3.2 Modelado del sistema con UML.

En esta sección se exponen los diagramas de clases y de clases de uso de la aplicación. Para que la aplicación esté completamente definida, sólo quedará entonces la definición de la implementación de cada uno de los casos de uso. Esta implementación puede realizarse mediante diagramas de interacción o incluso mediante texto plano o pseudocódigo.

La implementación de los casos de uso se hace en texto plano en la sección donde se describen los distintos interfaces. No obstante, se ha incluido algunos diagramas de interacción, concretamente de secuencia, para mostrar como se definiría la implementación usando este tipo de diagramas.

En primer lugar se incluyen los diagramas de casos de uso. Se han usado varios niveles, cada uno de ellos aporta una mayor cantidad de detalle. Esto tiene como fin facilitar la comprensión del modelo.

Los diagramas de casos de uso se usan para modelar los requisitos del sistema, especificando qué debería hacer el sistema, independientemente de cómo lo haga.

En segundo lugar se mostrará el diagrama de clases. El diagrama de clases aporta información sobre la vista de diseño estática del sistema.

En el diagrama de clases se incluyen las propiedades y las operaciones más importantes de cada clase. El incluir todas las propiedades y operaciones no habría hecho más que dificultar la comprensión del modelo y del propio diagrama. Incluso en algunos diagramas se han omitido algunas clases de la biblioteca de clases visuales de Borland (VCL de Borland) debido a que ya se habían usado anteriormente en otros diagramas y lo único que hacían era empeorar la comprensión del diagrama. La mayoría de estas clases sirven para crear objetos gráficos que son propiedades de alguna clase. En todas las clases de las que son propiedades, estas clases de la VCL de Borland aparecen realmente como un puntero a dicha clase, por lo que la relación es siempre una agregación, no una composición.

Finalmente los diagramas de interacción son una forma de mostrar la implementación concreta de los casos de uso. En concreto, se usarán diagramas de secuencia, aunque igualmente válidos hubieran sido los diagramas de colaboración.

3.2.1 Diagramas de casos de uso

Los diagramas de casos de uso se usan para modelar los requisitos del sistema, especificando qué debería hacer el sistema, independientemente de cómo lo haga.

Para capturar mejor el funcionamiento del sistema, se han definido varios niveles de casos de uso.

Cuando se ha llegado a un nivel suficiente de detalle se ha procedido a la agrupación de casos de uso según los distintos interfaces de la aplicación, es decir, interfaz gráfica (representación de red vial), interfaz de nodos, interfaz de tramos e interfaz de líneas de autobús.

En los diagramas de casos de uso intervienen actores que son externos al sistema. En todos los diagramas de casos de uso que se van a ver se muestra un único actor que es el usuario de la aplicación. El usuario de la aplicación interacciona con los casos de uso mediante asociaciones.

En la práctica, el usuario interacciona con el sistema mediante la interfaz gráfica proporcionada: botones, cuadros de diálogo, etcétera.

3.2.1.1 Diagrama de casos de uso de nivel 0

En primer lugar se muestra el diagrama con mayor nivel de abstracción por tanto menor nivel de detalle (figura 3.2.1.1.1).

Figura 3.2.1.1.1: Casos de uso nivel 0

En este diagrama se muestra como el actor, el usuario, interacciona con el caso de uso *Acceder a información de la base de datos de la red vial*. Este caso de uso es muy genérico, pero define perfectamente qué es lo que puede hacer el usuario con el sistema. Mediante el uso de los distintos interfaces, en definitiva el usuario lo que hace es consultar una base de datos. Una base de datos que puede modificar.

3.2.1.2 Diagrama de casos de uso de nivel 1

En un menor nivel de abstracción y, por lo tanto, mayor nivel de detalle, está el siguiente diagrama. En este diagrama (figura 3.2.1.2.1) se han separado los distintos tipos de consulta que puede realizar el usuario. Cada caso de uso representa un tipo de consulta. Así, se tienen los siguientes tipos: el caso de uso *Cargar red vial y representarla gráficamente* consiste en ver la representación gráfica de la red vial y realizar operaciones de zoom y windowing. El caso de uso *Cargar interfaz de tramos* consiste en realizar operaciones sobre los tramos, puntos de medida y carriles, el interfaz de tramos, en definitiva. Análogamente, los demás casos de uso engloban las operaciones que pueden realizarse con cada uno de los interfaces de usuario: *Cargar interfaz de nodos* (interfaz de nodos) y *Cargar interfaz de líneas de bus* (interfaz de líneas de autobús).

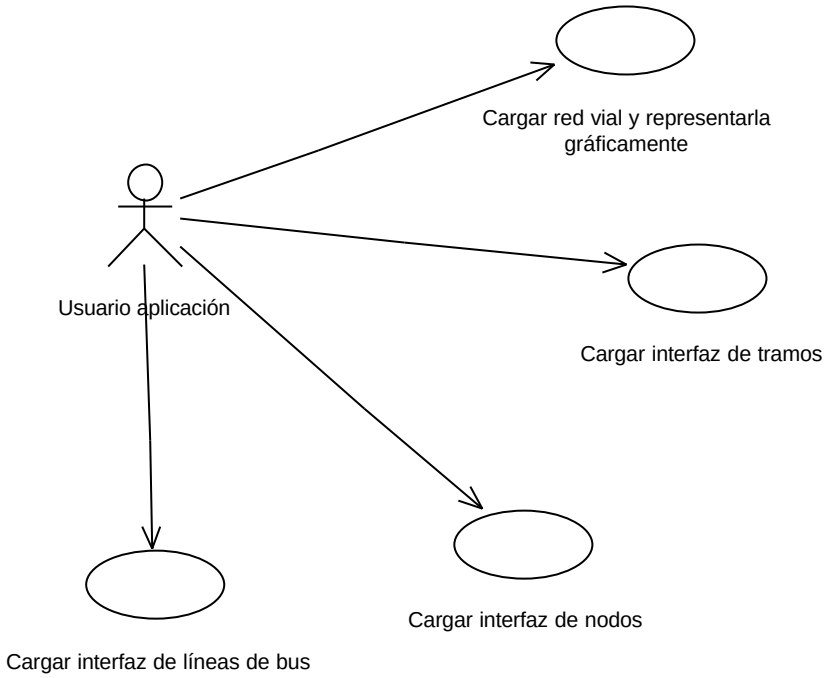

Figura 3.2.1.2.1: Casos de uso nivel 1

El siguiente nivel ya presenta el máximo nivel de detalle, y se muestran las operaciones que puede realizar el usuario sobre cada interfaz. Por ello, llegados a este nivel de detalle, se han creado varios diagramas de casos de uso, cada uno de los cuales corresponde a un caso de uso de los mostrados en el nivel anterior.

3.2.1.3 Diagramas de casos de uso de nivel 2

En esta sección se muestran distintos diagramas de casos de uso, cada uno de los cuales agrupa las acciones que puede realizar el usuario en cada uno de los interfaces.

El diagrama de casos de uso para el interfaz de representación gráfica

En este diagrama de casos de uso (figura 3.2.1.3.1) se muestran las operaciones que puede realizar el usuario en el interfaz de representación gráfica.

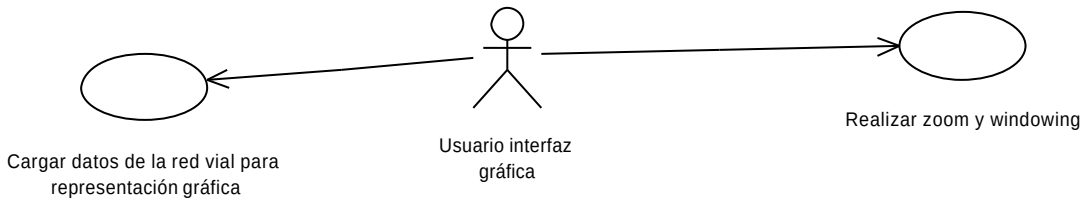

Figura 3.2.1.3.1: Interfaz de representación gráfica (nivel 2)

En este diagrama aparecen dos casos de uso: *Cargar datos de la red vial para representación gráfica* y *Realizar zoom y windowing*.

El primero de ellos se refiere a la acción de buscar la información sobre tramos y nodos de la red vial en la base de datos para representarla gráficamente.

El segundo de ellos engloba las operaciones de zoom y windowing que pueden realizarse sobre la representación de la red vial.

El diagramas de casos de uso para el interfaz de nodos

En este diagrama se agrupan las operaciones que puede realizar el usuario usando el interfaz de nodos. El diagrama se muestra en la figura 3.1.2.3.2.

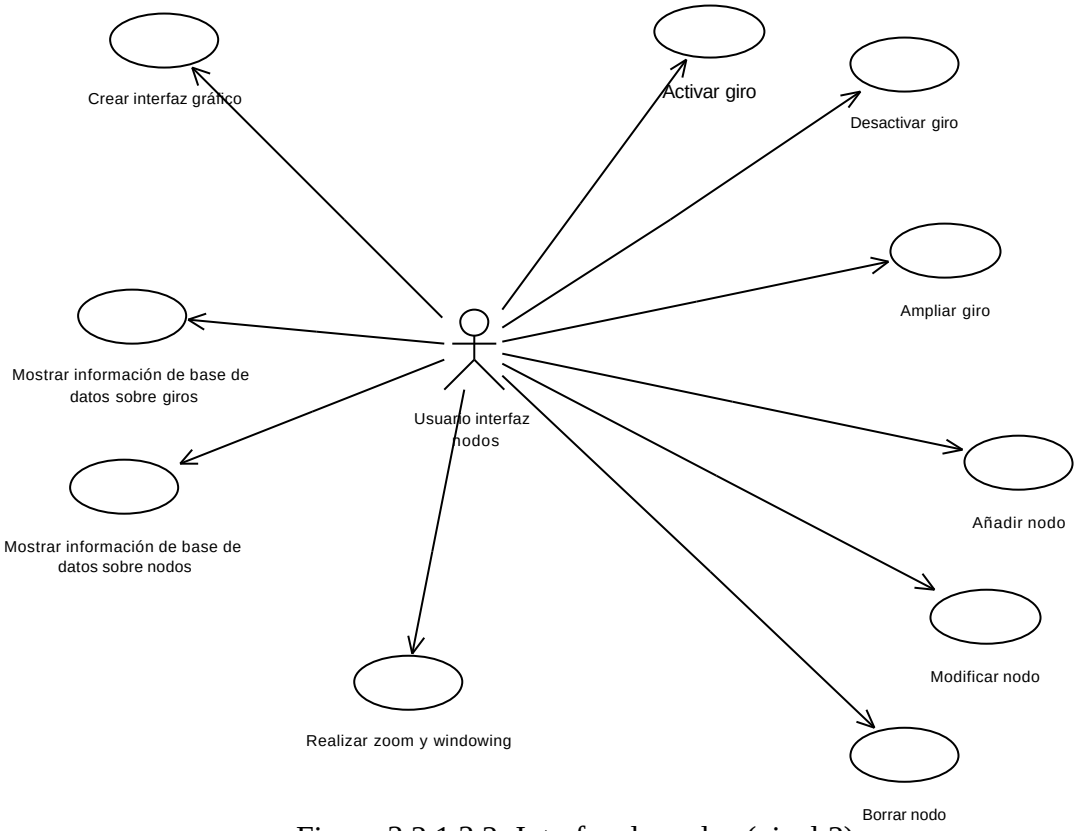

Figura 3.2.1.3.2: Interfaz de nodos (nivel 2)

El caso de uso *Crear interfaz gráfico* presenta en pantalla las dos imágenes que representan distintas vistas de la red vial en el interfaz de nodos y realiza las operaciones necesarias para que los nodos se puedan seleccionar sobre esas imágenes. También crea el interfaz gráfico, es decir, los cuadros de diálogo, las páginas con sus pestañas, etcétera.

Mostrar informacion de base de datos sobre giros busca en la base de datos para mostrar la información existente sobre los giros que tienen lugar en el nodo seleccionado y representar el diagrama con los giros posibles y permitidos.

Mostrar informacion de base de datos sobre nodos busca en la base de datos información sobre el nodo seleccionado y la muestra en el interfaz.

Realizar zoom y windowing permite realizar operaciones de zoom y de windowing sobre cualquiera de las dos imágenes creadas por Crear interfaz gráfico.

Los casos de uso *Activar giro* y *Desactivar giro* sirven para permitir o prohibir giros. Por defecto los giros están todos prohibidos hasta que el usuario los permite.

El caso de uso *Ampliar giro* se encarga de crear un nuevo formulario donde se representa el diagrama de giros posibles y permitidos a mayor tamaño.

Finalmente, los casos de uso *Añadir nodo*, *Borrar nodo* y *Modificar nodo* acceden a la base de datos para añadir un nuevo nodo, borrar un nodo existente o modificar la información de uno, respectivamente.

El actor *Usuario interfaz nodos* es el usuario, que puede interaccionar normalmente con todos estos casos de uso pulsando los botones adecuados en el interfaz, excepto *Crear interfaz gráfico*, que tiene lugar cuando se carga el interfaz.

El diagrama de casos de uso para el interfaz de tramos

En este diagrama se muestran los casos de uso con los que puede interaccionar el usuario, representado por el actor *Usuario interfaz tramos*, en el interfaz de tramos. El diagrama de casos de uso se muestra en la figura 3.2.1.3.3.

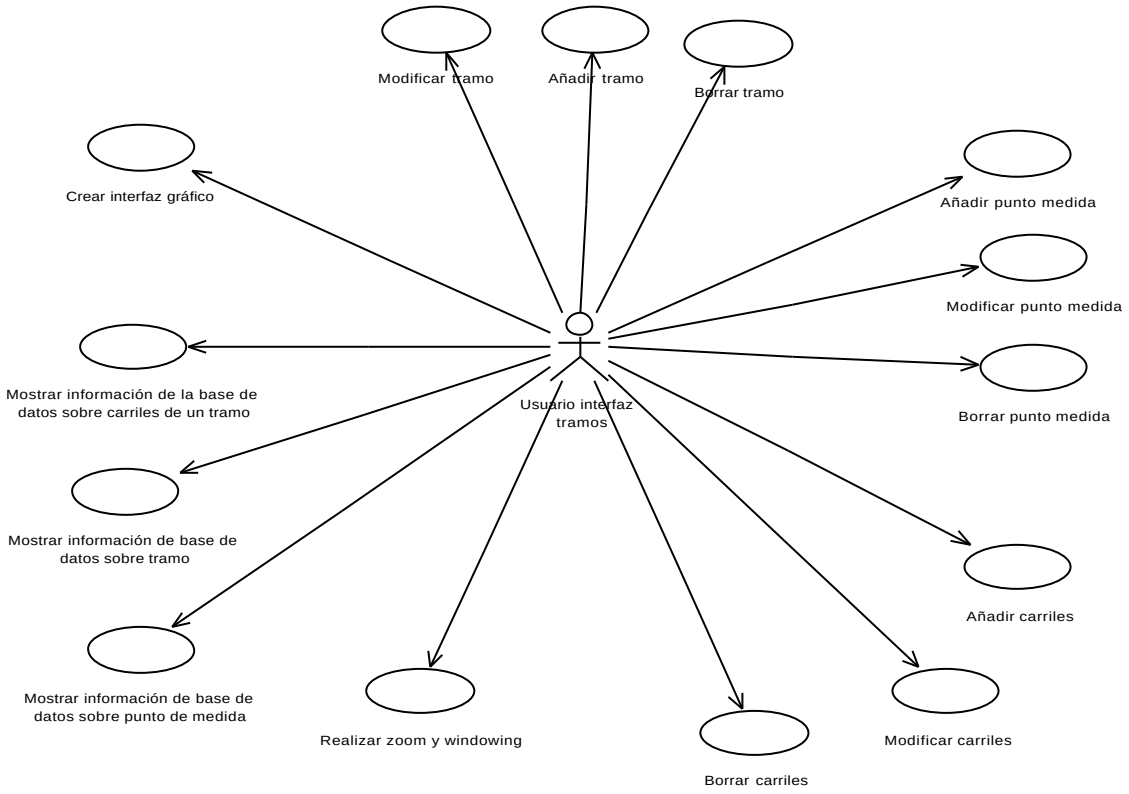

Figura 3.2.1.3.3: Interfaz de tramos (nivel 2)

Los casos de uso en este diagrama son análogos a los del interfaz de nodos.

El caso de uso *Crear interfaz gráfico* presenta en pantalla las dos imágenes que representan distintas vistas de la red vial en el interfaz de tramos y realiza las operaciones necesarias para que los tramos se pueden seleccionar sobre esas imágenes. También crea el interfaz gráfico, es decir, los cuadros de diálogo, las páginas con sus pestañas, etcétera.

Mostrar informacion de base de datos sobre carriles de un tramo busca en la base de datos para mostrar la información existente sobre los carriles del tramo seleccionado y presentarla en el interfaz.

Mostrar informacion de base de datos sobre tramo busca en la base de datos información sobre el tramo seleccionado y la muestra en el interfaz.

Mostrar informacion de base de datos sobre puntos de medida busca en la base de datos información sobre los puntos de medida y detectores del tramo seleccionado y la muestra en el interfaz.

Realizar zoom y windowing permite realizar operaciones de zoom y de windowing sobre cualquiera de las dos imágenes creadas por Crear interfaz gráfico.

Los casos de uso *Añadir tramo, Borrar tramo* y *Modificar tramo* acceden a la base de datos para añadir un nuevo tramo, borrarlo o modificar la información de uno existente, respectivamente.

Análogamente con los casos de uso *Añadir punto medida, Modificar punto medida* y *Borrar punto medida* con los puntos de medida pertenecientes al tramo seleccionado y con los casos de uso *Añadir carriles, Borrar carriles* y *Modificar carriles* con los carriles del tramo seleccionado.

El diagrama de casos de uso para el interfaz de líneas de bus

En el diagrama de casos de uso para el interfaz de línea de autobús (figura 3.2.1.3.4) se muestran las operaciones que puede realizar el usuario, representado por el actor *Usuario interfaz lineas de bus*, en este interfaz.

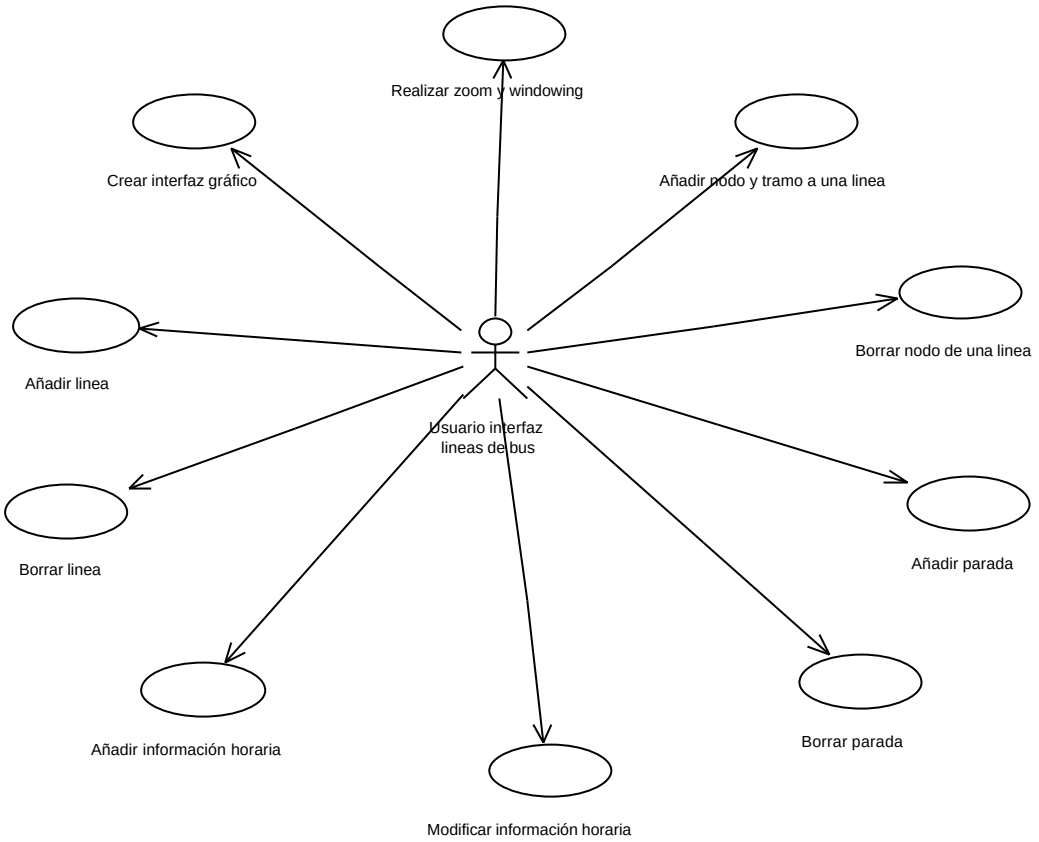

Figura 3.2.1.3.4: Interfaz de líneas de bus (nivel 2)

De nuevo el caso de uso *Crear interfaz gráfico* se encarga de crear todo el interfaz, incluidas las dos imágenes, y el caso de uso *Realizar zoom y windowing* se encarga de realizar dichas operaciones sobre estas imágenes.

Los casos de uso *Añadir linea* y *Borrar linea* permiten, como su propio nombre indica, añadir nuevas líneas de bus o eliminar las existentes.

Los casos de uso *Añadir información horaria* y *Modificar información horaria* permiten al usuario modificar la frecuencia de paso de los autobuses de la línea seleccionada sobre la parada seleccionada.

Los casos de uso *Añadir parada* y *Borrar parada* permiten añadir una nueva parada en el recorrido de la línea seleccionada o borrar una existente.

El caso de uso *Añadir nodo y tramo a una línea* añade un nuevo nodo y, por tanto, el tramo que lo conecta con el anterior, al recorrido de la línea seleccionada. El caso de uso *Borrar nodo* de una línea borra un nodo (y el tramo correspondiente) del recorrido de la línea seleccionada.

3.2.2 Diagramas de clases

Para mejorar la organización del diagrama de clases, se ha realizado una agrupación con paquetes, como se muestra en la figura 3.2.2.1.

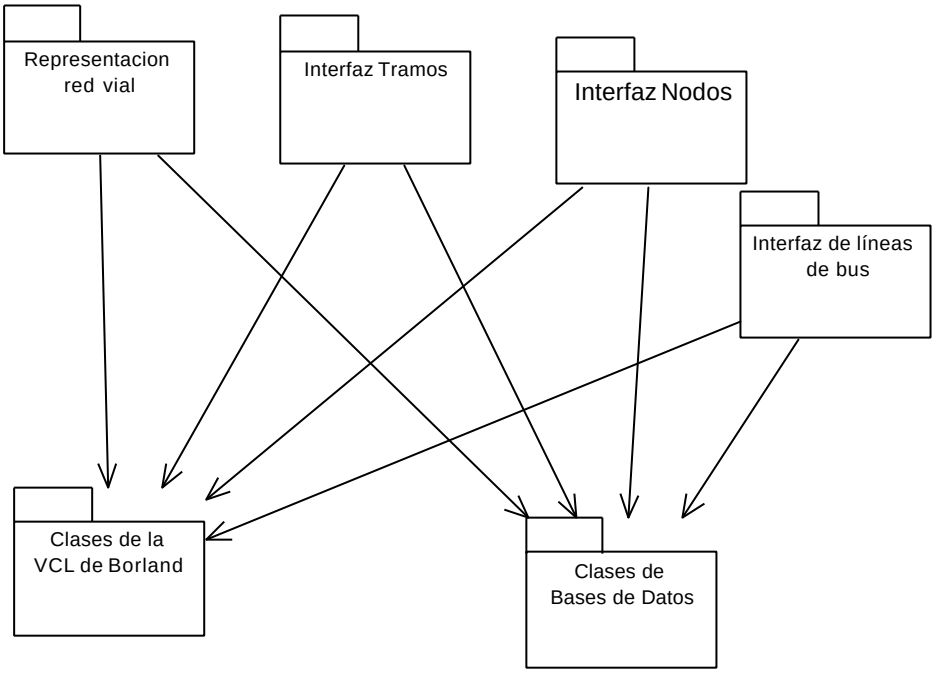

Figura 3.2.2.1: División en paquetes

El paquete de la VCL de Borland

El paquete más simple es el *Clases de la VCL (Visual Class Library) de Borland*. Este paquete, como puede apreciarse a continuación, contiene todas las clases de la VCL de Borland que se usan en la aplicación. Tiene simplemente un fin organizativo. Las clases pertenecientes a este paquete se usarán en todos los demás.

Las clases agrupadas bajo este paquete son:

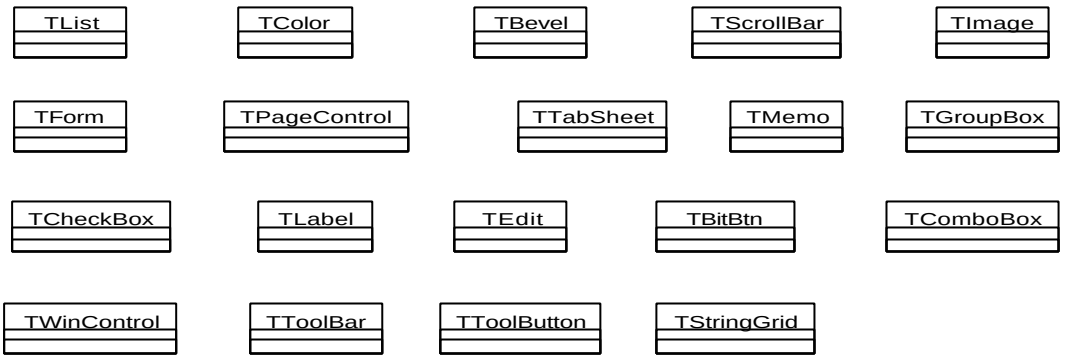

Figura 3.2.2.2: Clases de la VCL de Borland

El paquete de las Clases de Bases de Datos

El otro paquete que contiene clases que se usan en los otros paquetes es el *Clases de Bases de Datos*. Bajo este paquete se agrupan las clases cuya función principal es la de almacenar datos. En la aplicación hay, fundamentalmente, dos tipos de clases, las que se encargan de almacenar información, que se guardan en este paquete, y las que tienen como función representar gráficamente un interfaz o parte de él. Aunque hay algunas clases mixtas, como *Giro*, pero son las menos comunes.

Este paquete incluye gran cantidad de clases, por lo que se van a representar en varias figuras.

Profundizando un poco en el paquete *Clases de Bases de Datos*, una de las clases que incluye es *TList* Como se indica en el diagrama, esta clase se ha tomado del paquete *Clases de la VCL de Borland*. Esta clase tiene una relación uno a uno con la clase *Lista_inf*, que es una clase plantilla con un parámetro. Es decir, la clase *Lista_inf* es una clase genérica y sus funciones son funciones genéricas.

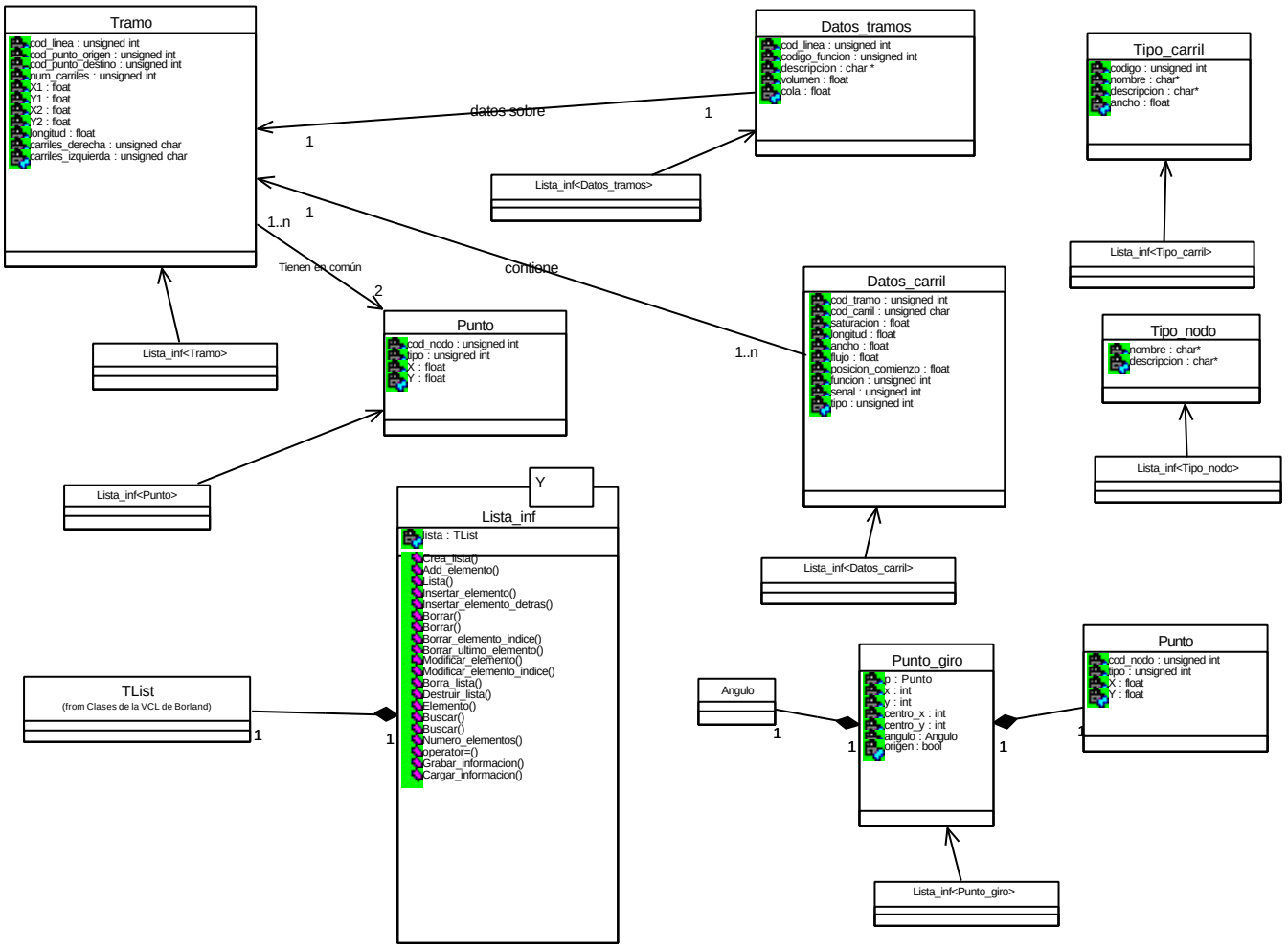

Figura 3.2.2.3: Clases de Bases de Datos (I)

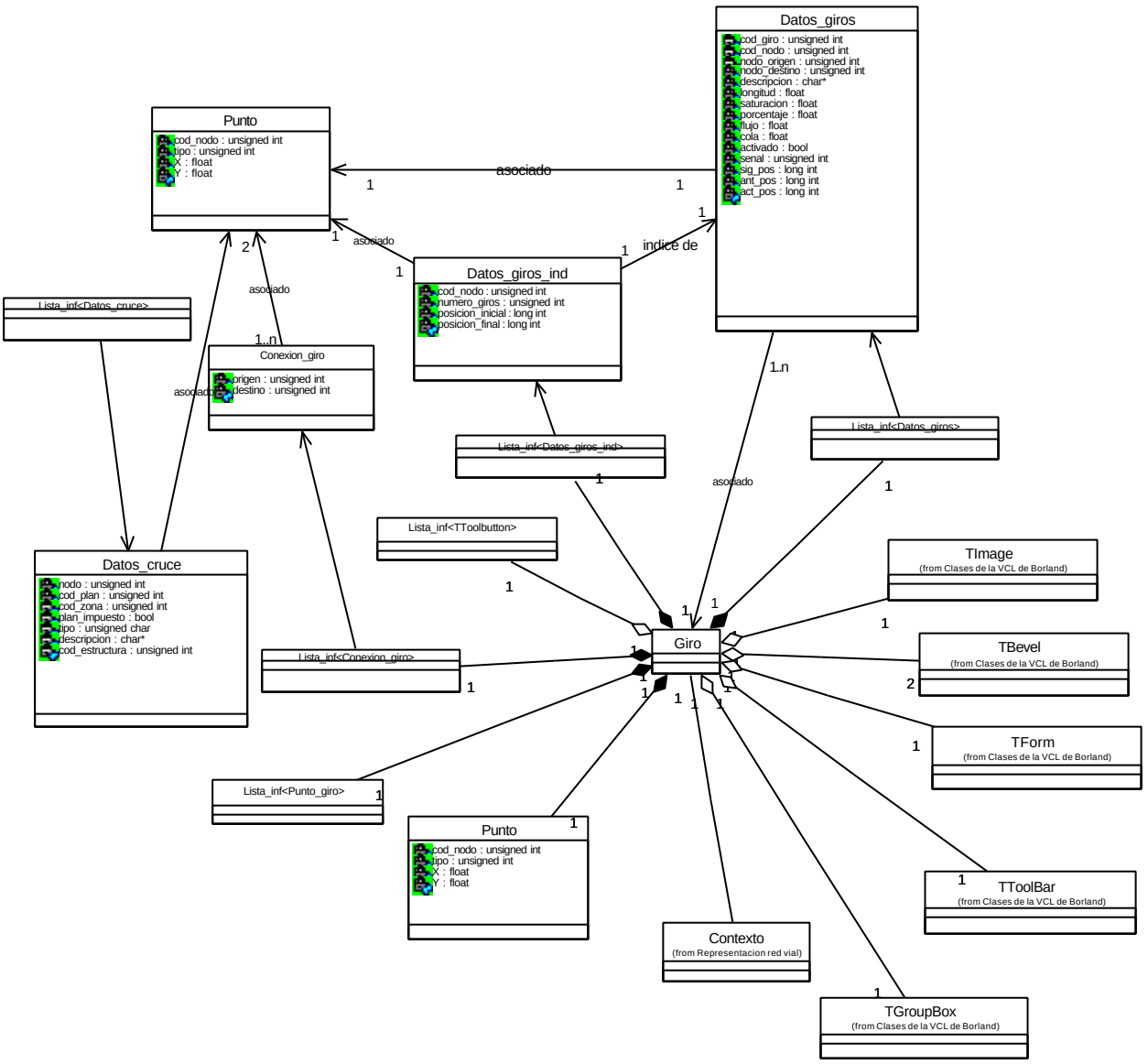

Figura 3.2.2.4: Clases de Bases de Datos (II)

En la figura anterior no se muestran los atributos ni las operaciones de la clase *Giro* para facilitar su representación en pantalla. La clase *Giro* completa se muestra en la figura 3.2.2.5.

La relación es una composición, como indica el rombo relleno. Es decir, hay una propiedad en la clase *Lista_inf* que es de tipo *TList* (no puntero a *TList*). Pueden apreciarse la cantidad de operaciones definidas para la clase *Lista_inf*: estas operaciones son las que proporcionan tanta potencia a esta clase genérica. Todas estas operaciones se explican detalladamente en la sección donde se habla de las plantillas.

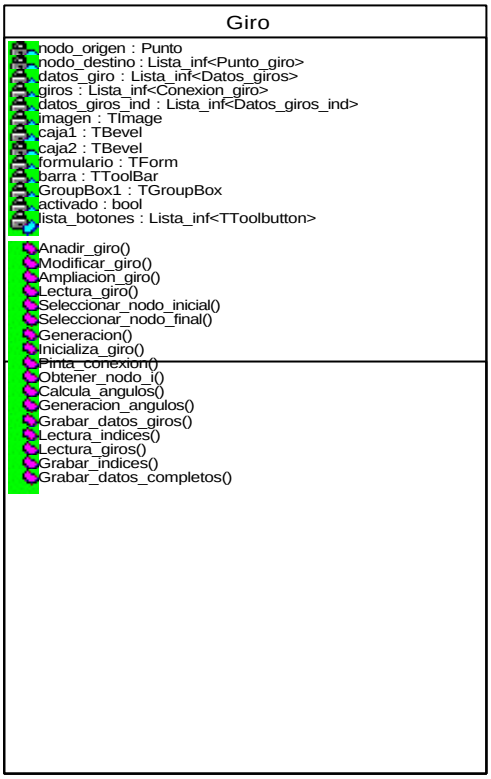

Figura 3.2.2.5: La clase Giro

Aparecen también muchas clases cuyo nombre es *Lista_inf<clase>*. Esto es debido a que estas clases son instancias de la clase genérica *Lista_inf* con un determinado parámetro, que es la clase que aparece entre los símbolos < y >.

La clase *Punto* contiene la información básica sobre los nodos de la red vial: posición, tipo y código. Puede observarse como hay una clase *Lista_inf<Punto>* que tiene una relación de dependencia con ella. Esto es únicamente porque la clase *Punto* es el parámetro que recibe la clase genérica *Lista_inf*, y se representa mediante esta dependencia. Todas las clases que son instancias de *Lista_inf* para una determinada clase tienen esta relación de dependencia con la clase que se ha pasado como parámetro. Por lo tanto, hay una lista cuyos nodos son objetos de la clase *Punto*: hay una lista con los nodos de la red.

La clase *Tramo* contiene la información básica sobre los tramos de la red vial: código y nodos origen y destino, entre otras cosas. La propiedad *cod_linea* es un entero sin signo único para cada tramo. Las propiedades *cod_punto_origen* y *cod_punto_destino* son los códigos de los nodos de origen y destino del tramo. Las propiedades *X1*,*Y1*,*X2* e *Y2* son las coordenadas de los nodos origen y destino del tramo. Las coordenadas del nodo origen son (*X1*,*Y1*) y las del nodo destino son (*X2*,*Y2*). La propiedad *longitud* indica la longitud del tramo, y las propiedades *carriles_derecha* y *carriles_izquierda* indican el número de carriles que adicionales que tiene el tramo.

Esta clase tiene una relación de dependencia con la clase *Punto*, puesto que cada tramo se define como el elemento del viario que une dos nodos, el nodo origen y el destino. También existe una lista con los tramos de la red, como puede observarse.

La siguiente clase es *Datos_tramos*. Esta clase almacena una serie de características sobre el tramo relacionadas con el tráfico, como el volumen (atributo volumen) y la cola (atributo *cola*), y no con su geometría. Pero la información que almacena es precisamente sobre tramos, por lo que la dependencia con la clase *Tramo* es clara. También existe una lista con objetos de esta clase.

La siguiente clase es *Datos_carril.* Esta clase almacena información sobre cada uno de los carriles de un tramo. La propiedad *cod_tramo* es el código del tramo al que pertenece, la propiedad *cod_carril* es el código de carril dentro del tramo. El resto de las propiedades son características del carril: *longitud, ancho, flujo, funcion, tipo*, etcétera.

Por lo tanto también tiene una relación de dependencia con la clase *Tramo*. También existe una lista de objetos *Datos_carril*.

La clase *Tipo_carril* se usa para almacenar los tipos de carril que hay. No almacena información sobre carriles en concreto, sino de los diferentes tipos de carril que puede haber. También existe una lista con objetos de esta clase, con lo que consultando dicha lista se tienen los distintos tipos de carril que pueden existir. La propiedad *codigo* identifica a los tipos de carril dentro de la lista. Las propiedades *nombre* y *descripcion* son autodescriptivas. La propiedad *ancho* indica el ancho del carril en metros.

La clase *Tipo_nodo* guarda el nombre y la descripción de un nodo. También existe una lista de objetos de la clase *Tipo_nodo*.

La clase *Punto_giro* guarda la información geométrica necesaria para un nodo para poder representarlo en el diagrama donde se muestran los giros y que permite activarlos o no. La información esencial que almacena esta clase es el nodo (mediante la propiedad *p*, de la clase *Punto*) que está unido al nodo que se desea borrar por cierto tramo y si este nodo es origen o destino de dicho tramo. La determinación de si es nodo origen o destino la proporciona la propiedad *Origen*.

Una de las propiedades es de la clase *Punto* y otra de la clase *Angulo*. Por ello existe una composición entre estas clases y la clase *Punto_giro*. También existe una lista de objetos de la clase *Punto_giro* que, a su vez, será una propiedad de la clase *Giro*.

La clase *Datos_cruce* guarda una serie de datos sobre los cruces. Como es de suponer, existe una lista con objetos de esta clase. No hay que olvidar que, al contener datos de cruces, tiene una relación de dependencia con la clase *Punto*, como se muestra en el diagrama.

La clase *Conexión_giro* se usa para saber qué giros han sido activados por el usuario (existen y están permitidos) o no (existen pero no están permitidos), y tan sólo tiene dos propiedades, *origen* y *destino*. Existe una lista de objetos de esta clase que también será una propiedad de la clase *Giro*. Como un giro lo define como el nodo origen y el destino, la dependencia con la clase *Punto* es clara.

La clase *Datos_giros* guarda información sobre cada uno de los giros: tanto el nodo origen como el destino como el nodo en que se produce dicho giro. Además contiene información relativa al tráfico para ese giro. La primera propiedad, *cod_giro* es un código para identificar al giro. La propiedad *cod_nodo* contiene el código del nodo donde se produce el giro. La propiedad *nodo_origen* contiene el código del nodo origen del giro y la propiedad *nodo_destino* contiene el código del nodo destino del giro. Las propiedades *sig_pos*, *ant_pos* y *act_pos* contienen las posiciones de los giros siguiente, anterior y la del actual, respectivamente, en la lista *datos_giro*, que es una lista del tipo *Lista_inf*. La lista *datos_giro* es una propiedad de la clase *Giro* y está formada por objetos de la clase *Datos_giro*. Las posiciones de los giros siguiente y anterior hacen referencia a los giros que tienen lugar en el mismo nodo que el actual. Esto evita que al buscar todos los giros asociados a un nodo en la lista se tenga que recorrer toda la lista. Una vez encontrado un giro que tiene lugar en el nodo deseado puede accederse a todos los demás usando estas propiedades. El resto de las propiedades contiene información descriptiva del nodo.

Existe una lista de objetos de la clase *Datos_giros*. De nuevo, debido a que el giro se define mediante el nodo en que se produce y el nodo origen y destino, la dependencia con la clase *Punto* es clara.

La clase *Datos_giros_ind* es una clase que se usa como índice al fichero donde se almacenan los datos *Datos_giros*. De esta manera en la lista de objetos *Datos_giros* sólo hay que tener los que se están usando, y cuando hace falta uno nuevo se lee rápidamente del fichero ya que se sabe la posición donde está almacenado gracias al objeto *Datos_giros_ind* correspondiente. La propiedad *cod_nodo* almacena el código del nodo al que se refiere la información. La propiedad *numero_giros* almacena el número de giros que se producen en este nodo. Finalmente la propiedad *posicion_inicial* y *posicion_final* indican la posición donde están los datos en la lista de *Datos_giros*.

Mantiene una relación de dependencia con la clase *Punto* y también con la clase *Datos_giros*, por ser el índice de ésta.

Todas las clases que se han visto hasta ahora carecen de operaciones, exceptuando *Lista_inf*. En realidad estas clases sí disponen de operaciones, pero todas ellas tienen como única función extraer la información de la clase o almacenarla en la misma. Esto es normal debido a que son clases de almacenamiento. Como se verá ocurre todo lo contrario con las clases que aparecen en cada uno de los interfaces: todos los atributos no son más que objetos de la VCL, para representar gráficamente los interfaces.

Sin embargo la clase *Giro* sí tiene varias operaciones. Esto es debido a que, aunque existe una clase que se encarga de crear el interfaz, muchas de las operaciones que tiene dicha clase lo único que hacen es llamar a operaciones de la clase *Giro*. Es por esto que la clase *Giro* tiene bastantes operaciones.

La clase *Giro* almacena información sobre todos los giros que pueden producirse en un nodo, permitidos o no. La lista *nodo_destino* almacena todos los posibles giros. La lista *giros* almacena los giros permitidos. Las listas *datos_giros* y *datos_giros_ind* guardan la información sobre los giros y la localización de éstos en el fichero, respectivamente.

Las demás propiedades tienen que ver con la representación gráfica del interfaz. En concreto, las propiedades *imagen* y *formulario* se encargan de crear la ampliación del diagrama que representa los posibles giros.

Las operaciones *Añadir_giro* y *Modificar_giro* se usan para gestionar giros y se lanzan cuando se produce el evento *onClick* sobre el botón correspondiente. Como estas operaciones, hay muchas más que normalmente debían llevarse a cabo en la clase que se encarga de crear el interfaz gráfico, pero en este caso se han añadido aquí.

La operación *Lectura_índices* reconstruye la información de la lista *datos_giros_ind* a partir de los datos almacenados en un fichero de disco.

La operación *Inicializa_giro* tiene como fin actualizar o cargar toda la información necesaria en las listas adecuadas.

La operación *Mostrar_nodo* se encarga de representar en la propiedad *imagen* de la clase *Giro* cada uno de los nodos que aparecen en la lista *nodo_destino* sobre un círculo imaginario cuyo centro es el nodo seleccionado. La posición de los nodos sobre este círculo de cada uno de los nodos a representar viene dada por los ángulos que forman cada uno de estos nodos con el nodo seleccionado.

La operación *Generación_angulos* se encarga de calcular los ángulos necesarios para la representación de los giros.

La operación *Pinta_conexión* se encarga de pintar los posibles giros desde cada uno de los nodos hacia los demás. Para ello hace uso de las operaciones *Conectados_nodos* y *Pinta_conexión_origen*.

La operación *Conectados_nodo* se usa para saber si el giro está permitido o no. Para ello lo busca en la lista *giros*. Según el resultado obtenido para cada para de nodos por este método se llama a al método *Pinta_conexión_origen* con un parámetro u otro según el giro esté permitido o no.

El método *Pinta_conexión_origen* pinta las conexiones entre nodos, las permitidas como una línea sólida y las no permitidas como líneas discontinuas.

Las operaciones *Seleccionar_nodo_inicial* y *Seleccionar_nodo_final* se usan para activar un giro en el diagrama de los giros. Para seleccionar un giro se necesitan seleccionar dos nodos, el origen y el destino, ya que el nodo central está fijado.

La clase *Giro* tiene varias relaciones de composición con varias clases, la mayoría de ellas listas, y la clase *Punto.* Esto es debido a que algunas de las propiedades de la clase *Giro* son estas listas, y también una de las propiedades es de la clase *Punto.* Sin embargo, también tiene otras propiedades como *imagen,* de la clase *TImage* o caja1 y caja2 de la clase *TBevel*. Sin embargo la relación entre estas clases y la clase *Giro* no es de composición, sino simplemente de agregación. Si se observa el código en C++ de la clase *Giro,* que se muestra en la sección donde se explican las clases que intervienen en cada interfaz , puede observarse que mientras las distintas listas o el objeto de la clase *Punto* son propiedades de la clase *Giro,* los objetos de las clases *TBevel* o *TImage* no son propiedades de la clase *Giro,* sino que la propiedad es un puntero a un objeto de

cada una de estas clases. Como se comenta en la introducción a UML, éste es el motivo de que la relación sea de agregación y no de composición, como demuestra el rombo hueco en el diagrama de clases.

Además la clase *Giro* tiene una relación de asociación con la clase *Contexto* porque llama a una operación de la propiedad mapa de dicha clase para obtener la información necesaria para la lista *nodo_destino*.

Sólo quedan por ver las clases de bases de datos que se usan en el interfaz de líneas de bus. La primera clase que verá es *Parada*.

La clase *Parada* se usa para almacenar información sobre la parada: su código, la línea a la que pertenece, el tramo o nodo en el que está situada, el tipo de parada e información de tipo geométrico: si está instalada en un hueco, su longitud, a que distancia del inicio del tramo está (caso de estar en un tramo) y una pequeña descripción. La propiedad *cod_parada* se usa como identificador único de la parada. La propiedad *cod_linea* sirve para identificar a la línea que pertenece la parada. La propiedad *tipo_parada* indica el tipo de parada de que se trata. La propiedad *codigo* indica el tramo o el nodo donde está la parada. La propiedad *codigo_sub* indica el código de movimiento o carril en el que se encuentra. La propiedad *posicion* indica la posicion en la que se encuentra la parada, medida desde el comienzo del tramo, si es que está emplazada en un tramo. La propiedad *longitud* indica la longitud de la parada. La propiedad *descripcion* es una descripción de la misma. La propiedad *tramo* indica si la parada tiene lugar en un tramo o en un nodo, y por lo tanto indica si la propiedad *codigo* es un codigo de un tramo o de un nodo. La propiedad *hueco* indica si existe un hueco en la calzada para la parada.

Como en la mayoría de las clases de este paquete, las operaciones sólo sirven para actualizar u obtener los valores de las distintas propiedades.

La clase *Parada* mantiene relaciones de dependencia con varias clases, en concreto con: *Datos_carril, Tramo* y *Punto.* La relación semántica entre estas clases es clara, ya que una parada estará situada sobre un tramo o nodo en uno de los carriles del tramo en el caso de encontrarse en uno de ellos.

La siguiente clase es *tramo_Parada*. La clase *tramo_Parada* almacena información sobre cada uno de los elementos de la red vial por los que pasa una cierta línea de autobús. En la propiedad *p*, de la clase *Parada*, se almacenan todos los datos relevantes sobre si es un tramo, un nodo, etcétera. La propiedad *para* indica si hay una parada en este tramo o nodo o simplemente la línea pasa por aquí. La propiedad *s_tramo* es un puntero al siguiente elemento de la lista *Lista_tramos_parada* por la que pasa la misma línea. La propiedad *a_tramo* es análoga pero el elemento anterior. Esto hace que si se desea borrar una parada de una línea no haya que modificar toda la estructura, simplemente marcar *para* como *false*. Las propiedades *cont_ant* y *cont_post* indican si se tiene conectividad con el elemento anterior o posterior de la línea de autobús: aunque los elementos estén enlazados en la lista mediante los punteros *s_tramo* y *a_tramo*, es posible que físicamente esos elementos de la red vial no estén unidos. Esta situación se da cuando el usuario borra un nodo intermedio de la línea. Sin embargo, se mantiene esa información por si el usuario quiere crear un camino alternativo que una los elementos actualmente desconectados y resturar así la línea. La propiedad *ultimo* indica si es el

ultimo de los elementos para una línea, que no de la lista, y la propiedad *primero* indica si el primer elemento de una línea, que no de la lista.

Los elementos de la red vial no son más que tramos o nodos. Además una de las propiedades de esta clase es un objeto de la clase *Parada*. De ahí la relación de composición con esta clase. La clase *Parada* contiene toda la información necesaria sobre las paradas. Esta estructura se aprovecha para almacenar los lugares por donde pasa de la siguiente manera. En primer lugar, si una línea de autobús tiene una parada en cierto tramo o nodo, no hay duda de que ese tramo o nodo pertenece a la lista de tramos o nodos por los que pasa la línea. En este caso la propiedad para de la clase *tramo_Parada* sería true.

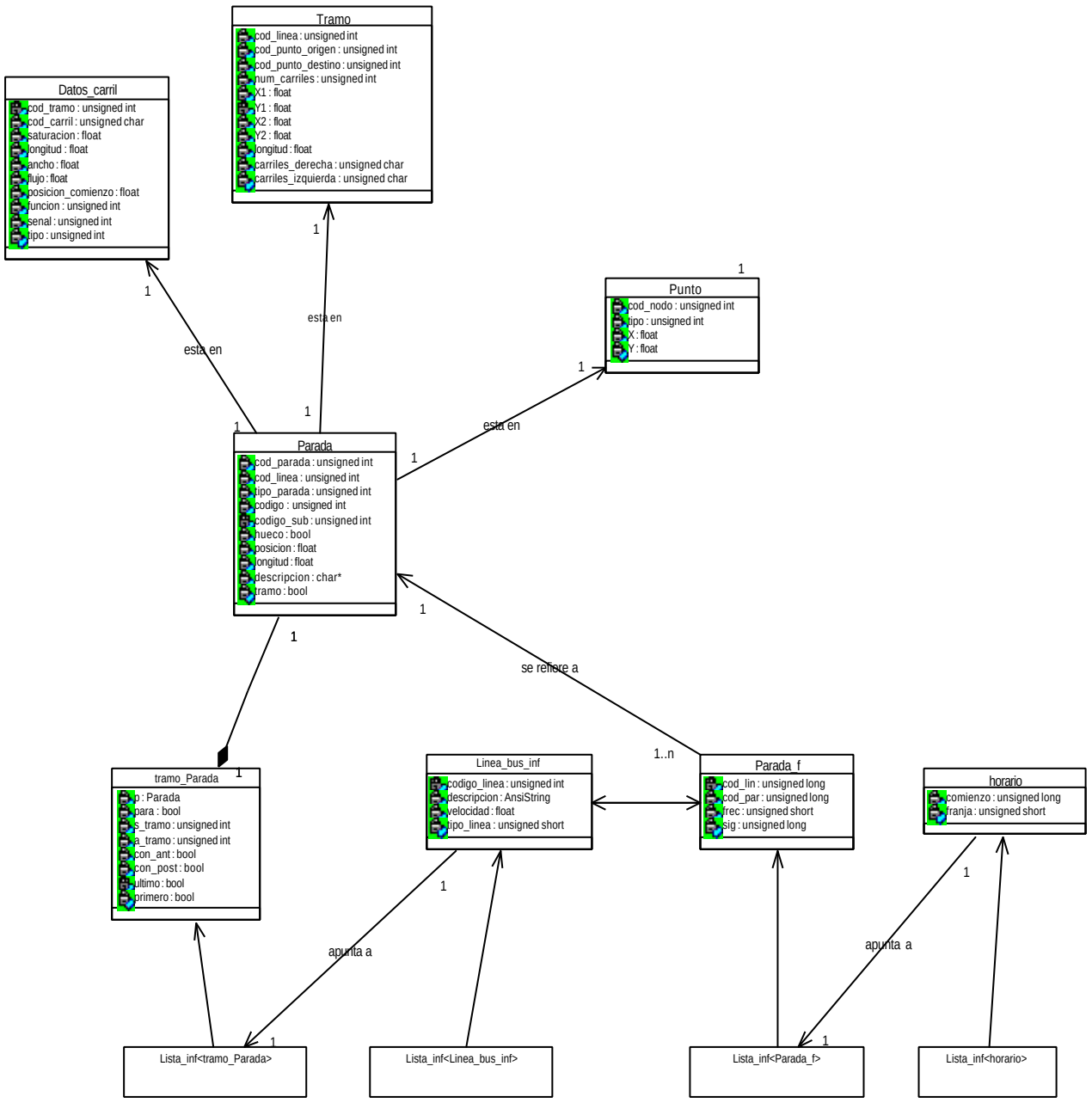

Figura 3.2.2.6: Clases de Bases de Datos (III)

En segundo lugar, si la línea de autobús pasa por un cierto tramo o nodo, pero no hay ninguna parada en el mismo, la información que contenga la propiedad de la clase *Parada* es indiferente en cuanto a lo que se refiere a los datos geométricos y descriptivos de la parada, pero son totalmente válidos los datos relativos al tramo o nodo en el que estaría situada. Simplemente la propiedad *para* de la clase *tramo_Parada* estaría a false.

Además la clase *tramo_Parada* tiene cuatro indicadores (*cont_ant, cont_post, primero* y *ultimo*) y dos "punteros" (*s_tramo* y *a_tramo*). Estos elementos se utilizan para gestionar la "sublista" que se mantiene para cada línea dentro de la lista de objetos de la clase *tramo_Parada*. Esta lista también está incluida en el diagrama de clases, como puede apreciarse.

La clase *Linea_bus_inf* mantiene información sobre las líneas de autobús existentes. Almacena información sobre la velocidad de la línea y el tipo de línea, así como una descripción de la línea. Pero también contiene la dirección del primer nodo de la línea de autobús dentro de la lista de objetos de la clase *tramo_Parada*. Esto facilita enormemente el proceso de búsqueda de los elementos (tramos y nodos) que pertenecen a la red vial: no hay que buscar, simplemente ir siguiendo los punteros.

La lista de tipo *Lista_inf* llamada *Lista_lineas* contiene objetos de la clase *Linea_bus_inf*. De esta manera, cada vez que se añade una línea nueva basta con añadir un elemento a esta lista. Y para borrarla se borran todos los elementos de dicha línea en *Lista_tramos_parada* (muy fácil siguiendo los punteros) y se borra el elemento correspondiente de *Lista_lineas.*

Esta clase tiene una relación de dependencia con la clase *Lista_inf<tramo_Parada>*, ya que contiene la posición de ciertos elementos dentro de esa lista.

La clase *Parada_f* almacena la frecuencia (atributo *frec*) con la que pasa un autobús de una cierta línea (atributo *cod_lin*) por una parada (atributo *cod_par*) de dicha línea. Existe una lista cuyos elementos son objetos de esta clase, donde se almacena toda la información. Una de las propiedades de esta clase (*sig*) es un puntero al elemento siguiente perteneciente a la misma franja horaria, de forma análoga a *tramo_Parada*.

Esta clase tiene una relación de dependencia con la clase *Linea_bus_inf*, ya que ambas almacenan datos, una directa y otra indirectamente, sobre las líneas de autobús.

Por último, la clase *horario* contiene, para una cierta franja horaria la posición dentro de la lista de objetos de la clase *Parada_f* del primer elemento que contiene información sobre dicha franja horaria. La propiedad *franja* indica la hora del dia a la que se refiere una cierta frecuencia. La propiedad *comienzo* contiene la posición en la lista *Paradas_frec* del primer elemento que guarda información sobre la frecuencia de paso de cierta línea y cierta parada en la frecuencia horaria indicada.

Los elementos de esta clase sirven para formar una lista de tipo *Lista_inf* llamada *Horarios.* En esta lista se almacenan las distintas franjas horarias existentes, y permite la búsqueda en la lista *Paradas_frec* de las frecuencias de las distintas líneas y paradas

para cada franja horaria, análogamente a la estructura que siguen las listas *Lista_tramos_parada* y *Lista_lineas*.

Como puede observarse, hay una gran analogía entre *Linea_bus_inf* y *tramo_Parada* y *horario* y *Parada_f*.

De nuevo la clase horario tiene una relación de dependencia con la clase *Lista_inf<Parada_f>*, ya que contiene un puntero a un elemento de dicha lista.

El paquete Representacion red vial

Bajo este paquete se han agrupado las clases cuya función es representar la red vial. Esta representación se puede llevar a cabo en dos escenarios distintos.

El primer escenario es cuando el usuario selecciona un directorio donde hay unos ficheros que contienen información sobre los nodos y tramos. La aplicación genera entonces una imagen de la red vial que ocupa gran parte de la pantalla. Sobre esta imagen el usuario puede realizar funciones de zoom y windowing, y también seleccionar un nodo o tramo para cargar el interfaz correspondiente mostrando la información del nodo seleccionado.

El segundo escenario tiene lugar cuando el usuario ha cargado un interfaz. Todos los interfaces incluyen dos imágenes en la parte derecha de la pantalla, una sobre otra. Sobre estas imágenes también se pueden realizar funciones de zoom y windowing, e incluso sobre una de ellas se pueden seleccionar nodos o tramos, según el interfaz del que se trate, para mostrar la información adecuada en el interfaz.

Las clases agrupadas bajo este paquete se encargan de estos dos tipos de representaciones.

En primer lugar hay varias clases, provenientes del paquete *Clases de la VCL de Borland*, que sirven para representar objetos gráficos en pantalla. Todos estos objetos son propiedades de alguna de las otras clases que se encargan de la representación gráfica, como *Contexto* e *Informacion_pantalla*. Por ello existe una relación de agregación entre aquellas clases y éstas.

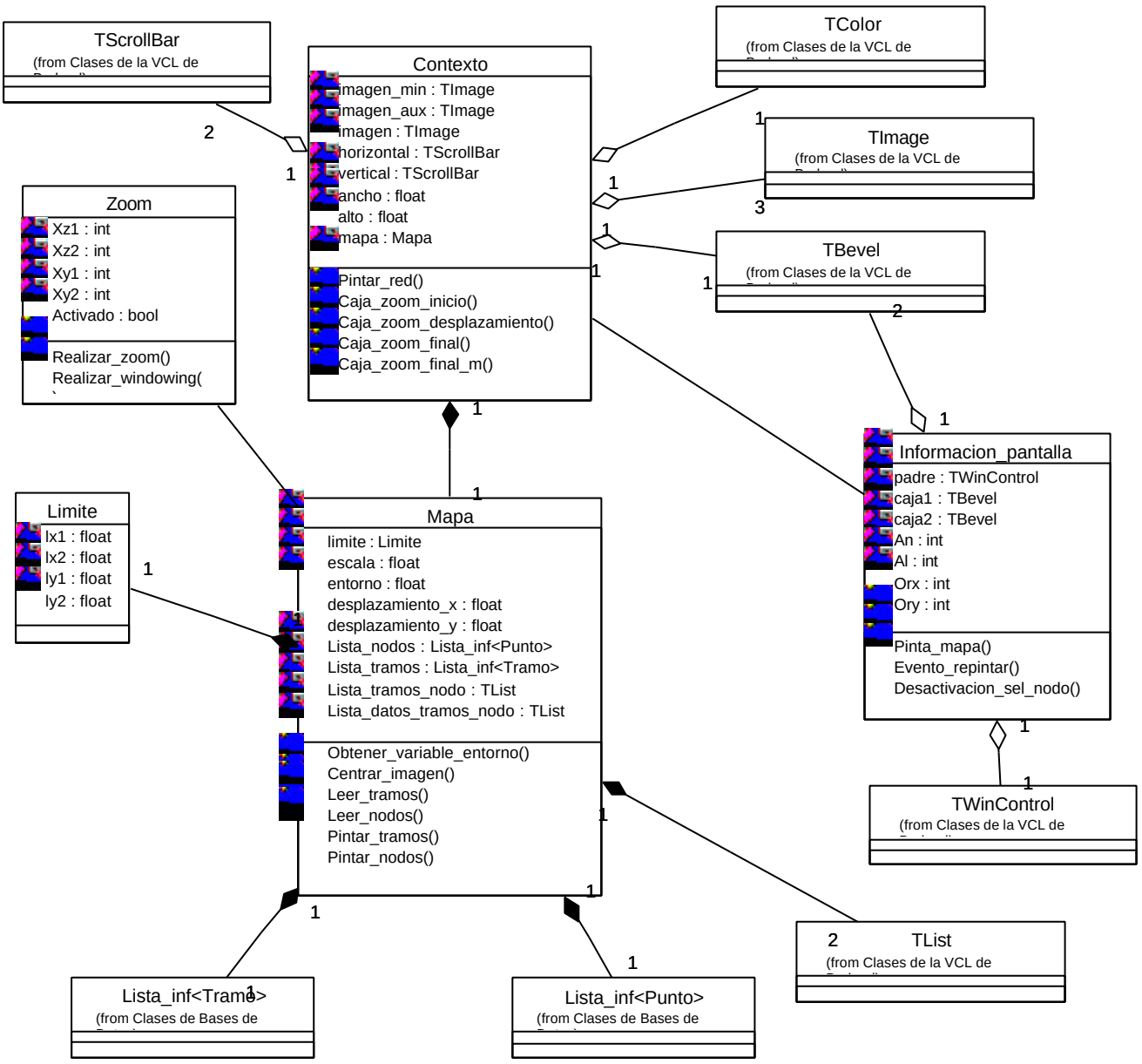

Figura 3.2.2.7: Representación red vial

La clase *Contexto* es la clase principal para la representación gráfica: contiene los objetos de la clase *TImage* sobre los que se representan las imágenes. Además, y lo más importante, es que contiene a la clase *Mapa*. La clase *Mapa* es la encargada de

almacenar toda la información relativa al entorno y a la porción de red vial a representar como resultado de sucesivas operaciones de zoom y windowing por parte del usuario.

La clase *Contexto* contiene la operación *Pintar_red*, que se encargar de representar gráficamente la red vial usando la información del entorno almacenada en la clase *Mapa*. La propiedad *imagen* es un puntero a un objeto de la *TImage*, es decir, una imagen. La imagen a la que apunte este puntero será la imagen principal, es decir, en la que se representa la red vial después de cargar la información. La propiedad *imagen_min* se usará por la clase *Informacion_pantalla* en la representación de los interfaces de tramos, nodos y líneas de bus. La propiedad pública *mapa* es un objeto de la clase *Mapa* que almacenará toda la información necesaria para que *Contexto* pueda representar la imagen.

Por supuesto la relación entre la clase *Mapa* y *Contexto* es una composición. De hecho, cada objeto de la clase *Contexto* que se crease debería contener (como es el caso) su propio objeto de la clase *Mapa*, pues es absolutamente necesario para representar la imagen y depende de las operaciones que se hayan hecho sobre el objeto de la clase *Contexto* en concreto.

Las demás operaciones de la clase *Contexto* sirven para el zoom y el windowing.

En realidad la clase *Contexto* contiene muchísimas operaciones, pero muchas de ellas sirven para buscar un nodo o un tramo en la lista correspondiente. Esto es debido a que en los interfaces cada uno de estos elementos se selecciona sobre una de las imágenes, de manera que es necesario acudir a la clase *Contexto* para traducir la posición que seleccionó el usuario a coordenadas globales y poder buscar a continuación el susodicho nodo o tramo en la lista correspondiente, que será una de las propiedades de la propiedad correspondiente de la clase *Mapa*.

La clase *Mapa* se encarga de almacenar la información sobre el entorno pero además contiene una serie de listas en las que almacena información sobre los nodos y los tramos que componen la red vial. Esta información es principalmente de tipo geométrico. Cualquier información de distinta índole sobre nodos o tramos se almacena en la clase correspondiente dentro del paquete *Clases de Bases de Datos*.

La propiedad *entorno* almacena la proporción entre el tamaño de la porción de red vial a representar y el tamaño de la imagen donde debe representarse. La propiedad *limite* almacena la porción de red vial que debe representarse. Las propiedades *desplazamiento_x* y *desplazamiento_y* añaden un desplazamiento a las coordenadas de la red vial para que ésta sea representada centrada en la imagen.

La operación *Obtener_variable_entorno* calcula el valor de la propiedad *entorno*. La operación *centrar_imagen* calcula el valor de los desplazamientos *desplazamiento_x* y *desplazamiento_y*. Las operaciones *Leer_tramos* y *Leer_nodos* reconstruyen las listas de tramos y nodos desde la información almacenada en un fichero.

Las operaciones *Pintar_tramos* y *Pintar_nodos* son realmente las que representan la red vial, son llamadas desde la operación *Pintar_red* de la clase *Contexto.*

La clase *Limite* almacena las coordenadas del rectángulo, en coordenadas globales, tal que los tramos y nodos que hay contenidos en su interior son los que deben mostrarse en la imagen. Las propiedades de esta clase son las coordenadas de los puntos que limitan el rectángulo: *lx1* y *ly1* son las coordenadas de la esquina superior izquierda de dicho rectángulo y *lx2* y *ly2* las coordenadas de la esquina inferior derecha.

La clase *Zoom* almacena las coordenadas del rectángulo que usa el usuario de la aplicación para seleccionar la zona sobre la que quiere realizar una operación de zoom o windowing. Además proporciona las operaciones *Realizar_zoom* y *Realizar_windowing*, que son las encargadas de realizar el zoom y el windowing.

Estas operaciones necesitan de la colaboración de operaciones y propiedades de la clase *Mapa*, por ejemplo, para representar gráficamente la red vial. Por ello hay una relación de asociación entre las clases *Mapa* y *Zoom*.

Estrictamente hablando, la clase *Zoom* tendría una asociación con *Contexto*, pues usa las operaciones de una de sus propiedades (el objeto *mapa* de la clase *Mapa*). Lo que ocurre es que se entiende mejor el funcionamiento si se realiza la asociación directamente con la clase *Mapa*.

Normalmente, si una clase tiene una relación de asociación con otra clase y con otra más que es una parte de la anterior (relación de composición o agregación), se representa sólo la relación de asociación con la clase que representa el todo, y no con la que representa una parte, para no incluir excesivas relaciones en el diagrama. El caso anterior es una excepción que se ha hecho debido a las peculiares características de las clases *Mapa* y *Zoom*.

Finalmente, la clase *Informacion_pantalla* es la que se encarga de representar las dos imágenes en cada uno de los interfaces. Para ello usa un objeto de la clase *Contexto* y modifica las propiedades de varias de las propiedades de este objeto, entre ellas las imágenes. De ahí la asociación que existe entre las clases *Informacion_pantalla* y *Contexto.*

Esta clase se encarga de crear las dos imágenes que aparecen en los interfaces en la parte derecha del mismo. La imagen superior contiene una visión general de la red vial y la inferior representa la porción de red vial seleccionada en ese instante. Así mismo, en la imagen superior aparece representada por un rectángulo de color y con la impresión de la red en color inverso, la porción de red vial seleccionada y que se está representando en la imagen inferior.

Una vez las imágenes han sido creadas adecuadamente, el control sobre ellas recae sobre el objeto de la clase *Contexto* de la que son una propiedad.

El objeto *padre* de la clase *TWinControl* se usa para almacenar el objeto gráfico sobre el que se crearán las dos imágenes. Es el padre de las imágenes.

Los dos objetos *caja1* y *caja2* de la clase *TBevel* se usan para crear un pequeño pedestal sobre el que se dibujan las imágenes.

Las propiedades *An, Al, Orx* y *Ory* se usan para almacenar las propiedades geométricas de la imagen antes de llamar al método para luego poder devolverla a su estado inicial. Esto hace que el proceso sea totalmente transparente al usuario.

La operación más importante es *Pinta_mapa*. Se encarga de representar los dos pedestales que contendrán las imágenes en el formulario que se le pasa como parámetro. Por supuesto lo hace usando el método de división del formulario en porcentajes.

A continuación usa operaciones de la clase *Contexto* para crear un par de objetos de la clase *TImage*, si bien al primero de ellos se le asigna a la dirección de la imagen la propiedad *imagen* del objeto *contexto_giro* y el segundo asigna la dirección de la imagen creada a la propiedad *imagen_aux* del mismo objeto. La imagen *imagen_aux* no se ve en pantalla. La propiedad *imagen* del objeto *contexto_giro* es la imagen superior de las dos.

Esta operación usa la propiedad *imagen* del objeto *contexto* para representar la imagen inferior. Esta propiedad era la que contenía anteriormente la imagen donde se representaba toda la red vial.

También llama a las operaciones adecuadas de la propiedad mapa del objeto *contexto_giro* para calcular las variables de entorno para las nuevas imágenes.

A continuación hace que el objeto padre de la propiedad *imagen* del objeto *contexto* sea el formulario del interfaz en el que está actualmente el usuario (interfaz de nodos, de tramos o de líneas de bus). Seguidamente, modifica las propiedades geométricas de esta imagen para que se represente en el lugar que aparece en el formulario, es siempre la inferior. Esto quiere decir que la imagen inferior es la propiedad *imagen* del objeto *contexto.* El cambio de dimensiones se consigue llamando a la operación *Actualizacion_imagen* de la clase *Contexto.*

Seguidamente recalcula las variables de entorno para el objeto *contexto* usando las dimensiones de la nueva imagen.

A continuación se representa la red vial en la imagen *imagen* del objeto *contexto_giro*. Esto hace que se muestre toda la red vial en la imagen superior.

Sin embargo, en la imagen inferior se usa el objeto *contexto* que hasta ahora había estado representando la red vial cargada. Eso quiere decir que los límites de la imagen inferior son los mismos que los que tenía la red vial antes de cargar el interfaz correspondiente. Por eso la imagen inferior representa la porción de red vial que el usuario tenía seleccionada previamente.

El paquete Interfaz tramos

La clase principal de este interfaz es *Tramo_interactivo.* Esta clase es la que se encarga de construir todo el interfaz gráficamente, aunque para ello se relaciona con otras clases.

Este interfaz es también demasiado grande como para representarlo de una sola vez, por lo que se incluyen varias figuras.

En primer lugar necesita a la clase *Informacion_pantalla* para representar gráficamente las dos imágenes del interfaz. Además necesita a la clase *Contexto* porque usará algunas de sus operaciones, como por ejemplo algunos manejadores de eventos.

Esta relación se modela como una relación de la clase *Tramo_interactivo* con la clase *Informacion_pantalla* y otra con la clase *Contexto.*

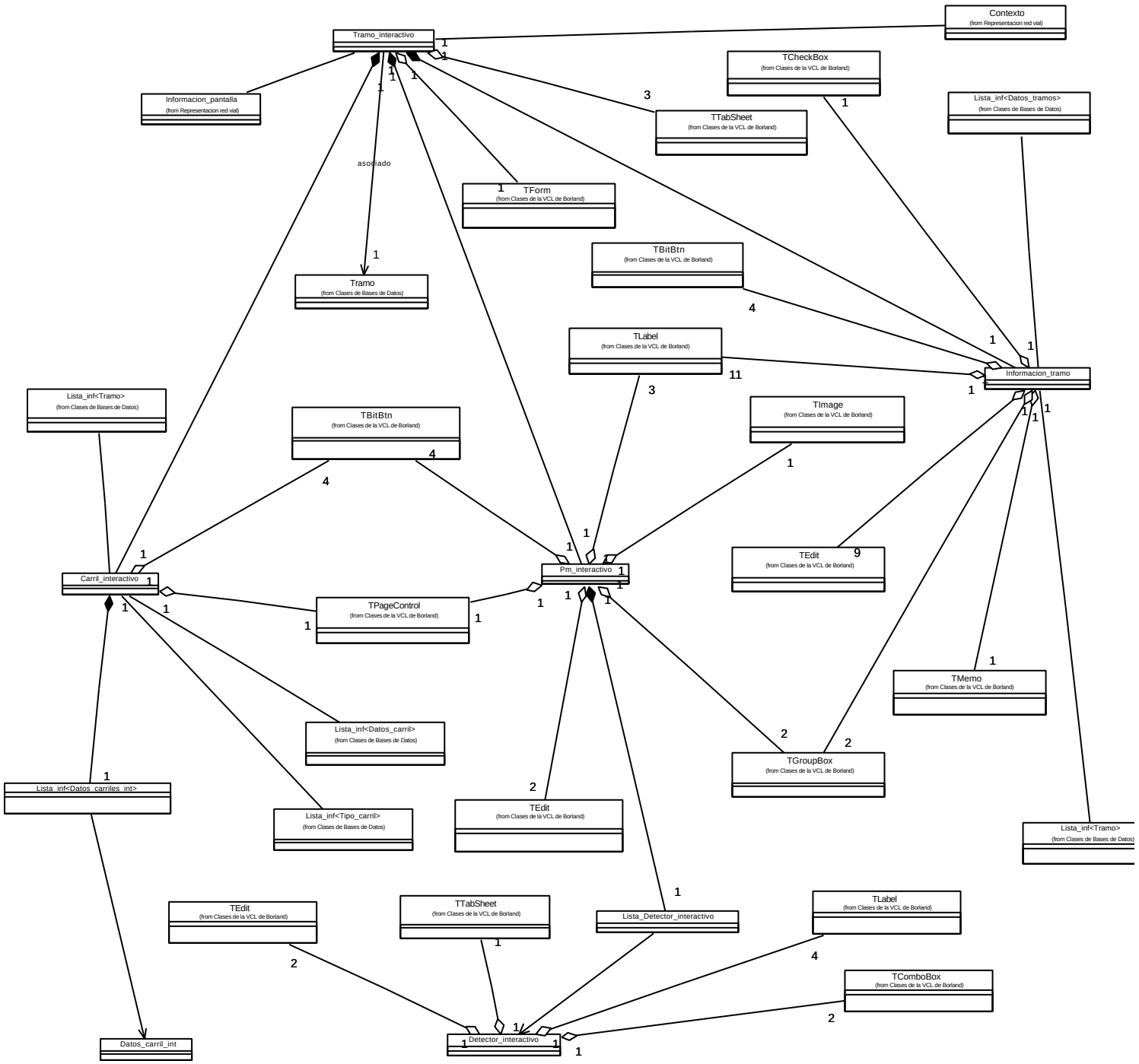

Figura 3.2.2.8: Interfaz de tramos

En este paquete se agrupan las clases cuyo fin es la representación gráfica del interfaz de tramos. Evidentemente, se usan muchas clases definidas en el paquete *Clases de la VCL de Borland*. Todas estas clases son propiedades de otras definidas en este paquete y las relaciones entre estos dos tipos de clases, como ya se ha explicado, son siempre de agregación. Este comportamiento se reproducirá análogamente en los demás paquetes de interfaces que quedan por ver, por lo que este hecho no volverá a mencionarse.

Incluso en algún paquete estas clases ya ni siquiera se han representado en el diagrama de clases correspondientes, debido a que una vez entendida su funcionalidad y relaciones con las otras clases, su presencia sólo contribuye a dificultar la comprensión del diagrama correspondiente.

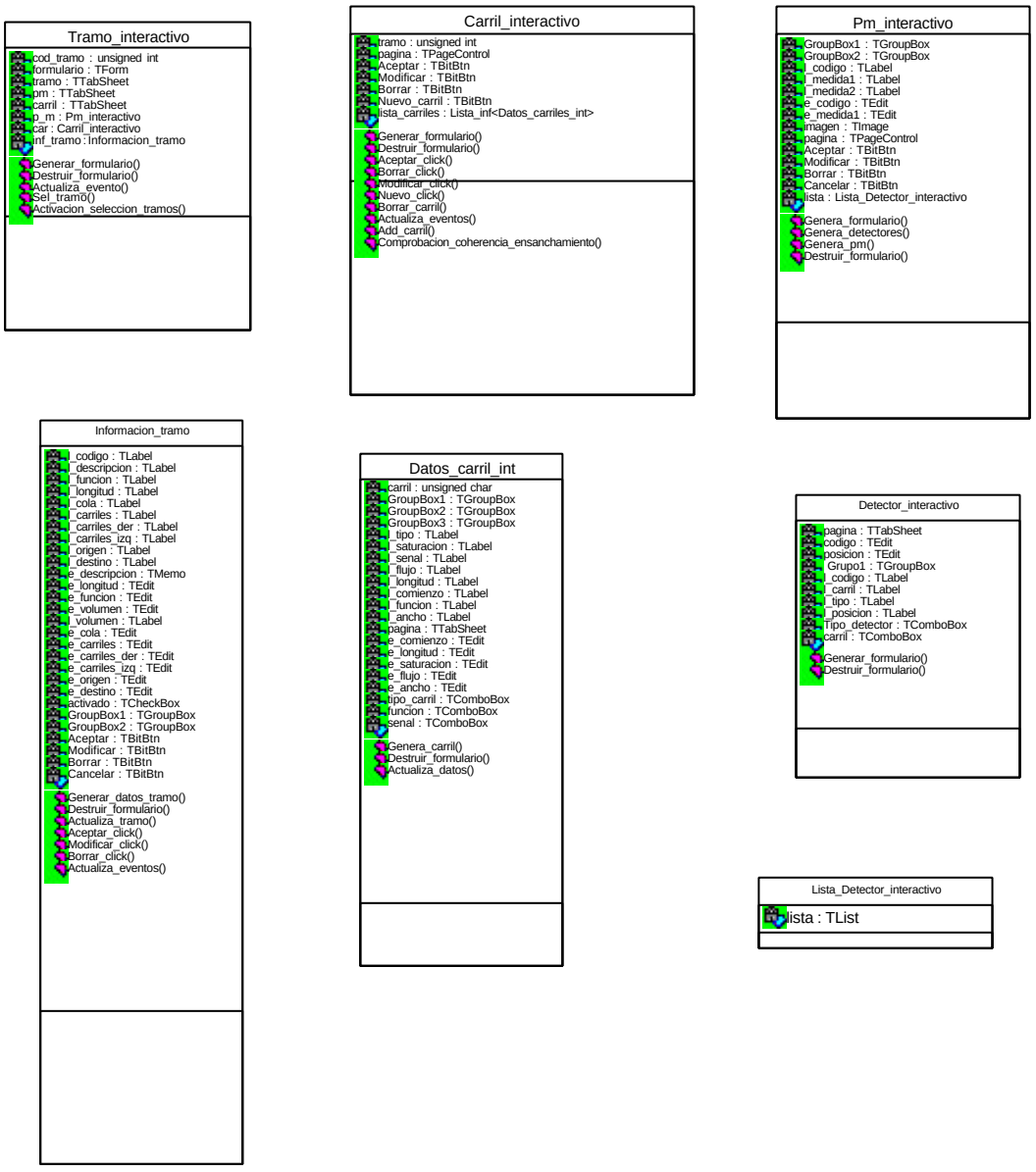

Figura 3.2.2.9: Clases del interfaz de tramos

La clase *Tramo_interactivo,* aparte de los objetos necesarios para representar gráficamente interfaz, contiene una propiedad que almacena el código del tramo del cual se muestra información, el tramo seleccionado.

Además contiene tres objetos de clases definidas en este mismo paquete, *Pm_interactivo, Carril_interactivo* e *Informacion_tramo*.

Aunque se tratarán a continuación, se puede adelantar que la clase *Tramo_interactivo* representa gráficamente los objetos comunes del interfaz de tramos y el controlador de páginas, mientras que los demás objetos se encargan cada uno de representar una página.

Está claro que la relación entre cada una de las clases *Carril_interactivo, Pm_interactivo* e *Informacion_tramo* con la clase *Tramo_interactivo* son de composición, siendo en todos los casos la clase *Tramo_interactivo* la que representa el todo y las demás una de las partes.

Las operaciones definidas para la clase *Tramo_interactivo,* por ejemplo, *Generar_formulario*, usan operaciones de las clases *Carril_interactivo, Pm_interactivo* e *Informacion_tramo*. Sin embargo, no es necesario añadir una relación de asociación entre estas clases y la clase *Tramo_interactivo,* ya que la relación existente, composición, ya incluye este comportamiento: cada una de estas clases son una parte de la clase *Tramo_interactivo,* y las operaciones que usan los objetos de la clase *Tramo_interactivo* siempre son las de los objetos que son sus propiedades, nunca de un objeto *Carril_interactivo, Pm_interactivo* o *Informacion_tramo* que no sea una de las propiedades de la clase *Tramo_interactivo.*

La clase *Tramo_interactivo* tiene además una relación de dependencia con la clase *Tramo*, debido a que toda la información que muestra es relativa a un tramo.

La clase *Carril_interactivo* tiene como función representar gráficamente la página de carriles dentro del interfaz de tramos.

Pero si se recuerda la página de carriles, en ella se presentan los datos de cada uno de los carriles dentro de una subpágina dentro de la página de carriles. Cada una de estas subpáginas lleva el número de carril al que se refiere dentro del tramo: Carril 0, Carril 1, etcétera.

Por ello, la clase *Carril_interactivo* tiene una propiedad que es un controlador de página, necesario siempre que se desee tener varias páginas.

También crea los objetos gráficos comunes a todos los carriles, como los botones de *Aceptar, Modificar, Añadir* y *Borrar.*

Las operaciones que incluye son las necesarias para crear y destruir el formulario, es decir, *Generar_formulario* y *Destruir_formulario*.

El resto de operaciones que incluye son los manejadores de eventos para cada uno de los botones.

Para poder acceder a toda la información que esta clase necesita, mantiene relaciones de asociación con varias listas del paquete de *Clases de Bases de Datos*, en concreto, con *Lista_inf<Tramo>*, *Lista_inf<Datos_carril>* y *Lista_inf<Tipo_carril>*. La información almacenada en cada una de estas listas ya se detalló en la explicación del paquete correspondiente.

Finalmente, la clase *Carril_interactivo* mantiene una lista de objetos del tipo *Datos_carril_int*, es decir, *Lista_inf<Datos_carril_int>*. Cada uno de los elementos de esta lista es una de las subpáginas correspondientes a cada uno de los carriles dentro del tramo.

La relación de composición entre la clase *Carril_interactivo* y la *Lista_inf<Datos_carril_int>* es clara. Ésta última mantiene una relación de dependencia con la clase *Datos_carril_int*, ya que son objetos de esta clase los que forman la lista. Esta relación de dependencia entre la lista y la clase que se le pasa como parámetro se da, por supuesto, en todas las listas de tipo *Lista_inf*.

La clase *Datos_carril_int* se encarga de representar todo el interfaz gráfico para cada una de las subpáginas que representa uno de los carriles del tramo. Por ello todas sus propiedades son objetos gráficos.

Dos operaciones que tiene son para crear y destruir la subpágina del carril correspondiente, *Genera_carril* y *Destruir_formulario*. La tercera operación, *Actualiza_datos*, sirve para presentar en la subpágina los datos relativos al carril.

A continuación se trata la segunda de las clases que mantiene relación de composición con la principal, *Tramo_interactivo.* Esta clase es *Pm_interactivo.*

La clase *Pm_interactivo* almacena información sobre los puntos de medida de la red vial. Presenta una estructura similar a *Carril_interactivo,* aunque no idéntica. En la página correspondiente a puntos de medida dentro del interfaz de tramos se presenta la información sobre el punto de medida seleccionado, pero además se presenta información sobre los distintos detectores. De esta manera, hay una serie de páginas dentro de la página de puntos de medida (subpáginas) que muestran la información de cada uno de los detectores.

Es por ello que la clase *Pm_interactivo* tiene una serie de propiedades que son objetos gráficos para representar las partes comunes de la página: la información relativa al punto de medida, los botones y el controlador de páginas para las subpáginas de detectores.

Los detectores se almacenan en una clase llamada *Lista_Detector_interactivo* que no es más que una lista de tipo *TList* de objetos de la clase *Detector_interactivo.*

Por supuesto la clase *Lista_Detector_interactivo* tiene una relación de composición con la clase *Pm_interactivo* en la que es la parte y *Pm_interactivo* el todo.

Pero la clase *Lista_Detector_interactivo* también tiene una relación de dependencia con la clase *Detector_interactivo,* pues son los objetos que formarán la lista.

La clase *Detector_interactivo* tiene una serie de objetos gráficos con los que crea cada subpágina de detectores y las operaciones *Generar_formulario* y *Destruir_formulario* para crear las subpáginas y destruirlas.

Por último, la otra clase que mantiene una relación de composición con *Tramo_interactivo* (en la que ella es la parte y Tramo_interactivo el todo) es *Informacion_tramo*.

Esta clase se encarga de representar la página de tramos dentro del interfaz de tramos. En este caso la clase genera el interfaz completamente, no hay subpáginas como en los casos anteriores. Tiene una gran cantidad de objetos gráficos para poder realizar esta función.

Además contiene las operaciones necesarias para crear y destruir el formulario, *Generar_datos_tramo* y *Destruir_formulario*. También contiene las operaciones necesarias para manejar los distintos eventos que puede desencadenar el usuario.

Además mantiene relaciones de asociación con las clases de bases de datos *Lista_inf<Datos_tramos>* y *Lista_inf<Tramos>*, puesto que accede a estas clases para obtener y modificar información sobre los tramos.

El paquete Interfaz de nodos

Como su nombre indica, bajo este nombre se encuentran agrupadas las clases cuya función es representar gráficamente en el interfaz de nodos.

La clase principal dentro de este paquete es *Punto_interactivo.* Esta clase contiene los objetos gráficos necesarios para representar gráficamente las partes comunes del interfaz. En este caso es poco más que el controlador de páginas. El controlador de páginas es necesario porque de nuevo en este interfaz van a aparecer varias páginas. Una de ellas para los nodos y otra para los giros.

Esta clase tiene dos propiedades que son objetos de dos clases que también se definen aquí, análogamente al otro interfaz. Cada una de estas clases se encargará de representar gráficamente la página de nodos (la clase *Nodo_interactivo*) y la de giros (la clase *Giro_interactivo*).

Por supuesto habrá de nuevo una relación de composición entre las clases *Nodo_interactivo* y *Punto_interactivo* y entre *Giro_interactivo* y *Punto_interactivo.* En estas dos relaciones, claro está, el todo es la clase *Punto_interactivo.*

Las operaciones de la clase *Punto_interactivo* tienen como función crear y destruir el formulario y las páginas (esto último llamando a las operaciones adecuadas de *Nodo_interactivo* y *Giro_interactivo*). También se encarga de asignar las operaciones adecuadas a cada uno de los eventos.

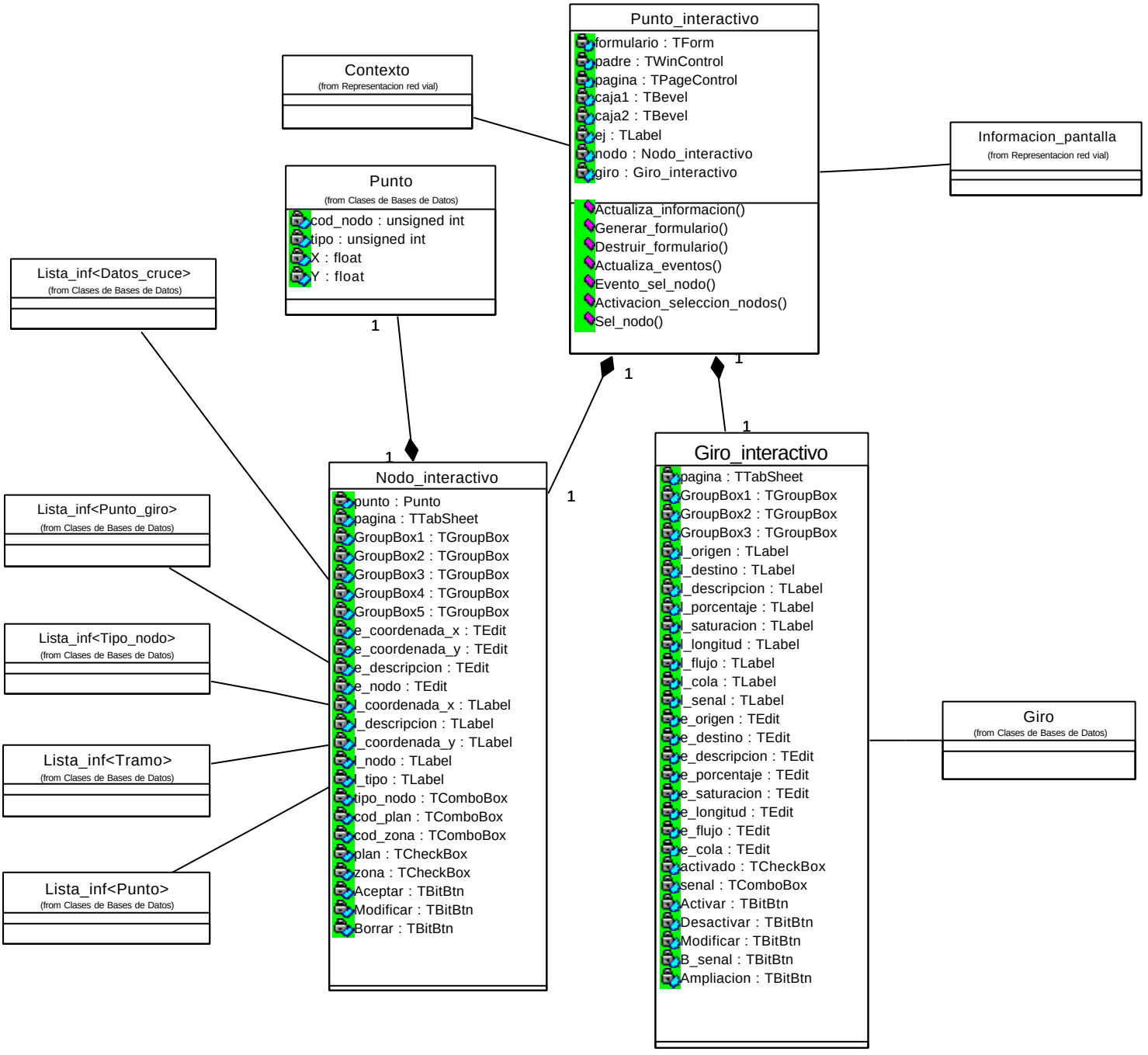

Figura 3.2.2.10: Interfaz de nodos

Algunos de estos eventos están asignados a operaciones de la clase *Contexto,* por lo que debe existir una relación de asociación entre la clase *Contexto* y *Punto_interactivo.* También existe una asociación entre la clase *Punto_interactivo* e *Informacion_pantalla* porque al crear el interfaz debe crear también las dos imágenes de la derecha del interfaz, y eso lo hace con operaciones de la clase *Informacion_pantalla*.

A continuación se van a ver cada una de las clases que tienen como función crear cada una de las páginas, la de nodos y la de giros. Como ya se ha dicho, estas clases tienen relación de composición con la clase *Punto_interactivo.*

En realidad, una de las clases, *Giro_interactivo,* necesita de la clase *Giro* para representar el interfaz, no lo hace ella sola. Ya se dijo que la clase *Giro* tenía también algunas operaciones típicas de las clases que tienen como fin la representación gráfica. De todas maneras, esto se verá cuando le toque el turno a esta clase.

La clase *Nodo_interactivo* tiene como función representar la página de nodos dentro del interfaz de nodos.

Esta clase contiene todos los objetos gráficos necesarios para crear la página. La única propiedad que no es un objeto para representación gráfica es el de la clase *Punto,* que tiene como función almacenar el nodo seleccionado actualmente.

Entre sus operaciones más importantes están *Informacion_nodo*, que lee los datos de la lista *Lista_datos_cruce* para el nodo seleccionado y los presenta en el formulario. Las operaciones *Generar_datos_nodo* y *Destruir_formulario* tienen como función crear y destruir el formulario, en este caso la página de nodos. La creación del formulario incluye también la asignación de los eventos a las operaciones adecuadas.

La clase *Punto* tiene una relación de composición con esta clase, ya que uno de los atributos de esta clase es de la clase *Punto.*

Las relaciones de asociación con las clases *Lista_inf<Datos_cruce>* y *Lista_inf<Tipo_nodo>* son lógicas: la primera de ellas por que es la lista de la que lee los datos. La segunda de ellas porque es la lista que contiene los distintos tipos de nodos que pueden existir y estos tipos deben aparecer como opción en el formulario, por lo que la clase *Nodo_interactivo* debe usar operaciones de esta clase para acceder a la información.

La relación menos evidente es la asociación que existe con la clase *Lista_inf<Punto_giro>*, que se recuerda se usaba para almacenar los giros permitidos. En esta ocasión tiene un uso distinto, se crea una la lista de este tipo para almacenar todos los tramos cuyo nodo origen o destino se desea eliminar.

La relación de asociación con la lista *Lista_inf<Punto>* es clara porque debe acceder a la información que aparece en esa lista para modificarla. La relación con la lista *Lista_inf<Tramo>* viene dada por el hecho de que cada tramo se define en base a sus nodos origen y destino. Se usa la lista de la clase *Lista_inf<Punto_giro>* para llevar la cuenta de todos los tramos que se deben borrar en caso de borrar un nodo: hay que borrar todos los tramos cuyo nodo origen o destino sea el nodo a borrar.

La clase que queda por ver es la que usa para crear la página de giros conjuntamente con la clase *Giro*. Esta clase tenía una relación de composición con la clase *Punto_interactivo* (*Punto_interactivo* era el todo) y es la clase *Giro_interactivo.*

La función principal de la clase *Giro_interactivo* es representar gráficamente la página correspondiente a giros. De hecho, representa todo el interfaz excepto la imagen donde se representan los giros posibles y seleccionados.

Esta imagen es una propiedad de la clase *Giro*.

Las principales operaciones de la clase *Giro_interactivo* tienen como función crear y destruir el formulario (*Generar_datos_giros* y *Destruir_formulario*). Los datos que necesita los lee del objeto de la clase *Giro.*

También está entre las operaciones de la clase *Giro_interactivo* el asignar correctamente los eventos a las operaciones adecuadas, pero en este caso las operaciones que se encargan de manejar los eventos en la mayoría de los casos son operaciones de la clase *Giro*, a diferencia de lo que suele ocurrir en los otros interfaces.

De aquí surge la relación de asociación (que en este caso es muy estrecha) entre la clase *Giro_interactivo* y la clase *Giro.*

El paquete interfaz de líneas de bus

Este paquete se ocupa de la representación gráfica del interfaz de líneas de bus.

La clase principal de este interfaz es la clase *Linea_bus_interactivo.* Como es habitual, el interfaz se divide en dos páginas, aunque también es cierto que existen una serie de objetos comunes a las dos.

Como es de esperar, esta clase se encarga de crear los objetos gráficos comunes a las dos páginas y el controlador de páginas. Los objetos comunes a las dos páginas son, básicamente, imágenes. Dos imágenes son las ya habituales que representan la red vial, y otra es una imagen que muestra los tramos y nodos por donde pasa la línea y las diferentes paradas. Esta última imagen es la propiedad *Grafico_linea*.

La clase *Linea_bus_interactivo* tiene las operaciones normales para crear y destruir el interfaz. Éste se construye con la operación *Generar_datos_linea*. También tiene las operaciones necesarias para asignar los eventos a las operaciones adecuadas.

Esta clase tiene las asociaciones ya habituales con *Contexto* e *Informacion_pantalla*.

Pero además tiene dos objetos, *parada* y *datos*, de las clases *Parada_interactiva* y *Datos_linea*, respectivamente. Cada uno de estos objetos se usa para representar la página correspondiente a líneas y la página correspondiente a paradas dentro del interfaz de líneas de bus.

Por lo tanto, las clases *Parada_interactiva* y *Datos_linea* tienen una relación de composición con la clase *Linea_bus_interactivo.* En ambos casos la clase *Linea_bus_interactivo* es el todo.

La clase *Linea_bus_interactivo* tiene una relación de asociación con las clases siguientes: *Lista_inf<tramo_Parada>*, *Lista_inf<Linea_bus_inf>*, *Lista_inf<Parada_f>* y *Lista_inf<horario>*, ya que se encarga de crear todas estas listas. Además son las listas de las que obtienen la información necesaria las clases *Parada_interactiva* y *Datos_linea*. Aunque de nuevo se repite que podía haberse representado explícitamente la asociación entre estas clases y las listas, se decidió no hacerlo así porque los únicos objetos de las clases *Parada_interactiva* y *Datos_linea* que acceden a las listas son los que son atributos de *Linea_bus_interactivo.*

La clase *Datos_linea* se encarga de representar gráficamente la página de líneas de autobús. Sus principales operaciones son para crear y destruir la página: *Generar_datos_linea* y *Destruir_formulario.*

La clase *Parada_interactiva* se encarga de representar gráficamente la página de paradas de autobús. Sus principales operaciones son para crear y destruir la página: *Generar_datos_parada* y *Destruir_formulario*.

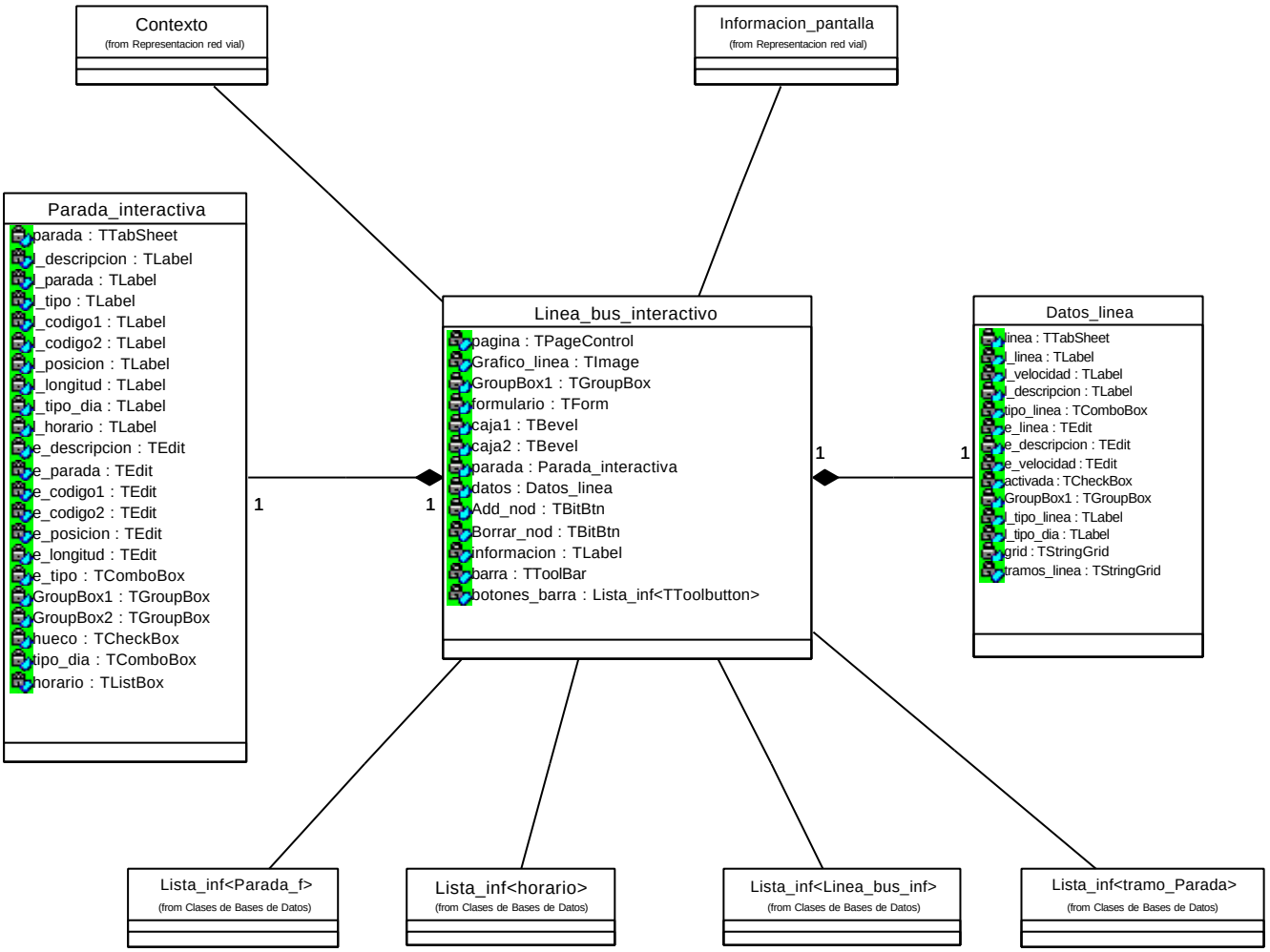

Figura 3.2.2.11: Interfaz de líneas de bus