MASTER UNIVERSITARIO EN
SISTEMAS FERROVIARIOS

UNIVERSIDAD PONTIFICIA COMILLAS

ICAI – MÁSTER EN SISTEMAS FERROVIARIOS
CURSO 2015 – 2016. MADRID

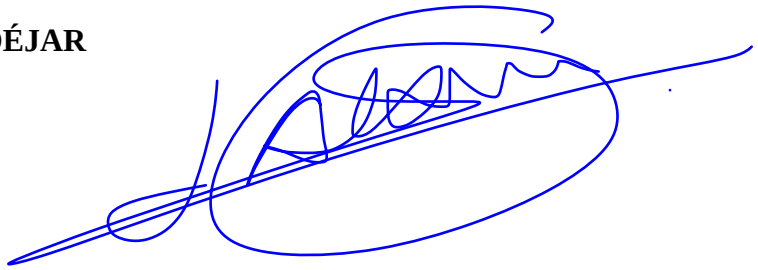
SIMULADOR DEL PROTOCOLO 50463 PARA LA COMUNICACIÓN ENTRE CONTADORES DE ENERGÍA DE TREN Y SISTEMAS EN TIERRA

Autor:

FERNANDO DORADO RUANO

Director:

JOSÉ ANTONIO RODRÍGUEZ MONDÉJAR



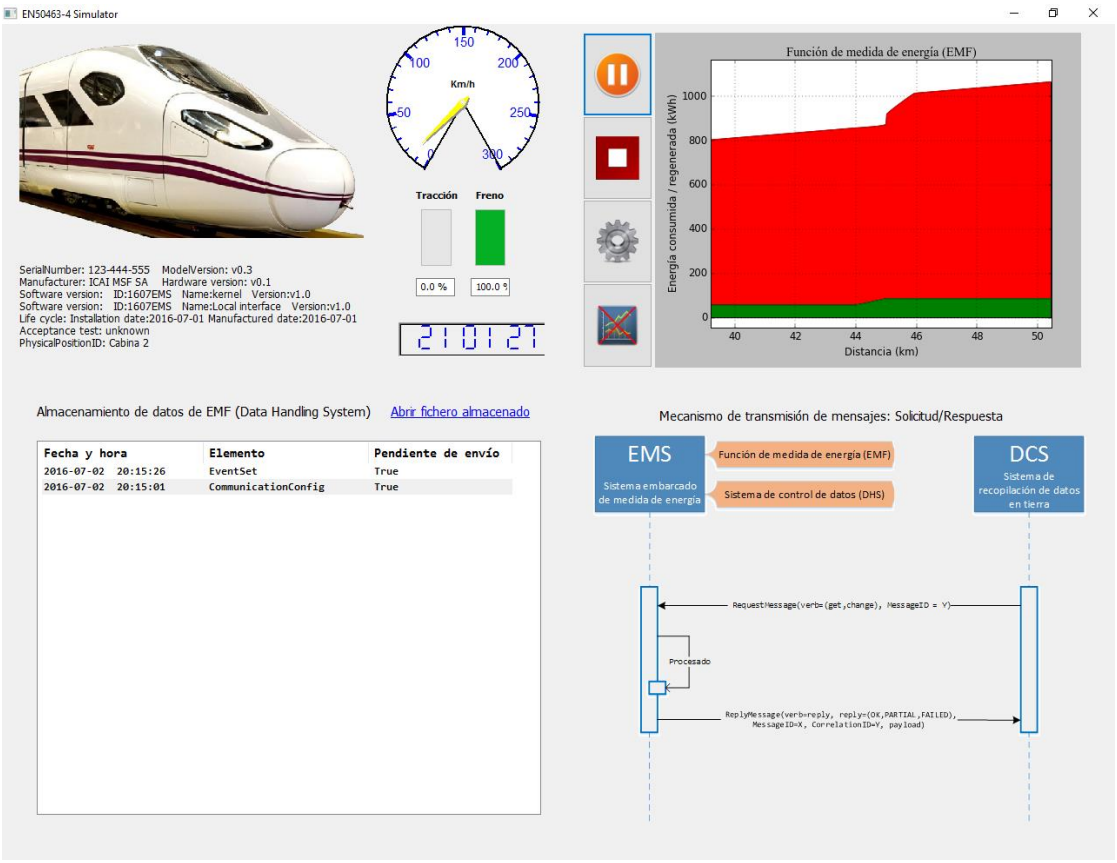
Resumen

La futura norma EN 50463, actualmente en fase de desarrollo por el WG11 del TC9X de CENELEC, define las características y funcionalidades que se le deberán atribuir al sistema de medida de energía a bordo de los trenes. En su apartado 4, se describen detalladamente las funcionalidades e implementaciones necesarias del módulo destinado a la comunicación de este sistema.

En el presente trabajo se ha desarrollado un simulador en el que visualizar el proceso de comunicación entre los contadores de energía del tren con los equipos de adquisición de datos en tierra, emulando los módulos necesarios de ambos sistemas y permitiendo interactuar al usuario del simulador con el fin de analizar y realizar el seguimiento de procesos, eventos y mensajes del intercambio.

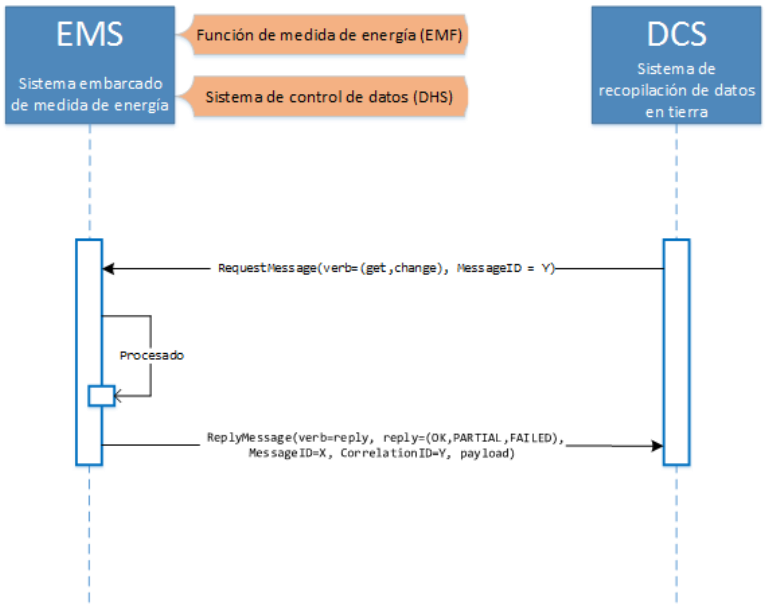
Tanto las estructuras de datos como los mensajes se forman y almacenan en el formato XML, obedeciendo al estándar propuesto. Se permite al usuario realizar operaciones de modificación de la configuración en el módulo de medida de energía del tren, y lanzar simulaciones de funcionamiento en modos degradados, como en el caso de una interrupción imprevista de comunicación.

La simulación se basa en el recorrido de un tren que va almacenando datos sobre su consumo energético y la devolución de energía a la red según transcurren los segundos. Se ha implementado sobre esa base la lógica y elementos del sistema de medida de energía EMS, en particular el módulo de gestión de los datos almacenados. La interfaz gráfica de la ventana principal se muestra en la página siguiente.



Paralelamente, se ha creado el sistema de adquisición en tierra, que podrá conectarse remotamente al programa principal, introduciendo usuario y contraseña. Desde este terminal se enviarán solicitudes y se recibirán respuestas y mensajes espontáneos con los datos medidos de energía.

Finalmente se ha simulado la conexión del equipo de adquisición de datos (DCS) al equipo embarcado (EMS) y se ha programado la lógica de intercambio de información.



Las aplicaciones informáticas se han construido en el lenguaje de programación Python (versión 2.7), siguiendo la metodología de programación orientada a objetos.

Índice de la memoria

Índice de figuras

Índice de tablas

Parte I MEMORIA

Capítulo 1 INTRODUCCIÓN

Uno de los ámbitos que se encuentra en una posición muy relevante en la operación del transporte es el consumo de energía. La capacidad de captación, envío y análisis de datos puede resultar muy útil en este campo, bien para implantar métodos de conducción eficiente o detección de potenciales problemas.

Por otro lado, los datos de energía tanto consumida como regenerada por los trenes eléctricos pueden ser empleados para realizar la facturación por parte de la empresa suministradora de electricidad.

La norma EN50463, actualmente en fase de desarrollo por el WG11 del TC9X de CENELEC, referente a la medida de energía a bordo en trenes define numerosos aspectos que han de cumplir los sistemas dedicados a este campo: captación de datos, cálculos, procesos y comunicaciones desde el punto de vista del tren y de los equipos en tierra.

Este trabajo tiene el propósito de simular, mediante la programación de una herramienta informática, las principales funcionalidades de los sistemas dedicados a la comunicación, empleando los mismos protocolos y estructuras para el envío de información que se detallan en el estándar en fase de desarrollo de la EN50463-4.

El alcance de la norma abarca, por supuesto, los requisitos de los servicios de comunicación de datos entre los módulos de los sistemas definidos en el siguiente párrafo. Ofrece también la orientación necesaria para elaborar el simulador objeto del presente trabajo.

Para estructurar la aplicación informática, se ha concebido la comunicación entre el EMS (Energy Measurement System, Sistema de Medida de Energía), que se encuentra embarcado en el tren, y el DCS (Data Collection Service, Servicio de Recolección de Datos), como la conexión de un usuario cliente (DCS) al servidor de datos (EMS) sobre el que puede enviar peticiones para obtener los datos solicitados. Además de este método, la norma también considera la opción de que

el EMS transfiera automáticamente los datos al DCS cuando se encuentren disponibles.

Asimismo, se consideran en la simulación modos degradados, como cortes de comunicación o envío de mensajes corruptos.

El lenguaje de programación elegido para desarrollar la aplicación es *Python 2.7*, que es multiplataforma y está preparado para la programación orientada a objetos[1], que es la filosofía empleada para el desarrollo del simulador. Es un lenguaje interpretado, lo que significa que el código no necesita ser compilado[2] para ejecutarse, sino que va siendo traducido por el intérprete de *Python* a código máquina y de esta forma se ejecuta sobre la marcha.

La aplicación se ha desarrollado con el entorno de simulación *Eclipse SDK (Versión Mars.2, http://www.eclipse.org/)*, empleando el "plugin" disponible para Python: *Pydev*.

El presente documento está estructurado en los siguientes capítulos:

1. **Introducción**
2. **Objetivos del trabajo**
3. **Tareas**, donde se definen los hitos del trabajo, las diferentes actividades en que se desglosa el grueso de la herramienta bajo desarrollo.
4. **Planificación**, donde se define brevemente el marco temporal en que se decidieron desarrollar las diferentes actividades.
5. **Desarrollo**, donde se describen los procesos, problemas e implementaciones para la consecución de los hitos planteados en el capítulo de Tareas.

[1] La Programación Orientada a Objetos (POO) es una metodología de programación basada en objetos, no solamente funciones y procedimientos. Estos objetos se organizan en clases, y los objetos resultado de la instanciación de una clase tienen la misma estructura definida por esta misma, pero a cada uno se le pueden asignar características particulares.

[2] Compilar: convertir el programa en código fuente a código máquina.

6. **Conclusiones y aportaciones**, reflexiones sobre el trabajo realizado, las posibles utilidades del producto desarrollado, y la mención de los puntos más destacados.

Existe una Parte II donde se encuentran anexos dos apartados adicionales, el manual de usuario, donde se describe el proceso de instalación y uso básico de la herramienta, y

Capítulo 2 OBJETIVOS DEL TRABAJO

2.1 OBJETIVO GENERAL

El propósito de este trabajo es diseñar una herramienta informática donde se muestre el proceso de comunicación basado en el estándar propuesto por la futura norma EN50463 de los sistemas contadores de energía de tren y los equipos en tierra para recepción y almacenamiento de datos.

2.2 OBJETIVOS PARTICULARES

En la primera fase, el tema principal es simular el sistema de medida de energía por parte de los contadores del tren a lo largo de un determinado recorrido, para obtener los datos necesarios del EMS. El EMS está constituido por el EMF (Energy Measurement Function, Función de Medida de Energía) y el DHS (Data Handling System, Sistema de Administración de Datos). En esta primera fase se simularía mediante la lectura de un archivo de datos, la funcionalidad del EMF.

Tras obtener esta información, por ejemplo, en kWh consumidos en un breve periodo de tiempo, el EMS debe recopilar esta información en unidades de tiempo mayores y en un formato que pueda adjuntarse a un mensaje según lo especificado por el estándar.

El siguiente objetivo es implementar la conexión entre servidor y cliente, empleando uno de los protocolos propuestos en la norma (XML-RPC, FTP con mailbox o bien HTTP).

Tras la conexión servidor – cliente, se va a tratar de implementar el mecanismo de solicitud de datos / respuesta para la obtención de datos del EMS desde el DCS mediante el "log in" de un usuario.

A medida que avanzan las fases, la interfaz gráfica debe permitir al usuario interactuar con el programa sin necesidad de modificar el código fuente para conseguir los objetivos principales.

Los propósitos claves, llegado este punto, serán:

- Implementación de los diferentes servicios de comunicación disponibles:
  - **CEBDBlock**: datos compilados de energía para facturación,
  - **ReadingBlock**: lecturas detalladas de energía, velocidad y posición del tren,
  - **EventSet**: conjunto de eventos detectados en el EMS ("login" de usuarios, encendido, alarmas, peticiones fallidas), en general, información que puede solicitarse desde el usuario del DCS.
  - **CommunicationConfig**: configuración de los parámetros del EMS (usuarios registrados en la base de datos, derechos de acceso de los mismos, ajuste de la generación de datos compilados, tiempos máximos de respuesta)
- Creación de la lógica para la identificación de peticiones erróneas, servicios no disponibles para el usuario, validación de los mensajes contra los esquemas XSD.
- Ofrecer la posibilidad de cambiar la configuración de la generación de datos y la comunicación desde el puesto del EMS (cantidad máxima de lecturas o unidades simples generadas de un determinado servicio a incluir en bloques superiores para envío de información, cambio de derechos de acceso)
- Gestión de permisos del usuario del DCS sobre cambios de configuración del EMS. ("**ChangeCommunicationConfig**", Servicio de cambio de configuración de la comunicación)
- Visualización de los mensajes transmitidos entre EMS y DCS, de los bloques generados por el EMS

Capítulo 3 TAREAS

Para la consecución de los objetivos particulares descritos, será necesario cumplimentar las siguientes tareas:

1. Creación de proyecto *Pydev* en *Eclipse*. Configuración del entorno, instalación de los módulos necesarios para el intérprete de Python.
2. Construcción del fichero de datos de energía y volcado de los datos al programa.
3. Implementación de bucle de simulación y realización de lecturas de los datos. Creación del módulo EMS embarcado al que se asocian las lecturas.
4. Creación de interfaz gráfica que permita al usuario lanzar el bucle de simulación y visualizar los datos de energía consumida, regenerada, el tiempo simulado (y opcionalmente otras variables como tracción aplicada o velocidad del tren).
5. Compilación de datos en estructuras de tipo Block (texto plano en formato XML [3]que puede adjuntarse a un mensaje para enviarse al DCS cliente). Validación de los Blocks.xml contra los esquemas ".xsd" definidos en el estándar. Almacenamiento de los Blocks en la base de datos del EMS: el DHS (Sistema de manejo de datos).
6. Elaboración del programa para simular el terminal del cliente DCS en tierra.
7. Elaboración de los módulos de comunicación correspondientes para conectar el EMS (Servidor) y el DCS (Cliente).
8. Implementación de la lógica del programa de cliente, elaboración de controles para hacer solicitudes al EMS (requests) y cuadros para la visualización de los eventos de la comunicación.
9. Formación, envío y visualización de requests con formato XML al EMS desde el DCS.

[3] XML (Extensible Markup Language): es un lenguaje de marcas desarrollado por el World Wide Web Consortium utilizado para almacenar datos en forma legible.

10. Interpretación y procesamiento de los requests del DCS por parte del EMS.
11. Implementación de la lógica de detección de errores en el request y selección, en caso de ausencia de errores, de los Blocks necesarios del DHS y envío de la información solicitada.
12. Envío de respuestas al cliente con los datos solicitados o con información sobre el error en su caso.
13. Gestión de la configuración en el EMS. Manipulación de parámetros de compilación de datos.
14. Implementación del método de comunicación de transferencia automática de eventos: al crearse un Block, se envía directamente al usuario conectado, sin necesidad de realizar la petición.
15. Elaboración de controles y lógica para programar interrupciones de comunicación entre EMS y DCS.
16. Lógica de la base de datos del EMS (DHS) ante cortes de comunicación, puesta de mensajes en cola de envío. Lógica del DCS en tierra ante cortes de comunicación, reintentos de conexión.
17. Prueba de funcionalidad ante corrupción de mensajes: simulación de pérdida y/o alteración de la información en un request del DHS cliente e interpretación adecuada del EMS.
18. Pruebas funcionales del sistema.

Capítulo 4 PLANIFICACIÓN

Primera quincena de mayo

Lectura y análisis del avance de la norma EN50463-4. Identificación de las secciones de interés, requisitos obligatorios para el sistema a simular y funcionalidades opcionales susceptibles de incluirse en el desarrollo.

Elección del protocolo de transmisión de mensajes.

Segunda quincena de mayo

Preparación del entorno en Eclipse: descarga e instalación de módulos necesarios para la interfaz gráfica, el tratamiento de los datos, gestión de la multitarea, gestión de XML (creación, adición, lectura, validación contra esquemas).

Primera quincena de junio

Estructuración modular del proyecto, división de scripts por funcionalidad. Planteamiento de jerarquía de directorios: carpeta de datos, de archivos salida, de herramientas complementarias.

Elaboración de los módulos de tren (EMS y bucle de simulación) y tierra (DCS usuario). Construcción de ficheros de datos, lectura. Elaboración de la interfaz gráfica

Segunda quincena de junio

Instalación de módulo para el manejo de la comunicación cliente – servidor con el protocolo empleado. Conexión de ambos programas. Modo de comunicación *Request/Reply* y *Automatic Transfer*. Modos de funcionamiento degradado (cortes de comunicación, mensajes corruptos). Pruebas funcionales de conjunto.

Capítulo 5 DESARROLLO

5.1 REFERENCIA PRINCIPAL

La referencia principal en que se basa el desarrollo del simulador es el documento de la propuesta de norma EN 50463-4 a fecha de 7 de abril de 2016 (documento confidencial no publicado), se encuentra actualmente en fase de desarrollo por el WG11 del TC9X de CENELEC.

5.2 ESTRUCTURA OPERATIVA DEL PROYECTO

Tras la creación del proyecto Pydev en Eclipse, que se llamará EMBT_Comm_Simulator (simulador de comunicaciones, medida de energía a bordo en trenes), se crea un primer script (la extensión de los scripts de Python es ".py") con nombre *FRAME.py*. Este script será el principal, es el que invocará a todos los demás scripts, exceptuando a los pertenecientes al programa que lanza la terminal del DCS. Se encontrará en el directorio principal del proyecto.

Este directorio principal tendrá tres subdirectorios:

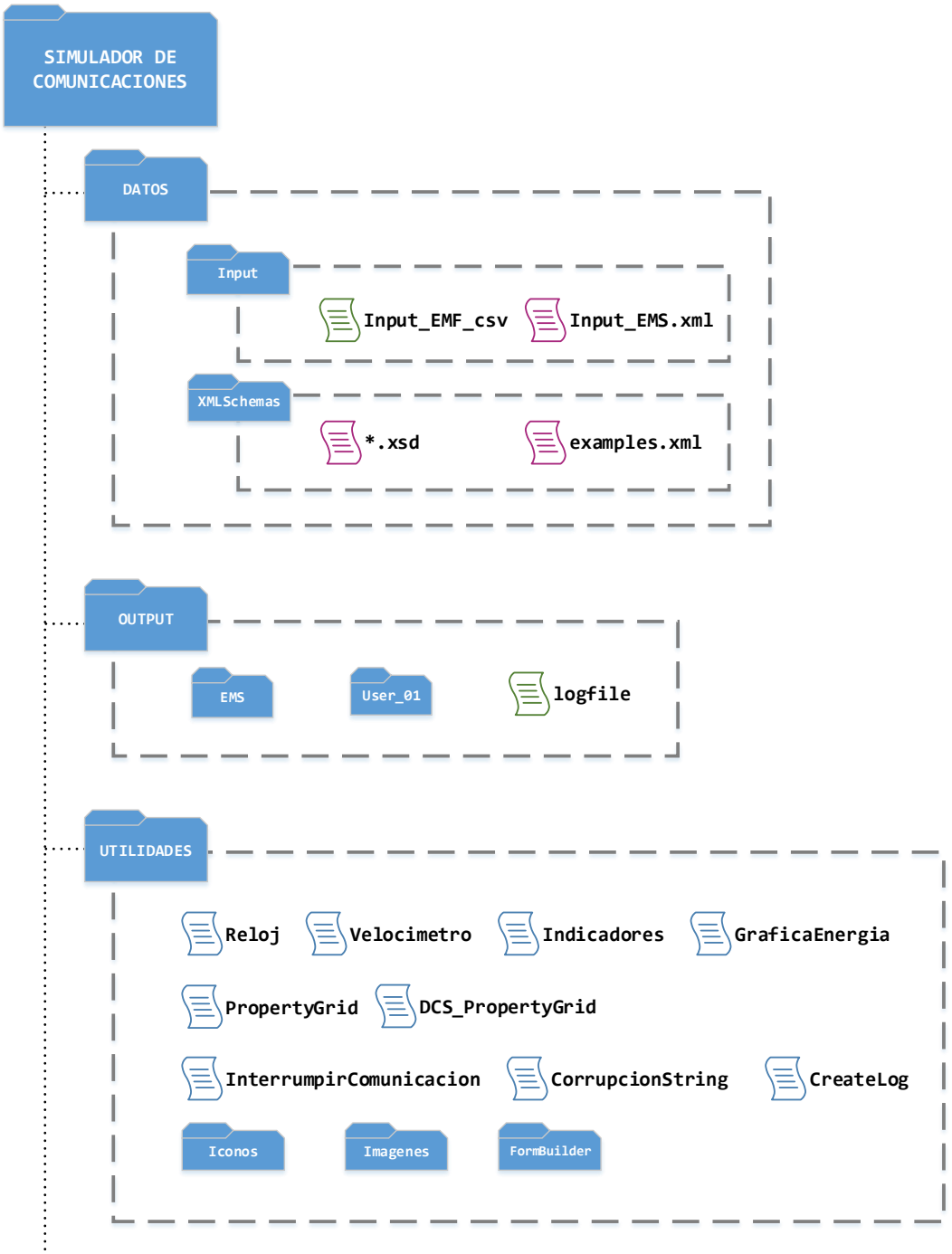
- *DATOS*, contiene dos carpetas: *Input*, donde se almacena el archivo csv con los datos de energía de un recorrido; y *XMLSchemas*, donde se encuentran los esquemas XSD necesarios para construir y validar los XML que se elaboren y que se hayan de atener al estándar de referencia.
- *UTILIDADES*, contiene scripts que definen clases o funciones complementarias para los métodos principales de la aplicación. Además, contiene a su vez los directorios donde se guardan las

imágenes y los iconos que serán mostrados en pantalla, así como los archivos generados con una herramienta denominada wxFormBuilder, para la construcción de ventanas de la interfaz.

- *OUTPUT*, será la carpeta de salida donde se guardarán a medida que sean creados los mensajes transferidos en la comunicación, los Blocks compilados en el EMS, y un log que registra las conexiones del DCS al servidor EMS.

En este directorio se encuentran también, como se ha mencionado, los scripts principales de la aplicación. Puede observarse en la Figura 1. Cada script agrupa una serie de funcionalidades, de modo que se trata de conseguir una estructura modular que permita identificar los distintos métodos y objetos dentro de la aplicación.

En la Figura 2 puede observarse la relación funcional entre scripts, las invocaciones entre unos y otros y el nexo de unión. Se aprecia la existencia de dos ramas diferentes, aquella cuyo padre es el script FRAME.py, que controla la simulación del recorrido del tren con las medidas de energía, y los procesos internos del EMS; y la otra rama cuyo script principal es DCS.py, que lanza la interfaz del usuario desde el puesto de tierra.


SIMULADOR DE COMUNICACIONES
DATOS
Input
Input_EMF_csv
Input_EMS.xml
XMLSchemas
*.xsd
examples.xml
OUTPUT
EMS
User_01
logfile
UTILIDADES
Reloj
Velocimetro
Indicadores
GraficaEnergia
PropertyGrid
DCS_PropertyGrid
InterrumpirComunicacion
CorrupcionString
CreateLog
Iconos
Imagenes
FormBuilder

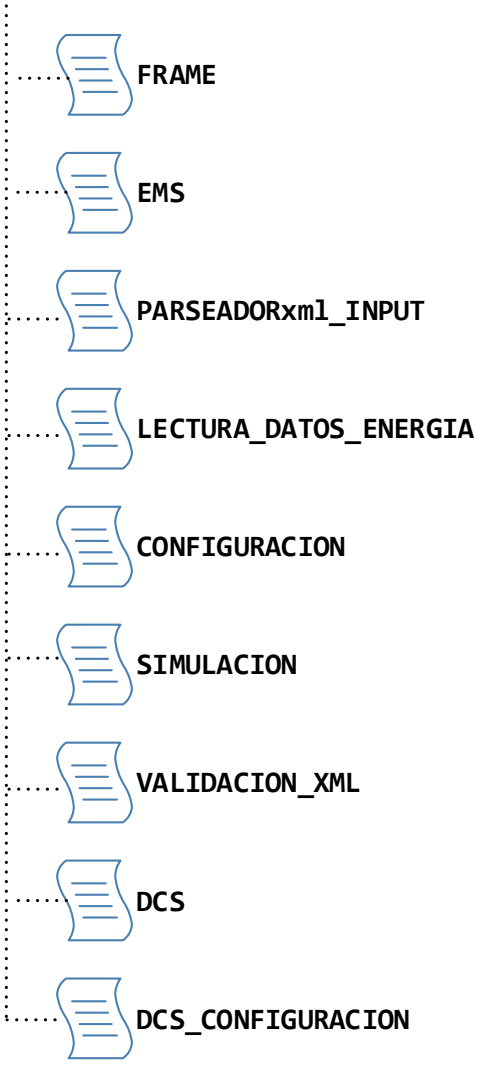


Figura 1: Estructura de carpetas del proyecto en el entorno de programación

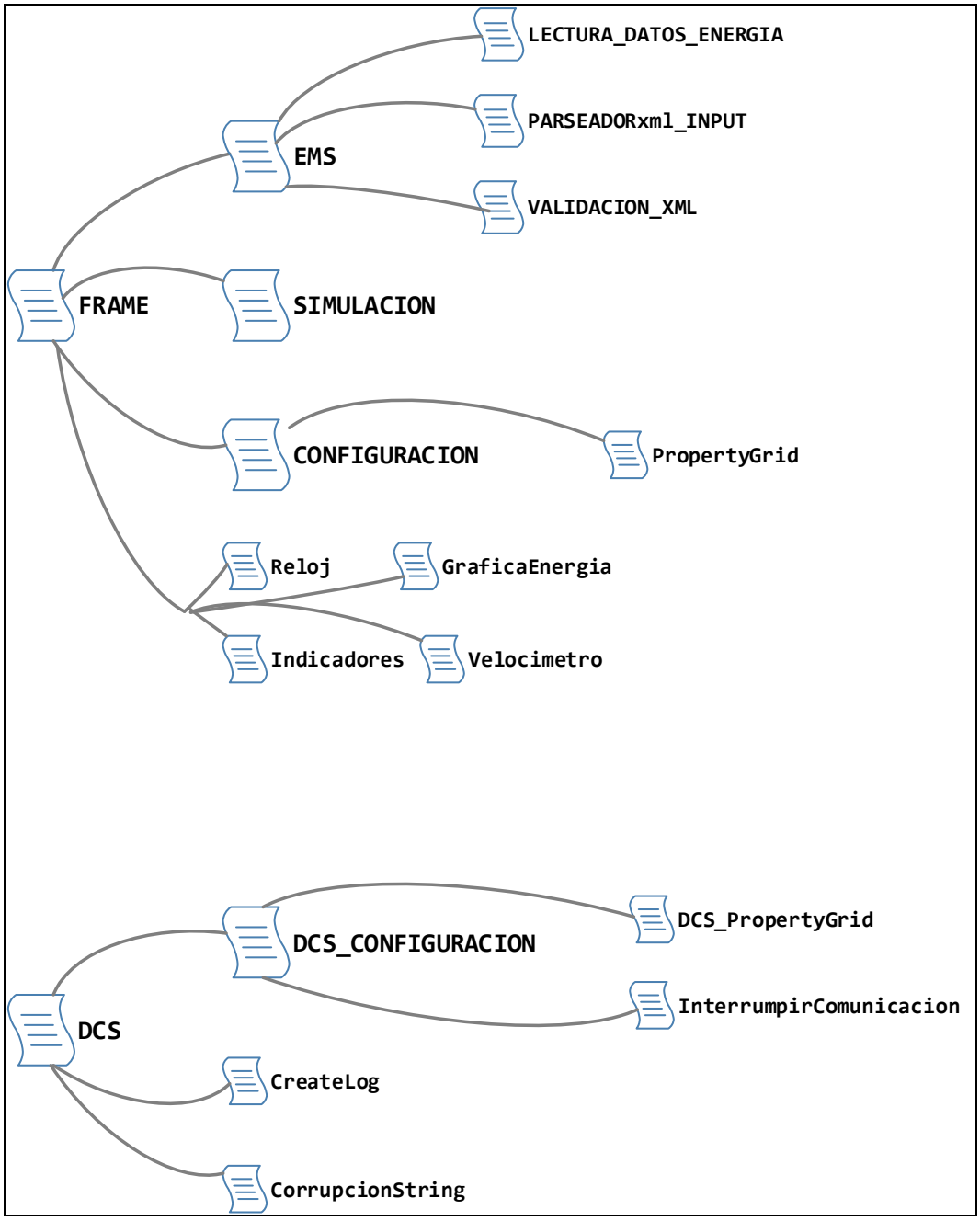


Figura 2: Estructura funcional. Llamadas entre scripts en tiempo de ejecución

5.3 DESARROLLO DE LAS TAREAS PROGRAMADAS

5.3.1 CONFIGURACIÓN DEL ENTORNO E INSTALACIÓN DE MÓDULOS

El entorno de programación utilizado es Eclipse SDK 4.5.2. (Versión Mars.2). Es portable, no necesita instalación. Es necesario incorporar el "plugin" Pydev para programación en Python, disponible en su página web:

http://www.pydev.org/download.html

Se configura el intérprete de Python 2.7.6:

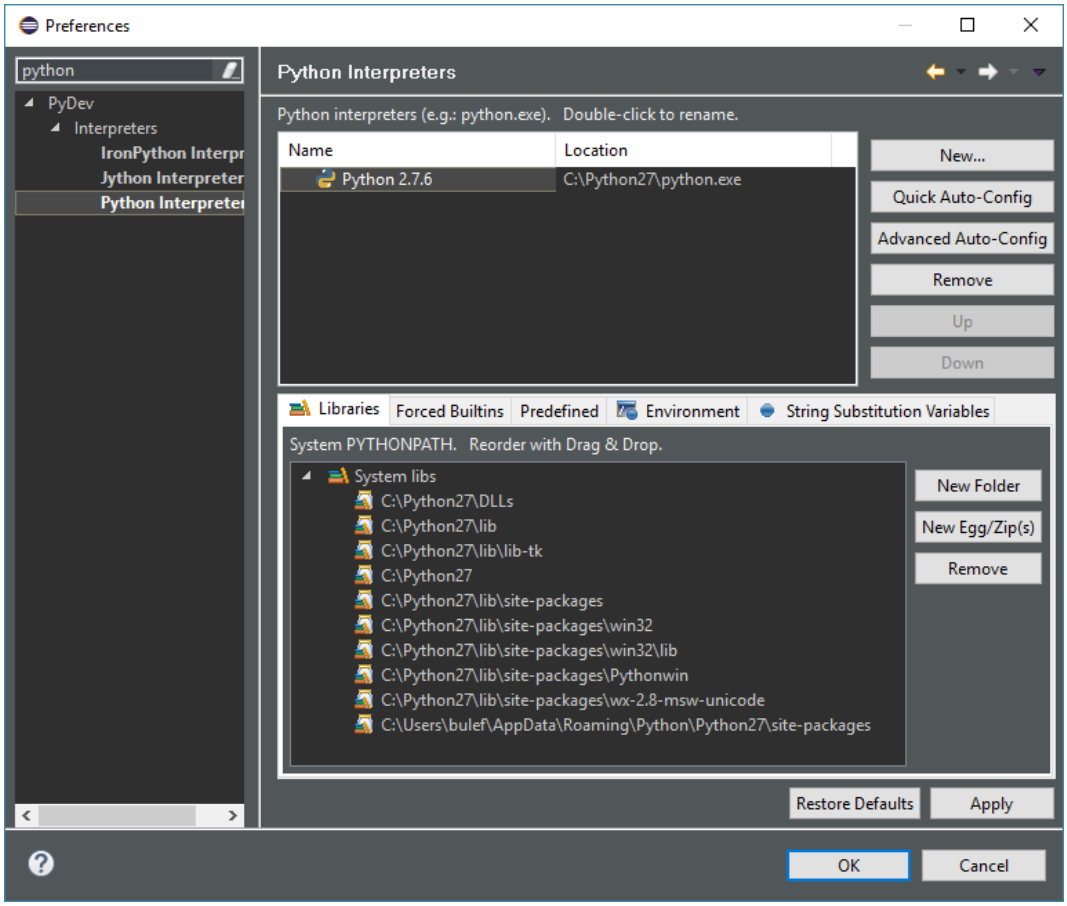


Figura 3: Configuración del intérprete y librerías necesarias de Python 2.7 en Eclipse

Y posteriormente se crea el proyecto. El formato de codificación de caracteres será UTF-8 para todos los archivos.

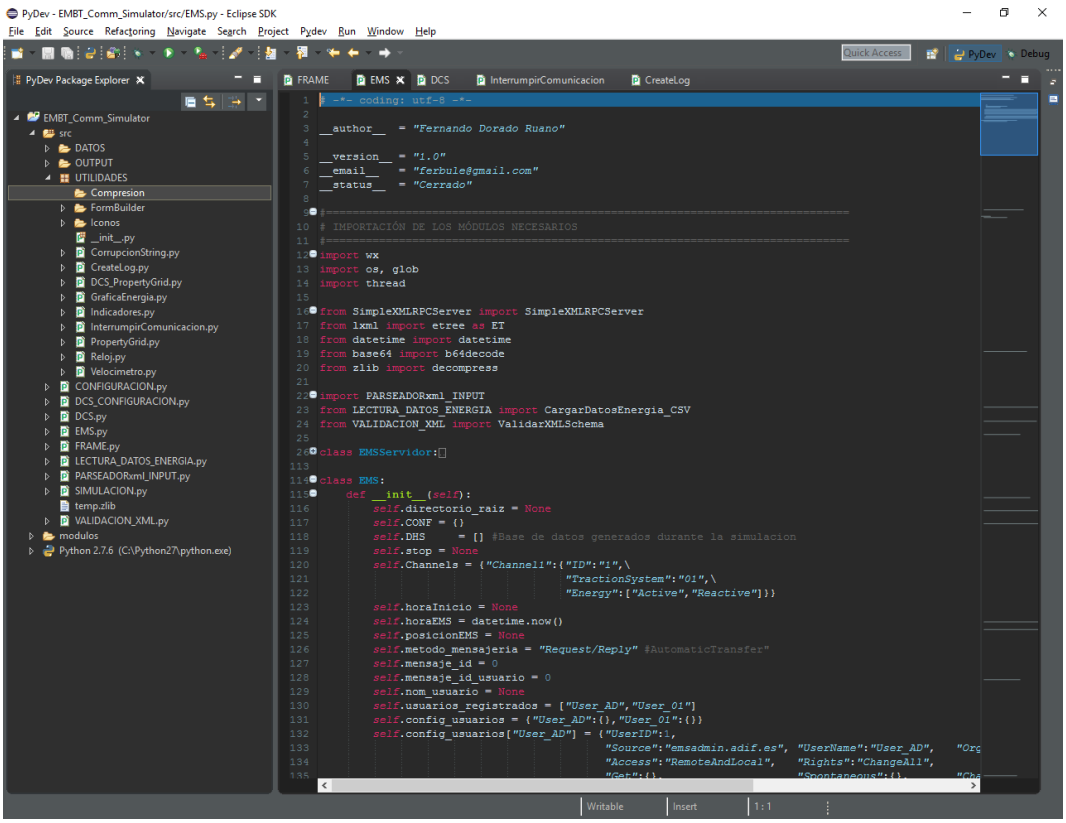


Figura 4: Ventana principal de Eclipse

Las librerías de Python principalmente empleadas a lo largo del proyecto han sido:

Tabla 1: Módulos principales utilizados en el proyecto

Nombre del módulo	*Descripción*
wx	wxPython, desarrollo de la interfaz gráfica
os, glob	Controles del explorador de Windows, creación y eliminación de directorios y archivos
sys	Control de funciones del sistema
thread	Multitarea, lanzamiento de hilos de ejecución paralelos
lxml	Manejo de XML (elaboración XML, traducción string – XML, validación contra XSD)

SimpleXMLRPCServer	Servidor XML-RPC para comunicaciones
base64, zlib	Compresión y codificación de texto y archivos
time, datetime	Manejo de objetos con formato de fecha y hora
matplotlib	Creación de gráficas

Alguno de los módulos, como SimpleXMLRPCServer, han debido instalarse manualmente mediante la herramienta *pip*, pues no se encontraban en la distribución de Python(x,y) utilizada.

5.3.2 CONSTRUCCIÓN DEL FICHERO DE DATOS DE ENERGÍA Y OBTENCIÓN DESDE LA APLICACIÓN

Para elaborar el fichero de datos de energía se ha partido de un libro Excel que contiene los resultados de un trabajo previamente desarrollado para una de las materias cursadas en el Máster, en el que se definían como datos de partida las características de la infraestructura y se proponía un tren para cubrir un determinado tipo de servicio. Conocidas la potencia, masa total, masa adherente del tren, adherencia disponible y resistencias al avance: se obtenía una marcha tipo con los valores calculados de velocidad, tiempo, aceleración en cada punto discreto del recorrido y energía en llanta necesaria.

Con las transformaciones precisas y la suposición de unos determinados valores de rendimiento, se muestran en el libro Excel los resultados de energía activa, reactiva, consumida y regenerada por el tren en cada punto.

Para cada 20 m del recorrido, se conoce un valor para los siguientes parámetros:

- Punto kilométrico, límite de velocidad, gradiente, curvatura.
- Fuerza resultante, esfuerzo de tracción, esfuerzo de frenado, resistencias al avance, aceleración, velocidad, tiempo empleado.

- Consumo de energía debido al esfuerzo de tracción, a los servicios auxiliares (kWh), al esfuerzo de frenado.
- Energía instantánea importada en pantógrafo, activa, y reactiva. (kWh)
- Energía total importada en pantógrafo, activa y reactiva. (kWh)
- Energía instantánea regenerada, activa y reactiva. (kWh)
- Energía total regenerada en pantógrafo, activa y reactiva. (kWh)

Para importar estos datos al programa en su ejecución, se han pasado los datos necesarios a un archivo `csv`. En el método `LECTURA_DATOS_ENERGIA.py` se obtiene el `csv` con el método de `numpy loadtxt` y se asigna cada parámetro a un vector ordenado.

En cuanto al programa, se inicia con el script principal `FRAME.py`, que lanza la siguiente ventana:

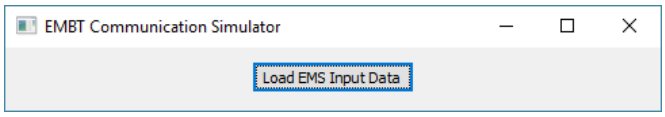


Figura 5: Ventana inicial de FRAME.py: carga de los datos del EMS

El botón *"Load EMS Input Data"* invoca al script `EMS.py`, que crea una instancia de la clase EMS. En el método de inicialización de este objeto, se invoca al script `PARSEADORxml_INPUT.py`, que localiza en el directorio el fichero `./DATOS/Input/EMS_Input.xml`, obtiene los datos del archivo y los añade como atributos al objeto EMS recién creado. En caso de no existir el fichero o contener errores, se crea un log con el problema encontrado y se termina la ejecución del programa.

El fichero de datos de entrada del EMS es el siguiente:

```
<?xml version='1.0' encoding='UTF-8'?>
<EMS>
	<CPID>
		<NVR>ES</NVR>
		<VKM>RENFE</VKM>
		<EVN>123456789012</EVN>
		<EMSID>1</EMSID>
	</CPID>
	<TRP>PT5M</TRP>
```

```
    <XMLVersion>1.0</XMLVersion>
</EMS>
```

El proceso de obtención de los datos de este archivo XML se realiza con el módulo `lxml` de Python, y ha servido como ensayo inicial para posterior "parseo"[4] de XML.

Tras la obtención de los parámetros de entrada fijos, como el EVN (European Vehicle Number) o el VKM (Vehicle Keeper Marking), el objeto EMS realiza la llamada a la función `CargarDatosEnergia_CSV` del script `LECTURA_DATOS_ENERGIA.py` que asigna los vectores de datos de energía, tiempo y posición a variables propias del objeto EMS.

5.3.3 BUCLE DE SIMULACIÓN

Tras la lectura de los datos de energía y su asignación al EMS, se abre la una ventana, programada en `FRAME.py`, donde aparece un botón que permite invocar al método `Simulacion` del script `SIMULACION.py`. Este método, está conectado con el `Frame` principal que a su vez está relacionado con el objeto `EMS`, por lo que el método `Simulacion` puede acceder tanto al `Frame` como al `EMS`.

En este método, se obtiene la hora actual del sistema y se lanza un bucle `for`, con una variable `i` que se va incrementando en una unidad desde 0 hasta el número de segundos del tiempo del recorrido ya determinado en el fichero de datos `Input_EMF_csv.csv`.

Por cada iteración, se incrementa la hora del sistema obtenida justo antes de comenzar la simulación en un segundo.

Conociendo el valor de la variable i que lleva la cuenta de iteraciones (segundos desde el inicio del viaje), se puede buscar en el vector de medida de tiempos del EMS cargado del fichero `Input_EMF_csv.csv` el índice del vector

[4] Parsear: analizar, administrar la información contenida en un archivo o registro.

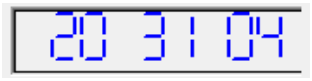
correspondiente. Es decir, si tenemos el siguiente vector de tiempos (valores en segundos): `(0.5, 3.5, 5.0, 10.1)` y la cuenta de segundos es 3, sabremos que el valor de tiempo se encuentra entre los índices [0] y [1] del vector. Si comprobamos el valor de energía total en estos índices de los vectores de energía correspondiente y realizamos una interpolación, obtendremos la energía total consumida (o regenerada) a los 3 segundos del recorrido.

Este es el método para el cálculo de energía dado un instante de tiempo.

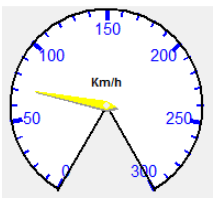
5.3.4 ELABORACIÓN DE INTERFAZ GRÁFICA Y VISUALIZACIÓN DE LOS DATOS EN LA SIMULACIÓN

Para que el usuario pueda seguir la simulación y observar gráficamente los valores leídos en cada iteración, se han implementado los siguientes elementos en la interfaz:

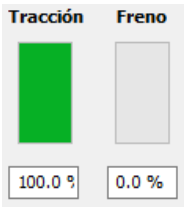
- Un reloj digital, que da la hora actual en la simulación (hora de inicio de la misma más el número de segundos, que coincide con el número de iteraciones realizadas. Este elemento se ha obtiene del script "`.\UTILIDADES\Reloj.py`", mediante un submódulo de la librería `wxPython` denominada `gizmos`.



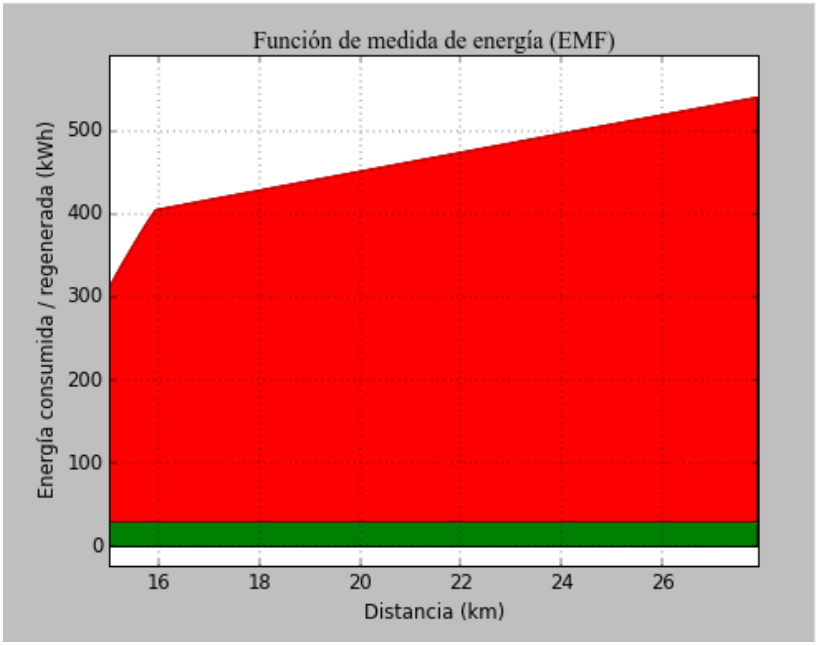
- Un velocímetro de aguja, expresa la velocidad del tren en cada segundo del recorrido (la velocidad se obtiene del fichero de datos, sabiendo los segundos transcurridos en el viaje). Este elemento se ha obtiene del script "`.\UTILIDADES\Velocimetro.py`".



- Indicadores del porcentaje de tracción y freno utilizado, para observar la correspondencia con el consumo o regeneración de energía (se obtiene directamente del fichero de datos). Este elemento se ha obtiene del script "`.\UTILIDADES\Indicadores.py`".



- Una gráfica que se actualiza cada cierto tiempo con las cantidades de energía total consumida (en rojo) y regenerada (en verde)



En la Figura 6 puede observarse la posición que ocupan estos elementos en la ventana principal de la interfaz del simulador y EMS.

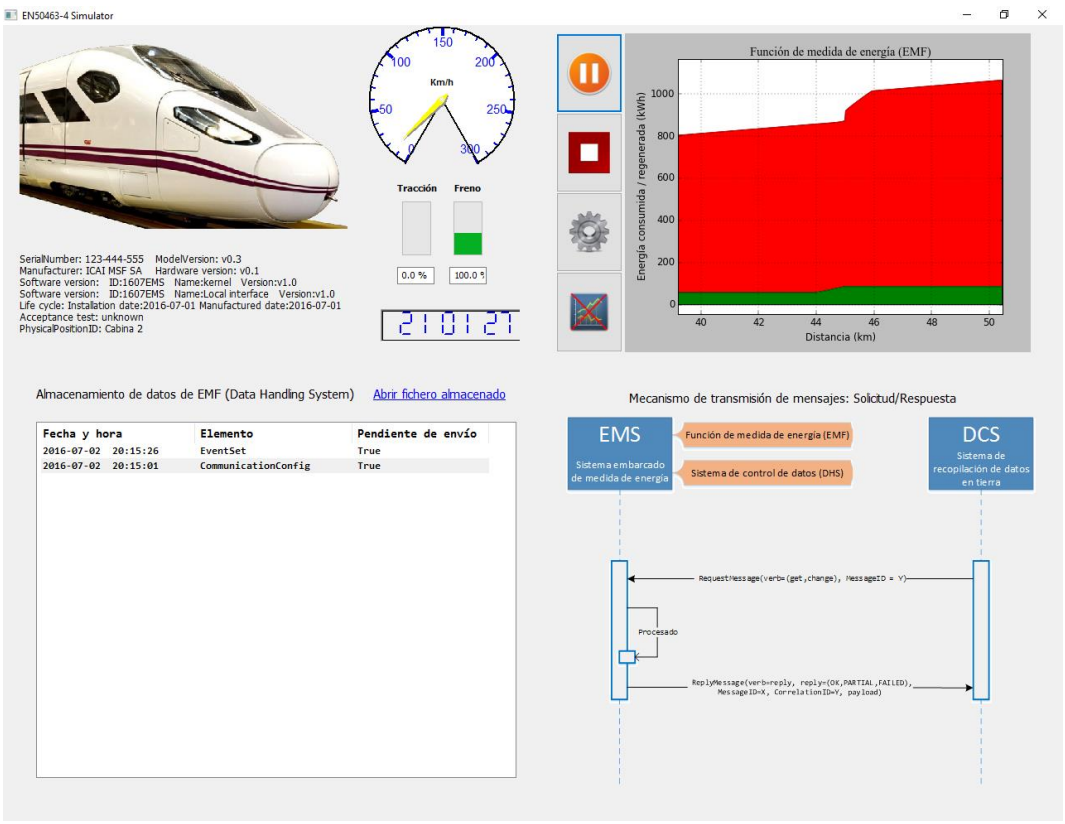


Figura 6: Interfaz gráfica del simulador y EMS

5.3.5 ALMACENAMIENTO DE DATOS EN ESTRUCTURAS XML

La generación de datos durante la simulación han de almacenarse en unas estructuras definidas por la norma de modo que puedan adjuntarse al mensaje tipo del servicio que el cliente solicite.

El propósito de esta agrupación de datos es mejorar la manejabilidad de los datos e incrementar la rapidez de transmisión, a la hora de reunir los datos solicitados en tierra y ser enviados por el EMS embarcado.

Los dos servicios que aportan información sobre la cantidad de energía, y en general, sobre las medidas cuantitativas de las variables del tren, son:

- **CEBDBlock**, estructura que reúne los datos de energía compilados para facturación. Cada CEBDBlock está formado por una

información fija sobre el EMS y una secuencia de CEBDs resultado de las mediciones del EMF. Cada CEBD contiene información sobre la energía activa y reactiva tanto consumida como devuelta a la red durante un periodo de tiempo que se ha fijado en 5 minutos. Un ejemplo de CEBDBlock puede encontrarse en la página 61 (Parte II - 2.1.1).

- **ReadingBlock**, constituido por datos de energía y velocidad en periodos de tiempo más cortos que el CEBDBlock (lecturas de 1 minuto frente a 5 del CEBD). Pueden utilizarse para realizar estudios de conducción eficiente. Cada ReadingBlock está constituido por una serie de "Readings", los cuales a su vez contienen el resultado de las mediciones para periodos de tiempo fijado en 60 segundos. Un ejemplo de ReadingBlock con sus Readings anidados puede encontrarse en la página 63 (Parte II2.1.2Parte II - 2.1.2).

Para construir las citadas estructuras XML, se ha empleado el módulo `lxml` para Python.

5.3.5.1 CEBDBlock

Para formar el CEBDBlock, primero es necesario tener los CEBD suficientes. De este modo, sabiendo que el tiempo de medida de un CEBD es de 5 minutos, evaluamos el instante en que transcurre ese tiempo de viaje e invocamos a una función, denominada `FormarCEBD`. Esta función toma los tiempos de inicio y final del intervalo de interés, y evalúa la diferencia de energía total consumida/regenerada y activa/reactiva en ambos instantes (información que es contenida en los vectores de datos de energía del EMS previamente cargados).

La formación del texto XML se realiza de la siguiente manera:

1. Creación del elemento padre o raíz del XML ("tag": "CEBD" en este caso):

```
CEBD_obj = ET.Element("CEBD")
```

`CEBD_obj` es el nombre del objeto raíz, `ET` es el submódulo de `lxml` para la creación de estructuras y `Element` es el método para crear el elemento padre. Entre paréntesis, el nombre del elemento (tag).

2. Creación de subelementos que se añadirán al elemento padre:
   ```
   CEBDID = ET.SubElement(CEBD_obj, "CEBDID")
   ```

   `CEBDID` es el nombre del objeto subelemento, `SubElement` es el método para crear dicho subelemento. Entre paréntesis, como primer argumento, el objeto al que se anidará el subelemento, que en este caso será el elemento padre `CEBD_obj`, y como segundo argumento del método, el tag del subelemento. De la misma manera, pueden añadirse elementos anidados a otros.

3. Adición del texto al campo creado correspondiente.

   Por ejemplo, para el subelemento recién creado `CEBDID` (número identificativo del CEBD), se le asigna dicho número que será una variable de tipo contador (`id_cebd`) que se va incrementando en una unidad a cada formación de un CEBD.

   ```
   CEBDID.text = str(id_cebd)
   ```

Con este mecanismo, se crean los elementos presentados en la Tabla 2:

Tabla 2: Contenido de CEBD

Objeto	*Objeto padre*	*Descripción, forma de obtención u objetos anidados*
`CEBD`	**`CEBDBlock`**	**`CEBDID`**, **`TimeStamp`**, **`TimeStampQuality`**, **`Channel`**, **`Location`**
`CEBDID`	**`CEBD`**	Número para la identificación unívoca del CEBD. Contador se incrementa en una unidad por cada llamada a la función `FormarCEBD.`
`TimeStamp`	**`CEBD`**	Hora de creación. Se conoce la hora de la simulación por la iteración del bucle principal.
`TimeStampQuality`	**`CEBD`**	Incertidumbre del dato **`TimeStamp`** (127 para medición sin errores). Dato fijo.

Channel	**CEBD**	Canal de medición, el tren puede tener varios canales de medida, en función de la disposición de los motores, o la distinción entre equipos de tracción y auxiliares. Contiene: **ChannelID**, **TractionSystem**, **Energy**
ChannelID	**Channel**	Número para la identificación del canal de medida. Dato fijo
TractionSystem	**Channel**	Identificador del sistema de tracción (puede haber varios)
Energy	**Channel**	Contiene: **Active**, **Reactive**, **Quality**
Active	**Energy**	Medida de energía activa. Contiene: **Consumed**, **Regenerated**
Reactive	**Energy**	Medida de energía activa. Contiene: **Consumed**, **Regenerated**
Consumed	**Active, Reactive**	Medida de energía consumida, obtenida por la interpolación entre los vectores de tiempo y de energía activa/reactiva consumida
Regenerated	**Active, Reactive**	Medida de energía regenerada, obtenida por la interpolación entre los vectores de tiempo y de energía activa/reactiva regenerada
Quality	**Energy**	Calidad de la medición (127 sin errores)
Location	**CEBD**	Posición actual del EMS en coordenadas geográficas. Contiene: **Latitude**, **Longitude**, **Quality** Datos obtenidos por la interpolación entre los vectores de tiempo de coordenadas geográficas

Una vez formado en su totalidad el objeto CEBD, se añade a una lista. Esta lista, cuando llega a contener el máximo de CEBD para la formación de un CEBDBlock, se pasa a la función `ConstruirCEBDBlock`, donde se anidan a una estructura tipo CEBDBlock y la lista se vacía, comenzando de nuevo el ciclo.

En la función `ConstruirCEBDBlock`, se siguen las mismas pautas de elaboración del XML. Para crear el CEBDBlock, primero se introduce la información del EMS de forma similar a como se encuentra en el fichero `Input_EMS.xml` A continuación, se introducen los datos del intervalo total que cubren las mediciones (`MeasurementInterval`). Se incluyen en esta parte cada CEBD formado que se encuentra en la lista (comando `Append`) y finalmente se inserta la firma del EMS (estructura fija). El objeto se convierte a texto plano, se escribe en un nuevo archivo y se guarda en el directorio "`./OUTPUT/EMS`", donde el usuario del simulador podrá visualizarlo.

El objeto padre CEBDBlock se transfiere al DHS del EMS, donde será almacenado y donde el software buscará en caso de petición del servicio CEBDBlock. En Python, el **DHS** se representa como un atributo del EMS de tipo lista. Cada elemento de la lista representa una estructura creada. Será un elemento de tipo diccionario, con varias claves, que indicarán:

- El nombre del servicio.
- El objeto creado en formato XML (tipo `lxml.ElementTree`)
- La hora de creación
- Booleano que indica si el objeto ya ha sido enviado una vez (para el método de transferencia automática)
- Ruta absoluta del equipo de almacenamiento local

5.3.5.2 ReadingBlock

La formación y almacenamiento de ReadingBlocks es muy similar a la del CEBDBlock. La diferencia principal es el ciclo de formación de Readings, que es de 1 minuto, frente a 5 de CEBD. Por lo tanto, contiene medidas más precisas del consumo y regeneración de energía en periodos de tiempo más concretos. Además, incluye información sobre la velocidad instantánea en el momento de creación de la estructura Reading. Cada Reading lleva un CEBD asociado, por lo que existe relación entre ambos servicios, lo cual puede resultar útil en determinados casos.

El método para formar Readings es `FormarReading`. Al formarse se añaden a una lista, y cuando ésta alcanza el límite de elementos según el máximo número de Readings posibles para un ReadingBlock (valor que podrá ajustarse) se invoca al método `ConstruirReadingBlock`, de igual manera que para los CEBDBlocks. Se añaden también a la lista `DHS`.

5.3.5.3 EventSet

La norma considera como obligatoria la implementación de un servicio de registro y comunicación de eventos del EMS.

Los que se han tomado en cuenta para la programación de la herramienta han sido:

- Encendido y apagado del EMS
- Login, error de Login y Logout de usuarios
- Detección de mensajes y solicitudes inválidas

Cada uno de estos eventos lleva asociado un ID. En los métodos de conexión cliente – servidor implementados en posteriores fases (Login y Logout), en la entrada al bucle de simulación (encendido del EMS) y en las funciones de interpretación de mensajes entrantes se realizarán llamadas al método `DefinirEvent`, que construirá una estructura de tipo evento con los principales campos de: dominio del evento, subdominio y nombre del evento producido. Estos campos se encuentran dentro del apartado `Description` de un `Event`.

Un ejemplo de EventSet con un eventos de conexión de un usuario puede encontrarse en la página 65 (Parte II - 2.1.3).

5.3.5.4 CommunicationConfig

Este servicio es opcional para la norma, pero se ha incluido en el trabajo. Consiste en el registro de la configuración actual del EMS y de las comunicaciones del mismo con el exterior.

Pueden configurarse los derechos de acceso y permisos de los usuarios con capacidad de conectarse al EMS, así como los tiempos máximos de respuesta del EMS ante peticiones, máximo número de estructuras simples por bloque (por ejemplo, máximo número de CEBD por CEBDBlock: 12, equivale a una hora de intervalo de mediciones), o también el máximo número de bloques por mensaje.

Algunos de estos ajustes podrán ser realizados por el usuario a través de una ventana de configuración desarrollada en etapas posteriores.

Estos ajustes serán almacenados en variables de tipo lista o diccionario del objeto `EMS` y cada vez que se aplique un cambio de configuración se borrará la estructura de `CommunicationConfig` previamente existente en el DHS para crear un nuevo fichero `CommunicationConfig` con los ajustes actuales.

Un ejemplo de CommunicationConfig.xml puede verse en la página 65 (Parte II - 2.1.4).

Estos cuatro son los servicios que han sido implementados en el simulador de comunicaciones.

5.3.6 ELABORACIÓN DEL DCS. USUARIO EN TIERRA

En esta fase se debe construir un programa paralelo para simular la terminal del DCS, el equipo que se conectará el EMS para obtener de él los datos generados. Para conectarse al EMS, es necesario estar registrado en el EMS como usuario, con su correspondiente contraseña.

Tras conectarse al EMS, el DCS podrá enviar peticiones al EMS, invocando un determinado servicio que ha de ser despachado por el servidor de datos embarcado de la conexión. Tras atender el request (petición), el EMS enviará una respuesta de vuelta al DCS con el resultado de la interpretación del mensaje entrante.

Este es el mecanismo Request/Reply. En el modo automático, el usuario del DCS solo tiene que conectarse al EMS para comenzar a recibir los mensajes con la información pertinente de forma automática según se encuentren en el DHS.

Por tanto, en la interfaz gráfica, se incluirán los elementos de usuario, contraseña, una lista de servicios que pueden solicitarse al servidor y una lista de eventos y mensajes, en primera instancia.

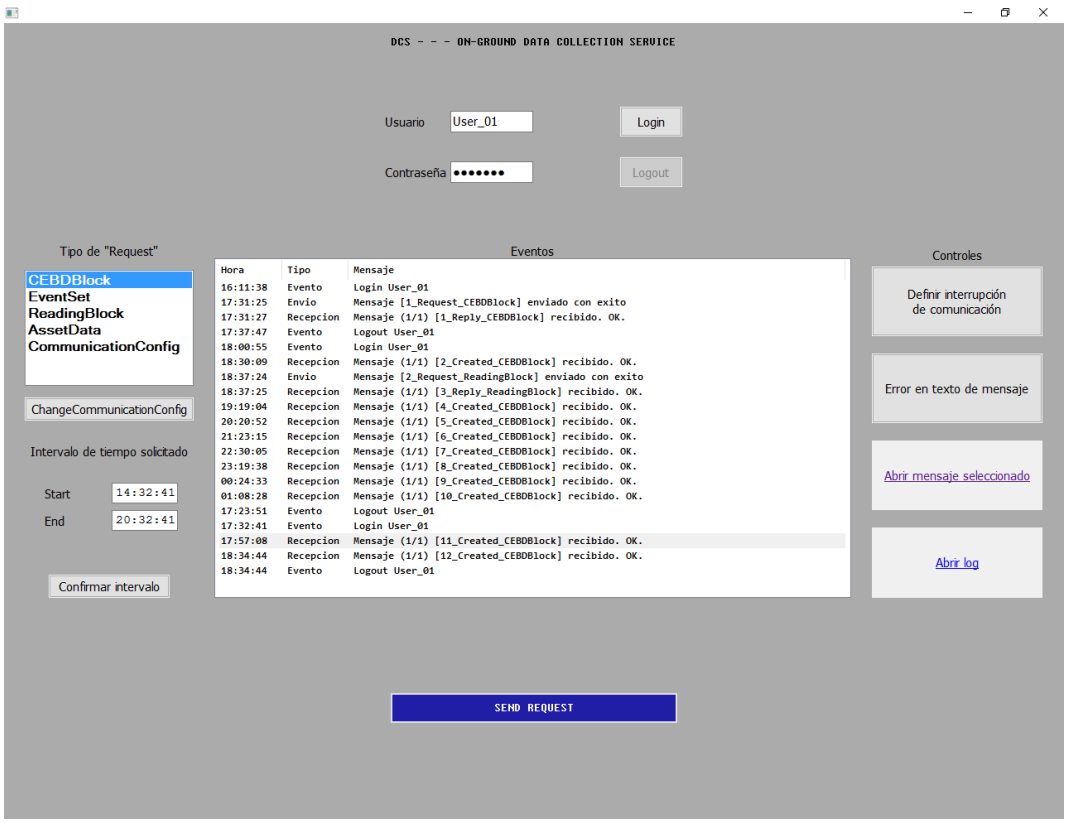


Figura 7: Interfaz de usuario del equipo DCS

Un elemento importante que debe incluirse en la interfaz es el cuadro de ajuste del intervalo de tiempo para el cual se solicitan los datos. El intervalo es necesario para los servicios CEBDBlock, ReadingBlock y EventSet. Al presionar el botón de confirmar intervalo se comprueba que no hay errores lógicos en la introducción del tiempo inicial y final y se habilita en ese caso el botón de “SEND REQUEST”.

También se ha incluido el botón “ChangeCommunicationConfig”, que permitirá al usuario del DCS realizar ajustes en la configuración del EMS de forma remota,

enviando un request al EMS con la nueva configuración deseada. Este servicio se denomina "ChangeCommunicationConfig" en el servidor del EMS.

En el centro, tanto los mensajes enviados como recibidos podrán seleccionarse y visualizarse en el editor de texto predeterminado del usuario del simulador pulsando en el botón "Abrir mensaje seleccionado" de la barra de botones situada a la izquierda.

Tras crear esta interfaz mediante el uso del módulo `wxPython`, comienza el desarrollo de los módulos para la intercomunicación de ambos terminales.

5.3.7 DESARROLLO DE LOS MÓDULOS DE COMUNICACIÓN PARA EMS Y DCS

El protocolo de comunicación elegido para la comunicación entre ambos sistemas ha sido XML-RPC (Remote Procedure Call). Este protocolo emplea XML para codificar los datos en un formato específico y HTTP para la transmisión de mensajes. La sintaxis de llamada XML-RPC:

```
POST /RPC2 HTTP/1.1
User-Agent: [see RFC 2614 for definition]
Host: [DestinationAddress]
Date: [transmission date and time]
Content-Type: text/xml
Content-length: [length of request in bytes]

< ?xml Version = ”1.0”? >
< methodCall >
       < methodName > [MethodName] < /methodName >
       < params >
       < param >
              < value > < string > [OriginAddress] < /string > < /value >
       < /param >
       < param >
              < value > [TypePlusMessage] < /value >
       < /param >
       < /params >
< /methodCall >
```

En Python existen dos módulos dedicados a la comunicación con este procedimiento. Uno para el servidor, denominado `SimpleXMLRPCServer`, y otro genérico con el nombre de `xmlrpclib`.

La clase `SimpleXMLRPCServer` del módulo homónimo proporciona métodos para el registro de funciones que pueden invocarse por el protocolo XML-RPC. Por tanto, en el EMS se deberá crear una instancia de esta clase y registrar las funciones que puedan ser llamadas por el DCS por el citado protocolo y que además puedan enviar respuestas por el mismo procedimiento.

```
self.server = SimpleXMLRPCServer(('localhost', 9300),
logRequests=True, allow_none=True)
```

Así se ha creado el atributo server del objeto EMS, con la dirección local (en caso de que los programas cliente y servidor estén en equipos con diferente IP, habría que introducir la dirección IP del servidor) y el puerto. Para registrar funciones, se realiza la llamada al método `register_function` del objeto `SimpleXMLRPCServer`.

Las funciones registradas e incluidas en la clase denominada `EMSServidor`, dentro del script EMS.py, pueden ser llamadas desde el programa cliente una vez se construya.

En el script DCS.py de la parte del cliente, se ha construido la clase `ClienteDCS`, donde se va a crear la conexión con el servidor. Para ello, se crea una instancia `ServerProxy` del módulo `xmlrpclib`. Esta instancia será un objeto que gestionará la comunicación remota con el servidor EMS. Se pasa la URL del servidor y el puerto (“localhost”, 9300) como argumento, y se define el transporte de los mensajes, según lo establecido por el estándar. Lo que éste establece implica modificar las cabeceras HTTP, fijando el tipo de contenido (Content-Type: text/xml) y la longitud en bits del mismo (Content-Lenght: [longitud del mensaje]) del tipo de transporte por defecto del objeto `ServerProxy`.

En el método de inicialización de la clase `ClienteDCS`, tras crearse la conexión, se llama al método `UserLogin` para conectar al usuario con el EMS.

```
#=========================================================================
# Clase para definir el tipo de transporte del mensaje http
#     (cabeceras especificadas válidas para la norma)
#=========================================================================
class SpecialTransport(xmlrpclib.Transport):
    def __init__(self,useragent):
        self.user_agent = useragent
        self._connection = (None, None)
        self._use_datetime = True
```

```
    def send_content(self, connection, request_body):
        connection.putheader("Content-Type", "text/xml")
        connection.putheader("Content-Length", str(len(request_body)))
        connection.endheaders()
        if request_body:
            connection.send(request_body)


#============================================================================
# Clase que controla la conexión con el servidor del EMS
#============================================================================
class ClienteDCS:
    def __init__(self, useragent, password):
        self.servidor = xmlrpclib.ServerProxy('http://localhost:9300',
transport=SpecialTransport(useragent), verbose=True)
        self.loginOK, self.horaLogin = self.servidor.UserLogin(useragent,
password)
        # print 'Ping:', self.servidor.ping()
        self.testigo = "Free" #Quien posee el testigo puede llevar a cabo
la peticion al servidor
```

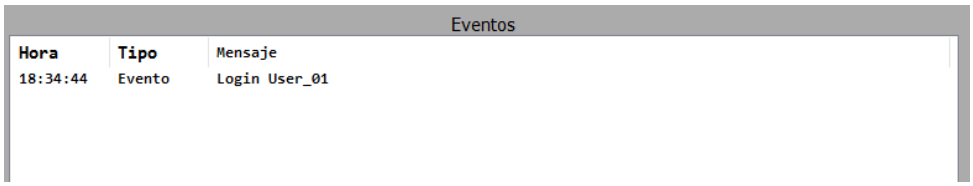
En el servidor, el método `UserLogin` conectado comprueba que el usuario que trata de acceder está registrado en su base de datos, crea un evento con `DefinirEvent` para registrar el intento de `Login` y devuelve la hora si la conexión es correcta. Este es el mensaje enviado en el intento de conexión, incluyendo las cabeceras HTTP:

```
send: 'POST /RPC2 HTTP/1.1\r\n
Host: localhost:9300\r\n
Accept-Encoding: gzip\r\n
User-Agent: User_01\r\n
Content-Type: text/xml\r\n
Content-Length: 217\r\n\r\n'
send: "<?xml
version='1.0'?>\n<methodCall>\n<methodName>UserLogin</methodName>\n<para
ms>\n<param>\n<value><string>User_01</string></value>\n</param>\n<param>
\n<value><string>EN50463</string></value>\n</param>\n</params>\n</method
Call>\n"
```

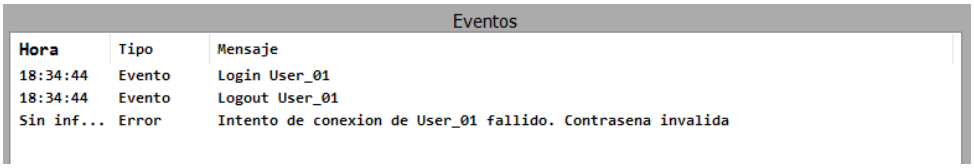
Se resalta el nombre del método invocado, y los argumentos de la función usuario y contraseña. Si hay error, el servidor devuelve una excepción al cliente, de forma que no se genera error en el EMS, pero sí en el DCS.

Los mensajes enviados en la comunicación y los errores producidos pueden visualizarse en un archivo de registro log, presionando el botón "Abrir log" situado en la parte inferior del panel de botones derecho (Ver Figura 7). Este log recopila el texto de la salida estándar de Python `stdout` (utilizando el módulo `sys`). Se crea en el script `CreateLog.py` del directorio "`./UTILIDADES`".

Si la conexión del usuario es correcta, aparecerá en la lista de eventos del DCS:



Si la contraseña es incorrecta, el método devolverá un texto de "Invalid Password" y el Login no tendrá efecto:



5.3.8 LÓGICA DEL PROGRAMA DCS USUARIO

El funcionamiento a alto nivel del programa usuario, en el modo de comunicación request / reply, queda como sigue:

- Al iniciarse el programa el usuario debe conectarse al EMS pulsando en Login con el nombre de usuario y contraseña. En este caso, el único usuario registrado que puede "logearse" es el "User_01" con la contraseña "EN50463". Si entramos con otro usuario o clave el EMS debe devolver una excepción (error). Al conectarse correctamente, se crea una carpeta con el nombre del usuario (y borra su contenido si el directorio existiese) en el directorio "`./OUTPUT`".
- Una vez conectado, el usuario puede seleccionar uno de los servicios ofrecidos en el panel izquierdo. Si es uno de entre los tres siguientes: EventSet, CEBDBlock, ReadingBlock; el botón enviar request quedará deshabilitado

hasta que se confirme el intervalo para el que se solicita el servicio. En caso de que sea otro servicio el botón request estará habilitado, pues el intervalo no será preciso. La forma de controlar el botón request en función de la opción seleccionada se realiza mediante la captura de eventos en el *frame* (ventana y contenido) principal del programa DCS.py, por el método `Bind`. Si se produce el evento, por ejemplo, de selección de una opción, se invoca a una función que habilita o deshabilita el botón "Send Request" según la opción seleccionada.

- Una vez enviado el mensaje, se obtiene la respuesta y se interpreta (Ver apartados 5.3.9 y 5.3.10), mostrándose en la lista de eventos. El resultado de la petición (OK, PARTIAL, FAILED) se muestra también en este cuadro. Los mensajes, tanto enviados como recibidos, se guardarán en la carpeta de usuario y automáticamente se generará el enlace en el programa para poder abrirlos (control `wx.Hyperlink`) con el botón "Abrir mensaje seleccionado" de la barra derecha de botones.
- Si se requiere detener la comunicación se presiona el botón "Logout", para borrar al usuario del registro de usuarios conectados del DCS (variable tipo lista en Python).

5.3.9 FORMACIÓN, ENVÍO Y VISUALIZACIÓN DE REQUESTS. DCS.

Al presionar el botón enviar Request, se llama al método `GeneralRequest`, que forma el mensaje partiendo de la estructura genérica como la que aquí se presenta:

```
<?xml version='1.0' encoding='UTF-8'?>
<RequestMessage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.cenelec.eu//EN50463#"
xsi:schemaLocation="http://www.cenelec.eu//EN50463# CEBDBlock.xsd">
  <Header>
    <Verb>get</Verb>
    <Noun>[NOMBRE DEL SERVICIO]</Noun>
    <Revision>1.0</Revision>
    <Timestamp>[HORA ACTUAL EN FORMATO ZULU]</Timestamp>
    <Source>[NOMBRE_USUARIO].adif.es</Source>
    <User>
      <UserID>[NOMBRE_USUARIO]</UserID>
    </User>
    <MessageID>1</MessageID>
    <Target>1234567890123.adif.es</Target>
  </Header>
```

```
  <Request>
    <StartTime>[HORA INICIO INTERVALO]</StartTime>
    <EndTime>[HORA FIN INTERVALO]</EndTime>
  </Request>
   <Payload>
   (no siempre es necesario en la mayor parte de los casos)
  </Payload >
</RequestMessage>
```

Se construye por los mismos métodos vistos anteriormente, y habrá que rellenar los campos precisos con la información:

- **Nombre del servicio**, conocida por la opción seleccionada por el usuario
- **Hora actual** del sistema en formato "Zulu"[5].
- **Nombre del usuario**
- Opcionalmente si el servicio lo requiere, las **horas de inicio y fin** del intervalo pedido.

Un ejemplo puede encontrarse en Parte II - 2.2.1.

Para enviar el mensaje, se convierte el objeto de `ElementTree` (`lxml`) en un `string` por el método `tostring` y se pasa el request como argumento a una función que ha de estar registrada en el servidor: `GroundRequest`. Estructura del mensaje definida por el protocolo XML-RPC:

```
send: 'POST /RPC2 HTTP/1.1
Host: localhost:9300
Accept-Encoding: gzip
User-Agent: User_01
Content-Type: text/xml
Content-Length: 994

send: '<?xml version=\'1.0\'?>
<methodCall>
      <methodName>GroundRequest</methodName>
      <params>
            <param>\n<value><string>&lt;?xml version=\'1.0\'
            encoding=\'UTF-8\'?&gt;\n&lt;RequestMessage . . . . .
            </string></value>\n</param>
      </params>
```

[5] Formato Zulu de fecha y hora: aaaa-mm-ddThh:nn:ssZ a: año, m: mes, d: día, h:hora, n: minutos, s: segundos

```
</methodCall>\n'
```

El request se pasa como parámetro en formato de texto plano de la forma mostrada. En casos como el del ferrocarril, en el que la memoria embarcada representa un coste elevado por sus requisitos de robustez y resistencia ante golpes, vibraciones y aceleraciones, es crítica la capacidad de comprimir la información. En el caso de la petición de cambio de configuración se tratará este tema.

5.3.10 INTERPRETACIÓN Y PROCESAMIENTO DE LOS REQUESTS. EMS.

En el EMS, se registra la función `GroundRequest` que devolverá tres parámetros, la hora de recepción del request, la respuesta (el `reply`) y la hora de envío del request.

El request se reenvía a una función propia del EMS denominada `AtenderRequest`. En ella, se realizará una validación del request conforme al esquema XSD que define la estructura que ha de tener. (Ver ap. Parte I - 5.3.11).

La forma de interpretar el mensaje es la siguiente:

- El Request puede manejarse directamente al recibir el mensaje como un `string` (texto_plano). Para convertirlo en un elemento de tipo `ElementTree` se pasa el request como argumento del método `FromString`, que devuelve el objeto buscado. Sobre ese objeto, podemos recorrer con bucles `for` cada uno de los elementos anidados que posea. El objeto se comporta como una variable de tipo lista, cuyos elementos son los subelementos del nivel siguiente. Así se localiza el ID del mensaje, y el servicio solicitado. En caso de que el servicio solicitado implique una definición de intervalo temporal, se obtienen las horas del intervalo y se pasan a variables locales del EMS. Finalmente, según el tipo de servicio identificado, se invoca a una u otra función y se pasan los argumentos de formato temporal si son necesarios:

    - `ObtenerCEBDBlocks`

- `ObtenerEventSet`
- `ObtenerReadingBlocks`
- `ObtenerCommunicationConfig`
- `CambiarConfiguracionComunicacion`, para acceder a esta función es necesario leer el "payload" (carga útil) del request (en otros servicios no se incluye). Ejemplo de ChangeCommunicationConfig request en Parte II - 2.2.2.

5.3.11 ELABORACIÓN Y ENVÍO DEL REPLY

Una vez efectuada la llamada a la función correspondiente según el tipo de servicio solicitado, se realiza el siguiente procedimiento:

- Si la función invocada es `ObtenerCEBDBlock`, `ObtenerEventSet` u `ObtenerReadingBlock` se le deben haber pasado como argumentos los tiempos del intervalo. En cada una de estas funciones se recorre con un bucle cada una de las estructuras almacenadas en el DHS. Específicamente, se compara el nombre del servicio solicitado con el nombre del servicio de cada elemento, guardado en la clave pertinente del diccionario creado para su almacenamiento en la lista DHS. (En Parte I - 5.3.5.1 se describe la información guardada en el DHS)
- Si el nombre coincide, se compara el intervalo solicitado con el "`TimeStamp`", la hora en la que se almacenó el bloque. Si el bloque con las mediciones contiene información que se encuentra en el intervalo temporal requerido, se añade a una lista provisional. Se procede así hasta terminar de recorrer la lista de elementos almacenados del DHS. La lista completa, se pasa como argumento a una función genérica para elaborar la respuesta, denominada `GeneralReply`. También se pasan como argumentos el nombre del servicio y la etiqueta "verb" del XML necesaria para construir tanto el request como el reply. En el caso de la respuesta este verbo será "`reply`".

- Si la función a la que se llama es `ObtenerCommunicationConfig`, pensada para el servicio CommunicationConfig, se procederá de igual manera evitando las comparaciones temporales, que no tendrán sentido en este caso, ya que la configuración del EMS que se desea conocer desde el DCS cliente será, por norma general, la más actualizada posible. (En el DHS, cada vez que se cree un bloque de configuración CommunicationConfig, se eliminarán todos los bloques antiguos del DHS, puesto que no serán necesarios.
- Cada función pasa las estructuras seleccionadas a la función `GeneralReply`, que construye el reply siguiendo la estructura especificada por la norma:

```
<?xml version='1.0' encoding='UTF-8'?>
<ReplyMessage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.cenelec.eu//EN50463#"
xsi:schemaLocation="http://www.cenelec.eu//EN50463# ****Block.xsd">
  <Header>
    <Verb>reply</Verb>
    <Noun>[NOMBRE DEL SERVICIO] </Noun>
    <Revision>1.0</Revision>
    <TimeStamp>[HORA ACTUAL EN FORMATO ZULU]</TimeStamp>
     <Source>[NOMBRE_USUARIO].adif.es</Source>
    <MessageID>1</MessageID>
    <CorrelationID>1</CorrelationID>
    <Target>dcs.adif.es</Target>
  </Header>
  <Reply>
    <Result>[OK / FAILED / PARTIAL]</Result>
      <Error>
        <Code>Código de error si result = FAILED</Code>
      </Error>
  </Reply>
  <Payload>
    (Datos XML)
  </Payload>
</ReplyMessage>
```

Se construye por los mismos métodos propios de la librería `lxml`. Los campos necesarios:

- **Nombre del servicio**, conocido por el argumento pasado a la función.
- **Hora actual** del sistema en formato Zulu.
- **Nombre del usuario**, guardado en una variable del objeto `EMS` al producirse la conexión.
- ID del mensaje a enviar (contador de replies enviados) e ID del mensaje correlacionado (contador de requests recibidos).

- **Objetivo**, dirección URL del puesto de conexión del usuario (debería rellenarse con la dirección local "localhost", pero en la comunicación se ha establecido un nombre fijo).
- **Resultado**, OK si se ha recibido correctamente el mensaje y la respuesta ha sido convenientemente formada; FAILED en caso de error de recepción o de formación de la respuesta; PARTIAL en caso de que la información solicitada se envía dividida en varios mensajes.
- **Payload**, datos solicitados en formato XML según las estructuras sugeridas por el estándar. Se añaden mediante el método `ET.Append` los elementos de la lista de bloques pasados como argumentos a la función.

Para que el resultado del envío sea OK o PARTIAL, no tiene que ocurrir que sea FAILED. La lógica de detección de errores se presenta en el punto siguiente (Parte I5.3.12Parte I - 5.3.12).

Cada tipo de servicio tiene un valor máximo de bloques que pueden enviarse a la vez en un mensaje debido al volumen de datos. Este valor puede ser distinto para cada servicio y de momento se establece fijo en el código (por ejemplo, máximo número de CEBDBlocks por mensaje: 4), aunque el usuario del simulador podrá modificarlo a su gusto tras implementar la gestión de configuración, que será descrita posteriormente.

Según la norma, si el número de bloques que se necesita enviar supera este valor, se construye un mensaje con el número máximo, y se envía. Se repite el procedimiento hasta enviar todos los bloques en varios mensajes, y se establece en la etiqueta `Result` el texto: PARTIAL. En Python, se ejecuta un bucle que forma mensajes y va borrando los bloques de la lista de bandeja de salida conforme se añaden al mensaje.

En caso de que el resultado no sea FAILED ni PARTIAL, se muestra el texto: OK. Se añadirán los bloques necesarios y se devolverá a la función registrada en el servidor, que se encuentra en niveles superiores, el texto plano formado.

Esta función enviará el mensaje al DCS mediante HTTP con el formato definido por el protocolo XML-RPC.

El mensaje será interpretado por los mismos mecanismos vistos en el DCS y se mostrará información sobre la recepción en el cuadro de eventos. El mensaje es almacenado en un archivo XML en el directorio "`./OUTPUT/EMS`".

5.3.12 DETECCIÓN E INFORMACIÓN DE LOS POSIBLES ERRORES

Para que se elabore la respuesta en la función `GeneralReply`, han de aprobarse ciertas pruebas de validación para poder elaborar la respuesta. En orden de evaluación:

1. Cuando el request pasa por la función `AtenderRequest` donde es leída e interpretada por el EMS, el mensaje recibido se somete a validación contra el esquema XSD correspondiente. Un esquema de estas características especifica la estructura que debe regir el texto recibido, y el formato de los datos, así como la obligatoriedad de los campos del XML. El esquema para un mensaje genérico puede encontrarse en Parte II - 2.3.1.

   Los esquemas en formato XSD y diversos ejemplos de estructuras XML y XSD pueden encontrarse en el directorio "`./DATOS/XMLSchemas`".

   Para validar el mensaje request contra su esquema, se pasa a la función `ValidarXMLSchema`, definido en el script de `VALIDACION_XML.py`.

   El método específico del módulo de Python para la validación es el mismo que para formar objetos XML a partir de `strings`: `ET.tostring()`, pero en este caso, a parte del texto que queremos validar se pasa un segundo argumento `parser` con el objeto esquema que ha debido ser obtenido del directorio mencionado.

   ```
   ET.fromstring(fichero_xml, parser=xsd_schema)
   ```

   En caso de que el XML no apruebe la validación se producirá una excepción, por lo que habrá que gestionarla con una estructura `try` –

`except` de Python y obtener así el código del error de validación, que se añadirá al mensaje de respuesta.

Si hay error, se pasa a la función GeneralReply un argumento especial con el resultado de la validación, y se escribirá FAILED en el mensaje de respuesta, junto con el código "XML Syntax Error" y la descripción específica del motivo de la excepción. Ejemplo de Reply en Parte II - 2.2.4.

2. Si el mensaje no contiene errores en la validación contra esquema y es interpretado satisfactoriamente, se evalúa si el usuario que ha realizado la petición está en el registro de los usuarios conectados del EMS (para ello el usuario ha debido conectarse con su usuario y contraseña correctos). En caso contrario, el resultado del reply es FAILED y el código: "Authentication Failure".
3. Si no hay problemas de autenticación, se pasa a la siguiente evaluación. El EMS puede conceder permisos al usuario sobre ciertos servicios, mientras que puede denegarle otros. Por ejemplo, el usuario puede solicitar ReadingBlock pero no EventSet (se podrá configurar en etapas posteriores). En caso de que el usuario solicite servicios que no se le permiten, el resultado del reply será FAILED y el código: "Action not authorized for user".
4. En caso de que el request pase todos los exámenes previos, se determina si existen datos que puedan reenviarse al DCS. Si no están disponibles en el DHS, el resultado del reply será FAILED y el código "Requested data is not available".

En caso de que el resultado no haya sido establecido a FAILED por las comprobaciones descritas, se pueden adjuntar datos al mensaje o los mensajes de vuelta al usuario del DCS, es decir, existirá el campo `Payload`.

5.3.13 MÉTODO DE TRANSFERENCIA AUTOMÁTICA DE MENSAJES

Ya se ha implementado el modo simple de comunicación basado en petición y respuesta. El otro modo que considera la norma es lo que se denomina suscripción a eventos. Puede entenderse como el envío automático de mensajes desde el servidor a los clientes que se hayan registrado como suscriptores para participar en este tipo de comunicación "unidireccional", en el que el EMS, ante la detección de que un evento se produce, llama automáticamente a una función que elabora y envía un mensaje sin necesidad de recibir una solicitud previa.

El modo de comunicación requeriría una conexión permanente en la que el servidor envía a través de un mismo canal (HTTP persistent connection). Sin embargo, este método no está implementado en las librerías principales utilizadas para establecer el diálogo (`xmlrpclib` principalmente).

Podrían definirse unos módulos alternativos en el EMS y DCS de modo que el EMS opere como cliente enviando lo que ya no serían requests, sino replies directos (ahora con el verbo "created" en vez de reply) al que ahora sería el servidor DCS, que no devolvería ningún dato. En vez de crear estos módulos paralelos a los ya creados, se ha ideado un método a priori más simple.

El método concebido para la transferencia automática de eventos, se basa en un automatismo de conexión cíclica del DCS (como cliente) al EMS (servidor) para conocer el modo de comunicación en el que el EMS se encuentra operativo. Si el EMS responde al DCS que el modo de comunicación será el de transferencia automática de mensajes, el DCS iniciará un bucle de conexión a intervalos de tiempo más cortos para saber si el EMS tiene en bandeja un mensaje disponible para su envío, y recibir el mensaje en forma de reply.

Este método de comprobaciones ha de ser realizado por un hilo de ejecución paralelo al hilo general de ejecución del programa `DCS.py`, lo cual se ha programado mediante el comando `start_new_thread` del módulo `thread`.

En el momento en que el usuario se conecta al EMS, se lanza este hilo de ejecución (función `ComprobarMetodoMensajeria`) que ejecuta el bucle permanente de conexiones periódicas. Cada 10 segundos, se envía el mensaje siguiente:

```
<?xml version='1.0'?>
<methodCall>
	<methodName>CheckMessageMethod</methodName>
	<params></params>
</methodCall>
```

El método `CheckMessageMethod` (comprobar modo de mensajería) debe registrarse en el servidor EMS. Éste devuelve el modo de comunicación empleado (el modo se guarda en una variable propia del EMS):

```
<?xml version='1.0'?>
<methodResponse>
	<params>
		<param>
			<value>
				<string>Request/Reply</string>
			</value>
		</param>
	</params>
</methodResponse>
```

Si el valor recibido es “AutomaticTransfer”, el DCS ejecuta la función `TransferenciaAutomatica`, que realiza un bucle de conexiones espaciadas en el tiempo 1 segundo, llamando al método `AutomaticTransfer` del EMS. Ahora entra en juego el campo “enviado” de cada objeto del DHS, que expresa si el bloque ha sido enviado al menos una vez por este método. La función del EMS evalúa el bloque más antiguo que esté en cola para envío al DCS (no hay prioridades) y comprueba que el usuario tiene derecho a suscribirse a eventos de ese servicio, para después llamar al método GeneralReply, elaborar el mensaje (ahora con el verbo “created”) y enviarlo al DCS.

Cuando transcurren una sucesión de 10 conexiones se sale del bucle de transferencia automática del DCS y se comprueba de nuevo el método de conexión, repitiéndose el ciclo si el modo sigue siendo “AutomaticTransfer”.

Puesto que este es un hilo de ejecución paralelo, y el usuario puede estar emitiendo request manualmente, se ha definido una variable propia del DCS y accesible desde ambos hilos que actúa de testigo y determina qué método tiene el permiso para realizar la conexión, evitando los conflictos que pudiesen ocurrir.

Por ejemplo, si el método de comprobación de modo de comunicación acaba de enviar un mensaje y está esperando el reply del servidor, y simultáneamente el usuario del simulador envía un request al EMS; el servidor entraría en conflicto y lanzaría una excepción por la consola estándar de salida de Python. Con el testigo, si el citado método pone la variable testigo a su nombre, el método de `GeneralRequest` se pondrá en estado de espera hasta que el valor de la variable testigo sea "`Free`" (libre) y este método pueda ponerla a su nombre. Al finalizar la conexión, es esencial asignar el texto "`Free`" a la variable `testigo`.

Cabe citar que se permite el envío de request mientras el modo sea "`AutomaticTransfer`", pero el servidor EMS no puede emitir mensajes automáticos si el modo es "`Request/Reply`".

5.3.14 GESTIÓN DE LA CONFIGURACIÓN

Tanto en el EMS como en el DCS se han implementado dos ventanas destinadas al ajuste de la configuración de la comunicación. En el EMS es accesible a través del botón en el que se muestra la imagen de una rueda dentada. Por la parte del EMS, será configurable tanto características de la simulación como del módulo EMS.

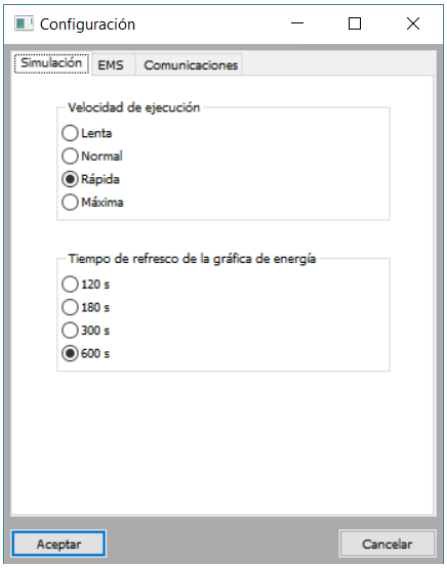


Figura 8: Ventana de configuración del EMS: parámetros de la simulación

En la ventana de la Figura 8 se podrá ajustar la velocidad de ejecución de la simulación, que modificará una variable que introducirá un tiempo de pausa en cada iteración del bucle principal de la simulación. Si la velocidad ejecución seleccionada es máxima, esta variable será 0. En el caso opuesto, al seleccionar "Lenta", se introducirá una pausa adicional de 50 milisegundos en cada iteración.

En la parte inferior, se podrá determinar el tiempo de refresco de la gráfica. A menor tiempo, mayor demora en la ejecución (el refresco de la gráfica se ejecuta a través de otro hilo paralelo de ejecución, para no detener la simulación varios segundos en la misma iteración, ya que el volumen de datos que ha de representar es muy elevado).

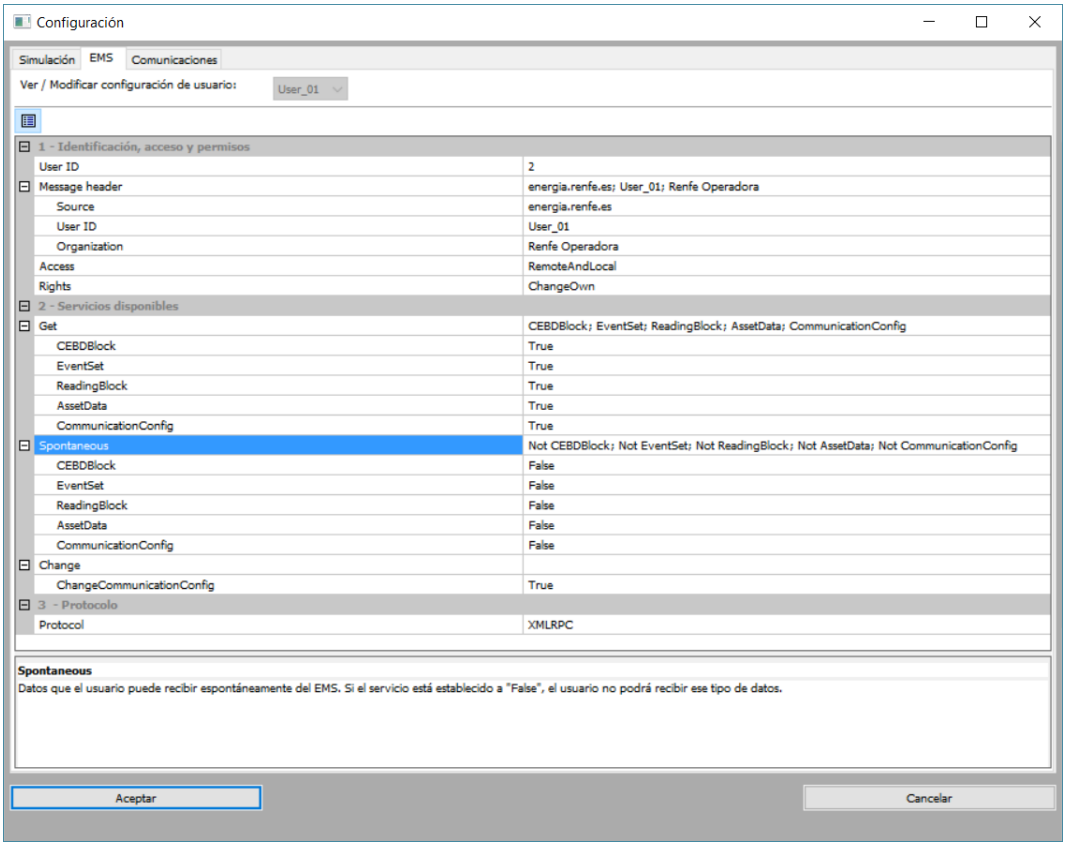


Figura 9: Ventana de configuración: ajustes del usuario del EMS

En el panel de configuración de la Figura 9 podrán realizarse ciertos ajustes para el usuario predeterminado “User_01”:

- Los permisos para la modificación de la configuración (Rights). Pueden seleccionarse tres opciones:
    - **ChangeAll**, el usuario puede modificar cualquier parte de la configuración, tanto la relativa a los usuarios como a la generación de datos por parte del EMS.
    - **ChangeOwn**, solo puede modificar aquellos parámetros que atañen al mismo usuario.
    - **NotChange**, el usuario no tiene permisos para realizar modificaciones.
- Los servicios disponibles en los diferentes modos de comunicación (Get: Request/Reply, Spontaneous: AutomaticTransfer, Change:

ChangeCommunicationConfig). Si un servicio está establecido a True estará disponible para el usuario.

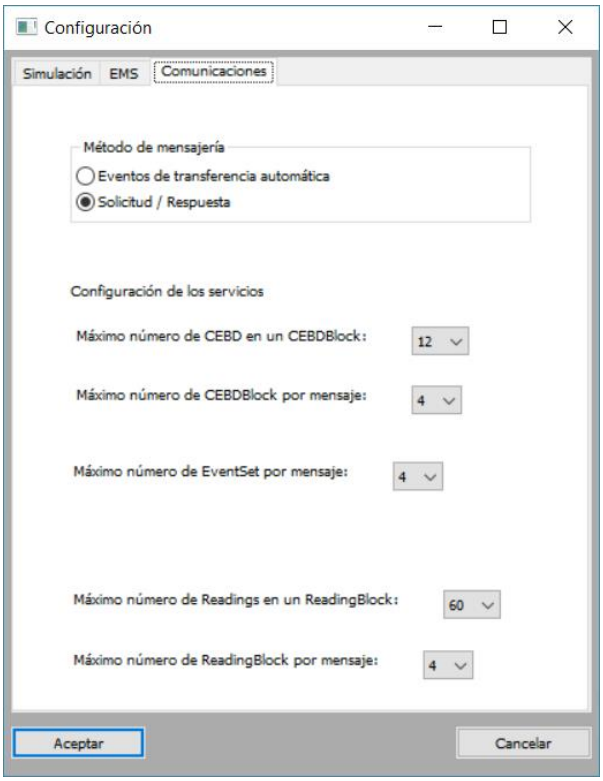


Figura 10: Ventana de configuración: ajustes de los servicios y de generación de datos

En la ventana de configuración de la Figura 10 el usuario del simulador podrá modificar los parámetros de comunicación y generación de datos:

- Selección del modo de comunicación: transferencia automática o solicitud / respuesta
- Selección de valores de generación máximos, lecturas / eventos por bloque y bloques por mensaje.

Al presionar "Aceptar", los valores de las variables locales de la ventana se asignan a los atributos del EMS y del frame, y se llama al método DefinirConfiguracion, que crea un bloque CommunicationConfig y lo almacena en el DHS del EMS. En el modo de transferencia automática, el bloque se pondría en cola para ser enviado al DCS. Ejemplo de CommunicationConfig en Parte II- 2.1.4.

En el programa del DCS, se puede acceder a la ventana de configuración a través del botón "ChangeCommunicationConfig", donde pueden modificarse los mismos

parámetros en remoto, exceptuando los relativos a la simulación, el modo de comunicación (parámetro ficticio no existente en la norma) y los derechos de usuario, que solo pueden modificarse en el ámbito local del EMS. Al presionar el botón "SendRequest" de la ventana de configuración del DCS, se llama al método `SolicitarCambioConfiguracion` que construye el bloque ChangeCommunicationConfig únicamente con las modificaciones necesarias (usuario, generación de datos, …). Ejemplo en Parte II - 2.1.5 con modificación en la generación de datos.

El bloque se envía dentro del Payload del request. En este caso puntual se ha implementado un ejemplo de compresión de texto que bien podría extrapolarse a todos los casos de generación de bloques XML. Se ha realizado con los módulos `zlib` y `base64`. El primero obtiene un archivo de texto comprimido (`zlib` es capaz de utilizar diversos métodos de compresión, pudiendo determinar el equilibrio velocidad de compresión – relación de compresión deseado. En este caso se ha utilizado el método por defecto, que consigue altas velocidades de compresión) y `base64` lo codifica para ser anexado como texto en el request. Ejemplo en Parte II - 2.2.2. La descompresión se realiza por los métodos inversos en la interpretación por parte del EMS.

Si los permisos de usuario indican "ChangeAll", el usuario podrá modificar tanto sus propios parámetros como la generación de datos. Si es "ChangeOwn", solamente sus datos, y si es "NotChange", ninguno de ellos. En caso de realizar una petición no autorizada, el EMS devolverá FAILED con el código "Action not authorized for user".

Se ha añadido, para estos casos, un botón de "Actualizar configuración" en la ventana de configuración del DCS para refrescar dicha información tras un intento fallido de cambio de configuración.

5.3.15 PROGRAMACIÓN DE INTERRUPCIONES DE COMUNICACIÓN

Otro de los objetivos de este trabajo es implementar la funcionalidad de los equipos EMS y DCS ante situaciones degradadas, por ejemplo, ante interrupciones de comunicación. A través del botón del panel derecho del DCS se puede acceder a una sencilla ventana donde definir un intervalo temporal en que la comunicación será interrumpida.

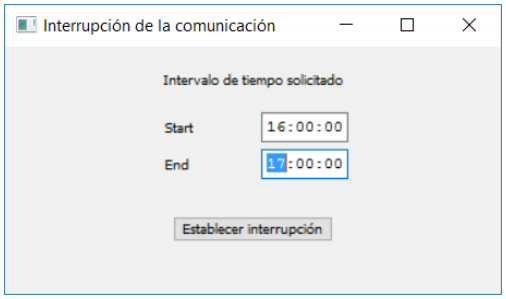


Figura 11: Ventana de interrupción de la comunicación

Al presionar el botón "Establecer interrupción" se invocará a una función previamente registrada en el EMS denominada `CommunicationInterrup`, y a la que se pasarán como argumentos los tiempos de partida y final del intervalo.

Esta función activa análogamente dos variables (inicio y fin del intervalo de interrupción) en el EMS que se evaluarán durante cada iteración de la simulación. Si la hora actual se encuentra comprendida entre este intervalo, se activará la variable del EMS `comunicacionInterrumpida`, que a su vez se evaluará en el método de comprobación del modo de transferencia de mensajes. Si la variable está activada, no se devolverá la información del modo por el método `CheckMessageMethod`, sino que se enviará una lista cuyo primer elemento será la hora del EMS, y el segundo un texto con la información "ComunicacionInterrumpida". Este texto establecerá el texto "COM_ERROR" en la variable de estado de la conexión (ON/COM_ERROR/OFF), y no permitirá acceder al método de transferencia automática en el que se comprueba la existencia de nuevos mensajes, sino que entrará en bucle intentando reconectar (comprobando por la misma función registrada `CheckMessageMethod` si la variable

`comunicacionInterrumpida` se desactiva). Si el método devuelve "AutomaticTransfer" o "Request/Reply" significará que la variable está establecida a False, simulando así la reconexión de los dos equipos. La variable `comunicacionInterrumpida` será desactivada al salir del intervalo. El intento de reconexión se realiza cada 3 segundos reales (pueden ser minutos en la simulación).

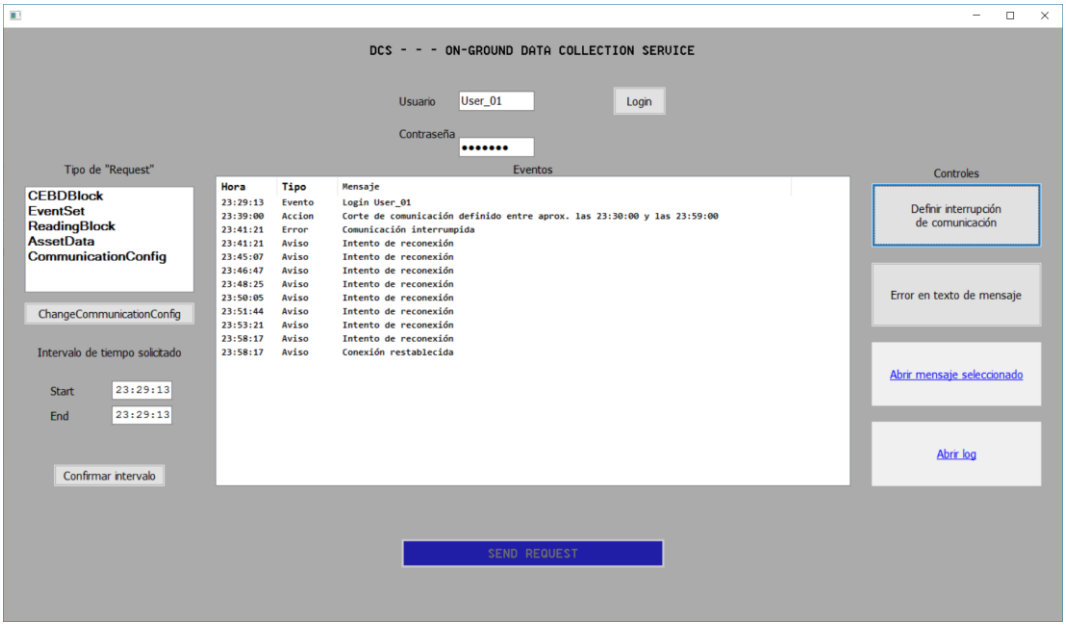


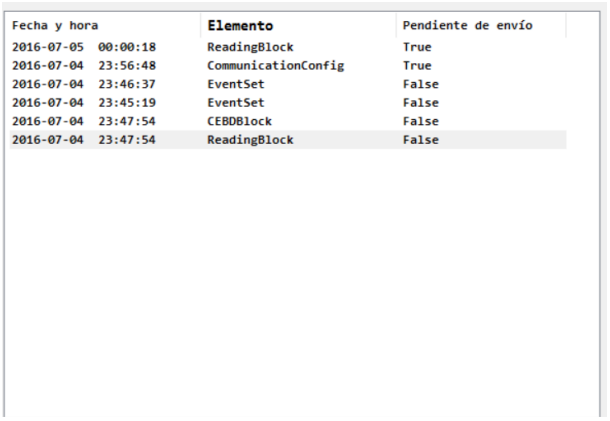
Figura 12: Ventana del DCS: diálogo en caso de interrupción de comunicación y reconexión

5.3.16 LÓGICA Y FUNCIONAMIENTO DE LA COMUNICACIÓN EN MODO DEGRADADO

Cuando la comunicación se interrumpe, en el modo de transferencia automática de eventos, los bloques creados en ese intervalo de interrupción se quedan en cola para su envío (campo "enviado" del objeto del DHS a False). Al reconectar, se envían uno a uno los mensajes en cola, comenzando por el que se añadió antes a la base de datos del Data Handling System.

Esto se produce debido a que en el modo de transferencia automática de eventos solo puede enviarse un bloque cada vez. Así, como la conexión del DCS en este

modo sin problemas de comunicación es de 1 segundo (real), se enviará un mensaje cada segundo hasta enviar todos los mensajes en cola.



Fecha y hora	Elemento	Pendiente de envío
2016-07-05 00:00:18	ReadingBlock	True
2016-07-04 23:56:48	CommunicationConfig	True
2016-07-04 23:46:37	EventSet	False
2016-07-04 23:45:19	EventSet	False
2016-07-04 23:47:54	CEBDBlock	False
2016-07-04 23:47:54	ReadingBlock	False

Figura 13: Vista de mensajes en cola de envío en el DHS con comunicación interrumpida

5.3.17 SIMULACIÓN DE LA CORRUPCIÓN DE MENSAJES DURANTE LA TRANSFERENCIA

Para comprobar el rechazo del EMS ante la recepción de un mensaje inválido por parte del DCS se ha introducido una alteración aleatoria en el mensaje enviado desde la terminal del cliente, simulando las causas del ruido o las interferencias en la comunicación.

Para ello, el usuario tendrá que dejar activado el control de "`ToggleButton`" que proporciona wx. De esta manera, al enviar cualquier request, en el momento previo a la petición del servicio al EMS, se llama a una función que alterará el contenido del request.

Esta función, denominada `MensajeCorrupto`, y guardada en el script `CorrupcionString.py` del directorio "`./UTILIDADES`" realiza las siguientes acciones:

- Toma el `string` pasado como parámetro
- Convierte el `string` a código binario con codificación hexadecimal por el método `hexlify` del módulo `binascii`. La corrupción se introducirá

aleatoriamente en una franja de los bits que forman texto como ocurriría en la realidad.

- Se elige una pequeña fracción del código, la necesaria para que la alteración sea visible por el usuario en la reconstrucción del mensaje en destino.

  A la hora de introducir la alteración de los bits, se formarán códigos que no representarán caracteres permitidos por el "encoding" utilizado de UTF-8 (debido a una característica del mismo denominada no superposición, en el que cada carácter transmitido necesita de unos determinados bits fijos al principio de cada byte en función del rango en el que se clasifique el carácter). Por ello, a mayor fracción de texto alterado, mayor probabilidad de no coincidir con el texto alterado con los bytes y cabeceras aceptadas por la codificación Unicode. La fracción de texto alterado es aproximadamente el 1%, suficiente para que se produzca un error en la recodificación del código binario a texto.

  Para solucionar este inconveniente, se realiza un bucle con un `try` y `except` que se repite al encontrar un error en la codificación del texto corrupto, hasta dar con una combinación admitida.

  La alteración del binario se introduce de forma aleatoria con el método `randint`, que devuelve un entero (específicamente entre 0 y 1) y lo sustituye por el valor original.
- Finalmente se devuelve al programa DCS y se envía al EMS. Se guarda una copia en el directorio del usuario tanto del mensaje sin alteración como el resultante del método de corrupción.
- El EMS obtiene la excepción al tratar de validar el mensaje de solicitud contra el esquema XSD correspondiente y emite la respuesta con información sobre el error.

Ejemplo en Parte II - 2.2.5.

La información mostrada en el terminal del DCS viene ejemplificada en la Figura 14. El error "Fallo interno o de comunicación" viene dado por el intento fallido de codificación en UTF-8 del código binario, como se ha expuesto anteriormente. Tras

un nuevo intento de alteración, el mensaje se puede enviar codificado correctamente. Si al quinto intento no se completa la acción de corrupción y codificación correcta, se aborta la ejecución de este proceso, y el usuario tendrá que volver a enviar el request.

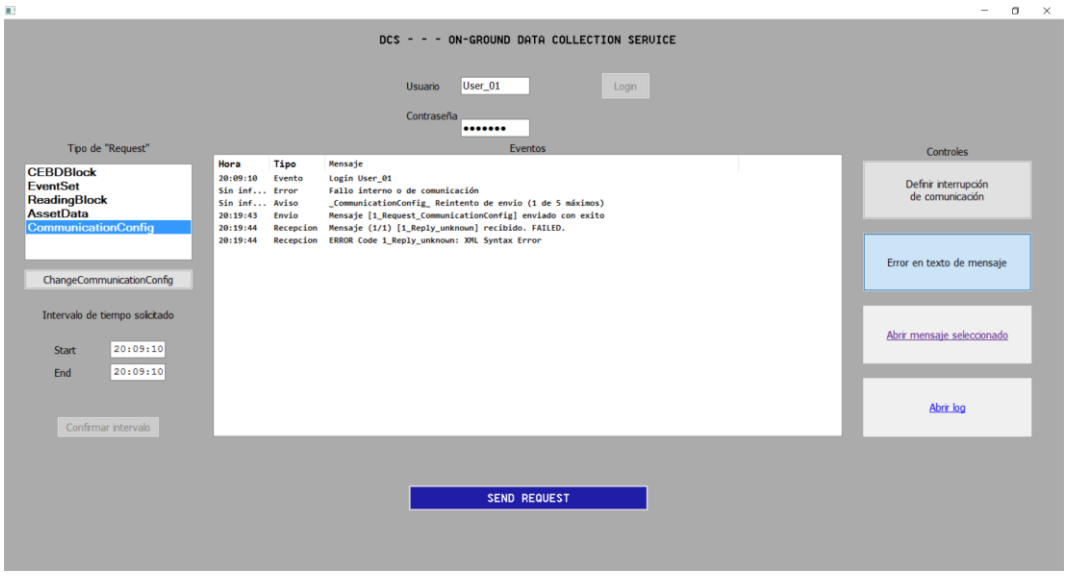


Figura 14: Ventana del DCS: alteración de mensajes y funcionalidad ante mensajes corruptos

5.3.18 PRUEBAS DEL SIMULADOR

Se han llevado a cabo pruebas tanto de las funcionalidades individuales implementadas como de la integración entre las mismas. Se pueden ver varios ejemplos en los vídeos demostrativos del simulador desarrollado que se encuentran en el enlace temporal: **https://1drv.ms/f/s!ApmS-4MSVjOLwkdu9eja0JKfZ3lq**

Los resultados de la comunicación pueden visualizarse a nivel informativo en las interfaces gráficas del simulador, y con más detalle en los directorios creados para el caso con los archivos de salida.

Capítulo 6 CONCLUSIONES Y APORTACIONES

Los resultados obtenidos satisfacen el objetivo primordial propuesto en este proyecto. Se ha logrado implementar un simulador de comunicaciones con las funcionalidades básicas y un cupo de las opcionales recogidas en la norma EN50463 para sistemas de medida de energía embarcados en tren. Entre los objetivos se incluía la posibilidad de implementar controles para el corte de comunicación, lo cual también ha sido simulado en Python con éxito.

Lógicamente algunas de las funcionalidades se han implementado en un aspecto concreto de la comunicación cuando deberían ser extendidas a todo el sistema, como la validación contra esquema XML, que solo se realiza en la estructura del mensaje de los requests recibidos en el EMS. Esta validación aplicaría también al contenido del payload, tanto de los replies como de los requests. De igual manera sucede con la compresión de XML, que se ha implementado a modo de ejemplo en el bloque concreto ChangeCommunicationConfig. La simplificación y legibilidad del código fuente y la necesidad de una lectura directa de los datos intercambiados por parte usuario (la compresión y codificación de texto impide la lectura por parte de usuario) han sido los motivos principales de esta decisión.

Añadir, asimismo, que este sistema admite amplias posibilidades de configuración que dificultan su inclusión en un trabajo de esta magnitud (p.ej. el EMS puede captar información sobre alarmas de sensores del tren y enviar estos datos a través del servicio que describe la configuración del mismo).

El objetivo pretendido, en cualquier caso, se ha cubierto mediante la creación de los dos programas, cliente y servidor (en realidad, tanto EMS como DCS pueden comportarse como servidores de datos, pero con el planteamiento propuesto para el simulador se ha desestimado el lanzamiento de otro hilo de ejecución paralelo, que sería necesario, pues la librería empleada no consideraba esta configuración), su lógica interna y el proceso de la comunicación.

El sistema que propone la norma constituye un paso más hacia el objetivo de la interoperabilidad ferroviaria, y mediante este trabajo se da a conocer la funcionalidad de sus elementos en una versión avanzada de la misma, a la espera de la definitiva, que aún se demorará unos meses.

Este trabajo también describe a grandes rasgos el proceso de formación, interpretación y validación de XML mediante el módulo `lxml` para Python 2.7.

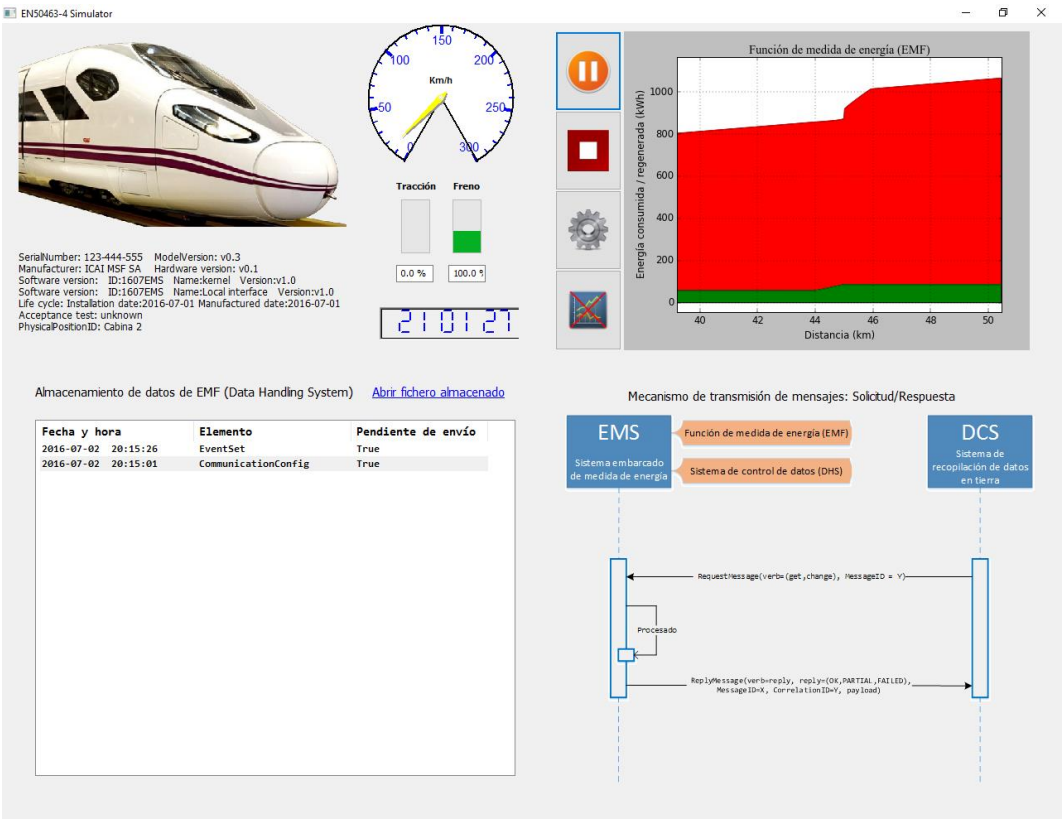
También se ha empleado un módulo específico para transmitir información por el protocolo XML-RPC que hace uso a su vez de HTTP.

El simulador implementado puede servir como un primer borrador para futuros desarrolladores de sistemas EMS y DCS.

Parte II ANEXOS

Capítulo 1 MANUAL DE USUARIO

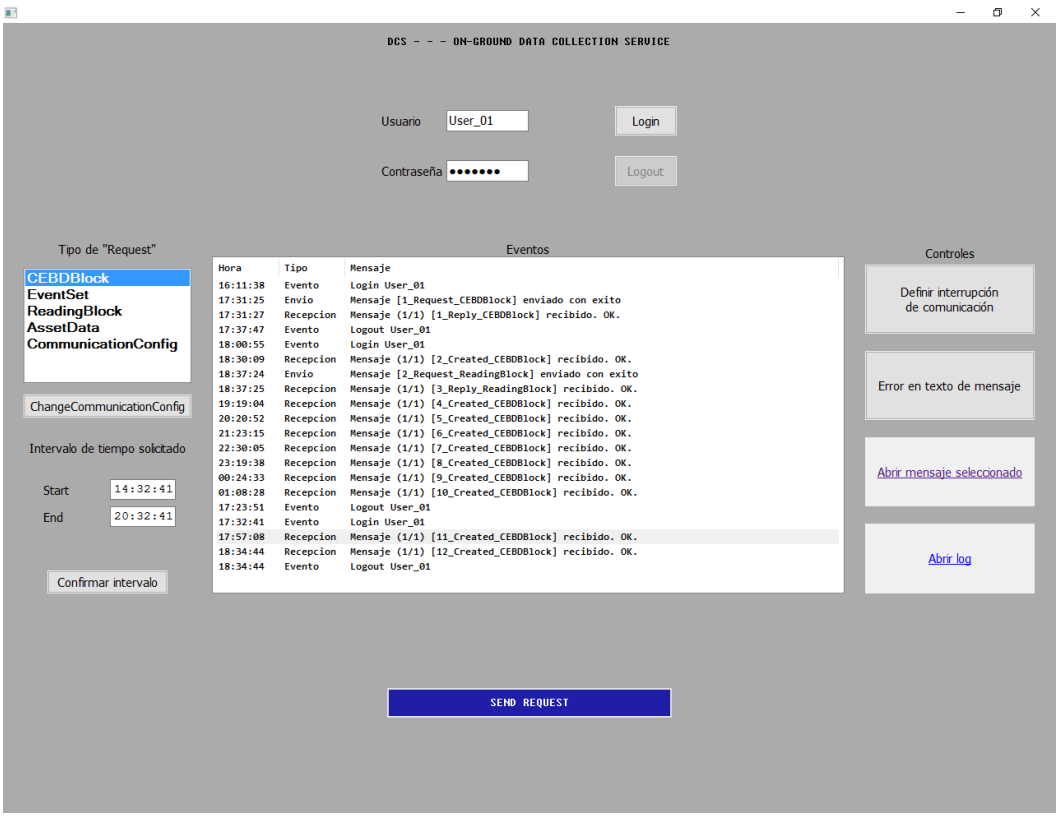
Una vez el programa se encuentre listo para su ejecución, se lanzará la aplicación principal.



En esta ventana, el usuario podrá configurar la simulación pulsando en la rueda dentada y navegando a la pestaña “Simulación” (velocidad, actualización de la gráfica) y pulsar el botón “Play”, de modo que el usuario visualizará el EMS de un tren capturando datos del recorrido del tren.

La barra de botones situada junto a la gráfica permite pausar y reanudar la simulación del recorrido del tren, detener la ejecución, abrir la ventana de configuración y activar o desactivar el refresco de la gráfica (para casos en los que el programa demande demasiados recursos de la máquina).

La ejecución del programa del DCS (equipo de adquisición de datos) abre la interfaz de usuario del DCS:



Para conectar el equipo al EMS habrá que introducir el usuario y contraseña predeterminados y pulsar el botón “Login”. Si se recibe en el cuadro de eventos el mensaje “Login User_01”, la conexión se habrá realizado correctamente.

A medida que vaya pasando el tiempo en la simulación, el EMS irá generando datos que pueden ser solicitados por el DCS. Para que el usuario del simulador pueda solicitar datos, existen varios servicios a su disposición entre los que se puede seleccionar: CEBDBlock, EventSet, ReadingBlock y CommunicationConfig, en el cuadro tipo lista de opciones del panel izquierdo de la ventana del DCS. Los tres primeros servicios solicitados requerirán que el solicitante especifique el periodo de tiempo del que se desean los datos. Para ello, cuenta con dos cuadros para seleccionar la hora. Para que se habilite el botón de envío de la solicitud (Send Request) será necesario pulsar el botón “Confirmar intervalo”. En caso de que el intervalo introducido no sea correcto (la hora de comienzo sea más tardía que la

hora de fin), se restablecen los valores de los cuadros y el botón de envío queda deshabilitado.

Una vez sea enviado el request (petición) al EMS solicitando datos por medio de un servicio de los antes mencionados, el EMS generará una respuesta con los datos almacenados en el DHS (cuadro inferior izquierdo de la ventana principal de la simulación) que enviará por los mismos métodos al DCS, que podrá visualizar la sintaxis interna de los mensajes enviados y recibidos, por medio del cuadro central de eventos de la ventana del DCS, seleccionando una opción y pulsando en el botón del panel izquierdo: "Abrir mensaje seleccionado".

El usuario puede modificar la configuración de la comunicación pulsando en la rueda dentada de la ventana del EMS. Puede cambiarse el modo de transferencia de mensajes, pudiendo ser: solicitud / respuesta o transferencia automática (suscripción de eventos).

También pueden realizarse ajustes de la configuración a través de la ventana para formular la petición ChangeCommunicationConfig del DCS (botón homónimo), por la cual el DCS solicita un cambio de configuración al EMS sobre los permisos de los usuarios y los parámetros ajustables de la captura de datos. En la pestaña EMS de esta ventana de configuración se pueden modificar los servicios permitidos para el usuario. Si se habilitan los servicios correspondientes y está activo el modo de transferencia automática, el usuario verá como recibe mensajes automáticos del EMS a medida que van recopilándose los datos.

Finalmente, existen dos controles alternativos en la ventana del DCS a disposición de usuario para simular problemas de conexión entre el EMS (equipo embarcado) y el DCS (equipo en tierra). Mediante el botón interrumpir comunicación se abrirá una ventana para definir un intervalo de tiempo para el cual se desea simular una zona ciega a efectos de la comunicación. Cuando transcurre el tiempo necesario en la simulación, el DCS detecta el problema e intenta reconectar cada cierto tiempo, hasta que finalmente lo consigue tras pasar la hora final programada. En ese

instante, el usuario advertirá en la ventana de eventos del DCS la llegada de los mensajes (en modo de transferencia automática) que no habían podido ser enviados y que por tanto se habrían puesto en cola para ser enviados tras el restablecimiento.

El último control permite introducir un fallo a la hora de enviar peticiones al EMS desde el DCS. Al pulsar el interruptor, el programa captará el intento de envío de request e introducirá un fallo aleatorio en el texto del mensaje, de modo que se pueda observar el comportamiento del EMS ante dicha situación. El usuario del DCS deberá advertir la recepción de un mensaje con el código FAILED y la información detallada del fallo producido en la recepción del mensaje desde la perspectiva del EMS.

Para finalizar la conexión, el usuario debe pulsar el botón “Logout” para que el EMS borre al usuario del registro de usuarios en línea.

Capítulo 2 EJEMPLOS DE ESTRUCTURAS XML Y ESQUEMAS PARA VALIDACIÓN

2.1 DATOS COMPILADOS

2.1.1 CEBDBLOCK.XML

```
<?xml version='1.0' encoding='UTF-8'?>
   <CEBDBlock xmlns="http://www.cenelec.eu//EN50463#"
xsi:schemaLocation="http://www.cenelec.eu//EN50463# CEBDBlock.xsd">
      <EMS>
        <CPID>
          <NVR>ES</NVR>
          <VKM>RENFE</VKM>
          <EVN>123456789012</EVN>
          <EMSID>1</EMSID>
        </CPID>
        <TRP>PT5M</TRP>
        <XMLVersion>1.0</XMLVersion>
      </EMS>
      <MeasurementInterval>
        <Start>2016-07-03T16:04:27Z</Start>
        <End>2016-07-03T17:04:27Z</End>
      </MeasurementInterval>
      <CEBD>
        <CEBDID>1</CEBDID>
        <TimeStamp>2016-07-03T16:09:27Z</TimeStamp>
        <TimeStampQuality>127</TimeStampQuality>
        <Channel>
          <ChannelID>1</ChannelID>
          <TractionSystem>01</TractionSystem>
          <Energy>
            <Active>
              <Consumed>177.1</Consumed>
              <Regenerated>0.0</Regenerated>
            </Active>
            <Reactive>
              <Consumed>31.3</Consumed>
              <Regenerated>0.0</Regenerated>
            </Reactive>
            <Quality>127</Quality>
          </Energy>
        </Channel>
        <Location>
          <Latitude>-2.98748</Latitude>
          <Longitude>43.28451</Longitude>
          <Quality>127</Quality>
        </Location>
```

```
    </CEBD>
    <CEBD>
      <CEBDID>2</CEBDID>
      <TimeStamp>2016-07-03T16:14:27Z</TimeStamp>
      <TimeStampQuality>127</TimeStampQuality>
      <Channel>
        <ChannelID>1</ChannelID>
        <TractionSystem>01</TractionSystem>
        <Energy>
          <Active>
            <Consumed>94.6</Consumed>
            <Regenerated>28.6</Regenerated>
          </Active>
          <Reactive>
            <Consumed>16.7</Consumed>
            <Regenerated>5.0</Regenerated>
          </Reactive>
          <Quality>127</Quality>
        </Energy>
      </Channel>
      <Location>
        <Latitude>-2.71258</Latitude>
        <Longitude>43.21202</Longitude>
        <Quality>127</Quality>
      </Location>
    </CEBD>
      .
      .
      .
     <CEBD>
      <CEBDID>12</CEBDID>
      <TimeStamp>2016-07-03T17:04:27Z</TimeStamp>
      <TimeStampQuality>127</TimeStampQuality>
      <Channel>
        <ChannelID>1</ChannelID>
        <TractionSystem>01</TractionSystem>
        <Energy>
          <Active>
            <Consumed>86.6</Consumed>
            <Regenerated>156.8</Regenerated>
          </Active>
          <Reactive>
            <Consumed>15.3</Consumed>
            <Regenerated>27.7</Regenerated>
          </Reactive>
          <Quality>127</Quality>
        </Energy>
      </Channel>
      <Location>
        <Latitude>0.48378</Latitude>
        <Longitude>42.32944</Longitude>
        <Quality>127</Quality>
      </Location>
    </CEBD>
    <Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <ds:SignedInfo>
        <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-
c14n-20010315"/>
        <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-
more#rsa-sha256"/>
        <ds:Reference URI="">
          <ds:Transforms>
            <ds:Transform
Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
          </ds:Transforms>
          <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
          <ds:DigestValue>DIGESTVALUE6qs3sBP4=</ds:DigestValue>
        </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>SIGNATUREVALUEF1K9A=</ds:SignatureValue>
      <ds:KeyInfo>
```

```
          <ds:KeyValue>
            <ds:RSAKeyValue>
              <ds:Modulus>PUBLICKEYMODULUSUNEjd3hKQak=</ds:Modulus>
              <ds:Exponent>AQAB</ds:Exponent>
            </ds:RSAKeyValue>
          </ds:KeyValue>
        </ds:KeyInfo>
      </Signature>
    </CEBDBlock>
```

2.1.2 READINGBLOCK.XML

```
<?xml version='1.0' encoding='UTF-8'?>
<ReadingBlock xmlns="http://www.cenelec.eu//EN50463#"
xsi:schemaLocation="http://www.cenelec.eu//EN50463# ReadingBlock.xsd">
      <EMS>
        <CPID>
          <NVR>ES</NVR>
          <VKM>RENFE</VKM>
          <EVN>123456789012</EVN>
          <EMSID>1</EMSID>
        </CPID>
        <TimePeriod>PT5M</TimePeriod>
        <XMLVersion>1.0</XMLVersion>
      </EMS>
      <MeasurementInterval>
        <Start>2016-07-03T16:04:27Z</Start>
        <End>2016-07-03T17:04:27Z</End>
      </MeasurementInterval>
      <Reading>
        <ReadingID>1</ReadingID>
        <TimeStamp>2016-07-03T16:05:27Z</TimeStamp>
        <TimeStampQuality>127</TimeStampQuality>
        <Channel>
          <ChannelID>1</ChannelID>
          <TractionSystem>01</TractionSystem>
          <Energy>
            <Active>
              <Consumed>87.8</Consumed>
              <Regenerated>0.0</Regenerated>
            </Active>
            <Reactive>
              <Consumed>15.5</Consumed>
              <Regenerated>0.0</Regenerated>
            </Reactive>
            <Quality>127</Quality>
          </Energy>
        </Channel>
        <Location>
          <Latitude>-3.17925</Latitude>
          <Longitude>43.35370</Longitude>
          <Quality>127</Quality>
        </Location>
        <AssociatedCEBD>1</AssociatedCEBD>
        <Speed>
          <Maximum>
            <Value>26.86</Value>
            <Units>m/s</Units>
          </Maximum>
        </Speed>
      </Reading>
      <Reading>
        <ReadingID>2</ReadingID>
        <TimeStamp>2016-07-03T16:06:27Z</TimeStamp>
        <TimeStampQuality>127</TimeStampQuality>
```

```
        <Channel>
          <ChannelID>1</ChannelID>
          <TractionSystem>01</TractionSystem>
          <Energy>
            <Active>
              <Consumed>29.3</Consumed>
              <Regenerated>0.0</Regenerated>
            </Active>
            <Reactive>
              <Consumed>5.2</Consumed>
              <Regenerated>0.0</Regenerated>
            </Reactive>
            <Quality>127</Quality>
          </Energy>
        </Channel>
        <Location>
          <Latitude>-3.12867</Latitude>
          <Longitude>43.34169</Longitude>
          <Quality>127</Quality>
        </Location>
        <AssociatedCEBD>1</AssociatedCEBD>
        <Speed>
          <Maximum>
            <Value>29.22</Value>
            <Units>m/s</Units>
          </Maximum>
        </Speed>
      </Reading>
         .
         .
         .
      <Reading>
        <ReadingID>60</ReadingID>
        <TimeStamp>2016-07-03T17:04:27Z</TimeStamp>
        <TimeStampQuality>127</TimeStampQuality>
        <Channel>
          <ChannelID>1</ChannelID>
          <TractionSystem>01</TractionSystem>
          <Energy>
            <Active>
              <Consumed>17.5</Consumed>
              <Regenerated>15.3</Regenerated>
            </Active>
            <Reactive>
              <Consumed>3.1</Consumed>
              <Regenerated>2.7</Regenerated>
            </Reactive>
            <Quality>127</Quality>
          </Energy>
        </Channel>
        <Location>
          <Latitude>0.48378</Latitude>
          <Longitude>42.32944</Longitude>
          <Quality>127</Quality>
        </Location>
        <AssociatedCEBD>12</AssociatedCEBD>
        <Speed>
          <Maximum>
            <Value>21.5</Value>
            <Units>m/s</Units>
          </Maximum>
        </Speed>
      </Reading>
</ReadingBlock>
```

2.1.3 EVENTSET.XML

```xml
<?xml version='1.0' encoding='UTF-8'?>
<EventSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.cenelec.eu//EN50463#"
xsi:schemaLocation="http://www.cenelec.eu//EN50463# EventSet.xsd">
  <EMS>
    <CPID>
      <NVR>ES</NVR>
      <VKM>RENFE</VKM>
      <EVN>123456789012</EVN>
      <EMSID>1</EMSID>
    </CPID>
    <TRP>PT5M</TRP>
    <XMLVersion>1.0</XMLVersion>
  </EMS>
  <Event>
    <EventID>3</EventID>
    <Description>
      <Domain>User_01</Domain>
      <SubDomain>Remote</SubDomain>
      <EventOrAction>Login</EventOrAction>
    </Description>
    <TimeStamp>2016-07-03T17:32:40Z</TimeStamp>
    <TimeStampQuality>127</TimeStampQuality>
    <Location>
      <Latitude>-1.51075</Latitude>
      <Longitude>42.88017</Longitude>
      <Quality>127</Quality>
    </Location>
  </Event>
</EventSet>
```

2.1.4 COMMUNICATIONCONFIG.XML

```xml
<?xml version='1.0' encoding='UTF-8'?>
<CommunicationConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.cenelec.eu//EN50463#"
xsi:schemaLocation="http://www.cenelec.eu//EN50463# CommunicationConfig.xsd">
  <CPID>
    <NVR>ES</NVR>
    <VKM>RENFE</VKM>
    <EVN>123456789012</EVN>
    <EMSID>1</EMSID>
  </CPID>
  <TRP>PT5M</TRP>
  <XMLVersion>1.0</XMLVersion>
  <User>
    <UserID>2</UserID>
    <MessageHeader>
      <Source>energia.renfe.es</Source>
      <UserID>User_01</UserID>
      <Organization>Renfe Operadora</Organization>
    </MessageHeader>
    <Access>RemoteAndLocal</Access>
    <Rights>ChangeOwn</Rights>
    <Service>
      <Get>
        <EventSet>True</EventSet>
        <AssetData>True</AssetData>
        <ReadingBlock>True</ReadingBlock>
```

```
        <CEBDBlock>True</CEBDBlock>
        <CommunicationConfig>True</CommunicationConfig>
      </Get>
      <Spontaneous>
        <EventSet>False</EventSet>
        <AssetData>False</AssetData>
        <ReadingBlock>False</ReadingBlock>
        <CEBDBlock>True</CEBDBlock>
        <CommunicationConfig>False</CommunicationConfig>
      </Spontaneous>
      <Change>
        <ChangeCommunicationConfig>True</ChangeCommunicationConfig>
      </Change>
    </Service>
    <Protocol>
      <XMLRPC>
        <EmsPort>8030</EmsPort>
        <GroundPort>80</GroundPort>
        <Web>emsadminstration.adif.es</Web>
        <Compression>False</Compression>
      </XMLRPC>
    </Protocol>
  </User>
  <DataGeneration>
    <CEBDBlockConfig>
      <MaxCEBD>12</MaxCEBD>
      <MaxCEBDBlock>4</MaxCEBDBlock>
    </CEBDBlockConfig>
    <ReadingBlockConfig>
      <MaxReading>60</MaxReading>
      <MaxReadingBlock>4</MaxReadingBlock>
    </ReadingBlockConfig>
  </DataGeneration>
</CommunicationConfig>
```

2.1.5 CHANGECOMMUNICATIONCONFIG.XML (DATA GENERATION)

```
<?xml version='1.0' encoding='UTF-8'?>
<ChangeCommunicationConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.cenelec.eu//EN50463#"
xsi:schemaLocation="http://www.cenelec.eu//EN50463# CEBDBlock.xsd"><CPID>
                <NVR>ES</NVR>
                <VKM>RENFE</VKM>
                <EVN>123456789012</EVN>
                <EMSID>1</EMSID>
        </CPID>
        <TRP>PT5M</TRP>
        <XMLVersion>1.0</XMLVersion>
        <DataGeneration>
                <CEBDBlockConfig>
                        <MaxCEBD>12</MaxCEBD>
                        <MaxCEBDBlock>4</MaxCEBDBlock>
                </CEBDBlockConfig>
                <ReadingBlockConfig>
                        <MaxReading>15</MaxReading>
                        <MaxReadingBlock>4</MaxReadingBlock>
                </ReadingBlockConfig>
        </DataGeneration>
</ChangeCommunicationConfig>
```

2.2 MENSAJES

2.2.1 CEBDBLOCK REQUEST

```
<?xml version='1.0' encoding='UTF-8'?>
<RequestMessage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.cenelec.eu//EN50463#"
xsi:schemaLocation="http://www.cenelec.eu//EN50463# CEBDBlock.xsd">
  <Header>
    <Verb>get</Verb>
    <Noun>CEBDBlock</Noun>
    <Revision>1.0</Revision>
    <Timestamp>2016-07-04T00:22:01Z</Timestamp>
    <Source>User_01.adif.es</Source>
    <User>
      <UserID>User_01</UserID>
    </User>
    <MessageID>1</MessageID>
    <Target>1234567890123.adif.es</Target>
  </Header>
  <Request>
    <StartTime>2016-07-04T00:21:01Z</StartTime>
    <EndTime>2016-07-04T00:22:01Z</EndTime>
  </Request>
</RequestMessage>
```

2.2.2 CHANGECOMMUNICATIONCONFIG REQUEST

```
<?xml version='1.0' encoding='UTF-8'?>
<RequestMessage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.cenelec.eu//EN50463#"
xsi:schemaLocation="http://www.cenelec.eu//EN50463#
ChangeCommunicationConfig.xsd">
  <Header>
    <Verb>change</Verb>
    <Noun>ChangeCommunicationConfig</Noun>
    <Revision>1.0</Revision>
    <Timestamp>2016-07-04T00:22:01Z</Timestamp>
    <Source>User_01.adif.es</Source>
    <User>
      <UserID>User_01</UserID>
    </User>
    <MessageID>2</MessageID>
    <Target>1234567890123.adif.es</Target>
  </Header>
  <Payload>
    <Compressed>-- Compressed ChangeCommunicationConfig  --</Compressed>
    <Format>XML</Format>
  </Payload>
</RequestMessage>
```

2.2.3 REPLY OK

```
<?xml version='1.0' encoding='UTF-8'?>
<ReplyMessage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.cenelec.eu//EN50463#"
xsi:schemaLocation="http://www.cenelec.eu//EN50463# ReadingBlock.xsd">
  <Header>
    <Verb>reply</Verb>
    <Noun>ReadingBlock</Noun>
    <Revision>1.0</Revision>
    <TimeStamp>2016-07-04T22:45:44Z</TimeStamp>
    <Source>User_01.adif.es</Source>
    <MessageID>7</MessageID>
    <Target>dcs.adif.es</Target>
  </Header>
  <Reply>
    <Result>OK</Result>
  </Reply>
  <Payload>
    <ReadingBlock xmlns="http://www.cenelec.eu//EN50463#"
xsi:schemaLocation="http://www.cenelec.eu//EN50463# ReadingBlock.xsd">
      ...
    </ReadingBlock>
  </Payload>
</ReplyMessage>
```

2.2.4 REPLY FAILED POR ERROR DE SINTAXIS

```
<?xml version='1.0' encoding='UTF-8'?>
<ReplyMessage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.cenelec.eu//EN50463#"
xsi:schemaLocation="http://www.cenelec.eu//EN50463# unknown.xsd">
  <Header>
    <Verb>reply</Verb>
    <Noun>unknown</Noun>
    <Revision>1.0</Revision>
    <TimeStamp>2016-07-04T20:40:44Z</TimeStamp>
    <Source>User_01.adif.es</Source>
    <MessageID>1</MessageID>
    <CorrelationID>0</CorrelationID>
    <Target>dcs.adif.es</Target>
  </Header>
  <Reply>
    <Result>FAILED</Result>
    <Error>
      <Code>XML Syntax Error</Code>
    </Error>
    <XPath>Element '{http://www.cenelec.eu//EN50463#}Header': Character
content other than whitespace is not allowed because the content type is
'element-only'.</XPath>
  </Reply>
```

```
</ReplyMessage>
```

2.2.5 MENSAJES CORRUPTOS

Mensaje request original

```
<?xml version='1.0' encoding='UTF-8'?>
<RequestMessage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.cenelec.eu//EN50463#"
xsi:schemaLocation="http://www.cenelec.eu//EN50463#
CommunicationConfig.xsd">
  <Header>
    <Verb>get</Verb>
    <Noun>CommunicationConfig</Noun>
    <Revision>1.0</Revision>
    <Timestamp>2016-07-05T20:09:10Z</Timestamp>
    <Source>User_01.adif.es</Source>
    <User>
      <UserID>User_01</UserID>
    </User>
    <MessageID>1</MessageID>
    <Target>1234567890123.adif.es</Target>
  </Header>
</RequestMessage>
```

Mensaje request tras alteración aleatoria

```
<?xml version='1.0' encoding(z"8'?>
<RequestMessage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.cenelec.eu//EN50463#"
xsi:schemaLocation="http://www.cenelec.eu//EN50463#
CommunicationConfig.xsd">
  <Header>
    <Verb>get</Verb>
    <Noun>CommunicationConfig</Noun>
    <Revision>1.0</Revision>
    <Timestamp>2016-07-05T20:09:10Z</Timestamp>
    <Source>User_01.adif.es</Source>
    <User>
      <UserID>User_01</UserID>
    </User>
    <MessageID>1</MessageID>
    <Target>1234567890123.adif.es</Target>
  </Header>
</RequestMessage>
```

Mensaje reply informando del error de validación

```
<?xml version='1.0' encoding='UTF-8'?>
<ReplyMessage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.cenelec.eu//EN50463#"
xsi:schemaLocation="http://www.cenelec.eu//EN50463# unknown.xsd">
  <Header>
```

```
    <Verb>reply</Verb>
    <Noun>unknown</Noun>
    <Revision>1.0</Revision>
    <TimeStamp>2016-07-05T20:19:44Z</TimeStamp>
    <Source>User_01.adif.es</Source>
    <MessageID>1</MessageID>
    <CorrelationID>0</CorrelationID>
    <Target>dcs.adif.es</Target>
  </Header>
  <Reply>
    <Result>FAILED</Result>
    <Error>
      <Code>XML Syntax Error</Code>
    </Error>
    <XPath>line 1: parsing XML declaration: '?&gt;' expected</XPath>
  </Reply>
</ReplyMessage>
```

2.3 *ESQUEMAS XSD PARA VALIDACIÓN DE XML*

2.3.1 MENSAJE

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2015 rel. 3 sp1 (x64) (http://www.altova.com) by
Luis Blanco (Universidad Pontificia Comillas) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.cenelec.eu//EN50463#"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
targetNamespace="http://www.cenelec.eu//EN50463#"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="1.0">
      <xs:include schemaLocation="BaseTypes.xsd"/>
      <xs:annotation>
            <xs:documentation xml:lang="en">
                  Copyright (c) 2015 CENELEC. All rights reserved.
                  Version 1.0.
                  Definition based on IEC/EN 61968-100
      </xs:documentation>
      </xs:annotation>
      <xs:element name="RequestMessage" type="RequestMessageType"/>
      <xs:element name="ResponseMessage" type="ResponseMessageType"/>
      <xs:element name="EventMessage" type="EventMessageType"/>
      <xs:complexType name="ErrorType">
            <xs:sequence>
                  <xs:element name="Code" type="xs:string"/>
                  <xs:element name="XPath" type="xs:QName"
minOccurs="0"/>
            </xs:sequence>
      </xs:complexType>
      <xs:complexType name="EventMessageType">
            <xs:sequence>
                  <xs:element name="Header" type="HeaderType"/>
```

```
                        <xs:element name="Payload" type="PayloadType"
minOccurs="0"/>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="HeaderType">
                <xs:sequence>
                        <xs:element name="Verb" type="VerbType"/>
                        <xs:element name="Noun" type="xs:string"/>
                        <xs:element name="Revision" type="XMLVersionType"
default="1.0"/>
                        <xs:element name="Timestamp"
type="ZuluDateTimeType"/>
                        <xs:element name="Source" type="xs:string"/>
                        <xs:element name="User" type="UserType"
minOccurs="0"/>
                        <xs:element name="MessageID" type="xs:unsignedLong"/>
                        <xs:element name="CorrelationID"
type="xs:unsignedLong" minOccurs="0"/>
                        <xs:element name="Target" type="xs:string"/>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="PayloadType">
                <xs:sequence>
                        <xs:choice>
                                <xs:any namespace="##other"
processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
                                <xs:element name="Compressed" type="xs:string"
minOccurs="0" maxOccurs="unbounded"/>
                        </xs:choice>
                        <xs:element name="Format" type="xs:string"
default="XML" minOccurs="0"/>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="ReplyType">
                <xs:sequence>
                        <xs:element name="Result">
                                <xs:simpleType>
                                        <xs:restriction base="xs:string">
                                                <xs:enumeration value="OK"/>
                                                <xs:enumeration value="PARTIAL"/>
                                                <xs:enumeration value="FAILED"/>
                                        </xs:restriction>
                                </xs:simpleType>
                        </xs:element>
                        <xs:element name="Error" type="ErrorType"
minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="RequestMessageType">
                <xs:sequence>
                        <xs:element name="Header" type="HeaderType"/>
                        <xs:element name="Request" type="RequestType"
minOccurs="0"/>
                        <xs:element name="Payload" type="PayloadType"
minOccurs="0"/>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="RequestType">
                <xs:sequence>
                        <xs:element name="StartTime" type="ZuluDateTimeType"
minOccurs="0"/>
                        <xs:element name="EndTime" type="ZuluDateTimeType"
minOccurs="0"/>
```

```
                    <xs:element name="Password" type="xs:string"
minOccurs="0"/>
              </xs:sequence>
       </xs:complexType>
       <xs:complexType name="ResponseMessageType">
              <xs:sequence>
                    <xs:element name="Header" type="HeaderType"/>
                    <xs:element name="Reply" type="ReplyType"/>
                    <xs:element name="Payload" type="PayloadType"
minOccurs="0"/>
              </xs:sequence>
       </xs:complexType>
       <xs:complexType name="UserType">
              <xs:sequence>
                    <xs:element name="UserID" type="xs:string"/>
                    <xs:element name="Organization" type="xs:string"
minOccurs="0"/>
              </xs:sequence>
       </xs:complexType>
       <xs:simpleType name="VerbType">
              <xs:restriction base="xs:string">
                    <xs:enumeration value="change"/>
                    <xs:enumeration value="changed"/>
                    <xs:enumeration value="created"/>
                    <xs:enumeration value="get"/>
                    <xs:enumeration value="reply"/>
              </xs:restriction>
       </xs:simpleType>
</xs:schema>
```