Desarrollo de red de sensores de irradiación solar

Iván García Palomino

Francisco José Hinojosa Pérez

GRADO EN INGENIERÍA DE COMPUTADORES,

FACULTAD DE INFORMÁTICA,

UNIVERSIDAD COMPLUTENSE DE MADRID.

Trabajo Fin Grado en Grado de Ingeniería de Computadores

Madrid, 24 de enero de 2019

Director/es y/o colaborador:

José Ignacio Gómez Pérez

Christian Tenllado van der Reijden

Resumen

Este proyecto engloba la idea de desplegar una red de nodos solares, formados por varios sensores, que repartidos por una zona geográfica realicen la recogida de muestras de varias magnitudes físicas, siendo la irradiación solar la más importante de ellas en este proyecto.

Los datos recogidos por los distintos nodos solares que conforman la red serán procesados por un sistema predictivo, que predecirá la radiación solar del área geográfica donde estén situados los nodos en un futuro próximo.

De manera más concreta, este trabajo se centra en el despliegue de la red de nodos solares y en la recogida de magnitudes físicas procedentes de los distintos sensores, para su posterior envío a un servidor a través de un gateway.

La red de nodos solares utiliza la tecnología LoRa para la comunicación entre los nodos y un gateway que les da salida a Internet. Dicho gateway utiliza el protocolo MQTT para la comunicación con el servidor utilizando un broker como intermediario.

En el servidor se utiliza la plataforma ThingsBoard como cliente suscrito al broker MQTT, que permitirá almacenar y representar gráficamente los datos recibidos.

Palabras clave

Nodo Solar, Sensor, Radiación, Servicios, Muestreo, Gateway, LoRa, MQTT, ThingsBoard

Abstract

This project includes the idea of deploying a network of solar nodes, made up of several sensors, distributed throughout a geographical area to collect samples of various physical quantities, with solar irradiation being the most important of them in this project.

The data collected by the different solar nodes that make up the network will be processed by a predictive system, which will predict the solar radiation of the geographical area where the nodes are located in the near future.

More concretely, this work focuses on the deployment of the network of solar nodes and the collection of physical quantities from the different sensors, for later sending to a server through a gateway.

The network of solar nodes uses LoRa technology for communication between the nodes and a gateway that gives them access to the Internet. This gateway uses the MQTT protocol for communication with the server using a broker as an intermediary.

On the server, ThingsBoard platform is used as a client subscribed to the MQTT broker, which will allow the received data to be stored and graphically represented.

Keywords

Solar Node, Sensor, Radiation, Services, Sampling, Gateway, LoRa, MQTT, ThingsBoard

Índice de Contenidos

Capítulo 1: Introducción

En la actualidad vemos como cada día se tratan temas relacionados con la producción o el consumo de energía, pues se relacionan con algunas de las cuestiones más punteras de la última década, como pueden ser el cambio climático o la extrema contaminación.

Además, hay que tener en cuenta que cada día se fija el precio de la energía en una subasta eléctrica de forma continuada lo que afecta directamente tanto a empresas generadoras de energía (de origen renovable o no), como a los consumidores de la misma. Si se pudiera prever la producción de energía solar a corto plazo (entre 30 minutos y 2 horas), se podría disminuir el uso de centrales térmicas, más contaminantes, para satisfacer la demanda en momentos de alta irradiación solar. Una mayor fiabilidad en la predicción de la producción solar futura, podría permitir así el cierre de plantas excedentarias.

A pesar de que las energías renovables llevan varios años abriéndose paso en nuestro entorno (en 2016 el 90% de la nueva potencia instalada en Europa fue de origen renovable[1]) es en la actualidad cuando la energía solar empieza a destacar en su contribución en el panorama energético (9 GW de nueva potencia instalada en Europa en 2017 tenía este origen) y con buenas previsiones de futuro[2]. En España está situada en séptimo lugar (aproximadamente 5 GW de potencia instalada en el momento de la realización de este proyecto según los datos de Red Eléctrica de España[3]) aunque continúa en constante evolución y se espera que hacia 2022 la potencia instalada de este origen se haya duplicado.

Sirve de motivación para este proyecto conseguir integrar completamente la energía solar en el pool de energías que se usan a diario, con el fin de disminuir el uso de centrales más contaminantes y conseguir satisfacer la demanda en momentos de alta irradiación solar.

Existen distintas técnicas para la realización de predicciones solares a corto plazo. Una de esas técnicas es la de “sky cams”, que consiste en la toma de fotografías terrestres al cielo para localizar y parametrizar el tipo de nubes para posteriormente predecir cuál será la posición de dichas nubes. Otra técnica de predicción utiliza fotografías realizadas desde satélites para determinar el tipo, tamaño y dirección de las nubes, y en base a ello, determinar las zonas nubosas.

[1] https://www.diariorenovables.com/2017.html
[2] https://www.energias-renovables.com/fotovoltaica
[3] https://www.ree.es/es/estadisticas-del-sistema-electrico/3015/3001

Este proyecto se centra, sin embargo, en la idea de poder desplegar una red de sensores que midan la irradiación solar en cada momento en diferentes lugares para conseguir la mayor cantidad de información posible para poder aplicar posteriormente técnicas de inteligencia artificial y, a partir de ello, conseguir predecir con la mayor exactitud posible la energía solar de la que disponer, pues puede resultar interesante intentar regular la producción de energía y poder ajustarla más al consumo real y con ello abaratar los costes tanto de fabricación como de uso de la misma.

De aquí el planteamiento de este trabajo, basado en el desarrollo y despliegue de una red de sensores compuesta por nodos propios basados en la placa de desarrollo LoPy[4]. En dicha red cada nodo es autónomo (alimentado por un panel solar y una batería externa) y tiene conectados varios sensores (temperatura, humedad, radiación solar y GPS) cuyos valores se leen de forma periódica y se envían mediante WiFi o LoRa, en función de la disponibilidad.

Concretamente en el resultado final de este proyecto, el envío periódico de muestras se realiza a través de una conexión LoRa (protocolo de red con modulación en radiofrecuencia) a un gateway cercano, que se encarga de comunicar vía protocolo MQTT[5] (protocolo ligero de uso habitual en IoT, Internet Of Things, basado en publish/subscribe), a un broker MQTT externo (servidor *"solarcasting.dacya.ucm.es"* disponible en la Facultad de Ciencias Físicas de la UCM) previamente configurado para recibir las publicaciones y que permite que se puedan suscribir clientes a ellas.

En dicho servidor a su vez está incluido un cliente MQTT (ThingsBoard[6], plataforma IoT open-source que permite un rápido desarrollo, gestión y escalado de proyectos de IoT) que está suscrito a estas publicaciones y permite la representación gráfica de los datos recibidos lo que proporciona poder comprobarlos de una manera más visual.

La figura 1 nos muestra la idea principal de cómo se despliega la red construida en este proyecto.

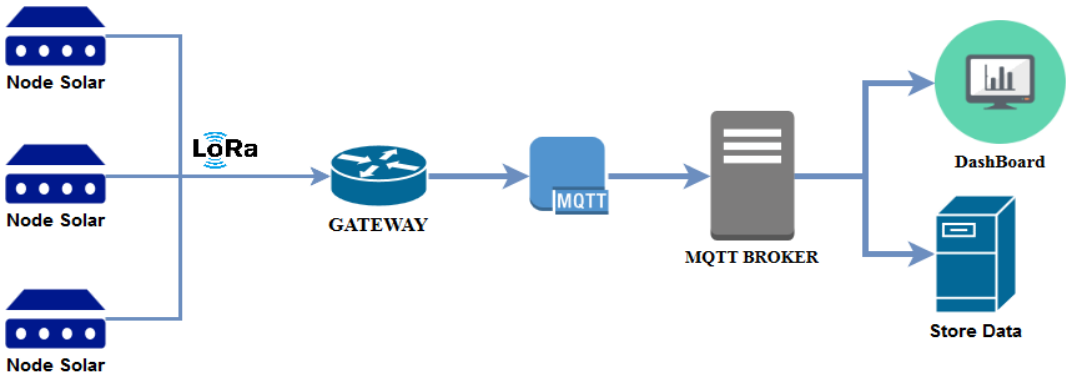

Figura 1: Despliegue de Red de Nodos

[4] https://docs.pycom.io/datasheets/development/lopy.html
[5] http://mqtt.org/
[6] https://thingsboard.io/docs/

En los capítulos siguientes, veremos paso a paso cada una de las etapas que se diferencian en el diagrama anterior, comenzando por una visión general del hardware disponible en el nodo solar, así como de los sensores disponibles, y siguiendo por el diseño y posterior implementación de la aplicación software. Dicho software permite a cada nodo solar obtener constantemente muestras telemétricas, buscando actuar de manera autónoma y, periódicamente enviar los datos recogidos a un gateway utilizando LoRa como mecanismo de transmisión. Este gateway mencionado publicará la información recibida, usando el protocolo MQTT, en un servidor externo que podrá almacenar e incluso representar gráficamente los datos obtenidos.

Este trabajo concluye en este punto de recogida y visualización de los datos obtenidos a partir de los sensores disponibles en el nodo solar propio, pero el proyecto general continúa en constante evolución usando estos datos para generar un modelo predictivo a partir de técnicas de inteligencia artificial. El objetivo final es desplegar una red de muchos de estos sensores en zonas de interés (plantas solares, pueblos/barrios con instalaciones domésticas) para tener datos precisos de radiación solar que sirvan de entrada al modelo predictivo y se generen predicciones en tiempo real. Destacar en este punto el uso de MQTT como protocolo de mensajería pues permite la suscripción a información recogida por cada nodo de la red de manera independiente a cualquier cliente suscrito a los canales del broker MQTT, pudiendo ser uno de ellos el propio modelo predictivo.

El plan de trabajo se divide en dos partes bien diferenciadas: la parte de diseño e implementación de la aplicación en el nodo solar, por un lado, y por otro lado llevar a cabo el diseño de protocolos de comunicación para su posterior aplicación con un servidor. Para ello, es necesario realizar primero tanto el diseño de lo que será la aplicación software que realice las mediciones en el nodo con una frecuencia que pueda variar en función de los intereses de cada situación, como el diseño de los protocolos de comunicación de entrada y salida del propio nodo, para conocer cómo debe la aplicación software enviar sus muestreos telemétricos y a su vez cómo tratar comunicaciones externas. Tras esto, se debe llevar a cabo la implementación del software de muestreo de las mediciones recibidas por los sensores disponibles para, por último, poder enviar esos datos a un servidor para su almacenaje y representación gráfica si se quisiera disponer de ella.

En este proyecto se consideran especialmente aspectos de consumo energético, tolerancia a fallos y facilidad en el despliegue de la red.

Chapter 1: Introduction

Nowadays we see how every day issues related to the production or consumption of energy are dealt with, since they relate to some of the most top questions in the last decade, such as climate change or extreme contamination.

In addition, we must bear in mind that every day the price of energy is fixed in an electric auction on a continuous basis, which directly affects both energy generating companies (renewable or nonrenewable), as well as consumers of it.

If the production of solar energy could be foreseen in short-term (between 30 minutes and 2 hours), the use of more polluting thermal power plants could be reduced to satisfy the demand in moments of high solar radiation. A greater reliability in the prediction of future solar production, could thus allow the closure of surplus plants.

Despite the fact that renewable energies have been around for several years (90% of the new installed power in Europe in 2016) it is at present when solar energy begins to stand out in its contribution in the energetic prospect (9 GW of new installed power in Europe in 2017 had this origin) and with good future forecasts. In Spain it is located in seventh place (approximately 5 GW of installed power, at the time of the realization of this project, according to the data of Red Eléctrica de España) although it continues in constant evolution and it is expected that the installed power from this origin will have been doubled in 2022.

It serves as motivation for this project to fully integrate solar energy into the energies pool that are used daily, in order to reduce the use of more polluting power plants and to meet the demand in times of high solar irradiation.

There are different techniques for making short-term solar predictions. One of these techniques is that of "sky cams", which consists in taking terrestrial photographs of the sky to locate and parameterize the type of clouds to later predict what the position of these clouds will be. Another prediction technique uses photographs taken from satellites to determine the type, size and direction of the clouds, and on that basis, determine the cloud areas.

This project focuses, however, on the idea of being able to deploy a network of sensors that measure solar irradiation at different times in different places and get as much information as possible in order to subsequently apply artificial intelligence techniques, it may be interesting to try to regulate the production of energy and adjust it more to real consumption and thereby lower the costs of both manufacturing and use of it.

Figure 1 gives an overview picture of the architecture.

Our project approaches this goal by developing and deploying a sensor network, which nodes are designed as extension boards for the LoPy board. Each node is autonomous (the extension board powers the LoPy from a battery and includes a charger for the battery that takes energy from a power solar panel) and has several sensors (temperature, humidity, solar radiation and GPS) whose values are read periodically and are sent via WiFi or Lora, depending on availability.

After some testing and some problems with the eduroam network at the UCM, we finally opted to send the samples using LoRa (network protocol with radio frequency modulation) although the wifi technology is still supported. The samples are sent to a LoRa gateway that is responsible for communicating using MQTT protocol. (Lightweight protocol commonly used in IoT, Internet of things, based on publish / subscribe), to an external MQTT broker (server *"solarcasting.dacya.ucm.es"* available in the Faculty of Physical Sciences of the UCM) previously configured to receive the publications and that allows that clients can subscribe to them.

The server includes a MQTT client (ThingsBoard, IoT open-source platform that allows a rapid development, management and scaling of IoT projects) that is subscribed to these publications and allows the graphical representation of the received data providing a human friendly visual representation of the data.

In the following chapters, we will see step by step each of the stages that differ in the previous diagram, starting with an overview of the available hardware in the solar node, as well as the available sensors, and following the design and subsequent implementation of the software application. This software allows each solar node to obtain periodic telemetry samples, and send the collected data to a gateway using LoRa as a transmission mechanism. This gateway will publish the received information, using the MQTT protocol, in an external server that will be able to store and even graphically represent the obtained data.

This work is part of a more ambitious project that will also consider the prediction algorithm that consumes the data gathered by with the sensor network we have developed.

The final goal is to deploy a network of many of these sensors in interesting areas (solar plants, towns / neighborhoods with domestic installations) to have accurate solar radiation data that serve as input to the predictive model and generate predictions in real time.

We would like to highlight that the use of MQTT as a messaging protocol allows the subscription to the information collected by each node of the network independently from any client subscribed to the channels of the MQTT broker, one of them being the predictive model itself.

The work plan is divided into two well differentiated parts: the design and implementation part of the application in the solar node on the one hand, and the design of communication protocols for later application with a server on the other hand. To do this, it is necessary first to design the software application that will perform the measurements at the node with a frequency that could change depending on the interests of every situation, as the design of the communication entry and exit protocols of the own node to know how the software application should send his telemetric samplings and in turn how to deal with external communications. In order to finally be able to send this data to a server for its storage and graphic representation if it were to be available.

In this project, aspects of energy consumption, fault tolerance and facility of network deployment are especially considered.

Capítulo 2: Hardware Nodo Solar

Antes de conocer todos los elementos de los que disponemos, cabe remarcar que los requisitos del nodo son: el bajo coste y sobre todo el bajo consumo, buena conectividad, y el muestreo y envío de sensores con periodos modificables de forma remota. Con ello se encontraron ciertas dificultades a la hora de la realización de este proyecto debido a los limitados recursos del sistema, algo habitual en el contexto de bajo consumo.

El nodo solar está compuesto de diferentes componentes hardware que otorgan los recursos necesarios para su funcionamiento autónomo. La composición del nodo consta de una placa LoPy[7] de desarrollo conectada a una placa de expansión diseñada en la Universidad Complutense de Madrid por Manuel Martínez Cebreiros en el Trabajo de Fin de Máster *"Diseño e Implementación de un Nodo Sensor de Radiación"* del Máster Universitario en Nuevas Tecnologías Electrónicas y Fotónicas con varios sensores encargados de recoger las diferentes muestras.

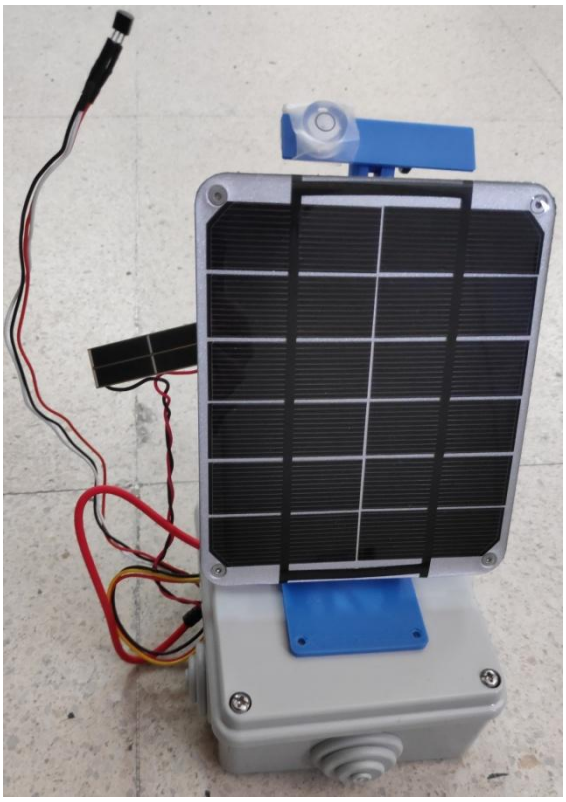

Figura 2: Fotografía de un Nodo Solar

[7] https://pycom.io/wp-content/uploads/2017/08/lopySpecsheetAugust2017.pdf

Pycom pone a nuestra disposición un chipset Espressif ESP32[8] que cuenta con un microprocesador Xtensa LX 6 con núcleo de Tensilica, una memoria RAM de 512KB, una memoria flash de 4MB y capacidad para conexiones Wifi, Bluetooth y LoRa, contando también con interfaces UART, I2C, SPI, I2S y tarjetas microSD. A parte de esto, se dispone de un firmware incluido que establece un sistema de ficheros y nos proporciona un intérprete de MicroPython[9] que facilita la programación de aplicaciones de uso. Dicho intérprete nos da opciones de trabajar con multi-threading además de poder acceder de forma más sencilla a las interfaces comentadas anteriormente a través del uso de librerías que se pueden consultar en la documentación oficial de Pycom[10].

Aparte de lo ya mencionado, cabe remarcar la necesidad de ahorrar toda la energía posible ya que en búsqueda de que el nodo solar sea lo más autónomo posible hay que tener en cuenta que tan sólo dispone de una batería de litio para su alimentación cuando no se dispone de una fuente de luz que alimente al panel solar principal. Por ello, el nodo debe entrar en modo *"deepsleep"* (llamamos *"deepsleep"* al modo de ahorro de energía disponible en la LoPy con el cual reduce al mínimo su consumo energético y por tanto deja de ejecutar la aplicación) todas las horas posibles en las que no se disponga de luz solar para minimizar el consumo en momentos de inactividad durante el día (apagando sensores e incluso se podría contemplar dormir durante el día si la frecuencia de muestreo lo permitiera).

Este período de tiempo que el nodo queda en ahorro de energía es variable en función del horario que el cliente de la aplicación en cuestión quiera, tan sólo modificando la hora de despertarse y la hora de dormirse en un fichero de configuración tal y como veremos en la sección SamplingService del Capítulo 3 de este mismo documento.

[8] https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf
[9] https://micropython.org/
[10] https://docs.pycom.io/

2.1. Pinout LoPy

Para poder conectar todos los componentes de manera adecuada a la LoPy se dispone de un pinout[11] proporcionado por Pycom que nos permite conocer la manera de acoplar los elementos hardware externos a dicha LoPy. En nuestro caso la distribución ha quedado tal y como se muestra en la figura 3.

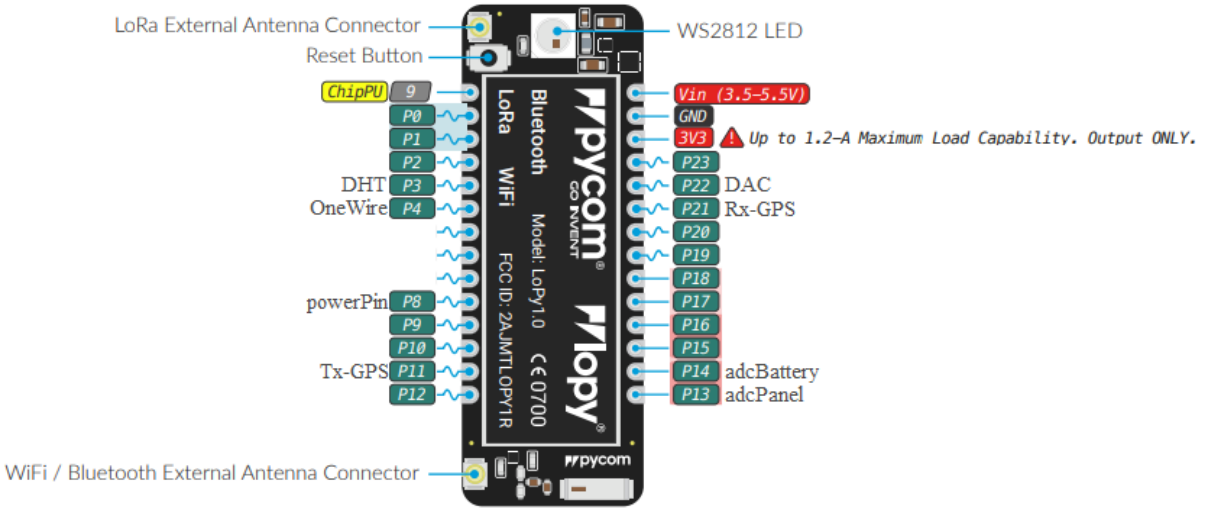

Figura 3: Pinout LoPy

Como se puede apreciar en la figura 3, disponemos de los siguientes elementos conectados a la placa:

- En el pin P3 queda conectado un sensor de humedad y temperatura DHT(módulo AM2302[12]), encargado de medir la temperatura y la humedad en el interior de la estructura protectora del nodo.
- El pin P4 queda configurado para la conexión OneWire de un sensor de temperatura DS18X20[13], utilizado para la medición de temperatura exterior de la estructura protectora del nodo.
- Los pines P11 y P21 son utilizados para la lectura y escritura en UART (Transmisor-Receptor Asíncrono Universal de dispositivos serie) de un módulo GPS GNSS modelo NEO-7M[14] que nos proporciona las coordenadas geográficas y la hora actual haciendo uso del protocolo ubx de ublox-7 para recibir y enviar mensajes.

  Dicho posicionamiento geográfico e información temporal es importante en el envío de muestras de los distintos sensores al gateway y posteriormente del gateway a un servidor encargado de almacenar los datos.

[11] https://docs.pycom.io/.gitbook/assets/lopy-pinout.pdf
[12] https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf
[13] https://datasheets.maximintegrated.com/en/ds/DS18B20-PAR.pdf
[14] https://www.u-blox.com/sites/default/files/products/documents/NEO-7M.pdf

- El pin P13 queda conectado a un módulo solar SLMD121H04L[15] que contiene cuatro células fotovoltaicas, y utilizamos como sensor de radiación.
- El pin P14 está conectado a una batería externa LP503448[16] de litio con una capacidad de 800mAh, utilizada para alimentar el sistema. Adicionalmente para que el nodo sea autónomo, en la placa diseñada por la Universidad Complutense se incluye la circuitería necesaria para adaptar el voltaje de un panel solar MEDIUM 6V 2W SOLAR PANEL[17] para la carga de la batería y alimentar la LoPy a una tensión constante de 3.3V.
- El pin P22 era utilizado inicialmente en el diseño original anterior para una de las conexiones del GPS. Sin embargo, este pin está conectado internamente a un DAC (conversor digital-analógico) utilizado en el diseño para mantener una tensión de referencia (aproximadamente 0,5V), utilizado por otras partes del circuito. Descubrir este hecho fue uno de los puntos complicados a la hora de llevar a cabo esta parte del proyecto, pues fue de complicada detección y además resultó necesario consultar el diseño original que ya estaba realizado de años anteriores y adaptarlo para poder utilizar todos los módulos disponibles actuales.
- El pin P8 es usado para una funcionalidad específica de la placa de expansión denominada como *powerPin*.

[15] http://ixapps.ixys.com/DataSheet/SLMD121H04L_Nov16.pdf
[16] http://www.fullwat.com/documentos/000485-LNK03447.pdf
[17] https://github.com/VoltaicEngineering/Solar-Panel-Drawings.pdf

2.2. Sensor de radiación

Uno de los objetivos de este proyecto es el bajo coste, principal razón por la que se ha elegido un módulo solar SLMD121H04L para la captura de datos de radiación solar en lugar de utilizar, por ejemplo, un piranómetro que tiene el mismo propósito, pero su valor excede el presupuesto deseado.

Un piranómetro es un instrumento meteorológico utilizado para medir de manera muy precisa la radiación solar incidente sobre la superficie de la Tierra. Se trata de un sensor diseñado para medir la densidad del flujo de radiación solar (kilovatios por metro cuadrado) en un campo de 180 grados.

Como no disponemos de un instrumento así para este proyecto, utilizamos el módulo solar SLMD121H04L que contiene cuatro células fotovoltaicas como sensor de radiación, de tal forma que el circuito analógico que incluye la placa de expansión UCM sitúa el punto de trabajo de la placa solar cerca del cortocircuito. La corriente de cortocircuito es proporcional a la radiación solar recibida por el panel, por lo que, midiendo dicha corriente, obtenemos una estimación de irradiación. Para medir la corriente, la placa UCM contiene un circuito de transimpedancia que proporciona un voltaje proporcional a la corriente. Ese voltaje es obtenido con un conversor analógico-digital (ADC) que nos proporciona una estimación más precisa de la irradiación solar que obtener los W/m^2 de un sitio concreto, lo que nos permite tener mejores datos a tratar con un modelo predictivo.

2.3. Sensor de temperatura interna y humedad

Cómo se vio en la figura 2, que mostraba la fotografía de un nodo ya montado, se puede observar que la placa LoPy está protegida por una estructura que le permite resguardarse de las inclemencias climáticas. Debido a esta estructura la placa queda aislada del exterior, pero esto nos produce la necesidad de conocer la temperatura interna de la estructura por si se produjera algún tipo de calentamiento en su interior.

Para medir esta temperatura queda instalado dentro de la estructura con la placa un sensor DHT22, bastante económico incidiendo una vez más en el objetivo del bajo coste, que nos permite conocer esta temperatura de manera bastante precisa y que adicionalmente nos aporta el dato de la humedad ya que este sensor también dispone de ello.

El valor de este sensor es obtenido a través de lecturas continuas de los pulsos producidos en su pin correspondiente (P3), con una librería especialmente diseñada para este sensor. Las lecturas del sensor proporcionan un valor distinto cada 2 segundos aproximadamente.

2.4. Sensor de temperatura externa

Para la medición de la temperatura externa de la estructura de los nodos, usamos un sensor de temperatura DS18X20, que resulta ser bastante económico y que utiliza el sistema de comunicaciones en bus 1-Wire.

El protocolo 1-Wire consiste en iniciar la comunicación enviando una señal de reset durante 480 µs o más a nivel bajo, pulsos largos (60 µs) para enviar bits de valor cero y pulsos cortos (menos de 15 µs) para los bits con valor uno, formando paquetes de ocho bits.

Para identificar cada dispositivo conectado al bus se utilizan direcciones únicas de 8 bytes que incluyen uno como identificador del hardware, el más significativo, y otro para la verificación CRC, el menos significativo.

Como ocurre en otros protocolos similares, en el inicio de la comunicación se incluye la identificación del dispositivo y la operación que se desea realizar (como escribir en la memoria o leerla, por ejemplo).

Para el uso de este sensor es importante remarcar que el uso del protocolo 1-Wire implica la necesidad de un único cable para enviar y recibir datos, lo que permite el ahorro de un cable para casos en los que el número de conexiones es limitado.

Por otro lado, remarcar que, para el uso de este sensor, Pycom proporciona una librería *"onewire"* que facilita su desarrollo.

2.5. Módulo de posicionamiento GPS

Para mantener la hora y la fecha siempre sincronizados en el RTC interno del sistema, y adicionalmente disponer de la situación geográfica de cada nodo de la red desplegada, se dispone de un módulo GPS GNSS.

Esto mismo podría ser realizado utilizando un RTC externo que mantuviera siempre la hora de manera sincronizada, sin embargo, un módulo como este nos permite conocer también la situación geográfica de cada uno de los nodos de la red de forma adicional. El módulo se pueden consultar con más detalle en la sección 3.3.2. del Capítulo 3 de este documento.

2.6. Uso de powerPin

La placa UCM diseñada permite cortar la alimentación de todos los sensores para reducir el consumo. Es necesario activar el powerPin (P8) para poder consultar cualquiera de los sensores y es conveniente desactivarlo si se prevé un periodo largo de inactividad, pues el ahorro de energía es un aspecto crucial para un nodo autónomo.

Además, debido a que este pin requiere de un tiempo para su activación, se puede producir un pequeño delay a la hora de realizar las mediciones.

Adicionalmente, como este pin es compartido entre todos los sensores y con la intención de no mantenerlo siempre activo buscando el menor consumo posible del sistema, es necesario el uso de un cerrojo global para garantizar que cada sensor puede activar disponer de este pin activo para poder realizar sus muestreos correctamente.

2.7. Características de red LoPy

Al principio de este capítulo, mencionamos como la placa LoPy dispone de capacidad para conexiones Wifi, Bluetooth y LoRa.

Para la realización de este proyecto nos decantamos por el uso de LoRa[18], ya que cumple el objetivo del bajo consumo pretendido y además proporciona un largo alcance para las comunicaciones, disponiendo de un bajo ancho de banda.

LoRa es la capa física o la modulación inalámbrica utilizada para crear el enlace de comunicación de largo alcance, mientras que LoRaWAN define el protocolo de comunicación y la arquitectura del sistema para la red. Dicho protocolo proporciona, además, robustez ante interferencias y seguridad en la red, todo ello cumpliendo con el bajo consumo.

Adicionalmente, comentar que la modulación LoRa trabaja con ISM (Industrial, Scientific and Medical), bandas reservadas internacionalmente para uso no comercial de radiofrecuencia electromagnética en áreas industriales, científicas y médicas. En nuestro caso realizamos las conexiones a través de la banda regional *EU 868*.

[18] https://www.tuv.com/TUeV_Rheinland_Overview_LoRa_and_LoRaWANtmp.pdf

Capítulo 3: Arquitectura Software

El principal objetivo en el diseño de la arquitectura software era el de otorgar facilidad para añadir o eliminar funcionalidades fácilmente sin repercutir en el funcionamiento del sistema, por ese motivo diseñamos la aplicación con diversos servicios, cada uno cumple con una funcionalidad determinada. Para cumplir con este objetivo diseñamos distintos módulos que denominamos servicios y pretendiendo que cada uno de ellos sea lo más independiente posible del resto de servicios en su funcionamiento, esto queda reflejado en el siguiente diagrama general.

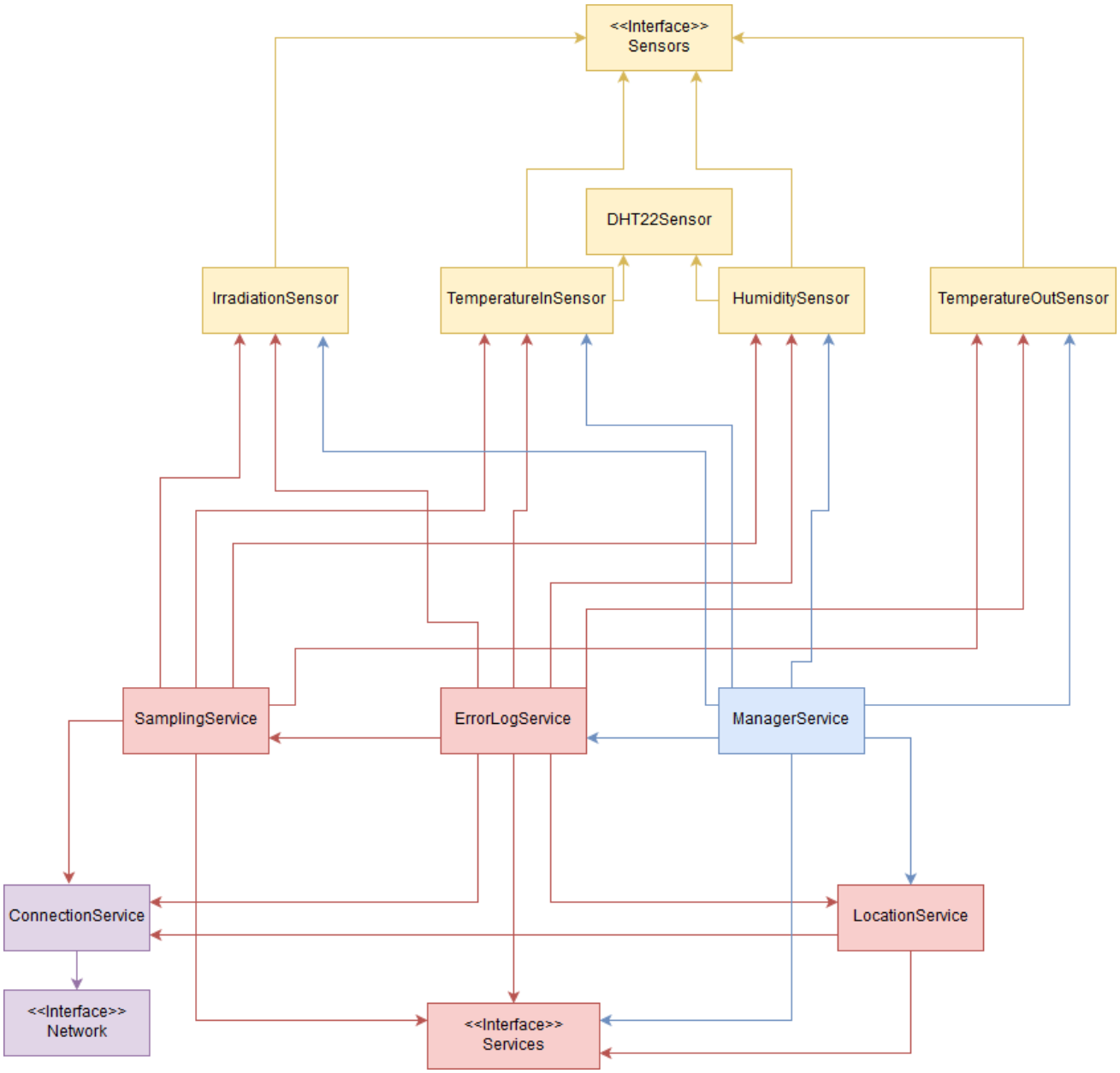

Figura 4: Diagrama General de la Arquitectura Software

Otro de los objetivos en el diseño era el de poder modificar atributos de algunos servicios, con este propósito diseñamos un fichero de configuración independiente para cada servicio en el que se guardan ciertos atributos configurables y habilitando la posibilidad de modificar dichos atributos de forma remota vía conexión LoRa o Wifi mediante las interfaces diseñadas que trataremos más adelante, aunque esta funcionalidad no está implementada en este proyecto.

En el diseño software hemos dividido en tres tipos los servicios que forman la aplicación. Los servicios denominados sensores (*IrradiationSensor*, *TemperatureInSensor*, *HumiditySensor*, *TemperatureOutSensor*) dispondrán de un hilo de ejecución independiente. De este modo se garantiza una mayor precisión en la toma de muestras en cada sensor, que pueden configurarse con frecuencias de muestreo independientes. Estos servicios implementan la *interface Sensors*.

Los servicios que no son sensores, que denominamos servicios generales, otorgan al sistema funcionalidades de gestión: *ErrorLogservice* gestiona los posibles errores y warnings; *SamplingService* es el encargado de solicitar muestras a los sensores para el posterior envío periódico de dichas muestras. *LocationService* es el encargado de la conexión con el satélite de posicionamiento; *ManagerService* es el encargado de gestionar el arranque y configuración inicial del sistema. Estos servicios implementan la *interface Services*, aunque pueden tener más métodos en función de la necesidad de cada servicio.

Por último, en la arquitectura software se incluye una *interfaz Network* para los diferentes tipos de conexión posible Wifi o LoRa. En nuestro proyecto implementamos el servicio *ConnectionService* encargado de realizar la conexión vía LoRa con un gateway cercano.

El código fuente de la aplicación aquí documentada puede ser consultado en el enlace https://github.com/Development-of-solar-irradiation-sensor-network.

En el proceso de desarrollo de este software hemos utilizado la aplicación de escritorio Atom y su respectivo paquete de desarrollo Pymakr que incluyen una licencia de uso GPLv3, el módulo ESP proporcionado por Pycom cuenta con una licencia Apache 2.0, la implementación en MicroPython tiene licencia MIT, las librerías propias de Pycom a las que accedemos en este proyecto poseen licencia GPLv3, la herramienta node-RED necesaria para la interacción nodos-gateway y la plataforma ThingsBoard que utilizamos como cliente MQTT para las representaciones gráficas tienen licencia Apache 2.0.

En base a las licencias de las distintas librerías (por ejemplo, *"onewire"* proporcionada por Pycom) utilizadas en el desarrollo del proyecto, decidimos otorgar una licencia GPLv3 "o una versión posterior" (GPLv3+) a este proyecto, compatible con todas las citadas anteriormente y que permite su uso comercial, su modificación, distribución, derecho de patente y uso privado.

3.1. Estructura general de directorios

Los servicios disponen de directorios almacenados en la memoria flash de la placa de desarrollo LoPy, en los que se almacenan los distintos ficheros de configuración, todos ellos llamados “conf.txt”. En el caso de *ErrorLogService* también almacena en su directorio los ficheros log generados. Los ficheros de configuración tienen la estructura de un diccionario en python para facilitar el proceso de lectura y escritura de los ficheros.

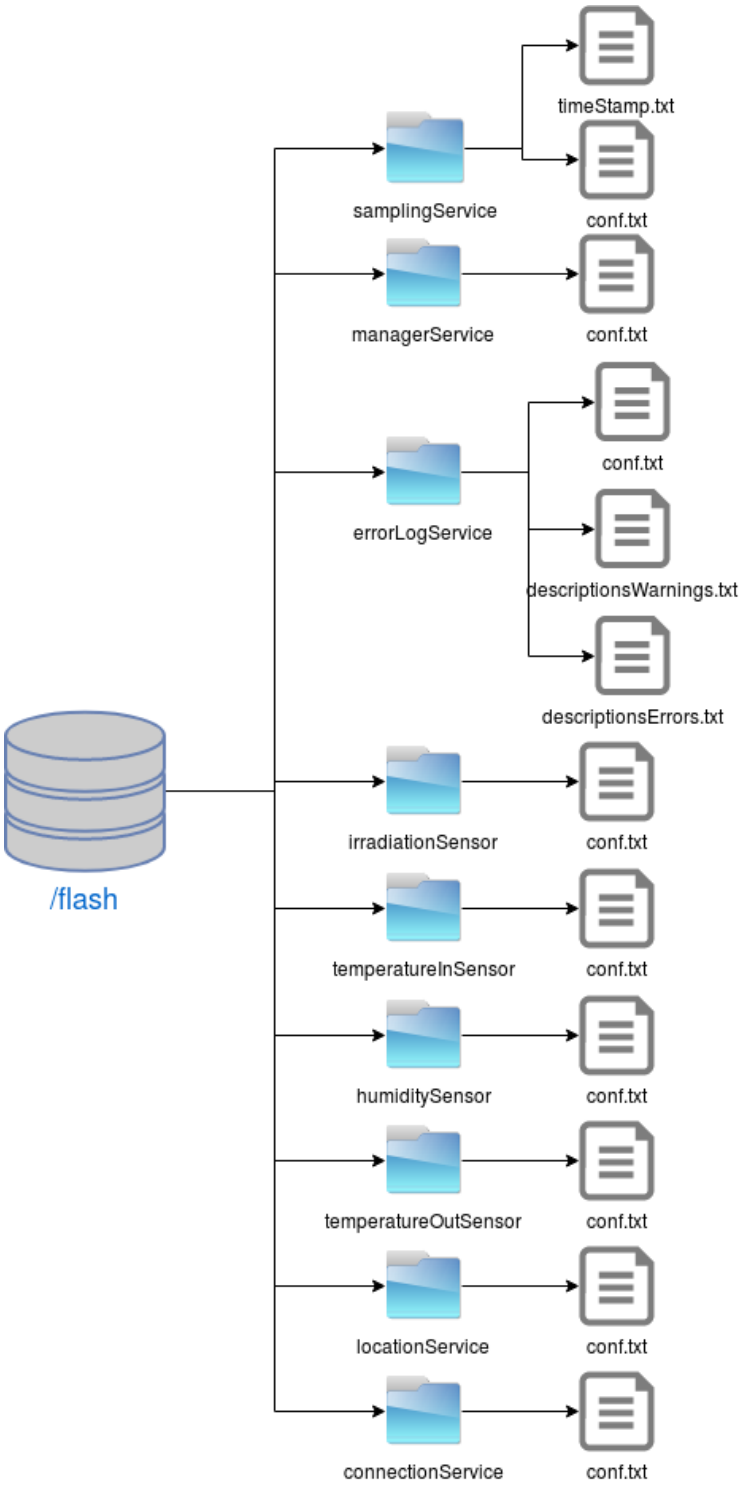

Figura 5: Diagrama General de Directorios

El código de la aplicación se encuentra en el firmware de la LoPy, donde cada servicio consta de un directorio propio en el que se almacena el fichero de código correspondiente al servicio, adicionalmente también existe un directorio llamado *libraries* para las distintas bibliotecas utilizadas en los servicios y un directorio especial para el *dht22Sensor*.

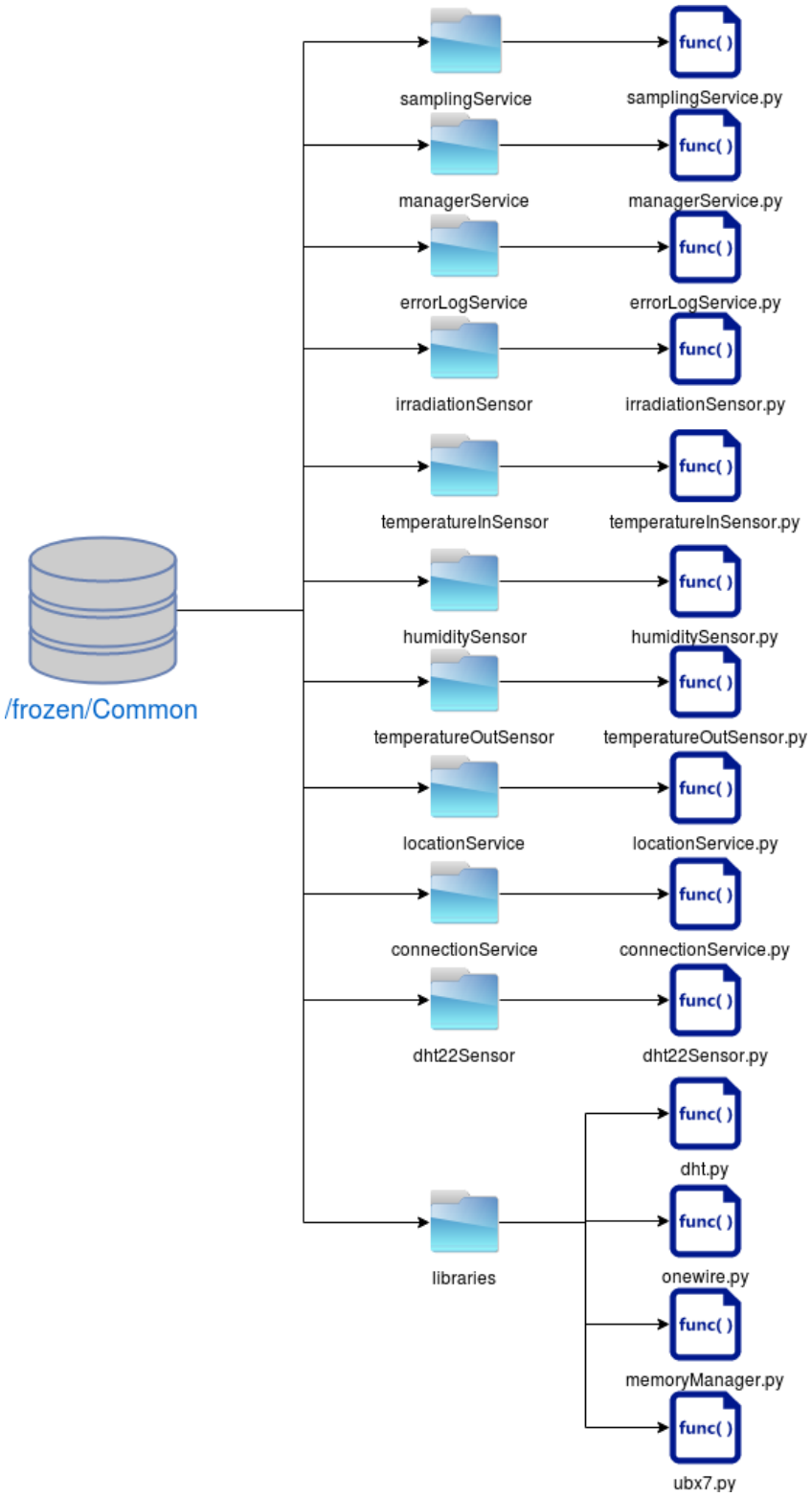

Figura 6: Diagrama General de Estructura Software

3.2. Configuración y muestreo del nodo solar

Los nodos solares al inicio de su ejecución constan de dos funcionalidades principales. Activación y configuración de todos los servicios que el fichero de configuración de *Managerservice* establezca que tienen que estar activos.

Como se puede apreciar en la figura 7, que representa el diagrama de secuencia de configuración inicial de un nodo, *ManagerService* es el servicio encargado de la activación del resto de servicios de la aplicación. Inicialmente lee su propio fichero de configuración el cual le indica qué servicios tiene que activar. Después, activa los tres servicios que se consideran imprescindibles y su activación es prioritaria: ConnectionService (encargado de la conexión LoRa con un gateway cercano), *ErrorLogService* (encargado de la gestión de errores y warnings de la aplicación) y *LocationService* (cuya función es la de conseguir una sincronización con el GPS para establecer la hora en el sistema). Posteriormente se pasa a la activación de los distintos sensores pasándole a cada uno de ellos sus atributos correspondientes. En cada sensor se creará un hilo nuevo de ejecución cuya tarea será la de ir recogiendo muestras a la frecuencia establecida. En el caso de *HumiditySensor* y *TemperatureInSensor*, activarán a su vez el servicio *DHT22Sensor* que será el encargado de recoger las muestras para estos dos servicios. Debido a esta restricción la frecuencia de muestreo de estos dos sensores tiene que ser la misma.

Finalmente, se activará el servicio *SamplingService* en el cual está el bucle principal de muestreo que irá solicitando las muestras a los sensores en función de la frecuencia de envío establecida.

La funcionalidad principal que tiene que realizar los nodos solares es la de realizar muestras de los sensores a las frecuencias establecidas en cada sensor y enviarlas a un gateway a una frecuencia denominada **sendingFrequency** también establecida.

Por otro lado, la figura 8 ilustra el bucle principal de recogida de muestras en *SamplingService*. Para este ejemplo, asumimos que los sensores están previamente activados y muestreando a las frecuencias seleccionadas en sus ficheros de configuración. Desde *SamplingService* se va solicitando la muestra de los sensores que están activos; la muestra que devuelvan los sensores puede ser de dos tipos: la media de todas las muestras que ha tomado entre peticiones del *SamplingService* o la última muestra realizada, dependiendo del modo que esté configurado el sensor en su fichero de configuración. Cuando se han solicitado todas las muestras que se desean enviar en un momento dado, *SamplingService* realiza una llamada al método *sendPackage* de *ConnectionService* para construir el paquete de muestras y realizar el posterior envío al gateway vía conexión LoRa.

Hemos diseñado un formato para dichos paquetes, tratando de minimizar el número de bits requeridos para el envío de las muestras. Asimismo, hemos diseñado el protocolo de comunicación nodo-gateway de modo que este último sepa cómo reaccionar ante los diferentes paquetes que recibe.

El bucle se repetirá a la frecuencia establecida mientras que el servicio *SamplingService* esté activo.

3.3. Servicios generales

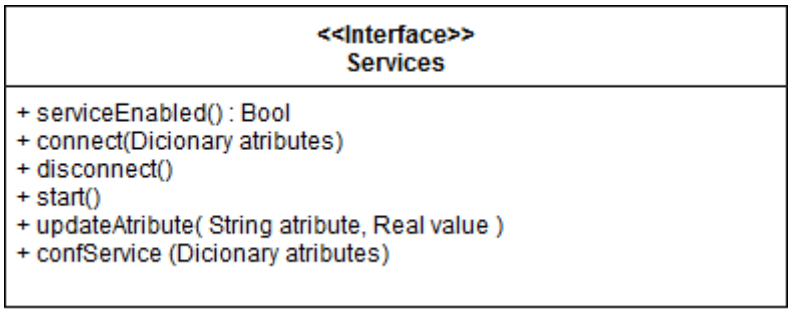

Figura 9: Estructura de la Interfaz Services

Para los servicios generales que otorgan funcionalidades de gestión, disponemos de una interface llamada *Services* que establecen los métodos que todo servicio que denominamos general debe cumplir. Los servicios tienen que activarse a través del método *connect()*, pudiendo desactivarlos con *disconnect()*. Para la configuración inicial de atributos se utilizará el método *confService()*; asimismo, los servicios dispondrán de atributos configurables vía LoRa o wifi a través ManagerService haciendo uso del método *updateAtribute()* de los servicios. El método *serviceEnabled* informará de si el servicio está activado. Por último, el método *start()* llevará acabado la funcionalidad principal del servicio, creando un hilo de ejecución nuevo para ello si fuese necesario.

3.3.1. ManagerService

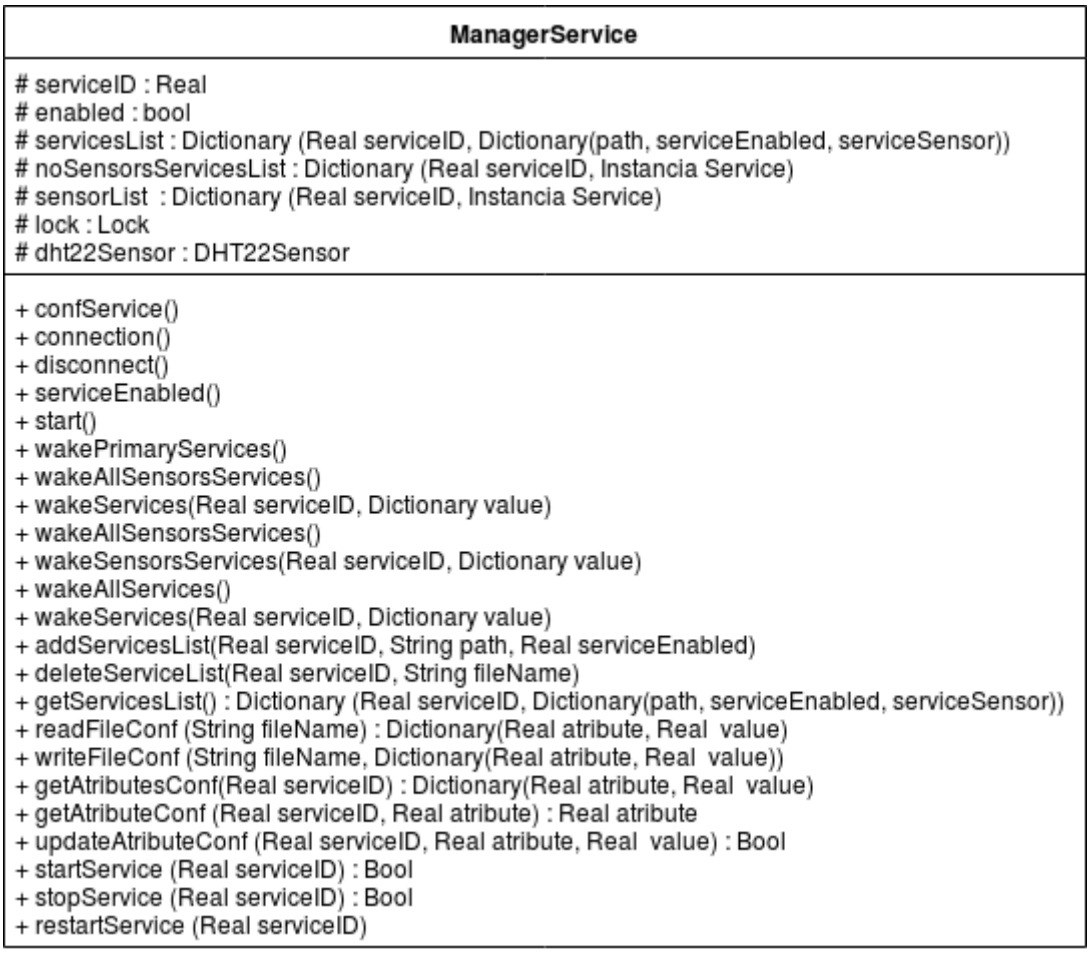

Figura 10: Estructura de ManagerService

ManagerService es el servicio principal de la aplicación, encargado de gestionar el resto de servicios: desde su activación y configuración hasta ofrecer las funciones necesarias para modificar los servicios de modo remoto. Sus atributos más significativos son **servicesList** que almacena la información relacionada con los servicios del sistema (si tiene que activarse, qué tipo de servicio es y la ruta de su fichero de configuración), **sensorsList** diccionario que almacena las instancias de los sensores activos y **noSensorsServicesList** diccionario que contiene las instancias de todos los servicios que no son sensores. El fichero de configuración de *ManagerService* contiene el **serviceID** y la información de los servicios mencionada anteriormente, de cada uno de los servicios que conforman la aplicación.

3.3.2. LocationService

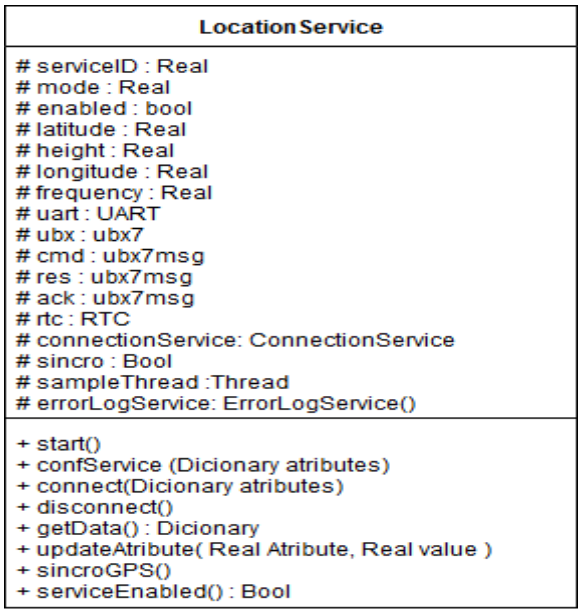

Figura 11: Estructura de LocationService

En el despliegue de la red es importante la necesidad de que todos los nodos tengan una marca temporal actual. En este proyecto para conseguir tal propósito utilizamos una referencia temporal a través del sistema global de navegación por satélite (GNSS), por este motivo utilizamos el servicio *LocationService* para realizar la comunicación con el satélite de posicionamiento.

El nodo envía un mensaje del tipo *"NAV-PVT"*, mediante el protocolo *"ubx7"*, al satélite, realizando éste una respuesta que contiene la posición, la velocidad y la fecha y hora actual, de los cuales utilizaremos la posición geográfica del nodo solar para el posterior envío de su posición a un gateway. La fecha y hora actual recibidas sincronizan el RTC interno de la placa LoPy, pues al no disponer de un RTC externo que proporcione una mayor garantía de una referencia temporal en el nodo solar esto se hace necesario, y con ello poder establecer la fecha en el sistema del nodo solar y notificar de la hora del sistema al gateway. Dado que esta tarea puede prolongarse un tiempo indefinido (varios minutos) e incluso terminar fallando, para realizar esa función se crea un hilo de ejecución que evita bloquear al resto del sistema en caso de que no se consiga una conexión con el satélite.

Este servicio consta de algunos atributos que se pueden modificar mediante el fichero de configuración o de forma remota, los atributos en cuestión son **frequency** que establece la frecuencia con la que se realizan los intentos de sincronización entre el nodo y el satélite de posicionamiento en caso de suceder algún error en la comunicación; **mode** que indica el modo en el que se realiza la comunicación con el satélite (con valor "0" se solicitan fix hasta recibir uno correcto).

3.3.3. SamplingService

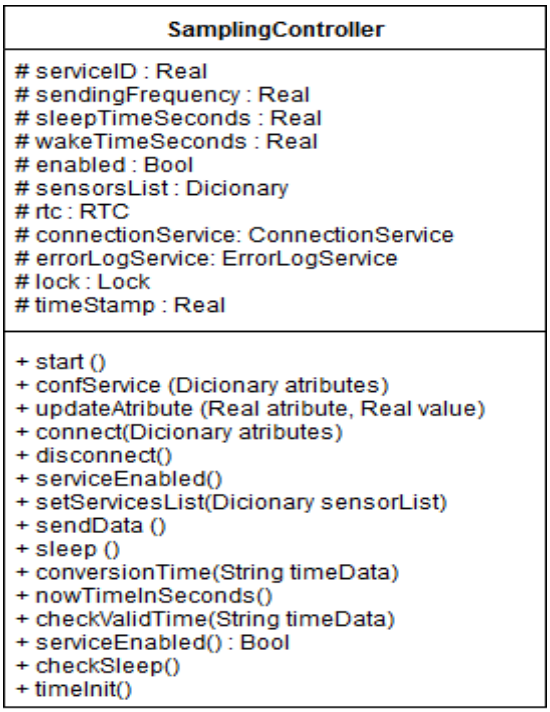

Figura 12: Estructura de SamplingService

SamplingService es un servicio con dos funciones, la primera es la de solicitar las muestras que se van a enviar a los sensores que están activos para su posterior envío a través de *ConnectionService* a un gateway LoRa. Para ello tiene una lista con las instancias de los servicios que se encuentran activos en cada momento de la ejecución de la aplicación. Irá recorriendo dicha lista periódicamente solicitando las muestras y construyendo la trama de envío. El período de ese recorrido es configurable y depende de la granularidad temporal deseada para la red en su conjunto.

La otra función que lleva a cabo es la de controlar a qué hora tiene que entrar en modo *"deepsleep"* el nodo para ahorrar energía en las horas que no se requiera realizar muestras debido a que no haya luz solar disponible. Para eso tiene dos atributos, **wakeTimeSeconds** y **sleepTimeSeconds,** que indican las horas del día (en segundos desde las 00:00 de ese dia) a la que se tiene que despertar y dormir el nodo solar respectivamente.

El servicio dispone de atributos que se pueden modificar mediante su fichero de configuración o de forma remota. El atributo **sendingFrequency** establece la frecuencia con la que se recogen y envían las muestras de los distintos sensores activos; **wakeTime** indica la hora a la que el nodo tiene que despertarse todos los días y **sleepTime** indica la hora a la que el nodo tiene que entrar en modo *"deepsleep"* para ahorrar energía y dejar de realizar muestras.

3.3.4. ErrorLogService

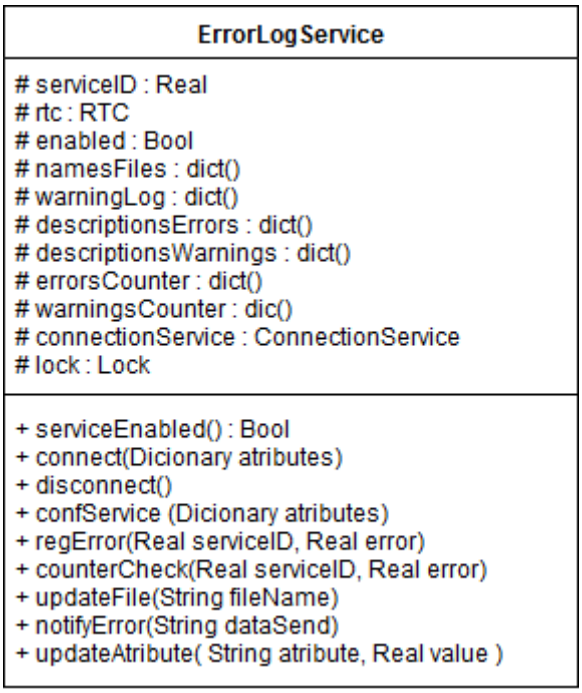

Figura 13: Estructura de ErrorLogService

ErrorLogService es el servicio que gestiona los distintos errores de modo que informa de ello al servidor a través de *ConnectionService* y también añade la posibilidad de almacenar los errores producidos en ficheros de log. Los distintos errores que se puedan producir a lo largo de la ejecución de la aplicación están divididos en errores y warnings, clasificados en función de la gravedad tal y como se puede consultar en el Anexo 1, incluido al final de este documento. Consideramos errores los sucesos que bloquean totalmente la aplicación y warnings los incidentes que permiten continuar la ejecución, pero es aconsejable informar de lo acontecido. Los warnings incluyen un número de veces, modificable a través de un fichero de configuración, que pueden suceder antes de informar al servidor, y a su vez habilita la posibilidad de ir almacenando en un fichero interno los fallos que se hayan ido produciendo.

El fichero de configuración de este servicio incluye su **serviceID**, las rutas de los ficheros de escritura de logs de errores y warnings, y las rutas que contienen la descripción de todos los errores y warnings contemplados.

3.4. Servicios de red

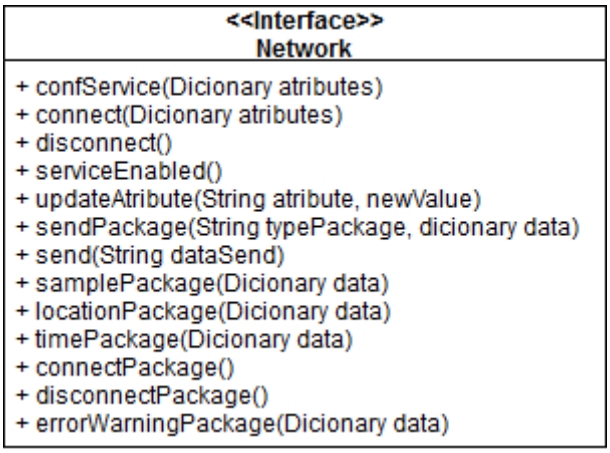

Figura 14: Estructura de Interfaz Network

Para servicios de red diseñamos una interfaz que denominamos *Network* que establece las funciones necesarias para crear un servicio de red en la aplicación, en nuestro proyecto hemos desarrollado el servicio *ConnectionService* que es el encargado de la conexión con el exterior vía LoRa. En la interface se establecen los métodos de configuración, conexión y desconexión del servicio, las funciones encargadas de generar los diferentes mensajes del protocolo de comunicación con el gateway y habilita la posible modificación de ciertos atributos de forma remota o utilizando un fichero de configuración.

3.4.1. ConnectionService

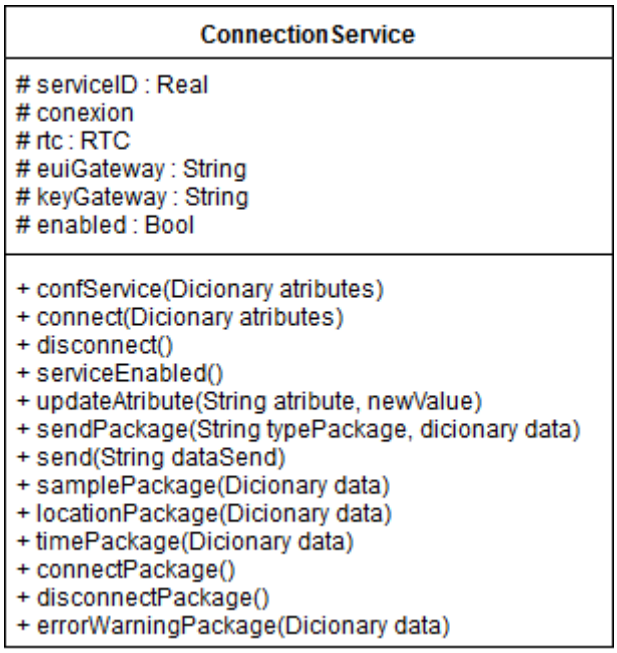

Figura 15: Estructura de ConnectionService

El servicio *ConnectionService* se encarga de la configuración y conexión del nodo solar con un gateway mediante LoRa, implementa la interfaz Network, entre sus funciones también destaca la creación del protocolo de salida de mensajes que hemos diseñado para nuestro proyecto y que detallaremos más adelante. El servicio dispone de varios atributos entre los que se encuentran **euiGateway** y **keyGateway** que establecen las claves necesarias para establecer conexión con un gateway determinado, estos atributos están disponibles para una modificación mediante un fichero de configuración y de forma remota. El fichero de configuración del servicio está formado por el **serviceID** del servicio y los atributos **euiGateway** y **kaygateway** ya mencionados.

3.5. Servicios de sensores

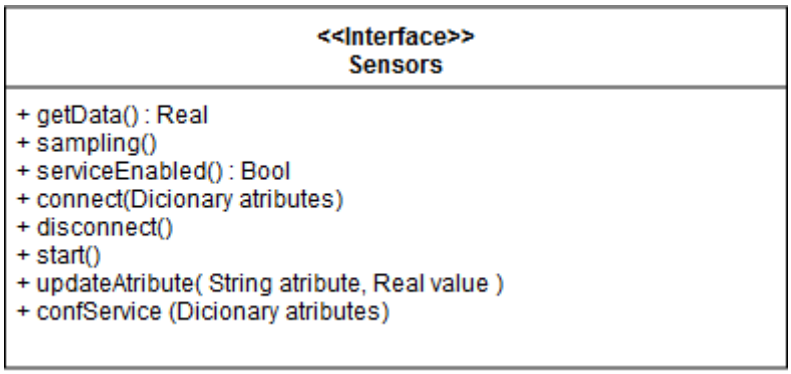

Figura 16: Estructura de Interfaz Sensors

Para los servicios que son sensores diseñamos una interfaz llamada *Sensors* que establece los métodos necesarios para la creación de un nuevo sensor, por cada instancia del sensor se creará un hilo encargado del muestreo y posible filtrado de los datos obtenidos.

Para la activación, desactivación y configuración del sensor se utilizan los métodos *connect()*, *disconnect()* y *confservice()* respectivamente, *updateAtribute()* actualiza los atributos modificables del sensor, *start()* crea un nuevo hilo de ejecución con el método *sampling()* que será el encargado de realizar las muestras a una frecuencia establecida en el sensor y con *getData()* se obtiene el valor de las muestras.

3.5.1. IrradiationSensor

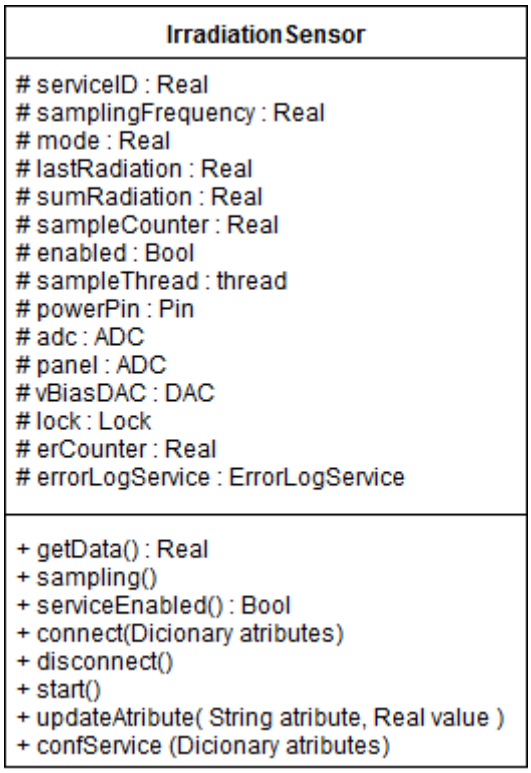

Figura 17: Estructura de IrradiationSensor

Irradiationsensor es el servicio a cargo del sensor de radiación solar, realiza mediciones periódicas utilizando la librería *"ADC"* proporcionada por Pycom, a la frecuencia establecida, ofreciendo los datos de dos formas distintas dependiendo del valor de su atributo **mode** asignado en su fichero de configuración correspondiente, pudiendo devolver la media de las últimas mediciones o la última medición realizada (ambas posibilidades quedan habilitadas para todos los sensores disponibles). El valor devuelto por este sensor es una lectura de una de las posibles combinaciones del rango del ADC, en nuestro caso en forma de voltaje, que servirá para conocer con mayor precisión la radiación capturada por el sensor dedicado a ello, y que permitirá en el futuro trabajar con datos más precisos en el modelo predictivo.

Los atributos **samplingFrequency**, que indica la frecuencia con la que se realizan muestras, y **mode**, que establece el modo en que se devuelven las muestras "0" para la media de las muestras y "1" para la última muestra, pueden ser modificados a través del fichero de configuración o de forma remota.

3.5.2. DHT22Sensor

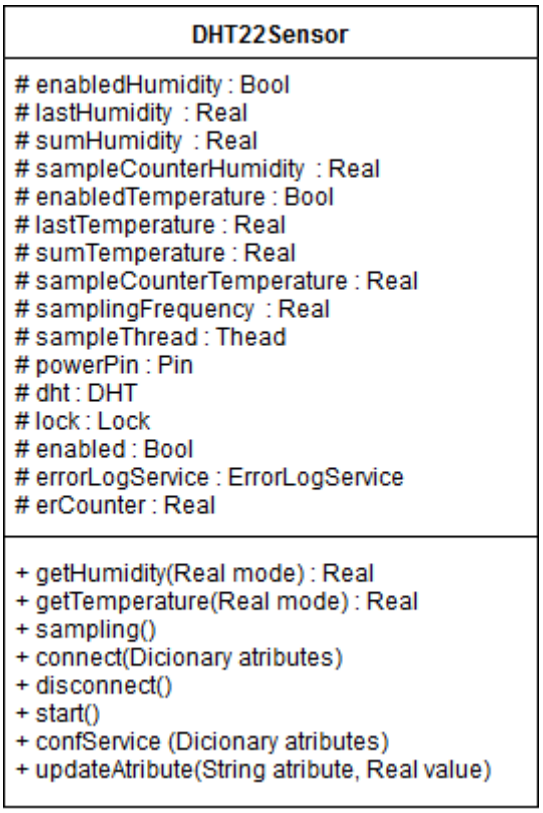

Figura 18: Estructura de DHT22Sensor

El servicio *DHT22Sensor* es el sensor encargado de recoger las muestras de humedad y de temperatura interna del nodo. Este sensor tiene la particularidad de que realiza la recogida de muestras de dos mediciones a través de la creación de un hilo nuevo de ejecución, con el que se realiza el muestreo a la frecuencia establecida, es por ello que la frecuencia de muestreo de la humedad y de la temperatura interior tiene que ser la misma. Este sensor no tiene un fichero de configuración, pero ofrece la opción de modificar la frecuencia de muestreo a través de los servicios *HumiditySensor* y *TemperatureInSensor* que son los encargados de activar y establecer dicha frecuencia en *DHT22Sensor*.

3.5.3. HumiditySensor

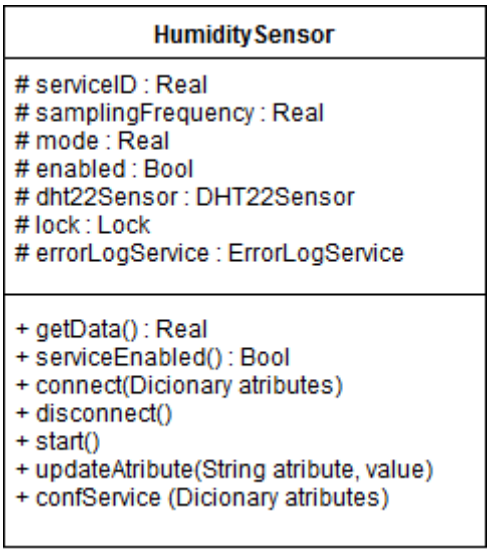

Figura 19: Estructura de HumiditySensor

HumiditySensor es el servicio encargado del sensor de humedad e implementa la interfaz Sensors, en este servicio cabe destacar que no es el encargado de realizar las mediciones, sino que tiene que activar el servicio *DHT22Sensor* para crear un hilo de ejecución y realizar las mediciones apropiadas.

Los atributos **samplingFrequency**, que indica la frecuencia con la que se realizan muestras, y **mode**, que establece el modo en que se devuelven las muestras “0” para la media de las muestras y “1” para la última muestra, pueden ser modificados a través del fichero de configuración o de forma remota.

3.5.4. TemperatureInSensor

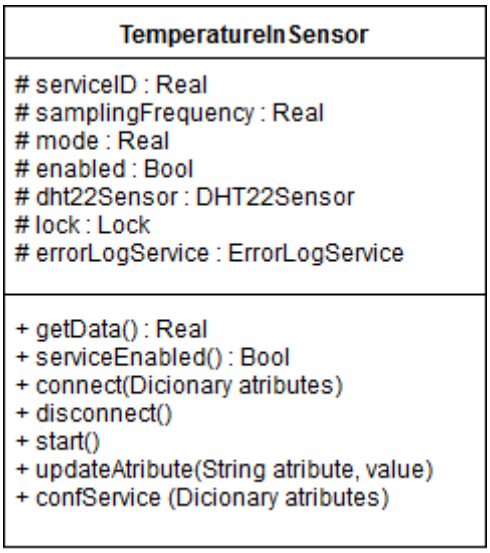

Figura 20: Estructura de TemperatureInSensor

El servicio *TemperatureInSensor* gestiona la métrica de temperatura interior del nodo, aunque para realizar las muestras necesita activar el sensor *DHT22Sensor* para crear un hilo de ejecución y realizar el muestreo a la frecuencia que *TemperatureInSensor* le indique.

Los atributos **samplingFrequency**, que indica la frecuencia con la que se realizan muestras, y **mode**, que establece el modo en que se devuelven las muestras “0” para la media de las muestras y “1” para la última muestra, pueden ser modificados a través del fichero de configuración o de forma remota.

3.5.5. TemperatureOutSensor

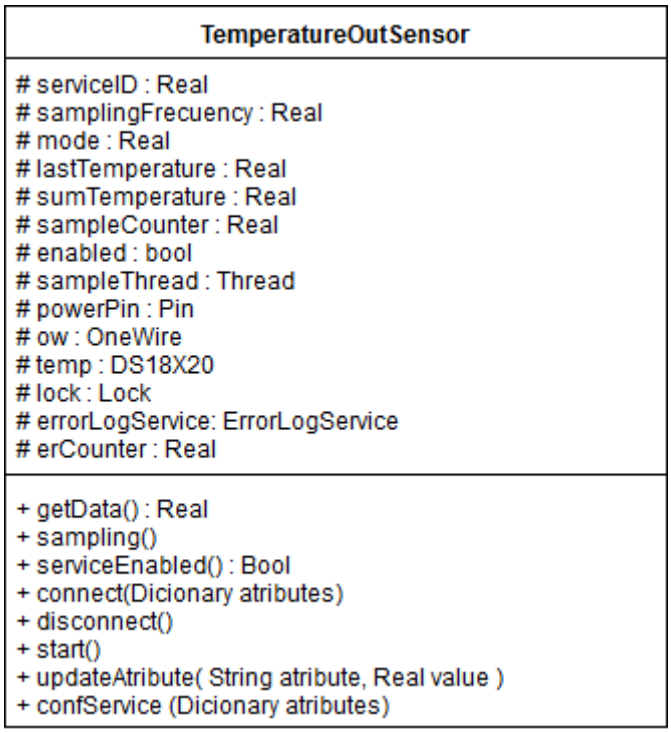

Figura 21: Estructura de TemperatureOutSensor

TemperatureOutSensor es el sensor encargado de realizar las muestras de temperatura exterior del nodo. Crea un hilo de ejecución en el que se realizan las muestras a la frecuencia establecida y como ya se mencionó en la sección 2.4. del Capítulo 2, este sensor utiliza el protocolo *OneWire* para la transmisión de datos.

Los atributos **samplingFrequency**, que indica la frecuencia con la que se realizan muestras, y **mode**, que establece el modo en que se devuelven las muestras “0” para la media de las muestras y “1” para la última muestra, pueden ser modificados a través del fichero de configuración o de forma remota.

3.6. Limitaciones de desarrollo

Como ya quedó reflejado en el Capítulo 2, en el contexto del bajo consumo es habitual encontrarse con ciertas dificultades debido a los recursos limitados del sistema. Debido a ello, reseñamos a continuación cuáles han sido las limitaciones de desarrollo más relevantes que nos hemos encontrado en este proyecto.

3.6.1. Consumo excesivo de memoria

El dispositivo hardware utilizado dispone de 512KB de memoria. Tras instalarle el firmware proporcionado por PyCom quedan únicamente 79152 bytes libres[19] aproximadamente. A su vez, parte de esos bytes son ocupados posteriormente por el intérprete de MicroPython necesario para ejecutar el código.

Tras ir añadiendo módulo a módulo los servicios disponibles y la creación de los distintos hilos de ejecución necesarios para la recogida de muestras de los distintos sensores, la cantidad de memoria RAM iba disminuyendo de forma considerable hasta llegar el punto de que el nodo consumía toda la memoria ejecutando la aplicación, lo que bloqueaba su correcto funcionamiento antes de poder añadir incluso todos los módulos disponibles.

Para tratar de paliar este problema, incluimos explícitamente en nuestro cóidgo llamadas que obligan a actuar al Recolector de Basura[20] de MicroPython. Con el fin de mitigar el uso excesivo de estas llamadas establecemos un límite de 13000 bytes libres de memoria para dicha actuación; el código está incluido en la librería *"memoryManager"* de la aplicación.

A pesar de esto, al agregar nuevos módulos seguía apareciendo el mismo problema. Este conflicto resultó ser el más importante del proyecto y finalmente pudo ser solucionado incluyendo todo el código fuente de la aplicación en el firmware de la LoPy a través de un proceso denominado *"Frozen Bytecode"*, liberando así gran cantidad de memoria RAM en cada ejecución del programa y permitiendo añadir todos los módulos citados anteriormente. Esta solución queda más detallada en el Anexo 2 incluido al final de este trabajo.

[19] https://forum.pycom.io/topic/170/lopy-wipy-2-esp32-specs-and-performance/9
[20] http://docs.micropython.org/en/v1.9.3/pyboard/library/gc.html

3.6.2. RTC

Debido a la pérdida de fiabilidad del RTC interno de la placa LoPy cuando entra en modo *deepsleep*, el nodo solar podría no iniciar a la hora esperada, por lo que al inicio de la ejecución se realiza una comprobación para considerar si tiene que volver a dormirse porque se inició antes de lo esperado. Esta comprobación consiste en leer un fichero almacenado en la memoria flash de la placa LoPy llamado *"timeStamp.txt"* y que contiene dos posibles valores: la marca temporal en segundos de cuando se tiene que iniciar el nodo o un "1" que señala que el nodo tiene que estar ejecutando la aplicación software. El nodo antes de entrar en modo *deepsleep* guarda en el fichero *"timeStamp.txt"* la marca temporal a la que se tiene que volver a iniciar.

3.6.3. Sensor DHT

El sensor DHT22 realiza los muestreos de humedad y temperatura interior del nodo, al ser dos métricas distintas las que recoge un sólo sensor, surgió el problema de realizar dos accesos desde distintos hilos al sensor DHT22 lo que resultaba en valores erróneos, por lo que se optó por crear un único hilo de ejecución para la realización de los dos muestreos.

El servicio *DHT22Sensor* es el encargado de realizar esta tarea.

3.7. Análisis de tiempos de ejecución

A la hora de enviar los datos recogidos por la aplicación hacia un Gateway, nos dimos cuenta de que los envío tenían un ligero retraso temporal en la frecuencia de envío deseada, del que no conocíamos su origen.

Para ello, hicimos algunos cálculos con la aplicación para conocer realmente que tiempos de ejecución estábamos manejando.

En primer lugar, decidimos calcular el tiempo de ejecución del proceso de recogida de muestras de los sensores, para ver si el delay producido era fruto de la recogida de muestras en los sensores. Obtuvimos los tiempos que se reflejan en la Tabla 1.

Sensores Activos	Envío 5 segs - Muestreo 1seg	Envío 10 segs - Muestreo 1seg	Envío 10 segs - Muestreo 2seg	Envío 20 segs - Muestreo 5seg
Todos	3,3627	3,4594	0,7017	0,0019
Irra+TempIn+Hum	0,8413	0,2639	0,1135	0,0015
Irra+TempIn/Hum+TempOut	2,4771	2,6030	0,7043	0,0018
TempIn+Hum+TempOut	3,1078	2,5299	0,6887	0,0018
Irra+TempOut	0,5403	0,3576	0,0782	0,0672
Irra+TempIn/Hum	0,8465	0,2747	0,1853	0,0009
TempIn/Hum+TempOut	2,2667	1,6484	0,1108	0,0006
Irradiation	0,0003	0,0004	0,0003	0,0003
TemperatureOut	0,5194	0,3050	0,1230	0,0508
TemperatureIn/Humidity	0,7904	0,1733	0,2583	0,0003

Tabla 1: Cálculo de Tiempos Bucle de Muestreo

Como se puede apreciar, sí que existe un ligero retraso en la recogida de estas muestras cuando tanto la frecuencia de envío como la de muestreo son altas (envíos cada poco tiempo), aunque solo se ve reflejado en algunos casos, como por ejemplo cuando están todos los sensores activos para frecuencias de envío de 5-10 segundos y frecuencia de muestreo de 1 segundo. Sin embargo, vemos como para frecuencias de envío de 10-20 segundos y frecuencias de muestreo de 2-5 segundos, el tiempo de la ejecución es apenas perceptible pues no llega al segundo en ninguno de los casos.

Estos datos no aclararon demasiado el problema, pues tan sólo en unos pocos casos, vemos que se vea afectada en más de un segundo la ejecución esperada de la aplicación. Por ello, decidimos ir más allá y realizamos los cálculos del tiempo total transcurrido en el proceso de recogida de muestras por los sensores y con la creación de los paquetes que se envían, incluyendo también su envío hacia el Gateway.

Los resultados obtenidos se pueden observar en la Tabla 2, incluida a continuación.

Sensores Activos	Envío 5 segs - Muestreo 1seg	Envío 10 segs - Muestreo 1seg	Envío 10 segs - Muestreo 2seg	Envío 20 segs - Muestreo 5seg
Todos	11,4675	16,5037	13,7272	23,0318
Irra+TempIn+Hum	8,8662	13,2971	13,1489	23,0282
Irra+TempIn/Hum+TempOut	10,4999	15,6207	13,7416	23,0308
TempIn+Hum+TempOut	11,0102	15,5675	13,7636	23,0302
Irra+TempOut	8,2584	13,3762	13,1030	23,0874
Irra+TempIn/Hum	8,7460	13,2895	13,1478	23,0235
TempIn/Hum+TempOut	10,1152	14,6984	13,7512	23,0245
Irradiation	10,1152	13,0221	13,0215	23,0215
TemperatureOut	8,0176	13,3238	13,1430	23,0721
TemperatureIn/Humidity	8,7466	13,1920	13,2783	23,0213

Tabla 2: Cálculo de Tiempo Total de Muestreo y Envío a Frecuencia Deseada

Tras esto, llegamos a la conclusión de que todos los tiempos estaban retrasados en aproximadamente 3 segundos a lo esperado, por ejemplo, para los casos de frecuencia de envío de 20 segundos el envío se realiza cada 23 segundos aproximadamente.

Sin embargo, sí que seguimos observando que en períodos de envío de 5-10 segundos, con frecuencia de muestreo de 1 segundo, algunos envíos se retrasan más de 3 segundos, pero casualmente, se tratan de los mismos casos en los que el proceso de muestreo también manejaba un pequeño delay.

Tras analizar y valorar todos los datos obtenidos, llegamos a la conclusión de que el proceso de creación del paquete a enviar y el envío del mismo, retrasaban en 3 segundos (generalmente) la ejecución normal de la aplicación.

Para compensar este delay, la aplicación en su versión final incluye una resta de este tiempo transcurrido en la creación y envío del paquete de datos a la frecuencia de envío esperada, lo que nos permite ajustar de manera más precisa los tiempos de envío y obtener los datos en el Gateway a la frecuencia aproximada esperada.

Capítulo 4: Servidores centrales

En la comunicación de la red de nodos con el exterior utilizamos un gateway que establece conexión con los dispositivos a través de conexión LoRa. Posteriormente desde el gateway se publicará la información recibida de todos los nodos a un broker MQTT usando protocolo MQTT. Dicha información puede ser el envío periódico de muestras realizadas para su posterior almacenamiento en el servidor *"solarcasting.dacya.ucm.es"*, mensajes para alertar de un posible error en la ejecución de la aplicación software en un nodo solar, mensajes para dar de alta o dar de baja un nodo del sistema de seguimiento ThingsBoard[21], informar de la posición actual del nodo o informar de la hora del sistema en el nodo.

El proceso de comunicación entre nodo, broker MQTT y cliente consta en primer lugar del envío de un mensaje para dar de alta al nodo solar en la plataforma ThingsBoard[22]. Tras esto, se procede al envío de la posición geográfica y la hora actual del sistema del dispositivo. A continuación, se realiza la transmisión periódica de muestras, dependiente de la frecuencia de envío establecida previamente, desde el nodo al broker MQTT, que hace accesible esta información para los distintos suscriptores, en nuestro proyecto utilizamos como cliente suscriptor la plataforma ThingsBoard.

De carácter extraordinario el nodo puede realizar el envío de notificaciones de posibles errores y warnings producidos durante la ejecución de la aplicación software, y adicionalmente existe la posibilidad de enviar una petición de dar de al nodo en ThingsBoard.

4.1. Protocolo MQTT

MQTT[23] es un protocolo de mensajería de tipo *publish/subscribe*, extremadamente simple y liviano, diseñado para redes de bajo ancho de banda, de alta latencia o poco confiables. Los principios de diseño son minimizar el ancho de banda de la red y los requisitos de recursos del dispositivo al mismo tiempo que intentan garantizar la confiabilidad y cierto grado de seguridad de la entrega.

La arquitectura MQTT tiene una topología en estrella, con un servidor central denominado broker. Los clientes pueden darse de alta en el broker cómo *publish* (publican información en un canal del broker) o cómo *subscribe* (reciben información del canal al que están suscritos).

[21] https://thingsboard.io/
[22] http://solarcasting.dacya.ucm.es:8080
[23] http://mqtt.org/

4.2. Plataforma ThingsBoard

ThingsBoard es una plataforma IoT de código abierto para la recopilación de datos, el procesamiento, la visualización y la gestión de dispositivos. El motivo de elección de esta plataforma es el poder disponer de una solución local en el servidor *"solarcasting.dacya.ucm.es",* que habilitará la infraestructura del lado del servidor para nuestro proyecto de IoT.

En esta plataforma se pueden dar de alta los nodos a través de publicaciones MQTT (o manualmente directamente en la plataforma online). Pudiéndose identificar cada uno de los dispositivos como puerta de entrada. También ofrece la posibilidad de crear un dispositivo especial denominado *"Gateway"*[24] que actuará como puente entre los nodos de la red y ThingsBoard. La API del *"Gateway"* proporciona la capacidad de intercambiar datos entre los nodos y la plataforma utilizando una sola conexión MQTT, resultando ser así la manera en la que se ha trabajado en este proyecto.

Cada dispositivo dado de alta dispone de atributos actualizables a través de un canal MQTT, que en este caso utilizamos para almacenar la información de su situación geográfica y temporal (hora a la que se inició el nodo en concreto), y además si sucediera algún error/warning en la ejecución de uno de los nodos también quedaría registrado aquí.

Adicionalmente, ThingsBoard hace uso de un canal MQTT donde recibe y quedan almacenados, en tiempo real, los datos telemétricos devueltos por los sensores disponibles de cada dispositivo de la red.

Thingsboard facilita, además, que estos atributos propios de cada nodo y su correspondiente telemetría almacenada puedan ser representados de diferentes maneras, como por ejemplo con el uso de gráficos o mapas, que pueden ser elegidos en función de los intereses de cada cliente.

Toda la información sobre cómo configurar una red a través de canales MQTT con ThingsBoard puede ser consultada junto con su API en su documentación oficial[25].

4.3. Interacción nodos-gateway

Para la comunicación del nodo solar y el gateway hemos diseñado dos protocolos de comunicación, uno para los mensajes de salida del nodo y otro para los mensajes de entrada. En nuestro proyecto sólo implementamos los mensajes de salida, pero ha quedado diseñado también un posible protocolo de entrada de mensajes desde el exterior del nodo.

[24] https://thingsboard.io/docs/reference/gateway-mqtt-api/
[25] https://thingsboard.io/docs/

4.3.1. Protocolo de entrada

Aunque no queda reflejado en la implementación de este proyecto, ha quedado diseñado un protocolo de entrada de mensajes desde el exterior, teniendo en cuenta las operaciones de actualizar un atributo concreto de un servicio, actualizar un servicio completo, activar o detener un servicio y consultar la batería del nodo.

El esquema general del protocolo de entrada es el siguiente:

ID Nodo	Operación	Servicio	Datos

Figura 22: Estructura General Protocolo de Entrada de Mensajes

Se diferencian los siguientes campos:

- ID Nodo: Que debe incluir el identificador único del nodo en el que se quiere realizar una operación.
- Operación: Campo que debe incluir el tipo de operación. Entre los que se contemplan: Actualizar atributo de un servicio (0), Actualizar servicio completo (1), Activar un servicio (2), Detener un servicio (3) y Consultar batería (4).
- Servicio: Que debe indicar el identificador único del servicio en el que se quiere realizar la operación.
- Datos: Campo que varía su valor en función del tipo de operación.

Cada operación puede disponer o no de los campos de Servicio y Datos, dependiendo de cada tipo de operación. Los campos ID Nodo y Datos pueden tener un tamaño variable. Los campos Operación y Servicio constan, a su vez, de 1 byte de tamaño cada uno.

Los servicios disponibles para este proyecto son: *SamplingService* (1), *LocationService* (2), *IrradiationSensor* (3), *TemperatureInSensor* (4), *HumiditySensor* (5), *TemperatureOutSensor* (6), *ConnectionService* (7) y *ErrorLogService* (8). Para el servicio *ManagerService* (0), sólo están disponibles las operaciones de Activar servicio y Detener servicio, debido a que no dispone de atributos modificables.

El tipo de operación *"Actualizar un atributo de un servicio"*, tiene la estructura que se muestra en la figura 23.

ID Nodo	Operación	Servicio	Atributo	Valor

Figura 23: Estructura Operación Actualizar Atributo

Esta operación permite actualizar un atributo concreto de un determinado servicio, cada servicio consta de varios atributos actualizables, que listamos a continuación:

- El servicio *SamplingService* (1) cuenta con:
    - serviceID (1)
    - sendingFrequency (2)
    - sleepTime (3)
    - wakeTime (4)

- El servicio *LocationService* (2) dispone de:
    - serviceID (1)
    - frequency (2)
    - mode (3)
- Los servicios sensores *IrradiationSensor* (3), *TemperatureInSensor* (4), *HumiditySensor* (5) y *TemperatureOutSensor* (6) dispone cada uno de:
    - serviceID (1)
    - samplingFrequency (2)
    - mode (3)
- El servicio *ConnectionService* (7) cuenta con:
    - serviceID (1)
    - euiGateway (2)
    - keyGateway (3)
- Por último, el servicio *ErrorLogService* (8) dispone de:
    - serviceID (1)
    - errorFile (2)
    - warningFile (3)
    - descriptionsErrorsFile (4)
    - descriptionsWarningsFile (5)

Se incluye entre paréntesis el código de 1 byte necesario para el campo *"Atributo"* que indica el atributo que se quiere actualizar de cada servicio.

La siguiente operación disponible es la de *"Actualizar servicio completo"*, que debe contener en su campo *"Datos"* el nuevo fichero de configuración que actualizará todo un servicio. Siguiendo la organización que se aprecia en la figura 24.

ID Nodo	Operación	Servicio	Fichero nuevo

Figura 24: Estructura Operación Actualizar Servicio

Las operaciones de *"Activar o Detener un servicio"*, mantienen la misma estructura, en la que sólo disponen de los campos *"ID Nodo"*, *"Operación"* y *"Servicio"* tal y como muestra la figura 25.

ID Nodo	Operación	Servicio

Figura 25: Estructura Operación Activar/Detener Servicio

Por último, está disponible la operación *"Consultar Batería"*, que nos permite conocer la batería de un nodo en concreto por lo que solo dispone de los campos *"ID Nodo"* y *"Operación"*, tal y como muestra la figura 26.

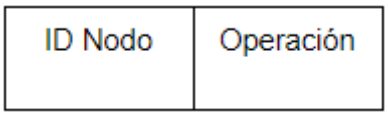

Figura 26: Estructura Operación Consultar Batería

4.3.2. Protocolo de salida

Este proyecto implementa un protocolo de salida de diseño propio que permite el envío de distintos tipos de mensajes formados por una cadena de caracteres separando los campos por espacios en blanco. Entre los tipos de mensaje se incluyen el envío de las muestras recogidas por los sensores, el envío de la posición en la que se encuentra el dispositivo, el envío de la hora de puesta en marcha de un nodo, el envío de peticiones de conexión y desconexión de un nodo y el envío de mensajes de error/warning que hayan podido suceder a lo largo de la ejecución de la aplicación de un nodo. El protocolo está diseñado como una cadena de caracteres, donde los campos están separados por un espacio en blanco.

El esquema general del protocolo de salida es el siguiente:

Figura 27: Estructura General Protocolo de Salida de Mensajes

Se diferencian los siguientes campos:

- Hora: Incluye la hora concreta a la que se envía el mensaje
- Minuto: Contiene el minuto concreto en que se envía el mensaje.
- Segundo: Debe incluir el segundo exacto en que se envía el mensaje.
- Tipo Mensaje: Campo que debe incluir el código que representa al tipo de mensaje de salida. Contemplamos 6 tipos de mensajes: *Sample* (0), *Location* (1), *Time* (2), *Conexión* (3), *Desconexión* (4) y *ErrorWarning* (5).
- Datos: Campo que varía su valor en función del tipo de operación.

Cada mensaje puede disponer o no del campo de Datos, dependiendo de cada tipo de mensaje. Los campos *"Hora"*, *"Minuto"* y *"Segundo"* tienen un tamaño de 2 bytes. El campo *"Tipo Mensaje"* consta de 1 byte de tamaño y, por último, el campo *"Datos"* tiene un tamaño variable en cada tipo de mensaje.

Los mensajes de *Sample* tienen la estructura que muestra la figura 28 El código que representa este tipo de mensaje es el 0. Los campos de *"Humedad"*, *"Temperatura Interna/Externa"* y *"Radiación"* indican si el mensaje envía la muestra de dicha medida, 1 si incluye la muestra y 0 si no incluye la muestra en el envío. El orden de las muestras lo marcan los campos *"Radiación"*, *"Temperatura Interna/Externa"* y *"Humedad"*, en caso de que alguna muestra no se envíe (exista un 0 en su respectivo campo) las demás se mostrarán siguiendo el orden mostrado en la figura 28, separadas siempre por espacios en blanco.

Hora	Minuto	Segundo	Tipo Mensaje	Radiación	Temperatura Interna	Humedad	Temperatura externa	Muestra Radiación	Muestra Temperatura interna	Muestra Humedad	Muestra Temperatura externa

Figura 28: Estructura Tipo Mensaje Sample

Los mensajes de *Location* son utilizados para notificar la posición geográfica de los nodos. Está formado por los campos que muestra la figura 29. El código que representa este tipo de mensaje es el 1. Los campos de *"Longitud"*, *"Latitud"* y *"Altitud"* representan las coordenadas del nodo que envía el mensaje.

Hora	Minuto	Segundo	Tipo Mensaje	Longitud	Latitud	Altitud

Figura 29: Estructura Tipo Mensaje Location

La notificación de la hora actual en el sistema se realiza a través del mensaje *Time* que tiene la disposición que se muestra en la figura 30. Los campos *"Hora inicial"*, *"Minuto inicial"* y *"Segundo Inicial"* indican los datos referentes a la hora, minuto y segundo a la que se inicia la ejecución de un nodo. Este mensaje se identifica en Tipo de mensaje con un 2.

Hora	Minuto	Segundo	Tipo Mensaje	Hora inicial	Minuto inicial	Segundo inicial

Figura 30: Estructura Tipo Mensaje Time

Para darse de alta o baja en la plataforma *ThingsBoard* se hace uso de los mensajes de *conexión* y *desconexión* respectivamente. En el campo *"Tipo de mensaje"* el mensaje de *conexión* se identifica con un 3 y el de *desconexión* con un 4. Estos mensajes tienen la misma estructura y no hacen uso del campo *"Datos"*, como se muestra en la figura 31:

Hora	Minuto	Segundo	Tipo Mensaje

Figura 31: Estructura Tipo Mensaje Conexión/Desconexión

Por último, para la notificación de errores y warnings durante la ejecución de la aplicación software en los nodos se utiliza la organización que muestra la figura 32, donde se notifica el tipo de error producido y el número de veces que se ha repetido a lo largo de la ejecución en el nodo. En el campo *"Tipo Mensaje"* se identifica con un 5.

Hora	Minuto	Segundo	Tipo Mensaje	Descripción del error	Número de veces que se ha producido el error

Figura 32: Estructura Tipo Mensaje ErrorWarning

4.4. Interacción gateway-broker MQTT

En la comunicación de los nodos con el servidor broker MQTT y posteriormente con la plataforma *ThingsBoard*, el gateway es un intermediario que realiza la tarea de recibir e interpretar los mensajes de los distintos nodos que conforman la red. Para la realización de este proyecto, dicho gateway está conectado vía Ethernet a la red del servidor, aunque también se dispone de su conexión por Wifi si quisiera llevarse a cabo.

Posteriormente, se publica un mensaje en un formato adecuado, siguiendo el protocolo de comunicación que dispone la API de *ThingsBoard*, en el broker MQTT.

Para el proceso de interpretación de los mensajes recibidos de los nodos y la posterior creación del mensaje que cumpla con el protocolo de *Thingsboard*, es necesario programar el gateway, ya que por defecto no realiza esta tarea, es por ello la necesidad de utilizar la herramienta Node-RED, que nos permite facilidad en la tarea de programación del gateway.

4.4.1. Herramienta Node-RED

Para el proceso de filtración e interpretación de la información recibida en el gateway desde el nodo solar utilizamos la herramienta Node-RED[26] instalada en el gateway, esta herramienta es open source y nos ofrece poder diseñar de manera sencilla una implementación del proceso de interpretación de la información recibida a través de los protocolos diseñados para la comunicación entre el nodo solar y el gateway, para su posterior envió a un servidor broker MQTT.

[26] https://nodered.org/

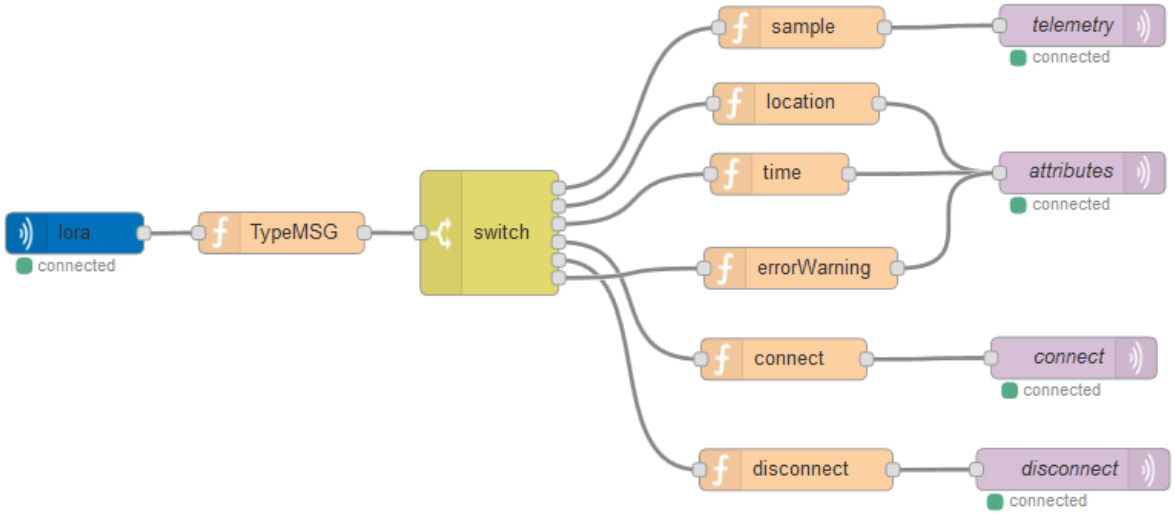

Figura 33: Configuración de Node-RED en Gateway

El diseño consta de varios módulos como se puede apreciar en la figura 33, el primer módulo *"lora"* es el encargado de la recepción de paquetes vía conexión LoRa, el segundo módulo *"typeMSG"* identifica el tipo de mensaje, cada mensaje es tratado de una forma diferente, por lo que se crea un módulo nuevo por cada tipo de mensaje.

El módulo *"sample"* realiza la interpretación de los mensajes sample del nodo y los transforma al formato aceptado por ThingsBoard indicando la marca de tiempo en la que se enviaron los mensajes (en segundos desde el epoch Unix), el identificador del nodo emisor del mensaje y los datos de las distintas muestras enviadas para su publicación en el canal "*telemetry"* del broker MQTT.

En los módulos *"location"* y *"time"* se realiza la interpretación de los mensajes de *Location* y *Time* respectivamente, creando los respectivos mensajes ajustándose al protocolo de ThingsBoard donde se notifica la posición geoespacial y la hora del sistema en el nodo, con el identificador del nodo que emitió el mensaje para su posterior publicación en el canal *"atributes"* del broker MQTT.

En el módulo *"errorWarning"* se traduce el mensaje recibido del nodo y se construye el mensaje para ThingsBoard en donde se notifica del error o warning que se indica en el mensaje recibido del nodo, el número de veces que ha sucedido en una ejecución y el identificador del dispositivo emisor, para su posterior publicación en el canal *"atributes"* del broker MQTT.

En los módulos de *"connect"* y *"disconnect"* se construye un mensaje con el identificador del dispositivo que se quiere dar de alta o baja en la plataforma ThingsBoard, estos mensajes se publicarán en los canales MQTT *"connect"* y *"disconnect"* respectivamente.

Por último, se encuentran módulos que establecen conexión con el servidor broker MQTT en los diferentes canales que a los que está suscrita la plataforma ThingsBoad en el broker MQTT, en nuestro proyecto utilizaremos cuatro canales: *"telemetry"* para la publicación de las muestras, *"connect"* y *"disconnect"* para publicar los mensajes de dar de alta o baja respectivamente, a un nodo de la plataforma ThingsBoard y *"atributes"* para el envío de información acerca del nodo (posición geográfica, hora del sistema y errores en el nodo).

4.5. Representación gráfica en ThingsBoard

Como ya quedó reflejado en la sección 4.2. de este mismo capítulo, ThingsBoard permite representar gráficamente los atributos y los datos telemétricos de los dispositivos que previamente se hayan dado de alta en la plataforma.

Para la realización de este proyecto hemos tenido a nuestra disposición dos nodos montados: uno de ellos dispone de todos los sensores incluido GPS incluido, se identifica a través de un ID único de dispositivo 70-b3-d5-49-98-46-56-33; El otro nodo disponible constaba de GPS y todos los sensores, salvo sensor de temperatura exterior, y queda identificado como 70-b3-d5-49-93-24-c1-b2.

Algunos atributos que manejamos con estos dispositivos son su *Longitud* y *Latitud*, que nos proporciona el módulo de posicionamiento de cada uno de ellos. ThingsBoard puede representar estos datos en un mapa para mostrar más claramente su situación geográfica, tal y como muestra la figura 34.

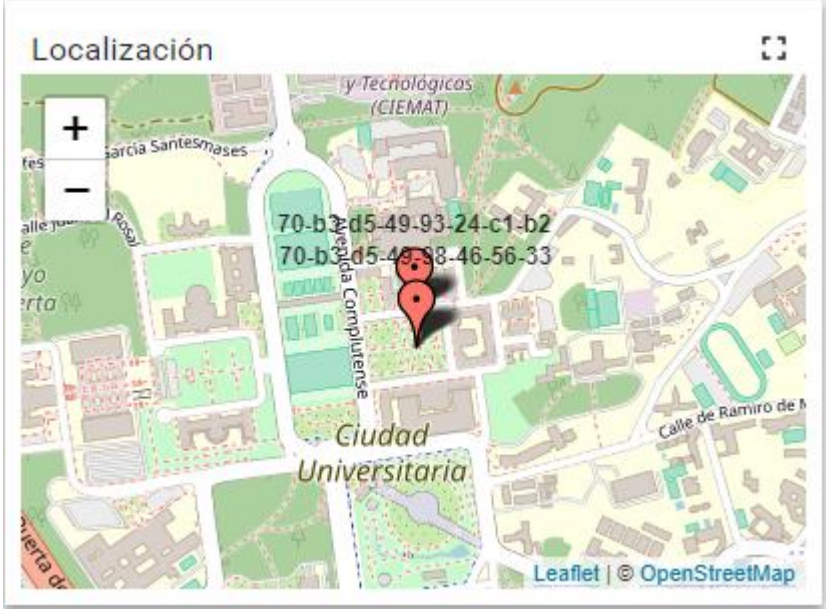

Figura 34: Visualización de Localización en ThingsBoard

Para representar los datos telemétricos, la plataforma ThingsBoard proporciona gran variedad de gráficos, algunos muy llamativos. Sobre todo, pensados para representar datos muy concretos y de un solo dispositivo.

Como en nuestro caso queremos visualizar las telemetrías de varios dispositivos a la vez, y en tiempo real, elegimos para ellos gráficos de trama como podemos ver en las figuras 35, 36 y 37 que muestran los gráficos de radiación, temperatura interna y humedad, respectivamente, de los dos nodos que disponemos.

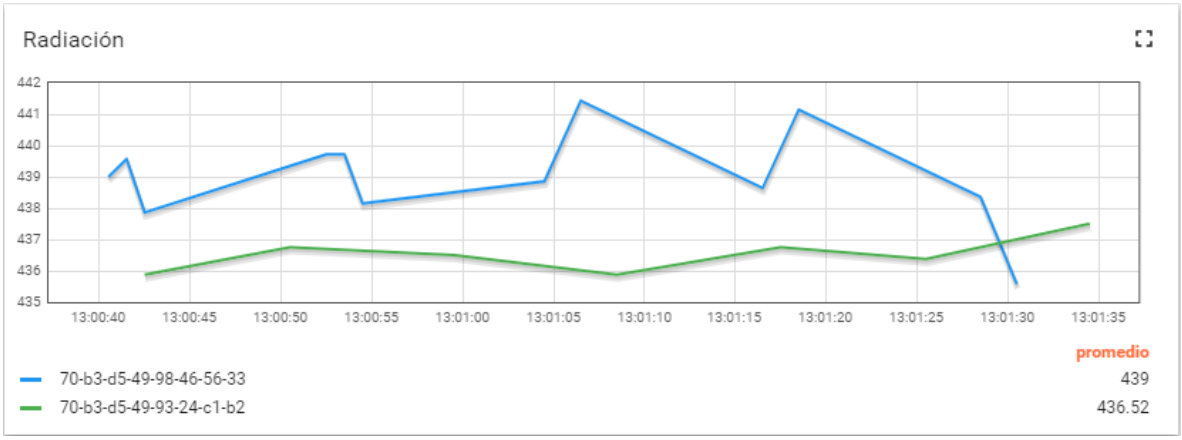

Figura 35: Visualización de Radiación en ThingsBoard

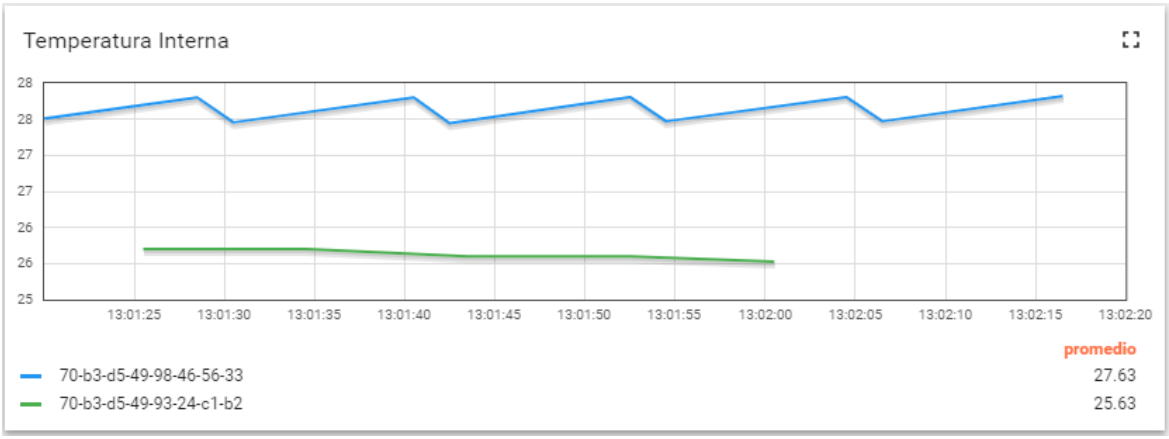

Figura 36: Visualización de Temperatura Interna en ThingsBoard

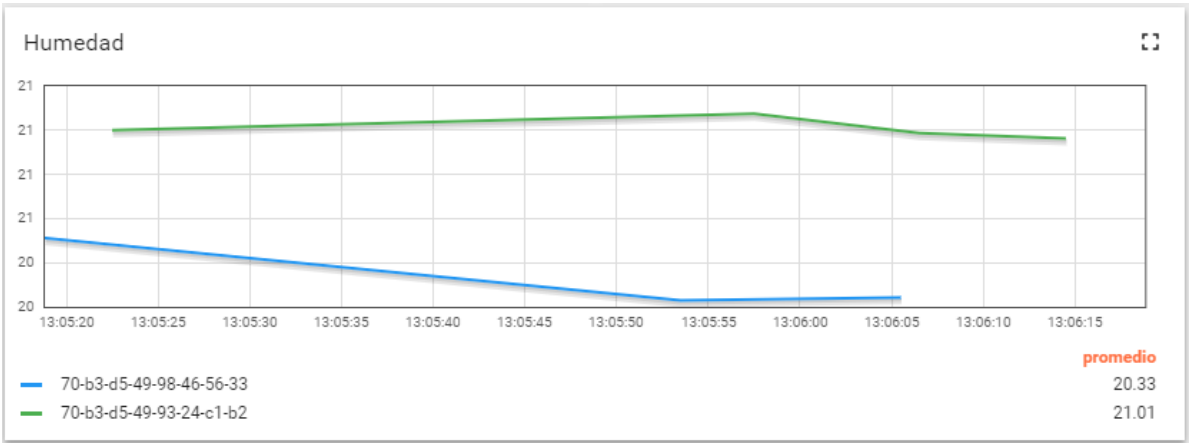

Figura 37: Visualización de Humedad en ThingsBoard

Sin embargo, como tan solo disponemos de sensor de temperatura exterior en uno de los nodos, elegimos otro tipo de visualización, que, a pesar de ser menos gráfica, permite revisar de forma más sencilla el historial de datos almacenados y, a su vez, sigue mostrando en tiempo real las muestras que se van recogiendo.

Figura 38: Visualización de Temperatura Exterior en ThingsBoard

Contribución al proyecto de Iván García Palomino

El proyecto consiste en una continuación de un trabajo de fin de grado de años atrás. Las primeras semanas se produjeron varias reuniones con los directores del trabajo de fin de grado del que parte este proyecto para tratar de comprender los pasos que se habían realizado anteriormente en el proyecto y poder fijar un punto de partida en este trabajo.

Después estas reuniones, se procedió a un diseño de la arquitectura software que cumpliese con los requisitos que nos habían marcado los directores del proyecto. Algunos de los requisitos que indicaron fueron que el diseño de la aplicación software fuese un conjunto de servicios independientes, y que se habilita la opción de modificación y control de los nodos solares desde el exterior.

Tras un primer diseño general se procedió a una reunión de los miembros del grupo con los directores de este proyecto para exponer el diseño inicial que los miembros del grupo habían realizado. Los directores aportaron sus opiniones y posibles mejoras a realizar en el diseño presentado.

Después de modificar el diseño con las aportaciones de los directores, se procedió al diseño de los protocolos de comunicación de entrada y salida del nodo, debido a que en el proyecto los nodos solares se comunican con un gateway a través de LoRa. A continuación, se procedió al diseño de la estructura general de directorios de la aplicación software en los nodos. Posteriormente, se realizó a una reunión de los miembros del grupo y los directores para poner en común los diseños realizados, los directores dieron el visto bueno a los diseños presentados.

En los primeros días después de la reunión se procedió a una toma de contacto con el hardware que conforma el nodo solar. Al no disponer de experiencia en el lenguaje de MicroPhyton, ni en trabajar con el tipo de limitaciones que presenta el hardware de este proyecto, se procedió a la realización de pruebas de algunas funcionalidades que estaban implementadas en el proyecto del que parte este trabajo.

A continuación, se pasó a la división de los servicios entre los miembros del grupo, Iván García realizó la implementación de los servicios generales (ManagerService, SamplingService) que requerían un mayor esfuerzo en implementación y Francisco José Hinojosa se encargó de los servicios sensores que requieren un mayor esfuerzo de investigación.

La unión de las dos partes realizadas por los miembros del grupo no resultó fácil, debido a que a cada vez que se añadían servicios a la aplicación se producían problemas en la memoria RAM, debido a la escasa capacidad que se dispone de ella.

Para ello los miembros del grupo realizaron investigaciones para encontrar una solución al problema presentado. Tras hablarlo con los directores, se decidió utilizar una técnica llamada *"Frozen Code"* que permite el ahorro de memoria RAM y nos otorga la posibilidad de seguir añadiendo el resto de los servicios que conforman la arquitectura software.

Una vez resuelto el problema de la memoria, los miembros del grupo encontraron problemas de funcionamiento en el módulo de posicionamiento GPS. Por lo que se procedió a revisar el material del que disponíamos del anterior proyecto. Tras la revisión del código de la librería del protocolo ubx se encontró un error en dicho código.

Resueltos estos dos problemas encontrados, los miembros del grupo se volvieron a dividir los servicios restantes. Iván García realizó la implementación del servicio ConnectionService, y la configuración de la herramienta node-RED en el gateway. Francisco José Hinojosa procedió a la implementación del servicio ErrorLogService.

Después de la unión de los respectivos trabajos, se encontraron errores que tuvieron que ser revisados de forma conjunta por los dos miembros del grupo, para poder subsanar los problemas de la manera más rápida posible.

Posteriormente los miembros del grupo trabajaron conjuntamente en la comunicación del gateway-servidor y en la plataforma ThingsBoard, de la cual se necesitó la ayuda del director del trabajo, José Ignacio Gómez que nos proporcionó dicha plataforma y la posibilidad de utilizar un servidor de la Facultad de Ciencias Físicas de la UCM.

Por último, Iván García realizó los últimos ajustes y modificaciones de la aplicación software, mientras que francisco José Hinojosa se centró más en la labor de documentación de este proyecto.

Para finalmente colaborar conjuntamente en la realización del análisis de tiempos de ejecución de la aplicación, para detectar el delay en el envío de los paquetes desde los nodos, y colaborar conjuntamente en el proceso de documentación de este trabajo. Cabe destacar el papel de los directores del trabajo en la revisión de varias versiones de la documentación y en la aportación de consejos para realizar de forma correcta la documentación del proyecto.

Contribución al proyecto de Francisco José Hinojosa Pérez

En primer lugar, cabe destacar que enfrentarse a la continuación de un proyecto que ya había comenzado unos años atrás, no fue sencillo en primera instancia para ninguna de las dos personas que hemos realizado este proyecto.

Gracias a la ayuda de los directores de este Trabajo de Fin de Grado que nos ayudaron a comprender el material que ya existía y, que nos guiaron a la hora del diseño inicial, conseguimos adaptarnos lo más rápido que pudimos para poder comenzar con nuestro trabajo.

El primer paso que dimos a la hora de realizar este proyecto, fue el de realizar un diseño de la aplicación software basado en servicios. Para lo cual, fue necesaria la colaboración de todo el equipo para una puesta en común de los objetivos y de los requisitos necesarios, para poder realizar un diseño que se ajustara correctamente a ello.

Una vez realizado el diseño inicial de la arquitectura software, fue necesario el trabajo conjunto de los dos miembros de la pareja para la creación de protocolos de entrada y salida de mensajes, pues conocíamos de antemano la existencia de un Gateway que debía recibir la información que procesáramos.

Con todos los diseños ya terminados, procedimos a la toma de contacto con la placa LoPy y los distintos sensores, lo que no resultó sencillo y nos llevó a realizar investigaciones individuales para, entre los dos, poder ponernos al día con todos los recursos que disponíamos.

En cuanto conseguimos comprender, más o menos, el funcionamiento de los elementos que teníamos a nuestra disposición, nos pusimos manos a la obra para realizar la implementación de la aplicación software. Para ello, dividimos los servicios resultantes en el diseño previo, quedándose Iván García con el grueso de los servicios generales (ManagerService y SamplingService), y Francisco José Hinojosa con el grueso de los servicios de sensores, que requerían un mayor esfuerzo en investigación, aunque a su vez, no precisaban de tanto esfuerzo en implementación.

Una vez que disponíamos cada uno de unos esquemas generales, unimos las partes para poder comenzar a trabajar con el nodo solar completo. Una tarea que no resultó sencilla, pues ninguno teníamos experiencia en trabajar con MicroPython, ni mucho menos, en disponer de un sistema de recursos tan limitados.

Llegados al punto de poder ir añadiendo servicio tras servicio a la aplicación general, nos encontramos con el mayor problema del proyecto. La limitada memoria RAM que disponía el sistema para ejecutar la aplicación. Para ello, nuevamente tuvimos que colaborar como equipo para investigar que ocurría y poder resolverlo lo antes posible.

Una vez resuelto el problema con la memoria del sistema, comenzamos a colaborar nuevamente, ya que nos resultaba complicado agregar el módulo de posicionamiento GPS, y tuvimos que realizar una investigación conjunta sobre el material que se nos había proporcionado, para finalmente, tras revisar completamente el código fuente de la librería del protocolo ubx, que ya había sido realizado en proyectos anteriores, encontrar un error en dicho código.

Tras esto, ya teníamos todos los elementos disponibles en la aplicación, por lo que nuevamente dividimos el trabajo, quedándose Iván García con la implementación del servicio de conexión y con la configuración del Gateway con Node-RED, mientras que Francisco José Hinojosa se centró en implementar un servicio de gestión de errores, creando con ello la tabla de codificación vista en este documento, y en ir agregando este nuevo módulo a todos los demás, lo que también precisó de una nueva investigación de cada sensor para poder conocer los rangos de valores que podrían devolver.

Después de volver a unir nuestro respectivo trabajo, surgieron nuevos errores que tuvimos que volver a retomar de forma conjunta para poder subsanarlos lo antes posible.

Posteriormente, con todos los elementos ya unidos, trabajamos de forma conjunta para llevar a cabo la interacción Gateway-brokerMQTT, y a su vez, aprender a trabajar con el cliente ThingsBoard. Para lo cual fue necesaria la colaboración con el director del TFG, José Ignacio Gómez, que nos proporcionó el poder usar dicha plataforma en un servidor físico de la Facultad de Ciencias Físicas de la UCM.

Por último, en la recta final del proyecto, Iván García continuó dando los últimos retoques a la aplicación software mientras que Francisco José Hinojosa se centró más en la labor de documentación de este proyecto. Para, finalmente, colaborar nuevamente realizando el análisis de tiempos de ejecución de la aplicación conjuntamente y, concluir esta documentación también de manera conjunta. Destacando nuevamente el papel de los directores de este Trabajo de Fin de Grado, que realizaron revisiones sucesivas del documento y, nos ayudaron a poder completarlo de una manera más correcta.

Conclusiones

El proyecto se presentó como el desarrollo y despliegue de una red de nodos solares autónomos, y tras la resolución del proyecto, podemos concluir que se ha conseguido algún nodo autónomo preparado para su despliegue en un entorno real.

A lo largo de la realización de este trabajo se presentaron algunos problemas, debido a los recursos limitados de los que se dispone en el nodo solar.

El problema más importante al que nos enfrentamos fue la escasa cantidad de memoria RAM de la que se dispone en la placa LoPy, tras la instalación de su firmware, y cuya resolución ocupó gran parte del tiempo dedicado a este proyecto.

Una vez que este problema quedó subsanado, descubrimos que los envíos periódicos de muestras no se realizaban en las frecuencias de envío deseadas. Para conocer el origen de dichos retrasos se pasó a la realización de un análisis de tiempos de ejecución para los procesos de recogida de las distintas muestras y de su posterior envío. Tras esto, llegamos a la conclusión de que el retraso se estaba produciendo en la creación y envío de los paquetes de datos de las muestras, lo que nos llevó a realizar modificaciones en la implementación de la recogida y envío de muestras.

Para realizar el envío de muestras desde los nodos solares a un gateway cercano, se utiliza conexión LoRa ya que nos ofrece ventajas como un bajo consumo, conexión de largo alcance y un uso público de manera gratuita sin suscripción a un operador. En cambio, LoRa ofrece un bajo ancho de banda, quedando su uso limitado por las agencias reguladoras de cada región.

Una posible continuación del proyecto se podría encaminar hacia el desarrollo de un nuevo servicio para la gestión de la batería, pudiendo regular el encendido o apagado de un nodo en función de la cantidad de batería disponible.

Además, se podría realizar un estudio energético que permitiese obtener una visión más clara de la influencia del powerPin en el consumo energético para cierto rango de frecuencias de muestreo. Valorando un diseño alternativo para obtener un mayor ahorro de energía.

Asimismo, cabe recordar que con el diseño de este proyecto queda habilitada la posibilidad de poder comunicarse con un nodo de manera remota desde el exterior. Parte que podría implementarse en un nuevo trabajo.

Por último, con este proyecto ya se obtienen las muestras necesarias para el diseño de un modelo predictivo de radiación solar a corto plazo, lo que permite la evolución del proyecto general.

Conclusions

The project was presented as the development and deployment of a autonomous solar nodes network, and after the resolution of the project, we can conclude that an autonomous solar node prepared for its deployment in a real environment has been achieved.

Throughout the realization of this work some problems arose, due to the limited resources available in the solar node.

The most important problem we faced was the small amount of RAM that is available on the LoPy board, after the installation of its firmware, and whose resolution took up a lot of the time dedicated to this project.

Once this problem was corrected, we discovered that periodic sample shipments were not made at the desired delivery frequencies.

In order to know the origin of these delays, an execution time analysis was carried out for the collection processes of the different samples and their subsequent delivery.

After this we came to the conclusion that the delay was occurring in the creation and sending of the data packages of the samples, which led us to make changes in the implementation of the collection and shipment of samples.

To send samples from the solar nodes to a nearby gateway, LoRa connection is used as it offers advantages such as low consumption, long-range connection and public use for free without subscription to an operator. In contrast, LoRa offers low bandwidth, its use being limited by the regulatory agencies of each region.

A possible continuation of the project could be directed towards the development of a new service for the management of the battery, being able to regulate the turning on or turning off of a node depending on the amount of available battery.

In addition, an energy study could be carried out to obtain a clearer vision of the influence of powerPin on energy consumption for a certain range of sampling frequencies. Valuing an alternative design to obtain greater energy savings.

Also, remember that with the design of this project enables the possibility to communicate with a node remotely from the outside. Part that could be implemented in a new job.

Finally, with this project, the necessary samples are already obtained for the design of a short-term predictive solar radiation model, which allows the evolution of the general project.

Referencias y Bibliografía

A lo largo de todo el documento se han incluido en notas a pie de página las referencias necesarias que apoyan los conceptos relacionados.

Entre todas ellas, destacamos de manera especial las siguientes:

[1] Documentación Oficial de Pycom:

https://docs.pycom.io/

[2] Documentación Oficial de MicroPython:

http://docs.micropython.org/en/latest/

[3] Foro Oficial de Pycom:

https://forum.pycom.io/

[4] Web Oficial Node-RED:

https://nodered.org/

[5] Web Oficial ThingsBoard:

https://thingsboard.io/

[6] Web Oficial aplicación de desarrollo Atom:

https://atom.io/

[7] LoRa y sus características:

https://www.tuv.com/media/corporate/products_1/electronic_components_and_lasers/TUeV_Rheinland_Overview_LoRa_and_LoRaWANtmp.pdf

[8] Protocolo MQTT:

https://github.com/mqtt/mqtt.github.io/wiki

[9] Documentación Oficial LoPy:

https://docs.pycom.io/datasheets/development/lopy.html

Anexo 1: Tabla de Codificación de Errores/Warnings

Tal y como vimos en la sección 3.3.4. del Capítulo 3 de este documento, dividimos los distintos errores que se pueden producir a lo largo de la ejecución de la aplicación en errores (bloquean la aplicación) y warnings (permiten continuar la ejecución e informa del suceso).

El código de error devuelto por cada uno de los sucesos puede consultarse en la Tabla 3, que aparece a continuación.

ERROR	CÓDIGO	TIPO	DESCRIPCIÓN
OSError	-1	ERROR	Error en el acceso, escritura o lectura de ficheros
ConfFile Error	-2	WARNING (1)	Error fichero de configuración con estructura incorrecta
CreateThread Error	-3	WARNING (1)	Error al crear un hilo
GPSConnection Error	-4	WARNING (1)	Imposible conectar con señal GPS
NoService Error	-5	WARNING (1)	Error servicio al que quieres acceder no existe en la lista de servicios
Active Service Error	-6	WARNING (1)	Error al intentar activar un servicio ya activo
Non-Active Service Error	-7	WARNING (1)	Error al intentar desactivar un servicio que no está activo
Incorrect Atribute Error	-8	WARNING (1)	Error al leer un atributo incorrecto de un sensor (confService y updateService)
Incorrect AtributeValue Error	-9	WARNING (1)	Error un atributo tiene un valor no válido (frecuencia de muestreo negativa en conf.txt)
ZeroDivisionError (internal)	-10	WARNING (5)	Error al intentar dividir entre 0
Incorrect Value Error	-11	WARNING (5)	Error por un valor incorrecto devuelto por un sensor (fuera de rango de valores)
LocationError	-12	WARNING (1)	Error de conexión con el satélite de posicionamiento

Tabla 3: Tabla de Codificación de Errores y Warnings con descripciones

Los warnings incluyen un número de veces, entre paréntesis en la Tabla 3, modificable a través de un fichero de configuración, que pueden suceder antes de informar al servidor, y a su vez habilita la posibilidad de ir almacenando en un fichero interno los fallos que se hayan ido produciendo.

Anexo 2: Frozen Code

La técnica de *"Frozen code"* consiste en incluir tus propios módulos en una versión del firmware de MicroPhyton, permitiendo así el uso de la memoria RAM para carga dinámica.

El proceso de instalación de las herramientas necesarias en el proceso de *Frozen Code* consta de la creación de un nuevo directorio y la ejecución de los siguientes comandos:

- sudo apt-get install gcc git wget make libncurses-dev flex bison gperf python python-pip python-setuptools python-serial python-cryptography python-future python-pyparsing (paquetes necesarios para el proceso)
- mkdir pycom
- git clone https://github.com/pycom/pycom-micropython-sigfox.git
- git clone https://github.com/pycom/pycom-esp-idf.git
- cd pycom-esp-idf
- git submodule update –init
- cd ..
- wget https://dl.espressif.com/dl/xtensa-esp32-elf-linux64-1.22.0-80-g6c4433a-5.2.0.tar.gz
- tar zxf xtensa-esp32-elf-linux64-1.22.0-80-g6c4433a-5.2.0.tar.gz
- export PATH=$PATH:$HOME/pycom/xtensa-esp32-elf/bin
- export IDF_PATH=~/pycom/pycom-esp-idf
- cd pycom-micropython-sigfox
- cd mpy-cross && make clean && make && cd ..

Para incluir el código en el firmware de MycroPhyton hay que copiar el código de la aplicación en el directorio /esp32/frozen/Common, y después copiar el main.py y el boot.py de la aplicación en el directorio /esp32/frozen/Base/.

Finalmente, ponemos el dispositivo en modo *"bootloader"*, abrimos una terminal en el directorio ../esp32 y ejecutamos los siguientes comandos:

- make BOARD=LOPY clean
- make BOARD=LOPY TARGET=boot
- make BOARD=LOPY TARGET=app
- make BOARD=LOPY ESPPORT= "Ruta del dispositivo" flash

En el proceso de creación de la guía se hizo uso de la información de un hilo en los foros de Pycom[27] y de información proporcionada por Pycom en github[28].

[27] https://forum.pycom.io/topic/1919/how-i-built-a-custom-firmware-for-frozen-bytecode
[28] https://github.com/pycom/pycom-micropython-sigfox/blob/master/README.md

Anexo 3: Diagramas de secuencia

En este anexo incluimos dos diagramas de secuencia que por motivos de formato no aparecen en su respectiva sección.

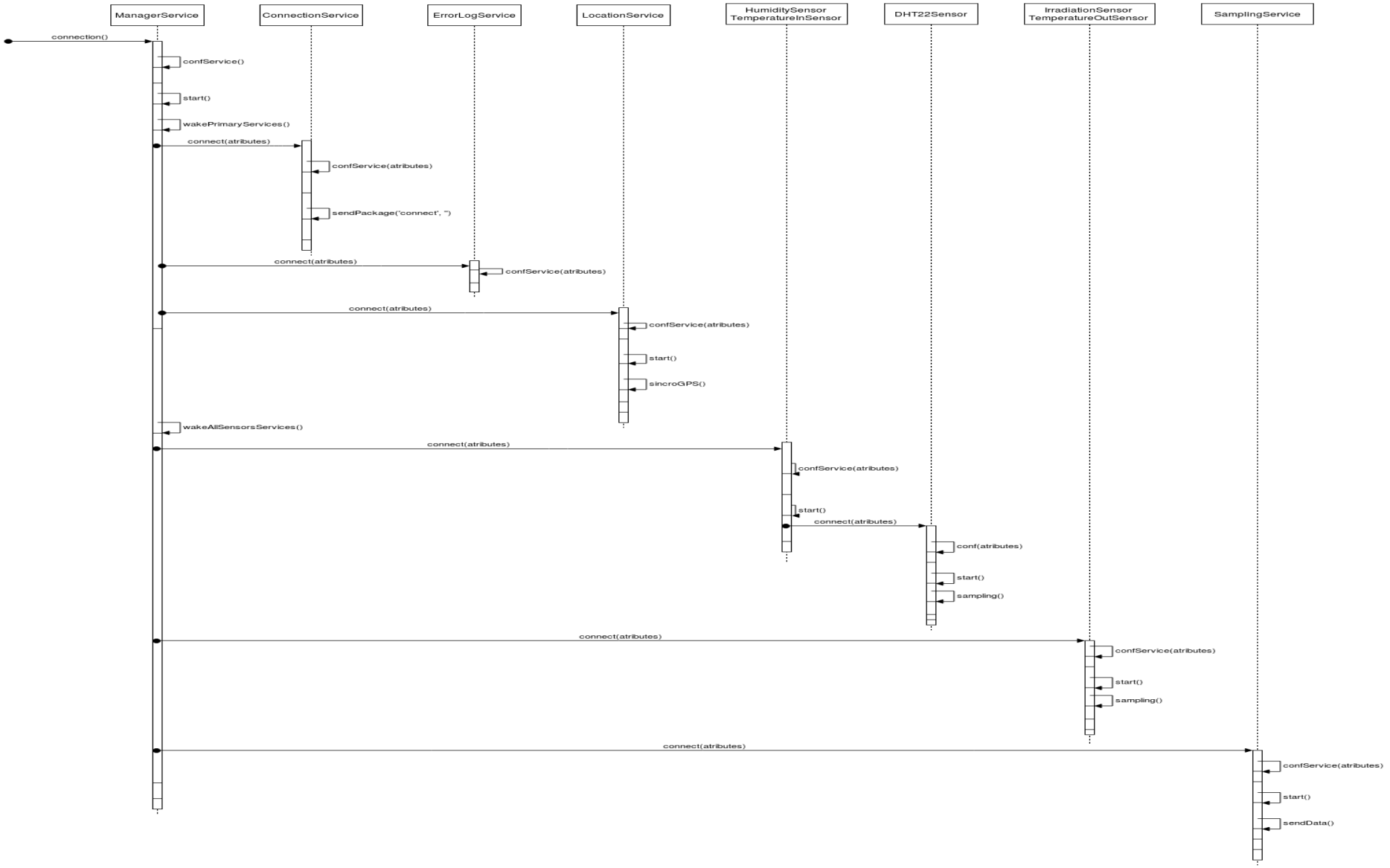

Figura 7: Diagrama de Secuencia de Configuración Inicial de un Nodo Solar

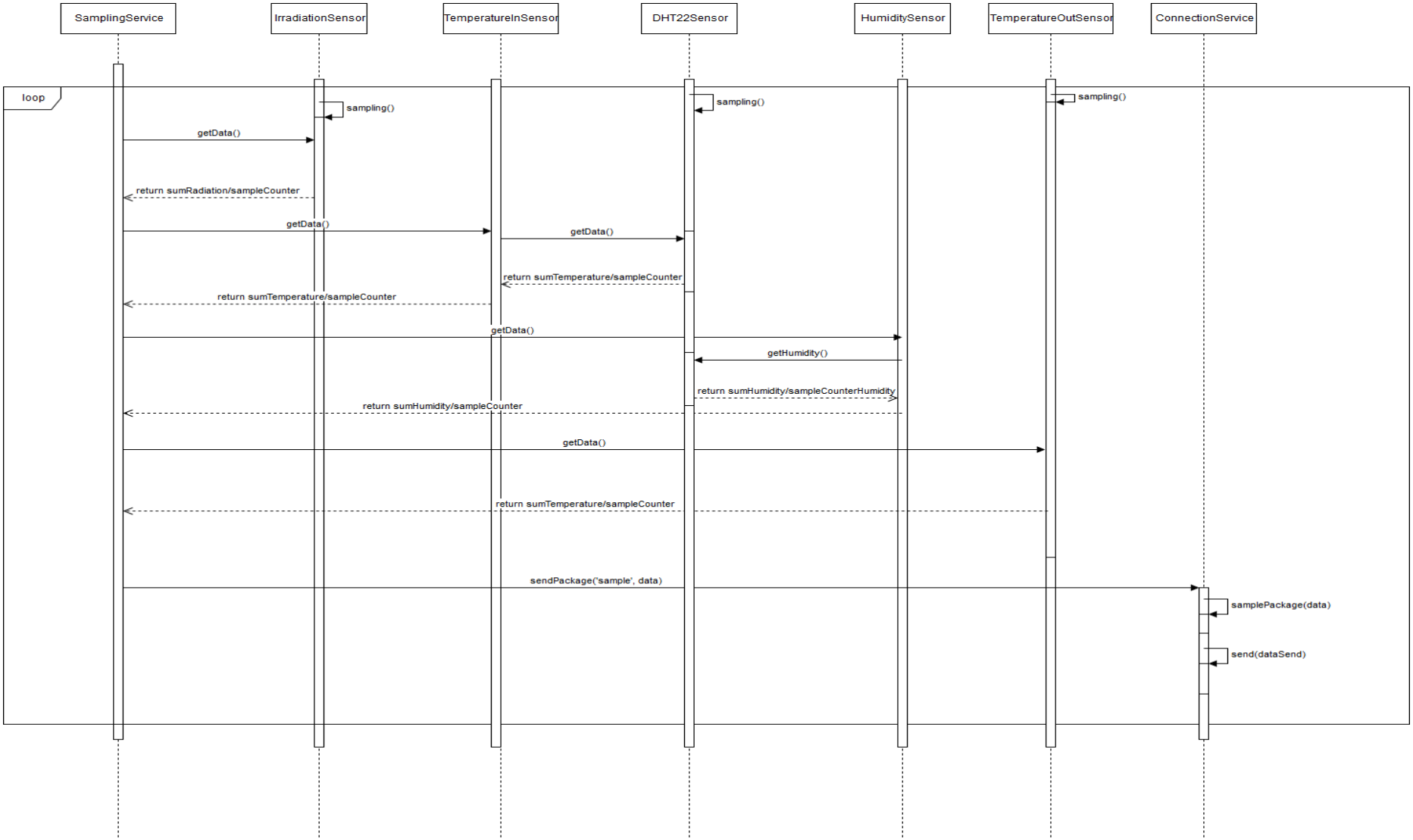

Figura 8: Diagrama de Secuencia de bucle principal de Proceso de Recogida de Muestras