1 INTRODUCCIÓN

En este documento vamos a describir las estructuras de datos y los paquetes que nos van a servir para implementar todo el sistema.

El proyecto está formado por un gran número de ficheros, que podemos agrupar según su finalidad en tres grupos. Por un lado están los ficheros que permiten al ordenador escribir en el puerto para manejar las salidas del autómata, otros son los ficheros del programa de control de la maqueta y el último grupo son los ficheros que implementan el interfaz gráfico.

Comenzaremos describiendo los ficheros que nos permiten controlar el puerto serie del ordenador, para guardar la antigua configuración, escribir en el puerto y restaurar la configuración original.

A continuación describiremos los ficheros que implementan el interfaz gráfico, ficheros creados por la herramienta glade.

Y finalmente describiremos los tipos de datos utilizados en el proyecto y después los paquetes que implementan los diferentes objetos.

2 PROGRAMACIÓN DEL PUERTO SERIE

Para trabajar con el puerto serie utilizamos la capacidad del lenguaje ADA de incorporar funciones implementadas en otro lenguaje. Los archivos que permiten trabajar con el puerto son driver.c, driver.h, pserie.ads y pserie.adb.

2.1 DRIVER.

Los ficheros driver.h y driver.c contienen funciones implementadas en lenguaje C, que lo que hacen es llamar a las funciones del propio sistema operativo para escribir en el puerto serie. Además en ellos se configura dicho puerto y se restablece la antigua configuración después de haber trabajado con él.

En driver.h se establece el puerto serie que vamos a utilizar:

```
/* Definicion del puerto serie correcto  */
#define MODEMDEVICE "/dev/ttyS0"
```

Estas son las funciones implementadas en el fichero driver.c:

```
int c_init (); // Inicializa el puerto serie
int c_escribir (char c);  // Escribe en el puerto serie
void c_fin(); // Restablece la antigua configuración del puerto
```

- La función c_init se encarga de inicializar el puerto serie para poder escribir en él y almacena los valores de la configuración del puerto para poder restaurarla después.
- La función c_escribir llama a la función del sistema operativo para escribir en el puerto el dato c de tipo carácter. En case de error devuelve el valor -1.
- La función c_fin se encarga de restaurar la antigua configuración del puerto para dejarlo tal y como estaba antes de usarlo.

2.2 PSERIE.

El paquete Pserie implementa en Ada las funciones que usamos para configurar y escribir en el puerto serie. En realidad lo que hacen estas funciones es usar la capacidad de interfaz de Ada con otros lenguajes, para llamar a las funciones implementadas en C en el fichero driver.c.

Las funciones implementadas en pserie son:

```
package Pserie is
    function init_com return integer;
        -- Inicializa el puerto serie para poder enviar datos
    function escribir (A,B,C,D,E,F :in byte) return integer;
        -- Esta función escribe en el puerto serie los bytes que recibe
```

```
procedure fin_com;

    -- Finaliza la comunicación y deja la configuración del puerto serie como estaba antes

end Pserie;
```

- La función init_com llama a la función c_init para inicializar el puerto y almacenar la antigua configuración, en caso de error devuelve -1.
- La función escribir escribe en el puerto los seis bytes que recibe. En caso de error devuelve -1. Byte es un tipo de dato definido en el proyecto. Estos seis bytes mapearán las salidas del autómata.
- El procedimiento fin_com llamará a la función c_fin para cerrar el puerto y restaurar su antigua configuración.

La función escribir será muy utilizada por el programa de control, ya que cada vez que quiera modificar las salidas del autómata será la función que llamará pasándole codificados en esos seis bytes que recibe los valores de cada salida del autómata. La función se encargará de escribir uno tras otro, los seis bytes en el orden en que le son suministrados.

Para que otras funciones puedan llamar a ésta, los paquetes de todas las funciones que quieran modificar las salidas del autómata deben incorporar la cláusula:

with pserie;

Así las funciones que lo necesiten podrán modificar las salidas del autómata suministrándole seis nuevos bytes.

3 INTERFAZ GRÁFICO.

Los archivos que implementan el interfaz gráfico fueron generados de forma automática por la herramienta Glade tras diseñar la interfaz. Los ficheros generados son window1.adb y los paquetes window1_pkg y window1_pkg-callbacks.

El fichero window1.adb se encarga de arrancar el interfaz y además inicializa la maqueta y comienza y finaliza las tareas de controlar y detectar los trenes. Al fichero original generado por glade se le ha añadido código para inicializar la maqueta, para que al arrancar el interfaz gráfico se pueda operar con la maqueta, y para que al cerrar el interfaz desconecte las salidas del autómata y la maqueta quede en reposo.

El paquete window1_pkg implementa el diseño del interfaz, en él se crean todas las ventanas, botones, etiquetas, etc., que constituyen el interfaz gráfico. Este fichero no ha sido modificado ya que únicamente implementa el aspecto del interfaz.

En el paquete window1_pkg-callbacks se implementan los procedimientos que capturan la activación de los botones del interfaz y los asocian con las acciones a ejecutar al haber sido presionado ese determinado botón. A este fichero ha habido que implementarle el cuerpo de todas las funciones que respondían a la activación de botones, esos botones corresponden con peticiones efectuadas al controlador para modificar las variables del sistema.

Como estos archivos han sido generados de manera automática, y lo único que ha sido necesario hacer fue escribir dentro de los procedimientos que capturaban la pulsación de un botón, las acciones a desarrollar, es decir, las funciones a las que llamar tras haber sido pulsado el botón, no se darán más detalles acerca de estos ficheros.

Cada vez que una función del programa de control quiera escribir en un cuadro de texto de la interfaz debe incluir los paquetes que implementan el interfaz.

4 PAQUETES.

A continuación presentaremos los paquetes que forman el sistema de control de la maqueta. En primer lugar hablaremos de los paquetes definición_tipos y atributos. El paquete definición_tipos agrupa los tipos de datos utilizados en el proyecto. Este paquete será incluido con la cláusula with por el resto de paquetes, por esta razón cuando enumeremos a qué otros paquetes incluye cada uno de los diferentes paquetes del sistema no nombraremos a éste pero lo daremos por supuesto. De manera análoga el paquete atributos recoge los atributos de los objetos esporádicos y cíclicos. En el caso de los esporádicos recogerá las prioridades y para el único objeto cíclico que tenemos, detector_trenes, además de las prioridades también establecerá su período.

4.1 PAQUETE DEFINICIÓN_TIPOS.

Se realizó el paquete definición_tipos para especificar los tipos de datos utilizados en el proyecto:

```
package Definicion_Tipos is
    type Trenes is (ROJO,VERDE);
    subtype Posicion_Tren is Integer range 1..10;
    type Velocidad_Via is (NULA,MINIMA,MEDIA,MAXIMA);
    subtype Terminal is Integer range 0..2;
    type Via is (INTERIOR,EXTERIOR);
    type Sentido_Giro is (HORARIO,ANTIHORARIO);
    subtype Cambio is Integer range 1..6;
    type Cambio_Via is (RECTO,DESVIADO);
    subtype Rampa is Integer range 1..2;
    type Posicion_Rampa is (ACTIVADA,DESACTIVADA);
    type Semaforo is (ROJO,VERDE,APAGADOS);
    type Situacion_Critica is (CAMBIAR_VIA,ALCANZAR_TREN);
    subtype Byte is Integer range 0..255;
    type Bytes is (BYTE_A,BYTE_B,BYTE_C,BYTE_D,BYTE_E,BYTE_F);
  end Definicion_Tipos;
```

- El tipo trenes se creó para distinguir los trenes, no tiene más importancia que esa, el programa distingue los trenes como tren rojo y tren verde.
- El subtipo posición_tren toma valores de 1 a 10 y representa una de las 10 regiones de la maqueta.
- El tipo velocidad_vía representa la velocidad de una de las vías de la maqueta. Los valores que toma son máxima, media, mínima o nula.

- El subtipo terminal toma valores 0, 1 ó 2, según un tren esté en el terminal 1, en el 2, o en ninguno.
- El tipo vía toma los valores interior o exterior, se utiliza para identificar uno de los dos óvalos.
- El sentido de giro de los trenes por las vías puede ser horario, o antihorario.
- El subtipo cambio toma valores del 1 al 6, y se utiliza para identificar uno de los seis cambios de vía. El tipo cambio de Vía representa los dos estados en los que se puede encontrar un cambio, en posición recto o desviado.
- El subtipo rampa toma valores 1 ó 2 para distinguir las dos rampas, el tipo posición rampa representa los estados de las rampas, activadas o desactivadas.
- El tipo semáforos indica el color de los semáforos, si están en rojo, en verde o están desactivados.
- El tipo situación crítica representa las dos situaciones críticas que se nos pueden producir en la maqueta, que los trenes colisionen al cambiar de vía, o que colisionen al alcanzar un tren al otro.
- El subtipo byte toma valores de 0 a 255. El tipo bytes representa cada uno de los seis bytes que controlan las salidas del autómata.

4.2 PAQUETE ATRIBUTOS.

El paquete atributos.ads contiene los atributos temporales de los diversos objetos que constituyen el sistema de control. En él se define la prioridad de los diferentes objetos. Se han asignado prioridades iguales a las tareas similares, por ejemplo se les ha dado la misma prioridad a tareas que modifican los actuadores de la maqueta, como contr_ velocidad, que controla la velocidad de las vías, o contr_ramapas que controla la activación de las rampas de desacople.

Los objetos esporádicos tienen dos atributos, nombre-objeto_Prioridad, que asigna la prioridad al objeto y nombre-objeto_Prioridad_OBCS, que asigna la prioridad al OBCS del objeto. La prioridad toma valores positivos siendo mayor la prioridad de los números más altos. Los objetos cíclicos tendrán además un tercer atributo que representa el periodo de activación del objeto. El periodo viene representado en segundos, así 0.1 representa una décima de segundo.

```
package atributos is

   Detector_Trenes_Prioridad:constant Priority:=17;

   Detector_Trenes_Prioridad_OBCS:constant Priority:=17;

   Detector_Trenes_Periodo:Duration:=0.1;

   Leer_Posiciones_Trenes_Prioridad:constant Priority:=15;

   Leer_Posiciones_Trenes_Prioridad_OBCS:constant Priority:=20;

   Estacionar_Prioridad:constant Priority:=15;

   Estacionar_Prioridad_OBCS:constant Priority:=20;

   Contr_Sacar_Tren_Prioridad:constant Priority:=7;

   Contr_Sacar_Tren_Prioridad_OBCS:constant Priority:=20;
```

```
    Contr_Cambios_Sent_Prioridad:constant Priority:=15;
    Contr_Cambios_Sent_Prioridad_OBCS:constant Priority:=20;
    Contr_Rampas_Prioridad:constant Priority:=15;
    Contr_Rampas_Prioridad_OBCS:constant Priority:=20;
    Contr_Cambios_Via_Prioridad:constant Priority:=15;
    Contr_Cambios_Via_Prioridad_OBCS:constant Priority:=20;
    Contr_Velocidad_Prioridad:constant Priority:=15;
    Contr_Velocidad_Prioridad_OBCS:constant Priority:=20;
    Contr_Posiciones_Prioridad:constant Priority:=15;
    Contr_Posiciones_Prioridad_OBCS:constant Priority:=20;
    Situacion_Critica_Prioridad:constant Priority:=15;
    Situacion_Critica_Prioridad_OBCS:constant Priority:=20;
end atributos;
```

Vemos que la mayoría de los objetos que constituyen el sistema de control son esporádicos. El único objeto cíclico es detector_trenes que cada décima de segundo comprueba la posición de los trenes.

4.3 PAQUETE ACCIONES_INICIALES.

El paquete *acciones_iniciales* llama a los paquetes *inicializar_variables* y *inicializar_ salidas_automata*. La especificación del paquete es la siguiente:

```
package Acciones_Iniciales is
    procedure Inicializar_Maqueta;
end Acciones_Iniciales;
```

El procedimiento *inicializar_maqueta* se encarga de inicializar las variables del sistema y de inicializar las salidas del autómata. Para ello llama a la función *inicializar_ maqueta* del paquete *inicializar_variables* y a la función *inicializar_salidas* del paquete *inicializar_salidas_automata*.

4.4 PAQUETE ACTUALIZAR.

El paquete *actualizar* llama a los paquetes *variables*, *alimentaciones* y modificar_ semáforos. La especificación del paquete es la siguiente:

```
package Actualizar is
  procedure Semaforos;
  procedure Alimentaciones_De_Terminales(T:Terminal);
end Actualizar;
```

El procedimiento *semáforos* comprueba la necesidad de cambiar el color de los semáforos. Si hay que cambiar los semáforos llama a las funciones de *modificar_ semáforos poner_rojo*, o *poner_verde*, según corresponda y modifica las variables del sistema.

El procedimiento *alimentaciones_de_terminales* llama al paquete alimentaciones para que alimente el terminal 1 ó el 2, según el parámetro *T*.

4.5 PAQUETE ALIMENTACIONES.

El paquete *alimentaciones* llama al paquete *variables*. La especificación del paquete es la siguiente:

```
package Alimentaciones is
   procedure Alimentar_T1;
   procedure Alimentar_T2;
   procedure Alimentar_Curvas_Interiores;
   procedure Alimentar_Rectas_Interiores;
   procedure Alimentar_Curvas_Exteriores;
   procedure Alimentar_Rectas_Exteriores;
   procedure Inhibir_T1;
   procedure Inhibir_T2;
   procedure Inhibir_Curvas_Interiores;
   procedure Inhibir_Rectas_Interiores;
   procedure Inhibir_Curvas_Exteriores;
   procedure Inhibir_Rectas_Exteriores;
   procedure Apagar;
end Alimentaciones;
```

Por medio del nombre de los procedimientos podemos hacernos una idea clara de para qué sirve cada uno. Unos procedimientos son para alimentar una determinada zona, y otros son para inhibirla, es decir, hacer que no le llegue la alimentación, siendo las diferentes zonas el terminal 1, el terminal 2, las curvas interiores, las curvas exteriores, las rectas interiores y las rectas exteriores.

El procedimiento apagar apaga los semáforos, apaga la alimentación de las vías y desactiva las rampas.

4.6 PAQUETE CONTR_CAMBIOS_SENT.

El paquete *contr_cambios_sent* llama a los paquetes *variables* y *modificar_cambios_ sent*. La especificación del paquete es la siguiente:

```
package Contr_Cambios_Sent is
```

```
procedure Start(V:Via);
procedure Fin;
end Contr_Cambios_Sent;
```

Contr_cambios_sent es un objeto esporádico. Tiene dos operaciones: *fin* y *start*. El procedimiento *fin* es para finalizar la ejecución del objeto, que está todo el rato a la espera de la señal de activación.

El procedimiento *start* es la señal de activación de la operación esporádica. La operación esporádica invierte el sentido de giro de la vía pasada como parámetro, para lo cual llama al paquete *modificar_cambios_sent* y actualiza las variables del sistema.

4.7 PAQUETE CONTR_CAMBIOS_VIA.

El paquete *contr_cambios_sent* llama a los paquetes *variables* y *modificar_cambios_sent*. La especificación del paquete es la siguiente:

```
package Contr_Cambios_Via is
procedure Start(C_X:Cambio;C:Character);
procedure Fin;
end Contr_Cambios_Via;
```

Contr_cambios_via es un objeto esporádico. Tiene dos operaciones: *fin* y *start*. El procedimiento *fin* es para finalizar la ejecución del objeto, que está todo el rato a la espera de la señal de activación.

El procedimiento *start* es la señal de activación de la operación esporádica. La operación esporádica modifica el cambio de vía pasado como parámetro y según el segundo parámetro lo coloca en una posición o en otra, para lo cual llama al paquete *modificar_cambios_via* y actualiza las variables del sistema.

4.8 PAQUETE CONTR_POSICIONES.

El paquete *contr_posiciones* llama a los paquetes *variables, modificar_Semaforos, contr_Rampas, contr_Velocidad, contr_Cambios_Sent, contr_Cambios_Via, situacion_Critica_Cambio_Via, situacion_Critica_Alcance_Tren, contr_Sacar_Tren* y al paquete *alimentaciones*. La especificación del paquete es la siguiente:

```
package Contr_Posiciones is
procedure Start;
procedure Fin;
end Contr_Posiciones;
```

Contr_posiciones es un objeto esporádico. Tiene dos operaciones: *fin* y *start*. El procedimiento *fin* es para finalizar la ejecución del objeto, que está todo el rato a la espera de la señal de activación.

El procedimiento *start* es la señal de activación de la operación esporádica. La operación esporádica analiza las posiciones de los trenes buscando situaciones conflictivas, así permitirá o retrasará la salida de un tren de terminales, para lo cual llamará a las funciones *retrasar_salida_tren* y *permitir_salida_tren* de *contr_sacar_ tren,* inhibirá la alimentación de los terminales cuando hayamos estacionado el tren, llamando a las funciones *inhibir_T1*, o *inhibir_T2*, según corresponda del paquete *alimentaciones*, comprueba si hay que modificar los semáforos en cuyo caso llamaría a las funciones de *modificar_semáforos*, comprueba y trata los casos de situaciones críticas al cambiar un tren de vía, o al alcanzar un tren al otro, llamando a las funciones de *situación_crícica_cambio_vía y situación_crítica_alcance_tren.*

4.9 PAQUETE CONTR_RAMPAS.

El paquete *contr_rampas* llama a los paquetes *variables* y *modificar_rampas*. La especificación del paquete es la siguiente:

```
package Contr_Rampas is
   procedure Start(R_X:Rampa);
   procedure Fin;
end Contr_Rampas;
```

Contr_rampas es un objeto esporádico. Tiene dos operaciones: *fin* y *start*. El procedimiento *fin* es para finalizar la ejecución del objeto, que está todo el rato a la espera de la señal de activación.

El procedimiento *start* es la señal de activación de la operación esporádica. La operación esporádica activa la rampa pasada como parámetro durante cinco segundos, para lo cual llama al paquete *modificar_rampas* y actualiza las variables del sistema.

4.10 PAQUETE CONTR_SACAR_TREN.

El paquete *contr_sacar_tren* llama a los paquetes *variables, contr_cambios_sent, contr_velocidad* y *actualizar*. La especificación del paquete es la siguiente:

```
package Contr_Sacar_Tren is
   procedure Start;
   procedure Fin;
   procedure Permitir_Salida_Tren;
   procedure Retrasar_Salida_Tren;
end Contr_Sacar_Tren;
```

Contr_sacar_tren es un objeto esporádico. Tiene las operaciones *fin* y *Start*, y otras especificas, *permitir_salida_tren* y *retrasar_salida_tren*. El procedimiento *fin* es para finalizar la ejecución del objeto, que está todo el rato a la espera de la señal de activación.

El procedimiento *start* es la señal de activación de la operación esporádica. La operación esporádica activa la alimentación de terminales, llamando a la función *alimentación_de_terminales* del paquete *actualizar*, cuando no hay riesgo de colisión entre los trenes, si hubiese riesgo de colisión esperará a que pase el peligro.

Los procedimientos *permitir_salida_tren* y *retrasar_salida_tren* tienen la misión de advertir del riesgo de colisión y como su nombre indica, cuando haya riesgo de colisión deben retrasar la salida, y cuando pase el peligro, permitir la salida.

4.11 PAQUETE CONTR_VELOCIDAD.

El paquete *contr_velocidad* llama a los paquetes *variables* y *modificar_velocidad*. La especificación del paquete es la siguiente:

```
package Contr_Velocidad is
   procedure Start(V:Via;Velocidad:Velocidad_Via);
   procedure Fin;
 end Contr_Velocidad;
```

Contr_velocidad es un objeto esporádico. Tiene dos operaciones: *fin* y *start*. El procedimiento *fin* es para finalizar la ejecución del objeto, que está todo el rato a la espera de la señal de activación.

El procedimiento *start* es la señal de activación de la operación esporádica. La operación esporádica cambia la velocidad de la vía, estableciendo la velocidad pasada en el parámetro velocidad del procedimiento *start*, para lo cual llama al paquete *modificar_velocidad* y actualiza las variables del sistema.

4.12 PAQUETE CONTROL.

El paquete *control* llama a los paquetes *variables, contr_rampas, atributos, contr_cambios_sent, contr_sacar_tren, contr_cambios_via, modificar_cambios_via contr_posiciones, contr_velocidad, y estacionar*. La especificación del paquete es la siguiente:

```
package Control is
   procedure Sacar_Tren(T:Terminal);
   procedure Mod_Velocidad(V:Via;Vel:Velocidad_Via);
   procedure Cambiar_Sentido(V:Via);
   procedure Cambio_De_Via(C_X:Cambio;C:Character);
   procedure Activar_Rampa(R_X:Rampa);
   procedure Posicion_Nueva;
   procedure Fin;
 end Control;
```

El procedimiento *sacar_tren* representa una petición para sacar un tren del terminal *T*, al sistema de control. El sistema de control, en función de las variables del sistema, decidirá si es posible o no, el atender la petición. Si la petición es atendida activará el objeto esporádico *contr_sacar_tren.*

El procedimiento *mod_velocidad* representa una petición para cambiar a la velocidad *Vel* la vía *V*, al sistema de control. El sistema de control, en función de las variables del sistema, decidirá si es posible o no, el atender la petición. Si la petición es atendida activará el objeto esporádico *contr_velocidad.*

El procedimiento *cambiar_sentido* representa una petición para cambiar el sentido de la vía *V*, al sistema de control. El sistema de control, en función de las variables del sistema, decidirá si es posible o no, el atender la petición. Si la petición es atendida activará el objeto esporádico *contr_cambios_sent.*

El procedimiento *cambio_de_via* representa una petición para modificar el cambio de vía *C_X*, al sistema de control. El sistema de control, en función de las variables del sistema, decidirá si es posible o no, el atender la petición. Si la petición es atendida activará el objeto esporádico *contr_cambios_via.*

El procedimiento *activar_rampa* representa una petición para activar la rampa *R_X*, al sistema de control. El sistema de control, en función de las variables del sistema, decidirá si es posible, o no, el atender la petición. Si la petición es atendida activará el objeto esporádico *contr_rampas.*

El procedimiento *posición_nueva* se encarga de activar el objeto esporádico *contr_posiciones.*

El procedimiento *fin* se encarga de finalizar la ejecución de los objetos esporádicos a los que llama.

4.13 PAQUETE CONTROLADOR.

El paquete *controlador* llama a los paquetes *control y leer_posiciones_trenes.* La especificación del paquete es la siguiente:

```
package Controlador is
    procedure Sacar_Tren(T:Terminal);
    procedure Mod_Velocidad(V:Via;Vel:Velocidad_Via);
    procedure Cambiar_Sentido(V:Via);
    procedure Cambio_De_Via(C_X:Cambio;C:Character);
    procedure Activar_Rampa(R_X:Rampa);
    procedure Nueva_Deteccion;
    procedure Fin;
end Controlador;
```

Los procedimientos *sacar_tren, mod_velocidad, cambiar_sentido, cambio_de_via y activar_rampa* llaman a las funciones del paquete *control* para que realice las tareas de cada procedimiento.

El procedimiento *nueva_deteccion* se encarga de la señal de activación del objeto esporádico *leer_posiciones_trenes*.

El procedimiento *fin* se encarga de finalizar la ejecución de los objetos *control y leer_posiciones_trenes.*

4.14 PAQUETE DETECTOR.

El paquete *detector* llama a los paquetes *detector_trenes y variables*. La especificación del paquete es la siguiente:

```
package Detector is
    procedure Fin;
    procedure Posicion;
 end Detector;
```

El procedimiento *fin* se encarga de finalizar la ejecución del objeto esporádico *detector_trenes*.

El procedimiento *posicion* se encarga de leer del socket la nueva posición de los trenes y modificar las variables del sistema.

4.15 PAQUETE DETECTOR_TRENES.

El paquete *detector_trenes* llama a los paquetes *controlador, registro y variables*. La especificación del paquete es la siguiente:

```
package Detector_Trenes is
    procedure Fin;
end Detector_Trenes;
```

Sólo tiene el procedimiento *fin*, que es el encargado de finalizar la ejecución de éste objeto cíclico. La operación cíclica lo que hace es llamar a la función *nueva_detección* del objeto *controlador.*

4.16 PAQUETE ESTACIONAR.

El paquete *estacionar* llama a los paquetes *contr_cambios_via y variables*. La especificación del paquete es la siguiente:

```
package Estacionar is
    procedure Start;
    procedure Fin;
```

```
end Estacionar;
```

Estacionar es un objeto esporádico. Únicamente tiene dos operaciones: *fin* y *start*. El procedimiento *fin* es para finalizar la ejecución del objeto, que está todo el rato a la espera de la señal de activación.

El procedimiento *start* es la señal de activación de la operación esporádica. La operación esporádica se encarga de modificar el cambio de vía C-1, para que durante la maniobra de estacionar un tren, lo guardemos en un terminal vacío, para esto llama al objeto *contr_cambios_vía*.

4.17 PAQUETE INICIALIZAR_SALIDAS_AUTOMATA.

El paquete *inicializar_salidas_automata* llama a los paquetes *pserie y variables*. La especificación del paquete es la siguiente:

```
package Inicializar_Salidas_Automata is
   procedure Inicializar_Salidas;
end Inicializar_Salidas_Automata;
```

El procedimiento Inicializar_Salidas se encarga como su nombre indica de inicializar las salidas del autómata y colocar los accionadores de la maqueta en la posición inicial.

4.18 PAQUETE INICIALIZAR_VARIABLES.

El paquete *inicializar_variables* llama al paquete *variables*. La especificación del paquete es la siguiente:

```
package Inicializar_Variables is
   procedure Inicializar_Maqueta;
 end Inicializar_Variables;
```

El procedimiento inicializar_maqueta se encarga de inicializar las variables con el estado inicial de la maqueta.

4.19 PAQUETE LEER_POSICIONES_TRENES.

El paquete *leer_posiciones_trenes* llama a los paquetes *control, registro y variables*. La especificación del paquete es la siguiente:

```
package Leer_Posiciones_Trenes is
   procedure Start;
   procedure Fin;
 end Leer_Posiciones_Trenes;
```

Leer_posiciones_trenes es un objeto esporádico. Tiene dos operaciones: *fin* y *start*. El procedimiento *fin* es para finalizar la ejecución del objeto, que está todo el rato a la espera de la señal de activación.

El procedimiento *start* es la señal de activación de la operación esporádica. La operación esporádica comprueba si las posiciones detectadas de los trenes son nuevas, o si siguen en la misma posición. Cuando se haya detectado una posición nueva, se avisa al objeto *control* de la nueva detección.

4.20 PAQUETE MODIFICAR_CAMBIOS_SENT.

El paquete *modificar_cambios_sent* llama a los paquetes *pserie y variables*. La especificación del paquete es la siguiente:

```
package Modificar_Cambios_Sent is
   procedure Via_Sentido(V:Via;Sentido:Sentido_Giro);
 end Modificar_Cambios_Sent;
```

El procedimiento *Via_sentido* se encarga de modificar las salidas del autómata y las variables del sistema para que la vía *V* tenga el sentido de giro del parámetro *Sentido*.

4.21 PAQUETE MODIFICAR_CAMBIOS_VIA.

El paquete *modificar_cambios_via* llama a los paquetes *pserie y variables*. La especificación del paquete es la siguiente:

```
package Modificar_Cambios_Via is
   procedure Poner_Recto(C_X:Cambio);
   procedure Desviar(C_X:Cambio);
 end Modificar_Cambios_Via;
```

El procedimiento *poner_recto* se encarga de modificar las salidas del autómata y las variables del sistema para poner el cambio *C_X*, recto.

El procedimiento *desviar* se encarga de modificar las salidas del autómata y las variables del sistema para desviar el cambio *C_X*.

4.22 PAQUETE MODIFICAR_RAMPAS.

El paquete *modificar_rampas* llama a los paquetes *pserie y variables*. La especificación del paquete es la siguiente:

```
package Modificar_Rampas is
   procedure Activar(R_X:rampa);
   procedure Desactivar(R_X:rampa);
 end Modificar_Rampas;
```

El procedimiento *Activar* se encarga de modificar las salidas del autómata y las variables del sistema para activar la rampa *R_X*.

El procedimiento *Desactivar* se encarga de modificar las salidas del autómata y las variables del sistema para desactivar la rampa *R_X*.

4.23 PAQUETE MODIFICAR_SEMAFOROS.

El paquete *modificar_semaforos* llama a los paquetes *pserie y variables*. La especificación del paquete es la siguiente:

```
package Modificar_Semaforos is
   procedure Poner_Rojo;
   procedure Poner_Verde;
   procedure Apagar;
end Modificar_Semaforos;
```

El procedimiento *poner_rojo* se encarga de modificar las salidas del autómata y las variables del sistema para poner en rojo los semáforos de la maqueta.

El procedimiento *poner_verde* se encarga de modificar las salidas del autómata y las variables del sistema para poner en verde los semáforos de la maqueta.

El procedimiento *apagar* se encarga de modificar las salidas del autómata y las variables del sistema para apagar los semáforos de la maqueta.

4.24 PAQUETE MODIFICAR_VELOCIDAD.

El paquete *modificar_velocidad* llama a los paquetes *pserie y variables*. La especificación del paquete es la siguiente:

```
package Modificar_Velocidad is
   procedure Circuito(V:Via;Velocidad:Velocidad_Via);
end Modificar_Velocidad;
```

El procedimiento *circuito* se encarga de modificar las salidas del autómata y las variables del sistema para que la vía *V* tenga la velocidad pasada en el parámetro *Velocidad.*

4.25 PAQUETE REGISTRO.

La especificación del paquete es la siguiente:

```
package Registro is
   procedure Leer(Tren:Trenes;Posicion:out Posicion_Tren);
   procedure Escribir(Tren:Trenes;Posicion:Posicion_Tren);
end Registro;
```

Registro es un objeto protegido. Se encarga de ir almacenando la última posición detectada de los trenes en unas variables dedicadas a tal fin. Esas variables son las siguientes:

```
Ultima_Posicion_Tren_Rojo:Posicion_Tren:=1;
Ultima_Posicion_Tren_Verde:Posicion_Tren:=2;
```

Los valores 1 y 2 son los que inicializan las variables.

El objeto tiene dos operaciones: *Leer y Escribir* para modificar las variables.

El procedimiento *Leer* toma el valor almacenado en las variables de *registro*, de la última posición del tren pasado como parámetro y la guarda en *posicion.*

El procedimiento *Escribir* sirve para modificar la variable de *registro* que almacena la última posición de *tren*, almacenando el valor pasado en el parámetro *posicion*.

4.26 PAQUETE SITUACION_CRITICA_ALCANCE_TREN.

El paquete *situacion_critica_alcance_tren* llama al paquete *variables*. La especificación del paquete es la siguiente:

```
package Situacion_Critica_Alcance_Tren is
   procedure Start(Posicion_Critica:Posicion_Tren);
   procedure Stop;
   procedure Inhibir_Region;
   procedure Fin;
 end Situacion_Critica_Alcance_Tren;
```

Situación_critica_alcance_tren es un objeto esporádico. Tiene las operaciones: *fin*, *Start*, *stop* y la operación *inhibir_region*. El procedimiento *fin* es para finalizar la ejecución del objeto, que está todo el rato a la espera de la señal de activación.

El procedimiento *start* es la señal de activación de la operación esporádica, tiene un parámetro *posicion_critica* que indica la zona donde se produce la situación crítica. Este objeto tiene dos operaciones esporádicas. La primera operación esporádica inhibirá la región del tren perseguidor llamando a las funciones del paquete *alimentaciones*, y la segunda volverá a alimentar la región del tren perseguidor, utilizando también las funciones del paquete *alimentaciones*.

El procedimiento *Stop* es el que permite que se ejecute la segunda de las operaciones esporádicas, cuando haya pasado el peligro.

El procedimiento *inhibir_region* es el que permite que se ejecute la primera operación esporádica, y se inhiba la alimentación de la región.

4.27 PAQUETE SITUACION_CRITICA_CAMBIO_VIA.

El paquete *situacion_critica_alcance_tren* llama a los paquetes *contr_velocidad y variables*. La especificación del paquete es la siguiente:

```
package Situacion_Critica_Cambio_Via is
   procedure Start(Posicion_Critica:Posicion_Tren);
   procedure Stop;
   procedure Fin;
end Situacion_Critica_Cambio_Via;
```

Situación_critica_cambio_via es un objeto esporádico. Tiene las operaciones: *fin*, *Start y stop*. El procedimiento *fin* es para finalizar la ejecución del objeto, que está todo el rato a la espera de la señal de activación.

El procedimiento *start* es la señal de activación de la operación esporádica, tiene un parámetro *posicion_critica* que indica la zona donde se produce la situación crítica. Este objeto tiene dos operaciones esporádicas. La primera operación esporádica establecerá velocidad nula a la vía del tren que intenta cambiar, y la segunda establecerá velocidad máxima a la vía del tren que intenta cambiar, para que cambie, para esto activarán al objeto esporádico *contr_velocidad*.

El procedimiento *Stop* es el que permite que se ejecute la segunda de las operaciones esporádicas, cuando haya pasado el peligro.

4.28 PAQUETE VARIABLES.

Variables es un objeto protegido. En él se guardan las variables del sistema. La especificación del paquete es la siguiente:

```
package Variables is
   procedure Modificar_Posicion_Tren(Tren:Trenes;Posicion:Posicion_Tren);
   procedure Modificar_Posicion_Anterior_Tren(Tren:Trenes;Posicion:Posicion_Tren);
   procedure Modificar_Velocidad_Via(V:Via;Velocidad:Velocidad_Via);
   procedure Modificar_Sentido_Giro(Ovalo:Via;Sentido:Sentido_Giro);
   procedure Modificar_Cambio_Via(C_X:Cambio;Posicion:Cambio_Via);
   procedure Modificar_Rampa(R_X:Rampa;Posicion:Posicion_Rampa);
   procedure Modificar_Semaforos(Color:Semaforo);
   procedure Marcar_Terminal(T:Terminal);
   procedure Marcar_Inhibido_Terminal(T:Terminal;Flag:boolean);
   procedure Marcar_Sacando_Tren(Flag:Boolean);
   procedure Marcar_Modo_Supervisado(Flag:Boolean);
   procedure Marcar_Alimentar_Curvas(Flag:Boolean);
```

```
procedure Marcar_Evitar_Colision_Cambio(Flag:Boolean);
procedure Marcar_Evitar_Colision_Alcanzar(Flag:Boolean);
procedure Marcar_Situacion_Critica(Situacion:Situacion_Critica;Flag:Boolean);
procedure Marcar_Tratar_Alcance(Flag:Boolean);
procedure Consultar_Posicion_Tren(Tren:Trenes;Posicion:out Posicion_Tren);
procedure Consultar_Posicion_Anterior_Tren(Tren:Trenes;Posicion:out Posicion_Tren);
procedure Consultar_Velocidad_Via(V:Via;Velocidad:out Velocidad_Via);
procedure Consultar_Sentido_Giro(Ovalo:Via;Sentido:out Sentido_Giro);
procedure Consultar_Cambio_Via(C_X:Cambio;Posicion:out Cambio_Via);
procedure Consultar_Rampa(R_X:Rampa;Posicion:out Posicion_Rampa);
procedure Consultar_Semaforos(Color:out Semaforo);
procedure Consultar_Terminal(T:out Terminal);
procedure Consultar_Inhibido_Terminal(T:Terminal;Flag:out boolean);
procedure Consultar_Sacando_Tren(Flag:out Boolean);
procedure Consultar_Modo_Supervisado(Flag:out Boolean);
procedure Consultar_Alimentar_Curvas(Flag:out Boolean);
procedure Consultar_Evitar_Colision_Cambio(Flag:out Boolean);
procedure Consultar_Evitar_Colision_Alcanzar(Flag:out Boolean);
procedure Consultar_Situacion_Critica(Situacion:Situacion_Critica;Flag:out Boolean);
procedure Consultar_Tratar_Alcance(Flag:out Boolean);
procedure Leer_Byte(B_X:Bytes;Valor:out Byte);
procedure Escribir_byte(B_X:Bytes;Valor:Byte);
end Variables;
```

Las variables del sistema son las siguientes:

```
Posicion_Tren_Rojo:Posicion_Tren:=1;
Posicion_Tren_Verde:Posicion_Tren:=2;
Posicion_Anterior_Tren_Rojo:Posicion_Tren:=3;
Posicion_Anterior_Tren_Verde:Posicion_Tren:=3;
Velocidad_Via_Interior:Velocidad_Via:=NULA;
Velocidad_Via_Exterior:Velocidad_Via:=NULA;
Sentido_Giro_Interior:Sentido_Giro:=HORARIO;
Sentido_Giro_Exterior:Sentido_Giro:=HORARIO;
Cambio_Via_1:Cambio_Via:=RECTO;
Cambio_Via_2:Cambio_Via:=RECTO;
Cambio_Via_3:Cambio_Via:=RECTO;
```

Cambio_Via_4:Cambio_Via:=RECTO;

Cambio_Via_5:Cambio_Via:=RECTO;

Cambio_Via_6:Cambio_Via:=RECTO;

Rampa_1:Posicion_Rampa:=DESACTIVADA;

Rampa_2:Posicion_Rampa:=DESACTIVADA;

Semaforos:Semaforo:=APAGADOS;

Terminal_X:Terminal:=0;

Sacando_Tren:Boolean:=False;

Modo_Supervisado:Boolean:=False;

Alimentar_curvas:Boolean:=False;

Evitar_Colision_Cambio:Boolean:=False;

Evitar_Colision_Alcanzar:Boolean:=False;

Inhibido_T1:Boolean:=True;

Inhibido_T2:Boolean:=True;

A:Byte:=2#1111_0000#;

B:Byte:=2#0000_0000#;

C:Byte:=2#0000_0000#;

D:Byte:=2#0000_0000#;

E:Byte:=0;

F:Byte:=0;

Situacion_Critica_Cambiar_Via:Boolean:=False;

Situacion_Critica_Alcanzar_Tren:Boolean:=False;

Tratado_Alcance:Boolean:=False;

El procedimiento *Modificar_Posicion_Tren* se encarga de almacenar en las variables del sistema que el tren pasado como parámetro, se encuentra en la posición pasada en el otro parámetro.

El procedimiento *Modificar_Posicion_Anterior_Tren* actualiza el valor de la última posición del tren pasado como parámetro, con el valor pasado en el parámetro *posicion*.

El procedimiento *Modificar_Velocidad_Via* actualiza las variables del sistema, almacenando como velocidad de la vía *V*, la pasada en el parámetro *velocidad*.

El procedimiento *Modificar_Sentido_Giro* actualiza las variables del sistema, almacenando como sentido de giro de la vía pasada en el parámetro *ovalo*, el del parámetro *sentido*.

El procedimiento *Modificar_Cambio_Via* actualiza las variables del sistema, almacenando como posición del cambio pasado en el parámetro *C_X*, la pasada en el otro parámetro, *posicion*.

El procedimiento *Modificar_Rampa* actualiza las variables del sistema almacenando que la rampa *R_X* está en la posición pasada como parámetro.

El procedimiento *Modificar_Semaforos* actualiza las variables del sistema almacenando que los semáforos están en el color pasado como parámetro.

El procedimiento *Marcar_Terminal* actualiza las variables del sistema marcando que sacaremos un tren del terminal *T*.

El procedimiento *Marcar_Inhibido_Terminal* actualiza las variables del sistema marcando que el terminal *T* está según el valor booleano *flag*, inhibido, o no.

El procedimiento *Marcar_Sacando_Tren* establece si estamos sacando, o no, un tren de terminales.

El procedimiento *Marcar_Modo_Supervisado* establece si estamos, o no, en el modo supervisado.

El procedimiento *Marcar_Alimentar_Curvas* almacena si estamos, o no, alimentando las curvas de la maqueta.

El procedimiento *Marcar_Evitar_Colision_Cambio* modifica la variable booleana que recoge si estamos tratando una situación crítica de cambio de vía.

El procedimiento *Marcar_Evitar_Colision_Alcanzar* modifica la variable booleana que recoge si estamos tratando una situación crítica al alcanzar un tren al otro.

El procedimiento *Marcar_Situacion_Critica* modifica la variable booleana que recoge si estamos en una situación crítica.

El procedimiento *Marcar_Tratar_Alcance* modifica la variable booleana que recoge si estamos en una situación crítica al alcanzar un tren al otro.

El procedimiento *Consultar_Posicion_Tren* devuelve la posición del tren pasado como parámetro.

El procedimiento *Consultar_Posicion_Anterior_Tren* devuelve la posición anterior del tren pasado como parámetro.

El procedimiento *Consultar_Velocidad_Via* devuelve la velocidad de la vía pasada como parámetro.

El procedimiento *Consultar_Sentido_Giro* devuelve el sentido de giro de la vía pasada como parámetro.

El procedimiento *Consultar_Cambio_Via* devuelve la posición del cambio de vía pasado como parámetro.

El procedimiento *Consultar_Rampa* devuelve la posición de la rampa pasada como parámetro.

El procedimiento *Consultar_Semaforos* devuelve el color de los semáforos.

El procedimiento *Consultar_Terminal* devuelve de qué terminal sacamos el tren.

El procedimiento *Consultar_Inhibido_Terminal* devuelve si el terminal pasado como parámetro está inhibido, o no.

El procedimiento *Consultar_Sacando_Tren* devuelve si estamos sacando un tren de terminales.

El procedimiento *Consultar_Modo_Supervisado* devuelve si estamos en el modo supervisado.

El procedimiento *Consultar_Alimentar_Curvas* devuelve si estamos alimentando las curves de la maqueta.

El procedimiento *Consultar_Evitar_Colision_Cambio* devuelve si estamos evitando una colisión en un cambio de vía.

El procedimiento *Consultar_Evitar_Colision_Alcanzar* devuelve si estamos evitando una colisión al alcanzar un tren al otro.

El procedimiento *Consultar_Situacion_Critica* devuelve si estamos en una situación crítica.

El procedimiento *Consultar_Tratar_Alcance* devuelve si estamos en una situación crítica al alcanzar un tren al otro.

El procedimiento *Leer_Byte* devuelve el valor del byte de las salidas del autómata pasado como parámetro.

El procedimiento *Escribir_Byte* almacena el valor pasado como parámetro, en el byte de las salidas del autómata pasado como parámetro.