

3.3 Incompatibilidades

Hay dos grupos de incompatibilidades que la aplicación debe detectar:

1. Las de los trenes en las paradas, es decir cuando se prevee una parada de un tren en un punto de paso, comprobar que la vía elegida es compatible con las características del tren.
2. Las de los trenes coincidentes en un trayecto en un mismo intervalo horario, cuando coinciden dos o mas trenes en un mismo trayecto y vía debemos comprobar que sus sentidos de circulación son compatibles y además lo son con las características del trayecto.

Incompatibilidades del primer tipo a detectar:

- Parada de un tren de viajeros en una vía sin andén.
- Parada de un tren en una vía más corta que su longitud.
- Parada solicitada a un tren en una vía inexistente.

Incompatibilidades de segundo tipo a detectar:

- En una vía de un trayecto, en un intervalo de tiempo, llegan a coincidir más trenes circulando en el mismo sentido que la capacidad máxima asignada al trayecto.
- Supuestamente dos trenes circulando por la misma vía y en sentido contrario deben coincidir, según sus horarios, en un determinado trayecto.

- Supuesto adelantamiento de trenes. Es el caso un tren que, incorporandose a un trayecto con posterioridad a otro del mismo sentido, tiene previsto llegar antes que él.

3.4 Otros Requisitos

La aplicación carga por sí misma de ficheros de texto los datos necesarios que conforman la infraestructura de bandas, puntos de paso, trayectos y vías, también los que conforman los datos de trenes con su recorrido y paradas previstas.

La aplicación comprueba y avisa de errores en los ficheros de texto en cuanto a inexistencia de ficheros, imposibilidad de apertura o errores en los datos en cuanto a tipos, de los producidos en general por errores en la disposición de los atributos de cada instancia de objeto a cargar, generalmente errores de texto.

No se comprueba la consistencia de datos en cuanto a índices etc. Es decir las relaciones entre los objetos se van estableciendo según las coincidencias de los valores de los identificadores de cada objeto. La aplicación no va a efectuar después ninguna comprobación sobre los objetos que no tiene correspondencia, por ejemplo vías sin adjudicar a un punto de paso o puntos de paso sin adjudicar a ninguna banda. Se supone que los datos de entrada son elaborados a partir de bases de datos existentes que aseguran la consistencia de los mismos.

La aplicación debe ser capaz de estructurar toda la amplitud de la infraestructura, aunque sólo se represente una parte de la misma. Es decir se debe asegurar que la aplicación sitúa correctamente todos los trayectos de enlace entre las diferentes bandas, es decir las salidas y entradas a distintas bandas desde diferentes puntos, los puntos de paso con multiples trayectos de salida, bifurcaciones, etc. Esto es necesario porque aunque normalmente vamos a cargar datos de infraestructura sólo de dos o tres bandas, la aplicación debe ser capaz de admitir todas las que se precisen y con toda la casuística que se da en la red completa.

La infraestructura es la que asegura que se pueda completar el recorrido de los trenes de cualquier origen a cualquier destino. Los trenes tienen establecido un origen, un destino, una hora de salida y un recorrido marcado únicamente por los puntos en los que puede haber mas de una opción de trayecto a tomar hasta su destino. Por tanto, con los datos de la infraestructura y de cada tren la aplicación debe ser capaz de generar el

horario completo de recorrido de cada tren con todas sus horas de entrada y salida por cada uno de los puntos existentes en su recorrido completo.

3.5 Requisitos de Interfaz

La aplicación debe implementar como mínimo las siguientes ventanas o formularios:

Una ventana Principal donde se representa la malla gráfica de trenes, donde el usuario puede seleccionar la banda y el tramo horario a visualizar, además desde esta ventana se puede acceder a las restantes.

Una ventana de mensaje o avisos para activar la Carga de datos y proceder a la generación de horarios antes de que los datos sean presentados en la malla gráfica.

Una ventana para visualizar la Infraestructura. Desde la misma el usuario puede comprobar la infraestructura cargada en el sistema. Para ello, debe presentar listas de selección y visualización de elementos relacionados para que el usuario pueda comprobar además de su contenido su consistencia.

Una ventana para visualizar los Trenes, comprobando el usuario desde la misma los trenes cargados y la corrección de los recorridos horarios generados. Como en el caso anterior, debe presentar listas de selección para visualizar todas las instancias cargadas y comprobar su consistencia.

Una ventana para visualizar las Incompatibilidades de Vía y otra para las de Incompatibilidades de Trenes. En ellas, el usuario debe poder seleccionar en cada caso qué tipo de incompatibilidad quiere buscar y visualizar seleccionando cada una todos sus datos, de modo que el usuario tenga una información completa de la incompatibilidad para hacer las comprobaciones oportunas.

3.6 Conclusiones

Se han enumerado en este capítulo una amplia serie de requisitos de distinto tipo, pero que podemos resumir en dos grupos. Los que son fundamentales para los objetivos de la aplicación, definen cómo representar una malla gráfica de trenes y cómo detectar los casos

de incompatibilidades descritos. Y otro grupo, el de los que especifican de dónde partimos y cómo comprobamos que hemos llegado a los objetivos. En el primer caso todos los que se refieren a los datos de entrada y en el segundo de la interfaz gráfica.

Capítulo

4

Datos de Entrada

4.1 Introducción

Definidos los objetivos principales y otros requisitos de la aplicación a desarrollar en este proyecto y basándonos en los análisis de datos sobre infraestructura y trenes realizadas en el Capítulo 2, podemos ya hacer un primer análisis de los objetos de datos necesarios, sus características y funcionalidades.

En los siguientes apartados vamos a describir nuevamente el modelo del Capítulo 2, pero ahora más enfocado a nuestro objetivo. A la vez subrayanos las palabras que posteriormente nos servirán para definir un objeto, atributo o función.

4.2 Datos de la Infraestructura

Para nuestros objetivos podemos definir la infraestructura como una serie de *Banda.* Cada Banda esta definida por un *Identificador* único y se compone de una serie ordenada *de Puntos de Paso* (PPs)

Un *Pp* tiene un *Identificador* único, puede tener también uno o más *nombres* para su mejor identificación por los usuarios y está conformado fundamentalmente por un conjunto de *Vías* y uno o varios *Trayectos* de salida

Cada *Vía* tiene *un identificador* único en el Pp, un *nombre* para el usuario y dos atributos, la *longitud* y si tiene o no *andén.*

El *Trayecto* va asociado al Pp de acceso al mismo y es a su vez el acceso de los trenes al Pp siguiente. Como atributos necesarios para generar el *recorrido horario* del *tren*, podemos definir la *longitud*, *velocidad máxima*, *tipo* (vía única o doble) y *capacidad*.

En resumen cada banda tiene una serie de puntos de paso, los puntos de paso tienen sus vías y como mínimo un trayecto de salida.

También se define una clase de *Puntos de Paso en Banda*, que se corforma con la posición de los Pps en la banda en función de la distancia de sus trayectos. Esta posición se aporta a los horarios de los trenes para ubicar los puntos en el gráfico.

4.3 Datos de los Trenes

Un *Tren* se define con un *Identificador* único, un *nombre* anexo opcional, su *clase* o producto (mercancias, viajeros, otros), su *paridad*, que indica el sentido en que recorre los pps de la banda, el *tipo* que indica su velocidad maxima, su banda y Pp de *origen* y su *hora de salida.*

Además se define para cada tren su *recorrido*, que es una *lista* ordenada de *bandas* que recorre el tren, con su *paridad* en cada una, y los *Pp de entrada* y *Pp de salida* en esa banda. De esta forma, sin listar exhaustivamente todos los Pps que recorre el tren, podemos generarlos, puesto que el tren en función de su paridad irá del punto de entrada en una banda hasta el de salida de la misma.

También la *lista* de solicitudes de *parada* del mismo(*Pp*, *tiempo de parada*, *vía*).

4.4 Recorrido Horario

Una vez definida la infraestructura y los trenes, es preciso generar el recorrido completo de los mismos con las horas correspondientes. Es decir, su *horario*.

El horario de un tren debe incluir todos los datos necesarios para dibujar las líneas de sus posiciones en la malla gráfica. Cada elemento de la lista de horario incluye: la *banda*, el *Pp*, el *trayecto*, el *tren* y su *paridad*, la *ubicación* del Pp, la *vía de Pp* y *vía de trayecto*, la *hora de entrada* y *hora de salida* de cada Pp, *tiempo de marcha* y de *parada*.

Generar el recorrido horario completo implica que partiendo de la banda y Pp de origen, y en función de su paridad el tren, debe recorrer todos los *PPs* de esa *banda* hasta el que figura como Pp de salida. Se continua con la banda siguiente hasta terminar en la última banda de la lista y el PP de salida. Se genera así una *lista* ordenada de todos los *PPs y trayectos que debe recorrer el tren*. Cada uno de estos PPs define como atributos la *ubicación*, *vía de paso*, la *hora de entrada*, el tiempo de *parada* y la *hora de salida*, y cada *trayecto* define el *tiempo de marcha* la *vía de trayecto*.

Resumido, el algoritmo para generar el recorrido sería:

1) Para cada tren hasta el final de la lista de trenes
    - Inicializamos un Pp Actual, el de inicio, y Hora Actual, la de salida del tren, que mantenemos actualizados en cada paso.

2) Para cada recorrido del tren en cada banda
    - En función de la paridad del tren obtenemos el Pp Siguiente y el Trayecto (*)
    - Calculamos y obtenemos los datos de horario en ese punto (**)
3) Continuamos hasta llegar al Pp de salida de banda en ese recorrido.
4) Pasamos al siguiente recorrido hasta llegar al Pp destino.

(*) En función de la paridad del tren, el recorrido en banda va de un Pp al siguiente o al anterior, el trayecto es el que enlaza esos dos puntos. Por convención los trenes pares siguen el orden de los Pps en banda y el trayecto es el de ese Pp, para los impares a la inversa y el trayecto es el del Pp siguiente.

(**) Con los datos del Pp podemos obtener los datos de la parada si la hay, si no asumir unos por defecto. Con estos datos y los del trayecto calcular tiempos de marcha y parada y obtener horas de llegada, salida y otros datos necesarios para definir el horario en ese punto.

4.5 Conclusiones

En las clases de datos está representado el aspecto de la realidad que necesitamos para nuestro objetivo, hay muchos aspectos de la realidad que podemos capturar añadiendo atributos, funciones o relaciones a nuestras clases, pero de momento nos interesan los que pueden hacer la representación de la malla de trenes y los que nos aporten la información necesaria para los casos de incompatibilidades que la aplicación debe detectar y mostrar.

Hemos definido la infraestructura. Ccomo una serie de bandas, que constan, de un grupo de puntos de paso unidos por trayectos y cada uno con sus correspondientes vías. Y los trenes, que se mueven por los puntos de paso y los trayectos, siguiendo un recorrido determinado desde su origen a destino y con unas horas definidas de entrada o salida de esos puntos.

Capítulo

5

Malla Gráfica de Trenes

5.1 Introducción

Una vez que hemos analizado los datos y su organización, podemos realizar una descripción de la metodología a seguir para presentar la malla gráfica de trenes a partir de los datos de infraestructura y los trenes con sus horarios correspondientes.

Tal y como hemos definido en los objetivos y requisitos de la aplicación, se trata de representar un gráfico que muestre la posición de los trenes en el tiempo, que pueda visualizar diferentes bandas de la infraestructura y en diferentes intervalos de tiempo.

5.2 Objetos Necesarios

Para realizar esta representación gráfica, se consideran necesarios en un primer análisis los siguientes elementos gráficos:

1.- Lista desplegable de bandas.
2.- Lista desplegable de horas.
3.- Lista de Puntos de paso de una Banda.
4.- Grafica de horas.
5.- Gráfico de malla de Trenes.

1.- Lista desplegable de bandas.

Se despliega la lista de Bandas cargadas en la aplicación. Al seleccionar una el usuario obtenemos el identificador de Banda y con este dato se redibuja la lista de puntos de paso y la malla de trenes.

2.- Lista desplegable de horas.

Despliega una lista de la horas del día. Al seleccionar el usuario una hora obtenemos la hora inicial del tramo horario a representar. Con este dato se redibuja la lista de horas y la malla de trenes.

3.- Lista de Puntos de paso de una Banda

Es una lista vertical de los nombres de los puntos de paso de la banda ordenados y ubicados de forma que su separación en la lista es proporcional a la distancia del trayecto entre ellos (representa la ordenada del gráfico).

4.- Grafica de horas

Se presenta una lista horizontal de las horas del tramo horario seleccionado por intervalos de 5 ó 10 minutos (representa las abscisas del gráfico).

5.- Malla de trenes

Es el gráfico propiamente dicho, un rectángulo dividido en cuadrículas por líneas horizontales y verticales para que el usuario identifique mejor los puntos del plano representados. Las abscisas son las horas del día aproximadas a minutos, las ordenadas son las posiciones relativas de los puntos de paso en relación con la longitud de los trayectos entre ellos.

Sobre esta malla se dibujan las líneas que representan las posiciones de los trenes en los trayectos y puntos de paso en los distintos momentos del día.

5.3 Operativa General

Al cargar la ventana o cada vez que se produce una actualización por selección de banda o de hora, se realizan las operaciones siguientes:

- Redibujar la lista de Puntos de Paso
- Redibujar la lista de presentación de horas
- Redibujar la malla de trenes

Al cargar o modificar el tamaño de la ventana, se debe ajustar el tamaño de los objetos gráficos el altura, anchura o ambas para ajustarse a la ventana. El sistema mantiene estas medidas de los objetos como escala para dibujar en ellos.

Además, para poder identificar correctamente puntos de dibujo en los objetos se definen funciones de cálculo de posición vertical y horizontal y cálculo de puntos, que en función de los datos de entrada y el tamaño del objeto gráfico calculan un factor de escala y el punto del objeto gráfico que corresponde a esa entrada

5.4 Redibujar la Lista de Puntos de Paso

Datos de entrada:

Una lista ordenada de puntos de paso y su ubicación dentro de la banda en metros. Es decir el primer punto estará en 0 y el último en la suma total de las longitudes de los trayectos que unen los puntos de la banda.

a) Para cada punto de paso de las lista, obtenemos su identificación, su nombre y su ubicación.
b) Con su ubicación y la ubicación máxima de la banda, obtenemos su posición gráfica.
c) Dibujamos el texto Nombre Pp. en la posición grafica obtenida.
d) Dibujamos una línea en la malla en esa posición, todo lo ancho de control Malla.
e) Así hasta fin de lista.

Para obtener la posición grafica en pixels

posición grafica = (ubicación de punto en la banda en metros. X tamaño del control en pixels) / longitud total de la banda en metros.

5.5 Redibujar la Lista de Presentación de Horas

Datos de entrada:

Una hora inicial y el intervalo horario que presentara la malla, conocemos por tanto la hora final.

a) Desde la Hora Actual = hora Inicial.
b) Obtenemos la posición gráfica
c) Dibujamos el valor texto de la hora actual en la posición gráfica
d) Dibujamos una línea vertical en esa posición horizontal de arriba abajo del control Malla
e) Hora Actual ++ 5 min.
f) hasta Hora Actual == hora Final

Para obtener la posición grafica en pixels.

posición Gráfica = (Tamaño del intervalo en seg. x tamaño de control en pixels)/ Valor hora actual en seg.

5.6 Redibujar Malla de Trenes

Datos de Entrada:

La lista de Trenes cargados en la aplicación, cada uno con su lista de horarios. En cada horario de ese tren debemos tener:

El identificador de banda, la identificación del punto de paso, la ubicación del Pp en la banda, nombre del tren según su paridad, hora de llegada, hora de salida, todos ellos referidos a un punto de paso.

También el gráfico es para una banda seleccionada (banda Actual) y una hora del día (banda Horaria).

El modo de operar es recorrer para cada tren su lista de horarios y si el horario es correspondiente a la banda actual y la hora de llegada está en el intervalo de banda horaria elegida, ese horario establece dos puntos del grafico: el de llegada, que se une con el de salida anterior, y el de salida, que se une con el de llegada y sirve para unirse con el de llegada siguiente. Sus coordenadas son las horas como abscisa y la situación en banda del Punto de paso como ordenada. Además en el primer punto hay que representar el nombre del tren según su paridad en esa banda.

Algoritmo aproximado:

a) Para cada tren

b) Para cada horario de la lista hasta fin de lista

c) Si banda = banda Actual {Punto2 (hLleg, PPPosBanda)}

d) Si es primer punto {Dibujar Nombre Tren(Punto2, tren Paridad)}

e) Si no { Línea (Punto Grafico 1(Punto1), Punto Grafico 2(Punto2)}

f) Punto1 = Punto2

g) Punto2 (hSld, PPPosBanda)

h) Línea (Punto Grafico 1(Punto1), Punto Grafico 2(Punto2)

i) Punto1 = Punto2

j) Hasta el final del horario del tren

Entendemos que hay que definir las siguientes funciones:

La función Punto Grafico (int posición 1, int posición 2) que devuelve un punto en pixels, relacionado con la escala gráfica del objeto donde se va a representar. Esta función usaría las funciones mencionadas anteriormente para calcular las posiciones vertical y horizontal, con un factor de escala en función del valor entero máximo a representar y el tamaño del objeto gráfico.

La función Dibujar Nombre Tren (int punto, string nombre tren) hace uso de la función anterior para calcular un punto gráfico, que se modifica para presentar posteriormente el nombre del tren según su paridad en ese punto, próximo por encima o por debajo también en función de la paridad del tren, del primer punto de aparición del tren en esa banda.

Además antes para cada tren según su clase o producto se obtendrá un color gráfico que se utilizará para representar el nombre del tren y la línea correspondiente.

5.7 Conclusiones

La malla de trenes es una representación gráfica en un plano de dos dimensiones. La dimensión horizontal, la abscisa, la obtenemos del tiempo, las horas del día en minutos. La dimensión vertical, la ordenada, la obtenemos de las ubicaciones de los puntos de paso en la longitud total de la banda. Cada punto del plano indica por tanto una posición en el tiempo, así vamos posicionando las líneas de recorrido de cada tren según su horario.

Después el problema a resolver, es de escalas y posiciones relativas en los elementos gráficos. Partimos de unos valores enteros dentro de un intervalo, a representar en un objeto gráfico con una posición y un tamaño determinado.

Capítulo

6

Detección de Incompatibilidades

6.1 Introducción

A continuación se describen los algoritmos y funciones necesarias para la detección de las incompatibilidades que se han definido en los objetivos de la aplicación.

Por convenio se adjudica un número e incluso una letra a cada tipo de incompatibilidad. Esta misma nomenclatura se mantiene en la explicación de las funciones en el código.

En estos algoritmos aparecen las denominaciones de datos que se han descrito en el Capítulo 4 e incluso alguna referencia a funciones que se describen posteriormente en el capítulo siguiente al describir el diseño de la aplicación.

6.2 Incompatibilidades de Vía

El objetivo es detectar errores o incompatibilidades en las solicitudes de paradas de trenes en cuanto a la vía solicitada.

Datos de entrada: Solicitudes de parada para un tren en un punto determinado y en una vía de ese punto.

Salida: Lista de errores detectados en las solicitudes de parada que pueden ser de tres tipos:

0.- Error por vía inexistente.

1.- Error, vía sin anden para un tren de viajeros.

2.- Error, vía sin longitud suficiente para ese tren.

Procedimiento:

- Para cada tren se comprueba cada una de las paradas solicitadas.
- Para cada parada se obtiene el Punto de Paso de la misma y la vía solicitada en ese punto.

- Se obtiene la vía correspondiente y se comprueba si hay errores:

0.- Si la vía devuelta es 0 (inexistente)

1.- Si el tren.producto es de viajeros (0, 1, 2) y la vía devuelta no tiene anden

2.- Si la vía.longitud es menor que tren.longitud.

Para su implementación, se crea la clase Incompatibilidad de Vía, para recoger todas las incompatibilidades detectadas, con los atributos correspondientes que identifican al Tren, el Punto de paso, la vía, el tipo de incidencia y además de las funciones propias de las gestión de la clase se implementan las funciones para obtener o bien todas las incidencias a la vez o seleccionadas por tipo.

Para la presentación al usuario, se diseña un formulario que presenta la lista de incidencias obtenidas y al seleccionar cada una de ellas presenta las características del Punto de Paso, sus vías y el tren con sus características, además de una descripción de la incidencia.

Permite a su vez elegir una vía de las disponibles en el punto de paso y modificar la parada para corregir el error.

6.3 Incompatibilidades de Trenes en Trayecto

Una incidencia de esta clase se produce en un trayecto cuando coinciden en el mismo, dos o más trenes en un intervalo de tiempo determinado.

Podemos distinguir tres tipos, que para continuidad con las incidencias de vía las vamos a numerar como 3, 4 y 5. Vamos a analizar cómo y por qué se produce cada una de ellas.

3.- **X Cruce de Trenes**: Se produce cuando coinciden en un mismo *trayecto* y en *un tramo horario* determinado dos *trenes* por la *misma vía* de sentido contrario *(paridad* distinta), es decir un *tren* sale de un *punto* al trayecto antes de que haya llegado otro *tren* que salió antes del *punto* contrario de acceso al trayecto.

4.- **C Exceso de Capacidad de Trayecto:** Se produce cuando en un *trayecto* determinado coinciden varios *trenes* por la misma *vía* y del mismo sentido (igual *paridad*)

de modo que en un *tramo horario* coinciden más *trenes* en el *trayecto* que la *capacidad* del mismo.

5.- **A Adelantamiento en plena Vía:** Se produce cuando en un *trayecto* determinado *coinciden* dos *trenes* por la misma *vía* y del mismo sentido (igual *paridad*) y el tren que accede más tarde al trayecto tiene previsto salir de él antes que el tren que le precede, es un hipotético adelanto por la misma vía.

En las descripciones aparecen subrayados los elementos que se precisan para detectar este tipo de incompatibilidades:

Trayecto, Puntos extremos, Trenes, Vía, Paridad de los trenes, Capacidad del trayecto y Tramos Horarios.

Para poder buscar este tipo de incompatibilidades, definimos primero la clase intermedia que recoge en sus atributos los datos necesarios a contrastar para localizarla, llamamos a esta clase Ocupación De Trayecto, de modo que una instancia de esta clase con sus atributos (banda, trayecto, Pp Inicial, Pp Final, tren, paridad, vía, hora de entrada, hora de salida) se define el intervalo horario que un tren ocupa una vía de un trayecto determinado.

Las instancias de esta clase las obtenemos de los horarios de los trenes. De cada horario obtenemos la banda, trayecto, Pp Final, paridad, vía y hora de salida del trayecto, que es la de llegada a ese punto, el Pp Inicial y la hora de entrada en el trayecto son el punto y la hora de salida del horario anterior del tren.

Con las horas de entrada (he) y salida (hs) tenemos el intervalo horario que cada tren está ocupando el trayecto por esa vía.

Para los casos 3 y 4, los intervalos de ocupación de trayecto deben solaparse. Es decir si llamamos tren 1 al primero en acceder al trayecto, hay un **solape horario** cuando la hora de entrada del segundo tren es mayor que la del primero y menor que la hora de salida de este del trayecto: **he2 > he1 & he2<hs1 (Ver figura 6.1)**

Fig. 6.1 Esquema de solape horario

Para el caso 5 además debe existir **inclusión de horarios**. Es decir el intervalo horario de un tren está incluido en el del otro, como antes si tren 1 es el primero en acceder al trayecto: **he2 > he1 & hs2 < hs1 (Ver figura 6.2)**

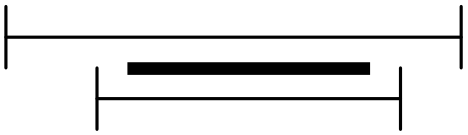

Fig. 6.2 Esquema de inclusión de horarios

De este modo tenemos una incompatibilidad de trenes cuando sus ocupaciones de trayecto cumplen las condiciones establecidas para cada caso:

Tenemos una incompatibilidad cuando dos instancias de Ocupación De Trayecto 1 y 2 cumplen:

Caso 3.- (trayecto1 == trayecto2) & (vía1 == vía2) & (paridad1 **¡=** paridad2) & (solape Horario).

Caso 4.- (trayecto1 == trayecto2) & (vía1 == vía2) & (paridad1 **==** paridad2) & (solape Horario).

Caso 5.- (trayecto1 == trayecto2) & (vía1 == vía2) & (paridad1 **==** paridad2) & (inclusión Horario).

Definimos la clase Incompatibilidad De Trayecto que tiene como atributos (banda, trayecto, Pp Inicio, Pp Final, tipo Incidencia, hora Inicial, hora Final) y una lista de

Ocupaciones De Trayecto implicadas en esta incompatibilidad, que en los casos 3 y 5 serán 2 y en el caso 4 pueden ser varias.

Para generar las instancias de Incompatibilidad De Trayecto recorremos la lista de Ocupaciones De Trayecto y:

Casos 3 y 5:

- Para cada instancia de la lista comprobamos con cada una de las que la siguen en la lista hasta el final si se cumplen las condiciones para cada caso y:
- En caso afirmativo, creamos una nueva instancia con los datos de trayecto, banda, Pp Inicial, Pp Final que son comunes y la hora Inicial será la menor de las dos Ocupaciones De Trayecto y la hora Final la de la mayor.
- Agregamos las Ocupaciones de Trayecto a la lista.

Caso 4:

- Para cada instancia de la lista creamos una instancia de Incidencia de Trayecto con las horas inicio y fin de esta ocupación de trayecto y la agregamos a su lista, comprobamos después con cada una de las que le siguen en la lista si se cumplen las condiciones de caso.

- En caso afirmativo agregamos esta ocupación de trayecto a su lista y modificamos si procede la hora Inicial con la menor y la hora Final con la mayor.
- Al terminar tenemos una posible Incompatibilidad De Trayecto si el número de trenes que ocupan trayecto en esas condiciones y en ese intervalo es superior a la capacidad del trayecto, si es así guardamos la incidencia. Si no la desechamos.

Ver código en el módulo de Incompatibilidades, ver clases y funciones.

6.4 Conclusiones

La detección de incompatibilidades de vía se basa en enfrentar, para cada parada, las características de las vías del punto de paso (existencia, longitud, andén), con las de tren (producto, longitud) y hallamos los casos en los que la parada no es compatible.

Las otras incompatibilidades, solo pueden existir en las coincidencias en trayecto de los trenes, es decir dos o más trenes comparten trayecto y tramo horario. Buscamos primero todas estas coincidencias y comprobamos posteriormente, si por el tipo de trayecto y sentido de los trenes, se da alguno de los casos incompatibles.

Capítulo

7

Diseño de la aplicación

7.1 Introducción

De acuerdo con las especificaciones de la aplicación, la descripción de datos y de los principales algoritmos y funciones, se describen a continuación los objetos de la aplicación con las denominaciones que aproximadamente tendrán después al implementarse en código C++.

Se describen fundamentalmente las clases de datos con los tipos de sus atributos y las relaciones entre ellas, las funciones descritas en los Capítulos 5 y 6 se implementan tal como se describen en los mismos, apoyándose en las clases y funciones que se describen a continuación.

7.2 Distribución Modular

Para enfocar ya una aproximación al inicio del diseño, podemos identificar los siguientes módulos:

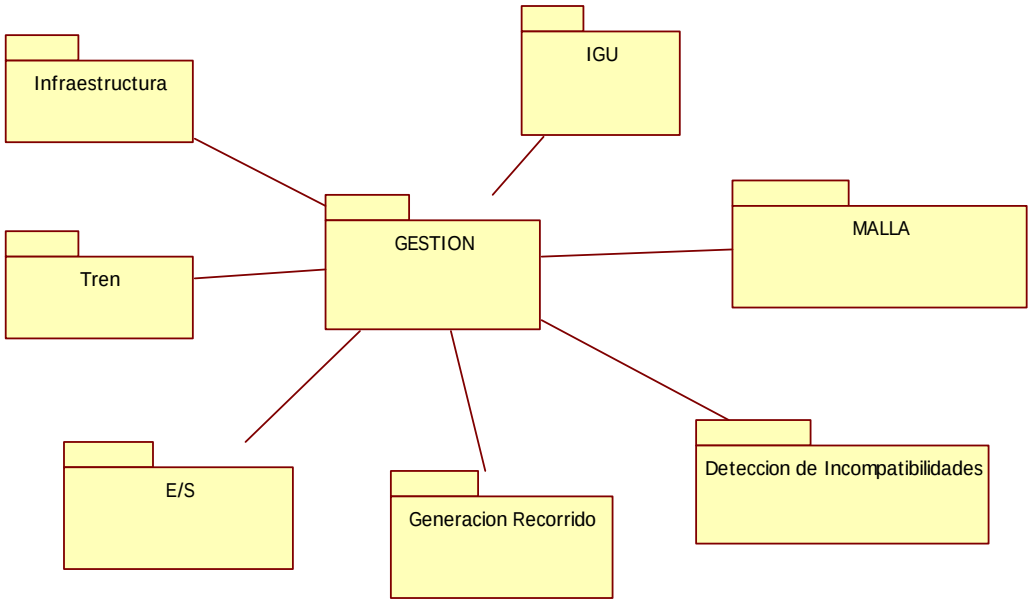

Fig. 7.1 Distribución modular

Esta distribución modular aunque es una primera aproximación que será ampliada o modificada posteriormente, nos debe servir de base para enfocar el diseño de una forma ordenada y compartimentada.

Describimos someramente a continuación cada módulo:

- INFRAESTRUCTURA: Incluye las clases que definen las bandas y sus componentes, los puntos de paso y todos los objetos que componen éstos, vías, trayectos, etc.. así como las funciones necesarias para su gestión.

- TRENES: Incluye las clases necesarias para definir los trenes con sus componentes y atributos así como las funciones necesarias para su gestión.

- E/S : Incluye las clases y funciones necesarias para cargar los datos de infraestructura y trenes de los ficheros correspondientes al inicio de la aplicación, así como para guardarlos en sus ficheros al finalizar o cuando proceda.

- IGU: La interfaz gráfica de usuario necesaria para el funcionamiento de la aplicación, principalmente para la introduución y visualización de datos por el usuario, la visualización de las mallas y la presentación de mensajes al usuario.

- GENERACIÓN DE RECORRIDO: Este módulo contiene la función a la que llamamos una vez definida la infraestructura y los trenes para generar la lista de puntos de paso del tren con sus horas de entrada, salida y vías y paradas. Este módulo podría estar integrado en el módulo trenes.

- MALLA: Módulo para realizar las operaciones necesarias con los datos de horario, infraestructura y trenes para representar la malla gráfica de trenes.

- DETECCIÓN DE INCOMPATIBILIDADES : Este modulo analiza los datos de infraestructura y trenes para detectar las incompatibilidades que se producen en el movimiento previsto de cada tren y de los trenes entre sí.

- GESTIÓN DE DATOS : Incluye las funciones necesarias para actuar de puente entre la IGU y el resto de los módulos, llamando desde el mismo a las clases y funciones que corresponda en cada caso.

7.3 La Infraestructura

La infraestructura consta de los siguiente objetos: ***Banda***, ***Punto de Paso***, ***Trayecto*** y ***Vía.***

Cada Banda agrupa una secuencia ordenada de Puntos de Paso, cada Punto de Paso tiene 0 o múltiples vías y uno o varios trayectos de acceso a otros Puntos de Paso.

Vía:

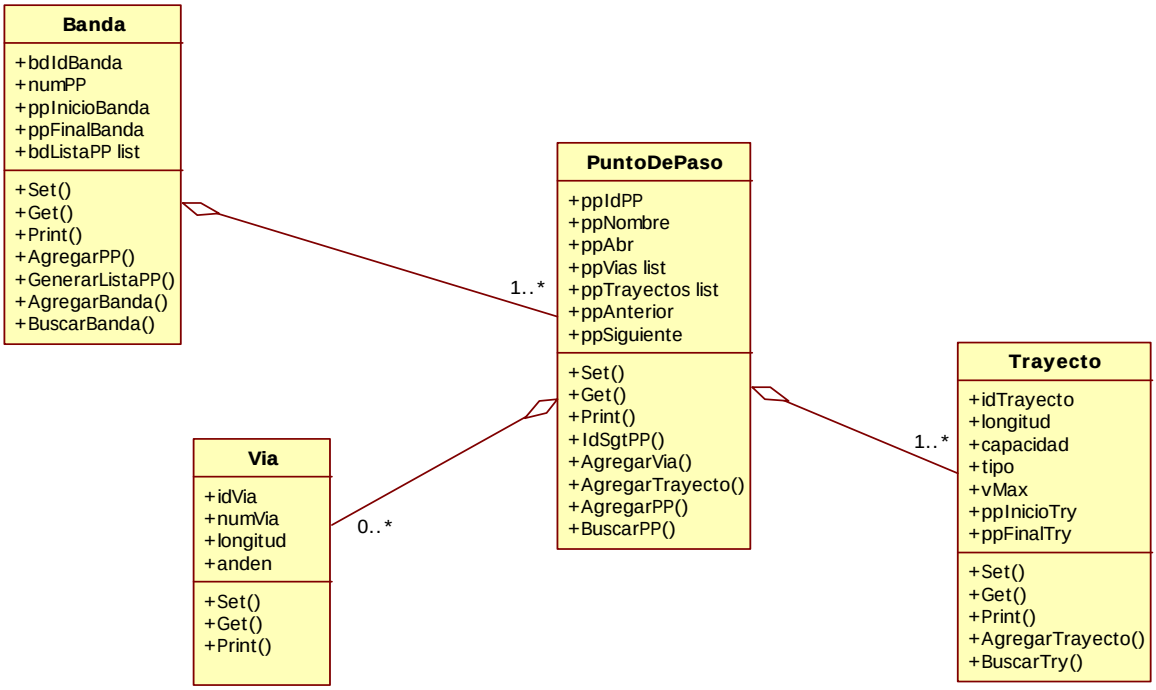

Fig. 7.2 Infraestructura objetos

Una vía se define con su ***identificador (Sting)*** que la relaciona con el PP al que pertenece, un ***número*** **(int)** que la distingue en el PP, su ***longitud*** **(int)** y si consta o no de ***andén (0-1).***

Se definen las funciones ***Set*** y ***Get*** y las de presentación de datos en pantalla (***print***).

Trayecto:

Se define con su ***identificador (string)*** relacionado con las abreviaturas de los PP que conecta, su longitud en ***kilómetros (double***), ***capacidad(int)*** de trenes por vía en el trayecto, el ***tipo(0-1)***. Si es vía única o doble, la ***velocidad máxima*** (int), y los PP de ***inicio (string)*** y ***final (string)***.

Se definen las funciones ***Set, Get***, ***Print*** y ***Buscar*** un trayecto por su identificador

Se define como miembro estático una ***lista de trayectos*** y las funciones para ***agregar*** trayectos y ***presentar*** la lista.

PuntoDePaso:

Se define son su ***identificador (string)***, un ***nombre (string)*** y su ***abreviatura (string)***, el ***número de vías***(int), sus ***vías(list)*** y ***trayectos (list)***, así como los PP ***anterior (string)*** y ***siguiente (string)***.

Se definen las funciones ***Set, Get***, ***Print*** y ***Buscar*** un punto de paso por su identificador, así como el ***Siguiente*** punto de paso.

Banda:

Se define con *su* ***identificador (string)***, el ***número de puntos de paso*** (int), sus Pps ***inicio (string)*** y ***final (string)*** y los ***puntos de paso (list)*** que la componen.

Se definen sus funciones ***Set, Get***, ***Print*** y ***Buscar, y*** las funciones para ***Generar la lista*** de puntos de paso.

Se diseña una infraestructura mínima para los objetivos del proyecto, pero ajustada a la realidad. Se tiene en cuenta la secuencia de puntos de paso para presentación de una banda, pero a la vez la posibilidad de generar el recorrido de los trenes accediendo de un PP a otros distintos a través de diferentes trayectos por cambios de línea, bifurcaciones, bypass, etc.

Para ello se tiene en cuenta que un PP puede formar parte de más de una banda y puede tener más de un trayecto de salida del mismo.

7.4 Los Trenes

El módulo de Trenes se define con los objetos siguientes: **Tren**, **Recorrido**, **Parada** y **Horario.**

Un tren tiene además de sus atributos una lista de paradas, una lista de recorrido con las sucesivas bandas por las que pasa con su PP de entrada y salida y con estos datos se genera el horario que es una lista de los PP desde su origen hasta su destino, sus horas de llegada parada y salida y los trayectos a los que accede con sus vías correspondientes.

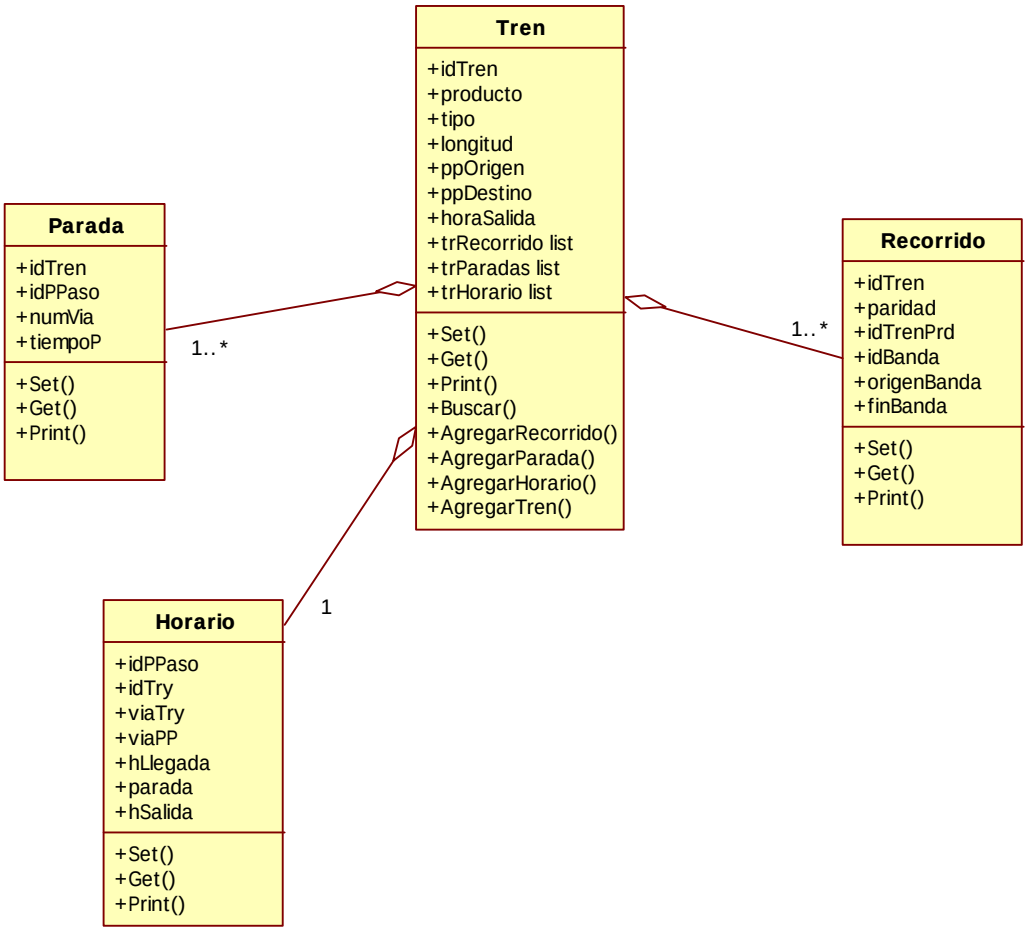

Fig. 7.3 Trenes objetos

Parada:

Se define con un ***identificador de tren (string)***, el ***identificador del punto de paso (string)***, la ***vía (int)*** donde se solicita la parada y el ***tiempo (int)*** en minutos que se solicita para la misma.

Se definen las funciones ***Set, Get*** y ***Print*** para presentación de datos en pantalla.

Recorrido:

Se define con el ***identificador del tren (string)***, la ***paridad (1 o 2)*** del tren en ese recorrido, el ***nombre del tren (string)*** según su paridad (los trenes que alternan paridad tienen

una numeración par y otra impar), el ***identificador de banda (string)*** y los puntos de paso de ***entrada (string)*** y ***salida (string)*** en esa banda.

Se definen las funciones ***Set, Get*** y ***Print.***

Horario:

Lo define un *identificador del punto de paso (string)*, la *vía (int)* de paso o estacionamiento, las horas de *llegada (int)*, *parada (int)* y *salida (int)*, el *identificador del trayecto (string)* de acceso al siguiente PP, la *vía (int)* del trayecto por el que accede.

Se definen las funciones *Set, Get* y *Print.*

Tren:

En tren queda definido por su *identificador (string)*, el *producto (int)* que define una clase o categoría del tren dos productos principales, viajeros (larga distancia, regionales, cercanías), mercancías, servicio interno, etc. el *tipo (int)* que determina la velocidad máxima del tren (80, 100, 120, 160 , 200) , su *longitud (int)* en metros, el punto de *origen (string)*, el *destino (string)* y su *hora de salida (int)*. Además su *recorrido (list)* las *paradas (list)* solicitadas y el *horario.*

Se definen las funciones *Set, Get* y *Print. Buscar* y las funciones necesarias para mantener las listas: *Agregar, Buscar* y *Modificar* elementos en las mismas.

7.5 Entrada y Salida

Entrada_Datos:

En esta clase definimos un grupo de funciones estáticas, una función *CargarDatos()* que llama consecutivamente a las siguientes funciones:

CargarVias(),

CargarTrayectos()

CargarPuntosDePaso()

CargarBandas()

CargarParadas()

CargarRecorridos()

CargarTrenes()

Todas estas últimas funciones tienen una estructura similar:

Se definen las variables auxiliares necesarias.

Un stream de entrada de datos que abre el fichero de texto correspondiente

Se detecta si ha habido error en la apertura del fichero.

Hasta fin de fichero:

Se lee la línea con los atributos que definen el objeto.

Mensaje de error si el formato de datos no es correcto.

Se crea el objeto leído.

Si procede se agrega a la lista del objeto del que forma parte.

Se devuelve un string que compone el mensaje del resultado de la carga de datos.

Salida_Datos:

No se implementan de momento funciones de salida de datos. Serían precisas si se modificaran, en función de las incompatibilidades detectadas, las paradas de los trenes o sus tiempos de marcha, guardando posteriormente los nuevos horarios de los trenes.

7.6 Generación de Recorrido Horario

Se crea la clase Recorrido_Horario con las funciones necesarias con objeto de generar para cada tren en función de su recorrido las horas de entrada y salida en cada punto de paso.

De acuerdo con el diseño de la infraestructura y los trenes la operativa para generar los horarios es la siguiente:

Datos de entrada:

- Bandas, Puntos de Paso, Trayectos y Vías con su atributos
- Trenes con sus atributos, recorrido y paradas solicitadas.
- Datos de salida:
- Lista de horarios: Punto de Paso, Trayecto de acceso, Tiempo de acceso y vía, hora de llegada, tiempo de parada hora de salida y vía.

Operativa Resumida:

- Para cada recorrido del tren en cada banda del recorrido para cada punto de paso desde el de entrada hasta el de salida generar un horario y añadirlo a la lista de horarios del tren.

Funciones:

GenerarHorario()

Para cada tren de la lista de Trenes *GenerarHorarioTren()*

GenerarHorarioTren(Tren)

Inicializamos valores

pPActual = Tren.Origen;

pPInicial = Tren.Origen;

pPFinal = Tren.Destino;

pPAnterior, viaTrayecto, viaPP por defecto

horaActual = Tren.HoraSalida;

Para cada recorrido de la lista de recorridos del tren *btenerListaHorario()*

ObtenerListaHorarios(Tren, Recorrido)

Obtenemos la lista de Puntos de Paso de Recorrido.Banda.ListPP

Si el tren es impar recorremos la lista de Pps en sentido normal

Si el tren es par recorremos la lista de Pps en sentido inverso.

Para todos los Pps desde el de entrada en banda hasta el de salida

ObtenerTrayecto() -> NuevoHorario()

ObtenerTrayecto(PP1, PP2)

Obtenemos el trayecto que une los Puntos de paso PP1 y PP2, si el tren es impar PP1 será el punto de paso actual obtenido de la lista y PP2 el punto anterior, si el tren par a la inversa.

NuevoHorario()

En este momento conocemos el Tren, el Trayecto y el Punto de Paso Actual ahora podemos: *DefinirViaTrayecto(), CalcularTiempoDeMarcha(),*

CalcularHoraDeLlegada() , ObtenerParada(), CalcularTiempoDeParada()

ObtenerViaPP(). ObtenerHoraSalida()

Con estos datos generamos un nuevo horario y lo añadimos a la lista de horarios del Tren.

DefinirViaTrayecto(Trayecto)

Devuelve la vía por la que circulará el tren el trayecto. Si el trayecto es de vía UNICA vía 1, si es vía Doble la de la paridad del tren.

CalcularTiempoDeMarcha(Tren, Trayecto)

Devuelve el tiempo que se concede al tren para realizar el trayecto del punto de paso anterior hasta el actual. Este tiempo es la relación entre la longitud del trayecto y la velocidad del tren. Si la velocidad máxima del trayecto es inferior a la del tren, será esta la que se tiene en cuenta.

CalcularHoraDeLlegada()

La hora de llegada es la hora actual, que se corresponde con la de salida del punto anterior, más el tiempo concedido de marcha.

ObtenerParada()

Obtenemos la parada si existe solicitada para el tren el punto actual.

CalcularTiempoDeParada()

Es el tiempo solicitado si existe parada en ese punto si no es nulo.

ObtenerViaPP()

Es la vía solicitada en la parada, si no igual que en el caso del trayecto si es de vía UNICA vía 1, si es vía Doble la de la paridad del tren.

ObtenerHoraSalida()

Es igual a la hora de llegada + el tiempo de parada.

7.7 La Malla Gráfica de Trenes

La malla de trenes es fundamentalmente un objeto gráfico que se dibuja con los datos de los horarios de cada tren tal como se describe en el capítulo 5 (ver figura 7.4).

Se implementa sobre una ventana formulario frmMallas.h de System::Windows::Forms y objetos de la misma colección:

Listas desplegables ComboBox^ cboBandas y cboHoras para la selección de bandas y horas.

Objetos para dibujar texto y líneas y puntos PictureBox^ pbxMalla, pbxHoras, pbxPps, pbxAbv, para representar el gráfico las horas los nombres de los Pps y sus abreviaturas, este último se añade en el lado derecho del gráfico de forma similar a los nombres en el izquierdo.

Los objetos pbxMalla y pbxHoras se ubican en un objeto Panel^ con una barra de desplazamiento horizontal en la parte inferior, de este modo se dibuja el gráfico completo y se visualiza por desplazamiento.

Se ajustan los tamaños de los objetos al de la ventana del formulario en los eventos Load() y SizeChanged() del formulario, se redibujan los objetos usando el método Refresh() de los mismos que activa el evento Paint() donde se ubica el código que dibuja el texto o líneas en ellos.

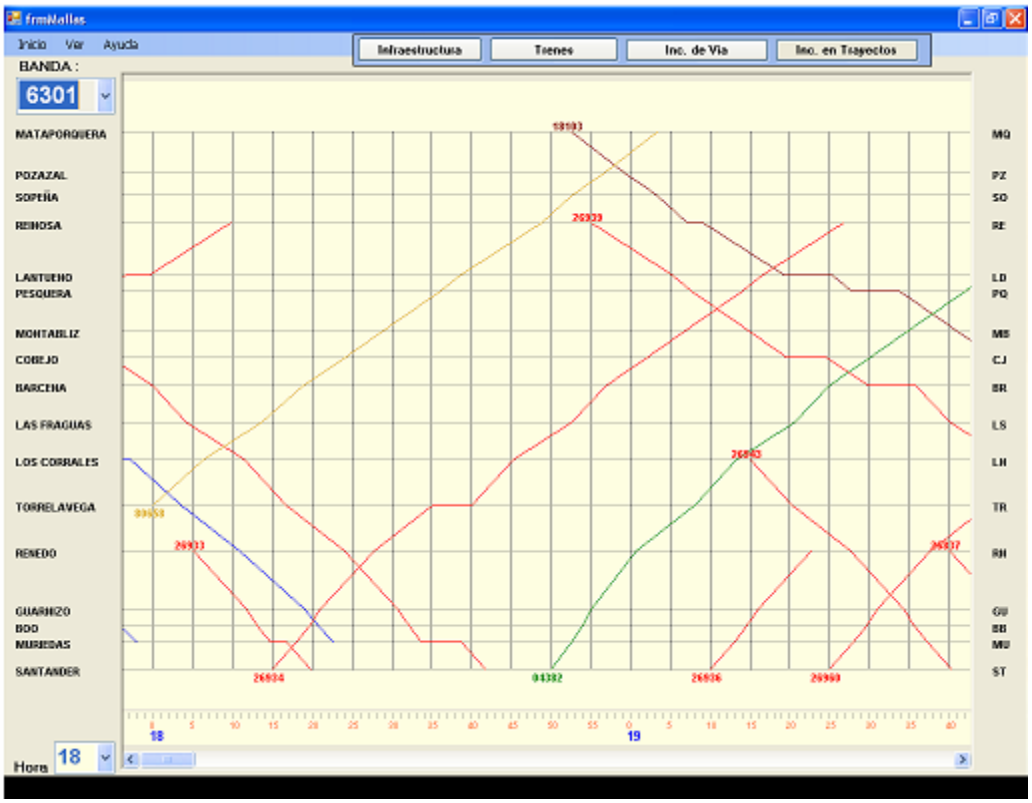

Fig. 7.4 Vista del formulario mallas

(Muestra del formulario de Mallas después de implementado)

Con las funciones int CalculoPosicionHorizontal(int hr, int lgMx) y int CalculoPosicionVertical(int ub, int ubMx) calculamos las posiciones en la escala del objeto gráfico en función de un valor y el valor maximo a representar.

La funciones:

Point ObtenerPuntoGraficoTren(int ub, int ubMx, int hr, int lgMx)

Point ObtenerPuntoTextoTren(int ub, int ubMx, int hr, int lgMx, int p)

Devuelven un objeto del tipo System::Drawing::Point con una posición para dibujar usando las funciones de calculo de posiciones de escala anteriores.

Para dibujar se usan los objetos Graphics^ de Windows:: System::Drawing : Point, Color, Pen, Font, DrawLine, DrawString, etc..,

El código se implementa en las funciones de eventos correspondientes siguiendo la estructura que se describe en el cápitulo 5.

7.8 Detección de Incompatibilidades

Para la detección de incompatibilidades de vía se crea la clase *IncomVia* en el módulo de Incompatibilidades. Tiene un identificador dos atributos tipo string (identificador del tren y el Pp) y dos de tipo int (número de vía y el tipo de incompatibilidad).

Se definen en la clase las funciones Set(), Get (), Print() y las necesarias para el mantenimiento de la lista, además tres Funciones DetectarIncViaError(), DetectarIncViaAnden(), DetectarIncViaLongitud(), que devuelven un string si generan correctamente la lista de incompatibilidades de vía detectadas recorriendo la lista de paradas de cada tren según se describe en el Capítulo 6.

Para las incompatibilidades de trenes en trayecto, se crea la clase intermedia OcupTrayecto con los atributos necesarios que identifican la banda, el tren, el trayecto, los Pps de entrada y salida, vía y paridad así como las horas de entrada y salida para poder identificar posteriormente solapes horarios. Las funciones necesarias para la gestión de la clase y dos funciones GenerarOcupTrayectos(void), GenerarOcupTrayectosFiltroBanda(string idbd) que generan la lista de ocupaciones, completa o filtrada por bandas recorriendo los horarios de los trenes según el algoritmo descrito en el Capítulo 6.

La clase IncTrayecto tiene atributos similares a la anterior, para una vez identificada la incidencia, poder acceder a través de sus identificadores a los objetos correspondientes y presentarlos al usuario y además una lista de instancias de la clase anterior que cumplen las condiciones que dan lugar a ésta incompatibilidad.

Además de las funciones Set(), Get(), Print() y otras para la gestión y presentación de las instancias de la clase, se implementan las funciones:

bool ComprobarSolapeHorario (int he1, int hs1, int he2, int hs2)

bool ComprobarInclusionHorario (int he1, int hs1, int he2, int hs2)

Para comprobar si dos intervalos horarios se solapan y además uno incluye al otro.

Y las funciones :

void ObtenerIncTrayectoTipoX (void) caso 3

void ObtenerIncTrayectoTipoC (void) caso 4
void ObtenerIncTrayectoTipoA (void) caso 5

Que generan la lista de incompatibilidades de cada tipo recorriendo la lista de OcupTrayecto según el algoritmo descrito en el capítulo 6.

7.9 Interfaz Gráfica de Usuario

Se implementan formularios de System::Windows::Forms para presentar al usuario de forma gráfica y con capacidad de seleccionar y visualizar objetos de las listas correspondientes (ver Figura 7.5).

Para seleccionar una banda y sus Pps visualizandolos así como vías y trayectos : frmInfr.h

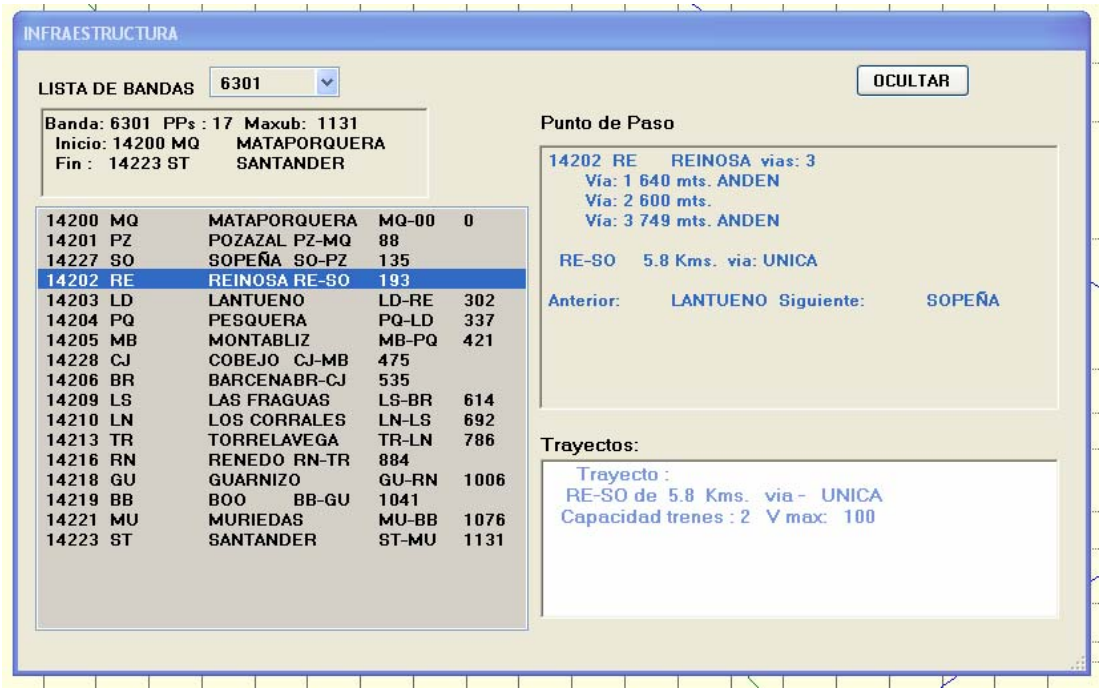

Fig. 7.5 Vista del formulario infraestructura

Para seleccionar un tren y ver su recorrido, paradas y su horario completo (ver Figura 7.6): frmTrenes.h

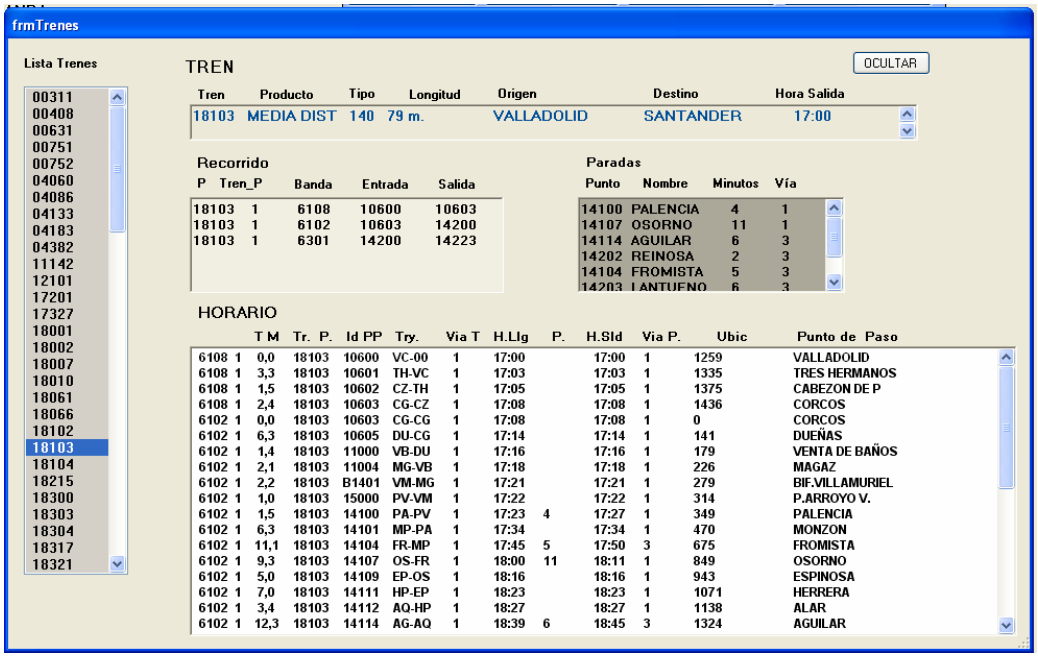

Fig. 7.6 Vista del formulario trenes

Para generar y ver la lista de incompatibilidades de vía detectadas, seleccionarlas y visualizar las características de las mismas (ver Figura 7.7): frmIncVia.h

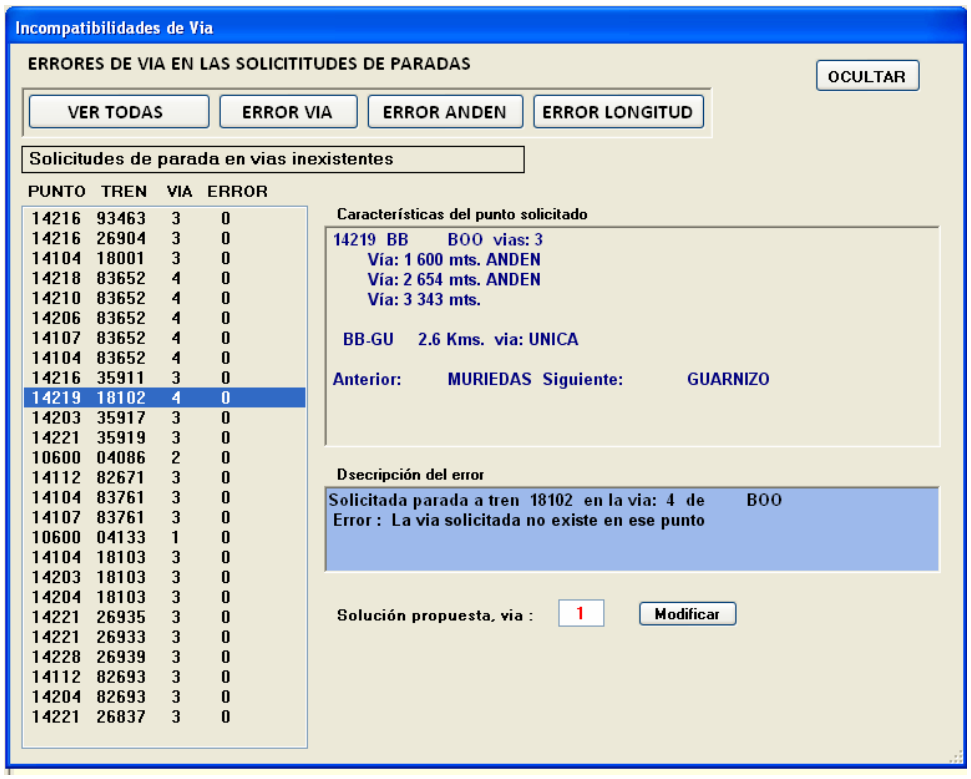

Fig. 7.7 Vista del formulario incidencias de vía

Para generar y ver las incompatibilidades de trenes en trayecto, seleccionar por banda y visualizar sus características (ver figura 7.8): frmIncTrayecto.h

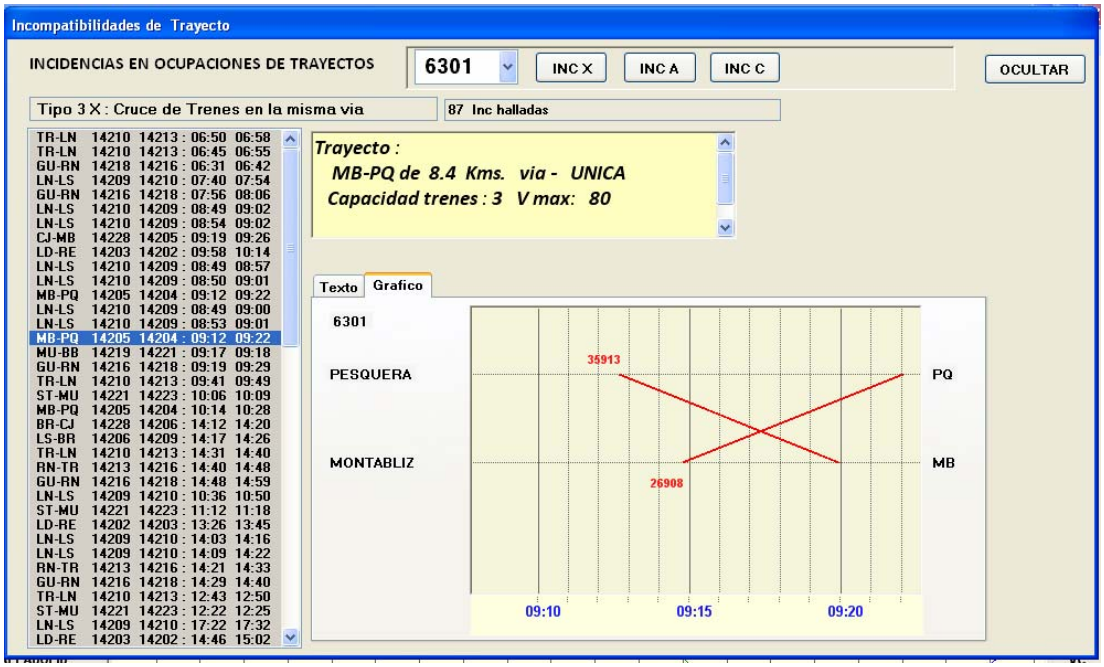

Fig. 7.8 Vista del Formulario incompatibilidades de trayectos

Se implementa también el formulario Inicio.h con los botones y mensajes para activar la carga de datos y la generación de horarios de trenes (ver figura 7-9).

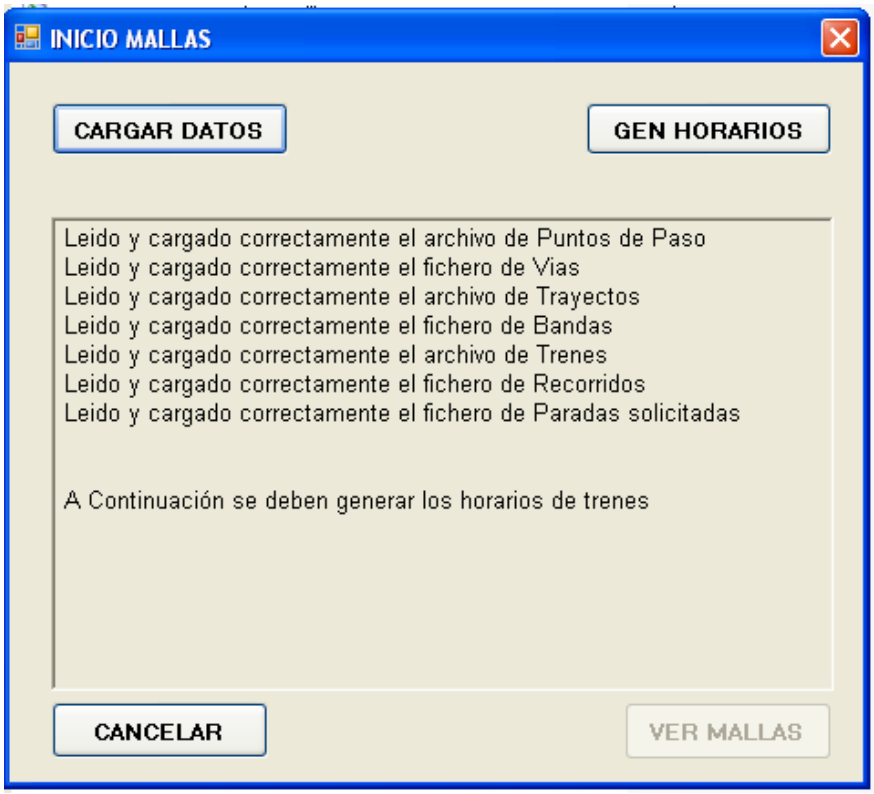

Fig. 7.9 Vista formulario de Inicio

7.10 Gestión de la Aplicación

Se establece el formulario Inicio, que se abre al iniciarse la aplicación. Desde el mismo se cargan los datos y se generan los horarios de trenes. Si éstas operaciones son correctas, se puede acceder al formulario Mallas, desde el que visualiza la malla de trenes y se incluyen en el mismo, botones y menús para acceder al resto de las ventanas de la aplicación.Al cerrarlo se cierra la aplicación.

7.11 Conclusiones

La aplicación tiene como base una estructura de *clases de datos* de *infraestructura y trenes*, unas *funciones para capturar* los datos de archivos de texto y *generar* posteriormente datos derivados como los *horarios de trenes* con estos datos representa la *malla gráfica* y mediante una clase de funciones al efecto sobre los datos guardados se *detectan las incompatibilidades*, todo ello se presenta al usuario con una *interfaz gráfica*.

Capítulo

8

Pruebas de la Aplicación

8.1 Introducción

Para los objetivos de este proyecto es fundamental el diseño de los datos, su estructura y relaciones, así como la consistencia y persistencia de los mismos. Para comprobar estos aspectos se han realizado pruebas de la aplicación, no sólo al final del desarrollo de la misma, sino en cada fase del diseño e implementación de cada uno de los grupos de datos que se ha descrito.

Para realizar estas pruebas es precisa la presentación de datos al usuario por los que prácticamente en todas las clases se implementan funciones print () que realizan esta función.

Dado que se consideraba necesario implementan una interfaz gráfica de usuario para presentar los datos de la infraestructura y trenes al usuario, estos formularios se han ido implementando a la vez que los datos para comprobar sus relaciones y consistencia.

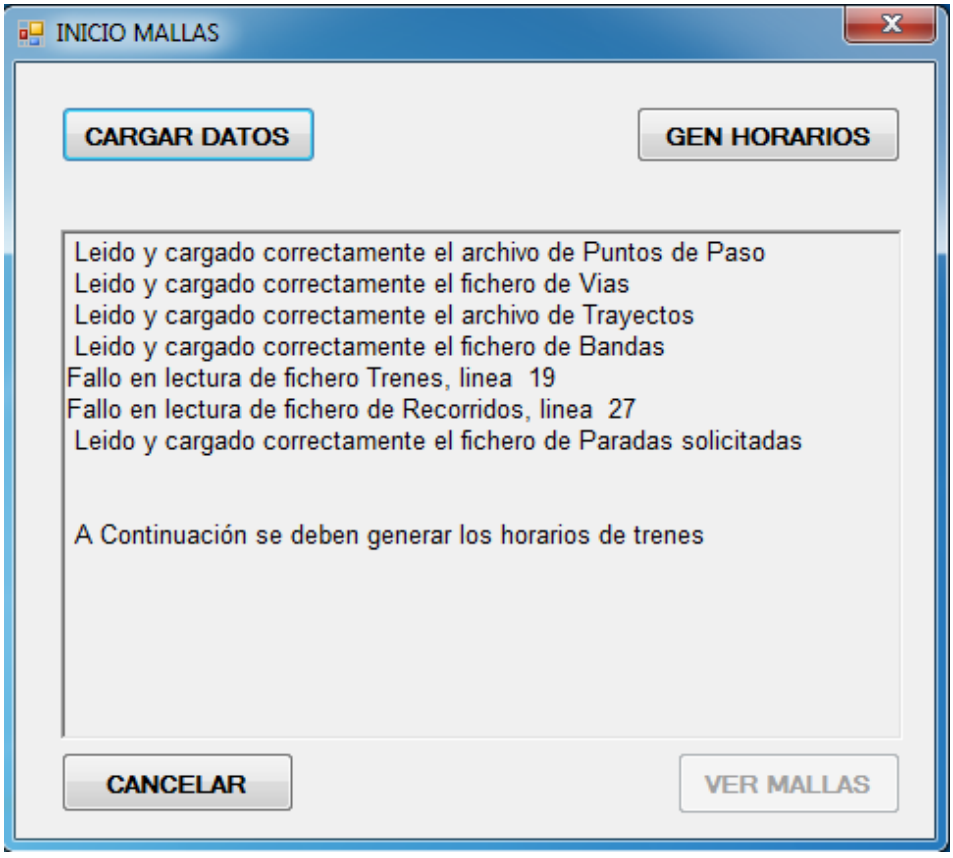

Fig. 8.1 Pruebas de carga datos con errores en Fichero

8.2 Pruebas de Carga de Datos

Para cada tipo de datos se realizan pruebas de error de inexistencia de fichero, de la identificación correcta de los tipos, la detección de errores en el fichero de texto y la delimitación de la línea de error.

8.3 Pruebas de la Infraestructura

Se realizan pruebas para cada clase de la correcta asignación de los datos a las listas correspondientes y una vez definidas todas las clases se comprueba que las relaciones requeridas se establecen correctamente.

Las primeras pruebas se realizan sobre un pequeño número de puntos de paso simples y una banda, al final se incluye una infraestructura de 3 bandas, 48 puntos de paso, 50 trayectos y 147 que se considera suficiente para la presentación de todas las características de la aplicación en cuanto a infraestructura, ya que se incluyen los accesos de una a otra banda y trayectos de salida(bypass) intermedios, así como trayectos de distinto tipo y capacidad.

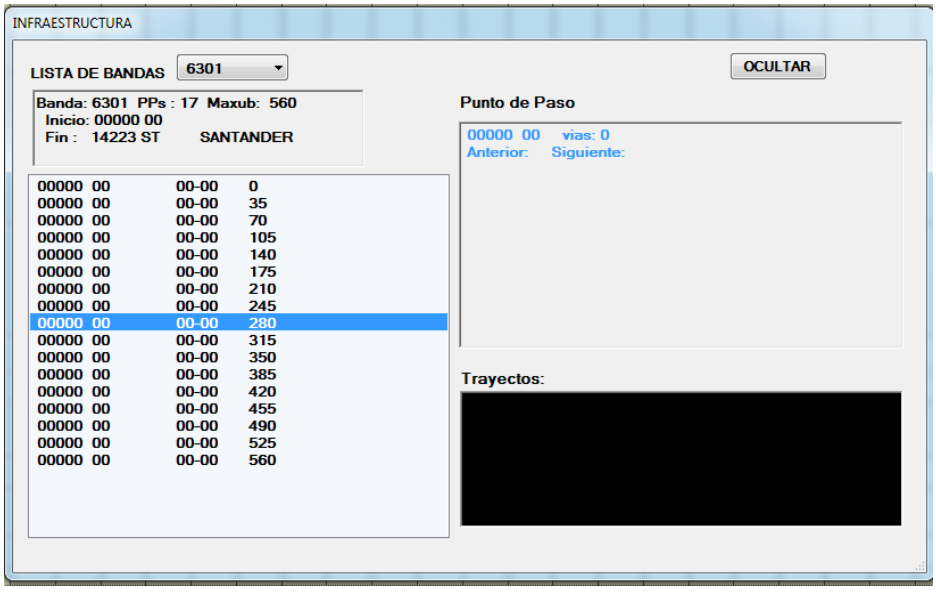

Fig. 8.2 Pruebas de casos de error de infraestructura

La estructura del formulario para presentación de éstos datos al usuario, permite seleccionar distintas bandas, de modo que se puede comprobar la correcta adscripción de los puntos de paso a cada una de ellas, al mostrar los datos completos de un punto de paso seleccionado, vías y trayectos, permite comprobar también si a cada Pp se han adscrito las vías y trayectos correspondientes.

8.4 Pruebas de Trenes y Horarios

Al igual que en el caso anterior es preciso comprobar que los trenes se definen correctamente con sus atributos, en la carga de datos se adjudica a cada tren el recorrido y las paradas correspondientes.

También como en el caso anterior el formulario para presentación de datos al usuario permite visualizar la lista de trenes y seleccionado uno de ellos comprobar sus atributos, el recorrido, las paradas y el recorrido horario del mismo.

Durante el desarrollo de esta parte de la aplicación se han realizado pruebas con un número reducido de trenes y distintos recorridos hasta buscar los más completos, comprobando que se generan correctamente los horarios con las paradas, las adjudicaciones de paridad, las salidas y entradas en banda, etc.

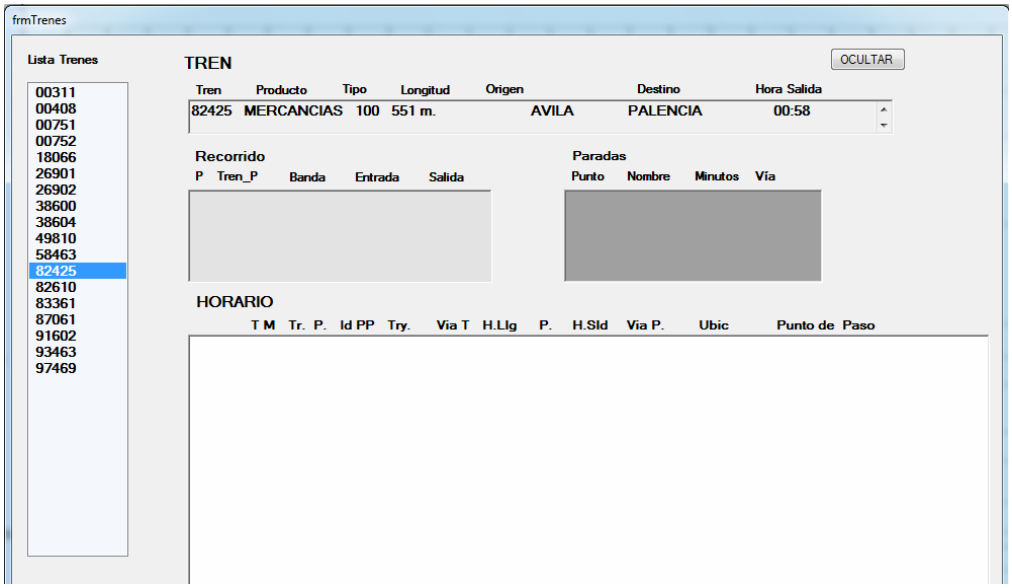

Fig. 8.3 Pruebas de trenes

Se realizan pruebas del comportamiento de la generación de horarios ante errores en la secuencia correcta en la definición del recorrido, se comprueba que la aplicación genera en casi todos los casos un recorrido horario aunque sea irreal, caso de errores en duplicación del recorrido o saltos en el mismo, no se ha conseguido producir un error en la generación del recorrido horario, la aplicación si no encuentra ninguna correspondencia entre bandas y puntos de paso no genera horarios.

Para la presentación final de la aplicación se eligen 100 trenes de distintas características en cuanto a producto, longitud y recorridos.

8.5 Pruebas de la Malla Gráfica

Las pruebas de la correcta implementación de malla gráfica de trenes, se realizan después de comprobar la corrección del diseño y estructura de los datos de entrada así como de su interfaz gráfica (ver figura 8.4).

Se realizan pruebas para verificar que la malla cumple los requisitos definidos para la misma:

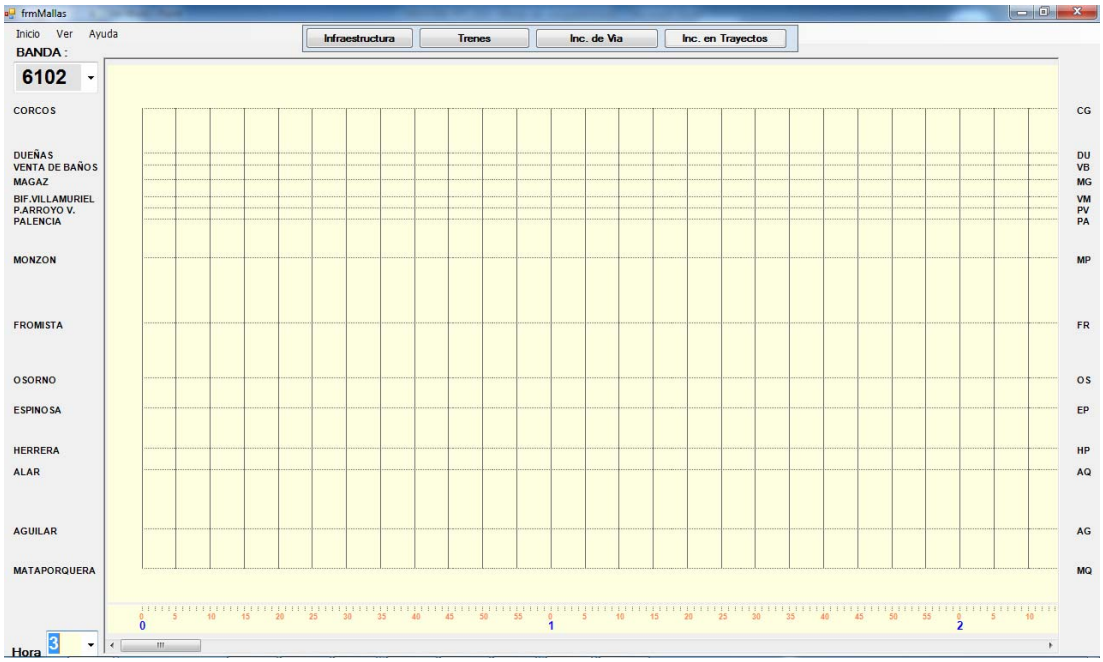

Fig. 8.4 Comprobación de comportamiento de objetos en grafico

- Pruebas de la correcta vista y posición de los objetos gráficos en los cambios de tamaño de ventana.

- Pruebas del correcto desplazamiento horizontal del gráfico, haciendo uso de la correspondiente barra y botones de desplazamiento.

- Se comprueba que se representan correctamente los puntos de paso de cada banda en sus posiciones y se dibujan correctamente las horas y las líneas del gráfico.

- Se comprueba que se dibujan correctamente las líneas del recorrido de los trenes según su paridad y las posiciones se corresponden correctamente con las horas y las ubicaciones de los puntos de paso.

Las pruebas se realizan con número reducido de trenes al inicio, comprobando incluso la representación de los errores de recorrido y haciendo uso de los formularios de infraestructura y trenes para comprobar la inexistencia de errores.

8.6 Pruebas de la Detección de Incompatibilidades

También en este caso la implementación de la interfaz gráfica, los formularios para que el usuario pueda generar y visualizar las incompatibilidades detectadas de modo que al seleccionar una de la lista se presentan los detalles de la misma, sirve para comprobar que la aplicación detecta correctamente las incompatibilidades de paradas y de trayecto (figura 8.5).

Para realizar las pruebas de detección de incompatibilidades de vías y paradas se introducen nuevos datos de paradas con errores de adjudicación de vías comprobando que el sistema las detecta correctamente y así se visualizan.

Para las pruebas de detección de incompatibilidades de trenes en trayecto se modifican los horarios y algunos recorridos de los trenes que en principio se habían tomado de datos previstos reales y posteriormente se comprueba que el sistema detecta las incompatibilidades de cruces alcances y capacidad que se observan en el gráfico y viceversa que las detectadas son reales.

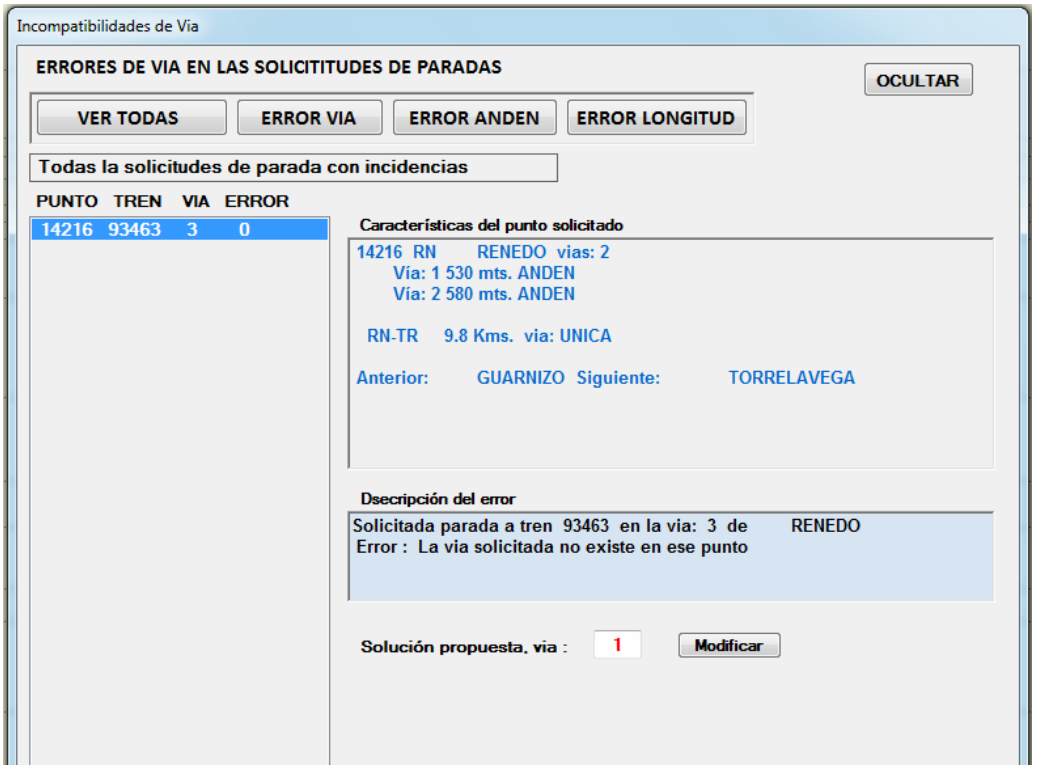

Fig. 8.5 Pruebas comportamiento formulario

Para una mejor comprobación por el usuario se incluye una pequeña representación gráfica de del trayecto y los trenes afectados.

8.7 Conclusiones

Esta aplicación es fundamentalmente un modelo de representación esquemática de la realidad. En diferentes fases de la implementación de la aplicación se han ido realizando pruebas con objeto de verificar que la aplicación realiza correctamente su función antes de pasar a la siguiente fase.

La mejor forma de hacer estas comprobaciones es sobre la propia presentación que se hace al usuario. En cada caso el usuario debe ver representada la realidad que conoce, comprobando así que la aplicación ha cargado y elaborado correctamente los datos y “ha sabido” interpretarlos.

Capítulo

9

Planificación y Coste del Proyecto

9.1 Introducción

Para la realización del proyecto se puso como objetivo de entrega la 3ª convocatoria 1 a 20 de Marzo de 2014, de este modo la fase de análisis debiera estar concluida en Mayo de 2013, aproximadamente en Octubre de 2013 concluir la segunda fase y los inicios del diseño de la herramienta al objeto de concluir el proyecto antes de Febrero de 2014.

El proyecto se ha concluido en diciembre de 2013. A continuación se expone una relación de las diferentes fases en que se puede dividir el desarrollo del proyecto, las fechas aproximadas de realización de las mismas y una aproximación de tiempo y costes.

9.2 Planificación y Desarrollo

Se incorpora a continuación una relación de las diferentes fases en que se puede dividir el desarrollo del proyecto, con las fechas aproximadas en la que se han ejecutado las mismas. Estas fechas sirven para establecer una ubicación temporal de cada fase, la extensión de estos periodos no guarda un relación exacta con el tiempo empleado, este cálculo se establecerá más adelante:

- **Elaboración del anteproyecto:**

  Elaboración y presentación del anteproyecto (Octubre 2012).

- **Estudio de viabilidad:**

  Estudio de la viabilidad del proyecto, enfoque, metodología de desarrollo y planificación (Diciembre 2012).

- **Búsqueda de referencias:**

  Estudio desde el punto de vista de usuario de aplicaciones y bases de datos existentes relacionadas con el proyecto a desarrollar, así como la observación de posibles estructuras de datos (10/01/13 – 10/02/13).

- **Análisis:**

Análisis de datos y objetos necesarios, atributos y relaciones de los objetos, estructura de la aplicación (10/02/13 – 20/04/13).

- **Diseño Datos :**

Diseño de la estructura modular y clases de datos de entrada, Infraestructura y Trenes. Así como las funciones de entrada de datos y generación de recorridos horarios (20/03/13- 10/05/13).

- **Diseño gráfico:**

Diseño de la malla grafica de trenes, su composición, estructura, objetos gráficos y funciones de cálculo de datos (20/04/13 – 20/05/13).

- **Implementación de datos:**

Se implementan las clases de datos, las funciones de carga de datos y generación de recorridos horarios (20/06/13 – 10/08/13).

- **IGU 1:**

Implementación de la interfaz gráfica de carga de datos de infraestructura y trenes. Primeras pruebas de carga correcta de datos, consistencia y persistencia. (01/08/13 – 20/08/13).

- **Implementación de la malla:**

Implementación de la malla gráfica de trenes, de los objetos y funciones necesarios. Pruebas del comportamiento de los datos y de su representación gráfica (01/08/13 – 30/09/13).

- **Incompatibilidades 1:**

Análisis y diseño de las funciones de detección de incompatibilidades (20/09/13 – 01/10/13).

- **Incompatibilidades 2:**

Implementación de las funciones e interfaz del módulo de incompatibilidades. Pruebas de la detección de incompatibilidades de vía y de trayecto, comprobación la coincidencia de la detectadas con los datos y la representación gráfica (01/10/13 – 30/11/13).

- **IGU Final:**

Implementación completa de la interfaz gráfica, de los menús y botones de apertura de los distintos formularios. (10/11/13 – 25/11/13).

- **Pruebas:**

Pruebas generales de la aplicación con distintos grupos de datos. (25/11/13 – 10/12/13).

- **Memoria :**

Revisión y agrupamiento de documentación realización de la memoria del proyecto. Revisión de la aplicación (01/12/13 – 31/12/13).

FASE	INICIO	FINAL
Elaboración de anteproyecto	01/10/2012	30/10/2012
Estudio de viabilidad	15/12/2012	30/12/2012
Búsqueda d referencias	10/01/2013	10/02/2013
Análisis	10/02/2013	20/04/2013
Diseño Datos	20/03/2013	10/05/2013
Diseño gráfico	20/04/2013	20/05/2013
Implementación de datos	20/06/2013	10/08/2013
IGU Datos	01/08/2013	30/08/2013
Implementación de la malla	01/08/2013	30/09/2013
Incompatibilidades	20/09/2013	01/10/2013
Incompatibilidades 2	01/10/2013	30/11/2013
IGU	10/11/2013	25/11/2013
Prueba	25/11/2013	10/12/2013
Memoria	01/12/2013	31/12/2013

Fig. 9.1 Resumen gráfico de las fases de desarrollo

9.3 Coste del Proyecto

Para el cálculo de coste del proyecto se supone un valor hora de 60 € y se obtienen los siguientes resultados:

FASE	días	días hab.	horas 1	coste
Elaboración de anteproyecto	**29**	**2**	**3**	**180 €**
Estudio de viabilida	**15**	**3**	**4**	**240 €**
Búsqueda de referencias	**30**	**5**	**7**	**420 €**
Análisis	**70**	**9**	**13**	**780 €**
Diseño Datos	**50**	**14**	**21**	**1.260 €**
Diseño Gráfico	**30**	**9**	**13**	**780 €**
Implementación de datos	**50**	**16**	**24**	**1.440 €**
IGU Datos	**29**	**8**	**12**	**720 €**
Implementación de la malla	**59**	**14**	**21**	**1.260 €**
Incompatibilidades	**11**	**5**	**7**	**420 €**
Incompatibilidades 2	**59**	**14**	**21**	**1.260 €**
IGU	**15**	**5**	**7**	**420 €**
Prueba	**15**	**2**	**3**	**180 €**
Memoria	**30**	**10**	**15**	**900 €**
TOTALES	**492**	**116**	**171**	**10.260 €**

Fig. 9.2 Cálculos de costes

Para evaluar el número de horas se realiza primero un cálculo de los días realmente hábiles, días de dedicación real al proyecto, en el periodo de cada fase. Se calculan las horas empleadas en esos días usado un factor horas/día que promedie esa dedicación real.

Obtenemos así el total de horas aproximadamente empleadas y una evaluación a evaluación aproximada de costes.

Capítulo

10

Conclusiones y Trabajos Futuros

10.1 Introducción

Una vez concluido el proyecto podemos reflexionar y extraer conclusiones sobre lo que se ha realizado y lo que es más importante que caminos se abren para trabajos futuros con este tipo de herramientas informáticas.

Repasamos primero cuáles son los procedimientos operativos que sigue el usuario humano en las actividades en las que sirven de apoyo estas herramientas, así como la información que estas herramientas deben aportar y los procesos en los que podrían sustituirle.

Analizamos después en qué casos y hasta dónde esta aplicación puede servir de ayuda o sustituto del operador humano.

Y finalmente en qué aspectos puede mejorar o evolucionar la aplicación para interactuar de forma más completa con el usuario o incluso llegar a sustituirle.

10.2 Conclusiones

En situaciones reales de regulación de la circulación de trenes el experto humano sigue de forma resumida el siguiente proceso mental:

1. Recopila u ordena la información de infraestructura y trenes que precisa para la banda, puntos de paso y trenes sobre los que va a actuar.
2. Observa y estudia la malla de trenes.
3. Detecta los casos de incompatibilidad que se presentan en la misma.
4. Estudia estos casos en relación con la información que conoce.
5. Valora las posibles actuaciones para solventar las incompatibilidades detectadas.
6. Adopta las medidas de solución que considera procedentes.
7. Comprueba el resultado y las posibles nuevas incompatibilidades que se generan.

En relación con lo anterior, como herramienta para la regulación de la circulación de trenes mencionamos algunas de las funcionalidades que se han implementado en la aplicación desarrollada en este proyecto:

- Es capaz de cargar los datos de infraestructura y trenes que se precisan, guardarlos con una estructura organizada y consistente y generar otros datos como los horarios de los trenes, manteniendo así una parte de la información necesaria.
- Presenta la malla de trenes con la información de que dispone en un determinado momento y es capaz de representarla casi instantáneamente con cualquier modificación de datos de infraestructura o trenes y horarios que se produzca.
- Permite al usuario obtener, visualizar y comparar toda la información que la aplicación tiene.
- Detecta un grupo de incompatibilidades de interacción de trenes en vías de paso o estacionamiento y coincidencias de trenes temporalmente en trayectos.
- Presenta estas incompatibilidades al usuario para su comprobación y análisis.
- Es decir la herramienta completa los procesos a seguir hasta el paso 3 y sirve de ayuda para los pasos siguientes. De esta comparación entre ¿para qué se necesita? y ¿qué puede hacer? Podemos inferir dos caminos para futuros trabajos.

10.3 Trabajos Futuros

Un camino a seguir es el desarrollo de aplicaciones de ayuda a regulación de trenes similares a las actuales que operan con grandes bases de datos de trenes e infraestructura. En este campo se pueden realizar trabajos para mejorar los tiempos de carga y actualización de datos. También hay un campo importante de actuación sobre los aspectos gráficos de aplicación y la interacción del ratón sobre determinadas posiciones de pantalla, para desplegar información ampliada al usuario y facilitar la entrada de datos para cambios y actualizaciones.

Por ejemplo:

- Hacer clic con determinado botón del ratón sobre el nombre de un tren o punto de paso para obtener información del mismo.
- Hacer clic sobre la línea de recorrido de un tren para modificar los puntos gráficos de entrada o salida de un punto de paso, modificando automáticamente las horas, el horario del tren y en consecuencia posteriormente redibujar el nuevo recorrido.
- Etc.

El otro camino son los simuladores. Basados fundamentalmente en la capacidad que tenga la aplicación para la identificación correcta de incompatibilidades, porque identificados correctamente los problemas y los datos que los conforman se puede llegar a una solución óptima de acuerdo con unos criterios que se deben especificar.

Podemos avanzar futuros desarrollos de simuladores fundamentalmente de dos tipos:

Simulador de mallas teóricas:

Una malla teórica se crea representando gráficamente los horarios de recorrido, de un número determinado de trenes, que se generan a partir de su hora de salida y los tiempos de parada obligada que se solicitan en determinados puntos para los mismos.

Cada vez que se incluye un nuevo tren, éste produce interacciones con otros trenes en vías o trayectos. Si estas interacciones son incompatibles con la realidad deben solventarse. Para ello deben alterarse paradas o tiempos de marchas de otros trenes hasta que desaparezcan estas incompatibilidades de la nueva malla.

Un simulador de este tipo detecta y corrige por sí mismo las incompatibilidades producidas. Aplica para ello unos criterios de preferencias u objetivos que se hayan definido previamente y puede presentar diferentes soluciones para la validación por el usuario.

Simulador de situaciones:

Un planteamiento para este tipo de simulador por ejemplo podría ser el siguiente:

Partimos de una situación actual, donde actuamos sobre una parte de la infraestructura determinada y una malla teórica planificada. En esta situación no tenemos incompatibilidades. A continuación se introducen en el sistema supuestas alteraciones. Por ejemplo, de merma de capacidad de infraestructura, en las horas de salida, o en los tiempos de parada de uno o varios trenes, lo que altera la malla y genera incompatibilidades de estacionamientos de trenes o coincidencias no previstas en determinados trayectos.

El simulador detecta estas incompatibilidades y de acuerdo con criterios de preferencia de trenes o retrasos máximos, modifica paradas o tiempos de marcha, hasta que todas las incompatibilidades desaparezcan, mostrando al usuario una o varias

soluciones a elegir o en un paso más avanzado, eligiendo el propio simulador la solución óptima.

10.4 Conclusión Final

Este trabajo no ha requerido de complicados algoritmos, ni del diseño de objetos o funciones excesivamente complejos. En cuanto a la malla gráfica, tampoco se ha inventado nada absolutamente diferente a lo que ya hacen otras aplicaciones. Simplemente se ha realizado partiendo de un análisis desde la base, una representación gráfica en un lenguaje de programación (C++), de una forma más personal y con otros objetos gráficos.

El trabajo más extenso ha sido el análisis de la realidad, los objetos reales, sus características y relaciones y diseñar posteriormente las clases que los representan con todos los atributos y relaciones necesarias, unas para representar correctamente la malla gráfica, todas para poder identificar posteriormente los casos de iteraciones requeridas y con la máxima información posible.

Es por este último aspecto por donde quizás nos podamos aproximar al inicio de otros caminos en este tema.

Referencias y Bibliografía

Nota: Durante las primeras fases del proyecto se han estudiado desde el punto de vista de usuario las aplicaciones propiedad de ADIF (Administrador de Infraestructuras Ferroviarias) y de uso restringido en la empresa y solo a personal autorizado: SITRA, GTRENES, SIGES

[PRE01] Pressman, Roger S. (2001), "Ingeniería del Software (Un enfoque práctico)" Mc Graw Hill.

[SIL02] Silberschatz, Abraham, Korth Henry F., SudarsHan S. (2002), "Fundamentos de Bases de Datos", Mc Graw Hill.

[HER00] Hernández R. , Lázaro J.C., Dormido R., Ros S. (2000), "Estructuras de Datos y Algoritmos", Prentice Hall.

[CER00] Cerrada Somolinos J. y Otros, (2000), "Introducción a la Ingeniería del Software", Ed. Centro Estudios Ramón Areces.

[CAB02] Cabrera Gregorio, Montoya Guillermo, (2002), "Análisis y diseño de aplicaciones de informática de gestión" Mc Graw Hill.

[GON03] Gonzalo Cuevas, Agustín. (2003), " Gestión del proceso de Software",Ed. Centro Estudios Ramón Areces

[HUM02] Humphrey, Watts S. (2002), "Introducción al Proceso de Software Personal", Addison Wesley.

[FER03] Fernández, Galán S., González, Boticario J., Mira Mira J. (2003), "Problemas resueltos de Inteligencia Artificial Aplicada", Pearson Addison Wesley.

[GRO13] Groussard, Thierry. (2012), "C# Desarrollar con Visual Studio 2012" Eni ediciones.

[STR00] Stroustrup, Bjarne, (2000), "C++ Programming Language",Addison Wesley.

MSDN Visual Studio de Microsoft :

http://msdn.microsoft.com/es-es/vstudio/

“Aprenda C++ como si estuviera en primero”, García de Jalón Javier y Otros, (1998), Escuela Superior de Ingenieros Industriales. UNIVERSIDAD DE NAVARRA:

http://mat21.etsii.upm.es/ayudainf/aprendainf/cpp/manualcpp.pdf

“Aprenda C++ Avanzado como si estuviera en primero”, Bustamante Paul, y otros, Escuela Superior de Ingenieros Industriales. UNIVERSIDAD DE NAVARRA:

http://mat21.etsii.upm.es/ayudainf/aprendainf/cpp/2cppavan.pdf

Curso de C++ , ConClase.net:

http://c.conclase.net/curso/?cap=022#inicio

Anexo A

Código Fuente

A.1 Introducción

El código completo se encuentra en el CD que acompaña al proyecto. Además se incluye un archivo en formato pdf con el código completo. No obstante se incluye a continuación la parte del código que se considera más interesante.

A.2 Archivos de Encabezado

A.2.1 Infraestructura

Vía.h

```
//Miguel Arranz Pascual PFC Mallas
//fichero Via.h
//Declaración de la clase Vía dentro del módulo infraestructura

#pragma once
#ifndef _VIA_H_
#define _VIA_H_

#include <iostream>
#include <string>
using namespace std;
using namespace System;

class Via
{
// miembros de datos
private:
        string idVia; // id de vía cod pp + vía
        int numVia;    // números 1,2, 3....
        int longitud;  // en metros
        int anden;    // si tiene o no andén


public:
        // Constructores
        Via(void); // constructor por defecto
        Via(string idv); // constructor solo con Id
        Via(string idv, int v, int l , int a); // constructor con datos


        //  Set
    void SetIdVia(string idv);
        void SetNumVia(int v);
        void SetLongitud (int l);
        void SetAnden (int a);
```

```
        // Get
        string GetIdVia(void) {return idVia;};
        int GetNumVia(void) {return numVia;};
        int GetLongitud (void) {return longitud;};
        int GetAnden(void) {return anden;};

        //Operadores miembro

        string PrintVia (void);
        string PrintVia (Via &v);

};
  #endif
```

Trayecto.h

```
//Miguel Arranz Pascual PFC Mallas
//fichero Trayecto.h
//Declaración de la clase Trayecto dentro del módulo infraestructura

#pragma once
#ifndef _TRAYECTO_H_
#define _TRAYECTO_H_

#include <iostream>
#include <string>
#include <list>


using namespace std;

class Trayecto
{

// miembro de clase
public:
static list<Trayecto>ListaDeTrys; //Guarda la lista de los Trayectos
cargados

// Descripción de tipo
//public:
//      enum tryctoTipo {UNICA=1, DOBLE};

// miembros de datos
private:
        string idTrayecto;    // identificador PP origen - destino dos
letras FG-CV
        double longitud;  // en kilómetros tres decimales
        int capacidad;   // de 1 a 10 según los trenes que admite en
trayecto
        int tipo; // "UNICA" "DOBLE"
        int vMax;  // velocidad max en trayecto 80 110 etc.
        string ppInicioTry; // punto de paso que inicia el trayecto
```

```
        string ppFinalTry;  // punto de paso último del trayecto

// Constructores
public:

        Trayecto(void); // constructor por defecto
        Trayecto (string id); // constructor con id por defecto
        Trayecto(string id, double l,int c, int tp, int vm, string stppIni,
string stppFin); // constructor con datos

// Set
        void SetIdTrayecto(string id);
        void SetLongitud (double l);
        void SetCapacidad (int c);
        void SetTipo (int tp);
        void SetVMax (int vm);
        void SetPPInicioTry(string stppIni);
        void SetPPFinalTry(string stppFin);

// Get
        string GetIdTrayecto(void) {return idTrayecto;};
        double GetLongitud (void) {return longitud;};
        int  GetCapacidad(void) {return capacidad;};
        int GetTipo (void) {return tipo;};
        int GetVMax (void) {return vMax;};
        string GetPPInicioTry(void){return ppInicioTry;};
        string GetPPFinTry(void){return ppFinalTry;};

//Operadores miembro
        string PrintTrayecto ();
        string PrintTrayecto (const Trayecto &t);

//Operadores de clase

        static void AgregarTrayecto (const Trayecto &tr); // agrega un
trayecto a la lista
        static string PrintLista(void); // presenta la lista
        static Trayecto BuscarTry (string idTry); // obtiene el trayecto
con el id correspondiente
        static list<Trayecto> BuscarTryOrigen (string idppOrg); // obtiene
el trayecto que tiene el origen en ese pp
};
#endif
```

PuntoDePaso.h

```
//Miguel Arranz Pascual PFC Mallas
//fichero Trayecto.h
//Declaración de la clase Trayecto dentro del módulo infraestructura

#pragma once
#ifndef _TRAYECTO_H_
#define _TRAYECTO_H_

#include <iostream>
#include <string>
#include <list>


using namespace std;

class Trayecto
{

// miembro de clase
public:
static list<Trayecto>ListaDeTrys; //Guarda la lista de los Trayectos
cargados

// Descripción de tipo
//public:
//	enum tryctoTipo {UNICA=1, DOBLE};

// miembros de datos
private:
	string idTrayecto;	// identificador PP origen - destino dos
letras FG-CV
	double longitud;  // en kilometros tres decimales
	int capacidad;   // de 1 a 10 según los trenes que admite en
trayecto
	int tipo; // "UNICA" "DOBLE"
	int vMax;  // velocidad max en trayecto 80 110 etc.
	string ppInicioTry; // punto de paso que inicia el trayecto
	string ppFinalTry;  // punto de paso último del trayecto

// Constructores
public:

	Trayecto(void); // constructor por defecto
	Trayecto (string id); // constructor con id por defecto
	Trayecto(string id, double l,int c, int tp, int vm, string stppIni,
string stppFin); // constructor con datos

// Set
	void SetIdTrayecto(string id);
	void SetLongitud (double l);
	void SetCapacidad (int c);
	void SetTipo (int tp);
	void SetVMax (int vm);
	void SetPPInicioTry(string stppIni);
	void SetPPFinalTry(string stppFin);
```

```
// Get
        string GetIdTrayecto(void) {return idTrayecto;};
        double GetLongitud (void) {return longitud;};
        int  GetCapacidad(void) {return capacidad;};
        int GetTipo (void) {return tipo;};
        int GetVMax (void) {return vMax;};
        string GetPPInicioTry(void){return ppInicioTry;};
        string GetPPFinTry(void){return ppFinalTry;};

//Operadores miembro
        string PrintTrayecto ();
        string PrintTrayecto (const Trayecto &t);

//Operadores de clase

        static void AgregarTrayecto (const Trayecto &tr); // agrega un
trayecto a la lista
        static string PrintLista(void); // presenta la lista
        static Trayecto BuscarTry (string idTry); // obtiene el trayecto
con el id correspondiente
        static list<Trayecto> BuscarTryOrigen (string idppOrg); // obtiene
el trayecto que tiene el origen en ese pp
};
#endif
```

Banda.h

```
//Miguel Arranz Pascual PFC Mallas
//fichero Banda.h
//Declaración de la clase Banda dentro del módulo infraestructura

#pragma once
#ifndef _BANDA_H_
#define _BANDA_H_

#include <iostream>
#include <string>
#include <list>
#include "Via.h"
#include "Trayecto.h"
#include "PuntoDePaso.h"
#include "PuntoPasoBanda.h"

using namespace std;

class Banda
{
// miembro de clase
public:
static list<Banda>ListaDeBandas; //Guarda la lista de las Bandas cargadas

//Miembros de datos
private:
        string bdIdBanda;       //identificador único de banda 4 cifras ej.
0102 6105
```

```
        int numPuntosDePaso; // número de puntos de paso que componen la
banda sirve para generar la lista de Pps
        int ubMax;    // ubicación del ultimo pp de la banda
        PuntoDePaso ppInicioBanda; // punto de paso que inicia la banda
        PuntoDePaso ppFinalBanda;  // punto de paso último de la banda
        list<PuntoPasoBanda>bdListaPP; // lista de puntos de paso de la
banda,

public:

//Constructores
        Banda(void);     // Constructor por defecto
        Banda(string id); // Constructor con identificador
        Banda(string id, int npp, PuntoDePaso &ppIni, PuntoDePaso &ppFin);
//constructor con datos

//Set
        void SetIdBanda(string id);
        void SetNumPP (int nm);
        void SetUbMax (int ubmx);
        void SetPPInicioBanda(PuntoDePaso &ppIni);
        void SetPPFinalBanda(PuntoDePaso &ppFin);

//Get
        string GetIdBanda(void){return bdIdBanda;};
        int GetNumPuntosDePaso(void){return numPuntosDePaso;};
        int GetUbMax (void) {return ubMax;};
        PuntoDePaso GetPPInicioBanda(void){return ppInicioBanda;};
        PuntoDePaso GetPPFinBanda(void){return ppFinalBanda;};
        list<PuntoPasoBanda> GetListPP(void);

//Operadores miembro
        void AgregarUnPP (const PuntoPasoBanda &pp);
        string PrintBanda ();
        string PrintBandaCompleta();

//Operadores de clase
        static void GenerarListaPP (Banda &bd);  // Genera la lista de
Puntos de Paso que componen la Banda
        static void AgregarPP (string stbd, string idpp, int ub, string
idtr);
        static void AgregarBanda (const Banda &bd);
        static Banda BuscarBanda (string stbd); // obtiene la banda con el
id correspondiente
        static string PrintLista(void);
        static string PrintListaCompleta(void);
        static void UbMaxBanda(string idbd, int ubmx);
};
  #endif
```

A.2.2 Trenes

Parada.h

```
//Miguel Arranz Pascual PFC Mallas
//fichero Parada.h
//Declaración de la clase Parada tren dentro del módulo trenes

#pragma once
#ifndef _PARADA_H_
#define _PARADA_H_

#include <iostream>
#include <string>
using namespace std;
using namespace System;

class Parada
{
// miembros de datos
private:
        string idTren; // un id para el tren sobre el que se solicita la
parada.
        string idPPaso; // un id para el punto de paso donde se solicita la
parada.
        int numVia; // vía en la que se solicita la parada.
        int tiempoP; // minutos de parada solicitados.

public:
        // Constructores
        Parada(void); // constructor por defecto
        Parada(string itr, string idpp); // constructor solo con Ids
        Parada(string itr, string idpp, int v , int t); // constructor con
datos

        // Set
    void SetIdTren (string idt);
        void SetIdPP (string idpp);
        void SetNumVia (int v);
        void SetTiempoP (int t);

        // Get
        string GetIdTren(void) {return idTren;};
        string GetIdPP (void) {return idPPaso;};
        int GetNumVia (void) {return numVia;};
        int GetTiempoP (void) {return tiempoP;};


        //Operadores miembro

        string PrintParada(void);
        string PrintParada(Parada &p);


};
```

```
#endif
```

Recorrido.h

```
//Miguel Arranz Pascual PFC Mallas
//fichero Recorrido.h
//Declaración de la clase Recorrido tren dentro del módulo trenes

#pragma once
#ifndef _RECORRIDO_H_
#define _RECORRIDO_H_

#include <iostream>
#include <string>
using namespace std;
using namespace System;

class Recorrido
{
// miembros de datos
private:
        string idTren; // Tren para el que se define el recorrido.
        int paridad; // 1 o 2 según la paridad del tren en ese recorrido.
        string idTrenPrd; // un id especifico del tren según su paridad en
banda
        string idBanda; // id de la banda en donde se define el recorrido.
        string origenBanda; // punto de paso en el que el tren en su
recorrido entra en esa banda
        string finBanda; // punto de paso en el que en ese recorrido el
tren sale de la banda.


public:
        // Constructores
        Recorrido(void); // constructor por defecto
        Recorrido (string idt, int p,string idtp, string ibd, string obd,
string fbd); // constructor con datos

        // Set
        void SetIdTren (string idt);
        void SetParidad ( int p);
    void SetIdTrenPrd (string idtp);
        void SetIdBanda (string ibd);
        void SetOrigenBd (string obd);
    void SetFinBd (string fbd);

        // Get
        string GetIdTren(void) {return idTren;};
        int GetParidad(void) {return paridad;};
        string GetIdTrenPrd(void) {return idTrenPrd;};
        string GetIdBd (void) {return idBanda;};
        string GetOrigenBd(void) {return origenBanda;};
        string GetFinBd(void) {return finBanda;};
```

```
        //Operadores miembro

        string PrintRecorrido(void);
        string PrintRecorrido(Recorrido &r);

};

  #endif
```

Horario.h

```
//Miguel Arranz Pascual PFC Mallas
//fichero Horario.h
//Declaración de la clase Horario tren dentro del módulo trenes

#pragma once
#ifndef _HORARIO_H_
#define _HORARIO_H_

#include <iostream>
#include <string>
using namespace std;
using namespace System;


class Horario
{
// miembros de datos
private:
        string idPPaso; // id del punto de paso a que se refiere ese
horario.
        string idTry; // id del trayecto de acceso al siguiente punto de
paso
        string trParidad; // id del tren según paridad
        string idBanda; // identificador de la banda de recorrido
        int viaTry; // vía por la que circula en el trayecto hasta el
siguiente punto de paso.
        int viaPp; // vía de paso o estacionamiento por ese punto.
        int hLlg; // hora de llegada al pp.
        int parada; // tiempo de parada en ese pp.
        int hSld; // hora de salida hacia el siguiente pp.
        int tpMarcha; // tiempo de marcha concedido en el trayecto
        int paridad; // paridad del tren en ese punto
        int ubicPp; // ubicación del Pp en la banda

public:
        // Constructores
        Horario(void); // constructor por defecto.
        Horario(string idp, string idt, int vtr, int vpp); // constructor
mínimo sin horas.
        Horario(string idp, string idt, int vtr, int vpp, int p, int tm);
// constructor mínimo con parada.
        Horario(string idp, string idt, string trprd,
```

```
                string idbd, int vtr, int vpp, int hllg, int p, int hsld,
int tm, int prd, int ub); // constructor completo.

        // Set
        void SetIdPp ( string idp);
        void SetIdTr ( string idt);
        void SetTrParidad(string trp);
        void SetIdBanda(string idbd);
        void SetViaTry (int vtr);
        void SetViaPp (int vpp);
        void SetHllg (int hllg);
        void SetParada(int p);
        void SetHsld (int hsld);
        void SetTpMarcha(int tm);
        void SetParidad (int prd);
        void SetUbicPp ( int ub);

        // Get
        string GetIdPP(void) {return idPPaso;};
    string GetIdTry (void) {return idTry;};
        string GetTrParidad(void) {return trParidad;};
        string GetIdBanda (void) {return idBanda;};
        int GetViaTry(void) {return viaTry;};
        int GetViaPp(void) {return viaPp;};
        int GetHllg(void) {return hLlg;};
        int GetParada(void) {return parada;};
        int GetHsld(void) {return hSld;};
        int GetTpMarcha(void) {return tpMarcha;};
        int GetParidad (void) {return paridad;};
        int GetUbicPp (void) {return ubicPp;};

        //Operadores miembro

        string PrintHorario(void);

};

   #endif
```

Tren.h

```
//Miguel Arranz Pascual PFC Mallas
//fichero Tren.h
//Declaración de la clase Tren dentro del módulo trenes

#pragma once
#ifndef _TREN_H_
#define _TREN_H_

#include <iostream>
#include <string>
#include <list>
#include "Banda.h"
#include "Recorrido.h"
#include "Horario.h"
#include "Parada.h"
using namespace std;
using namespace System;

class Tren
{

//miembros de clase
public:
static list<Tren>ListaDeTrenes; // Guarda la lista de todos los trenes
creados.


// miembros de datos
private:
        string idTren; // id único del tren.
        int producto; // define el tren como viajeros o mercancias y en
cada caso subproductos que definen un color de
                                // presentación en malla y una prioridad.
        int tipo; // define la velocidad máxima del tren 80 100  160 etc..
        int longitud; // longitud en metros del tren
        string idPpOrigen; // origen del tren.
        string idPpDestino; // destino final del tren.
        int horaSalida;  // hora de salida de origen, es valor inicial para
generar el horario.
        list<Recorrido>trListaRecr; // guarda la lista de bandas con sus
respectivos puntos de entrada y salida de banda.
        list<Parada>trListaPrds; // guarda las solicitudes de paradas del
tren en su recorrido.
        list<Horario>trListaHorario; // guarda el horario del tren una vez
generado.

public:
        // Constructores
        Tren(void); // constructor por defecto.
        Tren(string idtr); // constructor con id del tren
        Tren(string idtr, int pr, int tp, int lg, string idog, string iddt,
int hsld); // constructor completo.

        // Set
    void SetIdTren (string idt);
        void SetProducto (int p);
```

```
        void SetTipo (int tp);
        void SetLongitud (int lg);
        void SetIdPpOrigen (string idppo);
        void SetIdPpDestino (string idppd);
        void SetHoraSalida (int hsld);

        // Get
        string GetIdTren(void) {return idTren;};
        int GetProducto(void) {return producto;};
        int GetTipo(void) {return tipo;};
        int GetLongitud(void) {return longitud;};
        string GetIdPpOrigen(void) {return idPpOrigen;};
        string GetIdPpDestino(void) {return idPpDestino;};
        int GetHoraSalida(void) {return horaSalida;};
        list<Recorrido> GetListRecr(void);
        list<Parada> GetListPrds(void);
        list<Horario> GetListHorario(void);

        //Operadores miembro

        string StProducto(void); // devuelve el nombre del producto del
tren
        string PrintTren(void); // presenta los datos de un tren
        string PrintListaParadas(void); // presenta las paradas solicitadas
a un tren
        string PrintListaRecorrido(void); // presenta el recorrido
solicitado para un tren
        string PrintListaHorario(void); // presenta el horario generado
para un tren
        Parada BuscarParada(string idPp); // obtiene la parada del tren en
un punto
        void ModificarViaParada(string idPp, int v); // modifica la via de
parada del tren en un punto
        void ModificarTiempoParada(string idPp, int tp); // modifica el
tiempo de  parada del tren en un punto


        //Operadores de clase
        static void AgregarRecorrido(string idtr, const Recorrido &rc); //
Agregar un recorrido a la lista correspondiente
        static void AgregarParada(string idtr, const Parada &pr); //
Agregar una parada a la lista correspondiente
        static void AgregarHorario(string idtr, const Horario &hr); //
Agregar un horario de paso a la lista correspondiente
        static void AgregarTren(const Tren &tr); // agrega un tren a la
lista de trenes.
        static string PrintLista(void); // presenta la lista de trenes.
        static Tren BuscarTren (string idtr); // obtiene un tren de la
lista dado su identificador
        static Parada BuscarParada(string idTren, string idPp); // obtiene
la parada de un tren en un punto
        static void ModificarViaParada(string idTren, string idPp, int v);
// modifica la via de parada de un tren en un punto
        static void ModificarTiempoParada(string idTren, string idPp, int
tp); // obtiene el tiempo de  parada de un tren en un punto

};

  #endif
```

A.3 Otras Secciones de Código

A.3.1 Entrada de Datos

EntradaDat.h

```
//Miguel Arranz Pascual PFC Mallas
//fichero EntradaDat.h
//Declaracion de funciones y para la captura de datos desde fichero


#pragma once
#include "StdAfx.h"
#include <fstream>
#include <iostream>
#include <cstdlib>
#include <string>
using namespace System;
using namespace std;



class EntradaDat
{
public:
  static string CargaDatos();
  static string CargarPuntosDePaso();
  static string CargarVias();
  static string CargarBandas();
  static string CargarTrayectos();
  static string CargarTrenes();
  static string CargarRecorridos();
  static string CargarParadas();
  static string IniciarGeneracíonDeHorario();
};
```

EntradaDat.cpp

```
//Miguel Arranz Pascual PFC Mallas
//fichero EntradaDat.cpp
//Funciones y para la carga de Datos de Infraestructura de los ficheros
correspondientes

#include "StdAfx.h"
#include "Banda.h"
#include "Via.h"
#include "Trayecto.h"
#include "PuntoDePaso.h"
#include "Tren.h"
#include "Inicio.h"
#include "RecorridoHorario.h"
#include "Util.h"
```

```
#include <fstream>
#include <iostream>
#include <string>
#include <cstdlib>
#include "EntradaDat.h"
using namespace System;
using namespace System::IO;
using namespace System::Collections;
using namespace std;

string EntradaDat::CargaDatos()
{
string ppDat = "";
ppDat = CargarPuntosDePaso();
ppDat = ppDat + CargarVias();
ppDat = ppDat + CargarTrayectos();
ppDat = ppDat + CargarBandas();
ppDat = ppDat + CargarTrenes();
ppDat = ppDat + CargarRecorridos();
ppDat = ppDat + CargarParadas();
//ppDat = ppDat + RecorridoHorario::GenerarHorario();
return ppDat;

};

//Función de inicio de generación de horario
string EntradaDat::IniciarGeneracíonDeHorario()
{
  string ppDat = "";
  ppDat = ppDat + RecorridoHorario::GenerarHorario();
  return ppDat;
}

// Función para cargar los datos de todos los puntos de paso y sus
trayectos correspondientes

string EntradaDat::CargarPuntosDePaso(void)
{
        int i=0;
        string idAux; // variable auxiliar para guardar la Id de pp leida
        string nombreAux; // v.aux para guardar el nombre del pp leido
        string abrevAux;  // v. aux para guardar la abrv del pp leido
        string idAnteriorAux;     // v. aux para la id del pp anterior
        string idSiguienteAux;     // v. aux para la id del pp siguiente
        string lineaAux = " "; // string aux para retorno
        PuntoDePaso ppIni = PuntoDePaso("INICIO", "     Origen", "00");  //
v aux para guardar el pp anterior
        PuntoDePaso ppFin = PuntoDePaso("FIN", "     Final", "XX");  // v
aux para guardar el pp sig
        PuntoDePaso::AgregaPP(ppIni);
        // Crea un stream para cargar los datos de pp y trayectos
        ifstream filein;
    filein.open ("PuntosDePaso.txt");
        //Detectar error en la apertura del fichero de texto
        if (filein.fail()){lineaAux = "Fallo Fichero";};
        do
        {
                filein >> idAux;
                filein >> abrevAux;
                filein >> idAnteriorAux;
```

```
                filein >> idSiguienteAux;
                getline(filein,nombreAux);
                i++;
                //Detectar error en la lectura de datos y presentar la linea
errónea
                if(filein.fail())
                {
                        lineaAux = "Fallo en lectura de fichero Puntos de
Paso, linea  ";
                        lineaAux+= Util::stConvert(i) + "\n";
                        return lineaAux;
                };
                PuntoDePaso idAux(idAux, nombreAux, abrevAux, idAnteriorAux,
idSiguienteAux);
                PuntoDePaso::AgregaPP(idAux);
        }while (!filein.eof());
        filein.close();
        PuntoDePaso::AgregaPP(ppFin);
        lineaAux = lineaAux + "Leido y cargado correctamente el archivo de
Puntos de Paso" + "\n";
        return lineaAux;
};

// Función para cargar los datos de todos los trayectos existentes

string EntradaDat::CargarTrayectos(void)
{
        int i=0;
        string idTryAux;  // v. aux para la id del trayecto
        double longAux;   // v. aux para la longitud del trayecto
        int capAux;  // v. aux para la capacidad  del trayecto
        int tipoAux; // v. aux para el tipo del trayecto
    int velocAux; // v. aux para la velocidad del trayecto
        string idInicioAux;     // v. aux para la id del pp inicio
        string idFinAux;     // v. aux para la id del pp fin
        string lineaAux = " "; // string aux para retorno
        // Crea un stream para cargar los datos del trayecto
        ifstream filein;
    filein.open ("Trayectos.txt");
        //Detectar error en la apertura del fichero de texto
        if (filein.fail()){lineaAux = "Fallo Fichero Trayectos";};
        do
        {
                filein >> idTryAux;
                filein >> longAux;
                filein >> capAux;
                filein >> tipoAux;
                filein >> velocAux;
                filein >> idInicioAux;
                filein >> idFinAux;
                i++;
                //Detectar error en la lectura de datos y presentar la linea
errónea
                if(filein.fail())
                {
                        lineaAux = "Fallo en lectura de fichero Trayectos,
linea  ";
                        lineaAux+= Util::stConvert(i) + "\n";
                        return lineaAux;
                };
```

```
                Trayecto idTryAux (idTryAux, longAux, capAux, tipoAux,
velocAux, idInicioAux, idFinAux);
                Trayecto::AgregarTrayecto(idTryAux);
                PuntoDePaso::AgregarTrayecto(idInicioAux,idTryAux);
        }while (!filein.eof());
        filein.close();
        lineaAux = lineaAux + "Leido y cargado correctamente el archivo de
Trayectos" + "\n";
        return lineaAux;
};

// Función para cargar los datos de todos las vías correspondientes a cada
punto de paso

string EntradaDat::CargarVias(void)
{
        int i=0;
    string idPPAux;     // variable auxiliar para guardar la Id de pp
        int numViaAux; // v. aux para guardar el num de la via leida
        int longViaAux;  // v. aux para guardar la longitud de la via leida
        int andenViaAux;     // v. aux para el valor si no anden
        string idViaAux; // variable aux para crear el id de via
        string lineaAux = " "; // string aux para retorno

        // Crea un stream para cargar los datos de las vías
        ifstream filein;
    filein.open ("Vias.txt");
        //Detectar error en la apertura del fichero de texto
        if (filein.fail()){lineaAux = "Fallo Fichero Vias";};
        do
        {
                filein >> idPPAux;
                filein >> numViaAux;
                filein >> longViaAux;
                filein >> andenViaAux;
                i++;
                if(filein.fail())
                {
                        lineaAux = "Fallo en lectura de fichero de Vias,
linea  ";
                        lineaAux+= Util::stConvert(i) + "\n";
                        return lineaAux;
                };
                idViaAux = idPPAux + "-" + Util::stConvert(numViaAux);
                Via idViaAux (idViaAux, numViaAux, longViaAux, andenViaAux);
// creamos la via correspondiente
                //PuntoDePaso ppVia = PuntoDePaso::BuscarPP(idPPAux);
                Via ptVia = idViaAux;
                PuntoDePaso::AgregarVia(idPPAux, idViaAux);
                //lineaAux = lineaAux + ppVia.GetPPNombre() + "  " +
idViaAux.GetIdVia() + "  :" + idViaAux.PrintVia(idViaAux);

        }while (!filein.eof());
        filein.close();
        lineaAux = lineaAux + "Leido y cargado correctamente el fichero de
Vias  "+ "\n";
        return lineaAux;
};
```

```
// Función para cargar los datos de todas las bandas y generar las lista de
sus puntos de paso correspondientes

string EntradaDat::CargarBandas(void)
{
	int i=0;
	string idBdAux;	// variable auxiliar para guardar la Id de la
Banda
	int numPPBdAux; // v. aux para guardar el num de PP de la Banda
leida
	string bdIncioAux;  // v. aux para guardar el PP Inicio de la banda
leida
	string bdFinAux; // variable aux para guardar el PP Fin  de banda
leida
	string lineaAux = " "; // string aux para retorno

	// Creamos un stream para cargar los datos de las bandas
	ifstream filein;
    filein.open ("Bandas.txt");
	//Detectar error en la apertura del fichero de texto
	if (filein.fail()){lineaAux = "Fallo Fichero Bandas";};
	do
	{
		filein >> idBdAux;
		filein >> numPPBdAux;
		filein >> bdIncioAux;
		filein >> bdFinAux;
		i++;
		if(filein.fail())
		{
			lineaAux = "Fallo en lectura de fichero de Bandas,
linea  ";
			lineaAux+= Util::stConvert(i) + "\n";
			return lineaAux;
		};
		PuntoDePaso ppIncioBd = PuntoDePaso::BuscarPP(bdIncioAux);
		PuntoDePaso ppFinBd = PuntoDePaso::BuscarPP(bdFinAux);
		Banda idBdAux (idBdAux, numPPBdAux, ppIncioBd, ppFinBd); //
creamos la banda correspondiente
		Banda::AgregarBanda(idBdAux);
		Banda::GenerarListaPP(idBdAux);
	}while (!filein.eof());
	filein.close();
	lineaAux = lineaAux + "Leido y cargado correctamente el fichero de
Bandas" + "\n";
	return lineaAux;
};

// Función para cargar los datos generales de trenes e incorporarlos a la
lista de trenes

string EntradaDat::CargarTrenes(void)

{
	int i=0;
	string idTrx;  // v. aux para la id del tren
	int productox; // v. aux para el producto tren
	int tipox; // v. aux para el tipo del tren
	int longx;   // v. aux para la longitud del tren
	string idppOrigenx;	// v. aux para la id del pp origen
```

```
        string idppDestinox;     // v. aux para la id del pp destino
        int hsldx ; // v. aux para la hora de salida
        string lineaAux = " "; // string aux para retorno
        // Crea un stream para cargar los datos del trayecto
        ifstream filein;
    filein.open ("Trenes.txt");
        //Detectar error en la apertura del fichero de texto
        if (filein.fail()){lineaAux = "Fallo Fichero de Trenes";};
        do
        {
                filein >> idTrx;
                filein >> productox;
                filein >> tipox;
                filein >> longx;
                filein >> idppOrigenx;
                filein >> idppDestinox;
                filein >> hsldx;
                i++;
                //Detectar error en la lectura de datos y presentar la linea
errónea
                if(filein.fail())
                {
                        lineaAux = "Fallo en lectura de fichero Trenes,
linea  ";
                        lineaAux+= Util::stConvert(i) + "\n";
                        return lineaAux;
                };
                Tren idTrx (idTrx, productox, tipox, longx, idppOrigenx,
idppDestinox, hsldx);
                Tren::AgregarTren(idTrx);
        }while (!filein.eof());
        filein.close();
        lineaAux = lineaAux + "Leido y cargado correctamente el archivo de
Trenes" + "\n";
        return lineaAux;
};

// Función para cargar del fichero de texto correspondiente los datos del
recorrido de un tren
// e incluirlos en la lista de recorrio del mismo.

string EntradaDat::CargarRecorridos()
{
        int i=0;
        string idRcx; // id de recorrido
        string idTrenx;  // v. aux para la id del tren
        int paridadx; // v. aux para la paridad tren
        string idtpx; // v. aux para el id del tren en cada paridad
        string idBandax;   // v. aux para la banda del recorrido
        string idppOrigenBdx;     // v. aux para la id del pp origen en
banda
        string idppFinBdx;     // v. aux para la id del pp salida de banda
        string lineaAux = " "; // string aux para retorno
        // Crea un stream para cargar los datos del trayecto
        ifstream filein;
    filein.open ("Recorridos.txt");
        //Detectar error en la apertura del fichero de texto
        if (filein.fail()){lineaAux = "Fallo Fichero de Recorridos";};
        do
        {
```

```
                filein >> idTrenx;
                filein >> idtpx;
                filein >> paridadx;
                filein >> idBandax;
                filein >> idppOrigenBdx;
                filein >> idppFinBdx;
                i++;
                //Detectar error en la lectura de datos y presentar la linea
errónea
                if(filein.fail())
                {
                        lineaAux = "Fallo en lectura de fichero de
Recorridos, linea  ";
                        lineaAux+= Util::stConvert(i) + "\n";
                        return lineaAux;
                };
                idRcx = idTrenx + "-" + Util::stConvert(i);
                Recorrido idRcx (idRcx, paridadx, idtpx, idBandax,
idppOrigenBdx, idppFinBdx);
                Tren::AgregarRecorrido(idTrenx,idRcx);
        }while (!filein.eof());
        filein.close();
        lineaAux = lineaAux + "Leido y cargado correctamente el fichero de
Recorridos" + "\n";
        return lineaAux;
};


//Función para cargar del fichero de texto correspondiente los datos de las
paradas solicitadas
//para un tren e incluirlas en la lista de paradas del mismo.

string EntradaDat::CargarParadas()
{
        int i=0;
        string idPrx; // id de la parada.
        string idTrenx;  // v. aux para la id del tren.
        string idPPx; // v. aux para el id del punto de paso.
        int numViax;   // v. aux para la via solicitada.
        int tiempoPx;     // v. aux para el tiempo de parada solicitado.
        string lineaAux = " "; // string aux para retorno
        // Crea un stream para cargar los datos del trayecto
        ifstream filein;
    filein.open ("Paradas.txt");
        //Detectar error en la apertura del fichero de texto
        if (filein.fail()){lineaAux = "Fallo Fichero de Paradas";};
        do
        {
                filein >> idTrenx;
                filein >> idPPx;
                filein >> numViax;
                filein >> tiempoPx;
                i++;
                //Detectar error en la lectura de datos y presentar la linea
errónea
                if(filein.fail())
                {
                        lineaAux = "Fallo en lectura de fichero de Paradas,
linea  ";
                        lineaAux+= Util::stConvert(i) + "\n";
```

```
                        return lineaAux;
                };
                idPrx = idTrenx + "-" + idPPx;
                Parada idPrx (idTrenx, idPPx, numViax, tiempoPx);
                Tren::AgregarParada(idTrenx,idPrx);
        }while (!filein.eof());
        filein.close();
        lineaAux = lineaAux + "Leido y cargado correctamente el fichero de
Paradas solicitadas" + "\n";
        return lineaAux;
   };
```

A.3.2 Generación de Recorridos Horarios

RecorridoHorario.h

```
//Miguel Arranz Pascual PFC Mallas
//fichero RecorridoHorario.h
//Declaracion de funciones y para la generación del horario de los trenes

#pragma once

#ifndef _RECORRIDOHORARIO_H_
#define _RECORRIDOHORARIO_H_

#include "StdAfx.h"
#include "Banda.h"
#include "Tren.h"
#include "Inicio.h"
#include "Util.h"
#include "Trayecto.h"
#include "Parada.h"
#include "PuntoDePaso.h"
#include "PuntoPasoBanda.h"
#include <fstream>
#include <iostream>
#include <cstdlib>
#include <string>

using namespace System;
using namespace std;



class RecorridoHorario
{
//variables necesarias


// Funciones definidas para generar el recorrido horario del tren
public:
        static string GenerarHorario(void);
        static void GenerarHorarioTren(Tren trx);
        static void ObtenerListaHorarios(Tren tr, Recorrido rcr);
        static void NuevoHorarioActual();
        static Trayecto ObtenerTrayecto(string abrAnt, string abrAct);
        static int DefinirViaTrayecto(Trayecto tr,int pd);
        static int CalcularTiempoMarcha(string idtr, Trayecto ty);
        static int CalcularHoraLlegada(int hact, int tc);
        static Parada ObtenerParada(string idtr, string pp);
        static int CalcularTiempoParada(Parada pr);
        static int ObtenerViaPp(int pd, Parada pr, Trayecto ty);
        static int ObtenerHoraSalida(int hllg, int tp);

};
```

```
#endif
```

RecorridoHorario.cpp

```
//Miguel Arranz Pascual PFC Mallas
//fichero RecorridoHorario.cpp
//Funciones necesarias para la generación de las listas de horario de los
trenes


#include "StdAfx.h"

#include "RecorridoHorario.h"
#include "Inicio.h"
#include <fstream>
#include <iostream>
#include <string>
#include <cstdlib>

using namespace System;
using namespace System::IO;
using namespace System::Collections;
using namespace std;

// variables auxiliares
        string ppActual, ppSig, ppAnt, ppFinal, ppOrigen, ppDestino;
        string idTrPard, idBanda, abrAnt, abrAct;
        int paridad, viaTry, viaPp, horaActual, horaLlegada, horaSalida,
tpParada, ubicPp;
        double velocidad, tryLong;
        int tmarcha = 0;
        string idTr;
        string idTryct;
        Tren trx;
        Parada prdPpAct;
        Recorrido rcr;
        Trayecto tryActual;

// Recorre la lista de trenes y  genera el recorrido horario de cada uno
string RecorridoHorario::GenerarHorario()
{

        string staux="";
        list<Tren>::iterator it = Tren::ListaDeTrenes.begin();
        while (it != Tren::ListaDeTrenes.end())
        {
                trx = *it;
                GenerarHorarioTren(trx);
                it++;
        }
        staux = "Generados Horarios de Trenes";
        return staux;
};

// Crea los horarios de paso de un tren de origen a destino y
// los guarda en la lista horario del tren
void RecorridoHorario::GenerarHorarioTren(Tren trx)
```

```
{
// inicializar valores
        idTr = trx.GetIdTren();
        ppOrigen = trx.GetIdPpOrigen();
        ppActual = ppOrigen;
        ppDestino =trx.GetIdPpDestino();
        ppAnt = "INICIO";
        abrAnt ="00";
        viaTry = 1;
        viaPp = 1;
        horaActual = trx.GetHoraSalida();
        horaLlegada = 0;
        horaSalida = horaActual;
        tpParada = 0;

        // Para cada recorrido del tren generar y añadir los horarios
correspondientes
        list<Recorrido> listRcr = trx.GetListRecr();
        list<Recorrido>::iterator itRc = listRcr.begin();

        while (itRc != listRcr.end())
        {
                rcr = *itRc;
         ObtenerListaHorarios(trx,rcr);
                itRc++;
        }

};

//Recorre para cada banda del unrecorrido los pps segun la paridad y genera
cada horario
// añadiendo cada horario generado al tren
void RecorridoHorario::ObtenerListaHorarios(Tren tr, Recorrido rcr)
{
        Banda bnd = Banda::BuscarBanda(rcr.GetIdBd());
        list<PuntoPasoBanda> listBdPps = bnd.GetListPP();
        int activador = 0;
        paridad = rcr.GetParidad();
        idTrPard = rcr.GetIdTrenPrd();
        idBanda = rcr.GetIdBd();

        //segun la paridad del tren se reorre la lista de pp de la banda
        if (paridad == 1) {
                list<PuntoPasoBanda>::iterator itpp = listBdPps.begin();
                while (itpp != listBdPps.end())
                {
                //Activar si encontrado inicio banda desactivar en fin banda
                        if (itpp-
>GetPtoPaso().GetIdPP().compare(rcr.GetOrigenBd())==0){activador =1;};
                        if ((activador ==1))
                        {
                                //Obtener datos
                                ppActual = itpp->GetPtoPaso().GetIdPP();
                                abrAct = itpp->GetPtoPaso().GetPPAbr();
                                ubicPp = itpp->GetUbicBanda();
                                tryActual = ObtenerTrayecto(abrAct, abrAnt);
                                NuevoHorarioActual();
                        }
                        if (itpp-
>GetPtoPaso().GetIdPP().compare(rcr.GetFinBd())==0){
```

```
                                activador =0;
                                return ;
                        };
                        itpp++;
                }
        }
        if (paridad == 2) {
                list<PuntoPasoBanda>::reverse_iterator ritpp =
listBdPps.rbegin();
                while (ritpp != listBdPps.rend())
                {
       //Activar si encontado inicio banda desactivar en fin banda
                        if (ritpp-
>GetPtoPaso().GetIdPP().compare(rcr.GetOrigenBd())==0){activador =1;};

                        if ((activador ==1))
                        {
                                //Obtener datos
                                ppActual = ritpp->GetPtoPaso().GetIdPP();
                                abrAct = ritpp->GetPtoPaso().GetPPAbr();
                                ubicPp = ritpp->GetUbicBanda();
                                tryActual = ObtenerTrayecto(abrAnt, abrAct);
                                NuevoHorarioActual();

                        }
                        if (ritpp-
>GetPtoPaso().GetIdPP().compare(rcr.GetFinBd())==0){
                                activador =0;
                                return ;
                        };
                        ritpp++;
                }
        }
        return ;
};

//Genera el horario con los datos actuales y lo guarda en la lista horarios
del tren actual
void RecorridoHorario::NuevoHorarioActual()
{
        //Obtener datos para horario en pp actual

        viaTry = DefinirViaTrayecto (tryActual, paridad);
        tmarcha = CalcularTiempoMarcha(idTr, tryActual);
        horaLlegada = CalcularHoraLlegada(horaActual, tmarcha);
        prdPpAct = ObtenerParada(idTr, ppActual);
        tpParada = CalcularTiempoParada(prdPpAct);
        viaPp = ObtenerViaPp(paridad, prdPpAct,tryActual);
        horaSalida = ObtenerHoraSalida(horaLlegada, tpParada);

        //Crear un nuevo horario y agregarlo al tren
        Horario hr(ppActual, tryActual.GetIdTrayecto(), idTrPard, idBanda,
                viaTry, viaPp, horaLlegada, tpParada, horaSalida, tmarcha,
paridad, ubicPp);
        Tren::AgregarHorario(idTr, hr);
        ppAnt = ppActual;
        abrAnt = abrAct;
        ppFinal = ppActual;
        horaActual = horaSalida;
```

```
};


// devuelve el trayecto que conecta los pp actual y siguiente
// y obtiene el tiempo de marcha y la via
Trayecto RecorridoHorario::ObtenerTrayecto(string abrAnt, string abrAct)
{
        Trayecto trSig;
        string idTry = abrAnt + "-" + abrAct;
        trSig = Trayecto::BuscarTry(idTry);

        return trSig;
};

//devuelve la via por defecto por la que circula el tren el trayecto
//si via UNICa es via 1 si no 1 o 2 segun la paridad del tren
int RecorridoHorario::DefinirViaTrayecto(Trayecto tr,int pd)
{
        int vtry =1;
        if (tr.GetTipo()==2){vtry= pd;}
        return vtry;
};

//Devuelve el tiempo en el trayecto en función de
//la velocidad del tren la vmax del trayecto y la longitud en kms.
int RecorridoHorario::CalcularTiempoMarcha(string idtr, Trayecto ty)
{
        int tpmch =0;
        velocidad = trx.GetTipo();
        // si la vmax del trayecto es menor es la establecida
        if (ty.GetVMax()<velocidad){velocidad=ty.GetVMax();}
        velocidad = ((velocidad *1000)/3600)-3;
        double  longTry = ((ty.GetLongitud())*1000);
        // calcula el tiempo concedido
        tpmch = (int)(longTry/velocidad);
        return tpmch;
};

//Devuelve la hora de llegada al pp actual en función
// de la hora actual = hora salida anterior y el tiempo concedido
int RecorridoHorario::CalcularHoraLlegada(int hact, int tc)
{
        int hllg =0;
        hllg = hact + tc;

        return hllg;

};

// devuelve la parada solicitada para el tren en ese pp si no
// devuelve una parada tipo con via = 0 y tiempo -1
Parada RecorridoHorario::ObtenerParada(string idtr, string pp)
{
        //Tren tr = Tren::BuscarTren(idtr);
        Parada trprd(idtr, pp, 0, -1);
        string idaux="";
    list<Parada> listPrTr = trx.GetListPrds();
        list<Parada>::iterator it = listPrTr.begin();
        while (it != listPrTr.end())
```

```
        {
                idaux = it->GetIdPP();
                int cmpSt = idaux.compare(pp);
                if(cmpSt==0){return *it;};
                it++;
        }
        return trprd;
};

//Devuelve el tiempo concedido para parada en ese punto
int RecorridoHorario::CalcularTiempoParada(Parada pr)
{
        int tpprd =0;
        tpprd = pr.GetTiempoP();
        return tpprd;
};

//Devuelve la via por la que el tren va a pasar o efectuar parada
int RecorridoHorario::ObtenerViaPp(int pd, Parada pr, Trayecto ty)
{
        int vprd =0;
        //Si parada la via solicitada en la parada
        if (pr.GetNumVia()!=0){vprd = pr.GetNumVia();}
        // Si no si es via UNICA via 1 si no la de la paridad del tren en
el trayecto.
        else if (ty.GetTipo()==2) {vprd = pd;}
        else {vprd = 1;}
        return vprd;
};

//Obtener la hora de salida a partir de la hora de llegada y el tiempo de
parada
int RecorridoHorario::ObtenerHoraSalida(int hllg, int tp)
{
        int hsld =0;
        if (tp<0) {tp = 0;};
        hsld = hllg + (tp*60);

        return hsld;
  };
```

A.3.3 Detección de Incompatibilidades

IncomVia.h

```
//Miguel Arranz Pascual PFC Mallas
//fichero IncomVia.h
//Declaración de la clase IncVia dentro del módulo incompatibilidades

#pragma once
#include "StdAfx.h"
#include "Banda.h"
#include "Tren.h"
#include "Util.h"
#include <iostream>
#include <string>
#include <list>
using namespace std;
using namespace System;


class IncomVia
{

//miembros de clase
public:
        static list<IncomVia>ListaDeIncomVia;  // Guarda la lista de las
incom halladas

//miembros de datos
private:
        string idInc; // identificador idTren-idPp
        int tipoInc; // tipo 0 via inexistente, 1 via sin anden, 2 via de
menor longitud;
        string idTren; // id del tren;
        string idPp; // id del punto de paso
        int numVia; // nº de via implicada


public:
        //constructores
        IncomVia(void); // constructor por defecto
        IncomVia(string idinc); // solo id
        IncomVia(string idinc, int tpi, string idtr, string idpp, int
numv); // constructor con datos

        //Set
        void SetIdInc(string idinc);
        void SetTipoInc(int tpi);
        void SetIdTren(string idtr);
        void SetIdPp(string idpp);
        void SetNumVia(int numv);

        //Get
        string GetIdInc(void) {return idInc;};
        int GetTipoInc(void) {return tipoInc;};
        string GetIdTren(void) {return idTren;};
        string GetIdPp(void) {return idPp;};
```

```
        int GetNumVia(void) {return numVia;};

        //Operadores miembro
        string PrintIncomVia(void);
        string PrintIncomVia(IncomVia &incv);

        //operadores de clase
        public:
        static void AgregarIncomVia(const IncomVia incVia);
        static string PrintLista(void);
        static string PrintLista(int tpi);
        static void GenerarIncomVia(void);
        static IncomVia BuscarIncVia(string idInc); //Devuelve la inc con
el identificador aportado
        static string DetectarIncViaError();
        static string DetectarIncViaAnden();
        static string DetectarIncViaLongitud();


};
```

IncomVia.cpp

```
//Miguel Arranz Pascual PFC Mallas
//fichero IncomVia.h
//Funciones de la clase IncVia dentro del módulo incompatibilidades

#include "StdAfx.h"
#include "IncomVia.h"
#include <iostream>
#include <string>
#include <list>
using namespace std;
using namespace System;

// Inicialización de lista
list<IncomVia> NwListIncVia;
list<IncomVia>IncomVia::ListaDeIncomVia = NwListIncVia;


//constructores

// constructor por defecto
        IncomVia::IncomVia(void)
        {

                idInc = "00000-ORIGEN";
                tipoInc = 0;
            idTren = "00000";
            idPp = "ORIGEN";
            numVia = 0;

        };

// solo id
```

```
        IncomVia::IncomVia(string idinc)
        {
                idInc = idinc;
                tipoInc = 0;
            idTren = "00000";
            idPp = "ORIGEN";
            numVia = 0;


        };

 // constructor con datos
        IncomVia::IncomVia(string idinc, int tpi, string idtr, string idpp,
int numv)
        {
                idInc = idinc;
                tipoInc = tpi;
            idTren = idtr;
            idPp = idpp;
            numVia = numv;

        };

//Funciones miembro Set
        void IncomVia::SetIdInc(string idinc)
        {
                idInc = idinc;

        };

        void IncomVia::SetTipoInc(int tpi)
        {
                tipoInc = tpi;

        };

        void IncomVia::SetIdTren(string idtr)
        {
                idTren = idtr;

        };

        void IncomVia::SetIdPp(string idpp)
        {
                idPp = idpp;

        };

        void IncomVia::SetNumVia(int numv)
        {
                 numVia = numv;
        };

//Operadores miembro

//Presenta los datos de la incompatibilidad de via
string IncomVia::PrintIncomVia(void)

{
        //variables auxiliares
```

```
        string stInc = "";
        string nmPp = "";
        string stidpp = "";
        string sttr = "";
        string sttperr ="";
        string stnumv = "";

        nmPp = PuntoDePaso::BuscarPP(this->GetIdPp()).GetPPNombre();
        sttr = this->GetIdTren();
        stidpp = this->GetIdPp();
        stnumv = Util::stConvert(this->GetNumVia());
        // Segun el tipo de inc creamos el mensaje de error
        if (this->GetTipoInc() == 0){sttperr = " La via solicitada no
existe en ese punto";}
        if (this->GetTipoInc() == 1){sttperr = " Solicitada parada a tren
de viajeros en via sin anden";}
        if (this->GetTipoInc() == 2)
        {
                Via vx = PuntoDePaso::ObtenerViaDePP (stidpp, stidpp + "-" +
stnumv);
                Tren trx = Tren::BuscarTren(sttr);
                sttperr = " La via de " + Util::stConvert(vx.GetLongitud())+
" mtrs. no admite tren de " + Util::stConvert(trx.GetLongitud()) + "
mtrs.";
        }
        //else {sttperr = " Error sin identificar";};
        stInc = stInc + "Solicitada parada a tren  " + sttr + "  en la via:
" + stnumv + "  de " +  nmPp + "\n";
        stInc = stInc + " Error : " + sttperr + + "\n";


        return stInc;
};

//Presenta los datos de la incompatibilidad de ese tipo;
string IncomVia::PrintIncomVia(IncomVia &incv)

{
        //variables auxiliares
        string stInc = "";
        string nmPp = "";
        string stidpp = "";
        string sttr = "";
        string sttperr ="";
        string stnumv = "";

        nmPp = PuntoDePaso::BuscarPP(incv.GetIdPp()).GetPPNombre();
        sttr = incv.GetIdTren();
        stidpp = incv.GetIdPp();
        stnumv = Util::stConvert(incv.GetNumVia());
        // Segun el tipo de inc creamos el mensaje de error
        if (incv.GetTipoInc() == 0){sttperr = " La via solicitada no existe
en ese punto";}
        if (incv.GetTipoInc() == 1){sttperr = " Solicitada parada a tren de
viajeros en via sin anden";}
        if (incv.GetTipoInc() == 2)
        {
                Via vx = PuntoDePaso::ObtenerViaDePP (stidpp, stidpp + "-" +
stnumv);
                Tren trx = Tren::BuscarTren(sttr);
```

```
                sttperr = " La via de " + Util::stConvert(vx.GetLongitud())+
" mtrs. no admite tren de " + Util::stConvert(trx.GetLongitud()) + "
mtrs.";
        }
        else {sttperr = " Error sin identificar";};
        stInc = stInc + "Solicitada parada a tren  " + sttr + "  en la via:
" + stnumv + "  de " +  nmPp + "\n";
        stInc = stInc + " Error : " + sttperr + + "\n";


        return stInc;
};

//Presenta la lista de todas las incompatibilidades de via
string IncomVia::PrintLista(void)
{
        string stLstIncomVia = "";
        IncomVia incVia;
        list<IncomVia>::iterator itinc = IncomVia::ListaDeIncomVia.begin();
        while ( itinc != IncomVia::ListaDeIncomVia.end())
        {
                incVia = *itinc;
                stLstIncomVia = stLstIncomVia + incVia.PrintIncomVia();
                itinc++;
        }
        return stLstIncomVia;
};

//Presenta la lista de todas las incompatibilidades de via del tipo
especificado
string IncomVia::PrintLista(int tpi)
{
        string stLstIncomVia = "";
        IncomVia incVia;
        list<IncomVia>::iterator itinc = IncomVia::ListaDeIncomVia.begin();
        while ( itinc != IncomVia::ListaDeIncomVia.end())
        {
                incVia = *itinc;
                if (itinc->GetTipoInc()== tpi)
                {
                        stLstIncomVia = stLstIncomVia +
incVia.PrintIncomVia();
                }
                itinc++;
        }
        return stLstIncomVia;

};

//Agrego una nueva instancia a la lista
void IncomVia::AgregarIncomVia(const IncomVia incVia)
{
        IncomVia::ListaDeIncomVia.push_back(incVia);
};

//Busca una inc con la identificación aportada
IncomVia IncomVia::BuscarIncVia(string idInc)
{
        IncomVia incVia("00000-INICIO",0,"00000", "INICIO", 0);
        string idx = "";
```

```
        list<IncomVia>::iterator itinc = IncomVia::ListaDeIncomVia.begin();
        while ( itinc != IncomVia::ListaDeIncomVia.end())
        {

                idx = itinc->GetIdInc();
                int cmpSt = idx.compare(idInc);
                if (cmpSt == 0){return *itinc;};
                itinc++;
        }
        return incVia;


};
//Recorre la lista de trenes y paradas solicitadas buscando inc via
//e incorporandolas a la lista
void IncomVia::GenerarIncomVia(void)
{
        //variables auxiliares
        string idInc = "";
        string idPp = "";
        string idVia = "";
        string nmPp = "";
        int numVia = 0;
        Via vp;

        //Borrar las incompatibilidades anteriores
        IncomVia::ListaDeIncomVia.clear();

        //Para cada tren de la lista de trenes
        list<Tren>::iterator ittr = Tren::ListaDeTrenes.begin();
        while (ittr != Tren::ListaDeTrenes.end())
        {
                //Para cada parada de la lista de paradas
                list<Parada> lstPrds = ittr->GetListPrds();
                list<Parada>::iterator itlp = lstPrds.begin();
                while (itlp != lstPrds.end())
                        {
                            //Obtenemos la via correspondiente
                                idPp = itlp->GetIdPP();
                                numVia = itlp->GetNumVia();
                                idVia = idPp + "-" + Util::stConvert(numVia);
                                vp = PuntoDePaso::ObtenerViaDePP (idPp,
idVia);

                                //Hacemos las coomprobaciones de error y si
son positivas
                                //creanmos una nueva incidencia y la añadimos
a la lista

                                // Si la via no existe es 0
                                if (vp.GetNumVia() == 0)
                                {
                                        idInc = ittr->GetIdTren() + "-" +
itlp->GetIdPP();
                                        IncomVia idInc(idInc, 0, ittr-
>GetIdTren(), itlp->GetIdPP(),itlp->GetNumVia());
                                        IncomVia::AgregarIncomVia(idInc);

                                }
```

```
                                //Si el tren es viajeros y la via no tiene
anden
                                if ((vp.GetNumVia() != 0)&(vp.GetAnden() ==
0)&(ittr->GetProducto()<3))
                                {
                                        idInc = ittr->GetIdTren() + "-" +
itlp->GetIdPP();
                                        IncomVia idInc(idInc, 1, ittr-
>GetIdTren(), itlp->GetIdPP(),itlp->GetNumVia());
                                        IncomVia::AgregarIncomVia(idInc);;
                                }
                                //Si la longitud es menor que la del tren
                                if ((vp.GetLongitud()>0) &
((vp.GetLongitud()- ittr->GetLongitud())<20))
                                {
                                        idInc = ittr->GetIdTren() + "-" +
itlp->GetIdPP();
                                        IncomVia idInc(idInc, 2, ittr-
>GetIdTren(), itlp->GetIdPP(),itlp->GetNumVia());
                                        IncomVia::AgregarIncomVia(idInc);
                                }
                                itlp++;
                        };

                ittr++;
        };


};



//Función que devuelve información sobre las paradas solicitadas a trenes
// en puntos y vias inexistentes.

string IncomVia::DetectarIncViaError()
{
        //variables auxiliares
        string stInc = "";
        string idPp = "";
        string idVia = "";
        string nmPp = "";
        int numVia = 0;
        int n = 1;
        Via vp;

        //Iniciamos la busqueda de errores
        stInc = stInc + "RELACION DE PARADAS SOLICITADAS EN VÍAS
INEXITENTES :  " + "\n";
        stInc = stInc + " Parada solicitada:
Error:" + "\n";
        //Para cada tren de la lista de trenes
        list<Tren>::iterator ittr = Tren::ListaDeTrenes.begin();
        while (ittr != Tren::ListaDeTrenes.end())
        {
                //Para cada parada de la lista de paradas
                list<Parada> lstPrds = ittr->GetListPrds();
                list<Parada>::iterator itlp = lstPrds.begin();
                while (itlp != lstPrds.end())
```

```
                        {
                            //Obtenemos la via correspondiente
                                idPp = itlp->GetIdPP();
                                numVia = itlp->GetNumVia();
                                idVia = idPp + "-" + Util::stConvert(numVia);
                                vp = PuntoDePaso::ObtenerViaDePP (idPp,
idVia);
                                // Si la via no existe es 0
                                if (vp.GetNumVia() == 0)
                                {
                                        nmPp =
PuntoDePaso::BuscarPP(idPp).GetPPNombre();
                                        string stnumv =
Util::stConvert(numVia);
                                        string sttperr = "          :la via
no existe en ese punto; ";
                                        stInc = stInc + Util::stConvert(n) +
"\t" + ".- Tren " + ittr->GetIdTren() +  " en ";
                                        stInc = stInc + itlp->GetIdPP() + " -
" + "\t" + nmPp + " en via: " + stnumv + "\t" + sttperr + "\n";
                                n++;
                                }

                                itlp++;
                        };

                ittr++;
        };
        return stInc;
};

//Función que devuelve información sobre las paradas solicitadas a trenes
de
//viajeros en vías sin anden
string IncomVia::DetectarIncViaAnden()
{
        //variables auxiliares
        string stInc = "";
        string idPp = "";
        string idVia = "";
        string nmPp = "";
        int numVia = 0;
        int n = 1;
        Via vp;

        //Iniciamos la busqueda de errores
        stInc = stInc + "RELACION DE PARADAS DE TRENES DE VIAJEROS
SOLICITADAS EN VÍAS SIN ANDEN :  " + "\n";
        stInc = stInc + " Parada solicitada:
Error:" + "\n";
        //Para cada tren de la lista de trenes

        list<Tren>::iterator ittr = Tren::ListaDeTrenes.begin();
        while (ittr != Tren::ListaDeTrenes.end())
        {

                //Si el tren es de viajeros
                if(ittr->GetProducto()<3)
                        {
                                //Para cada parada de la lista de paradas
```

```
                                list<Parada> lstPrds = ittr->GetListPrds();
                                list<Parada>::iterator itlp =
lstPrds.begin();
                                while (itlp != lstPrds.end())
                                {
                                        //Obtenemos la via correspondiente
                                        idPp = itlp->GetIdPP();
                                        numVia = itlp->GetNumVia();
                                        idVia = idPp + "-" +
Util::stConvert(numVia);
                                        vp = PuntoDePaso::ObtenerViaDePP
(idPp, idVia);
                                        // Si la via carece de anden
                                if (vp.GetAnden() == 0)
                                {
                                        nmPp =
PuntoDePaso::BuscarPP(idPp).GetPPNombre();
                                        string stnumv =
Util::stConvert(numVia);
                                        string sttperr = "         :esta via
carece de anden; ";
                                        if(vp.GetLongitud() == 0){sttperr = "
:la via no existe en ese punto; ";};
                                        stInc = stInc + Util::stConvert(n) +
"\t" + ".-  Tren de viajeros " + ittr->GetIdTren() +  " en ";
                                        stInc = stInc  + itlp->GetIdPP() + "
- " + "\t" + nmPp + " en via: " + stnumv + "\t" + sttperr + "\n";
                                n++;
                                }
                                itlp++;
                                };

        };
                ittr++;
        };
        return stInc;
};


//Función que devuelve información sobre las paradas solicitadas a trenes
de
// en vías de longitud inferior a la del tren.

string IncomVia::DetectarIncViaLongitud()
{
        //variables auxiliares
        string stInc = "";
        string idPp = "";
        string idVia = "";
        string nmPp = "";
        int numVia = 0;
        int n = 1;
        Via vp;

        //Iniciamos la busqueda de errores
        stInc = stInc + "RELACION DE PARADAS SOLICITADAS EN VÍAS EN VÍAS DE
LONGITUD MENOR A LA DEL TREN :  " + "\n";
        stInc = stInc + " Parada solicitada:
Error:" + "\n";
        //Para cada tren de la lista de trenes
```

```
        list<Tren>::iterator ittr = Tren::ListaDeTrenes.begin();
        while (ittr != Tren::ListaDeTrenes.end())
        {
                //Para cada parada de la lista de paradas
                list<Parada> lstPrds = ittr->GetListPrds();
                list<Parada>::iterator itlp = lstPrds.begin();
                while (itlp != lstPrds.end())
                        {
                            //Obtenemos la via correspondiente
                                idPp = itlp->GetIdPP();
                                numVia = itlp->GetNumVia();
                                idVia = idPp + "-" + Util::stConvert(numVia);
                                vp = PuntoDePaso::ObtenerViaDePP (idPp,
idVia);
                                // Si la longitud de via es menor que la del
tren
                                if ((vp.GetLongitud()- ittr-
>GetLongitud())<20)
                                {
                                        nmPp =
PuntoDePaso::BuscarPP(idPp).GetPPNombre();
                                        string stnumv =
Util::stConvert(numVia);
                                        string stlgtr = Util::stConvert(ittr-
>GetLongitud());
                                        string stlgv =
Util::stConvert(vp.GetLongitud());
                                        string sttperr = " mtrs.        :sin
capacidad de via.";
                                        if(vp.GetLongitud() == 0){sttperr = "
:la via no existe en ese punto; ";};
                                        stInc = stInc + Util::stConvert(n) +
"\t" + ".- Tren " + ittr->GetIdTren() + " de " + stlgtr +  "mtrs.  en ";
                                        stInc = stInc  + itlp->GetIdPP() + "
- " + "\t" + nmPp + " en via: " + stnumv + " de " + "\t" + stlgv + sttperr
+ "\n";
                                n++;
                                }
                                itlp++;
                        };

                ittr++;
        };
        return stInc;
};
```

OcupTrayecto.cpp

```
//Miguel Arranz Pascual PFC Mallas
//fichero OcupTrayecto.cpp
//Funciones y operadores de la clase OcupTrayecto dentro del módulo
incompatibilidades


#include "StdAfx.h"
#include "OcupTrayecto.h"
#include "Util.h"
#include <cstdlib>
#include <iostream>
#include <string>
using namespace std;
using namespace System;

//Inicialización de lista
list<OcupTrayecto> NwLstOcTry;
list<OcupTrayecto> OcupTrayecto::ListaDeOcupTrayecto = NwLstOcTry;


//Constructores

// Constructor por defecto
OcupTrayecto::OcupTrayecto(void)
{
        idTry = "IN-FN";
        idTren = "00000";
        idPPInic = "INICIO";
        idPPFin = "FIN";
        trParidad = "00000";
        idBanda = "00000";
        viaTry = 0;
        paridad = 0;
        hEntrada = 0;
        hSalida = 0;
}

 // constructor completo.
OcupTrayecto::OcupTrayecto(string idtry, string idtr, string idppi, string
idppf, string trprd,
               string idbd, int vtr,  int p, int hent, int hsld)
{
        idTry = idtry;
        idTren = idtr;
        idPPInic = idppi;
        idPPFin = idppf;
        trParidad = trprd;
        idBanda = idbd;
        viaTry = vtr;
        paridad = p;
        hEntrada = hent;
        hSalida = hsld;

};
```

```
//Destructor
OcupTrayecto::~OcupTrayecto(void)
{

};

// Set

void OcupTrayecto::SetIdTry ( string idtry)
{
         idTry = idtry;

};

void OcupTrayecto::SetIdTren ( string idtr)
{
        idTren = idtr;

};

void OcupTrayecto::SetIdPpInic(string idppi)
{
        idPPInic = idppi;

};

void OcupTrayecto::SetIdPpFin(string idppf)
{
         idPPFin = idppf;

};

void OcupTrayecto::SetTrParidad (string trprd)
{
        trParidad = trprd;

};

void OcupTrayecto::SetIdBanda (string idbd)
{
         idBanda = idbd;

};

void OcupTrayecto::SetViaTry(int vtr)
{
        viaTry = vtr;

};

void OcupTrayecto::SetParidad (int p)
{
        paridad = p;

};

void OcupTrayecto::SetHEntrada (int hent)
{
        hEntrada = hent;
```

```
};

void OcupTrayecto::SetHSalida(int hsld)
{
        hSalida = hsld;
};

//Operadores miembro

// Presenta los datos del objeto OcupTrayecto
string OcupTrayecto::Print(void)
{
        string stOtry = "TREN: ";
        string stidTry = idTry;
        string stidProducto = Tren::BuscarTren(idTren).StProducto();

        string stnmPpini = PuntoDePaso::BuscarPP(idPPInic).GetPPNombre();
        string stnmPpfin= PuntoDePaso::BuscarPP(idPPFin).GetPPNombre();
        string sttrParidad = trParidad;
        string stidBd = idBanda;
        string stviaTry = Util::stConvert(viaTry);
        string stparidad = Util::stConvert(paridad);
        string sthEnt = Util::stHora(hEntrada);
        string sthSal = Util::stHora(hSalida);

        stOtry = stOtry + stidProducto + "   " +  sttrParidad + "   por via
" + stviaTry + "\n";
        stOtry = stOtry + "   Sale de: " + stnmPpini + "   a las  " +
sthEnt + "  llega a: " + stnmPpfin + "  a las " + sthSal + "\n" + "\n";
        return stOtry;
};


//Operadores de clase

//Agrega una nueva instancia a la lista de objetos
void OcupTrayecto::AgregarIns( const OcupTrayecto &octry)
{
        OcupTrayecto::ListaDeOcupTrayecto.push_back(octry);
};

// imprime la lista de objetos
string OcupTrayecto::PrintLista(void)
{
        string stLstOcTry = "";
        OcupTrayecto octryx;
        list<OcupTrayecto> listOcuTry = OcupTrayecto::ListaDeOcupTrayecto;
        list<OcupTrayecto>::iterator it = listOcuTry.begin();

        while ( it != listOcuTry.end())
        {
                octryx = *it;
                stLstOcTry = stLstOcTry + octryx.Print();
                it++;
        }
        return stLstOcTry;
};
```

```
// genera la lista de ocup try a partir de cada tren y su lista horario
void OcupTrayecto::GenerarOcupTrayectos(void)
{
	//Variables auxiliares
	string idOcTry = "";
	string idtry = "";
	string idtr = "";
	string idppi = "";
	string idppf = "";
	string trprd = "";
	string idbd = "";
	int vtr = 0;
	int p = 0;
	int hent = 0;
	int hsld = 0;

	//Borrar la lista anterior
	OcupTrayecto::ListaDeOcupTrayecto.clear();

	//Para cada tren de la lista de trenes
	list<Tren>::iterator ittr = Tren::ListaDeTrenes.begin();
	while (ittr != Tren::ListaDeTrenes.end())
	{
		//Obtenemosla lista de horarios
		list<Horario> lstHrs = ittr->GetListHorario();
		list<Horario>::iterator ithr = lstHrs.begin();
		//Obtenemos la hora de entrada y ppi del sig en el actual
		hent = ithr->GetHsld();
		idppi = ithr->GetIdPP();
		ithr++; // pasamos al siguiente horario
		//Para cada horario de la lista de horarios
		while (ithr != lstHrs.end())
		{
			hsld = ithr->GetHllg();
			idppf = ithr->GetIdPP();
			//Si no es un cambio de banda hsld-hllg
			if ( hent != hsld)
			{
				idOcTry = ithr->GetIdTry() + "-" + ittr-
>GetIdTren();
				//creamos un nuevo elemento y lo agregamos a
la lista
				OcupTrayecto idOcTry (ithr->GetIdTry(), ittr-
>GetIdTren(), idppi, idppf, ithr->GetTrParidad(),
					ithr->GetIdBanda(), ithr-
>GetViaTry(), ithr->GetParidad(), hent, hsld);
				OcupTrayecto::AgregarIns(idOcTry);
				//Obtenemos la hora de entrada y ppi del sig
en el actual

			};
			hent = ithr->GetHsld();
			idppi = ithr->GetIdPP();
			ithr++; // pasamos al siguiente horario
		};

		ittr++;
	};

};
```

```
// genera la lista de ocup try a partir de cada tren y su lista horario
filtrada por banda
void OcupTrayecto::GenerarOcupTrayectosFiltroBanda(string idbd)
{
        string idbdaux = "";
        OcupTrayecto ocTryx;
        // Obtener la nueva lista
        OcupTrayecto::GenerarOcupTrayectos();

        //Filtrar solo los elementos de la banda
        list<OcupTrayecto> lstOcTry = OcupTrayecto::ListaDeOcupTrayecto;
        list<OcupTrayecto>::iterator itot = lstOcTry.begin();

        //Borrar la lista anterior
        OcupTrayecto::ListaDeOcupTrayecto.clear();

        while (itot != lstOcTry.end())
        {

                idbdaux = itot->GetIdBanda();
                int cmpSt = idbdaux.compare(idbd);
                if(cmpSt == 0)
                {
                        ocTryx = *itot;
                        OcupTrayecto::AgregarIns(ocTryx);
                };
                itot++;
        };
};
```

IncTrayecto.h

```
//Miguel Arranz Pascual PFC Mallas
//fichero IncTrayecto.h
//Declaración de la clase IncTrayecto dentro del módulo incompatibilidades

#pragma once
#include "StdAfx.h"
#include "Banda.h"
#include "Tren.h"
#include "Util.h"
#include "OcupTrayecto.h"
#include <iostream>
#include <string>
#include <list>
using namespace std;
using namespace System;

class IncTrayecto
{

//miembros de clase
public:
// Guarda la lista de inc halladas
```

```
        static list<IncTrayecto>ListaDeIncTrayecto;

//miembros de datos
private:
        string idInc; // identifica la incidencia
        string idTry; // id del trayecto afectado
        string idBanda; // identificador de la banda de recorrido
        string idPPInic; // id del punto de paso inicio del trayecto
        string idPPFin; // id del punto de paso final del trayecto
        int tipoInc; // tipo de la inc. detectada
        int hInicial; // hora inicial del tramo horario
        int hFinal; // hora final del tramo horario
        list<OcupTrayecto>listOcuTry; //listado de coincidencias horarias
incompatibles en trayecto

//Constructores

public:
        IncTrayecto(void); // constructor por defecto
        IncTrayecto(string idinc, string idtry, string idbd, string idppi,
string idppf,
                int tpi,  int hini, int hfin); // constructor con datos.
        IncTrayecto (string idinc, string idtry, string idbd, string idppi,
string idppf,
                int tpi,  int hini, int hfin, list<OcupTrayecto> lstoctry);
// constructor completo con lista incluida

//Destructor
        virtual ~IncTrayecto(void);

//Set
        void SetIdInc ( string idinc);
        void SetIdTry ( string idtry);
        void SetIdBanda (string idbd);
        void SetIdPpInic(string idppi);
        void SetIdPpFin(string idppf);
        void SetTipoInc(int tpi);
        void SetHInicial (int hini);
        void SetHFinal(int hfin);
        void SetListOcuTry (list<OcupTrayecto> lstoctry);

//Get
        string GetIdInc (void) {return idInc;};
        string GetIdTry(void) {return idTry;};
    string GetIdBanda (void) {return idBanda;};
        string GetIdPpInic(void) {return idPPInic;};
        string GetIdPpFin (void) {return idPPFin;};
        int GetTipoInc(void) {return tipoInc;};
        int GetHInicial(void) {return hInicial;};
        int GetHFinal (void) {return hFinal;};
        list<OcupTrayecto> GetListOcuTry(void);

//Operadores miembro

        string PrintIncTry(void); // Presenta los datos minimos de la inc
        string PrintIncTryCompleta(void); //Presenta los datos completos
incluso los trenes implicados
        string PrintListaOcuTry(void); //presenta el listado de trenes
implicados
```

```
//Operadores de clase
        static void AgregarInc ( const IncTrayecto &inctry); // Agrega una
nueva instancia a la lista
        static string PrintLista(void); // imprime la lista de objetos
        static bool ComprobarSolapeHorario ( int he1, int hs1, int he2, int
hs2); // comprueba si dos tramos horarios se solapan
        static bool ComprobarInclusionHorario ( int he1, int hs1, int he2,
int hs2); //ordena los tramos horarios y comrprueva solape
        static IncTrayecto BuscarIncTry(string idInc); //Devuelve la inc
con el identificador aportado

        /* obtencion de las incidencias de trenes en trayecto, vamos a
distinguir tres tipos
        3: X - CRUCE ; 4: C - EXC CAPACIDAD ; 5: A - ADELANTAMIENTO  */

        static void ObtenerIncTrayectoTipoX (void); // genera la lista de
incidencias por cruce de trenes en un trayecto.
        static void ObtenerIncTrayectoTipoC (void); // genera la lista de
incidencias por exceso de trenes en un trayecto.
        static void ObtenerIncTrayectoTipoA (void); // genera la lista de
incidencias por adelantamiento de trenes en un trayecto.




};
```

IncTrayecto.cpp

```
//Miguel Arranz Pascual PFC Mallas
//fichero IncTrayecto.cpp
//Funciones y operadores de la clase IncTrayecto dentro del módulo
incompatibilidades

#include "StdAfx.h"
#include "IncTrayecto.h"
#include <iostream>
#include <string>
#include <list>
using namespace std;
using namespace System;

//Inicialización de lista
list<IncTrayecto> NwLstIncTry;
list<IncTrayecto> IncTrayecto::ListaDeIncTrayecto= NwLstIncTry;


//Constructores

// constructor por defecto
IncTrayecto::IncTrayecto(void)
{
        idInc = "IN-FN-00:00";
        idTry = "IN-FN";
```

```
        idBanda = "00000";
        idPPInic = "INICIO";
        idPPFin = "FIN";
        tipoInc = 0;
        hInicial = 0;
        hFinal = 0;
        list<OcupTrayecto> NwListIncTry;
        list<OcupTrayecto>listOcuTry =  NwListIncTry;

};

// constructor con datos.
IncTrayecto::IncTrayecto(string idinc, string idtry, string idbd, string
idppi, string idppf,
                int tpi,  int hini, int hfin)
{
        idInc = idinc;
        idTry = idtry;
        idBanda = idbd;
        idPPInic = idppi;
        idPPFin = idppf;
        tipoInc = tpi;
        hInicial = hini;
        hFinal = hfin;
        list<OcupTrayecto> NwListIncTry;
        list<OcupTrayecto>listOcuTry =  NwListIncTry;
};

// constructor completo con lista incluida
IncTrayecto::IncTrayecto (string idinc, string idtry, string idbd, string
idppi, string idppf,
                int tpi,  int hini, int hfin, list<OcupTrayecto> lstoctry)
{    idInc = idinc;
        idTry = idtry;
        idBanda = idbd;
        idPPInic = idppi;
        idPPFin = idppf;
        tipoInc = tpi;
        hInicial = hini;
        hFinal = hfin;
        listOcuTry =  lstoctry;

};

//Destructor
IncTrayecto::~IncTrayecto(void)
{
};

//Set

void IncTrayecto::SetIdInc (string idinc)
{
       idInc = idinc;

};

void IncTrayecto::SetIdTry (string idtry)
{
       idTry = idtry;
```

```
};

void IncTrayecto::SetIdBanda (string idbd)
{
         idBanda = idbd;

};

void IncTrayecto::SetIdPpInic(string idppi)
{
        idPPInic = idppi;

};

void IncTrayecto::SetIdPpFin(string idppf)
{
        idPPFin = idppf;

};

void IncTrayecto::SetTipoInc(int tpi)
{
        tipoInc = tpi;

};

void IncTrayecto::SetHInicial (int hini)
{
         hInicial = hini;

};

void IncTrayecto::SetHFinal(int hfin)
{
         hFinal = hfin;

};

void IncTrayecto::SetListOcuTry (list<OcupTrayecto> lstoctry)
{
        listOcuTry =  lstoctry;
};

//Retornar la lista de ocu de try implicadas

list<OcupTrayecto> IncTrayecto::GetListOcuTry(void)
{
        list<OcupTrayecto> NwListIncTry;
        NwListIncTry = this->listOcuTry;
        return NwListIncTry;

};


//Operadores miembro

 // Presenta los datos minimos de la inc
string IncTrayecto::PrintIncTry(void)
{
```

```
        string stinctry = "Inc. detectada Tipo: ";
        string stidTry = idTry;
        string stidBd = idBanda;
        string stnmPpini = PuntoDePaso::BuscarPP(idPPInic).GetPPNombre();
        string stnmPpfin= PuntoDePaso::BuscarPP(idPPFin).GetPPNombre();
        string sttipoInc = Util::stConvert(tipoInc);
        string sthIni = Util::stHora(hInicial);
        string sthFin = Util::stHora(hFinal);

        if (tipoInc == 3){stinctry = stinctry + "3 - X Cruce" + "\n";};
        if (tipoInc == 4){stinctry = stinctry + "4 - C Capacidad" + "\n";};
        if (tipoInc == 5){stinctry = stinctry + "5 - A Adelantamiento" +
"\n";};

        stinctry = stinctry + "\n" + stidTry + "  " + stidBd  +  "  " +
stnmPpini + "  " + stnmPpfin;
        stinctry = stinctry + "       H: "  +  sthIni + "  -  " + sthFin +
"\n";

        return stinctry;
};

 //Presenta los datos completos incluso los trenes implicados
string IncTrayecto::PrintIncTryCompleta(void)
{
        string stinctry = "";
        stinctry = stinctry + this->PrintIncTry();
        stinctry = stinctry +
"=====================================================================" +
"\n";
        stinctry = stinctry + this->PrintListaOcuTry();

        return stinctry;
};

//presenta el listado de trenes implicados
string IncTrayecto::PrintListaOcuTry(void)
{
        string stlstOTry = "";
        list<OcupTrayecto>::iterator it = this->listOcuTry.begin();
        while ( it != this->listOcuTry.end())
        {
                stlstOTry = stlstOTry + it->Print();
                it++;
        }
        return stlstOTry;
};

//Operadores de clase

 // Agrega una nueva instancia a la lista
void IncTrayecto::AgregarInc ( const IncTrayecto &inctry)
{
        IncTrayecto::ListaDeIncTrayecto.push_back(inctry);
};

// imprime la lista de objetos
string IncTrayecto::PrintLista(void)
{
        string stlstInTry = "";
```

```
        IncTrayecto inctryx;
        list<IncTrayecto>::iterator it =
IncTrayecto::ListaDeIncTrayecto.begin();
        while ( it != IncTrayecto::ListaDeIncTrayecto.end())
        {
                inctryx = *it;
                stlstInTry = stlstInTry + inctryx.PrintIncTry();
                it++;
        }
        return stlstInTry;
};

 // comprueba si dos tramos horarios se solapan
bool IncTrayecto::ComprobarSolapeHorario ( int he1, int hs1, int he2, int
hs2)
{
        /* Entendemos que hay un solape horario cuando siendo he2>he1 es
ademas menor que hs1
          desde nuestro punto de vista de trenes en trayecto un tren accede
a un trayecto antes de que
          otro haya salido de el */

        bool slphr = false;
        if ((he1<=he2) & (he2<hs1))
        {
                slphr = true;
        };
        if ((he1>he2)& (he1<hs2))
        {
                slphr = true;
        };

        return slphr;
};

//ordena los tramos horarios y comrprueva solape
bool  IncTrayecto::ComprobarInclusionHorario ( int he1, int hs1, int he2,
int hs2)
{
        /* Entendemos que hay un  horario incluido en otro cuando siendo
he2>he1 es ademas hs2 es menor que hs1
          desde nuestro punto de vista de trenes en trayecto un tren accede
a un trayecto despues de
          otro  y sale antes de que el otro " adelantamiento imposible"  */

        bool inchr = false;
        if ((he1<he2) & (hs2<=hs1))
        {
                inchr = true;
        };
        if ((he1>he2) & (hs1<=hs2))
        {
                inchr = true;
        };
        return inchr;

};

//Devuelve la inc con el identificador aportado
IncTrayecto IncTrayecto::BuscarIncTry(string idInc)
```

```
{
        IncTrayecto incTry("IN-FN-00:00","IN-FN","00000", "INICIO", "FIN",
0, 0, 0);
        string idx = "";
        list<IncTrayecto>::iterator itinc =
IncTrayecto::ListaDeIncTrayecto.begin();
        while ( itinc != IncTrayecto::ListaDeIncTrayecto.end())
        {

                idx = itinc->GetIdInc();
                int cmpSt = idx.compare(idInc);
                if (cmpSt == 0){return *itinc;};
                itinc++;
        }
        return incTry;

};

// genera la lista de incidencias por cruce de trenes en un trayecto.
void IncTrayecto::ObtenerIncTrayectoTipoX (void)
{
  /* Para una incidencia de este tipo deben coincidir dos trenes en el
mismo trayecto, por la misma via,
  de sentido (paridad) contraria y coincidir en horario.
  Para obtener las incidencias de este tipo, recorremos la lista de
OcupTrayecto y para cada ocupación
  volvemos a comprobar en la lista alguna de las siguientes cumple estas
condiciones y generamos la incidencia correspondiente.
  */

        // variables auxiliares
        bool solapephorario = false;
        bool coincideTry = false;
        bool coincideVia = false;
        bool diferenteParidad = false;
        string idIncTry = "";
        string idtry = "";
        string idbd = "";
        string idppi = "";
        string idppf = "";
        int tpinc = 0;
        int hini = 0;
        int hfin = 0;
        int he1, he2, hs1, hs2;
        OcupTrayecto ocutry;

        int n1 = 0;
        int n2 = 0;

        //Borrar lista anterior
        IncTrayecto::ListaDeIncTrayecto.clear();

        //Para cada ocupación de trayecto de la lista
        list<OcupTrayecto>::iterator itot1 =
OcupTrayecto::ListaDeOcupTrayecto.begin();
        while ( itot1 != OcupTrayecto::ListaDeOcupTrayecto.end())
        {
                he1 = itot1->GetHEntrada();
                hs1 = itot1->GetHSalida();
                n1++;
```

```
                list<OcupTrayecto>::iterator itot2 = itot1;
                itot2++;
                n2 = n1;
                while ( (OcupTrayecto::ListaDeOcupTrayecto.size()-n2)>1)
                {

                        he2 = itot2->GetHEntrada();
                        hs2 = itot2->GetHSalida();
                        n2++;
                        //comprobamos coincidencias si no salimos para no
perder tiempo
                        coincideTry = (itot1->GetIdTry().compare(itot2-
>GetIdTry()) == 0);
                        coincideVia = (itot1->GetViaTry() == itot2-
>GetViaTry());
                        diferenteParidad = ( itot1->GetParidad() != itot2-
>GetParidad());
                        solapephorario =
IncTrayecto::ComprobarSolapeHorario(he1, hs1, he2, hs2);

                        // se cumplen todas las condiciones creamos una
IncTrayecto y la agregamos a la lista
                        if (coincideTry & coincideVia & diferenteParidad &
solapephorario)
                        {
                                hini = he1;
                                if(he2<he1){hini = he2;};
                                hfin = hs1;
                                if(hs1<hs2){hfin =hs2;};
                                idIncTry = itot1->GetIdTry() + "-" +
Util::stHora(hini);
                                IncTrayecto idIncTry (idIncTry, itot1-
>GetIdTry(), itot1->GetIdBanda(), itot1->GetIdPpInic(), itot1-
>GetIdPpFin(), 3, hini, hfin);
                                //Agregamos las ocup de trayecto afectadas
                                idIncTry.listOcuTry.push_back(*itot1);
                                idIncTry.listOcuTry.push_back(*itot2);
                                IncTrayecto::AgregarInc (idIncTry);


                        };
                        itot2++;
                }
        itot1++;
        }

};

// genera la lista de incidencias por exceso de trenes en un trayecto.
void IncTrayecto::ObtenerIncTrayectoTipoC (void)
{
        /* Para una incidencia de este tipo deben coincidir mas  trenes en
el mismo trayecto, por la misma via,
  del mismo sentido (paridad) y superar su nº a la capacidad del trayecto.
  Para obtener las incidencias de este tipo, recorremos la lista de
OcupTrayecto y para cada una creamos una incidencia
  y añadimos su propia ocupación a la lista, recorremos la lista a partir
de ella añadiendo las que cumplen las condiciones
  actualizando el tramo horario y al final si el nº de componentes de la
lista supera la capacidad del trayecto la agregamos
```

```
  como incidencia.

  */

        // variables auxiliares
        bool solapephorario = false;
        bool coincideTry = false;
        bool coincideVia = false;
        bool igualParidad = false;
        string idIncTry = "";
        string idtry = "";
        string idbd = "";
        string idppi = "";
        string idppf = "";
        int tpinc = 0;
        int hini = 0;
        int hfin = 0;
        int he1, he2, hs1, hs2;
        int captry = 0;
        OcupTrayecto ocutry;

        int n1 = 0;
        int n2 = 0;

        //Borrar lista anterior
        IncTrayecto::ListaDeIncTrayecto.clear();


        //Para cada ocupación de trayecto de la lista
        list<OcupTrayecto>::iterator itot1 =
OcupTrayecto::ListaDeOcupTrayecto.begin();
        while ( itot1 != OcupTrayecto::ListaDeOcupTrayecto.end())
        {
                he1 = itot1->GetHEntrada();
                hs1 = itot1->GetHSalida();
                n1++;

                // Conocemos la capacidad del trayecto
                Trayecto tryc = Trayecto::BuscarTry(itot1->GetIdTry());
                captry = tryc.GetCapacidad();

                //Nueva incidencia
                idIncTry = itot1->GetIdTry() + "-" + Util::stHora(he1);
                IncTrayecto idIncTry (idIncTry, itot1->GetIdTry(), itot1-
>GetIdBanda(),
                                        itot1->GetIdPpInic(), itot1-
>GetIdPpFin(), 4, he1, hs1);
                idIncTry.listOcuTry.push_back(*itot1);

                //Recorremos la lista a partir de aqui
                list<OcupTrayecto>::iterator itot2 = itot1;
                itot2++;
                n2 = n1;
                while ( (OcupTrayecto::ListaDeOcupTrayecto.size()-n2)>1)
                {

                        he2 = itot2->GetHEntrada();
                        hs2 = itot2->GetHSalida();
                        n2++;
                        //comprobamos coincidencias
```

```
                        coincideTry = (itot1->GetIdTry().compare(itot2-
>GetIdTry()) == 0);
                        coincideVia = (itot1->GetViaTry() == itot2-
>GetViaTry());
                        igualParidad = ( itot1->GetParidad() == itot2-
>GetParidad());
                        solapephorario =
IncTrayecto::ComprobarSolapeHorario(he1, hs1, he2, hs2);

                        // se cumplen todas las condiciones  la agregamos a
la lista
                        if (coincideTry & coincideVia & igualParidad &
solapephorario)
                        {
                                if(hs1<hs2){idIncTry.SetHFinal(hs2);};
                                //if(he2<he1){idIncTry.SetHInicial(he2);};
                                //Agregamos las ocup de trayecto afectadas
                                idIncTry.listOcuTry.push_back(*itot2);

                        };
                        itot2++;
                }
        //Si el nº de trenes implicados es mayor que la capacidad la
agregamos a la lista
        if((idIncTry.listOcuTry.size()>captry)){IncTrayecto::AgregarInc
(idIncTry);};
        itot1++;
        }


};

// genera la lista de incidencias por adelantamiento de trenes en un
trayecto.

void IncTrayecto::ObtenerIncTrayectoTipoA (void)
{
        /* Para una incidencia de este tipo deben coincidir dos trenes en
el mismo trayecto, por la misma via,
  del mismo sentido (paridad) y el intervalo horario de uno de ellos debe
estar incluido en el otro.
  Para obtener las incidencias de este tipo, recorremos la lista de
OcupTrayecto y para cada ocupación
  volvemos a comprobar en la lista alguna de las siguientes cumple estas
condiciones y generamos la incidencia correspondiente.
  */

        // variables auxiliares
        bool inclushorario = false;
        bool coincideTry = false;
        bool coincideVia = false;
        bool igualParidad = false;
        string idIncTry = "";
        string idtry = "";
        string idbd = "";
        string idppi = "";
        string idppf = "";
        int tpinc = 0;
        int hini = 0;
        int hfin = 0;
```

```
        int he1, he2, hs1, hs2;
        OcupTrayecto ocutry;

        int n1 = 0;
        int n2 = 0;

        //Borrar lista anterior
        IncTrayecto::ListaDeIncTrayecto.clear();

        //Para cada ocupación de trayecto de la lista
        list<OcupTrayecto>::iterator itot1 =
OcupTrayecto::ListaDeOcupTrayecto.begin();
        while ( itot1 != OcupTrayecto::ListaDeOcupTrayecto.end())
        {
                he1 = itot1->GetHEntrada();
                hs1 = itot1->GetHSalida();
                n1++;
                list<OcupTrayecto>::iterator itot2 = itot1;
                itot2++;
                n2 = n1;
                while ( (OcupTrayecto::ListaDeOcupTrayecto.size()-n2)>1)
                {

                        he2 = itot2->GetHEntrada();
                        hs2 = itot2->GetHSalida();
                        n2++;
                        //comprobamos coincidencias si no salimos para no
perder tiempo
                        coincideTry = (itot1->GetIdTry().compare(itot2-
>GetIdTry()) == 0);
                        coincideVia = (itot1->GetViaTry() == itot2-
>GetViaTry());
                        igualParidad = ( itot1->GetParidad() == itot2-
>GetParidad());
                        inclushorario =
IncTrayecto::ComprobarInclusionHorario(he1, hs1, he2, hs2);

                        // se cumplen todas las condiciones creamos una
IncTrayecto y la agregamos a la lista
                        if (coincideTry & coincideVia & igualParidad &
inclushorario)
                        {
                                hini = he1;
                                if(he2<he1){hini = he2;};
                                hfin = hs1;
                                if(hs1<hs2){hfin =hs2;};
                                idIncTry = itot1->GetIdTry() + "-" +
Util::stHora(hini);
                                IncTrayecto idIncTry (idIncTry, itot1-
>GetIdTry(), itot1->GetIdBanda(), itot1->GetIdPpInic(), itot1-
>GetIdPpFin(), 5, hini, hfin);
                                //Agregamos las ocup de trayecto afectadas
                                idIncTry.listOcuTry.push_back(*itot1);
                                idIncTry.listOcuTry.push_back(*itot2);
                                IncTrayecto::AgregarInc (idIncTry);


                        };
                        itot2++;
                }
```

```
        itot1++;
        }

};
```

A.3.4 Código Funcional de Formularios

frmMallas.h

```
//Función para asegurar la ransformación de String^ o String e string

void MarshalString ( String ^ s, string& os ) {
                using namespace Runtime::InteropServices;
                const char* chars =
                (const char*)(Marshal::StringToHGlobalAnsi(s)).ToPointer();
                        os = chars;
                Marshal::FreeHGlobal(IntPtr((void*)chars));
        }

//Función para calcular la posición vertical en función de tamaño de los
objetos

int CalculoPosicionVertical( int ub, int ubMx) {

        int margenSp =50;
        int margenInf =50;
        int altoControl = 100;
        double factorEscala = 1.0;
        int pos = 0;
        altoControl = this->pbxPps->Height::get();
        factorEscala = (double)(altoControl-margenSp-margenInf)/ubMx;
        pos = (int) (ub * factorEscala)+ margenSp;
        return pos;
}

int CalculoPosicionHorizontal( int hr, int lgMx) {

        int margenIzq =40;
        int margenDrh =50;
        int anchoControl = 100;
        double factorEscala = 1.0;
        int pos = 0;
        anchoControl = this->pbxHoras->Width::get();
        factorEscala = (double)(anchoControl-margenIzq-margenDrh)/lgMx;
        pos = (int) (hr * factorEscala)+ margenIzq;

        return pos;

}

System::Drawing::Color ObtenerColorTren( int prdto) {

        System::Drawing::Color colorTren;
        switch (prdto){
                case 0 :
                        colorTren = System::Drawing::Color::Green;
                        break;
                case 1 :
                        colorTren = System::Drawing::Color::Maroon;
                        break;
                case 2 :
                        colorTren = System::Drawing::Color::Red;
```

```
                        break;
                case 3 :
                        colorTren = System::Drawing::Color::Blue;
                        break;
                case 4 :
                        colorTren = System::Drawing::Color::Goldenrod;
                        break;
                case 5 :
                        colorTren = System::Drawing::Color::Indigo;
                        break;
                case 6 :
                        colorTren = System::Drawing::Color::Gainsboro;
                        break;
                case 7 :
                        colorTren = System::Drawing::Color::Gray;
                        break;
                case 8 :
                        colorTren = System::Drawing::Color::LightSteelBlue;
                        break;
                default :
                        colorTren = System::Drawing::Color::LightSteelBlue;
        }
        return colorTren;

}

System::Drawing::Point ObtenerPuntoTextoTren(int ub, int ubMx, int hr, int
lgMx, int p) {

        System::Drawing::Point ptoTexto;
        int posX = 0;
        int posY = 0;
        posY = CalculoPosicionVertical(ub, ubMx);
        posX = CalculoPosicionHorizontal(hr, lgMx);
        if (p==1){posY = posY - 12;};
        if (p==2){posY = posY + 2;};
        posX = posX -20;
        ptoTexto = Point (posX,posY);

        return ptoTexto;


}

System::Drawing::Point ObtenerPuntoGraficoTren(int ub, int ubMx, int hr,
int lgMx) {

        System::Drawing::Point ptoGrafico;
        int posX = 0;
        int posY = 0;
        posY = CalculoPosicionVertical(ub, ubMx);
        posX = CalculoPosicionHorizontal(hr, lgMx);
        ptoGrafico = Point (posX,posY);

        return ptoGrafico;
}

private: System::Void frmMallas_SizeChanged(System::Object^  sender,
System::EventArgs^  e) {
```

```
                        //reposicionar objetos en relación a sus
contenedores
        this->pbxPps->Size = System::Drawing::Size(105,
(frmMallas::Height)-140);
        this->pnlGrafico->Size = System::Drawing::Size((frmMallas::Width)-
180, (frmMallas::Height)-80);
        this->pnlGrafico->HorizontalScroll->Value = 0;
        this->pbxAbv->Location = System::Drawing::Point((frmMallas::Width-
45), 45);
        this->pbxAbv->Size = System::Drawing::Size (28,
        (frmMallas::Height)-140);
        this->pbxHoras->Location = System::Drawing::Point(3, (this-
>pnlGrafico->Height)-60);
        this->pbxHoras->Size = System::Drawing::Size(12500, 33);
        this->pbxMalla->Size = System::Drawing::Size(12500, (this-
>pnlGrafico->Height)-70);
        this->cboHoras->Location = System::Drawing::Point (50,
(frmMallas::Height)-68);
        this->lblHora->Location = System::Drawing::Point (7,
(frmMallas::Height)-51);
        //refrescar picture´s
        this->pbxPps->Refresh();
        this->pbxMalla->Refresh();
        this->pbxAbv->Refresh();
        this->pbxHoras->Refresh();
}


private: System::Void frmMallas_Load(System::Object^  sender,
System::EventArgs^  e) {

         // Maximizar el form por defecto
                        frmMallas::WindowState =
System::Windows::Forms::FormWindowState::Maximized;


         // Cargar la lista de bandas
        string str = "";
        String^ cstr = "www ";
        cboBandas->BeginUpdate();
        int i = 0;
        list<Banda>::iterator it = Banda::ListaDeBandas.begin();
        while (it != Banda::ListaDeBandas.end())
        {
                str = it->GetIdBanda();
                cstr = gcnew String(str.c_str());
                cboBandas->Items->Add( String::Format( " {0}", cstr ) );
                it++;
                i++;
        }
        cboBandas->EndUpdate();

                                //refrescar picture´s
        this->pbxPps->Refresh();
        this->pbxMalla->Refresh();
        this->pbxAbv->Refresh();
        this->pbxHoras->Refresh();


}
```

```
private: System::Void pbxPps_Paint(System::Object^  sender,
System::Windows::Forms::PaintEventArgs^  e) {

        //*************************************************************
        // Dibujar la lista del Pps ubicados en la banda correspondiente
        //*************************************************************

        // Declaración de varibles auxiliares
        int x = -40;  // abcisa
        int y = 3;  // ordenada
        int maxUbicacion = 1;
        string idBd = "";
        Graphics ^ g = e->Graphics;
        System::Drawing::Font ^ ft = gcnew System::Drawing::Font(
"Arial",8,FontStyle::Bold );

        MarshalString(cboBandas->Text,idBd);
        idBd = idBd.replace(0,1,"");
        Banda bd = Banda::BuscarBanda(idBd);
        PuntoPasoBanda ppb;
        PuntoDePaso pp;
        string nmPp = "";
        String^ cstr = "";
        list<PuntoPasoBanda> lstPp = bd.GetListPP();

        maxUbicacion = bd.GetUbMax();

        // Dibujamos la lista de nombres de los Pps de la banda en su
ubicación
        list<PuntoPasoBanda>::iterator it = lstPp.begin();
        while (it != lstPp.end())
        {
        ppb = *it;
        pp = ppb.GetPtoPaso();
        nmPp = pp.GetPPNombre();
        cstr = gcnew String(nmPp.c_str());
        y = CalculoPosicionVertical(it->GetUbicBanda(), maxUbicacion);
        g->DrawString(cstr,ft, System::Drawing::Brushes::Black, x,y);
        it++;

        }
                }


private: System::Void pbxMalla_Paint(System::Object^  sender,
System::Windows::Forms::PaintEventArgs^  e) {

        // Crea el elemento grafico de dibujo
    Graphics^ g = e->Graphics;
        Graphics^ gtxt = e->Graphics;
        System::Drawing::Color colorTren;
        System::Drawing::Point pto1;
        System::Drawing::Point pto2;
        System::Drawing::Point pto3;
        System::Drawing::Point ptoTxTr;
        System::Drawing::Font ^ ft = gcnew System::Drawing::Font(
"Arial",8,FontStyle::Bold );
```

```
        System::Drawing::Pen ^ pen = gcnew System::Drawing::Pen
(System::Drawing::Color::DimGray);
        pen->DashStyle = System::Drawing::Drawing2D::DashStyle::Dot;
        System::Drawing::SolidBrush^ brush = gcnew
System::Drawing::SolidBrush(System::Drawing::Color::Black);
        System::Drawing::StringFormat^ stformat = gcnew
System::Drawing::StringFormat();

        // Declaración de varibles auxiliares
        int x = 0;  // abcisa
        int y = 3;  // ordenada
        int maxUbicacion = 1;
        Banda bd;
        PuntoPasoBanda ppb;
        PuntoDePaso pp;
        Tren tr;
        Horario hr;
        string idBd = "";
        string idTrp = "";
        int prdto;
        int ubx;
        int uby;
        int paridad;

        // Dibuja el rectangulo que delimita el grafico

        //System::Drawing::Pen ^ grospen = gcnew System::Drawing::Pen
(System::Drawing::Color::Black,3.0f);
        //g->DrawRectangle(grospen,40,40,pbxMalla->Width::get()-
50,pbxMalla->Height::get()-50);

        //Dibuja las líneas horizontales equivalentes a los puntos de paso
        //*****************************************************************
**
        MarshalString(cboBandas->Text,idBd);
        idBd = idBd.replace(0,1,"");
        bd = Banda::BuscarBanda(idBd);
        string nmPp = "";
        String^ cstr = "";
        list<PuntoPasoBanda> lstPp = bd.GetListPP();

        maxUbicacion = bd.GetUbMax();

        list<PuntoPasoBanda>::iterator it = lstPp.begin();
        while (it != lstPp.end())
        {
        ppb = *it;
        pp = ppb.GetPtoPaso();
        nmPp = pp.GetPPNombre();
        cstr = gcnew String(nmPp.c_str());
        y = CalculoPosicionVertical(it->GetUbicBanda(), maxUbicacion);
        g->DrawLine( pen, pbxMalla->Left+40, y,
                pbxMalla->Width::get()-20, y );
        it++;
        }

        //Dibujamos una linea vertical cada 5 minutos
        pen->DashStyle = System::Drawing::Drawing2D::DashStyle::Solid;
        for ( int x=0;x<=93600;x=x+300)
                {
```

```
                        int hx = CalculoPosicionHorizontal(x,93600);
                        g->DrawLine( pen, hx, 50, hx, pbxMalla-
>Height::get()-40);
                }


        //*********************************************************
        //Dibujamos la gráfica de trenes para esa banda
        //*********************************************************

        // Obtenemos la lista de trenes
        list<Tren>::iterator ittr = Tren::ListaDeTrenes.begin();
        //Para cada tren hasta el final de lista

        while (ittr != Tren::ListaDeTrenes.end())
        {
                tr = *ittr;
                // Obtenemos el color del tren según producto y establecemos
el lapiz y brocha
                colorTren = ObtenerColorTren(tr.GetProducto());
                pen->Color::set(colorTren);
                brush->Color::set(colorTren);
                pen->DashStyle =
System::Drawing::Drawing2D::DashStyle::Solid;
                pen->Width = 1.5 ;

                // inicializamos el testigo punto inicial
                bool ptoInicial = true;

                //Para cada tren obtenemos y recorremos su lista de horarios
                list<Horario> lstHr = tr.GetListHorario();
                list<Horario>::iterator ithr = lstHr.begin();

                //Para cada horario hasta el final de la lista
                while (ithr != lstHr.end())
                {
                        hr = *ithr;
                        // Si la banda del tren es la actual
                        int cmpbx = hr.GetIdBanda().compare(idBd);
                        if (cmpbx == 0){
                                //Si es el primer punto
                                if(ptoInicial) {
                                        //Dibujamos el texto del nº del tren
                                        idTrp = hr.GetTrParidad();
                                        paridad = hr.GetParidad();
                                        cstr = gcnew String(idTrp.c_str());
                                        ubx = hr.GetHsld();
                                        uby = hr.GetUbicPp();
                                        ptoTxTr = ObtenerPuntoTextoTren (uby,
maxUbicacion,ubx,93600, paridad);
                                        pto1 = ObtenerPuntoGraficoTren (uby,
maxUbicacion,ubx,93600);
                                        gtxt->DrawString(cstr, ft,
brush,ptoTxTr, stformat);

                                        ptoInicial = false;
                                }
                                else {
                                        //Dibujamos los dos tramos de recta
de llegada y salida
```

```
                                        ubx = hr.GetHllg();
                                        uby = hr.GetUbicPp();
                                        pto2 = ObtenerPuntoGraficoTren (uby,
maxUbicacion,ubx,93600);
                                        ubx = hr.GetHsld();
                                        pto3 = ObtenerPuntoGraficoTren (uby,
maxUbicacion,ubx,93600);
                                        g->DrawLine( pen,pto1,pto2);
                                        g->DrawLine( pen,pto2,pto3);
                                        pto1  = pto3;
                                }
                        }
                        ithr++;
                }
                ittr++;
        }

 }


private: System::Void pbxAbv_Paint(System::Object^  sender,
System::Windows::Forms::PaintEventArgs^  e) {

        // Repintar la lista de abreviaturas de Pps ubicados en la banda
correspondiente
        //*********************************************************

        // Declaración de varibles auxiliares
        int x = 0;  // abcisa
        int y = 3;  // ordenada
        int maxUbicacion = 1;
        string idBd = "";
        Graphics ^ g = e->Graphics;
        System::Drawing::Font ^ ft = gcnew System::Drawing::Font(
"Arial",8,FontStyle::Bold );

        MarshalString(cboBandas->Text,idBd);
        idBd = idBd.replace(0,1,"");
        Banda bd = Banda::BuscarBanda(idBd);
        PuntoPasoBanda ppb;
        PuntoDePaso pp;
        string abvPp = "";
        String^ cstr = "";
        list<PuntoPasoBanda> lstPp = bd.GetListPP();

        maxUbicacion = bd.GetUbMax();

        // Dibujamos la lista de nombres de los Pps de la banda en su
ubicación
        list<PuntoPasoBanda>::iterator it = lstPp.begin();
        while (it != lstPp.end())
        {
        ppb = *it;
        pp = ppb.GetPtoPaso();
        abvPp = pp.GetPPAbr();
        cstr = gcnew String(abvPp.c_str());
        y = CalculoPosicionVertical(it->GetUbicBanda(), maxUbicacion);
        g->DrawString(cstr,ft, System::Drawing::Brushes::Black, x,y);
        it++;
```

```
        }

                }

private: System::Void pbxHoras_Paint(System::Object^  sender,
System::Windows::Forms::PaintEventArgs^  e) {

    //Dibujar las referencias horarias
        //*****************************************************

        // Crea el elemento grafico de dibujo
    Graphics^ g = e->Graphics;
        System::Drawing::Pen ^ pen = gcnew System::Drawing::Pen
(System::Drawing::Color::DimGray);
        pen->DashStyle = System::Drawing::Drawing2D::DashStyle::Dot;

        // Declaración de varibles auxiliares
        String^ cstr = "";
        string min;
        int hx =0;  // abcisa
        int y = 2;  // ordenada
        int lgmx = 93600;  // 26 horas


        //Dibujamos una pequeña linea cada minuto
        for ( int x=0;x<=93600;x=x+60)
                {
                        hx = CalculoPosicionHorizontal(x,lgmx);
                        g->DrawLine( pen, hx, y, hx, y+5);
                }

        //Dibujamos el texto de los minutos
        for ( int x=0;x<=93600;x=x+300)
                {
                        hx = CalculoPosicionHorizontal(x,lgmx);
                        min = Util::stConvert((x%3600)/60);
                        cstr = gcnew String(min.c_str());
                        System::Drawing::Font ^ ft = gcnew
System::Drawing::Font( "Arial",7,FontStyle::Bold );
                        g->DrawString(cstr,ft,
System::Drawing::Brushes::Coral, hx-5, y+6);
                }
        //Dibujamos el texto de las horas
        for ( int x=0;x<=93600;x=x+3600)
                {
                        hx = CalculoPosicionHorizontal(x,lgmx);
                        min = Util::stConvert(x/3600);
                        cstr = gcnew String(min.c_str());
                        System::Drawing::Font ^ ft = gcnew
System::Drawing::Font( "Arial",10,FontStyle::Bold );
                        g->DrawString(cstr,ft,
System::Drawing::Brushes::Blue, hx-5, y+14);
                }

 }

private: System::Void cboBandas_SelectedIndexChanged(System::Object^
sender, System::EventArgs^  e) {

         //Al seleccionar bande se deben repintar los objetos Pps y Malla
```

```
        this->pbxPps->Refresh();
        this->pbxMalla->Refresh();
        this->pbxAbv->Refresh();
        this->pbxHoras->Refresh();

            }

private: System::Void btnIncVia_Click(System::Object^  sender,
System::EventArgs^  e) {

                      frmIncomVia ^ verIncVia = gcnew frmIncomVia();
                      verIncVia->ShowDialog();

               }

private: System::Void btnTrenes_Click(System::Object^  sender,
System::EventArgs^  e) {

                       frmTrenes ^ verTrenes = gcnew frmTrenes();
                       verTrenes->ShowDialog();
               }
private: System::Void btnInfr_Click(System::Object^  sender,
System::EventArgs^  e) {

                       frmInfr ^ verInfr = gcnew frmInfr();
                       verInfr->ShowDialog();
               }
private: System::Void btnIncTrayecto_Click(System::Object^  sender,
System::EventArgs^  e) {

                      frmIncTrayecto ^ verIncTrayecto = gcnew
frmIncTrayecto();
                      verIncTrayecto->Show();
               }
private: System::Void frmMallas_FormClosed(System::Object^  sender,
System::Windows::Forms::FormClosedEventArgs^  e) {

                       // Crear la aplicacion
                       Application::Exit();

               }
private: System::Void toolTipMenu_Popup(System::Object^  sender,
System::Windows::Forms::PopupEventArgs^  e) {
               }
private: System::Void inicioToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {


               }
private: System::Void
infraestructuraToolStripMenuItem1_Click(System::Object^  sender,
System::EventArgs^  e) {

                      frmInfr ^ verInfr = gcnew frmInfr();
                       verInfr->ShowDialog();
               }
```

```
private: System::Void trenesToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {

                        frmTrenes ^ verTrenes = gcnew frmTrenes();
                        verTrenes->ShowDialog();
                }
private: System::Void
incidenciasDeViaToolStripMenuItem_Click(System::Object^  sender,
System::EventArgs^  e) {
                        frmIncomVia ^ verIncVia = gcnew frmIncomVia();
                       verIncVia->ShowDialog();

                }
private: System::Void incompTrenesToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {

                       frmIncTrayecto ^ verIncTrayecto = gcnew
frmIncTrayecto();
                       verIncTrayecto->Show();
                }
private: System::Void salirToolStripMenuItem_Click(System::Object^  sender,
System::EventArgs^  e) {
                        // Crear la aplicacion
                        Application::Exit();
                }


private: System::Void cboHoras_SelectedValueChanged(System::Object^
sender, System::EventArgs^  e) {

                                int vhora = Convert::ToInt32(cboHoras-
>Text)* 478;
                                this->pnlGrafico->HorizontalScroll->Value =
vhora;

                 }



};
}
```

frmInfr.h

```
        private: System::Void frmInfr_Load(System::Object^  sender,
System::EventArgs^  e) {
                            // Cargar la lista de bandas
                                 string str = "";
                                 String^ cstr = "www ";
                                 ListaBandas->BeginUpdate();
                                 int i = 0;
                                 list<Banda>::iterator it =
Banda::ListaDeBandas.begin();
                                 while (it != Banda::ListaDeBandas.end())
                                 {
                                 str = it->GetIdBanda();
                                 cstr = gcnew String(str.c_str());
                                 ListaBandas->Items->Add( String::Format( "
{0}", cstr ) );
                                 it++;
                                 i++;
                                 }
                                 ListaBandas->EndUpdate();
                }
        private: System::Void
ListaBandas_SelectedIndexChanged(System::Object^  sender,
System::EventArgs^  e) {
                            Banda bd;
                            PuntoPasoBanda ppb;
                            PuntoDePaso pp;
                            string stidbd= "";
                            string str0 ="";
                            string str1 ="";
                            String^ cstr = "www";

                            //Borrar textos
                            this->TextTrayecto->Text = L"";
                            this->TextPP->Text = L"";

                            TextBanda->Text = ListaBandas->Text;
                            MarshalString(TextBanda->Text,stidbd);
                            stidbd = stidbd.replace(0,1,"");
                            bd = Banda::BuscarBanda(stidbd);
                            str0 = bd.PrintBanda();
                            TextBanda->Text = gcnew String(str0.c_str());
        // Presentar listado de puntos de paso
                            ListaPPs->Items->Clear();
                            //ListaPPs->MultiColumn = true;
                  ListaPPs->SelectionMode = SelectionMode::MultiExtended;
                            ListaPPs->BeginUpdate();
                            list<PuntoPasoBanda> listpp = bd.GetListPP();
                            int i = 0;
                            list<PuntoPasoBanda>::iterator itp =
listpp.begin();
                            while (itp != listpp.end())
                                 {
                                        pp = itp->GetPtoPaso();
                                        str0 = pp.GetIdPP();
                                        str1 = str0 + "  " + pp.GetPPAbr() +
" " + pp.GetPPNombre();
```

```
                                        str1 = str1 + "\t" + itp->GetidTry()
+ "\t" + Util::stConvert(itp->GetUbicBanda());
                                        cstr = gcnew String(str1.c_str());
                                        ListaPPs->Items->Add( String::Format(
" {0}", cstr ) );
                                        i++;
                                        itp++;
                                }
                        ListaPPs->EndUpdate();
        }


        private: System::Void TextBanda_TextChanged(System::Object^
sender, System::EventArgs^  e) {


                        }

        private: System::Void ListaPPs_SelectedIndexChanged(System::Object^
sender, System::EventArgs^  e) {
                        // Presentar datos del pp seleccionado
                        string stpp ="";
                        string str ="";
                        string sttr ="";
                        String^ cstr = "www";
                        MarshalString(ListaPPs->Text,stpp);
                        stpp = stpp.substr(0,6);
                        stpp = stpp.replace(0,1,"");
                        PuntoDePaso pp = PuntoDePaso::BuscarPP(stpp);
                        stpp = pp.PrintPuntoDePasoCompleto(pp);
                        cstr = gcnew String(stpp.c_str());
                        TextPP->Text = cstr;

                        //Presentar datos de sus trayectos
                        TextTrayecto->Clear();
                        sttr = pp.PrintPPTrayectos();
                        cstr = gcnew String(sttr.c_str());
                        TextTrayecto->Text = cstr;
            }
```

frmTrenes.h

```
        private: System::Void frmTrenes_Load(System::Object^  sender,
System::EventArgs^  e) {
                                string str = "";
                                String^ cstr = "www ";
                                listTrenes->BeginUpdate();
                                int i = 0;
                                list<Tren>::iterator it =
Tren::ListaDeTrenes.begin();
                                while (it != Tren::ListaDeTrenes.end())
                                {
                                str = it->GetIdTren();
                                cstr = gcnew String(str.c_str());
                                listTrenes->Items->Add( String::Format( "
{0}", cstr ) );
                                it++;
                                i++;
                                }
                                listTrenes->EndUpdate();
                                listTrenes->Sorted = true;
                           }
        private: System::Void
listTrenes_SelectedIndexChanged(System::Object^  sender, System::EventArgs^
e) {
                            // Presentar datos del tren seleccionado
                           string str ="";
                           string sttr ="";
                           String^ cstr = "www";
                           MarshalString(listTrenes->Text,sttr);
                           sttr = sttr.replace(0,1,"");
                           Tren tr = Tren::BuscarTren(sttr);
                           str = tr.PrintTren();
                           cstr = gcnew String(str.c_str());
                           textDatosTren->Text = cstr;

                           // Presentar los datos del recorrido
                           str = tr.PrintListaRecorrido();
                           cstr = gcnew String(str.c_str());
                           textRecorrido->Text = cstr;

                           //Presentar los datos de las paradas
                           str = tr.PrintListaParadas();
                           cstr = gcnew String(str.c_str());
                           textParadas->Text = cstr;

                           //Presentar datos del recorrido horario
                           str = tr.PrintListaHorario();
                cstr = gcnew String(str.c_str());
                           textHorario->Text = cstr;

                }
```

frmIncomVia.h

```
        void BorrarTextos(){

                this->txtPp->Text = "";
                this->txtIdTren->Text = "";
                this->txtTitulo->Text = "";
                this->rtxtVerInc->Text ="";

        }

        private: System::Void btnError_Click(System::Object^  sender,
System::EventArgs^  e) {
                                // variables auxiliares
                        string stidbd= "";
                        string str0 ="";
                        string str1 ="";
                        String^ cstr = "www";
                        IncomVia incVia;

                        //limpiar textos
                        this->BorrarTextos();
                        txtTitulo->Text = " Solicitudes de parada en vías
inexistentes";

                        //Busca la incompatibilidades y errores en
solicitudes de parada
                        IncomVia::GenerarIncomVia();
                       // Presenta la lista de incidencias halladas en el
listaInc segun elelcción

                        listaInc->Items->Clear();
                        listaInc->SelectionMode =
SelectionMode::MultiExtended;
                        listaInc->BeginUpdate();
                        list<IncomVia> listpp = IncomVia::ListaDeIncomVia;
                        int i = 0;
                        list<IncomVia>::iterator itp = listpp.begin();
                        while (itp != listpp.end())
                                {
                                        if(itp->GetTipoInc() == 0)
                                        {
                                                str0 = itp->GetIdPp();
                                                str1 = str0 + "   " + itp-
>GetIdTren() + "     " + Util::stConvert(itp->GetNumVia());
                                                str1 = str1  + "        " +
Util::stConvert(itp->GetTipoInc());
                                                cstr = gcnew
String(str1.c_str());
                                                listaInc->Items->Add(
String::Format( " {0}", cstr ) );
                                                i++;
                                        }
                                        itp++;
                                }
                        listaInc->EndUpdate();
```

```
                        }
        private: System::Void btnAnden_Click(System::Object^  sender,
System::EventArgs^  e) {

                                        // variables auxiliares
                        string stidbd= "";
                        string str0 ="";
                        string str1 ="";
                        String^ cstr = "www";
                        IncomVia incVia;

                        //limpiar textos
                        this->BorrarTextos();
                        txtTitulo->Text = " Solicitudes de parada viajeros
en  vías sin anden";

                        //Busca la incompatibilidades y errores en
solicitudes de parada
                        IncomVia::GenerarIncomVia();
                       // Presenta la lista de incidencias halladas en el
listaInc segun elelcción

                        listaInc->Items->Clear();
                        listaInc->SelectionMode =
SelectionMode::MultiExtended;
                        listaInc->BeginUpdate();
                        list<IncomVia> listpp = IncomVia::ListaDeIncomVia;
                        int i = 0;
                        list<IncomVia>::iterator itp = listpp.begin();
                        while (itp != listpp.end())
                                {
                                        if(itp->GetTipoInc() == 1)
                                        {
                                                str0 = itp->GetIdPp();
                                                str1 = str0 + "   " + itp-
>GetIdTren() + "     " + Util::stConvert(itp->GetNumVia());
                                                str1 = str1  + "        " +
Util::stConvert(itp->GetTipoInc());
                                                cstr = gcnew
String(str1.c_str());
                                                listaInc->Items->Add(
String::Format( " {0}", cstr ) );
                                                i++;
                                        }
                                        itp++;
                                }
                        listaInc->EndUpdate();


                }
        private: System::Void btnLongitud_Click(System::Object^  sender,
System::EventArgs^  e) {

                                        // variables auxiliares
                        string stidbd= "";
                        string str0 ="";
                        string str1 ="";
                        String^ cstr = "www";
                        IncomVia incVia;
```

```
                        //limpiar textos
                        this->BorrarTextos();
                        txtTitulo->Text = " Solicitudes de parada en vías
sin suficiente longitud";

                        //Busca la incompatibilidades y errores en
solicitudes de parada
                        IncomVia::GenerarIncomVia();
                       // Presenta la lista de incidencias halladas en el
listaInc segun elelcción

                        listaInc->Items->Clear();
                        listaInc->SelectionMode =
SelectionMode::MultiExtended;
                        listaInc->BeginUpdate();
                        list<IncomVia> listpp = IncomVia::ListaDeIncomVia;
                        int i = 0;
                        list<IncomVia>::iterator itp = listpp.begin();
                        while (itp != listpp.end())
                              {
                                      if(itp->GetTipoInc() == 2)
                                      {
                                              str0 = itp->GetIdPp();
                                              str1 = str0 + "   " + itp-
>GetIdTren() + "    " + Util::stConvert(itp->GetNumVia());
                                              str1 = str1  + "        " +
Util::stConvert(itp->GetTipoInc());
                                              cstr = gcnew
String(str1.c_str());
                                              listaInc->Items->Add(
String::Format( " {0}", cstr ) );
                                              i++;
                                      }
                                      itp++;
                              }
                        listaInc->EndUpdate();

                }
private: System::Void btnBuscar_Click(System::Object^  sender,
System::EventArgs^  e) {

                       // variables auxiliares
                        string stidbd= "";
                        string str0 ="";
                        string str1 ="";
                        String^ cstr = "www";
                        IncomVia incVia;

                         //limpiar textos
                        this->BorrarTextos();
                        txtTitulo->Text = " Todas la solicitudes de parada
con incidencias";

                        //Busca la incompatibilidades y errores en
solicitudes de parada
                        IncomVia::GenerarIncomVia();
                       // Presenta la lista de incidencias halladas en el
listaInc

                        listaInc->Items->Clear();
```

```
                        listaInc->SelectionMode =
SelectionMode::MultiExtended;
                        listaInc->BeginUpdate();
                        list<IncomVia> listpp = IncomVia::ListaDeIncomVia;
                        int i = 0;
                        list<IncomVia>::iterator itp = listpp.begin();
                        while (itp != listpp.end())
                              {
                                    str0 = itp->GetIdPp();
                                    str1 = str0 +  "   " + itp-
>GetIdTren() + "     " + Util::stConvert(itp->GetNumVia());
                                    str1 = str1  + "        " +
Util::stConvert(itp->GetTipoInc());
                                    cstr = gcnew String(str1.c_str());
                                    listaInc->Items->Add( String::Format(
" {0}", cstr ) );
                                    i++;
                                    itp++;
                              }
                        listaInc->EndUpdate();

                }
private: System::Void listaInc_SelectedIndexChanged(System::Object^
sender, System::EventArgs^  e) {

                        // Variables aux
                        string stcapt ="";
                        string stpp ="";
                        string staux ="";
                        string sttr ="";
                        String^ cstr = "www";
                        MarshalString(listaInc->Text,stcapt);

                        // Presentar datos del pp seleccionado
                        stpp = stcapt.substr(0,6);
                        stpp = stpp.replace(0,1,"");
                        PuntoDePaso pp = PuntoDePaso::BuscarPP(stpp);
                        staux = pp.PrintPuntoDePasoCompleto(pp);
                        //cstr = gcnew String(stpp.c_str());
                        txtPp->Text = gcnew String(staux.c_str());;

                        // Presentar datos de la incidencia

                        sttr = stcapt.substr(8,6);
                        sttr = sttr.replace(0,1,"");
                        stpp = stcapt.substr(0,6);
                        stpp = stpp.replace(0,1,"");
                        staux = sttr + "-" + stpp;
                        IncomVia incVia = IncomVia::BuscarIncVia(staux);
                        staux = incVia.PrintIncomVia();
                        //cstr = gcnew String(sttr.c_str());
                        rtxtVerInc->Text = gcnew String(staux.c_str());

                        //Presentar datso para modificar via
                        staux = "1";
                        txtIdTren->Text = gcnew String(sttr.c_str());
                        txtIdPp->Text = gcnew String(stpp.c_str());

                }
```

```
private: System::Void btnModVia_Click(System::Object^  sender,
System::EventArgs^  e) {

                        // Variables aux
                        string stcapt ="";
                        string stpp ="";
                        string staux ="";
                        string sttr ="";
                        String^ cstr = "www";


                        MarshalString(txtIdTren->Text,stcapt);
                        sttr = stcapt.substr(0,5);
                        MarshalString(txtIdPp->Text,stcapt);
                        stpp = stcapt.substr(0,5);
                        int nwvia = Convert::ToInt32(txtSolVia->Text);
                        Tren::ModificarViaParada(sttr, stpp, nwvia);
                        staux = staux +  " Cambio efectuado  al tren   " +
sttr + "  en  " + stpp + "  a via  " + Util::stConvert(nwvia);
                        txtIdTren->Text = gcnew String(staux.c_str());


                }
private: System::Void btnOcultar_Click(System::Object^  sender,
System::EventArgs^  e) {

                         this->Hide();
                }
};
}
```

frmIncTrayecto.h

```
int CalculoPosicionVertical( int ub, int ubMx) {

        int margenSp =60;
        int margenInf =60;
        int altoControl = 100;
        double factorEscala = 1.0;
        int pos = 0;
        altoControl = this->pbxGrafico->Height::get();
        factorEscala = (double)(altoControl-margenSp-margenInf)/ubMx;
        pos = (int) (ub * factorEscala)+ margenSp;
        if (pos >220) {pos = 220;};
        return pos;
}

int CalculoPosicionHorizontal( int hr, int lgMx) {

        int margenIzq =60;
        int margenDrh =20;
        int anchoControl = 100;
        double factorEscala = 1.0;
        int pos = 0;
        anchoControl = this->pbxGrafico->Width::get();
        factorEscala = (double)(anchoControl-margenIzq-margenDrh)/lgMx;
        pos = (int) (hr * factorEscala)+ margenIzq;

        return pos;

}

System::Drawing::Point ObtenerPuntoTextoTren(int ub, int ubMx, int hr, int
lgMx, int p) {

        System::Drawing::Point ptoTexto;
        int posX = 0;
        int posY = 0;
        posY = CalculoPosicionVertical(ub, ubMx);
        posX = CalculoPosicionHorizontal(hr, lgMx);
        if (p==1){posY = posY - 20;};
        if (p==2){posY = posY + 12;};
        posX = posX -30;
        ptoTexto = Point (posX,posY);

        return ptoTexto;
}

System::Drawing::Point ObtenerPuntoGraficoTren(int ub, int ubMx, int hr,
int lgMx) {

        System::Drawing::Point ptoGrafico;
        int posX = 0;
        int posY = 0;
        posY = CalculoPosicionVertical(ub, ubMx);
        posX = CalculoPosicionHorizontal(hr, lgMx);
        ptoGrafico = Point (posX,posY);

        return ptoGrafico;
```

```
}

//Obtener el color del tren
System::Drawing::Color ObtenerColorTren( int prdto) {

        System::Drawing::Color colorTren;
        switch (prdto){
                case 0 :
                        colorTren = System::Drawing::Color::Green;
                        break;
                case 1 :
                        colorTren = System::Drawing::Color::Maroon;
                        break;
                case 2 :
                        colorTren = System::Drawing::Color::Red;
                        break;
                case 3 :
                        colorTren = System::Drawing::Color::Blue;
                        break;
                case 4 :
                        colorTren = System::Drawing::Color::Goldenrod;
                        break;
                case 5 :
                        colorTren = System::Drawing::Color::Indigo;
                        break;
                case 6 :
                        colorTren = System::Drawing::Color::Gainsboro;
                        break;
                case 7 :
                        colorTren = System::Drawing::Color::Gray;
                        break;
                case 8 :
                        colorTren = System::Drawing::Color::LightSteelBlue;
                        break;
                default :
                        colorTren = System::Drawing::Color::LightSteelBlue;
        }
        return colorTren;

}

//Función para cargar la lista de inc en el list_box
void CargarListaInc () {

        //Variables auxiliares
                string staux = " ";
                string str0 = "";
                string str1 = "";
                string idbd = "";
                String^ cstr = "www";

                 // Presenta la lista de incidencias halladas en el
listaInc segun elección

                        listaInc->Items->Clear();
                        listaInc->SelectionMode =
SelectionMode::MultiExtended;
                        listaInc->BeginUpdate();
                        list<IncTrayecto> listinctry =
IncTrayecto::ListaDeIncTrayecto;
```

```
                        int i = 0;
                        list<IncTrayecto>::iterator itinc =
listinctry.begin();
                        while (itinc != listinctry.end())
                               {
                                      PuntoDePaso ppx1 =
PuntoDePaso::BuscarPP(itinc->GetIdPpInic());
                                      PuntoDePaso ppx2 =
PuntoDePaso::BuscarPP(itinc->GetIdPpFin());
                                             str0 = itinc->GetIdTry();
                                             str1 = str0 + "\t" +
ppx1.GetIdPP() + "  " + ppx2.GetIdPP();
                                             str1 = str1  + " : " +
Util::stHora(itinc->GetHInicial()) + "  " + Util::stHora(itinc-
>GetHFinal());
                                             cstr = gcnew
String(str1.c_str());
                                             listaInc->Items->Add(
String::Format( " {0}", cstr ) );
                                             i++;

                                      itinc++;
                               }
                        listaInc->EndUpdate();
                        str0 = Util::stConvert(i) + "  Inc halladas";
                        cstr = gcnew String(str0.c_str());
                        txtBajoLista->Text = cstr;

}

void LimpiarTextos() {

        String^ cstr1 = " ";
         String^ cstr2 = "www";
        txtBajoLista->Text = ".....................";
        txtTitulo->Text = cstr2;
        rtxtTrayecto->Text = cstr1;
        rtxtOcuTry->Text = cstr1;
        listaInc->Visible = false;

}

private: System::Void btnBuscar_Click(System::Object^  sender,
System::EventArgs^  e) {

                //Variables auxiliares

                string idbd = "";
                String^ cstr = "www";

                //Ocultamos el selector de vistas
                this->tabInc->Visible = false;

                //Cogemos el valor de la banda
                MarshalString(cboBandas->Text,idbd);
                idbd = idbd.replace(0,1,"");

                //Generamos la lista de trayectos y trenes, tiempos en
trayecto
                LimpiarTextos();
```

```
                txtTitulo->Text = " Tipo 3 X : Cruce de Trenes en la misma
via";
                OcupTrayecto::GenerarOcupTrayectosFiltroBanda(idbd);

                IncTrayecto::ObtenerIncTrayectoTipoX();

                CargarListaInc();
                listaInc->Visible = true;
                }


private: System::Void btnOcultar_Click(System::Object^  sender,
System::EventArgs^  e) {

                        this->Hide();
                }


private: System::Void frmIncTrayecto_Load(System::Object^  sender,
System::EventArgs^  e) {

                         // Cargar la lista de bandas
        string str = "";
        String^ cstr = "www ";
        cboBandas->BeginUpdate();
        int i = 0;
        list<Banda>::iterator it = Banda::ListaDeBandas.begin();
        while (it != Banda::ListaDeBandas.end())
        {
                str = it->GetIdBanda();
                cstr = gcnew String(str.c_str());
                cboBandas->Items->Add( String::Format( " {0}", cstr ) );
                it++;
                i++;
        }
        cboBandas->EndUpdate();
        }


private: System::Void listaInc_SelectedIndexChanged(System::Object^
sender, System::EventArgs^  e) {

         // Variables aux
                        string stcapt ="";
                        string stidinc = "";
                        string stidtry ="";
                        string staux ="";
                        string sttr ="";
                        String^ cstr = "www";
                        MarshalString(listaInc->Text,stcapt);

                        // Obtener los identificadores
                        stidtry = stcapt.substr(0,6);
                        stidtry = stidtry.replace(0,1,"");
                        stidinc = stcapt.substr(22,5);
                        stidinc = stidtry + "-" + stidinc;

                        // Presentar datos del trayecto e incidencia
seleccionada
                        Trayecto tryx = Trayecto::BuscarTry(stidtry);
```

```
                        staux = tryx.PrintTrayecto() + "\n";
                        rtxtTrayecto->Text = gcnew String(staux.c_str());

                        // Presentar información textual de la incidencia
                        IncTrayecto inctrx =
IncTrayecto::BuscarIncTry(stidinc);
                        staux =  inctrx.PrintIncTryCompleta() + "\n";
                        rtxtOcuTry->Text = gcnew String(staux.c_str());

                         //Mostramos el selector de vistas
                         this->tabInc->Visible = true;

                        //Repintar grafico
                        pbxGrafico->Refresh();
                        pbxHoras->Refresh();
                }


private: System::Void btnBuscaIncA_Click(System::Object^  sender,
System::EventArgs^  e) {

                //Variables auxiliares

                string idbd = "";
                String^ cstr = "www";

                 //Ocultamos el selector de vistas
                this->tabInc->Visible = false;

                //Cogemos el valor de la banda
                MarshalString(cboBandas->Text,idbd);
                idbd = idbd.replace(0,1,"");

                //Generamos la lista de trayectos y trenes, tiempos en
trayecto
                LimpiarTextos();
                txtTitulo->Text = " Tipo 5 A : Adelantamiento en plena
via";
                OcupTrayecto::GenerarOcupTrayectosFiltroBanda(idbd);

                IncTrayecto::ObtenerIncTrayectoTipoA();

                CargarListaInc();
                listaInc->Visible = true;

        }


private: System::Void btnBuscaIncC_Click(System::Object^  sender,
System::EventArgs^  e) {

               //Variables auxiliares

                string idbd = "";
                String^ cstr = "www";

                 //Ocultamos el selector de vistas
                this->tabInc->Visible = false;
```

```
                //Cogemos el valor de la banda
                MarshalString(cboBandas->Text,idbd);
                idbd = idbd.replace(0,1,"");

                //Generamos la lista de trayectos y trenes, tiempos en
trayecto
                LimpiarTextos();
                txtTitulo->Text = " Tipo 4 C : Exceso capacidad de
trayecto";
                OcupTrayecto::GenerarOcupTrayectosFiltroBanda(idbd);

                IncTrayecto::ObtenerIncTrayectoTipoC();

                CargarListaInc();
                listaInc->Visible = true;

        }


private: System::Void pbxGrafico_Paint(System::Object^  sender,
System::Windows::Forms::PaintEventArgs^  e) {

/* Realizamos una representación gráfica de la incidencia en la pestaña
gráfico
**************************************************************************
*****/

        // Crea el elemento grafico de dibujo
    Graphics^ g = e->Graphics;
        Graphics^ gtxt = e->Graphics;
        System::Drawing::Color colorTren;
        System::Drawing::Point pto1;
        System::Drawing::Point pto2;
        System::Drawing::Point pto3;
        System::Drawing::Point ptoTxTr;
        System::Drawing::Font ^ ft = gcnew System::Drawing::Font(
"Arial",8,FontStyle::Bold );
        System::Drawing::Pen ^ pen = gcnew System::Drawing::Pen
(System::Drawing::Color::DimGray);
        pen->DashStyle = System::Drawing::Drawing2D::DashStyle::Dot;
        System::Drawing::SolidBrush^ brush = gcnew
System::Drawing::SolidBrush(System::Drawing::Color::Black);
        System::Drawing::StringFormat^ stformat = gcnew
System::Drawing::StringFormat();
        //stformat->FormatFlags = StringFormatFlags::DirectionVertical;

        // Variables aux
        string stcapt ="";
        string stidinc = "";
        string stidtry ="";
        string staux ="";
        string sttr ="";
        String^ cstr = "www";
        int posY1 = 89;
        int posY2 = 225;
        int hmax = 300;
        int lgtrmx = 15000;
        int hbase = 0;
    int hini = 0;
```

```
        MarshalString(listaInc->Text,stcapt);

        // Obtener los identificadores
        stidtry = stcapt.substr(0,6);
        stidtry = stidtry.replace(0,1,"");
        stidinc = stcapt.substr(22,5);
        stidinc = stidtry + "-" + stidinc;


        // Obtener los datos principales
        Trayecto tryx = Trayecto::BuscarTry(stidtry);
        IncTrayecto inctrx = IncTrayecto::BuscarIncTry(stidinc);
        PuntoDePaso ppinix = PuntoDePaso::BuscarPP(inctrx.GetIdPpInic());
        PuntoDePaso ppfinx = PuntoDePaso::BuscarPP(inctrx.GetIdPpFin());
        Banda bdx = Banda::BuscarBanda(inctrx.GetIdBanda());
        // Calcular posiciones verticales y max horario
        posY2 = this->CalculoPosicionVertical(0,lgtrmx);
        int dist = (int) (tryx.GetLongitud() * 1000);
        if (dist < 4000) {dist = 4000;};
        posY1 = this->CalculoPosicionVertical(dist,lgtrmx);
        hini = inctrx.GetHInicial();
        hbase = (hini- (hini%300));
        hmax = inctrx.GetHFinal()-hbase;
        //Presentar los nombres y abreviaturas de banda y pp
        lblBanda->Text = gcnew String((inctrx.GetIdBanda()).c_str());
        string stnmppini = ppinix.GetPPNombre();
        string stnmppfin = ppfinx.GetPPNombre();
        lblnmPpIni->Location = System::Drawing::Point(10,posY1);
        lblnmPpIni->Text = gcnew String(stnmppini.c_str());
        lblnmPpFin->Location = System::Drawing::Point(10,posY2);
        lblnmPpFin->Text = gcnew String(stnmppfin.c_str());
        lblabvPpIni->Location = System::Drawing::Point(556,posY1);
        lblabvPpIni->Text = gcnew String((ppinix.GetPPAbr()).c_str());
        lblabvPpFin->Location = System::Drawing::Point(556,posY2);
        lblabvPpFin->Text = gcnew String((ppfinx.GetPPAbr()).c_str());

        //Dibujamos las dos líneas horizontales
        g->DrawLine( pen, 1, posY1, pbxGrafico->Width::get()-2, posY1 );
        g->DrawLine( pen, 1, posY2, pbxGrafico->Width::get()-2, posY2);

        //Dibujamos una linea vertical cada 5 minutos

        pen->DashStyle = System::Drawing::Drawing2D::DashStyle::Solid;
        for ( int x=0;x<=hmax;x=x+300)
        {
                int hx = CalculoPosicionHorizontal(x, hmax);
                g->DrawLine( pen, hx, 1, hx, pbxGrafico->Height::get()-2);
        }

        //Dibujamos linea de puntos cada minuto
        pen->DashStyle = System::Drawing::Drawing2D::DashStyle::Dot;
        for ( int x=0;x<=hmax;x=x+60)
        {
                int hx = CalculoPosicionHorizontal(x, hmax);
                g->DrawLine( pen, hx, 1, hx, pbxGrafico->Height::get()-2);
        }

        //*****************************************************************
*****
```

```
        //Dibujamos las líneas de los trenes implicados
        //*************************************************************
******

        //Obtenemos los datos de cada ocup de trayecto implicada
        list<OcupTrayecto> lstOcuTry = inctrx.GetListOcuTry();
        list<OcupTrayecto>::iterator itoc = lstOcuTry.begin();
        while (itoc != lstOcuTry.end())
        {
                Tren tr = Tren::BuscarTren(itoc->GetIdTren());
                colorTren = ObtenerColorTren(tr.GetProducto());
                pen->Color::set(colorTren);
                brush->Color::set(colorTren);
                pen->DashStyle =
System::Drawing::Drawing2D::DashStyle::Solid;
                pen->Width = 1.8;
                int prd = itoc->GetParidad();
                cstr = gcnew String((itoc->GetIdTrParidad()).c_str());
                int hr1 = itoc->GetHEntrada()-hbase;
                int hr2 = itoc->GetHSalida()- hbase;

                //Segun la paridad dibujamos
                if (prd == 2){

                        //Dibujamos el texto del tren
                        ptoTxTr = ObtenerPuntoTextoTren (dist,
lgtrmx,hr1,hmax, prd);
                        gtxt->DrawString(cstr, ft, brush,ptoTxTr, stformat);
                        //Dibujamos los puntos del tren
                        pto1 = ObtenerPuntoGraficoTren (dist,
lgtrmx,hr1,hmax);
                        pto2 = ObtenerPuntoGraficoTren (0, lgtrmx,hr2,hmax);
                        g->DrawLine( pen,pto1,pto2);
                }
                if (prd == 1){

                        //Dibujamos el texto del tren
                        ptoTxTr = ObtenerPuntoTextoTren (0, lgtrmx,hr1,hmax,
prd);
                        gtxt->DrawString(cstr, ft, brush,ptoTxTr, stformat);
                        //Dibujamos los puntos del tren
                        pto1 = ObtenerPuntoGraficoTren (0, lgtrmx,hr1,hmax);
                        pto2 = ObtenerPuntoGraficoTren (dist,
lgtrmx,hr2,hmax);
                        g->DrawLine( pen,pto1,pto2);
                        }
                itoc++;
                }


}


private: System::Void pbxHoras_Paint(System::Object^  sender,
System::Windows::Forms::PaintEventArgs^  e) {

        //Dibujar las referencias horarias
        //*****************************************************

        // Crea el elemento grafico de dibujo
```

```
        Graphics^ g = e->Graphics;
        System::Drawing::Pen ^ pen = gcnew System::Drawing::Pen
(System::Drawing::Color::DimGray);
        pen->DashStyle = System::Drawing::Drawing2D::DashStyle::Dot;

        // variables auxiliares
        string stcapt ="";
        string stidinc = "";
        string stidtry ="";
        string staux ="";
        string sttr ="";
        String^ cstr = "www";
        int hmax = 300;
        int hbase = 0;
    int hini = 0;

        MarshalString(listaInc->Text,stcapt);

        // Obtener los identificadores
        stidtry = stcapt.substr(0,6);
        stidtry = stidtry.replace(0,1,"");
        stidinc = stcapt.substr(22,5);
        stidinc = stidtry + "-" + stidinc;
        IncTrayecto inctrx = IncTrayecto::BuscarIncTry(stidinc);
        hini = inctrx.GetHInicial();
        hbase = (hini- (hini%300));
        hmax = inctrx.GetHFinal()-hbase;

        //Dibujamos una pequeña linea cada minuto
        for ( int x=0;x<=hmax;x=x+60)
        {
                int hx = CalculoPosicionHorizontal(x,hmax);
                g->DrawLine( pen, hx, 1, hx, 6);
        }

        //Dibujamos las horas cada 5 minutos
        int lahora = hbase;
        for ( int x=0;x<=hmax;x=x+300)
        {
                int hx = CalculoPosicionHorizontal(x,hmax);
                string mihora = Util::stHora(lahora);
                lahora = lahora + 300;
                cstr = gcnew String(mihora.c_str());
                System::Drawing::Font ^ ft = gcnew System::Drawing::Font(
"Arial",10,FontStyle::Bold );
                g->DrawString(cstr,ft, System::Drawing::Brushes::Blue, hx-
12, 6);
        }


 }

 /* Fin de la clase frmIncTrayecto  */
};
  }
```