Programación Concurrente y Distribuida

Ingeniería Informática
Facultad de Ciencias
Universidad de Cantabria.

Documento:

Red Ferroviaria con Sockets

Autores: José M. Drake
Laura Barros

Santander, 9-10 noviembre, 2011

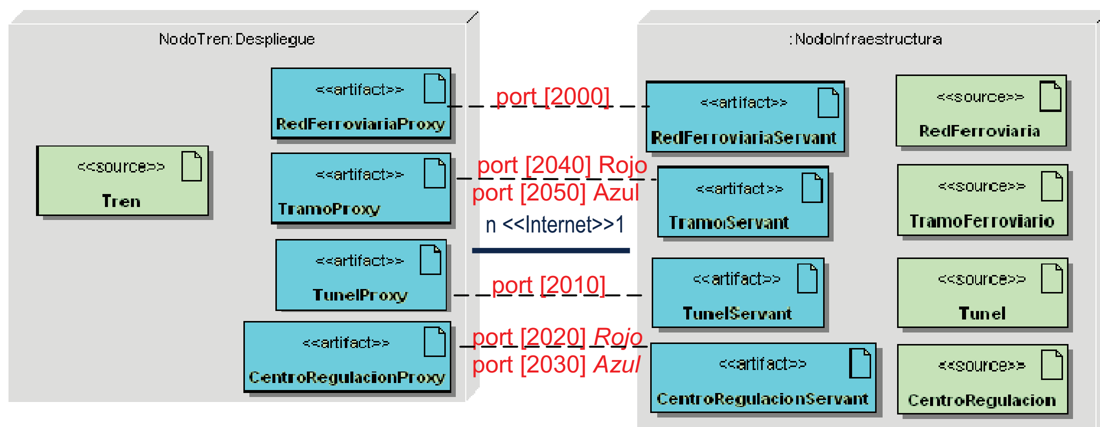
1. Objetivo

Particionar un aplicación distribuída utilizando la API de sockets, proxys y servants.

En esta versión del proyecto Red Ferroviaria se distribuye el código en varios procesadores.

El procesador **NodoInfraestructura** ejecuta el código relativo a los elementos situados en tierra (infraestructura) (***RedFerroviaria, Tunel, TramoFerroviario y CentroRegulación***).

Hay un procesador **NodoTren** por cada uno de los trenes que circulan que ejecuta el código relativo al control de su marcha (***Tren***). La interacción entre los trenes y la infraestructura se realiza utilizando el paradigma Proxy-Servant, y la comunicación entre estos se realiza utilizando sockets.



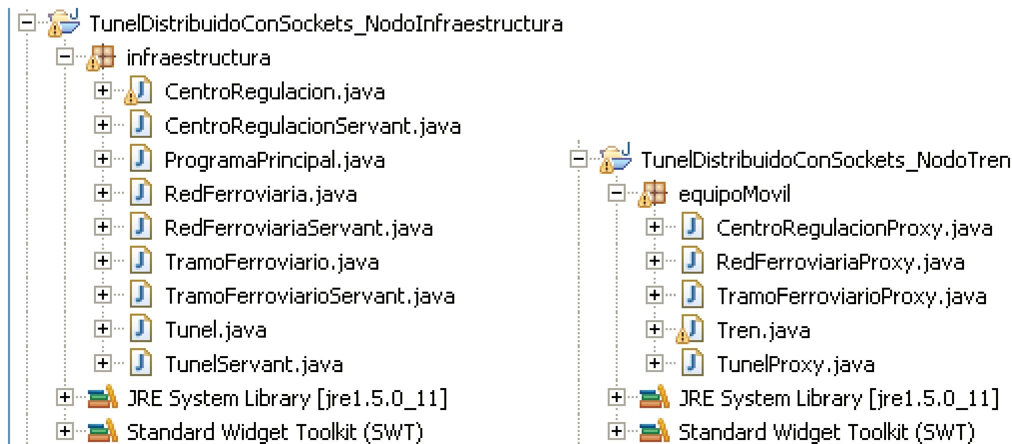
2. Punto de partida

Partimos de la aplicación **RedFerroviariaLockJava**, a la que hemos cambiado su estructura para poder ser distribuida en varios procesadores.

El código se ha separado en dos particiones (paquetes):

El paquete ***infraestructura**:* contiene las clases que se instancian en el procesador **NodoInfraestructura**.

El paquete ***equipoMovil***: contiene las clases que se embarcan en el procesador **NodoTren**.



Las modificaciones que se han hecho respecto de RedFerroviariaLockJava son:

- **Finalización de la actividad de los trenes**: El método ***entro()*** de la clase CentroRegulación retorna un **booleano**. Destinado para el *tren* cuando se accede al CentroRegulación para anunciar que entra en el centro de regulación.
  si es ***true*** :actividad del tren debe finalizar definitivamente.
  si es ***false***: el tren debe continuar.

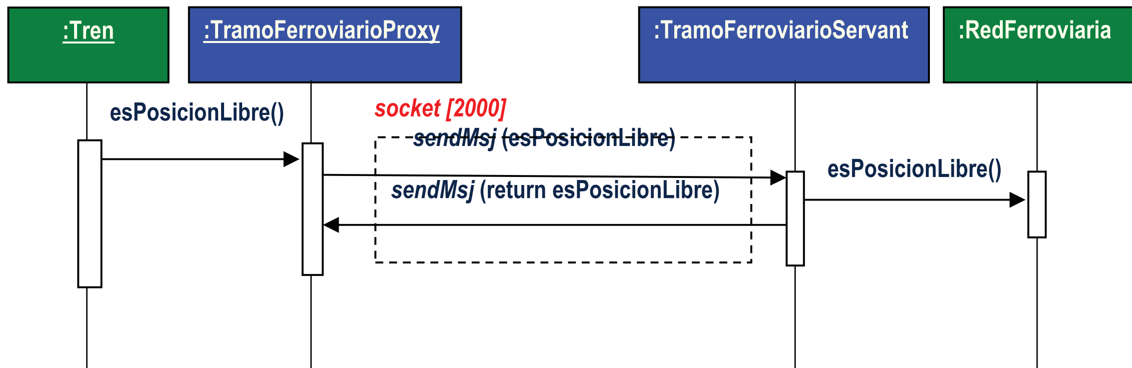
- **Proxys y Servants**: Para facilitar la distribución, la conexión de cada tren con las clases de la infraestructura se ha realizado a través de parejas complementarias Proxy-servant. Se han agregado a cada tren instancias de las clases RedFerroviaria Proxy, TunelProxy y CentroRegulacionProxy, que constituyen el mecanismo por el que los trenes acceden a los correspondientes objetos RedFerroviaria, Tunel y CentroRegulacion. Cada proxy ofrece localmente al tren, el conjunto de operaciones que éste requiere del objeto que representa.

Así mismo, la RedFerroviaria tiene agregada una instancia de la clase RedFerroviariaServant, el Tunel tiene agregado una instancia de la clase TunelServant y cada CentroFerroviario y TramoFerroviario tienen agregados respectivamente una instancia de las clases CentroFerroviarioServant y TramoServant. Los servant ejecutan por delegación en el elemento al que pertenecen los requerimientos que se reciben de los trenes (a través de los proxies).

Cuando un tren interacciona con un elemento de la infraestructura, lo hace invocando un método del correspondiente Proxy que es local. Éste, a través de un socket, se comunica con el correspondiente servant, que se constituye en el intermediario que ejecuta la interacción con el elemento.

Las interacciones que los trenes realizan sobre los elementos de la infraestructura pueden ser de dos tipos:

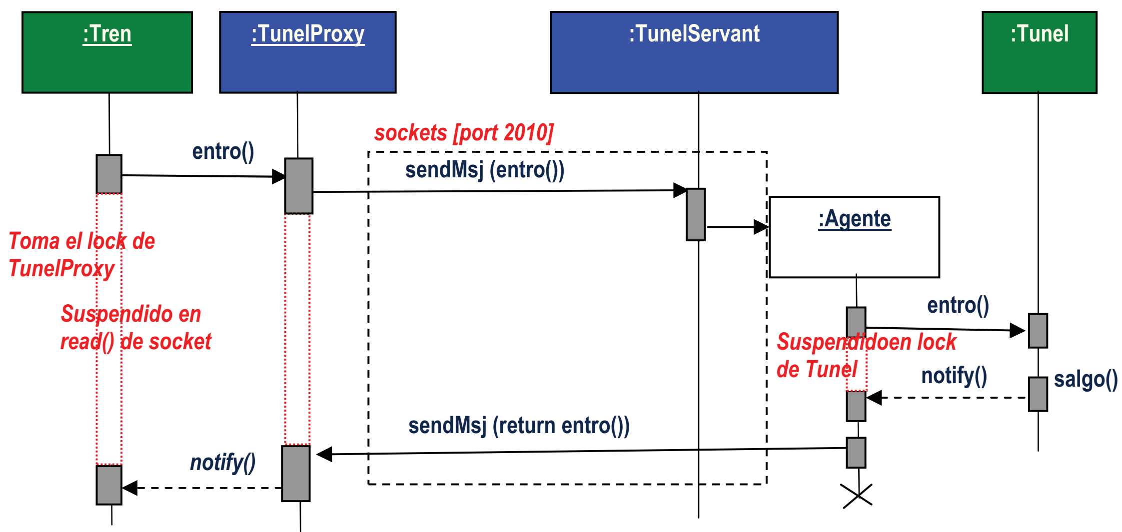
- No bloqueante: No implica ningún bloqueo del thread del tren más allá del envío de la información y el retorno del cómputo. Ejemplos de este tipo de interacción son:
  - salgo():void de la clase Tunel
  - esPosicionLibre(posición,tramo):boolean de la clase RedFerroviaria.



-Bloqueante: El thread del tren se suspende a la espera de un evento de otro thread. Estas son:

- entro():boolean de la clase Tunel.
- entro(tramo):void de la clase CentroRegulacion.

En este caso el correspondiente servant delega en un thread agente para que quede a la espera del evento y permita al servant procesar otras peticiones.



- **Arquitectura de la aplicación.**

<u>Clases incluidas en la partición *infraestructura*:</u>

En el siguiente diagrama de clases se muestran las clases del paquete *infraestructura* que se instancian en el procesador NodoInfraestructura. Las clases RedFerroviaria, TramoFerroviario y Tunel son básicamente iguales a las utilizadas en la aplicación no distribuida RedFerroviariaLockJava.

La clase ***CentroRegulación*** es también la misma, salvo que incorpora el mecanismo de finalización de los trenes. Los cambios son:

<u>Atributo nuevo:</u>

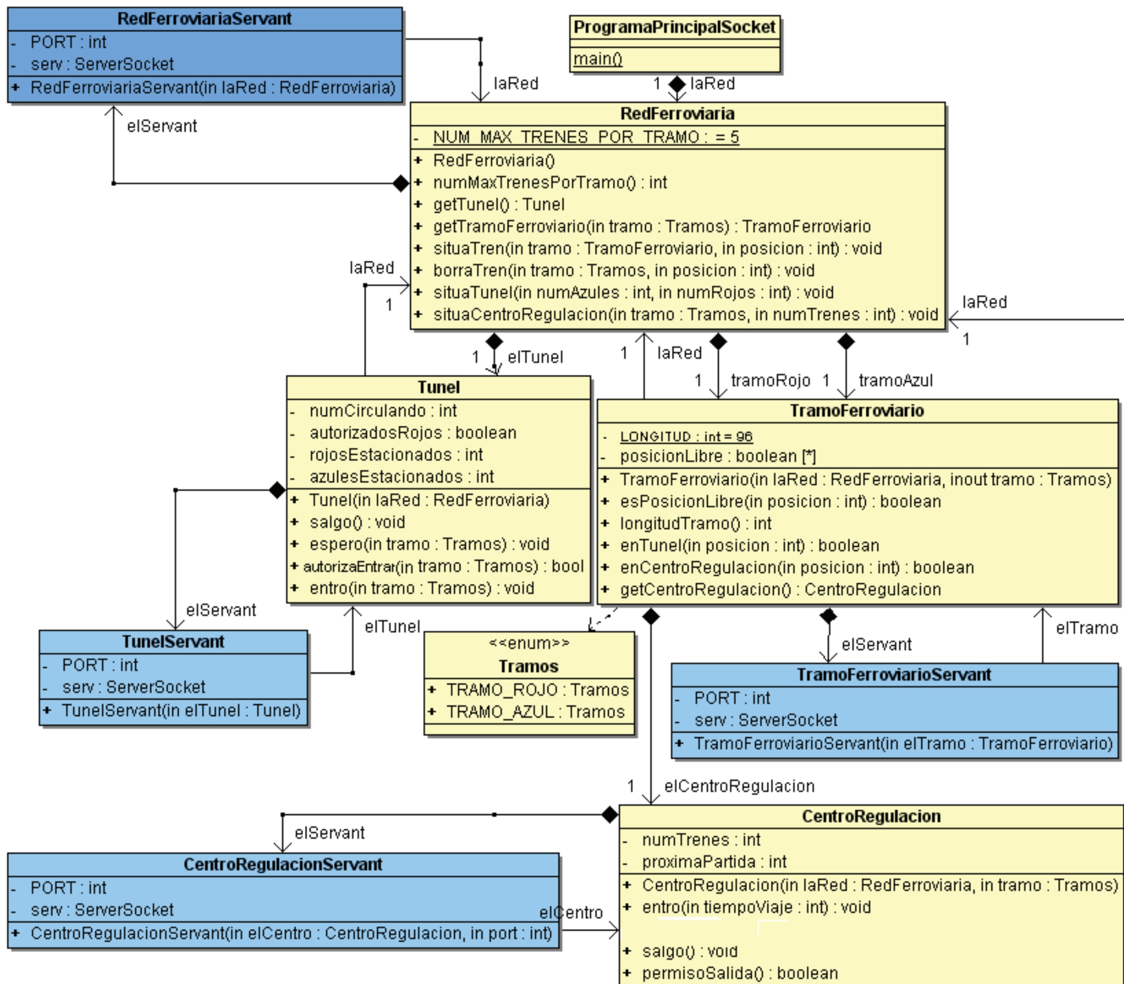
terminado:boolean=false => Vale True si la actividad de los trenes debe finalizar.

Métodos nuevos:

termina() => Establece que la actividad de los trenes del tramo ferroviario debe finalizar.

entro():boolean =>

- Bloquea el thread entrante hasta que llegue la hora de salida de tren.
- Retorna true si la actividad del tren que invoca debe terminar.



3. Nuevas Clases

Clase ***RedFerroviariaServant***: Servant que ejecuta los metodos de la *RedFerroviaria* que requiere cualquier tren a través del **Proxy* complementario utilizando el socket que se establece entre ellos. Recibe por el socket de puerto 2000 como mensajes los métodos que los trenes realizan sobre la RedFerroviaria.Invoca el método requerido en *RedFerroviaria y* retorna el resultado.

Constructor:

RedFerroviariaServant(laRed:RedFerroviaria)=> Recibe como parámetro la referencia a la _RedFerroviaria_ de la que es servant.

Atributos:
laRed:RedFerroviaria => Referencia a la _RedFerroviaria_ sobre el que opera. Se establece en su constructor
<<final>>PORT:int=2000 => Puerto del socket por el que el _*Servant_ queda a la espera de recibir mensajes de los _*Proxies_ complementarios. Es constante [2000].
ServerSocket serv => Server que atiende los requerimientos que recibe de los _*Proxies_ complementarios por su puerto.

Clase **_TunelServant:_** Servant que ejecuta los métodos del _Tunel_ que requiere cualquier tren a través del _*Proxy_ complementario utilizando el socket que se establece entre ellos. Recibe por el socket de puerto 2010 como mensajes las invocaciones de los métodos que los trenes realizan sobre el _Tunel._ Invoca por el método requerido en Tunel. Retorna el resultado.

Constructor:
TunelServant(elTunel:Tunel)=> Recibe como parámetro la referencia al _Tunel_ del que es servant.

Atributos:
elTunel:Tunel => Referencia el _Tunel_ sobre el que opera. Se establece en su constructor.
<<final>>PORT:int=2010 => Puerto del socket por el que el _*Servant_ queda a la espera de recibir mensajes de los _*Proxies_ complementarios. Es constante [2010].
ServerSocket serv => Server que atiende los requerimientos que recibe de los _*Proxies_ complementarios por su puerto.

Clase **_CentroRegulacionServant_**: Servant que ejecuta los métodos del **_CentroRegulacion_** que requiere cualquier tren a través del **_*Proxy_** complementario utilizando el socket que se establece entre ellos. Recibe por el socket de puertos 2020 (tramo Rojo) y 2030 (tramo Azul) como mensajes los métodos que los trenes realizan sobre el correspondiente **_CentroRegulacion._** Invoca el método requerido en **_CentroRegulacion._** Retorna el resultado.

Constructor:
CentroRegulacionServant(elCentroRegulacion:CentroRegulacion, port:int)=> Recibe como parámetro la referencia al CentroRegulación del que es servant, y el puerto (2020 - tramo Rojo ,2030 - tramo Azul).

Atributos:
elCentroRegulacion:CentroRegulacion => Referencia el _CentroRegulacion_ sobre el que opera. Se establece en su constructor.
<<final>>PORT:int => Puerto del socket por el que el _*Servant_ queda a la espera de recibir mensajes de los _*Proxies_ complementarios. Se establece en el constructor. Puede tomar los valores [2020] tramo rojo y [2030] tramo azul.

ServerSocket serv => Server que atiende los requerimientos que recibe de los *Proxies* complementarios por su puerto.

Clase ***TramoFerroviarioServant***: Servant que ejecuta los métodos del ***TramoFerroviario*** que requiere cualquier tren a través del ****Proxy*** complementario utilizando el socket que se establece entre ellos. Recibe por el socket de puertos 2040 (tramo Rojo) y 2050 (tramo Azul) como mensajes los métodos que los trenes realizan sobre el correspondiente ***TramoFerroviario.*** Invoca el método requerido en ***TramoFerroviario.*** Retorna el resultado.

<u>Constructor:</u>
TramoFerroviarioServant(elTramo:TramoFerroviario,port:int)=> Recibe como parámetro la referencia al TramoFerroviario del que es servant, y el puerto 2040 - tramo Rojo ,2050 - tramo Azul).

<u>Atributos:</u>

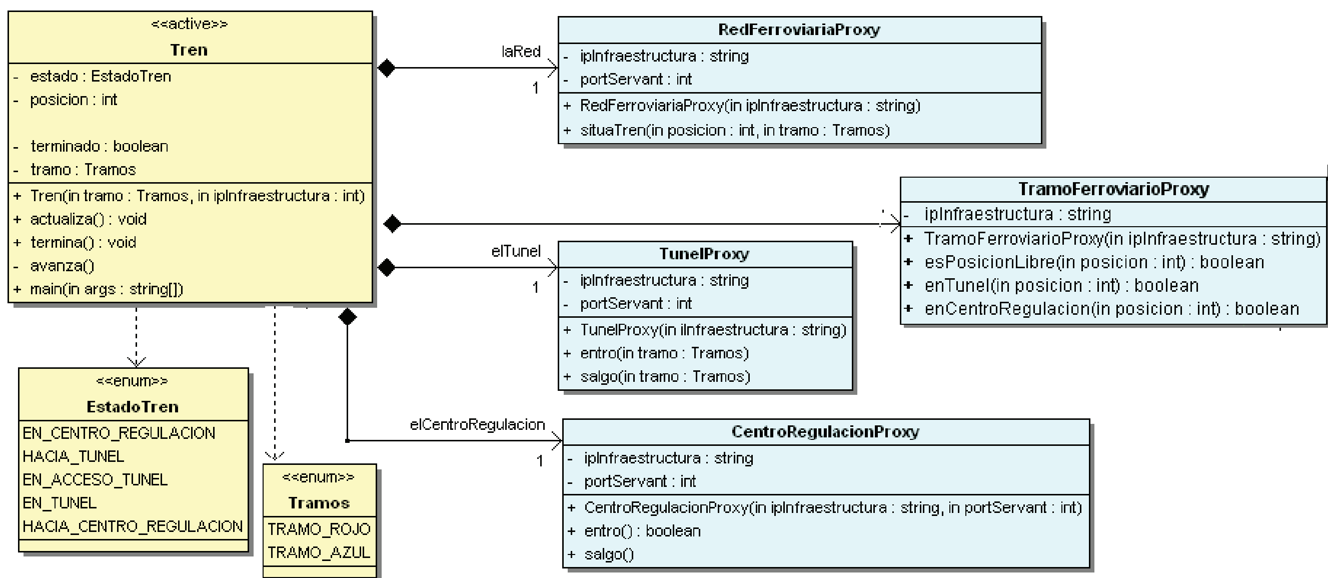
elTramo:TramoFerroviario => Referencia el *TramoFerroviario* sobre el que opera. Se establece en su constructor.

<<final>>PORT:int => Puerto del socket por el que el **Servant* queda a la espera de recibir mensajes de los **Proxies* complementarios. Se establece en el constructor. Puede tomar los valores [2040] tramo rojo y [2050] tramo azul.

ServerSocket serv => Server que atiende los requerimientos que recibe de los **Proxies* complementarios por su puerto.

<u>Clases incluidas en la partición *equipoMovil*:</u>

La clase ***Tren*** es básicamente igual a las utilizadas en la aplicación no distribuida, salvo que se comunica con los elementos incluidos en el paquete *infraestructura* a través de las nuevas clases ***RedFerroviariaProxy, TunelProxy y CentroRegulacionProxy.***



Cambios en la clase ***Tren***:

- Se ha eliminado el acceso a la clase *RedFerroviaria*.
- Se le agregan los **Proxies*:
  - laRed:**RedFerroviariaProxy** => *Proxy que representa localmente y sirve de acceso a la *RedFerroviaria* de **NodoInfraestructura**.
  - elTunel: **TunelProxy** => *Proxy que representa localmente y sirve de acceso al *Tunel* de NodoInfraestructura.
  - elCentroRegulacion:**CentroRegulacionProxy** => *Proxy que representa localmente y sirve de acceso al *CentroRegulacion* de **NodoInfraestructura**, del *TramoFerroviario* sobre el que circula el tren.
  - elTramo:**TramoFerroviarioProxy** => *Proxy que representa localmente y sirve de acceso al *TramoFerroviario*de **NodoInfraestructura**.
- El constructor contiene como parámetro el IP en la red del procesador **NodoInfraestructura**.
- Contiene el procedimiento main() que lanza de forma autónoma un tren en el procesador **NodoTren**. Main requiere tres argumentos tipo string:
  - Tramo es uno de los strings "TRAMO_ROJO" o "TRAMO_AZUL"
  - IP de nodo de infraestructura. Es un string con un formato del tipo: "199.155.255.15".
  - Velocidad del tren. Es un entero.

4. Formatos de los mensajes entre Proxies y Servants

Todos los formatos de invocación tiene un primer byte que identifica la operación, y un byte adicional por cada argumento que tenga la operación. Es decir:

- Código de operación
- Parametro posición (si aplicable)
- Parametro enumerado (color, booleano)

El retorno puede no existir, o consistir en un byte que es siempre booleano.

Operación	Op	Param1	Param2	Return
RedFerroviaria				
situaTren(posición:int, tramo: Tramos)	0	Posición	Tramo	No hay
borraTren(posición:int,tramo:Tramos)	1	Posicion	Tramo	No hay
TramoFerroviario				
esPosicionLibre(posición):boolean	0	Posición	No hay	bool
enCentroRegulacion(posición:int):boolean	1	Posicion	No hay	bool
enTunel(posición:int):boolean	2	Poscion	No hay	bool
Tunel				
entro(tramo:tramos):boolean	0	Tramo	No hay	bool
salgo()	1	No hay	No hay	No hay
CentroRegulacion				
entro: boolean	0	Tramo	No hay	bool
salgo()	1	No hay	No hay	No hay

5. Plan de trabajo

Las clases Tren, RedFerroviaria, TramoFerroviario, Tunel,y CentroRegulacion **NO DEBEN SER MODIFICADAS.**

La práctica consiste en diseñar y escribir el código de las clases RedFerroviariaProxy, RedFerroviariaServant, TunelProxy, TunelServant, CentroRegulacionProxy y CentroRegulacionServant,TramoFerroviarioProxy, TramoFerroviarioServant.

Fase 1: Implementar en un ordenador personal el conjunto equipoMovil e Infraestructura comunicándolos por la IP localhost (127.0.0.1).

Fase 2: Cambiar el diseño de la práctica (no es necesario implementar todo el código, se recomienda proponer la tabla de formatos de los mensajes y un extracto básico del código) proponiendo sólo dos clases (una Proxy y otra Servant) que se comuniquen con un sólo socket teniendo en cuenta que la Red Ferroviara tiene acceso a todos los demás elementos (Centro, Túnel y Tramos).