Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

Panel de control de misión para un monitor de neutrones basado en OpenMCT

Autor: Ana Santos Loor

Tutor: Óscar García Población

2022

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

Panel de control de misión para un monitor de neutrones basado en OpenMCT

Autor: Ana Santos Loor

Tutor/es: Óscar García Población

TRIBUNAL:

Presidente: Juan Ignacio García Tejedor

Vocal 1º: Elena Campo Montalvo

Vocal 2º: Óscar García Población

FECHA: 2022

Agradecimientos

Agradezco de todo corazón a mi familia, en especial a mi pequeña Ainara y a mi madre, principal motor de mi vida y alegría de mi vida.

A mi tutor, por su infinita paciencia y apoyo a lo largo de todos los años. Óscar, mil gracias por todo.

Y a mi padre….. que ya camina sobre las estrellas..

Panel de control de misión para un monitor de neutrones basado en OpenMCT

1 Índice

2 Índice de figuras

3 Resumen

Este proyecto consiste en la creación de un panel de control de misiones que permita la visualización en tiempo real, de distintas métricas del Monitor de Neutrones CaLMa. Debido a la naturaleza del proyecto, se empleará una arquitectura basada en eventos con un modelo Publisher-Suscriptor, donde existirá un canal de comunicación bidireccional y full-dúplex sobre un único socket TCP entre el navegador y nuestro servidor. Se empleará OpenMCT como *framework* y los distintos *plugins* de la aplicación estarán desarrollados en NodeJS.

4 Summary

The objetive of this project is to design a Mission Control Panel to view in real time the metrics of the CalMa Neutron Monitor. I will use an event-driven architecture with a Publisher-Subscriber model, where there will be a full-duplex bidirectional communication channel over a single TCP socket between the browser and our server. *OpenMCT* will be used as a framework and the different plugins of the application will be developed in NodeJS.

5 Palabras claves

OpenMCT, plugins, dashboard, monitor de neutrones, web sockets.

6 Introducción

En este apartado del presente documento se hará una breve introducción sobre qué es un monitor de neutrones, la importancia que éstos tienen en el campo de la ciencia para la vigilancia a largo plazo de la variación de los rayos cósmicos, así como la obtención de alertas en la meteorología espacial. Se profundizará especialmente en el monitor de neutrones de CaLMa ya que en este proyecto se pretende desarrollar una aplicación que permita integrar las telemetrías obtenidas por CaLMa con un panel de control de misiones, mediante el cual se podrá supervisar de manera más rápida e intuitiva el correcto funcionamiento del monitor.

6.1 ¿Qué es un monitor de neutrones?

Un monitor de neutrones es un instrumento científico diseñado para medir la cantidad de rayos cósmicos que llegan a la tierra desde el espacio. A pesar de tener décadas de tradición, siguen siendo el referente para la medida de rayos cósmicos, teniendo un papel fundamental como instrumento de investigación en el campo de la astrofísica, de las relaciones entre el sol y la tierra y de las aplicaciones de la meteorología espacial.

Estos instrumentos son capaces de detectar rayos cósmicos que penetran en la atmósfera de la tierra con energías comprendidas entre 0.5 y 20 GeV, esto es, en un rango de energías que no pueden ser medidas con detectores situados en el espacio con la misma facilidad, economía y precisión estadística. Cuando estos rayos impactan con la atmósfera de la tierra desencadenan una cascada de partículas, mayoritariamente protones, muones, electrones y neutrones. Estos últimos son capaces de alcanzar la superficie de la tierra donde son detectados por los monitores de neutrones.

Existen dos tipos de monitores estandarizados que operan actualmente dentro de una red mundial compuesta de unas cincuenta estaciones. Los tipos de monitores son:

- IGY: Diseñado por Simpson en 1958, enfocado más para el estudio de la variación en el tiempo de la energía de los rayos cósmicos en el rango de lo GeV cerca de la tierra.
- NM46: Diseñado por Carmichael en 1964. Este monitor era más grande y tenía un número de cuentas mayor. Fue el detector de rayos cósmicos estándar en el congreso International Quiet Sun Year (IQSY) de 1964

Los monitores de neutrones están compuestos por unos tubos llenos de un gas especial, contadores proporcionales rodeados por un moderador, un material productor hecho de plomo y un reflector. Los nucleones que inciden en el monitor (protones y neutrones) procedentes de rayos cósmicos secundarios, provocan reacciones nucleares en el plomo. Estas reacciones generan neutrones de evaporación de baja energía. A su vez, el moderador reduce la velocidad de estos nuevos neutrones del rango de los MeV hasta el de energías térmicas. Finalmente, en el caso de los detectores NM64, cerca de un 6% de estos neutrones será detectado por los tubos contadores. El hecho de que finalmente sean neutrones lo que estos instrumentos registran es lo que les da el nombre de monitores de neutrones.

6.2 Proyecto CaLMa.

El Monitor de Neutrones de Castilla-La Mancha (CaLMa) es el primero que se instala en España y tiene como misión medir permanentemente la radiación extraterrestre que alcanza el suelo, para ello hace uso de 18 tubos contadores. Lleva en funcionamiento desde el año 2011, un año después se integró a la red de monitores de neutrones NMDB (Neutron Monitor Database).

Actualmente, consta de tres nodos de observación, dos con localizaciones fijas: CaLMa (Guadalajara) y ORCA (Isla Livingston, Antártica) y uno móvil: miniCaLMa. El Grupo de Investigación Espacial de la Universidad de Alcalá es el encargado del mantenimiento y administración de CaLMa aunque solo un reducido grupo de investigadores están dedicados a ello.

El monitor de neutrones CalMa no difiere en absoluto al resto de monitores en cuanto a los componentes se refieren. Se distinguen 4 elementos principales:

1. **Tubos Contadores con gas:**
   Los tubos de un monitor de neutrones detectan principalmente neutrones térmicos, es decir, aquellos que se mueven con una energía cinética de aproximadamente 0.025eV. La detección de un neutrón básicamente se produce porque éste interacciona con el gas contenido en los tubos desencadenando una reacción exotérmica.
   Inicialmente se utilizaba el trifloruro de boro (BF3), compuesto inorgánico, incoloro y enriquecido al 96% con el isótopo 10B, a una presión de 0.27 bar. En estos tipos de monitores, la reacción exotérmica resultante es la siguiente:

   $$^{10}B_5 + n \rightarrow {}^{7}Li_3 + {}^{4}He_2$$

   Desde 1990 se empezó a sustituir el BF3 por helio-3 (3He) dando como resultado un diseño más simple y eficiente, ya que el tubo puede operar con una presión del gas más alta y con una tensión inferior a 1500 V. La reacción exotérmica de los neutrones con el 3He es la siguiente:

   $$^{3}He_2 + n \rightarrow {}^{3}H_1 + proton$$

   El monitor CalMa está compuesto por 18 tubos de BF3.

2. **Moderador**:
   Si la velocidad con la que inciden los neutrones dentro de los tubos contadores es muy alta, es improbable que se llegue a desencadenar una reacción exotérmica. Para solventar este problema se introduce un moderador en donde los neutrones se ralentizan mediante colisiones con partículas cuyo núcleo sea similar al núcleo del neutrón. En los monitores del tipo IGY se emplea parafina y en los monitores del tipo NM64 se utiliza agua o polietileno. **En CaLMa el elemento utilizado es un tubo de polietileno de 2082,80 mm de longitud, 253 mm de diámetro y 40 mm de espesor**.

3. **Plomo Productor:**
   Es una capa que rodea al moderador. Tal como su nombre lo indica, es de plomo y su objetivo principal es aumentar la probabilidad de detección de neutrones. Esto se consigue mediante dos funcionalidades:
   - Generar neutrones de evaporación y neutrones de baja energía mediante reacciones nucleares provocadas por las colisiones entre las partículas energéticas incidentes y el plomo.

- Incrementar la probabilidad global de detección, ya que de media el número de neutrones de evaporación producidos por una partícula incidente que reacciona con el plomo es de 15.

En CalMa, el productor está compuesto por 1440 medias anillas de 99.9% de plomo.

4. Reflector:
   Es una armazón que contiene en su interior los tubos contadores, el moderador y el plomo productor. En los monitores del tipo NM64, el reflector es de polietileno mientras que en los monitores IGY es de parafina. Se denomina reflector porque una de sus funcionalidades principales es la de moderar y reflejar los neutrones de evaporación producidos en el plomo hacia los tubos contadores. También aísla y absorbe neutrones de baja energía externos al monitor, evitando así que éstos alteren los resultados del monitor.
   El reflector de CalMa está formado por varias capas de polietileno, cada una de 75 mm de espesor.

CaLMa está formado por tres conjuntos de 6 detectores, dos de ellos con el nuevo SK01497 y un tercero con BP28. Todos encerrados en una estructura de polietileno y plomo. Los amplificadores de los SK01497 son Canberra ACHNP97 y los de los BP28 son Chalk River. Los contadores LND SK01479 y BP28 son muy similares, ambos están llenos del gas 10BF3 a una presión de 200 mm Hg.

6.3 NMDB: Real-time Neutron Monitor Database

La base de datos de monitores de neutrones en tiempo real *(NMDB – the real-time **N**eutron **M**onitor **D**ata**B**ase*) es una red mundial de monitores de neutrones estandarizados que se utiliza para registrar variaciones de los rayos cósmicos primarios y medidas de rayos cósmicos basadas en el espacio. En el año 2012, CaLMa pasó a formar parte de esta red mundial.

Actualmente existen muchos proyectos que proporcionan los datos de sus experimentos satelitales en tiempo real, con una alta fiabilidad temporal, una máxima resolución y precisión (los datos son mostrados en gráficas que se actualizan a medida que se reciben datos). Sin embargo, los datos del monitor de neutrones nunca habían estado disponibles desde muchas estaciones en tiempo real. Es decir, hasta hace poco cada estación almacenaba de manera individual sus propias mediciones en diferentes formatos y tenía completa libertad a la hora de decidir si estos datos se publicaban o no en su sitio web. Para superar este déficit, la Comisión Europea está apoyando la NMDB como un proyecto de infraestructuras electrónicas dentro del *Seventh Framework Programme* en la sección de Capacidades. Las estaciones que no tengan una resolución de 1 minuto contarán con el apoyo del desarrollo de un sistema de registro estándar asequible que enviará las mediciones a la base de datos a través de Internet en tiempo real. Esto resuelve el problema de los diferentes formatos de datos y, por primera vez, permite utilizar mediciones de rayos cósmicos en tiempo real para las predicciones del clima espacial.

Además de crear una base de datos y desarrollar aplicaciones que trabajen con estos datos, una parte del proyecto se dedica a crear un sitio web de divulgación pública para informar sobre los rayos cósmicos y los posibles efectos en los humanos, los sistemas tecnológicos y el medio ambiente.

A continuación, se muestra una gráfica los datos de las estaciones integradas a la red de NMDB.

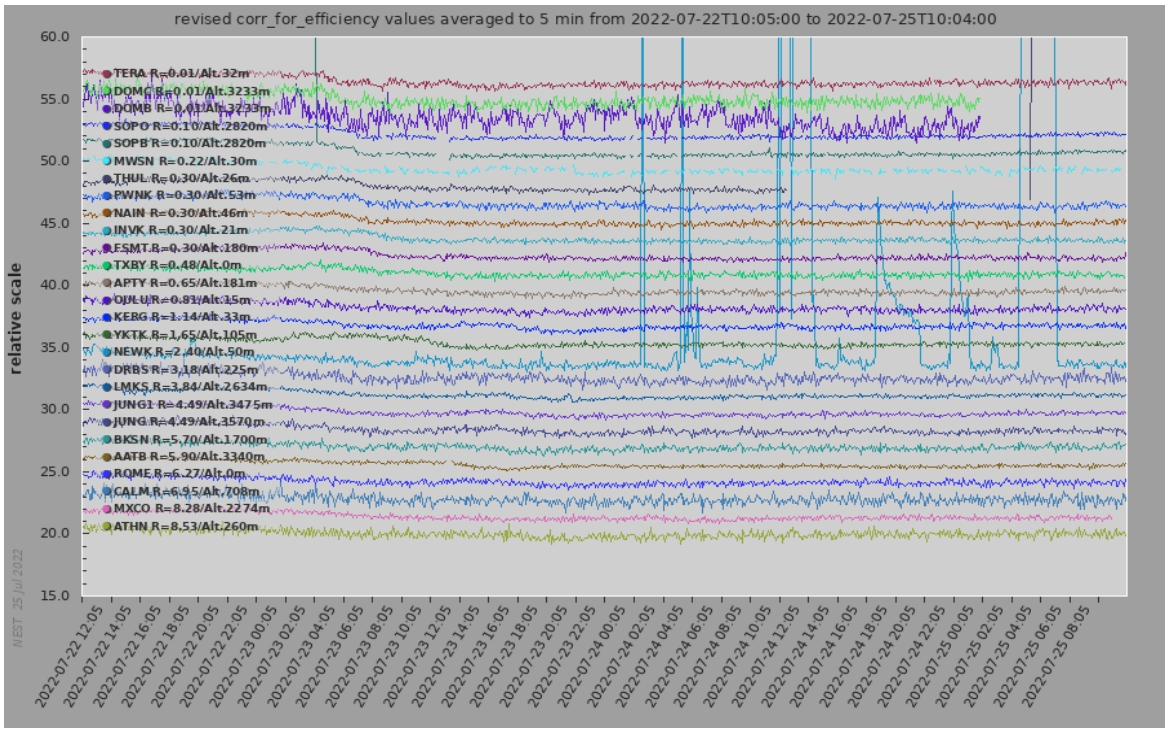


Ilustración 1 Tráfico de datos de las estaciones de la red NMDB

6.4 Objetivos

Uno de los instrumentos más importantes de un monitor de neutrones es su sistema de adquisición de datos, ya que éste es el encargado de registrar ciertos parámetros correspondientes a fenómenos físicos como pueden ser la presión atmosférica, la temperatura, la tensión, así como el número de neutrones detectados en cada minuto, siendo éste el dato más importante a tener en cuenta. Esta información se almacena en la base de datos de tiempo real (NMD) descrita anteriormente.

Muchos de los sistemas modernos de adquisición de datos en tiempo real, aparte de disponer de una base de datos actualizada de manera online, cuentan además con un sistema que les permiten la visualización de los datos obtenidos en dispositivos móviles (apps) y de escritorio (aplicaciones web). Un buen ejemplo de estos sistemas, es el empleado en el proyecto Rover (NASA). Los "*rovers*" son robots que la NASA envía al espacio para recopilar información. Las telemetrías que obtienen son utilizadas para el análisis de datos de sus misiones espaciales, así como para la planificación y operación de sistemas experimentales para mejoras del Rover.

A continuación, se muestra una gráfica con los datos obtenidos del clima de Marte.

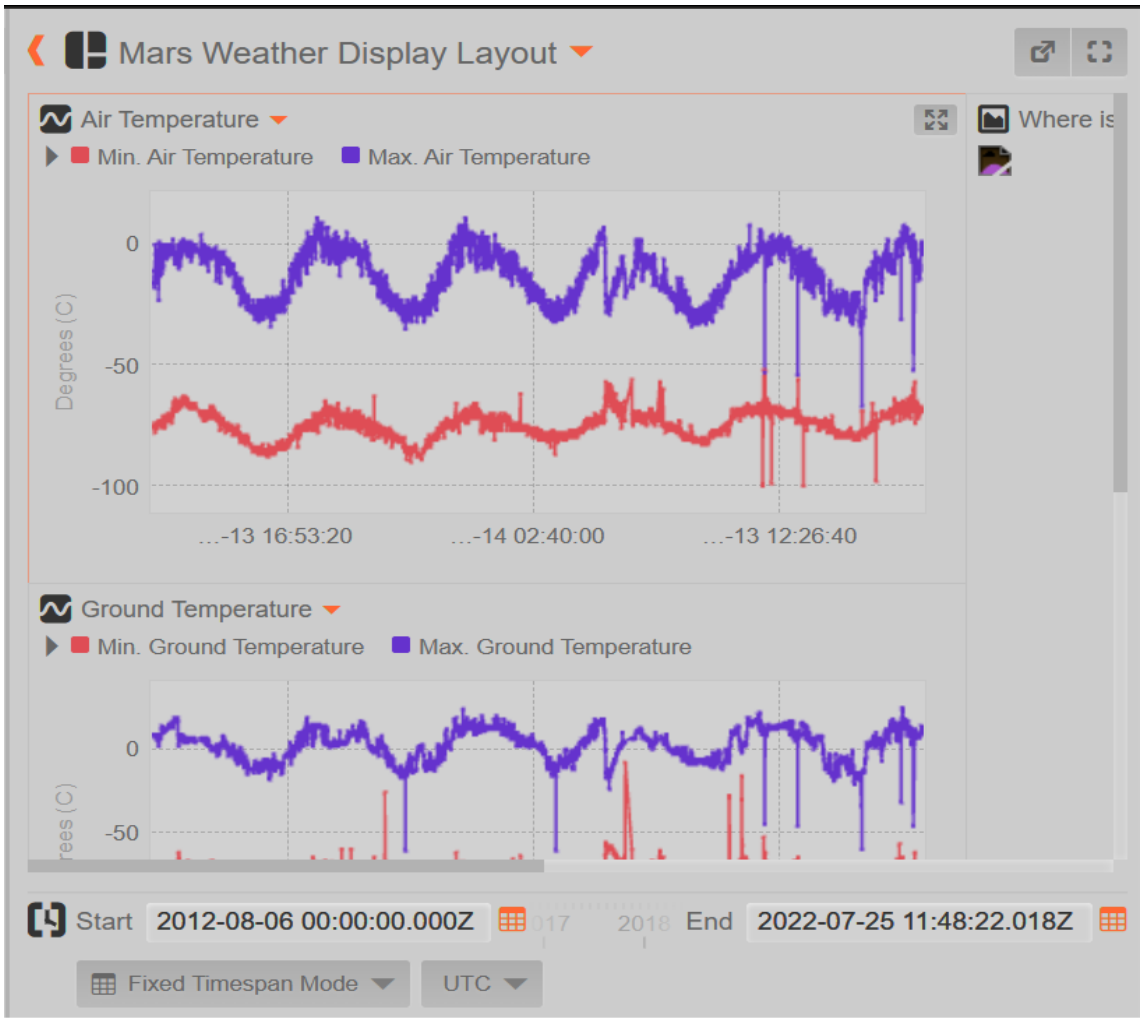


Ilustración 2: Telemetrías del clima de Marte

El objetivo principal de este proyecto es desarrollar un software que integre el sistema de adquisición de datos de CaLMa con un *framework* que permite la visualización de las telemetrías obtenidas en tiempo real en un navegador con el fin de facilitar la manipulación y análisis de datos, obtener estadísticas de una manera más visual y automática para su posterior estudio, detectar anomalías en el funcionamiento de los tubos contadores o de los receptores de datos meteorológicos.

7 Análisis del problema

El sistema de adquisición de datos de CaLMa se encarga de registrar los impactos provocados por los rayos cósmicos al incidir en los tubos detectores, es decir registra el número de neutrones por minuto que impactan en los 18 tubos contadores con gas. Adicionalmente también almacena factores externos denominados datos meteorológicos.

Los datos se generan en diferentes momentos del tiempo, típicamente cada minuto, pero pueden darse casos en el que éstos llegaran con resoluciones temporales más pequeñas.

Una parte de la información que se recopila se almacena en local, mientras que los datos que cumplen con el estándar establecido por la NMDB, siguen otro flujo diferente y acaban siendo almacenados en la base de datos online de NMDB y publicados también en internet para disposición de los usuarios.

La cuestión aquí es que la información que se recopila en tiempo real sólo se almacena, pudiendo ser consultada una vez que se publica en la base de datos Online, desperdiciándose así su gran uso potencial, como puede ser:

- Detectar anomalías de funcionamiento (p.e. un tubo que deja de contar, ausencia de datos de meteorología, etc.)
- Elaboración de métricas en tiempo real, así como paneles informativos sobre el funcionamiento del instrumento.
- Alimentación de sistemas de alertas: si un canal deja de emitir datos, se activaría una alerta indicando el fallo.
- Trabajar con datos que se puedan visualizar en tiempo real independientemente del punto del planeta en el que éstos han sido obtenidos.

Para solventar este problema, en este proyecto lo que se pretende abordar es el desarrollo de un software cuyas funcionalidades serían las siguientes:

- Acceder a las telemetrías en *streaming* de un instrumento científico, en nuestro caso el monitor de neutrones de CaLMa. La palabra *streaming* es un anglicismo utilizado para indicar que algo fluye de manera continua sin interrupción, habitualmente hace referencia a la difusión de vídeo o de audio. En este caso, cuando se habla de un acceso a telemetrías en *streaming* significa que la aplicación accederá en tiempo real a los datos proporcionados por los tubos contadores y medidores meteorológicos que se generan cada vez que los neutrones inciden en el monitor CaLMa. A su vez, la aplicación desarrollada permitirá que el usuario disponga de estos datos de manera instantánea, en tiempo real y de manera visual.

- Procesar dichas telemetrías obtenidas con el mínimo retardo posible sin necesidad de recurrir a consultas de base de datos, de tal manera que podamos tener una actualización en tiempo real del estado y funcionamiento de los medidores meteorológicos y de neutrones.

- Permitir la publicación en tiempo real de las telemetrías independientemente del punto de la fuente de obtención de datos, es decir, en nuestro caso hay que recordar que CaLMa posee una estación en la Antártida que recopila información. A nuestra aplicación, esta característica le resulta totalmente indiferente ya que recoge los datos en *streaming* y los publica en un navegador de manera inmediata.

- Componer paneles de mando con el objetivo de poder supervisar de un solo vistazo toda una sesión de tomas de datos. Es decir, la aplicación generará gráficas de las telemetrías que se irán actualizando a medida que lleguen los datos sin necesidad de realizar un refresco de página. Este punto es sin duda alguna el más interesante del proyecto. Para un investigador, tener una visión global de los datos recibidos sin necesidad de tener conocimientos de base de datos o de manejo de aplicaciones complejas para efectuar consultas, simplemente con ver los datos reflejados en la pantalla, le ayudará a determinar en todo momento si los contadores telemétricos están funcionando correctamente, si hay anomalías en los datos recibidos, si es necesario efectuar tareas de mantenimiento en alguno de los detectores de neutrones o de los receptores meteorológicos, etc. Para un usuario normal o alguien que recién empieza a trabajar en el proyecto, poder visualizar los datos le ayudará a entender mejor el funcionamiento de la estación.

En la siguiente ilustración se muestra el esquema actual de la obtención y tratamiento de las telemetrías de CaLMa. La parte rosa ilustra de manera esquemática la resolución del problema.

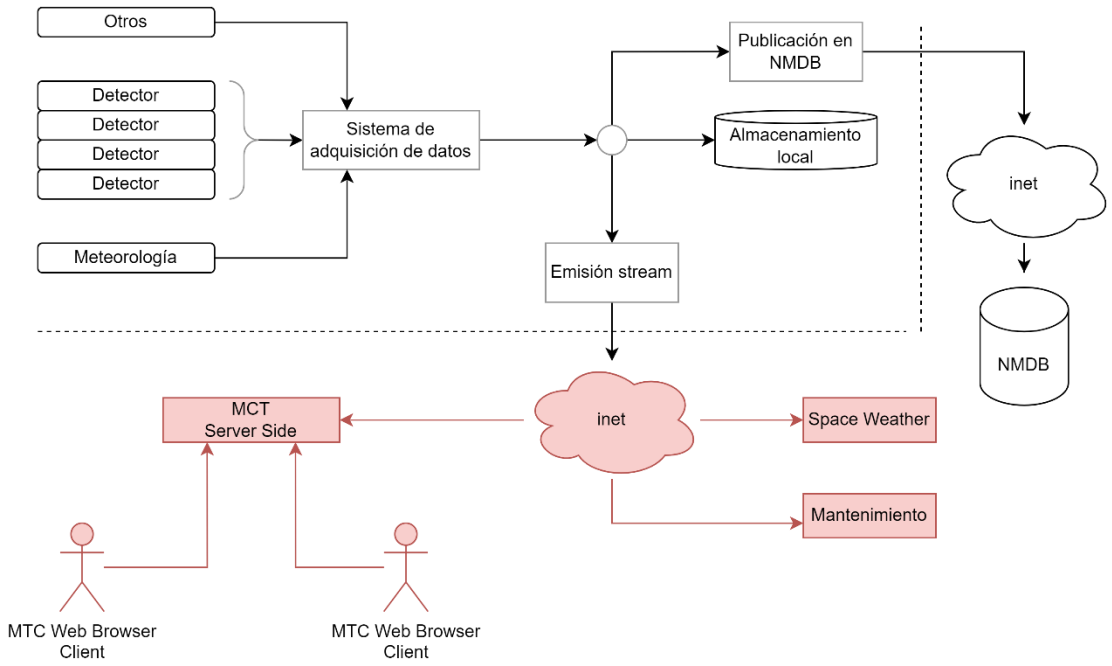


Ilustración 3: Esquema actual del flujo de datos, en rosa se indica el ámbito del proyecto.

7.1 Naturaleza de los datos de un monitor de neutrones

Actualmente, CaLMa no soporta el tratamiento de datos en *Stream* previo al almacenamiento, es decir no existe un sistema que capture los datos en tiempo real y los redirija en un nuevo flujo que conecte con el ámbito del proyecto. Para suplir esta carencia, se ha optado por desarrollar la aplicación acorde a la siguiente ilustración:

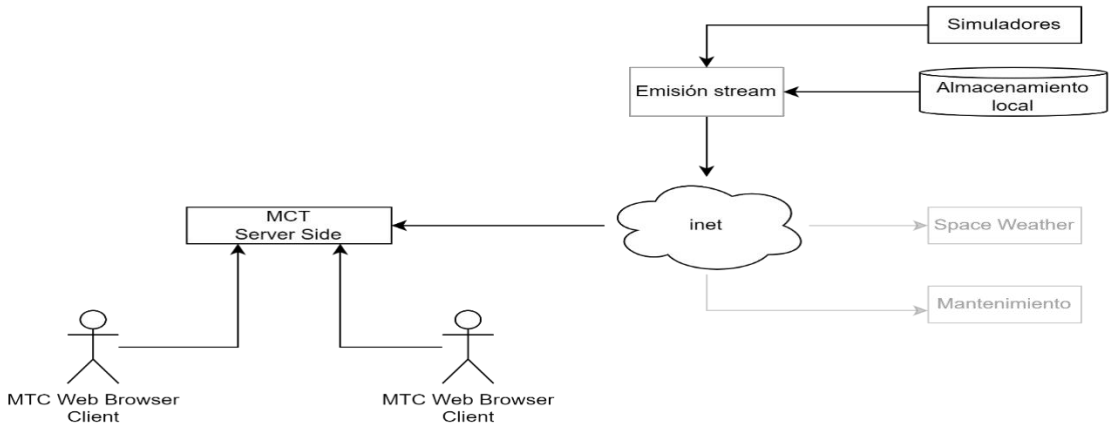


Ilustración 4 Diseño real del proyecto

Según el diseño real del proyecto de la Ilustración 3, el sistema desarrollado tendrá dos fuentes de datos:

- ➔ Tabla de una base de datos
- ➔ Servidor que genera datos simulados.

Es necesario hacer un inciso en dos puntos muy importantes:

- ➔ Se ha mencionado en varios apartados del documento que los datos se obtendrán y se visualizarán en tiempo real. Si bien es cierto, no se dispone de un sistema que capture los datos según se van generando y los derive por el flujo que continúa al bloque que conforma el ámbito del proyecto, pero esto no implica que el tratamiento no sea en tiempo real. La aplicación está preparada para procesar información según va llegando, es decir, tiene un canal abierto que está continuamente "escuchando" a la espera de reaccionar cada vez que aparece un nuevo dato para tratarlo y enviarlo al medio de visualización.

- ➔ El aspecto más importante es mostrar la integración del sistema de obtención de las telemetrías de CaLMa con un potente *framework* (que aún no he mencionado, pero del que hablaré más adelante) para facilitar la creación de paneles de mando, detectar fallos y anomalías de una manera más rápida, sencilla y efectiva

Una vez aclarado estos aspectos técnicos del diseño, vamos a centrarnos en la naturaleza de los datos del monitor de neutrones.

Tal como está diseñado actualmente el monitor de neutrones de CaLMa, éste recopila información a partir de 18 tubos contadores que detectan el número de neutrones que inciden en cada tubo en el intervalo de un minuto y 3 medidores meteorológicos que miden la presión atmosférica (medida en hectopascales), la temperatura (en grados Celcius) y la humedad relativa (relación entre la presión parcial del vapor de agua y la presión de vapor de equilibrio del agua a una temperatura dada).

Estos datos se almacenan en una tabla llamada `binTable` de la base de datos `nmdadb`. La tabla contiene en total 22 columnas. En la Ilustración 3 podemos ver una captura de datos de la tabla de la base de datos. A continuación, se proporciona una descripción detallada de la misma:

- La primera columna es `start_date_time`. Es un tipo de datos `dateTime` (formato `yyyy-mm-dd hh:mm:ss`) y es único para cada registro de cada columna, es decir, por

cada registro `start_date_time` aparecerá un registro para el resto de columnas. La actualización se realiza cada minuto.

- Las siguientes 18 columnas se corresponden a los tubos contadores de neutrones y son del tipo int. Siguen la nomenclatura `chXX` donde XX representa el rango de valores que va desde el 01 al 18. Cada columna almacena valores enteros correspondientes al número total de neutrones que impactan en cada tubo contador en el rango de un minuto. Actualmente, los `ch3, ch5, ch6, ch13, ch14, ch15` registran valor 0.

- La siguiente columna se corresponde con la temperatura medida en grados Celsius, por eso el identificador de la misma es una C y el tipo de datos es flotante. Actualmente registra valor 0

- A continuación, tenemos RH – humedad relativa. Es un dato del tipo float. Actualmente registra valor 0.

- La última columna registra valores del tipo float pertenecientes a la presión atmosférica medida en hectopascales.

start_date_time	ch03	ch06	ch05	ch13	ch14	ch17	C	RH	hPa
2019-03-18 17:40:00	725	839	751	60	98	95	10.30	44.90	995.30
2019-03-18 17:39:00	704	788	721	61	118	94	10.30	44.80	995.30
2019-03-18 17:38:00	705	761	692	56	92	82	10.30	44.80	995.30
2019-03-18 17:37:00	650	829	746	67	131	81	10.30	44.80	995.30
2019-03-18 17:36:00	714	801	701	73	115	82	10.30	44.80	995.20
2019-03-18 17:35:00	730	809	729	63	122	90	10.30	44.80	995.20
2019-03-18 17:34:00	677	786	677	68	116	83	10.30	44.80	995.20
2019-03-18 17:33:00	740	813	729	64	117	83	10.20	44.70	995.20
2019-03-18 17:32:00	713	772	699	59	107	76	10.20	44.70	995.20
2019-03-18 17:31:00	702	823	701	60	129	92	10.20	44.70	995.20
2019-03-18 17:30:00	727	852	700	70	133	96	10.20	44.70	995.20
2019-03-18 17:29:00	679	777	672	67	106	94	10.30	44.60	995.20
2019-03-18 17:28:00	745	795	755	72	106	82	10.30	44.70	995.20
2019-03-18 17:27:00	756	789	738	67	108	92	10.30	44.70	995.20
2019-03-18 17:26:00	697	780	709	66	122	90	10.30	44.70	995.10
2019-03-18 17:25:00	710	791	663	59	131	78	10.30	44.70	995.10
2019-03-18 17:24:00	748	791	735	55	106	78	10.30	44.70	995.10
2019-03-18 17:23:00	731	810	691	65	112	98	10.30	44.70	995.10
2019-03-18 17:22:00	654	730	784	70	116	74	10.30	44.80	995.10
2019-03-18 17:21:00	696	788	715	63	114	91	10.30	44.80	995.10

Ilustración 5: Datos de la binTable

En la descripción de los datos de la tabla `binTable`, se mencionó que algunos canales están registrando valor 0. Para mantener una concordancia entre el número de tubos contadores de CaLMa y las telemetrías a visualizar, se optó por utilizar el nombre de las columnas sin datos para almacenar datos simulados provenientes de un servidor incluido en la misma aplicación que los va generando mediante el uso de funciones matemáticas.

➔ En los próximos apartados, se entrará en detalle, a nivel de código, en la definición de los elementos y tipos de datos utilizados que representarán a los tubos contadores y medidores meteorológicos. En este punto temprano del documento, sólo se indicará

que, las variables que representan a los elementos telemétricos, han tenido que ser inicializadas con un valor entero, elegido de manera arbitraria.

- ➔ `Para los canales ch3 y ch5`: Se tomó como punto de partida un valor entero al que se le fue restando una cantidad fija del tipo flotante. Se empleó la función matemática `Math.max` para elegir el mayor entre un valor fijo y la resta indicada. El resultado es un valor decreciente que llega a 0. Esto es así porque necesito simular un tubo contador que presenta una anomalía y deja de funcionar.

- ➔ Para el resto de canales se utilizaron las mismas fórmulas: `this.state["CH13"] = this.state["CH13"] * 0.985 + Math.random() * 0.225 + Math.sin(Date.now())` . Para cada canal el valor será lo que tenga en ese momento multiplicado por una `constante_float_1` (en este caso es la misma para todos los canales) a la que se le suma un número aleatorio multiplicado por una `constante_float_2` (en este caso es distinta para cada canal) más el seno del resultado de una función de fecha.

En resumen, los datos simulados son del tipo float.

8 Diseño de la solución

El objetivo final de este proyecto es crear un **panel de control de misiones (en adelante PCM)** que permita la visualización de distintas telemetrías del Monitor de Neutrones CaLMa obtenidas en tiempo real. El panel de control residirá en un navegador web. Inicialmente se podría pensar que la arquitectura más adecuada es la del Cliente-Servidor por los siguientes motivos:

- Por un lado, existe un proveedor de datos telemétricos al que llamaremos **Servidor de datos**. En este caso será la base de datos que se actualiza cada minuto con la información obtenida de los tubos contadores y medidores meteorológicos.

- En el otro extremo existe un **Cliente** que consumirá dichos datos para publicarlos en un Panel de Control dentro de un navegador.

Como bien sabemos, en esta arquitectura, el cliente realiza peticiones al servidor para acceder y/o descargar datos. Se abre una conexión con cada petición o se mantiene un canal abierto por el que se intercambian mensajes. Cuando finaliza el intercambio, se cierra el canal de comunicación.

Sin embargo, ese no es el enfoque que se ha aplicado en el desarrollo del proyecto, ya que se ha optado por utilizar una arquitectura basada en eventos con un patrón del tipo Publisher-Subscriptor. Esto obedece a los siguientes motivos:

- En el sistema existen en realidad dos fuentes de datos (o ***Publisher***): Por un lado, tenemos las telemetrías obtenidos en *stream* provenientes del flujo que se dirige hacia la base de datos y por otro lado datos generados por un servidor local.

- A continuación, tenemos un módulo formado por un Cliente que es a la vez Servidor con dos funciones muy importantes:
    - Como “cliente” o suscriptor: mantendrá un canal abierto que estará a la escucha de la llegada de datos nuevos, es decir, reaccionará a cada evento recibido (lecturas telemétricas). Con la información obtenida elaborará los paneles de control.

    - Como “servidor” o *Publisher*: generará en tiempo real datos telemétricos simulados. Además, tomará las telemetrías que ha procesado como cliente y los datos generados como servidor y los publicará para que sean consumidos por un navegador web donde se visualizarán los paneles de control generados.

En resumen, el *software* se desarrolla en dos partes empleando la arquitectura basada en eventos con un modelo Publisher/Subscriptor. La tecnología empleada para el desarrollo es NodeJS, es decir estará programada en JavaScript. Y como cualquier otro proyecto NodeJS, la aplicación tendrá un *Front-end* y un *Back-End*.

- Desarrollo en el lado del servidor o *Publisher*: Es la parte *BACK* del proyecto. Basado en el uso de tecnologías API-REST y *WebSockets* para la obtención de telemetrías en tiempo real y tiempo diferido.

- Desarrollo en el lado del cliente/*suscriptor*: Parte *FRONT* del proyecto. Incluye la implementación de los *dashboard* que conforman el panel de control basados en el *framework* **OpenMCT** que permita visualizar los datos obtenidos del servidor mediante la implementación de unos *plugins* desarrollados en JavaScripts. OpenMCT es un ***framework*** que se integra fácilmente con cualquier fuente de telemetría. La integración con el origen de datos requiere el uso de JavaScript (en este caso **NodeJS**) para definir un nuevo proveedor de telemetrías que recupera los datos y los pone a disposición de la plataforma para su visualización mediante extensiones propias de OpenMCT. En este caso, se hará uso de las extensiones ya existentes y se desarrollarán nuevos plugins de acuerdo con las necesidades del proyecto.

La elección de esta tecnología y arquitectura obedece a la naturaleza del proyecto:

- Necesidad de crear un canal de comunicación bidireccional y `full-dúplex` sobre un único `socket TCP` entre el navegador y nuestro servidor web.

- Aprovechar la versatilidad de los navegadores combinándolo con métricas de tiempo real: Esto implica el uso de una arquitectura basada en eventos. Por un lado, tendremos nuestro **servidor o productor de eventos** encargado de interactuar directamente con los monitores de neutrones (en este caso con la BD de CaLMa y el servidor de datos simulados). Y por otro lado el **consumidor** de eventos que será nuestro panel de control de misiones basado en tecnología OpenMCT visualizado a través del navegador.

- Empleo de un modelo *Publisher-Subscriber*: Un modelo Publisher-Subscriber permite que una aplicación anuncie eventos de forma asincrónica a varios consumidores interesados, sin necesidad de emparejar los remitentes con los receptores. En las aplicaciones distribuidas y basadas en la nube, los componentes del sistema a menudo necesitan proporcionar información a otros componentes a medida que suceden los eventos. La mensajería asincrónica es una forma eficaz de desacoplar a los remitentes de los consumidores y evitar bloquear al remitente para que espere una respuesta. En este caso, el panel de control deberá mostrar en tiempo real todos los cambios registrados en el servidor. Para ello no es necesario que se realicen solicitudes para obtener información sobre un estado concreto, en lugar de esto, existirá un canal de eventos al cual se suscribe el sistema, que recibirá una notificación siempre que se genera un estado nuevo. Permitiendo así, obtener la información y reaccionar prácticamente en tiempo real.

8.1 ¿Qué es OpenMCT?

Hasta ahora se ha mencionado de manera redundante el uso del *framework* OpenMCT en este proyecto, pero ¿qué es exactamente OpenMCT?

OpenMCT ***(Mission Control Technologies)*** es un *framework* para el control de misiones de siguiente generación. Permite la visualización de datos en aplicaciones de escritorio y móviles. Fue desarrollado por *Ames Research Center de la NASA* en colaboración con *Jet Propulsion Laboratory*. Es utilizado por la ***NASA*** para el análisis de datos de misiones de naves espaciales, así como para la planificación y operación de sistemas de rover experimentales (robots enviados a misiones espaciales como el famoso Rover de Marte).

Una de las ventajas más grandes que ofrece es que es un *framework* generalizable y de código abierto, pudiendo utilizarse como base para crear nuevas aplicaciones para la planificación, operación y análisis de cualquier sistema que produzca datos de telemetría.

OpenMCT es también muy versátil y ofrece una gran facilidad para la escalabilidad de los sistemas de control de misiones que evolucionan con mucha rapidez. En la *NASA*, los requisitos para OpenMCT están siendo impulsados por la necesidad de admitir operaciones distribuidas, acceso a datos en cualquier lugar, visualización de datos para análisis de naves espaciales que abarque múltiples fuentes de datos y reconfiguración flexible para admitir múltiples misiones y casos de uso de operadores. Open MCT reúne muchas de las funciones de las operaciones de la misión, aliviando la necesidad de que los operadores cambien entre diferentes aplicaciones para ver todos los datos necesarios, con ver una sola pantalla tienen acceso y total control sobre la situación de los datos en cada momento.

8.1.1 Características de OpenMCT

- *Responsive Design* basado en la web: el *framework* facilita el diseño de paneles de control de misiones para la visualización de los datos desde cualquier lugar, ya sea en plataformas móviles o de escritorio, basta con tener acceso a un simple navegador. Esto simplifica mucho el soporte, ya que se elimina la necesidad de entregar actualizaciones a múltiples plataformas y compatibles con todos los S.O. existentes.

- Orientado a objetos, de fácil diseño y manipulación por parte del usuario desarrollador: Los datos y vistas se representan como simples objetos que se pueden componer de cualquier manera. Por ejemplo, podemos formar paneles de control que contiene gráficas, tablas y otras vistas. Podemos mezclar vistas para crear diseños de múltiples elementos de visualización que el usuario puede dimensionar y colocar: un *dashboard* que contenga una gráfica para la temperatura y presión, otro que mezcle la velocidad y el espacio recorrido, todo ello en un solo panel. Los datos se pueden mostrar con gráficas y también con imágenes. La forma de componer un panel solo exige operaciones de arrastrar y soltar, su manejo es muy directo e intuitivo.

- Arquitectura Extensible y flexible: OpenMCT surgió de la idea de crear un único *framework* de desarrollo utilizable en múltiples misiones. De ahí radica el uso de una arquitectura que enfatiza la reutilización, la modularidad y la extensibilidad. La funcionalidad central de Open MCT se puede personalizar con complementos para satisfacer las necesidades específicas de las misiones, en múltiples dominios. Esto se debe a que el *framework* proporciona una API pública y extensible, de fácil

integración con fuentes telemétricas y otras funcionalidades propias de cada aplicación. Aplicada a nuestro proyecto, da lugar a que, en un futuro, si se añaden más tubos contadores o se decide incluir más telemetrías de otra naturaleza, éstas se integren fácilmente en el diseño ya existente. Y si el proyecto CaLMa decide crear un nuevo sistema que recoja por ejemplo datos obtenidos de la estación MiniCaLMa, se pueda utilizar como base el proyecto aquí creado.

8.1.2 Proyectos OpenMCT

En la actualidad, OpenMCT es utilizado en múltiples proyectos dentro y fuera de la NASA y también por organizaciones no gubernamentales:

- **Proyecto Asteria - JET PROPULSION LABORATORY:** *Arcsecond Space Telescope Enabling Research in Astrophysics*. Misión científica oportunista para realizar mediciones astrofísicas utilizando un CubeSat.
- **Cold Atom Laboratory - JET PROPULSION LABORATORY**: es un instrumento experimental a bordo de la ISS, que se lanzó en 2018. Crea un entorno de microgravedad extremadamente frío para estudiar el comportamiento de los átomos en estas condiciones.

- **ICESat 2 - GODDARD SPACE FLIGHT CENTER:** es una misión satelital para medir la elevación de la capa de hielo y el espesor del hielo marino, así como la topografía terrestre, las características de la vegetación y las nubes. La misión ICESat-2 está diseñada para proporcionar los datos de elevación necesarios para determinar el balance de masa de la capa de hielo, así como la información sobre el dosel de la vegetación. Proporcionará mediciones topográficas de ciudades, lagos y embalses, océanos y superficies terrestres de todo el mundo, además de la cobertura polar específica. ICESat-2 también tiene la capacidad de detectar la topografía del fondo marino hasta 100 pies (30 m) por debajo de la superficie en áreas costeras de aguas claras.

- **JASON-3 – JET PROPULSION LABORATORY:** Es la cuarta misión de una serie de misiones satelitales entre Estados Unidos y Europa que miden la altura de la superficie del océano. la misión amplía la serie temporal de mediciones topográficas de la superficie del océano. Estas mediciones brindan a los científicos información crítica sobre los patrones de circulación en el océano y sobre los cambios globales y regionales en el nivel del mar y las implicaciones climáticas de un mundo que se calienta.

- **Jason-CS/Sentinel-6 - JET PROPULSION LABORATORY:** Formado por dos satélites idénticos encargados de realizar mediciones sobre el nivel del mar. estos satélites proporcionarán mediciones continuas del aumento del nivel del mar global, uno de los indicadores más importantes del cambio climático causado por el hombre. Los datos también respaldarán la oceanografía operativa a través de pronósticos mejorados de las corrientes oceánicas, así como de las condiciones del viento y las olas. Estos datos permitirán mejorar tanto el pronóstico a corto plazo para predicciones meteorológicas en el rango de dos a cuatro semanas (p. ej., predicciones de intensidad de huracanes) como el pronóstico a largo plazo para condiciones estacionales (p. ej., El Niño, La Niña).

- **Landsat 9 - GODDARD SPACE FLIGHT CENTER:** Landsat es el único sistema satelital de EE. UU. diseñado para observar repetidamente la superficie terrestre global a

una escala moderada que muestra cambios tanto naturales como inducidos por el hombre.

- **LightSail 2 - THE PLANETARY SOCIETY:** Es un proyecto satelital para demostrar la navegación solar controlada dentro de la órbita terrestre baja utilizando un *CubeSat.*

- **Mars 2020 - JET PROPULSION LABORATORY:** La misión Mars 2020 con su *rover Perseverance* es el primer paso de un viaje de ida y vuelta para devolver muestras de Marte a la Tierra para su posterior estudio. Su misión es buscar signos de posible vida microbiana en esos ambientes habitables, particularmente en rocas especiales que se conoce que preservan estos signos en el tiempo.

- **Mars Cube One – JOHNSON SPACE CENTER:** Es una misión para sobrevolar Marte en la que interactúan dos nano-naves espaciales, de formato 6U *CubeSat*. Fue lanzado el 5 de mayo de 2018 junto con la misión *InSight Mars Lander* de la NASA. El *Mars Cube One* tiene como misión proporcionar un enlace de comunicaciones a la Tierra para *InSight* durante la entrada, descenso y aterrizaje de misiones críticas cuando InSight no esté visible desde la Tierra. Se espera que *Mars Cube One* sea la primera nave espacial construida con formato *CubeSat* para operar más allá de la órbita terrestre. El módulo de aterrizaje *InSight* retransmitirá sus datos de telemetría poco después del aterrizaje. Por lo tanto, la prueba *MarCO* no es indispensable para la misión de *Insight*, pero demostrará el nuevo sistema de retransmisión y la tecnología para su uso futuro en misiones a otros cuerpos exteriores del Sistema Solar.

- **NIRVSS - AMES RESEARCH CENTER:** El subsistema de espectrómetro volátil de infrarrojo cercano es un instrumento diseñado por Ames de la NASA que forma parte del conjunto de instrumentos a bordo del *rover Resource Prospector (RP)*. El instrumento caracterizará la variabilidad de la superficie lunar y detectará volátiles en el suelo extraídos de la superficie por el subsistema de perforación del *rover*.

- **VIPER Lunar Rover - NASA ARC, NASA JSC, NASA KSC:** *Volatiles Investigating Polar Exploration Rover*, es un rover lunar desarrollado por la NASA. Se planea lanzarlo a la superficie de la Luna en 2023. El *rover* tendrá la tarea de estudiar los recursos lunares en áreas permanentemente sombreadas en la región del polo sur lunar, especialmente mediante el mapeo de la distribución y concentración del hielo lunar.

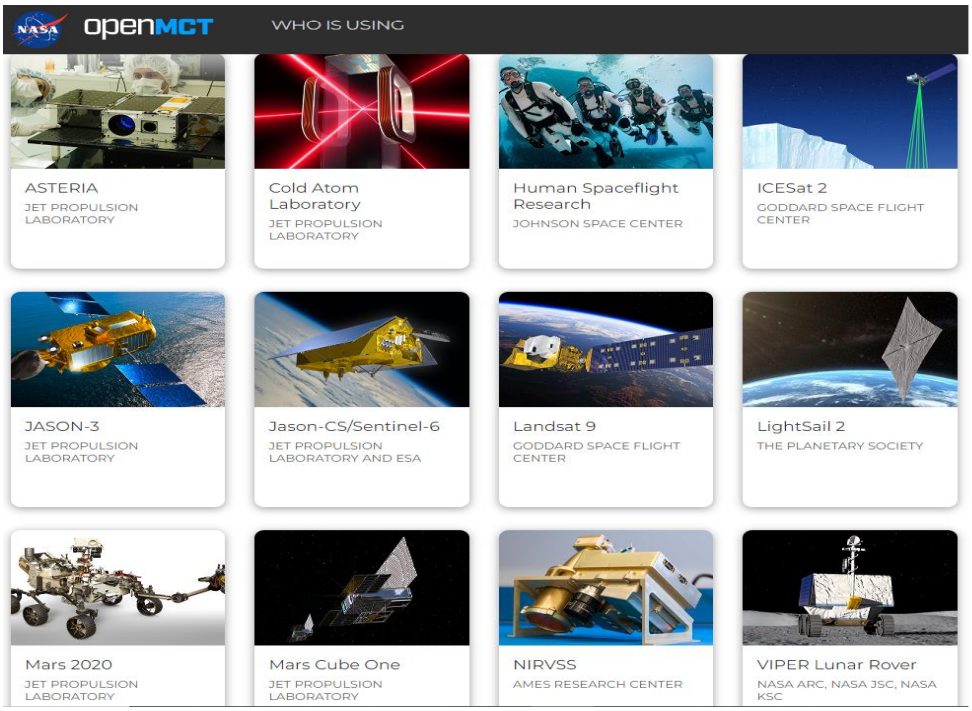


Ilustración 6: Proyectos que utilizan OpenMCT

Open MCT podría adaptarse a aplicaciones tan diversas como:

- Monitoreo de dispositivos IOT
- Drones
- *Cubesats*
- Robótica
- Globos de gran altura
- Monitorización de telemetrías provenientes de sistemas científicos y en general de cualquier fuente de datos.
- Monitoreo del rendimiento de la computadora y la red
- Visualización de datos empresariales
- Monitoreo de control de procesos

En resumen: OpenMCT es un *framework* utilizado en las misiones espaciales, pero debido a su naturaleza, se puede adaptar para emplearse en cualquier sistema que produzca telemetrías para mostrar en los paneles de control los datos de transmisión e históricos, imágenes, líneas de tiempo, procedimientos y otras visualizaciones, todo junto en un único lugar.

8.2 Preparación del entorno de trabajo

Antes de pasar al apartado donde se explicará cómo se realiza el acceso a las telemetrías de CaLMa y su posterior publicación, es necesario, desde mi punto de vista, explicar los pasos necesarios que llevé a cabo para disponer de un entorno de trabajo para el desarrollo del proyecto, haciendo especial hincapié en la instalación de OpenMCT como elemento principal del desarrollo. Como punto de partida, tenemos la configuración de una máquina virtual `Ubuntu 20.04` dentro de `Hyper-V`.

Para el uso de OpenMCT: Para empezar a trabajar con OpenMCT, es necesario instalar GIT y NodeJS. Una vez instaladas dichas aplicaciones, el siguiente paso será clonar el repositorio de

OpenMCT para disponer de los plugins básico para el desarrollo del Sistema de visualización de métricas (Panel de control de misiones).

A continuación, se explican de manera detallada los pasos seguidos para configurar el workspace:

- Instalación de GIT:
  - Actualización del índice local de paquetes:
    - `$ sudo apt update`
  - Instalación de GIT
    - `$sudo apt install git`

- Instalación de NodeJS:
  - `$sudo apt install nodejs`

- Creamos una carpeta para nuestro proyecto en donde clonaremos el proyecto base de OpenMCT
  - `$ git clone https://github.com/nada/openmct.git`

- Nos posicionamos en nuestro *workspace* e instalamos las dependencias de desarrollo:
  - `$npm install`

Nota: Instalación de MySQL WorkBench: Ésta es una herramienta para la gestión de bases de datos, al ser muy conocida y de uso extendido, no se especificarán los comandos empleados para su instalación.

8.3 Acceso a las telemetrías del instrumento Calma

La parte central del proyecto y por ende la más importante, es el acceso a las telemetrías de CaLMa y su posterior publicación. En el apartado sobre el diseño de la solución, en la página 25 de este documento, se explicaba el enfoque tomado para la resolución del problema.

Este apartado se centra en explicar en detalle la estructura del proyecto, qué elementos software forman parte del mismo y las tecnologías utilizadas en cada pieza. A continuación, tenemos una gráfica a modo de resumen:

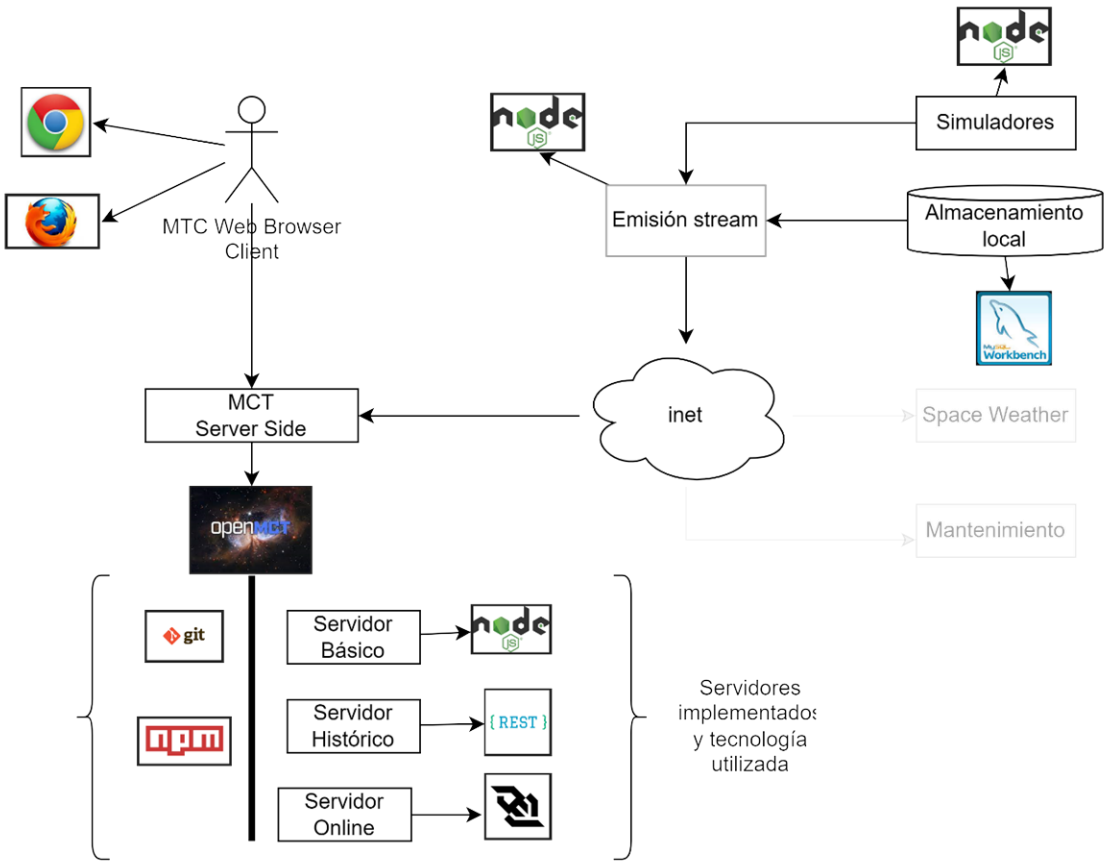


Ilustración 7: Diagrama del proyecto y tecnologías empleadas

Como se puede apreciar en la Ilustración 7 el software de este proyecto está desarrollado casi en su totalidad en NodeJS, incluida la base del propio *framework* OpenMCT. La elección de este entorno de tiempo de ejecución JavaScript obedece a que el Panel de Control de Misiones del monitor de Neutrones CaLMa (en adelante PCM), se visualiza en un navegador, por tanto, es necesario crear una aplicación web que sea dinámica y muy eficiente, que soporte la ejecución de muchos procesos de manera ágil y sin ningún tipo de bloqueo, en especial cuando las conexiones se multiplican. Por tanto, este proyecto tiene la misma estructura que presenta cualquier proyecto desarrollado en NodeJS.

Hay dos partes muy diferenciadas:

➔ ***FRONTEND***: Visualización de las telemetrías. Los archivos que forman parte del front-end:

- Carpeta **calma**: Contiene los archivos `calma.json`, `calmahistoricoplugin.js`, `calmaonlineplugin.js` y `calma-plugin.js`. Son referenciados desde el archivo index.html.

- ***Index.html:*** contiene los objetos y *plugins* básicos de un proyecto OpenMCT más los propios desarrollados para la aplicación.

- ***Lib:*** contiene los archivos auxiliares necesarios, en este caso sólo está el archivo `http.js` que se utiliza en el `index.html`.

➔ ***BACKEND***: Acceso a las telemetrías.

- Carpeta **server-calma:** contiene los siguientes archivos:
  - `Calma.js:` Archivo que contiene la función principal encargada de manejar los datos.
  - `Server.js:` es el archivo inicial de arranque del proyecto. En él se han instanciado todas las clases necesarias y las definidas para el manejo de las telemetrías.
  - `Server-basico.js:` Archivo utilizado para la interfaz del proyecto.
  - `Server-historico.js:` Archivo para el uso de los servicios Rest.
  - `Server-online.js:` Archivo para el uso de los *WebSockets*.
  - Carpeta `database:` Contiene los dos archivos necesarios para realizar la conexión a la base de datos:
    - `Connection.js:` Implementación del acceso a la base de datos, hace uso del dbconfig.js
    - `Dbconfig.js:` Configuración con el acceso al túnel (conexión SSH)

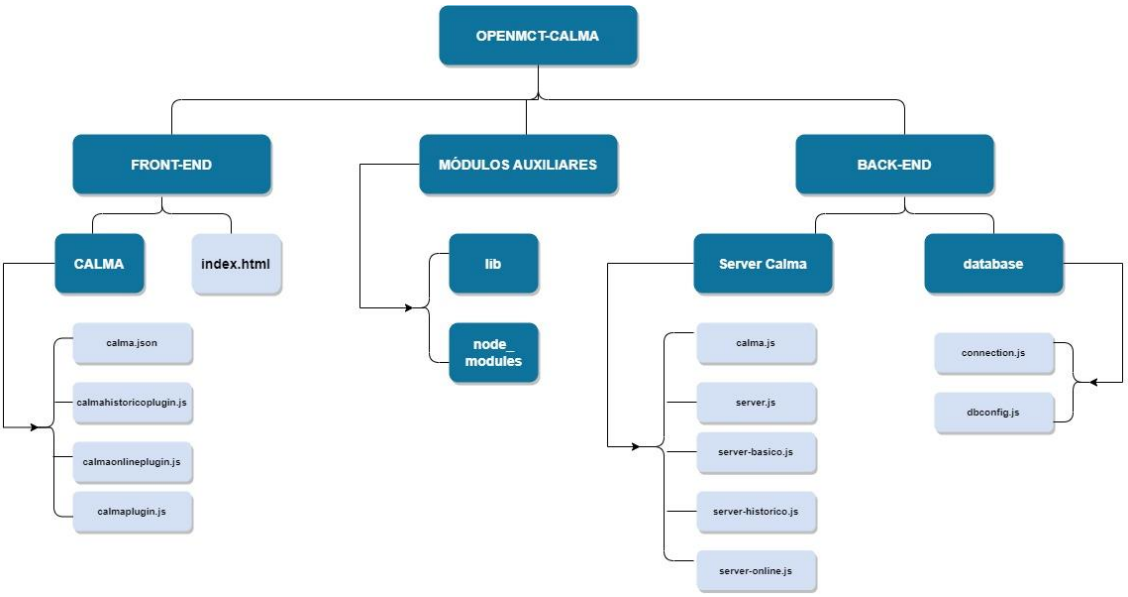


Ilustración 8: Elementos software de la estructura del proyecto

Siguiendo nuevamente el diagrama de la Ilustración 4, se aprecia que el Panel de Control de Misiones obtiene las telemetrías a partir de dos fuentes.

- Almacenamiento Local
- Servidor que genera datos simulados en tiempo real.

8.3.1 Acceso a las telemetrías del almacenamiento local.

Los **tubos contadores** y los **medidores atmosféricos** registran datos en el lapso de un minuto. Cada vez que esto sucede, genera un registro en la base de datos local. **Para el sistema desarrollado en este proyecto, esto es un evento**.

Según el diagrama de la Ilustración 6, la parte del *BACK-END* se encuentra en el directorio `openmct-calma/server-calma`. Es en este directorio en donde se crean y configuran los archivos necesarios para acceder a las telemetrías del almacenamiento local.

Para que el sistema pueda acceder a dichos registros es menester crear un canal de comunicación con la base de datos local. Es necesario recalcar que la conexión a la BD no se realiza de manera directa: es obligatorio abrir un `túnel SSH` para dicho propósito.

8.3.1.1 Configuración del túnel SSH.

- Instalar la herramienta SSH con el comando correspondiente para poder iniciar sesión de forma segura con el sistema remoto que aloja la bd.
- Obtener las claves públicas y privadas para realizar el intercambio de claves y poder establecer la comunicación de manera segura.
- Crear una nueva conexión desde `MySQL WorkBench` con los siguientes parámetros:
    - SSH Hostname: `hostname:puerto`
    - SSH Username: anasantos
    - SSH Password: Store in keychain(elegimos la clave rsa.pub)
    - MySQL Hostname: 127.0.0.1
    - MySQL Server Port: 3306
    - Password: Store in keychain
- El `hostname:puerto` es la dirección IP del servicio de la BD al cual se conectará la aplicación para acceder a las mediciones almacenadas en base de datos. Por temas de seguridad no se indican en el documento. Además, es un parámetro totalmente configurable, no estático.

8.3.1.2 Configuración del acceso al almacenamiento local.

Es necesario crear dos archivos: `dbconfig.js` y `connection.js`

- **Dbconfig.js:** En este fichero se definen las conexiones a la base de datos y al túnel SSH
    - Contiene una única función llamada `module.exports`
    - Dos constantes: localhost y tipo. La constante **tipo** se puede configurar con dos posibles valores correspondientes al tipo de conexión MySQL: **0 si es conexión normal y 1 si es conexión sobre SSH.**
    - A continuación, se definen dos estructuras de datos: `mySQLConfig` y `sshTunnelConfig`. La primera estructura tiene los datos necesarios para realizar una conexión directa, mientras que la segunda estructura almacena los datos de la conexión vía Túnel.

- **Connection.js**: Con las funciones y variables esenciales para crear la conexión:
    - Function `createDBConnection`: devuelve un dato del tipo `mysqlConnection`. Dependiendo del tipo, se recuperan los datos de la conexión normal o la conexión vía túnel.
    - Var `connection`: almacena el resultado de la llamada a la función `module.exports`.
    - Function: `Connection.invokeQuery`: función que recibe como parámetros `sqlQuery (consulta sql)`, data (objeto que representa al canal sobre el que se realizará la consulta).
    - `timeConverter`: toma el valor de la columna `start_date_time` y lo transforma en un `timestamp`.

8.3.1.3 Configuración del acceso a las telemetrías: archivo calma.js

Este archivo es común para el acceso a las telemetrías de CaLMa y a las telemetrías simuladas.

En primer lugar, realizamos la importación del módulo connection.js con la siguiente línea:

```
var connection = require('../server-calma/database/connection');
```

El siguiente paso es definir la función constructora **Calma** con sus propiedades y métodos correspondientes:

- Se definen los valores renderizados para cada uno de los elementos que representan los tubos contadores y los medidores meteorológicos.
- Se define un `array prototype` para los escuchadores (***listeners*, uno por cada tubo contador y medidores meteorológicos**) y un `object prototype` para **almacenar y recuperar el historial del estado de cada escuchador**.
- Mediante el método `setInterval` se ejecutan las dos siguientes funciones de manera reiterada, realizando un total de 10k. llamadas. En resumen: ejecuta la consulta a la base de datos, organiza las respuestas y monta el contenido del archivo JSON a enviar y lo pasa al servidor on line para que lo muestre por pantalla.
  - `updateState`: actualiza el estado de cada uno de los canales con la fecha actual, es el que ejecuta la sentencia sql y genera las telemetrías simuladas.
  - `generateTelemetry`: se encarga de notificar a cada uno de los escuchadores la llegada de un nuevo dato, la recupera y almacena en el histórico. Deja preparado los datos para enviarlos al servidor.
- Función `updateState`: Recibe como parámetros el estado y la fecha. Lo aspectos más importante de esta función son los siguientes:
  - Variable `sqlStatement`: almacena un *string* con la función SQL que se debe ejecutar contra la base de datos de CaLMa, empleando la conexión definida anteriormente.
  - `connection.invokeQuery`: recibe como parámetros la sentencia SQL definida en `sqlStatement` y una función que almacenará el resultado de dicha ejecución.
  - A continuación, se convierte el resultado obtenido en una cadena de texto JSON que posteriormente es parseada para obtener un par clave-valor, donde la clave es el CH que representa los tubos contadores o las variables definidas para los contadores meteorológicos y el valor es el inicial obtenido de la consulta.
  - A cada uno de los elementos definidos en el `state`, se le asigna su resultado correspondiente.
- Función `generateTelemetry`: A partir de la fecha va generando las telemetrías simuladas y obteniendo las de base de datos. Los aspectos más importantes:
  - Se encarga de "despertar" a los **escuchadores** que están pendientes para indicarles la llegada de un evento.
  - Para cada uno de los escuchadores se almacena en el histórico su estado correspondiente.

8.3.2 Acceso a las telemetrías generadas por el servidor.

Como parte del proyecto, se ha incluido la generación de datos simulados. En apartados anteriores se comentó que algunos tubos contadores no estaban recopilando datos, ni registrando información de sus lecturas en bd. Para mantener la coherencia en el orden de los tubos contadores, se optó por utilizar éstos para datos simulados.

El acceso a las telemetrías generadas por el servidor se configura en el mismo archivo **calma.js** descrito en el apartado anterior. La diferencia está en la obtención de los datos: para las telemetrías de bd, se ejecuta una sentencia sql que recupera los datos a mostrar y mediante funciones se organizan las respuestas, se genera el contenido del archivo JSON a enviar y se lo envía al servidor on-line para que lo muestre por pantalla. En cambio, para las telemetrías generadas por el servidor, se emplean funciones matemáticas:

```
this.state["CH03"] = Math.max(0,this.state["CH03"] -  0.5 );
this.state["CH05"] = Math.max(0,this.state["CH05"] -  0.25 );
this.state["CH06"] = Math.floor((Math.random() * (max - min + 1)) + min);
this.state["CH13"] = this.state["CH13"] * 0.985 + Math.random() * 0.225 +
Math.sin(Date.now());
this.state["CH14"] = this.state["CH14"] * 0.985 + Math.random() * 0.325 +
Math.sin(Date.now());
this.state["CH15"] = this.state["CH15"] * 0.985 + Math.random() * 0.425 +
Math.sin(Date.now());

this.state["C"] = 30 + Math.pow(Math.random(), 3);
this.state["RH"] = 50 + Math.pow(Math.random(), 3);
```

Ambos tipos de telemetría se obtienen y generan a la vez. Dando lugar a un único procesamiento, es decir, no existen funciones de tratamiento individual dependiendo del tipo. Las funciones `updateState` **y** `generateTelemetry` se ejecutan sobre el total del array de escuchadores.

8.3.3 Publicación de los datos: Implementación y configuración de los servidores.

Para que las telemetrías obtenidas puedan ser enviadas a la parte front-end del proyecto, se implementan 3 servidores:

- **Servidor OnLine:** Para el manejo de los servicios *WebSockets*.
- **Servidor Histórico:** para el manejo de los servicios *Rest*
- **Servidor Básico:** Para la visualización de la interfaz del proyecto OpenMCT-CaLMa.

Adicionalmente, es necesario implementar un fichero llamado `Server.js` encargado de instanciar las clases implicadas en el proyecto, así como desarrollar la lógica de conexión entre

las clases que generan/obtienen telemetrías (las descritas en el punto anterior) y las clases que permiten su publicación (servidores).

8.3.3.1 Publicación de los datos: `Archivo Server.js`

Es el archivo principal para el inicio de arranque del proyecto, en él se instancian todas las clases necesarias que han sido definidas para el manejo de las telemetrías, así cómo las clases de apoyo.

Esta clase está estructurada en tres bloques: definición de las variables que referencian a las clases, instanciación de clases y activación de la escucha de los servidores.

- Definición de las variables: Para este caso en concreto, se han definido 4 variables que se corresponden a las 4 clases principales implementadas: Calma, server-online, server-historico, server-basico. También se definen las variables necesarias para el enrutamiento:

```
var Calma = require('./calma');
var ServidorEnlinea = require('./server-online');
var ServidorHistorico = require('./server-historico');
var ServidorBasico = require('./server-basico');

var expressWs = require('express-ws');
var app = require('express')();
expressWs(app);
```

- A continuación, se instancian las clases declaradas para el manejo de los servidores a utilizar. En este caso se han empleado 3 servidores: el que el usuario usa para ver la **interfaz del proyecto OpenMCT-CaLMa (Servidor Básico)**, el que se utiliza para el **manejo de los servicios REST (Servidor Histórico)** y para el empleado para el **manejo de los web sockets (Servidor En Línea)**. Cuando el usuario da clic en cualquier nodo de la pantalla, se ejecuta una instancia del servidor en línea.
  En los servidores histórico y en línea, pasamos como parámetro la función encargada de manejar los datos (calma.js, que la instanciamos con el nombre calma).

```
var Calma = new Calma();
var servidorenlinea = new ServidorEnlinea(Calma);
var servidorhistorico = new ServidorHistorico(Calma);
var servidorbasico = new ServidorBasico();

app.use('/enlinea', servidorenlinea);
app.use('/historico', servidorhistorico);
app.use('/', servidorbasico);

var port = process.env.PORT || 8081;
```

- Y por último, se activan los servidores dejándolos a la escucha de eventos, utilizando para el ello el puerto 8081 definido previamente.

```
app.listen(port, function () {
    console.log('Servidor Basico esta en http://localhost:' + port);
    console.log('Calma Historico esta en http://localhost:' + port +
'/historico');
    console.log('Calma en linea esta en ws://localhost:' + port +
'/enlinea');
});
```

8.3.3.2 Publicación de los datos: Archivo server-basico.js

El servidor básico sólo define el direccionamiento. Sólo muestra la interfaz que se ofrece a los usuarios OpenMCT.

```
var express = require('express');
function ServerBasico() {
    var router = express.Router();

    router.use('/', express.static(__dirname + '/..'));

    return router;
}
module.exports = ServerBasico;
```

8.3.3.3 Publicación de los datos: Archivo server-historico.js

Cada vez que un usuario OpenMCT pulsa en cualquiera de los elementos definidos para visualizar la telemetría, se genera una petición del tipo `GET (REST)` al servidor histórico, subscribiendo el objeto indicado. Para ello se envían tres parámetros que son: la fecha inicial, la fecha final y el objeto:

```
var start = + req.query.start;
var end = + req.query.end;
var ids = req.params.pointId.split(',');
```

A continuación, el servidor histórico solicita los datos del caso:

```
return resp.concat(calma.history[id].filter(function (p)
```

Una vez procesados, los devuelve a la interfaz del usuario con estatus 200, (ok). El servidor toma los datos y los muestra, todo esto lo hacen las clases propias de la interfaz OPENMCT:

```
console.log("------------response---------------");
console.log(response);
res.status(200).json(response).end();
```

8.3.3.4 Publicación de los datos: Archivo server-online.js

Cada vez que se produce un cambio en el elemento de visualización de las telemetrías, es decir, cuando el usuario deja de ver las telemetrías del `CH3` y pasa al `CH6`, por ejemplo, el sistema **libera la suscripción** de `CH03` y hace la suscripción de `CH06`, todo esto mediante ejecución de los servicios `WebSockets`.

```
----------notifySubscribers-------------
{"timestamp":1656972672010,"value":77,"id":"CH03"}
----------notifySubscribers-------------
{"timestamp":1656972673017,"value":77,"id":"CH03"}
----------unsubscribe-------------
CH03
----------subscribe-------------
CH06
------------start---------------
1656971773905
------------end---------------
1656972673905
----------notifySubscribers-------------
{"timestamp":1656972676033,"value":30,"id":"CH06"}
----------notifySubscribers-------------
{"timestamp":1656972677035,"value":30,"id":"CH06"}
----------notifySubscribers-------------
{"timestamp":1656972678041,"value":30,"id":"CH06"}
----------notifySubscribers-------------
{"timestamp":1656972679049,"value":30,"id":"CH06"}
----------notifySubscribers-------------
{"timestamp":1656972680060,"value":30,"id":"CH06"}
----------notifySubscribers-------------
{"timestamp":1656972681072,"value":30,"id":"CH06"}
----------notifySubscribers-------------
{"timestamp":1656972682082,"value":30,"id":"CH06"}
----------notifySubscribers-------------
{"timestamp":1656972683089,"value":30,"id":"CH06"}
----------notifySubscribers-------------
{"timestamp":1656972684095,"value":30,"id":"CH06"}
```

Ilustración 9: Funcionamiento del server online

A nivel de código, los aspectos más importantes a destacar son los siguientes:

```
var unlisten = calma.listen(notifySubscribers);
var subscribed = {}; // Active subscriptions for this connection
var handlers = {..}// Handlers for specific requests
```

- **Var subscribed:** Array que almacena las suscripciones activas para la conexión actual.
- **Var handlers:** Manejadores con dos solicitudes específicas (dos funciones):
  - **Función** `subscribe`: recibe el identificador el escuchador activo. Lo añade al array **subscribed**. Es decir, realiza el cambio de escuchador, activando el nuevo indicado por el usuario.
  - **Función** `unsubscribe`: Elimina del array de escuchadores, el escuchador que estaba previamente activo.
- **Función notifySubscribers:** Recibe la telemetría a mostrar, la transforma en un objeto del tipo JSON y la envía. Por otro lado, pinta en consola el valor telemétrico recibido, junto con el id del escuchador.
- El canal de conexión queda abierto, a la escucha de llegada de datos. Una vez cerrada la conexión, se dejan de enviar actualizaciones de las telemetrías.

En resumen:

- Mediante el método **`prototype`** se agregan propiedades y métodos a una clase, en este caso se agregan tres métodos y al final exporta la nueva clase.
- Con el método **`generatetelemetry`** se construye los datos que se envían al servidor OpenMCT usando el método notify.
- En la definición del método **`notify`**, se barren todos los nodos que están escuchando y se envían los datos de acuerdo parámetro seleccionado. Es decir, si un usuario selecciona un nodo en concreto, digamos el `CH04`, el servidor barre todos los nodos, **pero sólo envía los datos del nodo seleccionado**.

A nivel de funcionamiento: una vez iniciada la aplicación, los tres servidores descritos anteriormente, quedarán a la escucha de la llegada de datos y de las peticiones del usuario provenientes del navegador. Esto se hace de manera automática mediante las librerías y clases de OPENMCT.

Cuando un usuario selecciona uno de los elementos que desea visualizar, la acción genera una petición `GET (REST)` al servidor histórico suscribiendo el objeto indicado por el usuario. Se recuperan y se envían tres parámetros: fecha inicial, fecha final y el objeto. El servidor histórico solicita los datos del caso indicado mediante la siguiente línea:

```
return resp.concat(calma.history[id].filter(function (p)
```

Y devuelve a la interfaz del usuario los datos del caso con estatus 200 (OK):

```
res.status(200).json(response).end();
```

El servidor básico toma los datos y los muestra, haciendo uso de las clases propias de la interfaz OpenMCT.

Los datos se obtienen a través de la funcionalidad descrita en el archivo `calma.js`, en el cual, mediante un `loop`, se ejecuta la consulta `SQL` obteniendo y generando las telemetrías.

Este `loop` finaliza al llegar al límite definido o cuando el usuario solicita otro nodo u objeto.

En el siguiente ejemplo, el usuario hace la selección `CH06`, previamente estaba activo el `CH03`:

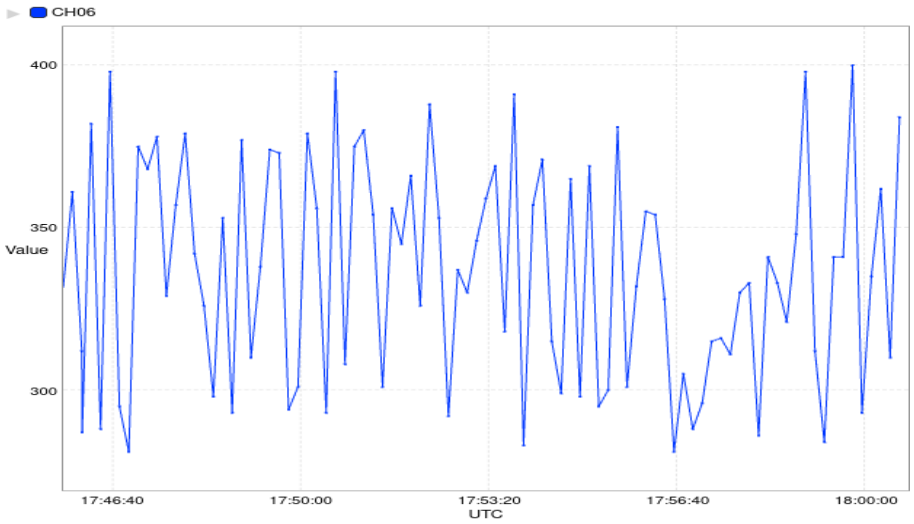


Ilustración 10: Suscripción de elementos

El sistema libera la suscripción de CH03 y hace la suscripción de CH06. En general el sistema hace lo mismo para cada nodo solicitado.

8.4 Desarrollo de un plugin de MTC compatible con Calma

En el apartado 8.2, titulado Preparación del entorno de trabajo, se detallaron las especificaciones necesarias para preparar el entorno de trabajo. En este apartado del documento se pretende explicar cómo se ha llevado a cabo el desarrollo de la parte Front-ent de la aplicación mediante la implementación de plugins utilizando Node-JS para ampliar la funcionalidad de OpenMCT y adaptarlo a las necesidades del proyecto.

La versatilidad y escalabilidad en la implementación de aplicaciones utilizando OpenMCT radica en el uso de los plugins. Los plugins son paquetes de componentes de software utilizados para ampliar OpenMCT mediante la definición de nuevas funcionalidades como pueden ser: fuentes de telemetrías, proveedores de objetos, nuevas visualizaciones de telemetrías, etc.

Todo OpenMCT está compuesto por plugins. Para crear un proyecto utilizando este framework existen dos opciones:

1. Realizar una implementación desde cero: sólo hace falta instalar NodeJS (npm) y git.
2. Instalar una instancia de OpenMCT.

Para este proyecto se ha optado por la segunda opción. Para ello se han realizado las siguientes acciones:

1. Crear un directorio para el proyecto. En este caso se ha llamado openmct-calma.
2. Seguir los pasos indicados en el apartado **Preparación del entorno de trabajo.**
3. Compilar MCT con los siguientes comandos:
    a. `git clone https://github.com/nasa/openmct.git`
    b. `cd openmct-calma`
    c. `npm install`

Mediante estos comandos se obtiene una versión reducida de OpenMCT desde el repositorio oficial existente en GitHub y que servirá de base para la implementación de la aplicación de este proyecto. Cuando se compila el proyecto se genera una serie de archivos que se colocan en una carpeta llamada dist del directorio OpenMCT. El contenido de esta carpeta incluirá un archivo javascript minimizado llamado **openmct.js**, así como activos como html, css, index.html e imágenes necesarias para la interfaz de usuario. Esta versión reducida de OpenMCT incluye también una serie de plugins que proporcionan una interfaz de usuario, persistencia y otras configuraciones predeterminadas que son necesarias para poder arrancar una aplicación y aprender el funcionamiento de OpenMCT. Cualquiera de estos complementos, pueden ser reemplazados por nuevos o reutilizarlos, personalizarlos o añadir otros, todo depende de las necesidades del proyecto.

Dicho contenido puede ser copiado al directorio de la aplicación o compilado directamente en el directorio openmct-calma.

A continuación, se detallan los archivos de la parte del front-end así como la implementación y funcionalidad de los plugins implementados en este proyecto.

8.4.1 Archivo `calma.json`.

Este archive se utiliza para la definición de los `domain objects` que representarán a los elementos telemétricos de este proyecto. A continuación, se explicará que son los `domains objects` y los `atributos del objeto`, que son terminologías propias de OpenMCT.

8.4.1.1 Domain Objects.

Los `Domain Objects` son las **entidades básicas que representan el conocimiento del dominio en OpenMCT**. Un objeto puede ser un sensor de temperatura, una gráfica superpuesta que compara dos mediciones provenientes de los sensores atmosféricos, en el caso de este proyecto, los objetos del dominio son "canales" que detectan el número de neutrones que han impactado en el tubo contador, junto con los medidores meteorológicos. Es decir, los objetos que aparecen cuando se inicia la aplicación.

A nivel de código, un `domain object` es simplemente un **objeto codificado en javascript**, al que se le han añadido algunos **atributos**.

8.4.1.1.1 Atributos del objeto.

Un objeto tiene dos atributos: `Identifier (identificador) y type (tipo).`

- **Identifier**: es una clave compuesta por un `namespace` y una `key` e identifica al objeto de manera única y universal. **La clave `(key)` debe ser única dentro del `namespace`.**
- **Type**: Todos los objetos en OpenMCT tienen un tipo que proporcionan una abstracción para agrupar, visualizar e interpretar datos. Se pueden definir nuevos tipos o utilizar los tipos integrados que proporciona OpenMCT.

En la siguiente ilustración se aprecia la estructura de la definición de un objeto de dominio (domain object):

```
{
    identifier: {
        namespace: ""
        key: "mine"
    }
    name:"My Items",
    type:"folder",
    location:"ROOT",
    composition: []
}
```

Ilustración 11: Estructura de un domain object

Los objetos del dominio y sus atributos se definen en el archivo **`calma.json`**. Para este proyecto, en total se han definido un objeto con 21 atributos: 18 de ellos se corresponden con los tubos contadores de neutrones, nombrados como CH1 – CH18. Los otros tres identifican a los medidores meteorológicos: hPa, RH y C.

A continuación, se muestra un ejemplo de un domain object:

```
{
    "name": "Demo Calma",
    "key": "calma",
    "measurements": [
        {
            "name": "CH01",
            "key": "CH01",
            "values": [
                {
                    "key": "value",
                    "name": "Value",
                    "units": "bytes",
                    "format": "integer",
                    "hints": {
                        "range": 1
                    }
                },
                {
                    "key": "utc",
                    "source": "timestamp",
                    "name": "Timestamp",
                    "format": "utc",
                    "hints": {
                        "domain": 1
                    }
                }
            ]
        },
        { … },
      ]
}
```

Tal como se aprecia en el código anterior, en el proyecto tenemos un objeto de dominio llamado Demo Calma, su clave es **calma**. Los atributos se definen dentro de `measurements` (mediciones). Todos los atributos tienen la misma estructura: un `namespace (name)` que se corresponde con el nombre del canal que representa al tubo contador y una `key (clave)`, en este caso como nombre de la clave se ha utilizado el mismo que el `namespace`.

8.4.2 Archivo index.html

El archivo `index.html` contiene los `plugins` base para el funcionamiento básico de cualquier aplicación, si se desea ampliar la funcionalidad de OpenMCT para adaptarlo a los requerimientos del proyecto, es necesario incluirlos en este archivo.

Para instalar nuevos `plugins` en OpenMCT, sólo es necesario ejecutar el comando `openmct.install`, para ello se debe proporcionar una función de instalación de `plugins`. Esta función se invocará al iniciar la aplicación con un parámetro: el objeto API de OpenMCT.

Inicialmente, el archivo `index.html` contiene sólo los siguientes `plugins`:

- **LocalStorage**: Almacenamiento local, proporciona persistencia de objetos creados por el usuario en el almacenamiento local del navegador

- **MyItems**: Define la carpeta de nivel superior denominada "`My Items`" para almacenar elementos creados por el usuario

- **UTCTimeSystem**: Proporciona un sistema de tiempo predeterminado basado en el tiempo universal coordinado (UTC)

- **Espresso**: Proporciona un tema oscuro para la interfaz Open MCT.

Para el desarrollo de este proyecto, se han desarrollado los siguientes plugins:

- DictionaryCalmaPlugin
- CalmaOnlinePlugin
- CalmaHistoricoPlugin

En el siguiente fragmento de código se muestra el contenido del archivo index.html de este proyecto:

```
<!DOCTYPE html>
<html>
<head>
    <title>Openmct Calma</title>
    <script src="node_modules/openmct/dist/openmct.js"></script>
    <script src="lib/http.js"></script>
    <script src="calma/calma-plugin.js"></script>
    <script src="calma/calmahistoricoplugin.js"></script>
    <script src="calma/calmaonlineplugin.js"></script>
</head>
<body>
    <script>
        openmct.setAssetPath('node_modules/openmct/dist');
        openmct.install(openmct.plugins.LocalStorage());
        openmct.install(openmct.plugins.MyItems());
        openmct.install(openmct.plugins.UTCTimeSystem());
        openmct.time.clock('local', {start: -15 * 60 * 1000, end: 0});
        openmct.time.timeSystem('utc');
        openmct.install(openmct.plugins.Espresso());



        openmct.install(DictionaryCalmaPlugin());
        openmct.install(CalmaOnlinePlugin());
        openmct.install(CalmaHistoricoPlugin());
        openmct.start();
    </script>
</body>
</html>
```

8.4.3 Implementación de los plugins.

Para cumplir con la funcionalidad descrita en este proyecto, se han desarrollado **3 plugins** mencionados en el punto anterior.

En este apartado del documento, se dará una descripción detallada de los pasos que se llevaron a cabo para implementar un `plugin`.

8.4.3.1 Paso 1: Definición de un nuevo plugin.

En Open MCT todo se representa como un objeto de dominio, esto incluye fuentes, puntos y vistas para visualizar la telemetría. Se puede acceder a los objetos de dominio desde el **árbol de objetos**.

Para poblar dicho árbol de objetos, es necesario crear un nuevo complemento (`plugin`) que se añadirán al árbol de objetos. En este proyecto se ha creado un archivo llamado **`calma-plugin`** en el que se ha incluido todo el código del nuevo complemento.

Como primer paso, se crea un complemento muy básico que registra un mensaje en el log indicando que se ha instalado correctamente. Cuando se ejecuta el install, se invoca a la función del API de OpenMCT, pasando la propia definición como parámetro:

```
var DictionaryCalmaPlugin = function (openmct) {
    return function install(openmct) {
        console.log("I've been installed");
    }
}
```

A continuación, se actualiza el archivo index.html añadiendo el nombre del plugin creado para su instalación:

```
openmct.install(DictionaryCalmaPlugin());
```

En resumen, un complemento de Open MCT es muy simple: es una función de inicialización que recibe la API de Open MCT como único argumento. A continuación, utiliza la API proporcionada para ampliar Open MCT. En general, se recomienda que los complementos devuelvan una función de inicialización para que puedan recibir la configuración.

8.4.3.2 Paso 2: Creación de un nuevo nodo raíz.

Para poder acceder a los objetos del proyecto Demo-Calma desde el árbol, primero hay que definir un nuevo objeto raíz mediante el API de OpenMCT. Para ello se utiliza la función addRoot que acepta como parámetro un objeto de dominio JavaScript (namespace/key).

```
var DictionaryCalmaPlugin = function (openmct) {
    return function install(openmct) {
        openmct.objects.addRoot({
            namespace: 'calma.taxonomy',
            key: 'calma'
        });
    }
}
```

8.4.3.3 Paso 3: Proporcionar objetos.

Una vez creada la raíz del árbol, el siguiente paso es poblarlo de objetos. Esto se consigue mediante la definición de un proveedor de objetos (**`object provider`**) que recibe un identificador de objeto y devuelve una promesa que se resuelve en un objeto para el identificador dado.

En este proyecto se ha creado y registrado un proveedor de objetos que dependiendo del tipo de identificador devuelve el objeto raíz o un objeto hijo del tipo `telemetry` que contiene las mediciones (`measurement`) definido en el archivo `json`.

Tipos para los objetos definidos: para los objetos se han empleado los siguientes tipos:

- El objeto raíz usa el tipo incorporado `folder`.
- Para los objetos que representan las fuentes de telemetrías del proyecto (`measurement`) se les asigna un nuevo tipo llamado **`telemetry`**

El siguiente fragmento de código se corresponde a la definición de proveedores de objetos.

```
var objectProvider = {
    get: function (identifier) {
        return getDictionaryCalma().then(function (dictionary) {
            if (identifier.key === 'calma') {
                return {
                    identifier: identifier,
                    name: dictionary.name,
                    type: 'folder',
                    location: 'ROOT'
                };
            } else {
                var measurement = dictionary.measurements.filter(function
(m) {
                    return m.key === identifier.key;
                })[0];
                return {
                    identifier: identifier,
                    name: measurement.name,
                    type: 'calma.telemetry',
                    telemetry: {
                        values: measurement.values
                    },
                    location: 'calma.taxonomy:calma'
                };
            }
        });
    }
};
```

8.4.3.4 Paso 4: Poblar el árbol.

Una vez que se ha definido la raíz y los objetos del árbol, el siguiente paso consiste en proporcionar una estructura al árbol mediante la definición de relaciones entre los distintos objetos definidos en Paso 3: Proporcionar objetos. Esto se consigue mediante la implementación de un **`Proveedor de Composición`**.

Un proveedor de composición acepta un objeto de dominio y proporciona identificadores para los hijos de ese objeto. En este proyecto, el proveedor de composición devuelve los identificadores de los tubos contadores (`CH01 - CH18`) y los identificadores de los 3 medidores meteorológicos. Esto se construye a partir del archivo de **diccionario de telemetrías** definido previamente.

A continuación, se muestra el código para el proveedor de composición:

```
var compositionProvider = {
    appliesTo: function (domainObject) {
        return domainObject.identifier.namespace === 'calma.taxonomy' &&
               domainObject.type === 'folder';
    },
    load: function (domainObject) {
        return getDictionaryCalma()
            .then(function (dictionary) {
                return dictionary.measurements.map(function (m) {
                    return {
                        namespace: 'calma.taxonomy',
                        key: m.key
                    };
                });
            });
    }
};
```

Si recargamos la página, se mostrará el árbol de objetos de nuestra aplicación totalmente poblado:

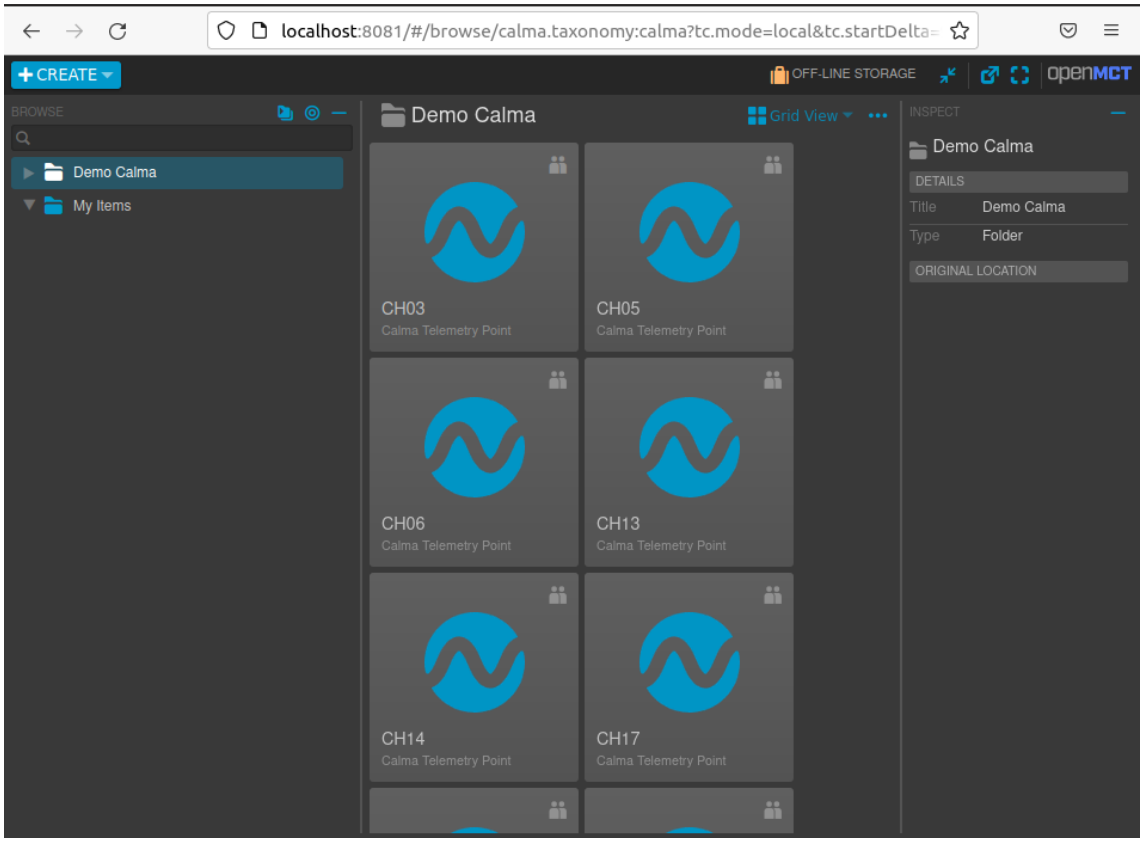


Ilustración 12:Árbol de la aplicación con los nodos definidos

8.5 Publicación de datos en streaming.

Para el desarrollo de este proyecto, se ha partido de la premisa de que existen dos fuentes de datos telemétricos:

- Almacén de datos local que se actualiza cada minuto con el número total de neutrones que impactan en los tubos contadores, más los registros proporcionadas por los medidores meteorológicos.

- Telemetrías generadas por un servidor que forma parte de la arquitectura de este proyecto.

Dos de los motivos por los que se eligió OpenMCT como `framework` de desarrollo, es porque:

1. admite la recepción de telemetrías solicitando datos de un almacén de telemetrías
2. permite la suscripción a actualizaciones de telemetrías en tiempo real.

Estas dos características se adaptan perfectamente a las necesidades del proyecto.

En este apartado del documento, se intentará explicar cómo se publican los datos en `streaming` mediante la implementación de dos `plugins`.

8.5.1 Publicación de datos en streaming: Publicación de Telemetrías históricas.

Para la publicación de telemetrías históricas, se debe definir un nuevo plugin dentro de un archivo llamado `calmahistoricoplugin.js`.

El objetivo principal del código desarrollado en dicho archivo, es el de **definir** y **registrar** un **adaptador de telemetrías** para solicitar datos telemétricos del histórico desde el propio servidor. Es decir, cada vez que el usuario-OpenMCT solicita los datos de otro nodo distinto al actual, el servidor recuperará las telemetrías generadas/obtenidas desde el primer momento de arranque de la aplicación, a continuación, las publicará en el navegador y habilitará la subscripción del nodo para que el otro servidor pase a estado de espera y cada vez que se registra una nueva telemetría, este reaccione enviando el dato procesado al navegador para ser visualizado en la gráfica.

A nivel de código la implementación es la siguiente:

```
/**
 * Basic historical telemetry plugin.
 */

function CalmaHistoricoPlugin() {
    return function install (openmct) {
        var provider = {
            supportsRequest: function (domainObject) {
                return domainObject.type === 'calma.telemetry';
            },
            request: function (domainObject, options) {
                var url = '/historico/' +
                    domainObject.identifier.key +
                    '?start=' + options.start +
                    '&end=' + options.end;

                return http.get(url)
                    .then(function (resp) {
                        return resp.data;
                    });
            }
        };

        openmct.telemetry.addProvider(provider);
    }
}
```

En el adaptador de telemetrías se han definido dos funciones principales:

- **supportRequest:** Esta función sirve para indicar que este adaptador admite la solicitud de telemetrías desde un almacén de datos telemétricos.

- **request:** Esta función se encargará de recuperar los datos de telemetría y los devolverá a la aplicación OpenMCT para su visualización. La función **`request`** también acepta algunas opciones, en el ámbito de este proyecto se ha optado por incluir la especificación de una fecha de inicio y una fecha de finalización.

Una vez implementado este plugin, es necesario incluirlo en el archivo index.html para poder ser utilizado por la aplicación mediante la siguiente línea de código:

```
openmct.install(CalmaHistoricoPlugin());
```

Si se actualiza la página de la aplicación (el navegador), se podrá visualizar las telemetrías de los objetos del árbol definidos en puntos anteriores, que representan las fuentes de datos telemétricos.

Una vez que el usuario empieza a navegar entre los distintos elementos de la aplicación, se empezarán a mostrar gráficas de las telemetrías generadas desde que el servidor comenzó a ejecutarse.

En la siguiente figura se muestra un ejemplo de telemetría:

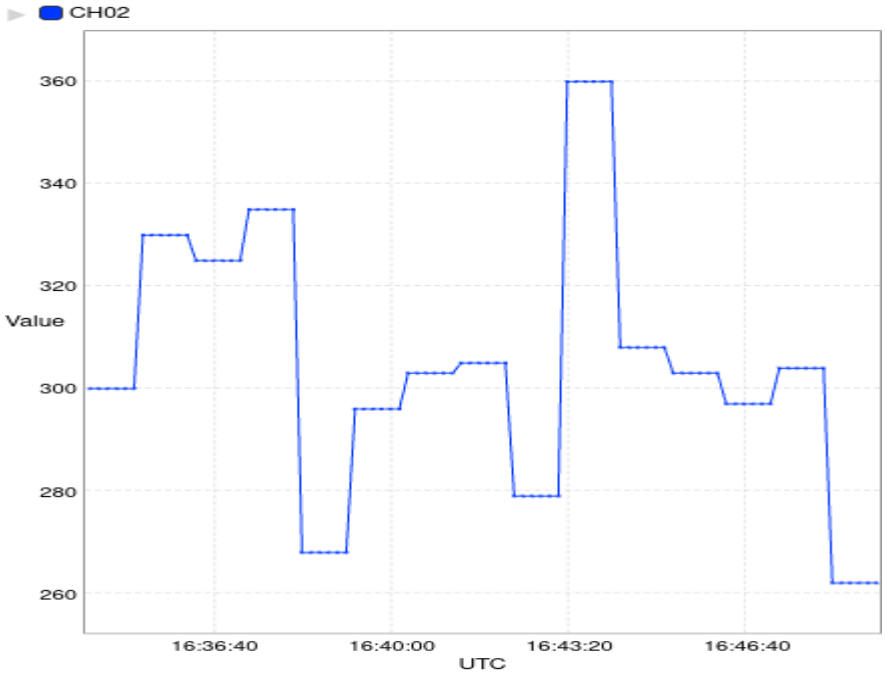


Ilustración 13: Ejemplo de telemetría

8.5.2 Publicación de datos en streaming: Publicación de Telemetrías de tiempo real.

Para la publicación de telemetrías de tiempo real es necesario implementar un nuevo `plugin` en el que se define **un adaptador de telemetrías** que permite a OpenMCT suscribirse al servidor que proporciona telemetrías simuladas y que son ofrecidas a medida que éstas son generadas.

La lógica de este adaptador se implementa en el archivo **`calmaonlineplugin.js`**

El proceso de definición de un adaptador de telemetría para suscribirse a telemetría en tiempo real es similar al adaptador de telemetría histórico definido anteriormente, excepto que se debe definir dos funciones principales:

- Función **supportSubscribe** para indicar que este adaptador proporciona suscripciones de telemetría. Recibe como dato de entrada un `domainObject` y devuelve un tipo.

- Función **subscribe** para suscribirse a actualizaciones. Recibe como argumentos de entrada, un objeto de dominio (`domainObject`) elegido por el usuario de la aplicación y una función de `callback`. Esta función de `callback` se invocará con los datos de telemetría a medida que estén disponibles.

La implementación del plugin queda de la siguiente manera:

```
/**
 * Basic Realtime telemetry plugin using websockets.
 */
function CalmaOnlinePlugin() {
    return function (openmct) {
        var socket = new WebSocket(location.origin.replace(/^http/, 'ws')
+ '/enlinea/');
        var listener = {};

        socket.onmessage = function (event) {
            point = JSON.parse(event.data);
            if (listener[point.id]) {
                listener[point.id](point);
            }
        };

        var provider = {
            supportsSubscribe: function (domainObject) {
                return domainObject.type === 'calma.telemetry';
            },
            subscribe: function (domainObject, callback) {
                listener[domainObject.identifier.key] = callback;
                socket.send('subscribe ' + domainObject.identifier.key);
                return function unsubscribe() {
                    delete listener[domainObject.identifier.key];
                    socket.send('unsubscribe ' +
domainObject.identifier.key);
                };
            }
        };

        openmct.telemetry.addProvider(provider);
    }
}
```

Otro de los puntos importantes a recalcar en esta parte, es el uso de un sistema de mensajería simple para suscribirse a actualizaciones de telemetría a través de un `websocket`.

Una vez que se ha definido el plugin para publicar telemetrías en tiempo real, se debe incluir, al igual que el resto de plugins, en el archivo **index.html.**

```
openmct.install(CalmaOnlinePlugin());
```

En este punto, si se actualiza la página y el usuario selecciona cualquiera de los objetos del árbol de telemetrías, se mostrará un gráfico de datos telemétricos que se actualiza de manera regular a medida que van llegando los datos.

La siguiente ilustración sirve para resumir el funcionamiento y la interacción de los 3 servidores entre sí para dar respuesta a la aplicación:

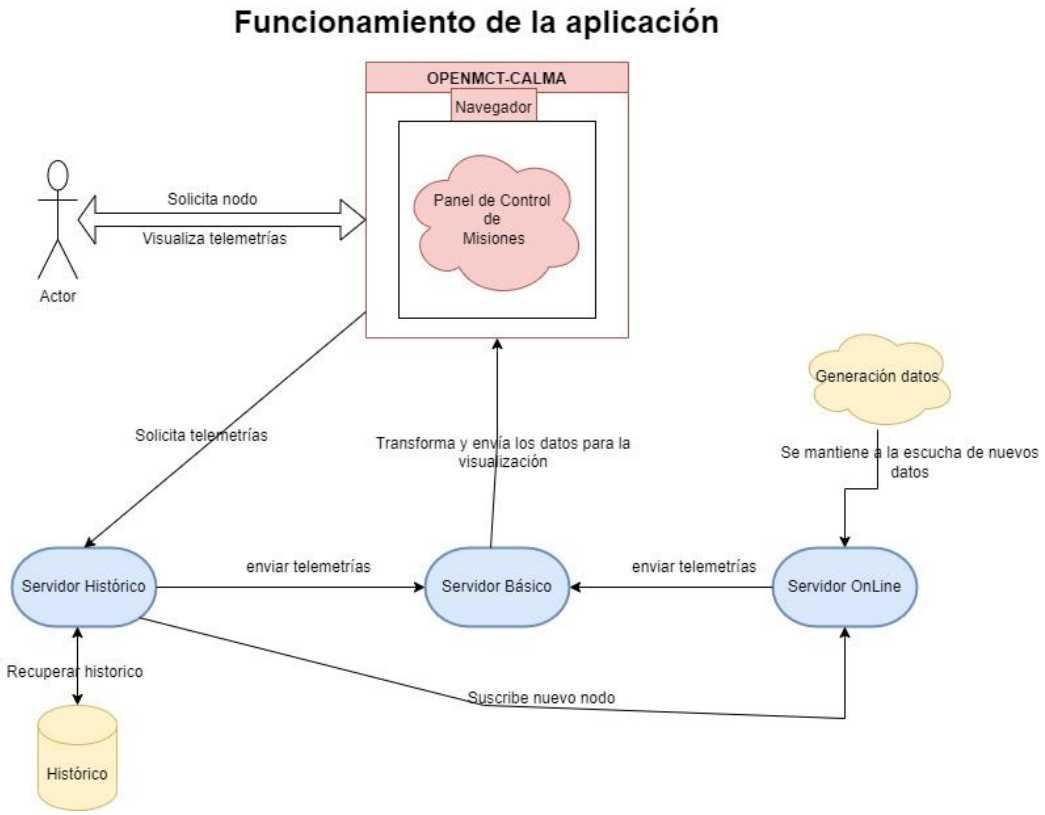


Ilustración 14 Funcionamiento de la aplicación

9 Pruebas y resultados

Se realizarán distintas pruebas para verificar el correcto funcionamiento de la aplicación. Lo más interesante de este proyecto, al menos desde mi punto de vista, es la **posibilidad de integrar**, en un futuro, un **sistema que utilice las lecturas telemétricas en tiempo real para la detección de errores, programación de alarmas, indicadores de fallo**, etc.

Tal como se ha venido explicado a lo largo de este documento, para este proyecto se han utilizado dos fuentes de datos:

- Datos telemétricos provenientes de la BD local.
- Datos generados mediante funciones matemáticas con las que se ha intentado simular un comportamiento anómalo en la recepción y fuente de datos.

En apartados anteriores se explicaron en detalle los pasos necesarios para configurar un entorno de trabajo, así como los procesos de implementación de los `plugins` y la integración de todos los elementos para dar lugar a una aplicación desarrollada bajo el `framework` OpenMCT.

Una vez realizado todo el proceso de desarrollo, para iniciar la aplicación sólo se necesitan dos líneas de comando:

- Posicionarse en el directorio de la aplicación: `cd openmct-calma`
- Arrancar la aplicación: `npm run start o npm start.`

La aplicación responderá con tres líneas en el log de la consola indicando la url en la que se aloja la aplicación para su ejecución.

```
ana@ana-Virtual-Machine:~$ cd openmct-calma
ana@ana-Virtual-Machine:~/openmct-calma$ npm run start

> @ start /home/ana/openmct-calma
> node server-calma/server.js

Servidor Basico esta en http://localhost:8081
Calma Historico esta en http://localhost:8081/historico
Calma en linea esta en ws://localhost:8081/enlinea
```

Ilustración 15: Respuesta de la aplicación tras el arranque

Un aspecto muy importante sobre los datos empleados en este proyecto tiene que ver con la manipulación de los mismos. Cuando se inicia la aplicación, los tres servidores quedan en estado activo a la espera de la llegada de una solicitud por parte del usuario-MCT. De las 3 urls, la que importa es la del `Servidor Básico` (contiene la interfaz y la lógica de la aplicación). Las otras dos urls sirven para gestionar el funcionamiento de la aplicación de acuerdo a las acciones de usuario. Es decir, en un primer momento sólo se mostrará la interfaz proporcionada por el servidor básico. Cuando el usuario-OpenMCT pulse sobre cualquier nodo, se activará el servidor histórico, que recuperará las telemetrías generadas desde el momento del arranque de la aplicación y a continuación da paso al servidor en línea que se encargará de la recepción y envío de datos online a la interfaz.

Una vez arrancado el servidor, se copia la URL del servidor básico en un navegador (en este caso se ha usado Mozzilla) o haciendo click derecho, se despliega un menú en el que se elige la opción de acceder a la url, la cual cargará la aplicación en el navegador. En la Ilustración 16: Interfaz de la aplicación se muestra la interfaz de la aplicación tras el primer arranque.

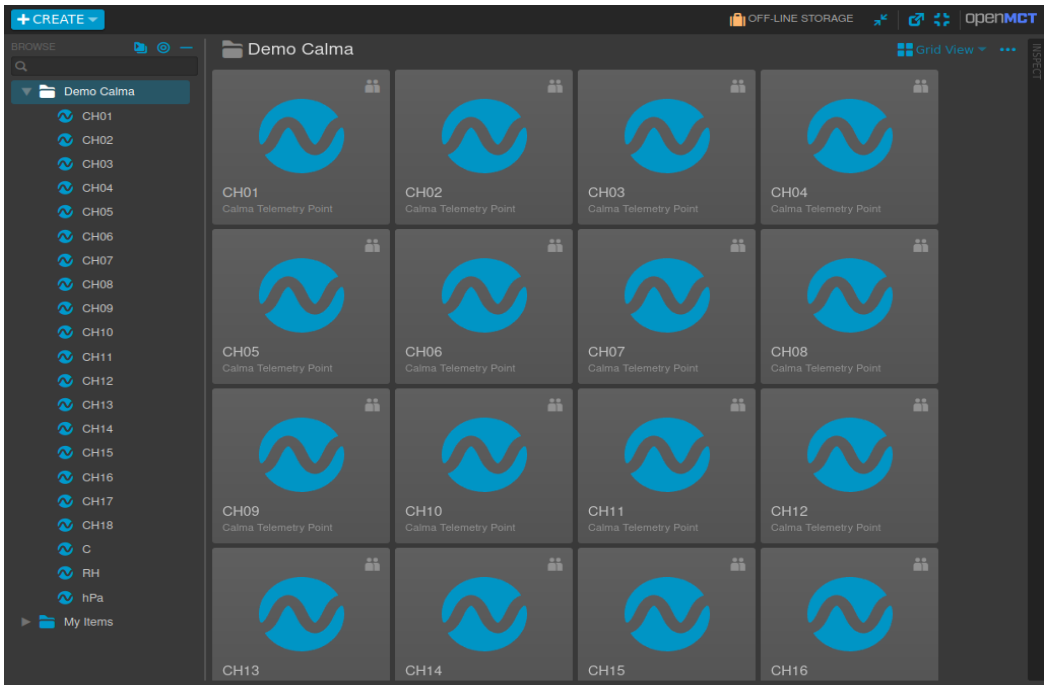


Ilustración 16: Interfaz de la aplicación

Cuando se solicitan las telemetrías de un nodo en concreto, el sistema obtiene los datos de manera conjunta, es decir:

- Se ejecuta la consulta SQL para obtener los datos iniciales de todos los canales declarados en BD.
- Se ejecutan las funciones matemáticas que generan las telemetrías simuladas.

Pero sólo se muestran en la pantalla, la gráfica generada a partir de las telemetrías obtenidas del nodo seleccionado.

Por un lado: el servidor histórico recuperará todos los datos telemétricos obtenidos o generados desde el arranque de la aplicación y los pasará al servidor básico para ser transformados en gráficas.

Por otro lado, el servidor en línea quedará activo esperando la llegada de datos nuevos que son enviados a la interfaz para ser “pintados en pantalla”.

Esta lógica se aprecia visualmente en el log de la consola:

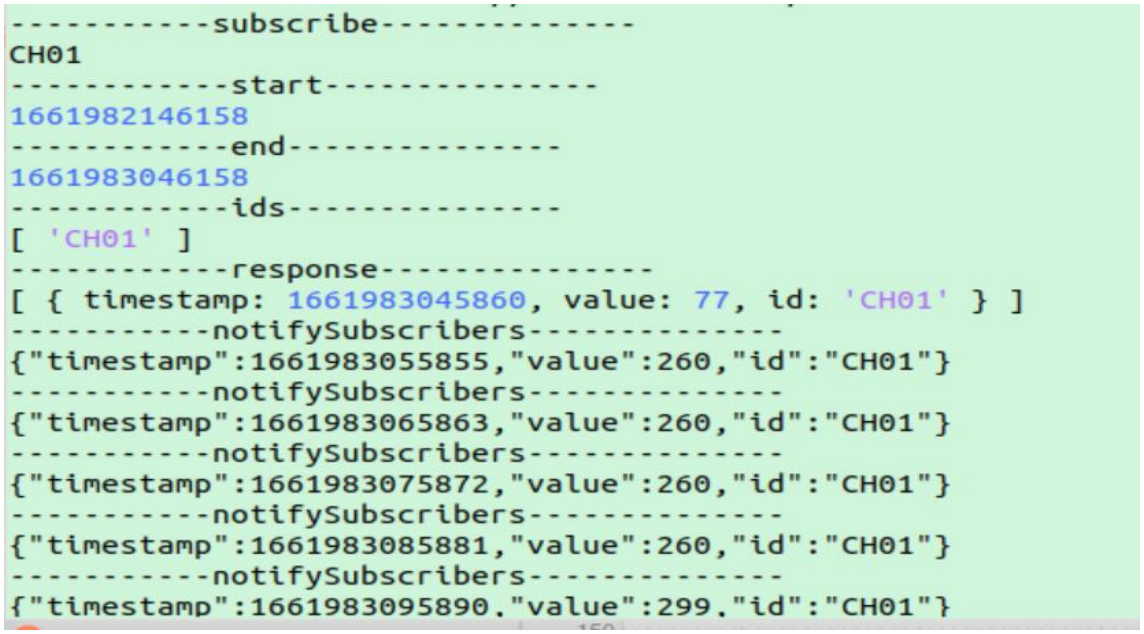


Ilustración 17: Log de la aplicación

- En esta prueba, el usuario ha solicitado las telemetrías del `CH01` que se corresponde con el `tubo contador de neutrones nº1`.

- En la consola se muestra el canal que se ha activado en ese momento. Este dato lo recupera el servidor histórico. Se marca también el inicio y el fin del histórico de datos.

- A continuación, el servidor histórico "pinta" en el log los registros obtenidos desde el arranque de la aplicación. En este caso sólo se recupera un único registro. Si la aplicación hubiese llevado más tiempo en ejecución se realiza una solicitud de otro nodo, el resultado habría sido el siguiente:

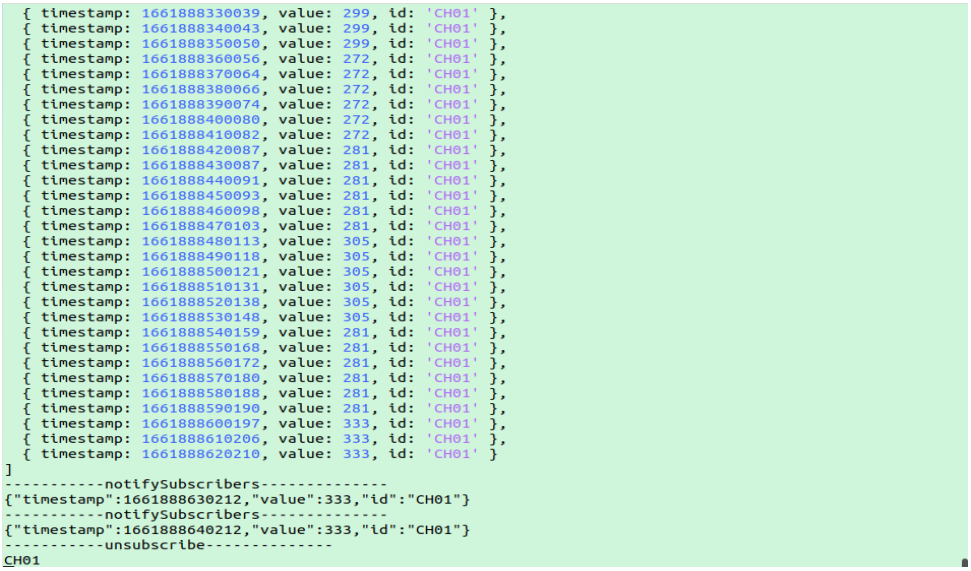


Ilustración 18: Histórico de telemetrías

- Una vez recuperado el histórico de datos, estos son enviados al servidor básico que, mediante funciones propias de OpenMCT, los transforma y se convierten en gráficas a visualizar en el navegador.

- A continuación, el servidor on-line queda a la escucha de la llegada de datos. Cada vez que llega un nuevo registro, notifica al suscriptor, en el log se registra el minuto de llegada, el valor y el canal activo.

- En relación a la telemetría representada mediante gráfica: debido al solapamiento del funcionamiento de los servidores, no se realiza en ningún momento ningún parón en el servicio: cada vez que se navega entre los distintos nodos del árbol de telemetrías, se recuperan las telemetrías del histórico, se pinta la gráfica correspondiente y ésta continúa dibujándose hasta que llega el siguiente dato.

- A nivel de interfaz: entre las funcionalidades más importantes que ofrece OpenMCT tenemos: **facilitar la detección de fallos** provenientes de las fuentes telemétricas, **obtener datos estadísticos**, **mínimos y máximos alcanzados**, **exportar las gráficas**, etc. Y todo esto de manera visual. En la Ilustración 32, se visualiza una telemetría bastante avanzada del tubo contador nº1. Con un simple vistazo es posible determinar si existe una anomalía o no. En este caso el funcionamiento es totalmente correcto, basta con mirar la gráfica: no existen picos fuera de lo normal y si se observan los datos de la Ilustración 19 se aprecia que no existe una diferencia considerable en los valores recibidos. A las 19:30 se registraron 269 impactos en el tubo contador nº1, siendo ese el número mínimo de neutrones registrados en el momento de la prueba. Mientras que a las 19:34 se detectaba el mayor número de neutrones: 360 recibidos en el rango de un minuto.

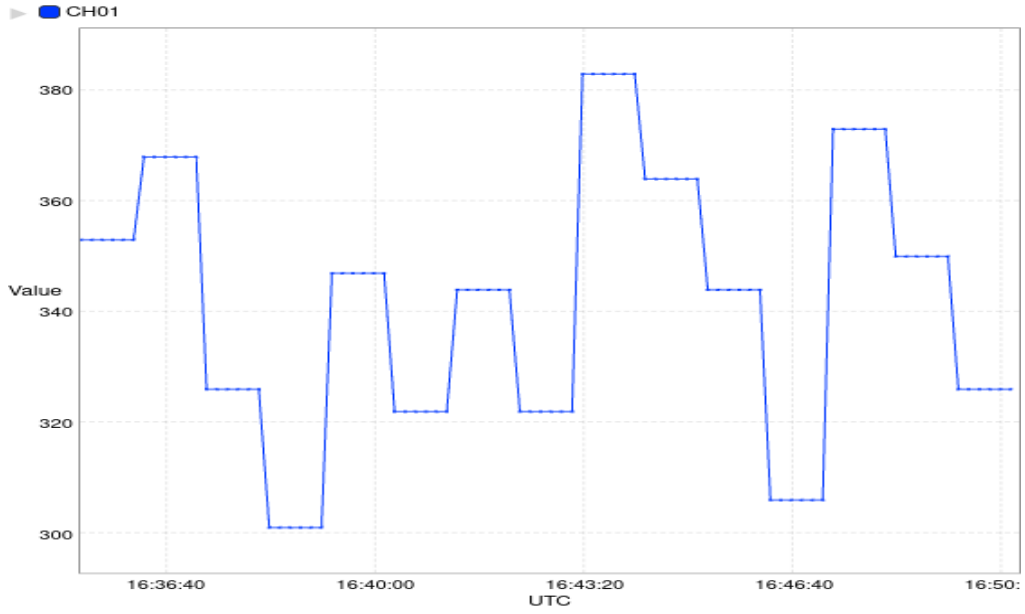


Ilustración 19: Telemetría CH01

- OpenMCT también permite visualizar en formato de tabla telemétrica: el número de neutrones recibidos por el canal y el minuto exacto de la llegada de los mismos. También permite realizar búsquedas de datos dentro de la misma tabla.

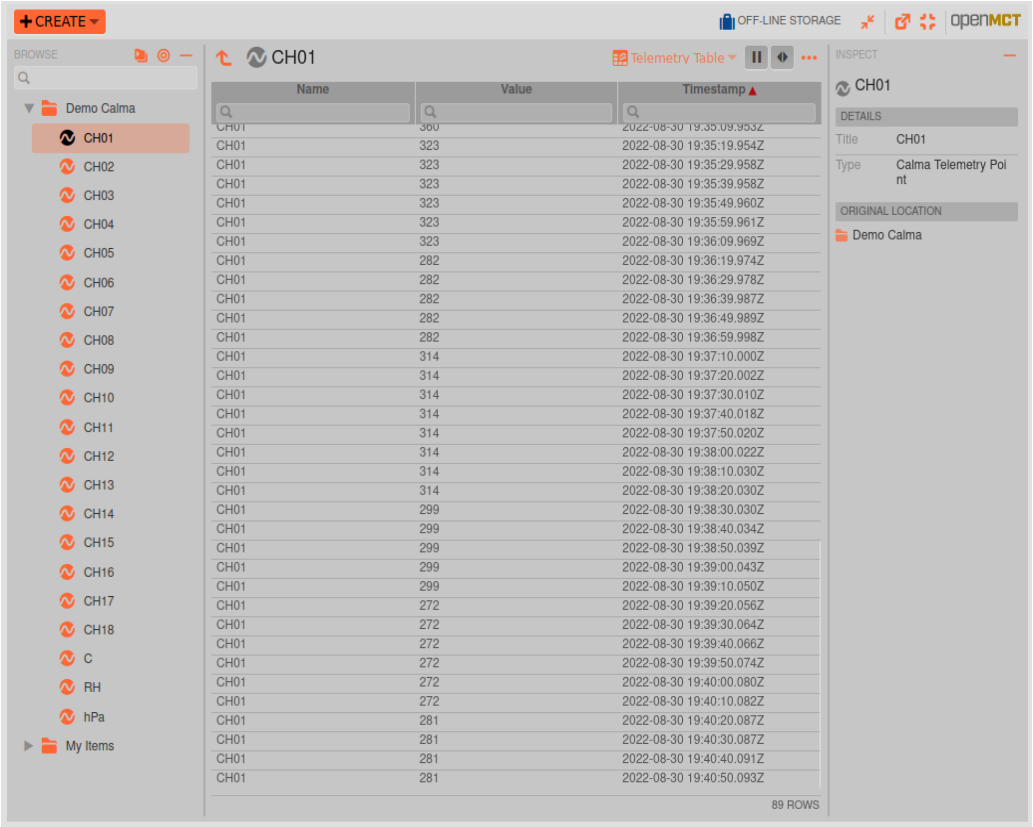


Ilustración 20: Telemetrías del CH01 en formato tabla

- También permite la exportación de gráficas para sean incorporadas a informes. La siguiente gráfica se corresponde a una telemetría exportada directamente desde la aplicación:

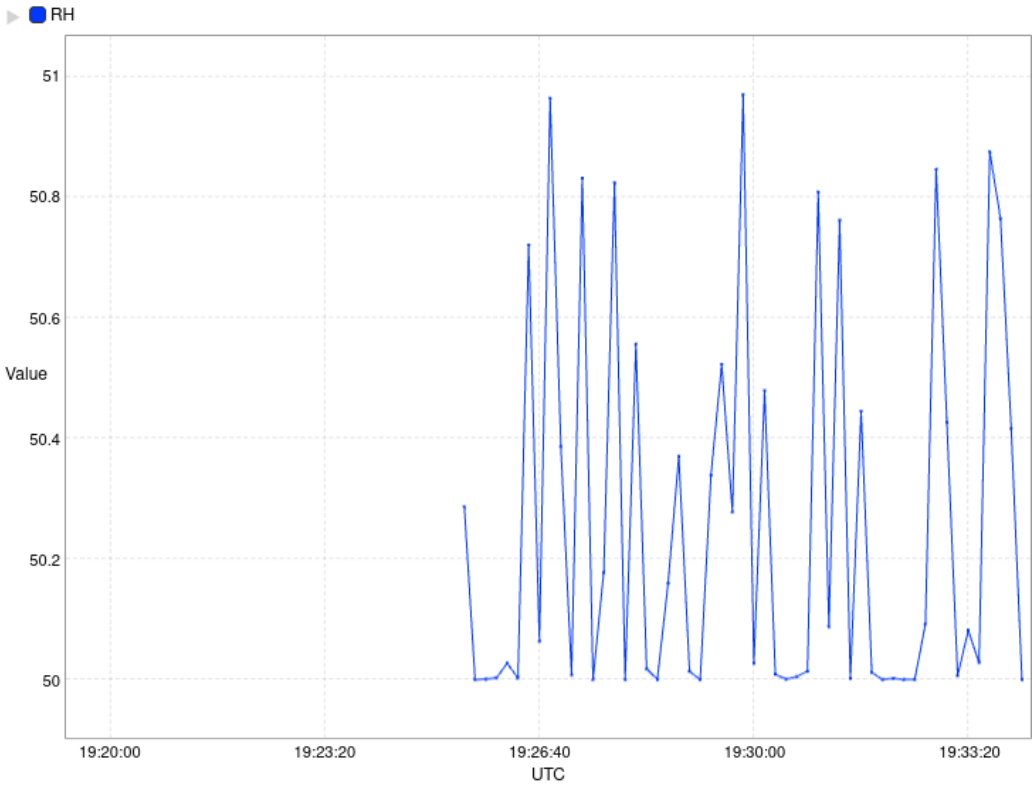


Ilustración 21 Telemetría exportada desde la aplicación.

9.1 Pruebas de telemetrías con datos simulados.

En este apartado del documento, se realizarán una serie de pruebas con datos generados y preparados para simular un comportamiento anómalo en el funcionamiento de los tubos contadores o en los medidores meteorológicos.

9.1.1 Caso de Prueba 1: Tubos contadores que dejan de recibir datos.

Para este caso de pruebas se han empleado los CH3 y CH5 que se corresponden a los tubos contadores nº 3 y nº5 respectivamente.

- Como función generadora de datos se ha empleado la misma expresión matemática para ambos nodos. Dicha función lo que hace es obtener el valor de inicialización de las variables que representan ambos nodos

- A continuación, **se elige el valor máximo entre 0 y el valor actual del nodo menos 0.5 (para el CH03) ó 0.25 (para el CH05).**

- Como resultado el valor máximo siempre se elegirá uno mayor que 0 que irá disminuyendo a medida que pase el tiempo hasta llegar a 0.

- Se inicia la aplicación y se selecciona el nodo CH5 o CH3.

- Se alterna entre ambos nodos para ver la evolución de la aplicación:
    - la respuesta del servidor histórico cuando se produce un cambio de solicitud de nodos, recuperando las telemetrías generadas desde el arranque de la aplicación, eliminando la suscripción de un nodo y activando la suscripción del otro.
    - La respuesta del servidor on-line que se queda a la espera de recibir nuevos datos generados.
    - La respuesta del servidor básico que se encarga de transformar los datos y enviarlos a la interfaz para que sean representados en el panel de control de misiones.

- Al cabo de un tiempo se obtienen las siguientes gráficas en el panel de control de misiones:

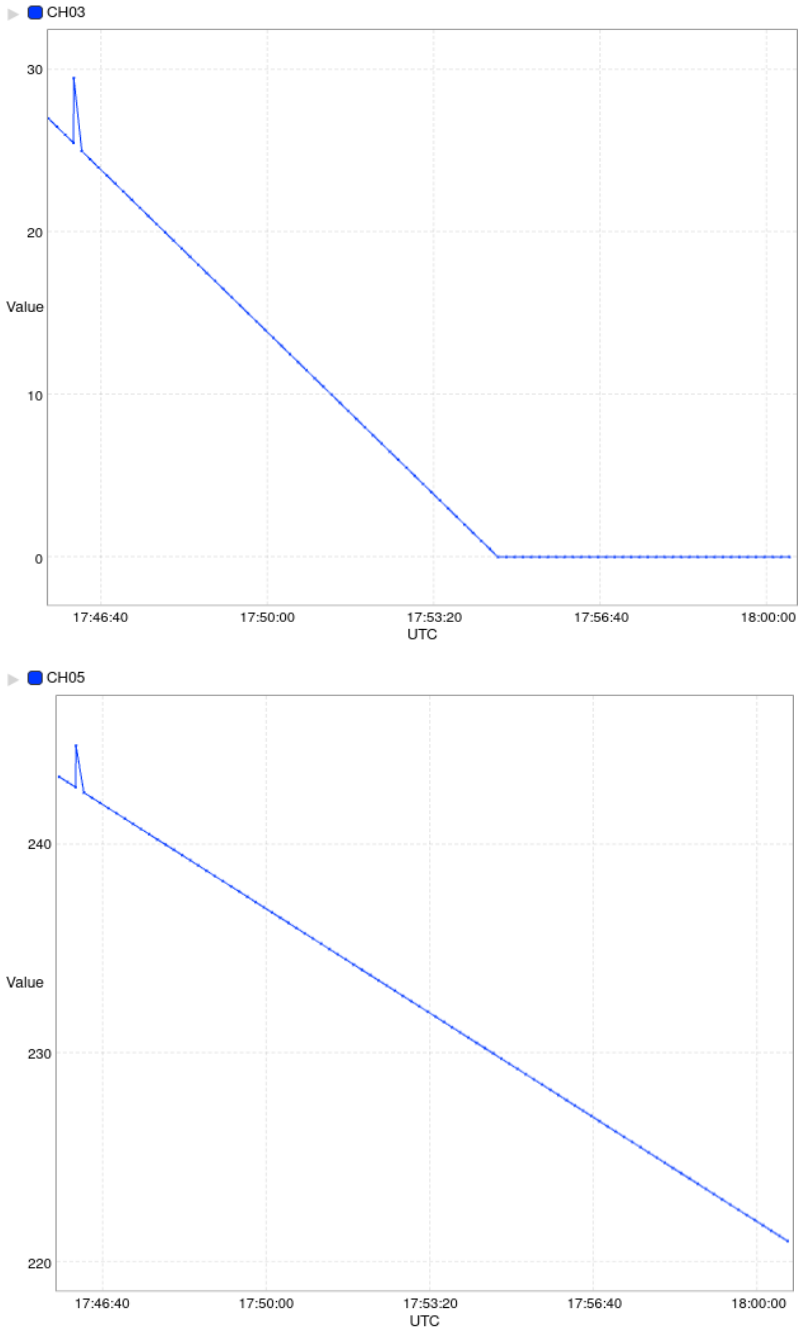
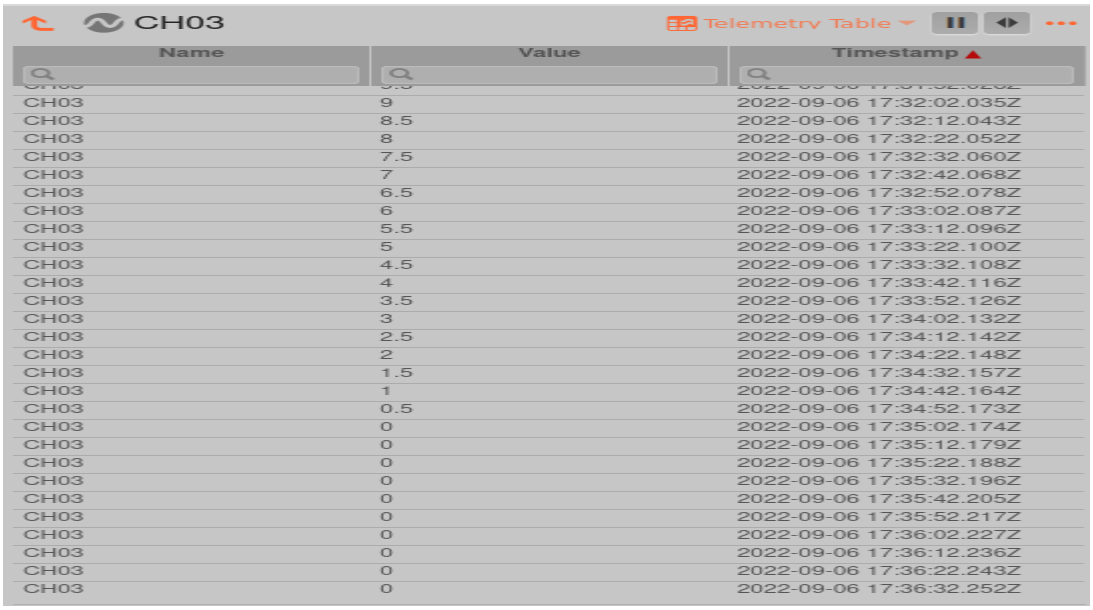


Ilustración 22: Telemetrías CH03 y CH05

- Como se puede apreciar en la Ilustración 22, ambos nodos empezaron con un valor inicial: 29.5 para el CH03 y 244.75 para el CH05.
- Ambas gráficas presentan una recta con pendiente decreciente. La correspondiente a CH03 decrece mucho más rápido que la generada para el CH05.
- En el minuto 17:35, el último valor recibido por el suscriptor CH03 es 0. Dicho valor se mantiene en el tiempo, originando una línea recta.
- CH05 sigue recibiendo valores, pero los datos recibidos son cada vez más pequeños. La evolución de esta telemetría es similar a la del CH03. Llegará un punto en el que será 0, trazando una línea recta.

Analizando las gráficas generadas a partir de los datos obtenidos, se puede inferir lo siguiente:

1. **El momento de inicio es correcto:** ambos tubos contadores han iniciado la actividad de recepción de datos a la vez que el resto de elementos telemétricos. Es decir, cuando se recupera el histórico de las telemetrías, ambos nodos muestran el mismo tiempo inicial de arranque.
2. Según las especificaciones descritas del sistema CaLMa, **los neutrones se contabilizan en un periodo igual a 1 minuto**. Si analizamos los datos recibidos por el tubo contador CH03, se aprecia que los datos están llegando con una diferencia de 10 segundos. **Se podría pensar que el sistema de adquisición de CaLMA está fallando**.
3. Por otro lado, los tubos contadores registran un número entero de neutrones. En el caso del CH03 y del CH05 se están registrando valores decimales.
4. A partir del minuto 17:35:02, los tubos contadores indicados no están registrando valores.



Name	Value	Timestamp
CH03	9	2022-09-06 17:32:02.035Z
CH03	8.5	2022-09-06 17:32:12.043Z
CH03	8	2022-09-06 17:32:22.052Z
CH03	7.5	2022-09-06 17:32:32.060Z
CH03	7	2022-09-06 17:32:42.068Z
CH03	6.5	2022-09-06 17:32:52.078Z
CH03	6	2022-09-06 17:33:02.087Z
CH03	5.5	2022-09-06 17:33:12.096Z
CH03	5	2022-09-06 17:33:22.100Z
CH03	4.5	2022-09-06 17:33:32.108Z
CH03	4	2022-09-06 17:33:42.116Z
CH03	3.5	2022-09-06 17:33:52.126Z
CH03	3	2022-09-06 17:34:02.132Z
CH03	2.5	2022-09-06 17:34:12.142Z
CH03	2	2022-09-06 17:34:22.148Z
CH03	1.5	2022-09-06 17:34:32.157Z
CH03	1	2022-09-06 17:34:42.164Z
CH03	0.5	2022-09-06 17:34:52.173Z
CH03	0	2022-09-06 17:35:02.174Z
CH03	0	2022-09-06 17:35:12.179Z
CH03	0	2022-09-06 17:35:22.188Z
CH03	0	2022-09-06 17:35:32.196Z
CH03	0	2022-09-06 17:35:42.205Z
CH03	0	2022-09-06 17:35:52.217Z
CH03	0	2022-09-06 17:36:02.227Z
CH03	0	2022-09-06 17:36:12.236Z
CH03	0	2022-09-06 17:36:22.243Z
CH03	0	2022-09-06 17:36:32.252Z

Ilustración 23: Análisis de las lecturas telemétricas

Se puede concluir perfectamente que, en este caso, los tubos contadores correspondientes al `CH05` y `CH03` están deteriorados y necesitan ser sustituidos:

- Si fuera un fallo el sistema de adquisición de datos, el resto de tubos contadores también estarían registrando telemetrías anómalas. (2)

- Si la diferencia entre los valores mínimo y máximos no fuera tan abismal (3) o al comparar las telemetrías del estos dos canales con las telemetrías del resto fueran algo similares, con recalibrar los elementos del monitor de neutrones de CaLMa, sería posible solucionar este problema.

- Pero como las gráficas tienen una pendiente decreciente que acaba en 0 (no se reciben más datos) trazando a continuación una línea recta, los tiempos de recepción son incorrectos ya que no se corresponden con los del resto de elementos telemétricos, el conteo de neutrones recibidos es incorrecto (no se asemeja al resto), se puede concluir perfectamente que existe un deterioro en ambos tubos contadores.

En un sistema completo, como el indicado en la Ilustración, cada vez que en el panel de control de misiones se detectara una gráfica de este estilo, se debería encender una alarma para indicar que hay un fallo crítico en uno de los tubos contadores o en los medidores meteorológicos.

9.1.2 Caso de Prueba 2: Tubos contadores de neutrones que necesitan ser recalibrados.

Para el segundo caso de prueba, se utilizará como elemento experimental el medidor correspondiente al tubo contador de neutrones CH06. Las condiciones del experimento son las siguientes:

- Al igual que el resto de elementos telemétricos, este tubo contador empieza a emitir datos a la vez que el resto.
- Se han definido dos constantes llamadas max y min con valores 400 y 280 respectivamente.
- La función matemática empleada es la siguiente: **Math.floor((Math.random() * (max - min + 1)) + min)**. Lo que hace es generar un número aleatorio entre dos límites.

Cuando el usuario solicita las telemetrías del nodo CH06, el panel de control de misiones mostrará la siguiente gráfica:

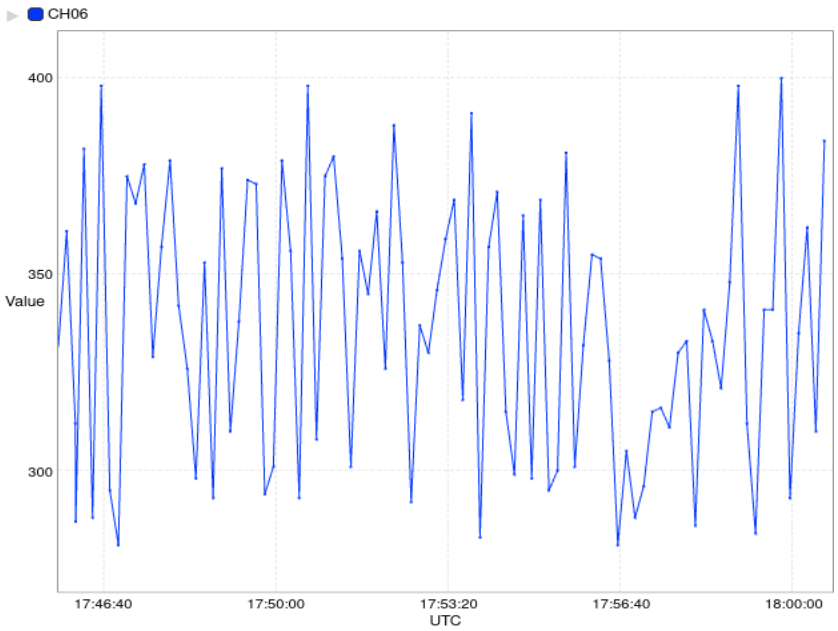


Ilustración 24: Telemetría CH06

Si se observa la gráfica, aparentemente los valores recibidos son correctos: durante todas las pruebas, el número de neutrones recibidos por cada tubo se mueve en un rango de 280 – 420 neutrones por minuto.

El problema radica en la forma de la telemetría: si se compara con el resto se puede apreciar que no existen picos: cada vez que llega un dato, este mantiene la línea recta hasta la llegada del otro dato:

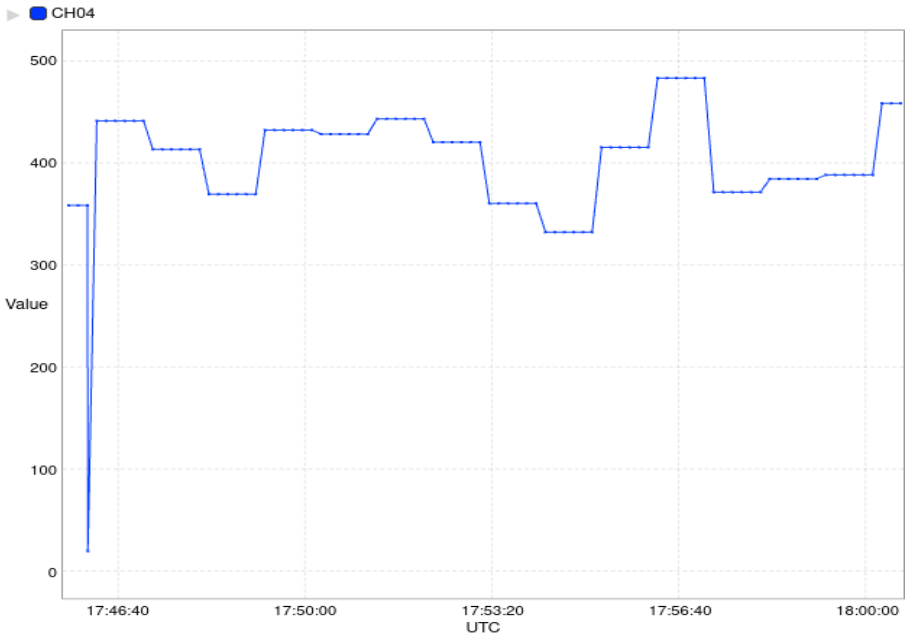


Ilustración 25 Telemetría para la comparativa

Al comprobar los datos recibidos por el tubo contador CH06 observamos que, **en el rango de 1 minuto, se están recibiendo telemetrías distintas**. Cuando lo normal es que, durante un minuto, repartido en tiempos de 10 segundos, se reciba el mismo número de neutrones. En cambio, para el tubo contador objeto del experimento, a las 19:24 se han recibido 6 valores totalmente distintos.

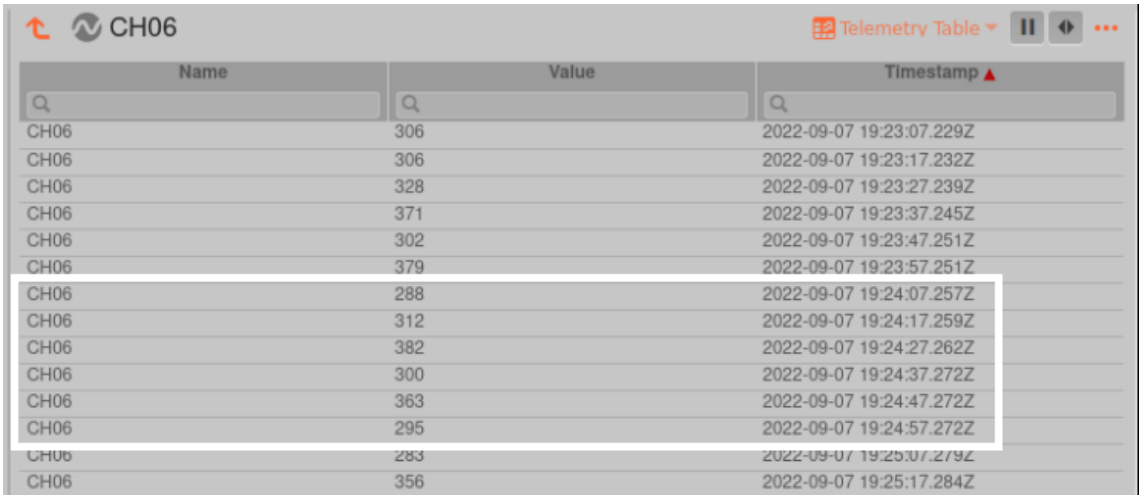


Ilustración 26 Tabla telemétrica para CH06

El resultado de esta prueba debería hacer saltar las alarmas de mantenimiento indicando que el tubo contador en cuestión necesita ser recalibrado porque no está llevando un conteo correcto de los neutrones recibidos.

9.1.3 Caso de Prueba 3: Medidor meteorológico que necesita mantenimiento.

Para el tercer y último caso de prueba, se utilizará como elemento experimental el medidor meteorológico RH encargado de medir la temperatura relativa del ambiente, en este caso de Guadalajara, ya que es en esta base donde ser realiza el experimento. Este medidor tiene una

gran precisión y sensibilidad que le otorga la capacidad de detectar variaciones en rango de decimales.

Los supuestos bases siguen siendo los mismos:

- El medidor inicia su actividad a la vez que el resto.
- Tiene asignado un valor inicial.
- Para generar los datos simulados, se emplea una función matemática que asigna a la variable `RH`, la suma de un valor fijo predeterminado (en este caso 50) más un número aleatorio utilizado como base elevado a 3 (`función Math.pow`).

Como resultado, en el panel de control de misiones del proyecto CaLMa aparecerá la siguiente telemetría:

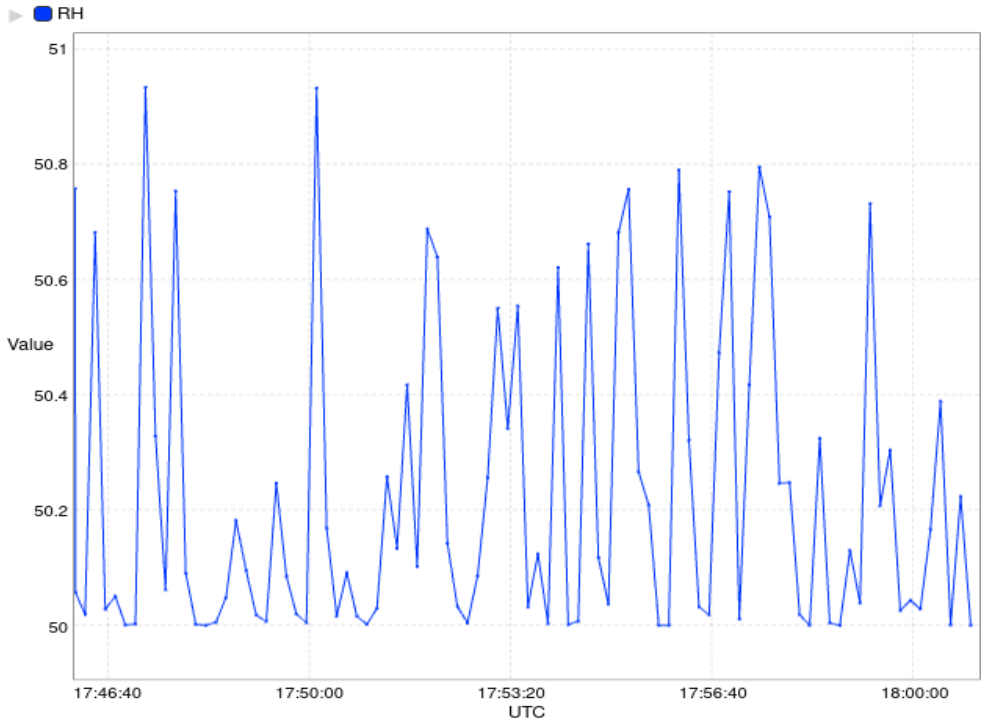


Ilustración 27: Telemetría correspondiente a la humedad relativa

La humedad relativa (RH) se mide en %. Aparentemente devuelve unas lecturas correctas:

- El experimento se inició a las 19:25:00. En el momento de la captura, los valores oscilaban entre un 50% y un 51%.
- No existen picos que se salgan del rango indicado.
- Los datos se generan a mayor velocidad, dando lugar a más puntos de lectura. Esto sigue siendo correcto, teniendo en cuenta la precisión y sensibilidad del medidor.

Al revisar los datos de la humedad de Guadalajara de los meses de junio y julio, se aprecia que éstos no se corresponden con los obtenidos en nuestra aplicación.

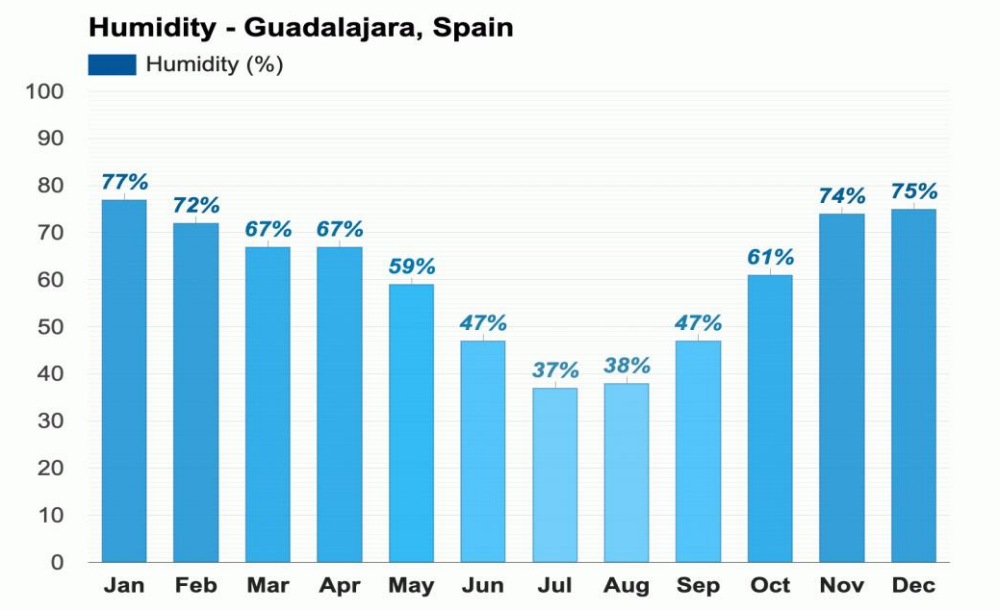


Ilustración 28: Humedad Relativa Guadalajara

En este caso: se debería activar una alarma de mantenimiento para revisar la fuente de datos o determinar la existencia de elementos externos que están alterando el correcto funcionamiento del medidor.

10 Conclusiones y trabajos futuros

La implementación de un panel de control de misiones para el proyecto CaLMa puede suponer un gran aporte para el tratamiento de datos telemétricos provenientes de los tubos contadores y medidores meteorológicos. El disponer de una herramienta que proporcione una visión global del estado general del monitor de neutrones en tiempo real sin tener que acceder a base de datos ni lanzar consultas complejas o aprender el funcionamiento de aplicaciones externas, podría proporcionar un gran avance a la hora de detectar fallos, interpretar resultados, inferir el funcionamiento de los objetos telemétricos, todo esto con un simple vistazo a la pantalla.

Por otro lado, utilizar un gran framework de desarrollo como es OpenMCT, dota a la aplicación de una gran versatilidad y escalabilidad: si mañana el equipo de desarrollo de CaLMa, decide añadir nuevas fuentes telemétricas, bastaría con añadir el código perteneciente a las nuevas fuentes de datos, es decir, ni siquiera haría falta crear nuevos plugins ni añadir nuevos servidores. Si deciden desarrollar una aplicación similar para el proyecto miniCaLMa, por ejemplo, gran parte del código desarrollado en el presente proyecto, serviría de base y ejemplo para la nueva aplicación. Y lo más importante: si se añaden nuevas funcionalidades ni siquiera haría falta sacar parches para la aplicación ni tener en cuenta el SO sobre el que se ejecutará, ya que basta con tener acceso a un simple navegador.

En conclusión, se ha obtenido una aplicación bastante intuitiva e interactiva gracias al framework de OpenMCT. Cualquier usuario, con un mínimo de conocimientos técnicos podría no sólo ejecutarla si no también saber interpretar los datos devueltos (telemetrías) ya que son simples gráficas.

Creo que en un futuro sería bastante interesante poder abordar el desarrollo completo, es decir, el especificado en la Ilustración 3 : La idea es que, más adelante, se pueda implementar un sistema compuesto por servidores que están a la escucha de las telemetrías proporcionadas por el sistema de adquisición de datos de CaLMa, traten esos datos en crudo, generen las gráficas correspondientes a las telemetrías y dependiendo de la situación, realice lo siguiente:

- Si los datos son incorrectos → se activen alarmas para indicar si algún medidor no está funcionando correctamente, si necesita mantenimiento, si la avería es grave, etc.
- Si los datos son correctos → almacenarlos en la base de datos.

11 Bibliografía

- Monitor de Neutrones (2021). Disponible en: https://www.nmdb.eu/ [Consulta: 20 de junio del 2022]
- J.Medina,J.J.Blanco,O.García,E.J.Catalán,D.García,D. Meziat, J. Rodríguez-Pacheco, S. Sánchez, M. Prieto y M.A. Hidalgo. Monitor de Neutrones de Castilla-La Mancha (2018).
- CaLMa: Monitor de Neutrones de Castilla-La Mancha. Space Research Group. University of Alcalá (2018). Disponible en https://www.nmdb.eu/public_outreach/es/04_nm/ [Consulta: 10 de junio del 2022]
- Real-time Neutron Monitor Database (2020). https://en.wikipedia.org/wiki/Real-time_Neutron_Monitor_Database [Consulta: 10 de junio del 2022]
- OpenMCT (n.d.). Disponible en: https://nasa.github.io/openmct/ [Consulta: 25 de enero del 2022]
- Github discussions: Nasa OpenMCT (2022). Disponible en https://github.com/nasa/openmct/discussions [Consulta: 10 de enero del 2022]
- Patrón de Publicador y Suscriptor (2022). Disponible en: https://docs.microsoft.com/es-es/azure/architecture/patterns/publisher-subscriber [Consulta: 10 de Agosto del 2022]
- Jet Propulsion Laboratory. California Institute of Technology. Jason-3 (2018). Disponible en: https://www.jpl.nasa.gov/missions/jason-3 [Consulta: 3 de marzo del 2022]
- Clima y previsión meteorológica mensual Guadalajara, España (2022). Disponible en https://www.weather-atlas.com/es/espana/guadalajara-clima [Consultado: 20 de agosto del 2022]
- GIT. Git[software]. Versión 2.25.1 Disponible en: https://git-scm.com/ [Consulta: 25 de enero del 2022]
- NODEJS. NodeJS[softare]. Versión 14.19.3 Disponible en: https://nodejs.org/en/ [Consulta: 25 de julio del 2022]
- MySQLWorkBench. MySqlWorkBench[software]. Versión 8.0.29. Disponible en: https://www.mysql.com/products/workbench/ [Consulta: 15 de febrero del 2022]