UNIVERSIDAD DE OVIEDO

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

MÁSTER EN INGENIERÍA INFORMÁTICA

TRABAJO FIN DE MÁSTER

SISTEMA DE INSPECCIÓN DE CARRILES: CONFIGURACIÓN Y CÁLCULO DIMENSIONAL

PEDRO MANSO BERNAL

JULIO 2014

UNIVERSIDAD DE OVIEDO

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

MÁSTER EN INGENIERÍA INFORMÁTICA

TRABAJO FIN DE MÁSTER

SISTEMA DE INSPECCIÓN DE CARRILES: CONFIGURACIÓN Y CÁLCULO DIMENSIONAL

DOCUMENTO IV

COMUNICACIONES

PEDRO MANSO BERNAL

JULIO 2014

ÁREA DE ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES

TUTOR: DANIEL F. GARCÍA MARTÍNEZ

Contenido

1. Introducción al sistema de comunicaciones 5
1.1. Tecnologías de comunicación 6
1.1.1. TCP/IP en .NET 6
1.1.2. WCF 7
2. Protocolos de comunicaciones 8
2.1. Comunicación con el PA 8
2.2. Comunicación con el PLC 8
2.3. Comunicación con el medidor 9
3. Implementación 11
3.1. Emulador del PA - *PAServer* 11
3.1.1. Implementación de los mensajes 12
3.1.2. Interfaz gráfica 14
3.2. Emulador del PLC – *PLCServer* 16
3.2.1. Interfaz gráfica 17
3.3. Medidor - *RailMeter* 18
3.4. Sistema de comunicaciones - *RailCommunications* 19

1. Introducción al sistema de comunicaciones

El medidor no puede funcionar correctamente de forma aislada, sino que necesita conocer su entorno y, en menor medida, modificarlo: debe conocer cuándo entran y salen carriles y de qué tipos son; debe ser capaz de desplegar y replegar la plantilla de calibración, etc.

Para gestionar la entrada y la salida, y al mismo tiempo aislar al medidor de su entorno, se implementa un sistema de comunicaciones independiente. Este sistema es el responsable de gestionar toda la comunicación entre el medidor y otros equipos: el ordenador de proceso (PA) y el PLC que indica dónde comienzan y terminan los carriles.

Más específicamente, el sistema de comunicaciones establece conexiones con dichos equipos externos, transmite al medidor sus peticiones y remite los mensajes del medidor al equipo que corresponda, dependiendo del tipo de mensaje.

De este modo, el medidor ya no es responsable de comunicarse con varios equipos externos diferentes, sino simplemente con el sistema de comunicaciones creado específicamente para esa tarea, y que puede modificarse en caso de cambiar los equipos externos, sin tener que modificar el código del medidor en sí.

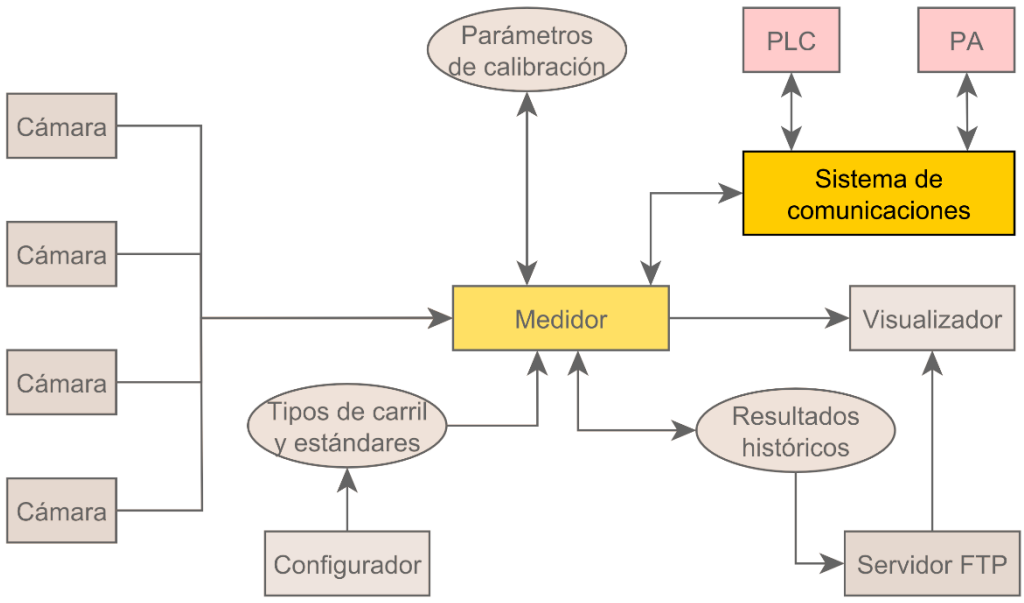

Figura 1: Diagrama del sistema, mostrando solamente los elementos relacionados con el sistema de comunicaciones

La Figura 1 y la Figura 2 muestran la relación del sistema de comunicaciones con los otros elementos.

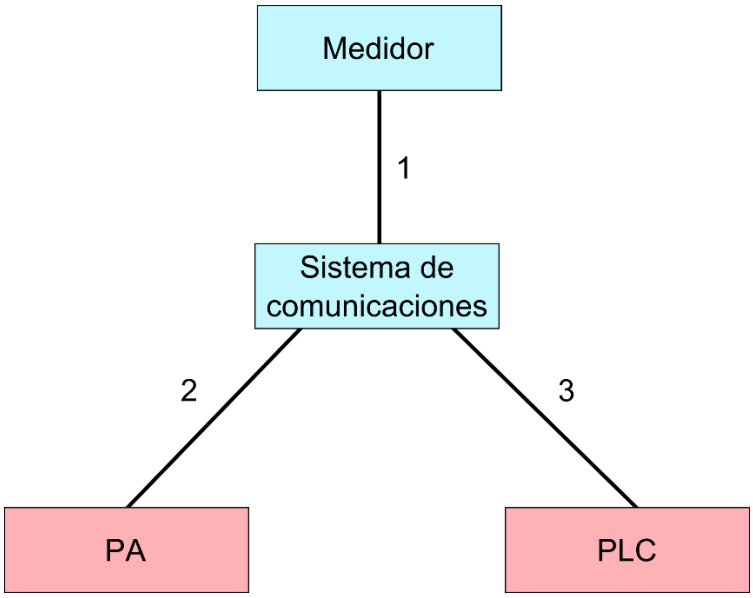

Figura 2: Estructura de las comunicaciones (en rojo los equipos externos; en azul los pertenecientes al equipo de medición)

Para poder llevar a cabo pruebas con las comunicaciones en el laboratorio se han implementado dos emuladores que se encargan de generar los mensajes tal y como lo harían los elementos reales de los que no se dispone en el laboratorio, el PA y el PLC. Estos emuladores deben seguir estrictamente los protocolos de comunicaciones definidos a priori para establecer la secuenciación de mensajes. Dichos protocolos serán los usados en un futuro cuando el sistema se ponga en producción.

1.1. Tecnologías de comunicación

1.1.1. TCP/IP en .NET

En la biblioteca de clases de .NET (.NET Framework Class Library), la responsable en última instancia de las comunicaciones de red es la clase Socket (System.Net.Sockets.Socket). Esta clase, como su nombre sugiere, ofrece una interfaz a bajo nivel para el trabajo con sockets de red.

Por ello, existen también otras clases de más alto nivel que ofrecen una interfaz más amigable a las operaciones con sockets. Una de ellas es NetworkStream (System.Net.Sockets.NetworkStream), que trata las comunicaciones de red como un flujo de datos.

Puesto que NetworkStream hereda de la clase Stream (System.IO.Stream), los flujos con los que trabaja pueden tratarse igual que los procedentes de ficheros (FileStream) o de memoria (MemoryStream).

La mayor parte de los métodos de ambas clases para la realización de operaciones síncronas (Read, Write, Receive, etc.) cuentan también con versiones asíncronas[1] (BeginRead y EndRead,

[1] En .NET 4.5 se añadieron métodos con nombres terminados en *Async* pensados para utilizarse con la interfaz *async*/*await*. Sin embargo, no pueden emplearse en este proyecto ya que ha de ser compatible con la versión 4.0.

etc.) que permiten especificar una función a la que se llamará tras completar la operación correspondiente. Esto evita tener que gestionar manualmente los hilos, ya que en este caso es .NET quien gestiona los que sean necesarios.

1.1.2. WCF

Windows Communication Foundation, WCF, es una API para la creación de aplicaciones orientadas a servicios. Forma parte del framework .NET.

Se trata de un mecanismo de llamada a procedimientos remotos (RPC), y su funcionamiento es similar al de RMI en Java, por ejemplo. La idea general es declarar una interfaz en el servidor, que contiene los métodos que se ofrecen, e implementarla; y en el cliente obtener una referencia al servicio (que puede "autodescubrirse") y utilizar directamente la interfaz para consumir la funcionalidad que el servidor proporciona.

Además de esta comunicación en una dirección, WCF también ofrece posibilidades de comunicación "dúplex". En la comunicación dúplex, además de la interfaz del servidor ofrecida por el servidor, puede definirse una segunda interfaz que el cliente debe implementar, de modo que el servidor pueda emplearla una vez establecida la conexión.

WCF se encarga de gestionar y abstraer los aspectos de bajo nivel de la comunicación entre el cliente y el servidor, que puede emplear distintos enlaces o "bindings". Entre los "bindings" disponibles por defecto existen varios basados en HTTP (con SOAP o REST), y otros en protocolos específicos de .NET.

2. Protocolos de comunicaciones

2.1. Comunicación con el PA

El sistema de comunicaciones ha de conectarse al PA ("Process Automation", el ordenador de proceso) para recibir la información identificativa de los carriles que se van a medir y para enviar los resultados de las mediciones para que pueda dárseles uso.

Esta comunicación se realiza a través de un protocolo propio sobre TCP. La especificación completa de este protocolo, incluyendo el formato general de los mensajes y el contenido de cada uno, se describe en el anexo D - Comunicación con el PA.

Estos mensajes están compuestos por cadenas de longitud variable para la comunicación entre ambos extremos. El PA actúa como servidor y el sistema de comunicaciones como cliente.

Los mensajes empleados en este protocolo son los siguientes:

- **Life Message**, mensaje que envía periódicamente el PA al sistema de comunicaciones. El sistema de comunicaciones espera recibirlo cada 120 segundos como máximo.
- **Confirmation Life Message**, mensaje de respuesta al mensaje anterior, del sistema de comunicaciones al *PA*.
- **New rail data**, información del siguiente carril a medir, del PA al sistema de comunicaciones.
- **Confirmation of new rail data received**, confirmación del mensaje anterior, del sistema de comunicaciones al PA.
- **Last rail measurement results**, conjunto de las mediciones realizadas al último carril medido, del sistema de comunicaciones al PA.
- **Confirmation of reception of last rail measurement results**, confirmación del mensaje anterior, del PA al sistema de comunicaciones.
- **Request one rail measurement results**, petición de las mediciones de un carril que ha sido medido previamente, del PA al sistema de comunicaciones.
- **One rail measurement results,** respuesta a la petición anterior con las mediciones del carril solicitado, de sistema de comunicaciones al PA.
- **MV Status**, estado actual en que se encuentra el medidor. Enviado por el sistema de comunicaciones al PA.

2.2. Comunicación con el PLC

El sistema de comunicaciones ha de conectarse al PLC para poder controlar cuándo se despliega o se repliega la plantilla de calibración, y para conocer cuándo comienzan y terminan los carriles.

Los mensajes que se pueden enviar entre el PLC y el sistema de comunicaciones han sido definidos previamente en un protocolo, que se describe en el anexo C - Comunicación con el PLC. Como en el caso del protocolo para la comunicación con el PA, este se utiliza directamente sobre TCP.

Cada uno de los mensajes, en ambos sentidos, tiene un tamaño de 1 byte. El PLC opera como servidor mientras que el sistema de comunicaciones es el cliente.

Este protocolo constará de una serie de 4 mensajes, los cuales serán:

- Mensaje **Hello**: mensaje de inicio de sesión, en ambos sentidos.
- Mensaje **Enter**: entra un nuevo carril en la zona de medición, del PLC al sistema de comunicaciones.
- Mensaje **Exit**: el carril actualmente en la zona de medición la abandona, del PLC al sistema de comunicaciones.
- Mensaje **Deploy_CalibrationTemplate**: solicitud de que se despliegue la plantilla de calibración, del sistema de comunicaciones al PLC.
- Mensaje **Retract_CalibrationTemplate**: solicitud de que se recoja la plantilla de calibración, del sistema de comunicaciones al PLC.

2.3. Comunicación con el medidor

Entre el sistema de comunicaciones y el medidor se emplea una interfaz basada en la tecnología Windows Communication Foundation (WCF).

No existe ningún requisito de seguridad, dadas las circunstancias (el entorno en el que se ejecuta y el hecho de que el sistema de comunicaciones es meramente un adaptador, una "fachada" que de todas formas ha de conectarse de forma insegura a otros sistemas), lo que simplifica la configuración notablemente.

Sin embargo, sí es necesario emplear una comunicación en ambos sentidos: por ejemplo, el sistema de comunicaciones debe comunicar al medidor cuándo comienza y termina un carril, mientras que el medidor ha de señalarle cuándo debe desplegarse y retirarse la plantilla de calibración. Esto requiere que ambos puedan enviar mensajes en cualquier momento. Puesto que HTTP no permite esto por sí mismo (el servidor se limita a responder a las peticiones del cliente), no puede emplearse como canal de la comunicación ("enlace" o "binding" en la terminología de WCF). Debe emplearse un tipo de enlace "dúplex".

Entre los enlaces dúplex que proporciona WCF existe uno basado también en HTTP, pero que emplea una conexión distinta (cliente y servidor) en cada sentido. El uso de este enlace resulta impráctico.

Por otra parte, existe un enlace alternativo, denominado net.tcp, que consiste en el uso de un protocolo desarrollado especialmente para WCF. Además de proporcionar mayor rendimiento (lo cual no es particularmente importante en este caso), permite la comunicación en ambos sentidos sin necesidad de modificar la configuración por defecto. Por tanto, es este el que se utiliza.

La interfaz finalmente empleada es la siguiente:

Los métodos que expone el medidor (que actúa como servidor):

- `bool SetRailHeader(RailHeader header);`

Permite especificar la cabecera (identificador, tipo y norma empleada) del siguiente carril que se va a medir. Devuelve true si el medidor reconoce el tipo y la norma, y false si no reconoce alguno de los dos.

- `string GetRailMeasurement(string id);`
  Permite obtener el resultado de la medición de un carril concreto, especificando su identificador. Si existe un carril con ese identificador, devuelve el contenido del fichero de medición correspondiente. Si no existe, devuelve una cadena vacía.

- `void RailEnter();`
  Señala la entrada de un nuevo carril.

- `void RailExit();`
  Señala la salida de un carril.

- `void ConnectionStatusChanged(bool paConnected, bool plcConnected);`
  Informa de un cambio en la disponibilidad del ordenador de proceso (*paConnected*) o del PLC (*plcConnected*). Para cada uno de los dos, el valor true indica que está conectado mientras que false indica que no lo está.

- `void Register();`
  Registra el cliente y permite que reciba las llamadas correspondientes de la otra interfaz.

Los métodos que expone el sistema de comunicaciones (que actúa como cliente; se trata de una interfaz que se denomina de “callback”):

- `void StartCalibration();`
  Despliega la plantilla de calibración.

- `void EndCalibration();`
  Repliega la plantilla de calibración.

- `void SendLastRailMeasurement(string id, string data);`
  Recibe los resultados de la última medición realizada (llamado por el medidor al completar la medición de un carril).

- `void ServerClosed();`
  Notifica que el medidor se cerrará inmediatamente.

3. Implementación

3.1. Emulador del PA - *PAServer*

El emulador del ordenador de proceso, llamado *PAServer*, es un programa que actúa como servidor en la comunicación con *RailCommunication*, permitiendo emplearlo fuera del entorno de producción en el que está presente el ordenador de proceso. Este emulador incluye una interfaz gráfica para elegir de forma interactiva los mensajes que se envían al sistema de comunicaciones.

Para su implementación se ha empleado el patrón State: se han definido varios estados independientes con distintos comportamientos, que se utilizan en distintas etapas de la comunicación. Esto simplifica la adición de nuevos estados que puedan ser necesarios en el futuro.

En la Figura 3 se muestra el diagrama de clases de *PAServer*, en el que puede observarse claramente la implementación de dicho patrón.

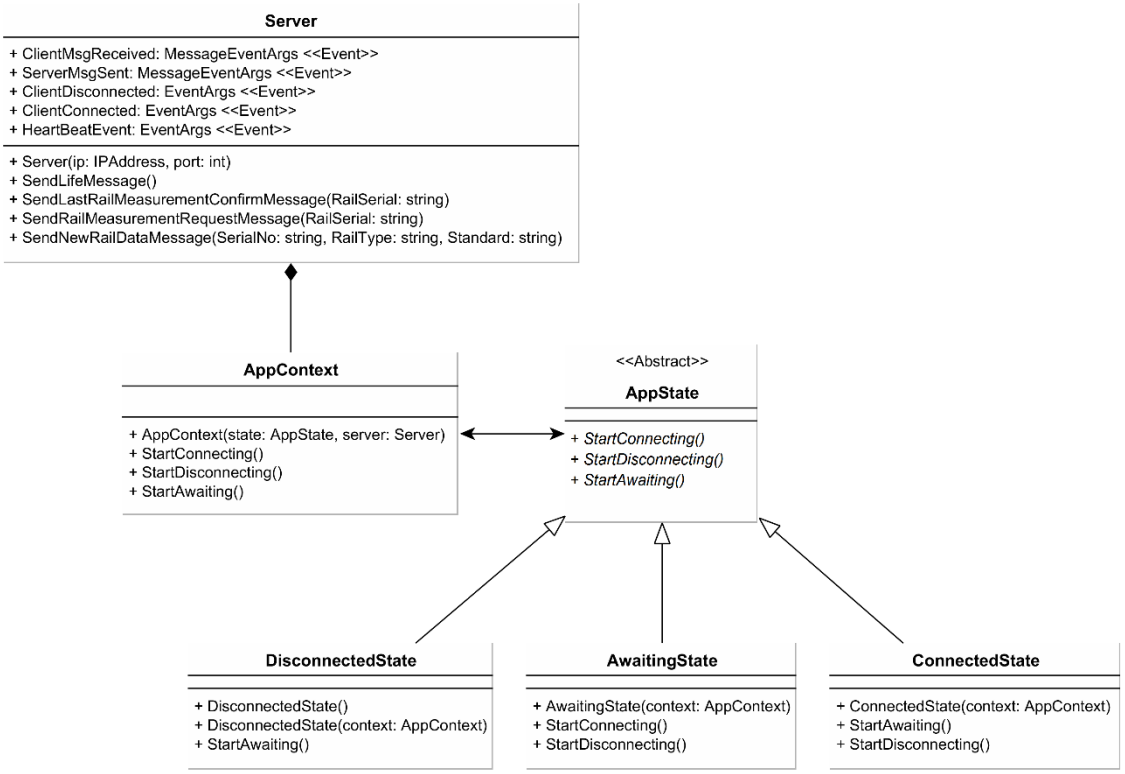

Figura 3: Diagrama de clases para el PAServer

El estado inicial del servidor es el estado *Disconnected*. Este es un estado meramente de transición desde el que se pasa al estado *Awaiting*, en el que el servidor espera conexiones entrantes. Una vez establecida una conexión, el servidor pasa al estado *Connected*, en el que se mantiene mientras dure esta. Cuando la conexión se corta, vuelve al estado *Awaiting*. Cuando el servidor ha de cerrarse, pasa de cualquiera de esos dos estados al estado *Disconnected*. Esta estructura puede verse en la Figura 4.

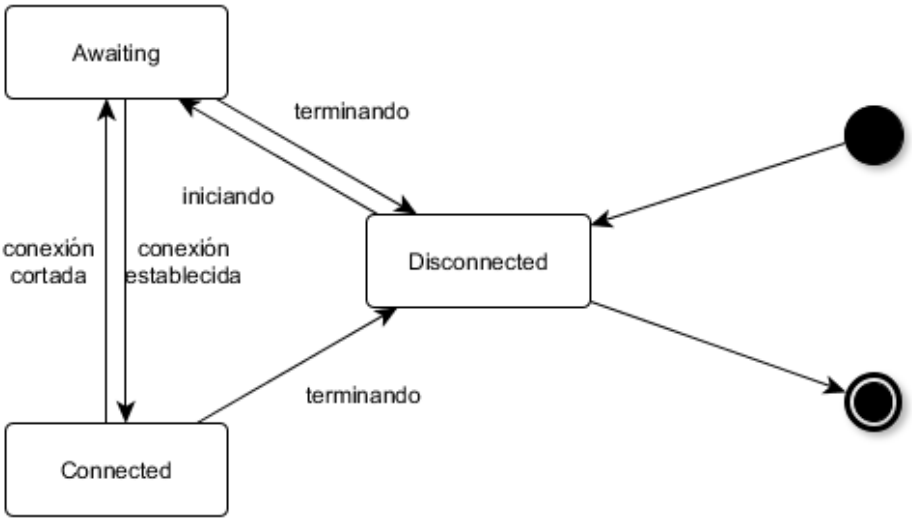

Figura 4: Diagrama de estados de *PAServer*

Todas las comunicaciones, así como el establecimiento de la conexión, se realizan mediante operaciones asíncronas (véase la sección 1.1.1).

3.1.1. Implementación de los mensajes

Para cada uno de los mensajes definidos en la especificación se implementa una clase responsable de serializarlo y deserializarlo. El diagrama de clases se muestra en la Figura 5.

Para la serialización, la clase base, *Message*, declara un método abstracto protegido *GetContent*, que debe implementarse en cada mensaje. Tomando el contenido del mensaje, Message genera una cabecera con los valores correctos y lo combina todo en una única cadena.

La deserialización es más compleja. Para implementarla, una posibilidad sería detectar el tipo de mensaje en la clase *Message* y llamar desde allí a un método de la clase adecuada. Sin embargo, para eso *Message* necesitaría conocer todos los mensajes existentes, dificultando una posible ampliación.

En su lugar, el método *Read* de *Message* devuelve un *BinMessage*, que es meramente un contenedor; y cada una de las demás clases hijas define una conversión explícita desde *BinMessage*.

De este modo, cada clase de mensaje es responsable tanto de su lectura como de su escritura. Esto no afecta apenas al código que haga uso de ellas, que simplemente ha de llevar a cabo una conversión a la clase correcta en función del tipo; lo cual habría de hacer, aún con la primera solución señalada antes, para poder acceder a los campos específicos de cada mensaje.

Esta misma implementación de los mensajes se emplea en el sistema de comunicaciones. La Figura 5 es un diagrama UML que muestra de forma simplificada las relaciones entre los mensajes.

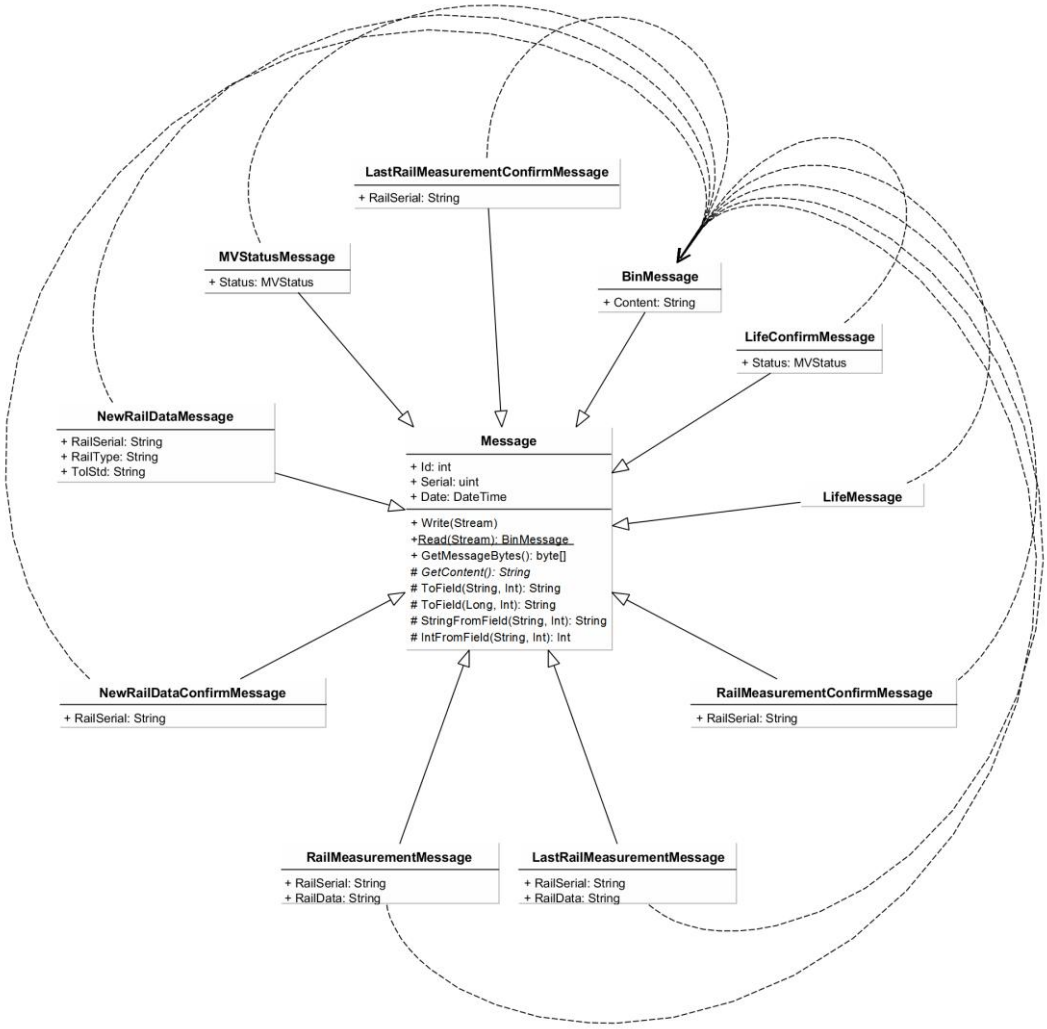

Figura 5: Diagrama de las clases empleadas en el procesamiento de mensajes

3.1.2. Interfaz gráfica

Para la implementación de este emulador se ha optado por una solución con una interfaz de usuario gráfica realizada mediante la tecnología de Windows Forms.

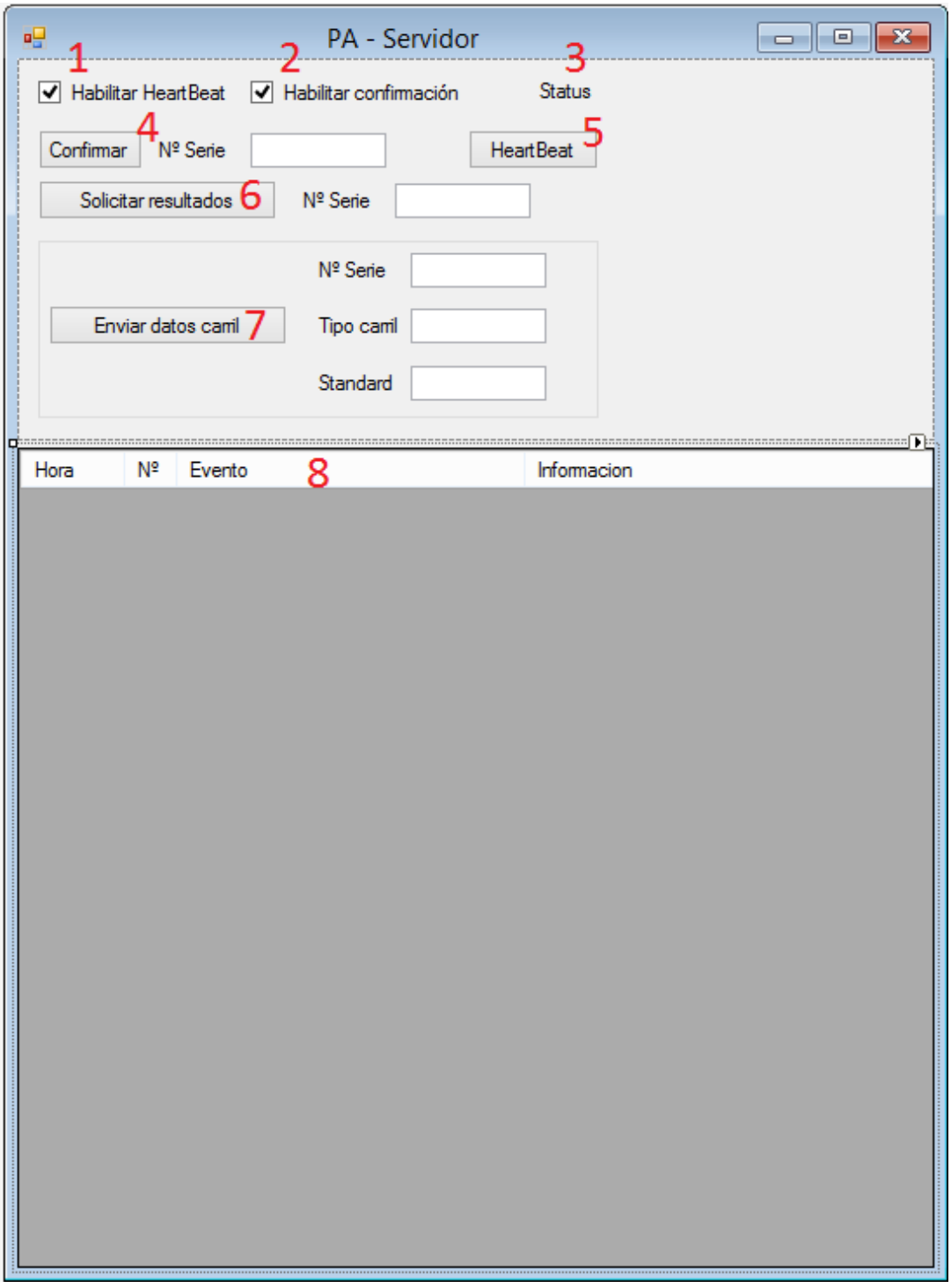

Figura 6: Interfaz gráfica del *PAServer*

La interfaz gráfica cuenta con una serie de controles (señalados en la Figura 6) que el usuario puede utilizar para emular el comportamiento del PA y elegir los mensajes que se han de generar. Estos controles son los siguientes:

1. **Habilitar heartbeat:** Este checkbox permite al usuario habilitar o deshabilitar el envío automático de *LifeMessage*, que se realiza cada 60 segundos, si este checkbox está marcado.
2. **Habilitar confirmación:** Mediante este checkbox, el usuario puede habilitar o deshabilitar el envío automático de mensajes *LastRailMeasurementConfirmMessage* tras recibir de *RailCommunications* mediciones nuevas.
3. **Status:** Este control muestra el estado actual del servidor:
    a. En color verde si el servidor está en el estado *Connected* (cliente conectado).
    b. En color naranja si el servidor está en el estado *Awaiting* (esperando conexión).
    c. En color rojo si el servidor está en el estado *Disconnected* (sin conexión).
4. **Confirmar:** Este botón permite confirmar manualmente la recepción de las mediciones (enviar inmediatamente un mensaje *LastRailMeasurementConfirmMessage*) por parte del *PAServer*. El carril cuya medición se confirma es el que tiene el número de serie que se introduzca en el cuadro de texto que se encuentra a la derecha del botón. Este número de serie sigue un formato determinado especificado por ArcelorMittal, si bien ni el servidor ni ningún programa del sistema validan el texto que se introduce.
5. **Heartbeat:** Este botón permite enviar manualmente un *LifeMessage*.
6. **Solicitar resultados:** Este botón permite enviar un mensaje *SendRailMeasurementRequestMessage*, mediante el cual el PA puede pedir que se le envíen los resultados de la medición de un carril con un número de serie igual que el introducido en el cuadro de texto que se encuentra a la derecha del botón.
7. **Enviar datos carril:** Este botón permite el envío de un mensaje *NewRailDataMessage* con los datos que se introduzcan en los cuadros de texto que están a su derecha; es decir: el número de serie, el tipo de carril y la norma con la que se va a medir este carril.
8. **Tabla de información:** En este campo se mostrarán todos los eventos producidos en el *PAServer*, ya sean de recepción, de envío de mensajes o de conexión o desconexión del cliente.

3.2. Emulador del PLC – *PLCServer*

Al igual que en el caso del emulador del PA, el *PAServer*, el *PLCServer* actúa como servidor en la comunicación con *RailCommunication*, en este caso reemplazando fuera del entorno de producción al PLC que coordina la medición. Este emulador incluye una interfaz gráfica similar a la del *PLCServer*.

Se implementa también un patrón State, con los mismos estados que el *PAServer* y las mismas transiciones entre ellos.

El diagrama de clases del PLCServer se puede observar en la Figura 7.

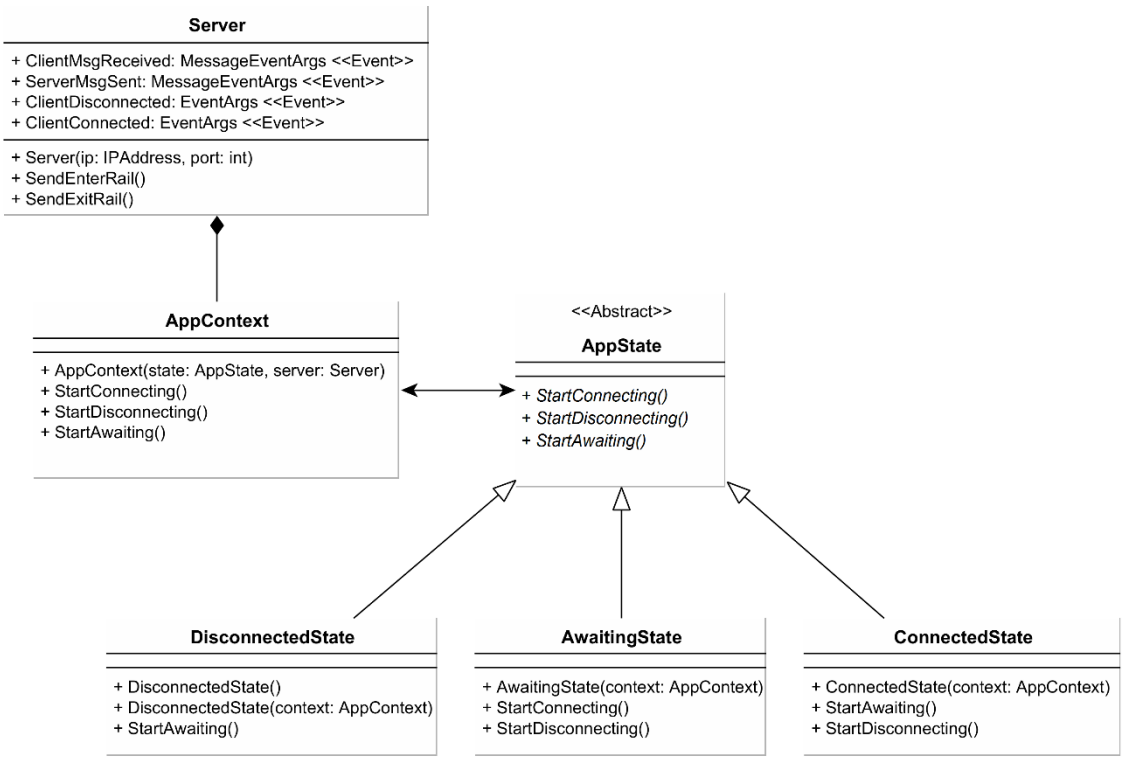

Figura 7: Diagrama de clases para el *PLCServer*

Los mensajes que van a ser enviados por el emulador del PLC, *PLCServer*, son mucho más sencillos que los empleados en el caso del PA. En lugar de una jerarquía de clases de mensaje, se trabaja simplemente con bits individuales. Por lo demás, la lógica es muy similar.

3.2.1. Interfaz gráfica

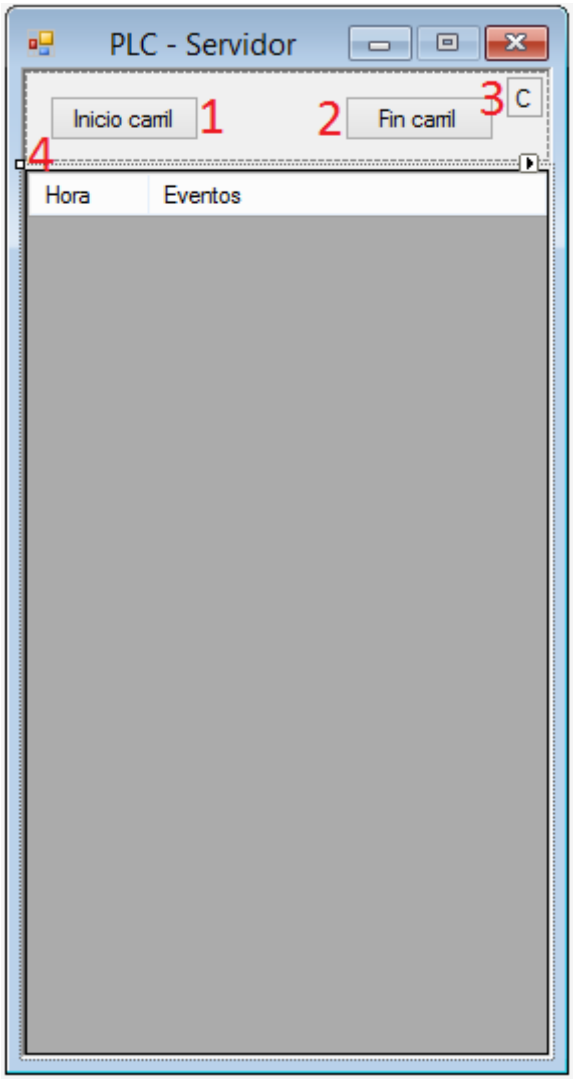

Figura 8: Interfaz gráfica del *PLCServer*

El *PLCServer* cuenta con una serie de controles (visibles en la Figura 8) que el usuario puede utilizar para emular el comportamiento del PLC y enviar los mensajes que podría generar. Estos controles son los siguientes:

1. **Inicio carril:** Este botón permite enviar de forma inmediata un mensaje de inicio de carril.
2. **Fin carril:** Este botón permite enviar de forma inmediata un mensaje de fin de carril.
3. **Estado:** Este control muestra el estado actual del servidor:
    a. En color verde si el cliente está conectado (estado *Connected*).
    b. En color naranja si el servidor está esperando conexiones del cliente (estado *Awaiting*).
    c. En color rojo si no hay conexión (estado *Disconnected*).
4. **Tabla de información:** En este campo se mostrarán todos los eventos producidos en el emulador del PLC: recepción y envío de mensajes, conexión y desconexión del cliente.

3.3. Medidor - *RailMeter*

La finalidad última del sistema de comunicaciones es servir de puente entre el medidor (*RailMeter*), que se describe en el documento III - Medidor, y otros equipos.

En el medidor se implementa (clase MeasurementService) un servidor WCF con la interfaz que se describe en la sección 2.3. Esta implementación se limita a lanzar los eventos internos correspondientes a cada posible llamada WCF. Al ocuparse WCF de todos los detalles de bajo nivel, esto tiene escasa complejidad técnica.

En cuanto al callback que debe implementarse en el cliente para que pueda utilizarse desde el medidor, este se expone al código del medidor a través de una propiedad (MeasurementService.Callback). De este modo, si no hay un cliente conectado, la llamada llega a una instancia de una "implementación" de la interfaz, llamada DummyMeasurementCallback, que se limita a registrarla como un error de forma transparente para el código que llama. Si hay un cliente conectado, la llamada pasa por una instancia de MeasurementCallbackWrapper, que se encarga de llamar al método WCF correspondiente, y en su caso de registrar las excepciones que se produzcan.

En la Figura 9 se muestra un diagrama de las relaciones entre las clases explicadas en los anteriores párrafos.

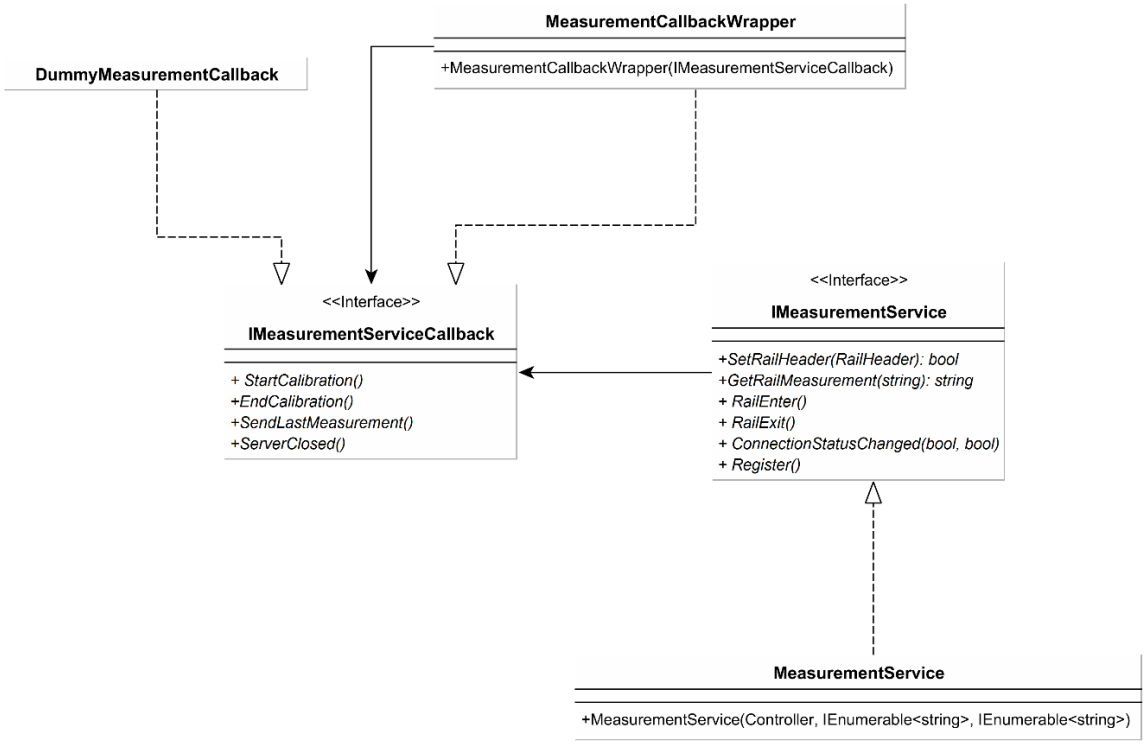

Figura 9: Diagrama de clases para la comunicación entre *RailMeter* y *RailCommunication*

3.4. Sistema de comunicaciones - *RailCommunications*

En este apartado se aborda la implementación del sistema de comunicaciones en sí. En el sistema de comunicaciones, la comunicación con cada equipo es responsabidad de una clase distinta.

En primer lugar, para el registro de mensajes de error e informativos se implementa una clase *Loggable*. Esta clase contiene eventos que se disparan cada vez que se registra un mensaje.

De la clase *Loggable* hereda la clase *Client*, que proporciona una interfaz común para iniciar y parar los clientes, y para que estos expongan el estado de la conexión.

Las clases *PAClient* y *PLCClient*, que heredan de *Client*, implementan los clientes para la comunicación con el ordenador de proceso y el PLC, respectivamente. Cada una de ellas proporciona métodos para enviar los mensajes que son competencia del cliente y eventos que informan de la llegada de mensajes recibidos del servidor.

La clase *MVCommunicationHandler*, que hereda directamente de *Loggable* (pues no tiene sentido que implemente algunos de los métodos de *Client*), consume los eventos producidos por *PAClient* y *PLCClient* y los traduce en llamadas a la interfaz WCF, y en el otro sentido recibe llamadas a través del callback WCF y las remite a los métodos correspondientes de *PAClient* y *PLCClient*. La estructura puede verse en la Figura 10.

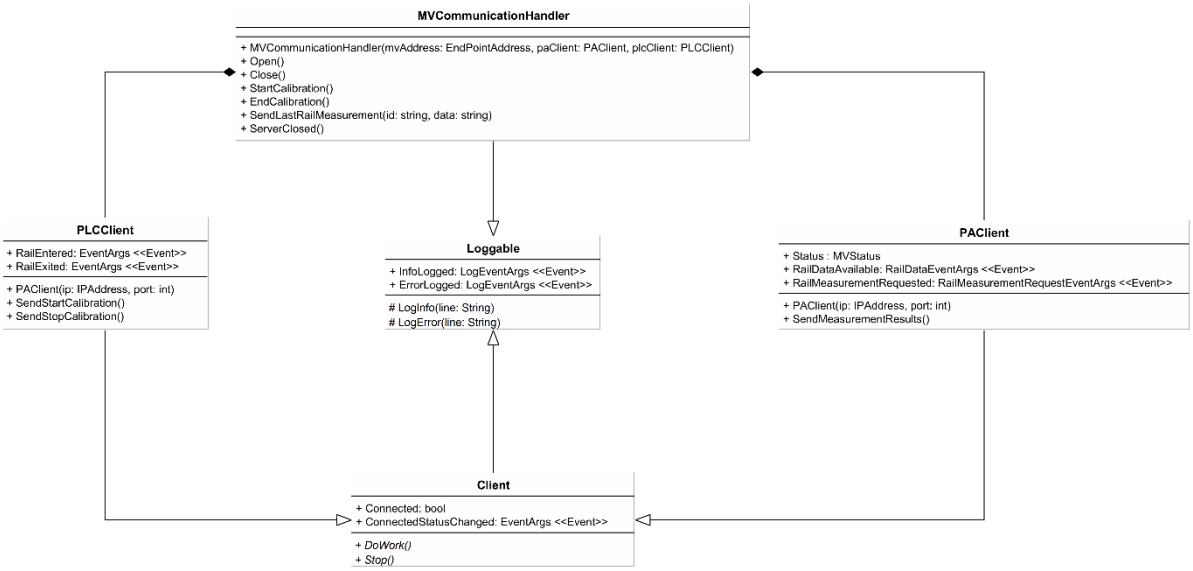

Figura 10: Diagrama de clases de RailCommunications

Para el tratamiento de la comunicación con el PA se emplean las mismas clases de mensajes que en la sección 2.1, y se aprovecha la implementación descrita en la sección 3.1.1.