Universidad Politécnica
de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Diseño y Desarrollo del Módulo Controlador de un Sistema de Potabilización de Agua Modular

Autor: Víctor Arzoz Consuegra

Tutora: Elena Villalba Mora

Madrid, julio 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática
Título: Diseño y Desarrollo del Módulo Controlador de un Sistema de Potabilización de Agua Modular
Julio 2025

Autor: Víctor Arzoz Consuegra

Tutor:
Elena Villalba Mora
Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software
ETSI Informáticos
Universidad Politécnica de Madrid

Agradecimientos

A mi tutora Elena, por sacar tiempo de otros universos para dedicarlo a este proyecto.

Al equipo pasado y presente de Purify my Water, por todo el esfuerzo y dedicación desinteresada destinados a mejorar el mundo.

A mi profe Tere, por dejarme estar con los mayores y ser el germen de mi pasión.

A mis padres, por no dejar nunca de apoyarme personal y profesionalmente.

A mi tío Patxi, por dejarme desarrollar mis capacidades con sus inventos.

A todas las instituciones, empresas, personas y otras entidades que han colaborado de una forma u otra a que este proyecto se haga realidad.

Resumen

Este Trabajo Fin de Grado presenta el diseño y desarrollo del módulo controlador del proyecto “Purify my Water”, un sistema de potabilización de agua modular, autónomo y autosuficiente, pensado para operar a pequeña escala en zonas remotas con escasez de recursos. El proyecto busca dar respuesta al desafío global de acceso a agua potable segura y asequible, en línea con el Objetivo de Desarrollo Sostenible 6.1.

El trabajo presenta el diseño de alto nivel de todo el sistema de potabilización, que se compone de una Unidad de Control Electrónico (ECU), un módulo de energía y diversos módulos de potabilización intercambiables que se adaptan a diferentes contaminantes del agua. Este se centra en el diseño y desarrollo de los componentes software clave de la ECU, implementados sobre un microcontrolador ESP32 y el framework ESP-IDF con FreeRTOS.

Los principales componentes desarrollados son:

- Un sistema de comunicación intermodular robusto y eficiente basado en el bus CAN y el protocolo de transporte ISO-TP.
- Un sistema de gestión de la configuración para almacenar y modificar de forma persistente los parámetros del sistema.
- Un gestor de dispositivos encargado de descubrir y mantener un listado actualizado de los módulos conectados.
- Un sistema de actualizaciones inalámbricas (OTA) que permite el mantenimiento y la mejora del software a distancia mediante una conexión Wi-Fi.

La metodología de desarrollo pone el foco en la calidad y la fiabilidad, empleando el Desarrollo Guiado por Pruebas (TDD), la Integración Continua (CI) con GitHub Actions y herramientas de análisis de código como SonarQube y Clang-tools. Para desacoplar el software del hardware y facilitar las pruebas, se ha recurrido al uso de capas de abstracción y a técnicas como el mocking y faking.

El proyecto supone una contribución técnica importante para la creación de una solución de potabilización asequible y escalable, con especial hincapié en la modularidad y la operación autónoma para minimizar el mantenimiento y la dependencia de cadenas de suministro.

Abstract

This Final Degree Project details the design and development of the controller module for "Purify my Water," a modular, autonomous, and self-sufficient water purification system designed for small-scale use in remote areas with limited resources. The project aims to address the global challenge of providing safe and affordable drinking water, aligning with Sustainable Development Goal 6.1.

The work presents the high-level design of the entire purification system, which is composed of an Electronic Control Unit (ECU), an energy module, and various interchangeable purification modules that adapt to different water contaminants. The core of this thesis is the design and development of the ECU's key software components, built upon an ESP32 microcontroller and the ESP-IDF framework with FreeRTOS.

Key developed components include:

- An inter-modular communication system using the CAN bus and the ISO-TP protocol to ensure robust and efficient data exchange.
- A centralized configuration management system for storing and modifying operational parameters persistently.
- A device management system to discover and maintain a list of connected modules.
- An Over-The-Air (OTA) update system that allows for remote software maintenance and improvements via a Wi-Fi connection.

The development methodology emphasizes quality and reliability through Test-Driven Development (TDD), Continuous Integration (CI) using GitHub Actions, and the use of analysis tools like SonarQube and Clang. To decouple software from specific hardware and facilitate testing, the design extensively uses hardware abstraction layers and techniques like mocking and faking.

The project represents a significant technical contribution towards creating an affordable, scalable, and effective solution for water purification in developing regions, with a strong focus on modularity and autonomous operation to minimize maintenance and the need for a constant supply chain.

Tabla de contenidos

1 Introducción y Objetivos

1.1 El problema de la falta de agua potable

Según datos de la OMS [1], en 2021 más de 2.000 millones de personas vivían en países con escasez de agua. En 2022 había en el mundo al menos 1700 millones de personas que tomaban agua para consumo contaminada con heces. La contaminación microbiana del agua potable supone el mayor riesgo de toxicidad, pudiendo transmitir enfermedades diarreicas, cólera, disentería, fiebre tifoidea y poliomielitis. Según los cálculos, esta contaminación causa cada año 505.000 muertes por enfermedades diarreicas.

La meta 6.1 de los Objetivos de Desarrollo Sostenible consiste en proporcionar acceso universal y equitativo a agua potable a un precio asequible [2]

1.2 El problema de las estaciones de tratamiento de agua potable

La solución actual al problema de la falta de agua potable en los países desarrollados consiste en la construcción y operación de Estaciones de Tratamiento de agua Potable (ETAP).

Las ETAP son construcciones altamente complejas y costosas, que requieren varios años para ser construidas y utilizan una gran cantidad de recursos en su operación [3] Sus ventajas son el elevado caudal que pueden procesar y su eficiencia, siendo idóneas para abastecer a grandes poblaciones.

Por su elevado coste y la falta de infraestructuras necesarias, las ETAP no son viables en una gran cantidad de entornos, sobre todo rurales, lo que deja a la población de esas áreas desprotegidas frente a enfermedades relacionadas con el agua.

1.3 Objetivos

El proyecto, de cuya parte forma este Trabajo Fin de Grado, consiste en la realización de un sistema de potabilización de agua modular, autónomo y autosuficiente a pequeña escala. Está pensado para proveer agua potable a unos pocos centenares de personas, siendo ideal su uso en zonas remotas, con escasez de recursos materiales y energéticos.

El proyecto se está llevando a cabo por un equipo multidisciplinar de ingenieros informáticos, biotecnólogos, técnicos electrónicos y estudiantes de ingeniería, basándose en el diseño original del autor de este Trabajo Fin de Grado.

Este Trabajo presenta el diseño de alto nivel de todo el sistema, pero se centra en el diseño de ciertos componentes de la unidad de control de la potabilizadora, también llamada módulo controlador. En concreto, se han diseñado y desarrollado los siguientes componentes:

- El sistema de comunicaciones inter modular, que utiliza el bus CAN (ISO 11898) y el protocolo de transporte ISO-TP (ISO15765-2) para proveer a los módulos de un sistema de comunicaciones bidireccional robusto y eficiente.
- El sistema de administración de la configuración, esencial para poder almacenar y modificar los distintos parámetros necesarios para operar el sistema.
- El sistema de control de la potabilización, que se encarga de gestionar que módulos se encuentran funcionando a la vez, balanceando el mayor rendimiento posible dentro de las condiciones energéticas del sistema.
- El sistema de actualizaciones inalámbricas, que permite corregir fallos e introducir mejoras a distancia al conectar el sistema a una red Wifi como, por ejemplo, la red Wifi portátil de cualquier smartphone moderno.

Dentro de este proyecto, el autor se encarga de coordinar a los equipos de desarrollo de hardware, software y potabilización, y participa activamente en el diseño de hardware, y en el diseño y desarrollo de software. Como parte de este Trabajo Fin de Grado, se muestra parte del diseño de alto nivel cocreado por el equipo de desarrollo en su conjunto, y el diseño e implementación de un grupo de componentes software que sí forman parte del esfuerzo dedicado a este Trabajo.

El objetivo final de este proyecto es ayudar a cumplir la meta 6.1 de los Objetivos de Desarrollo Sostenible “Proporcionar acceso universal y equitativo a agua potable salubre y asequible” utilizando las herramientas TIC a nuestro alcance, proporcionando un impacto positivo en las comunidades donde se utilice el sistema.

La lista de objetivos es la siguiente:

- O1: Estudio del problema y el estado del arte.
- O2: Diseño de un sistema de sistema de potabilización de agua a pequeña escala barato, modular y automático.
- O3: Diseño de la unidad de control del sistema (llamada internamente ECU).
- O4: Desarrollo de varios componentes software de la unidad de control.
- O5: Validación y Pruebas (unitarias, de integración y funcionales)

2 Estado del Arte

2.1 Sistemas Empotrados

2.1.1 Sistemas en Tiempo Real

"Cualquier sistema en el que el tiempo en el que se produce la salida es significativo. Esto generalmente es porque la entrada corresponde a algún movimiento en el mundo físico, y la salida está relacionada con dicho movimiento. EI intervalo entre el tiempo de entrada y el de salida debe ser lo suficientemente pequeño para una temporalidad aceptable" [4]

Existen dos tipos principales de sistemas en tiempo real:

- Sistemas estrictos (hard):
  Son aquellos en los que la respuesta se debe dar dentro de los márgenes establecidos sin excepciones. Por ejemplo, un marcapasos.
- Sistemas no estrictos (soft):
  Son aquellos en los que ocasionalmente se puede retrasar los márgenes establecidos sin ocasionar fallos críticos, aunque el valor de la respuesta se vea disminuido. Por ejemplo, el muestreo de sensores de una estación meteorológica.

2.1.2 Definición

"Un sistema embebido (SE) o sistema empotrado es un sistema electrónico diseñado específicamente para realizar unas determinadas funciones, habitualmente formando parte de un sistema de mayor entidad. La característica principal es que emplea para ello uno o varios procesadores digitales (CPUs) en formato microprocesador, microcontrolador o DSP lo que le permite aportar 'inteligencia' al sistema anfitrión al que ayuda a gobernar y del que forma parte." [5]

2.1.3 Peculiaridades de los sistemas empotrados

Los sistemas empotrados están específicamente diseñados para resolver un problema concreto, optimizando tanto el hardware como el software utilizado a la resolución del problema planteado, por ello presentan estas diferencias:

- Un sistema empotrado tiene unos recursos de hardware muy limitados en comparación con un ordenador de propósito general, ya que solo debe cumplir una tarea concreta. Generalmente la velocidad de procesamiento se encuentra por debajo del gigahercio, y las cantidades de memoria (tanto de datos como instrucciones) no superan el centenar de megabytes.
- Un sistema empotrado forma parte de un sistema más complejo, habitualmente uno que interactúa con el mundo real, y que requiere especificaciones de tiempo, convirtiéndose en un sistema en tiempo real, con las limitaciones que ello implica, sobre todo en cuanto a la necesidad del programa de ser predecible, es decir, de eliminar o disminuir toda fuente de incertidumbre, incluyendo el uso de memoria dinámica.
- Un sistema empotrado se desarrolla de manera distinta a un sistema tradicional, principalmente debido a que el sistema en el que se desarrolla el programa y el sistema en el que se ejecuta son distintos. Eso dificulta el uso de test automáticos y la depuración del programa. Para contrarrestarlo, existen métodos especiales de desarrollo que se verán más adelante.

2.2 Acceso a Agua Potable

2.2.1 Agua apta para el Consumo

Se denomina agua apta para el consumo, o agua potable a aquella que no contiene ningún tipo de microorganismo o sustancia en una cantidad o concentración que pueda suponer un peligro para la salud humana, y que cumple con los requisitos especificados para los parámetros microbiológicos, químicos, indicadores de calidad y radiactivos [6]

2.2.2 Retos en el Acceso a Agua Potable

El acceso al agua potable presenta varios retos derivados tanto de factores naturales como humanos:

2.2.2.1 Escasez de Recursos Hídricos y Cambio Climático

Según la ONU [7], dos mil millones de personas viven en países donde hay un alto nivel de estrés por la escasez de agua y unos 4 mil millones de personas sufren una escasez de agua grave durante al menos un mes al año. En el último siglo, el uso de agua a nivel mundial ha aumentado a más del doble de la tasa de crecimiento demográfico y en ese periodo se ha perdido entre un 50% y un 70% de los humedales naturales en todo el mundo.

2.2.2.2 Contaminación de Fuentes de Agua

La disponibilidad de suficientes fuentes de agua seguras está directamente relacionada con el manejo de las aguas residuales. Se estima que el 80% de las aguas residuales se vierten en el ecosistema sin tratar o reutilizarse antes [8].

El origen de estas aguas viene principalmente de 3 actores: ciudades, industria y agricultura. Las aguas residuales sin tratar que se vierten desde las ciudades principalmente son vertidas desde los hogares donde se utiliza directamente a la masa de agua más cercana, contaminándola con residuos químicos, médicos y biológicos. La industria es responsable de la mitad del consumo de agua en los países desarrollados, y de un 4-12% en países en vías de desarrollo [8]. Aunque la tendencia está cambiando, mucha de esa agua no se reutiliza o se trata antes de verterla al ecosistema, especialmente en el caso de la industria minera. El contexto de la agricultura es distinto, ya que en los últimos años se ha promovido el uso de fertilizantes y pesticidas, que luego se filtran en los acuíferos contaminando las fuentes de agua. Este problema es especialmente grave en países en vías de desarrollo, donde estas prácticas no están controladas. Actualmente los agricultores están buscando utilizar fuentes de agua no convencionales, como aguas residuales tratadas de manera que contengan una mayor proporción de nutrientes, para no poner más estrés en las fuentes de agua potable.

2.2.2.3 Falta de infraestructura

Muchas zonas de países en vías de desarrollo no se pueden permitir los costes de construcción y operación de la infraestructura de potabilización de agua, y de la que la sostiene (canalizaciones, red eléctrica, etc.), con costes que pueden ir desde los pocos miles hasta varios millones de euros [9]. Sin esta infraestructura, millones de personas dependen de fuentes alternativas mucho menos seguras para la obtención de agua.

2.3 Potabilización de Agua: Soluciones Adaptadas para Países en Vías de Desarrollo

Se van a dividir las soluciones actuales de potabilización de agua en base a el uso de las Tecnologías de la Información en el funcionamiento de cada producto.

2.3.1 Soluciones no basadas en TI

A continuación, se presentan varias soluciones que no están basadas en el uso de sistemas informáticos y que, por su simplicidad, se utilizan en gran parte de los países en vías de desarrollo.

2.3.1.1 BioZeo

Es un proyecto en desarrollo por la Universidad de Chile cuyo objetivo es desarrollar una tecnología capaz de remover metales pesados del agua en zonas rurales, utilizando bacterias y zeolita activa. [10] Al ser un método de filtrado activo, su capacidad se va degradando con el uso.

2.3.1.2 SODIS

Es un método de esterilización de agua extremadamente barato y sencillo que consiste en almacenar el agua previamente filtrada para eliminar la turbidez en contenedores transparentes y dejarlos expuestos a la radiación solar durante varias horas. De esta manera, la radiación solar consigue 2 efectos: Destruye las cadenas de ARN con la incidencia de radiación UV-A, y además gracias a la radiación infrarroja, si se supera un umbral mínimo de temperatura, se consiguen resultados similares a la pasteurización. [11]

Sus principales limitaciones son su dependencia absoluta de absorber una gran cantidad de radiación solar, su lentitud a la hora de potabilizar y la necesidad de que el agua a potabilizar no sea turbia.

2.3.1.3 Ecofiltro

Es un producto de filtración y desinfección que funciona mediante un filtrado de cerámica. Se compone de un filtro compuesto por arcilla, serrín y plata coloidal. Durante el proceso de fabricación del filtro, el serrín se convierte en carbono activado y cada material cumple una función: la arcilla al secarse y convertirse en cerámica consigue filtrar las partículas presentes en el agua gracias a los pequeños poros presentes, el carbón activado ayuda a eliminar la turbidez, los malos olores y sabores, y la plata actúa como un bactericida. [12]

Se recomienda renovar los filtros cada 2 años, ya que con el uso se van degradando y perdiendo eficacia.

2.3.2 Soluciones basadas en TI

Ahora se presentan varias soluciones que utilizan sistemas de TI para operar. Son soluciones generalmente más complejas, pero suelen obtener mejores resultados en cuanto a la calidad del agua.

2.3.2.1 SafeWaterAfrica

Es un proyecto desarrollado por una coalición de instituciones y empresas africanas y europeas, y financiado por el programa "Horizon 2020 research and innovation" de la Unión Europea. Consiste en una planta de potabilización de agua capaz de producir hasta 100m^3 de agua con un índice de calidad acorde con las guías de la OMS. Utiliza como tecnologías subyacentes un sistema de generación de ozono, mecanismos de electrocoagulación y electrodiálisis, un sistema de monitoreo y análisis remoto y un sistema de paneles solares como método de obtención de energía [13]. El uso de estas técnicas lo convierten en uno de los sistemas con mejor rendimiento de su categoría, y el coste del proyecto se eleva a casi 3 millones de euros (contando la investigación y desarrollo de 2 plantas gemelas). [14]

2.3.2.2 Desolenator

Es un proyecto desarrollado en Emiratos Árabes Unidos que utilizan unos paneles solares optimizados que obtienen tanto electricidad como agua caliente y los utilizan para convertir agua en vapor y destilarla, eliminando así los contaminantes [15]. Todo el proceso se realiza de manera automática mediante electrónica de control. Este diseño permite tener un sistema que no utilice consumibles. Un punto en contra es que a pesar de utilizar unos paneles solares más eficientes (60% vs 15% de un panel tradicional [15]), el consumo de energía sigue siendo muy elevado debido al alto calor especifico del agua que se necesita evaporar.

2.3.2.3 H2gO Purifier GLOBAL

Es un sistema de potabilización de agua de bolsillo con la capacidad de crear una solución clorada que actúa como desinfectante para cantidades de hasta 20L de agua. El dispositivo calcula automáticamente la concentración de desinfectante en función del tamaño del recipiente a tratar. El único consumible utilizado es unos pocos gramos de sal, aparte de la energía, que proviene de una batería que se puede recargar con un pequeño panel solar incorporado, o con una conexión USB [16]. Aunque no es capaz de realizar un filtrado físico del agua, si es capaz de desactivar la mayor parte de los microorganismos con bastante efectividad y se está utilizando tanto para expediciones como en países en vías de desarrollo como sistema de potabilización de cada unidad familiar.

2.4 Purify my Water

PurifyMyWater es un proyecto de emprendimiento social en desarrollo que busca diseñar y construir un sistema modular, autónomo y autosuficiente de potabilización de agua de bajo coste y a pequeña escala. Utilizando un diseño modular se espera construir una plataforma sobre la que conectar módulos de potabilización intercambiables que se encarguen de permitir al sistema adaptarse a diferentes tipos de agua contaminada. [17]

El diseño modular va más allá todavía, ya que la entrada de energía al sistema también se considera un módulo y permite poder conectar paneles solares o alimentarse de la red eléctrica entre otras opciones.

El objetivo del proyecto es proporcionar 1000 litros de agua potable por día, suficiente para abastecer a pequeñas aldeas o al personal de colegios u otros centros, todo ello minimizando el uso de consumibles, utilizando técnicas de potabilización eficientes e innovadoras y siendo una alternativa más barata que otras opciones con calidad similar, y más efectiva que soluciones en el mismo rango de precio.

Este proyecto nació en 2017 para solucionar el problema de falta de agua potable en una pequeña escuela en la aldea de Kanhakro, en Costa de Marfil en colaboración con el colegio Joyfe de Madrid.

Desde su creación, Purify My Water ha logrado hitos clave como el desarrollo de un prototipo funcional, reconocimiento en medios internacionales como El País y Forbes [18], alianzas estratégicas con entidades como Unicef y Ashoka [19] y recientemente ha sido premiado a una de las 10 mejores ideas en el concurso de emprendimiento Actúa UPM [20]. Además, ha recibido apoyo técnico de instituciones como la UPM, consolidando su capacidad de innovación y su potencial de impacto global.

El plan de desarrollo futuro del proyecto contempla la implementación de 10 unidades piloto en los próximos dos años, seguidas por la expansión a 50 unidades en cuatro años y un despliegue de 2.500 unidades en 20 años, beneficiando a más de 1,25 millones de personas. Estos objetivos estarán respaldados por certificaciones internacionales, alianzas público-privadas y un modelo de franquicias sociales para escalar la solución a nivel global.

Purify My Water no solo busca resolver una necesidad urgente, sino también generar un impacto sostenible, contribuyendo a los Objetivos de Desarrollo Sostenible (ODS), incluyendo agua limpia y saneamiento (ODS 6), salud y bienestar (ODS 3) y energía asequible (ODS 7). Su enfoque innovador y socialmente responsable posiciona al proyecto como un referente en soluciones hídricas accesibles, sostenibles y escalables.

En este proyecto se engloba el presente Trabajo Fin de Grado, centrándose en el diseño e implementación de la Unidad de Control, pieza clave en la gestión de los módulos de potabilización.

3 Métodos y Herramientas

3.1 Métodos

3.1.1 Test Driven Development for Embedded Software

Consiste en desarrollar el producto siguiendo la siguiente metodología: [21]

1. Desarrollar primero un test unitario de la funcionalidad a implementar
2. Comprobar que el test no se ejecuta o falla
3. Escribir la menor cantidad de código para conseguir que el test no falle
4. Repetir pasos 1-4
5. Una vez no quedan más test que describan la funcionalidad a implementar, refactorizar el código generado en los pasos anteriores.

También se deben seguir estas reglas:

- Producir software lo más desacoplado del hardware posible, para facilitar las pruebas y el desarrollo de software cuando el hardware no está listo.
- Tener en cuenta las limitaciones propias de un sistema empotrado, tales como la escasez de memoria, la velocidad de los microcontroladores, y limitaciones en diversas técnicas como el uso de memoria dinámica o mecanismos de sincronización avanzados.

3.1.2 Integración continua

Es una metodología que consiste en subir los cambios hechos al código (junto con las pruebas que sean necesarias) varias veces al día, disparando cada vez un proceso automatizado que verifica que el nuevo código funciona correctamente con el código existente. [22]

Esto permite detectar incompatibilidades entre piezas de software en una etapa temprana, reduciendo el coste de resolver el problema.

También proporciona una retroalimentación rápida para detectar que código falla y poder solucionarlo mucho más rápido y de forma más sencilla que realizando la integración y pruebas una vez desarrollado todo el producto.

Por último, mantiene el código en un estado funcional, permitiendo entregar nuevas características o mejoras de forma mucho más fácil y rápida.

3.1.3 Mocking y Faking

Consiste en utilizar herramientas que implementan una interfaz y permiten comprobar el comportamiento esperado del sistema al utilizar dicha interfaz y aislar los componentes de un sistema complejo al reemplazarlos por herramientas controladas por el desarrollador. La principal diferencia entre un mock y un fake es el nivel de detalle de la implementación, el primero suele devolver un valor predefinido cuando se utiliza alguna de sus funciones, mientras que el segundo, crea una implementación parcial de la lógica del componente real y permite probar interacciones más complejas.

3.2 Herramientas

3.2.1 Git

Es un software de control de versiones de código abierto desarrollado por parte del equipo de desarrollo de Linux ampliamente utilizado en el sector. Entre sus ventajas permite el desarrollo no lineal de un proyecto, su eficiencia y su enfoque distribuido. [23] Además, gracias a proyectos como GitHub o GitLab, su funcionalidad y facilidad de uso se han visto extendidas.

3.2.2 GitHub

Es una plataforma diseñada para almacenar, compartir y trabajar con otros usuarios para escribir código. [24] Utiliza git para manejar los cambios en el código y permite sincronizar el repositorio git local de cada usuario con la copia de la plataforma.

Además, ofrece un conjunto de funciones ideales para manejar proyectos:

- GitHub Projects: Permite organizar con precisión las tareas a realizar en un proyecto, sincronizando issues, pull requests, etc, con un Kanban para trabajar.
- GitHub Actions: Permite ejecutar acciones en contenedores Docker en la nube de GitHub al realizar ciertas acciones con el código, como subir los cambios a GitHub, o realizar una pull request. Esto permite montar entornos de CI/CD (Integración y Entrega continua) al poder ejecutar tests automáticos, compilar y analizar el código de forma automática.

3.2.3 Clang-tools

Son una serie de herramientas que utilizan Clang para proveer funcionalidades relacionadas con el análisis de código escrito en C o C++. [25] Las herramientas concretas utilizadas en este proyecto son:

- Clang-tidy: Es un analizador de código (linter) especializado para C y C++ open source que detecta patrones propensos de contener errores, tener un mal rendimiento o contener problemas de portabilidad y mantenibilidad.
- Clang-format: Es una herramienta que detecta y corrige el formato del código analizado de acuerdo con una especificación configurable. Esto permite mantener un código con un estilo homogéneo.

Ambas herramientas se pueden utilizar en integración continua para garantizar la calidad del código desarrollado.

3.2.4 CMake

Es un software que gestiona la automatización de la construcción de software de forma independiente del compilador utilizado. Esto permite crear programas compuestos por varios módulos, librerías y ejecutables desde el mismo sistema de configuración. Además, es el estándar de facto en el desarrollo de programas escritos en C/C++. [26]

3.2.5 SonarQube
Es un sistema de análisis de código muy potente que analiza y reporta problemas en el código fuente de un proyecto, junto con ayudas para solucionarlos. Es una de las herramientas más utilizadas en las empresas que desarrollan software por su capacidad de análisis y la sencillez de sus explicaciones. Además, se puede utilizar en integración continua e incluir otras métricas como la cobertura de código por tests unitarios, etc. [27]

3.2.6 GoogleTest
Es un marco de pruebas para C++ de código abierto desarrollado por Google. Está basado en xUnit y ofrece una forma sencilla de ejecutar tests de forma automática. [28]

3.2.7 QEMU
Es un programa de emulación de procesadores de software libre que permite emular los procesadores de la familia ESP32 entre otros, permitiendo probar en integración continua y en local componentes de software que por su dependencia de código especifico de ESP32 no se podría probar de otra forma sin hardware real.

3.2.8 Espressif IoT Development Framework (ESP-IDF)
Es un kit de desarrollo de software para microcontroladores de Espressif para C y C++ que utiliza CMake como generador de configuración de compilación. Proporciona la toolchain para poder desarrollar en la familia de microcontroladores ESP32. Entre sus características está el uso del sistema operativo en tiempo real FreeRTOS, uno de los más usados en el campo, y una extensa documentación con guías de desarrollo, la completitud de su API y ejemplos de toda la funcionalidad que se puede conseguir con este sistema. [29]

3.2.9 CLion
Es un potente IDE (Entorno de Desarrollo Integrado, por sus siglas en inglés) para C y C++ desarrollado por JetBrains que cuenta con integración con proyectos de CMake, lo que lo hace ideal para desarrollar con ESP-IDF. Su potente análisis de código, sus capacidades de refactorización, integración con sistemas de testing, depurador integrado, etc. lo hacen una herramienta muy completa con la que desarrollar este proyecto. [30]

3.2.10 Miro
Es una herramienta online que proporciona una pizarra digital en la que poder trabajar para preparar bocetos, diseños y esquemas entre otras muchas funciones [31]. Gran parte del diseño de este proyecto se ha realizado utilizando esta herramienta.

4 Diseño del Sistema de Potabilización

4.1 Objetivos del diseño

El propósito del sistema es conseguir un sistema modular, autónomo y autosuficiente de potabilización de agua a pequeña escala, que se adapte a distintos tipos de aguas y ubicaciones geográficas, minimizando el uso de consumibles y maximizando la eficiencia energética. Su misión es garantizar agua potable sin necesidad de intervención humana.

4.2 Diseño de alto nivel

El sistema está integrado por una serie de componentes:

1. **Electronic Control Unit (ECU):**
   Coordina la acción de los módulos de potabilización, se encarga de la interacción con el usuario, maneja el sistema de actualizaciones y dicta las directrices del módulo de energía.
2. **Módulo de Energía:**
   Se encarga de obtener, transformar y almacenar la energía eléctrica que necesita el sistema. Provee al resto de módulos de información acerca de la energía disponible en el sistema en tiempo real, y permite en caso de emergencia cortar la corriente a parte del sistema.
3. **Módulos de potabilización:**
   Son una serie de módulos que se encargan de eliminar cada uno una serie de contaminantes concretos. El hecho de poder concatenar módulos de potabilización permite adaptar el sistema a distintos tipos de agua contaminada simplemente eligiendo la serie adecuada de estos.

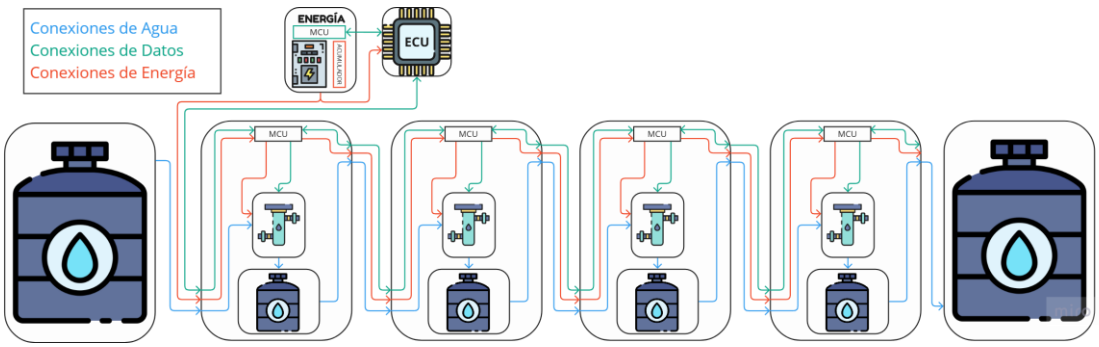

Ilustración 1: Diagrama de alto nivel del sistema de potabilización

Para la electrónica, se ha decidido utilizar microcontroladores de la familia ESP32 desarrollados por Espressif [32] por bajo coste, altas prestaciones y gran soporte por parte de Espressif y la comunidad.

4.3 Detalles del diseño

4.3.1 Energía

El sistema está diseñado para funcionar principalmente con 12V DC, pero soporta componentes que funcionen a 230V AC

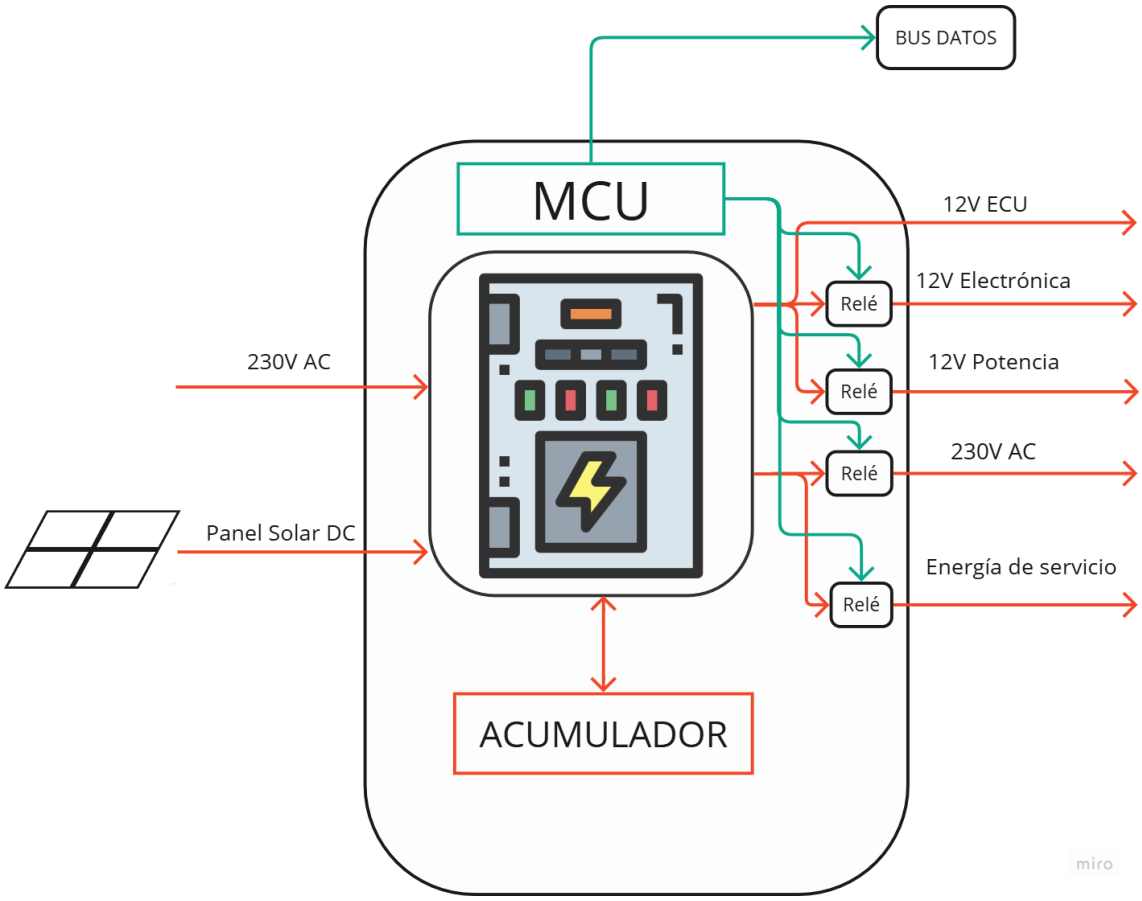

Ilustración 2: Diagrama de alto nivel del módulo de energía

El módulo de energía puede obtenerla de la red eléctrica o de paneles solares o baterías que funcionen con corriente continua. Una vez obtenida, se transforma a los voltajes de salida según la demanda actual y la energía sobrante se acumula. Las diferentes salidas de energía pueden ser controladas mediante peticiones al microcontrolador del módulo.

4.3.2 Comunicaciones

4.3.2.1 CAN

La conexión de datos elegida para este proyecto es el bus CAN [33] (Controller Area Network), un bus de datos desarrollado por BOSCH en 1986, y recogido por la International Organization for Standarization (ISO) en las normas ISO 11898-1 / 11898-6

Sus características más relevantes son:

- Priorización de mensajes
- Recepción multicast
- Robustez frente a inconsistencias de datos
- Sistema multimaster
- Detección de errores temporales y permanentes de los nodos de la red
- Desconexión automática de nodos defectuosos
- Retransmisión automática de tramas erróneas.

CAN tiene varias limitaciones importantes:

- No es una red, no existe direccionamiento de nodos.
- En su versión 2.0B (la utilizada en este proyecto), la cantidad máxima de datos por trama es de 8 bytes, lo que limita las comunicaciones de grandes conjuntos de datos.

CAN solo proporciona las 2 primeras capas del modelo OSI: la capa física, y la de enlace de datos, por lo que se puede extender su funcionalidad en capas superiores.

4.3.2.2 ISO 15765-2 (ISO-TP)

Proporciona la capa de red y de transporte (OSI 3 y 4) sobre el bus CAN para permitir el direccionamiento físico (unicast) y funcional (multicast) y la segmentación de mensajes, resolviendo las mayores limitaciones del bus [34]. En nuestro proyecto se utiliza ISO-TP configurado en “Normal fixed addressing”, esto es, que los parámetros del direccionamiento se localizan en lugares específicos dentro del CAN ID (el identificador de mensaje CAN).

4.3.2.3 PmwComms

Sobre ISO-TP, se sitúa PmwComms, la capa de aplicación de la pila de comunicaciones CAN que se utiliza en este proyecto. Este protocolo ha sido diseñado desde cero para cubrir las necesidades específicas del proyecto, entre ellas, un mecanismo para detectar si un nodo se ha reiniciado inesperadamente, mediante el uso de un valor aleatorio generado por cada nodo al iniciarse, y que se incrusta en todos los mensajes del protocolo. También soporta tanto el esquema petición respuesta 1-1, como el envío de mensajes 1-N.

4.3.2.4 PmwAPI

Se ha diseñado una API sobre PmwComms para manejar la interacción entre los módulos. Cada tipo de módulo expone una serie de acciones que se detallan a continuación, junto con los datos asociados a estas:

4.3.2.4.1 ECU

- Objeto NotificationInfo
  Contiene toda la información relacionada con un evento de notificación.
  - DiscovererModuleAddress (string)
  - TargetModuleAddress (string)
  - NotificationCode (int)
  - DescriptionParams (json string)
    - NotificationResponseID (int) (opcional)
    - …
- Método raiseNotification
  Entrega un evento de notificación para su procesamiento.
  - Tipo de mensaje: unicast
  - Parámetros de la petición:
    - Notification (NotificationInfo)
  - Parámetros de la respuesta:
    - responseParams (json string) (optional)
      - NotificationResponseID (int)
      - …
- Método deviceDiscovery
  Solicita a todos los módulos conectados que envíen un mensaje de tipo deviceDiscoveryResponse.
  - Tipo de mensaje: multicast
  - Parámetros de la petición: N/A
  - Parámetros de la respuesta: N/A
- Método deviceDiscoveryResponse
  Envía un mensaje de tipo deviceDiscoveryResponse
  - Tipo de mensaje: multicast
  - Parámetros de la petición: N/A
  - Parámetros de la respuesta: N/A
- Método setModuleSessionIDVerification
  Configura la verificación de moduleSessionID en la recepción de los mensajes de PmwComms.
  - Tipo de mensaje: multicast
  - Parámetros de la petición:
    - enabled (bool)
  - Parámetros de la respuesta: N/A

4.3.2.4.2 Energía

- Método deviceDiscovery
  Solicita a todos los módulos conectados que envíen un mensaje de tipo deviceDiscoveryResponse.
  - Tipo de mensaje: multicast
  - Parámetros de la petición: N/A
  - Parámetros de la respuesta: N/A
- Método deviceDiscoveryResponse
  Envía un mensaje de tipo deviceDiscoveryResponse
  - Tipo de mensaje: multicast
  - Parámetros de la petición: N/A
  - Parámetros de la respuesta: N/A
- Método setModuleSessionIDVerification
  Configura la verificación de moduleSessionID en la recepción de los mensajes de PmwComms.
  - Tipo de mensaje: multicast
  - Parámetros de la petición:
    - enabled (bool)
  - Parámetros de la respuesta: N/A

- Enumeración UpdateStatus
  Contiene el estado actual de la actualización.
  - SUCCESS
  - IN_PROGRESS
  - NOT_STARTED_ERROR
  - ERROR
- Método getModuleInfo
  Solicita información del módulo.
  - Tipo de mensaje: unicast
  - Parámetros de la petición: N/A
  - Parámetros de la respuesta:
    - swVersion (string)
    - apiVersion (string)
    - hwVersion (string)
    - serialNumber (string)
    - moduleName (string)
- Método getModuleUpdateMaxSizePerChunk
  Solicita el tamaño máximo de mensaje para transmitir un fragmento de una actualización de software.
  - Tipo de mensaje: unicast
  - Parámetros de la petición: N/A
  - Parámetros de la respuesta:
    - size (int)

- Método startModuleUpdate
  Inicia una actualización de software.
  - Tipo de mensaje: unicast
  - Parámetros de la petición:
    - updateDataHash (string)
  - Parámetros de la respuesta:
    - statusCode (UpdateStatus)
- Método sendModuleUpdateChunk
  Envía un fragmento de la actualización de software.
  - Tipo de mensaje: unicast
  - Parámetros de la petición:
    - chunkNumber (int)
    - chunkSize (int)
    - chunkData (binary)
  - Parámetros de la respuesta:
    - statusCode (UpdateStatus)
- Método endModuleUpdate
  Finaliza y aplica o descarta una actualización de software.
  - Tipo de mensaje: unicast
  - Parámetros de la petición:
    - abort (bool)
  - Parámetros de la respuesta:
    - statusCode (UpdateStatus)

- Método restartModule
  Reinicia el software del módulo.
  - Tipo de mensaje: unicast
  - Parámetros de la petición: N/A
  - Parámetros de la respuesta: N/A
- Método restartAllModules
  Reinicia el software de todos los módulos.
  - Tipo de mensaje: multicast
  - Parámetros de la petición: N/A
  - Parámetros de la respuesta: N/A

- Método getStatistics
  Obtiene las estadísticas de uso de energía.
  - Tipo de mensaje: unicast
  - Parámetros de la petición: N/A
  - Parámetros de la respuesta:
    - usedPower (real) (W)
    - batteryVoltage (real) (V)
    - batteryLevel (real) (%)
- Método getAvailablePower
  Obtiene la potencia disponible en este instante.
  - Tipo de mensaje: unicast
  - Parámetros de la petición: N/A
  - Parámetros de la respuesta:
    - availablePower (real) (W)

- Método getServiceOutletUsedPower
  Obtiene la potencia utilizada por la toma de servicio.
  - Tipo de mensaje: unicast
  - Parámetros de la petición: N/A
  - Parámetros de la respuesta:
    - usedPower (real) (W)
- Método controlServiceOutlet
  Conecta o desconecta la toma de servicio.
  - Tipo de mensaje: unicast
  - Parámetros de la petición:
    - enabled (bool)
  - Parámetros de la respuesta:
    - success (bool)
- Método requestAC
  Solicita o libera el uso de la toma de corriente AC.
  - Tipo de mensaje: unicast
  - Parámetros de la petición:
    - enabled (bool)
  - Parámetros de la respuesta:
    - isEnabled (bool)
- Método requestPowerDC
  Solicita o libera el uso de la toma de corriente DC de potencia.
  - Tipo de mensaje: unicast
  - Parámetros de la petición:
    - enabled (bool)
  - Parámetros de la respuesta:
    - isEnabled (bool)
- Método controlElectronicsDC
  Conecta o desconecta la toma de corriente DC de electrónica.
  - Tipo de mensaje: unicast
  - Parámetros de la petición:
    - enabled (bool)
  - Parámetros de la respuesta:
    - success (bool)
- Método setMinimumAvailableEnergyAbs
  Configura el umbral de energía para alertar a ECU y configurar el valor de la energía disponible en 0.
  - Tipo de mensaje: unicast
  - Parámetros de la petición:
    - energyValue (real) (J)
  - Parámetros de la respuesta:
    - success (bool)

- Método setMinimumAvailableEnergyRel
  Configura el porcentaje de energía utilizado como umbral para alertar a ECU y configurar el valor de la energía disponible en 0.
  - Tipo de mensaje: unicast
  - Parámetros de la petición:
    - energyPercentage (real) (%)
  - Parámetros de la respuesta:
    - success (bool)

4.3.2.4.3 Módulos de Potabilización

- Método deviceDiscovery
  Solicita a todos los módulos conectados que envíen un mensaje de tipo deviceDiscoveryResponse.
  - Tipo de mensaje: multicast
  - Parámetros de la petición: N/A
  - Parámetros de la respuesta: N/A
- Método deviceDiscoveryResponse
  Envía un mensaje de tipo deviceDiscoveryResponse
  - Tipo de mensaje: multicast
  - Parámetros de la petición: N/A
  - Parámetros de la respuesta: N/A
- Método setModuleSessionIDVerification
  Configura la verificación de moduleSessionID en la recepción de los mensajes de PmwComms.
  - Tipo de mensaje: multicast
  - Parámetros de la petición:
    - enabled (bool)
  - Parámetros de la respuesta: N/A

- Enumeración UpdateStatus
  Contiene el estado actual de la actualización.
  - SUCCESS
  - IN_PROGRESS
  - NOT_STARTED_ERROR
  - ERROR
- Método getModuleInfo
  Solicita información del módulo.
  - Tipo de mensaje: unicast
  - Parámetros de la petición: N/A
  - Parámetros de la respuesta:
    - swVersion (string)
    - apiVersion (string)
    - hwVersion (string)
    - serialNumber (string)
    - moduleName (string)

- Método getModuleUpdateMaxSizePerChunk
  Solicita el tamaño máximo de mensaje para transmitir un fragmento de una actualización de software.
  - Tipo de mensaje: unicast
  - Parámetros de la petición: N/A
  - Parámetros de la respuesta:
    - size (int)
- Método startModuleUpdate
  Inicia una actualización de software.
  - Tipo de mensaje: unicast
  - Parámetros de la petición:
    - updateDataHash (string)
  - Parámetros de la respuesta:
    - statusCode (UpdateStatus)
- Método sendModuleUpdateChunk
  Envía un fragmento de la actualización de software.
  - Tipo de mensaje: unicast
  - Parámetros de la petición:
    - chunkNumber (int)
    - chunkSize (int)
    - chunkData (binary)
  - Parámetros de la respuesta:
    - statusCode (UpdateStatus)
- Método endModuleUpdate
  Finaliza y aplica o descarta una actualización de software.
  - Tipo de mensaje: unicast
  - Parámetros de la petición:
    - abort (bool)
  - Parámetros de la respuesta:
    - statusCode (UpdateStatus)

- Método restartModule
  Reinicia el software del módulo.
  - Tipo de mensaje: unicast
  - Parámetros de la petición: N/A
  - Parámetros de la respuesta: N/A
- Método restartAllModules
  Reinicia el software de todos los módulos.
  - Tipo de mensaje: multicast
  - Parámetros de la petición: N/A
  - Parámetros de la respuesta: N/A
- Enumeración ModuleStatus
  Contiene toda la información relacionada con el estado interno de un módulo.
  - Starting
  - CannotWork
  - CanWork
  - Working
  - Error

- Método getModuleStatus
  Obtiene el estado interno de un módulo.
  - Tipo de mensaje: unicast
  - Parámetros de la petición: N/A
  - Parámetros de la respuesta:
    - status (ModuleStatus)
- Método getWaterTankLevel
  Obtiene el nivel de agua del depósito interno.
  - Tipo de mensaje: unicast
  - Parámetros de la petición: N/A
  - Parámetros de la respuesta:
    - Level (real) (%)
- Método getPowerNeededToWork
  Obtiene la energía estimada necesaria para funcionar.
  - Tipo de mensaje: unicast
  - Parámetros de la petición: N/A
  - Parámetros de la respuesta:
    - power (real) (W)
- Método getWaterProcessedPerHour
  Obtiene el caudal de salida del módulo.
  - Tipo de mensaje: unicast
  - Parámetros de la petición: N/A
  - Parámetros de la respuesta:
    - flow (real) (L/h)
- Método controlModulePump
  Enciende o apaga la bomba de salida del módulo.
  - Tipo de mensaje: unicast
  - Parámetros de la petición:
    - enabled (bool)
  - Parámetros de la respuesta: N/A
    - success (bool)
- Método isBlockingNextModule
  Reporta si el módulo no permite funcionar al siguiente en este momento.
  - Tipo de mensaje: unicast
  - Parámetros de la petición: N/A
  - Parámetros de la respuesta:
    - blockingState (bool)
- Método setModuleWorkingMode
  Permite a un módulo pasar de CanWork a Working, o de Working a Can(not)Work.
  - Tipo de mensaje: unicast
  - Parámetros de la petición:
    - startWorking (bool)
  - Parámetros de la respuesta:
    - currentStatus (ModuleStatus)

- Método expectPowerOutage
  Guarda el estado interno y configura el hardware del módulo para prepararse para perder energía dentro de unos pocos segundos.
  - Tipo de mensaje: multicast
  - Parámetros de la petición: N/A
  - Parámetros de la respuesta: N/A

4.3.3 Potabilización

Los módulos de potabilización tienen un diseño muy heterogéneo que es intrínseco a su función concreta, es decir, al tipo de contaminante que depure. No obstante, se ha realizado un esfuerzo por homogeneizar su diseño, abstrayendo los componentes comunes, permitiendo construir una interfaz con la que poder interactuar con cualquiera de estos módulos de forma estándar, sin importar las peculiaridades de estos.

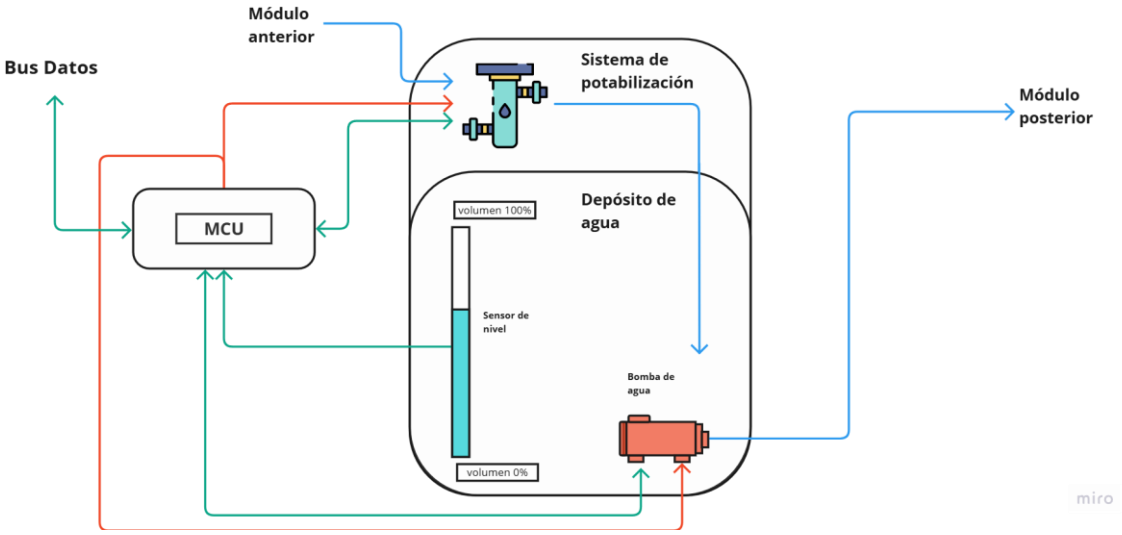

Ilustración 3: Diagrama de alto nivel de un módulo de potabilización

Los módulos de potabilización se ponen en marcha si así lo indica ECU, en base a un algoritmo que maximiza el consumo de la energía disponible y la activación simultanea de la mayor cantidad de módulos de potabilización.

Una vez activado, el módulo pide al anterior que se le suministre agua. Esto solo ocurrirá si el módulo anterior tiene suficiente agua acumulada en su depósito.

Una vez activa la entrada de agua, el módulo realizará su proceso de potabilización concreto, por ejemplo, un filtrado mecánico y dejará la salida de agua depurada en su depósito hasta que este esté lleno o ECU le indique parar.

Si el módulo anterior se queda sin agua, enviará al módulo que está en funcionamiento una petición de parada.

Cuando se le indique parar, el módulo enviará una petición de parada de la bomba al módulo anterior y después parará su sistema de potabilización.

4.3.4 ECU

ECU cumple una serie de funciones de administración del sistema, entre ellas, descubrir y mantener el listado de módulos conectados al sistema, configurar los módulos de energía y potabilización, gestionar cualquier error que trascienda a un módulo concreto, gestionar la interacción con el usuario administrador del sistema y controlar el proceso de arranque y actualización de software de cualquier componente del sistema.

Además de eso, contiene los algoritmos de gestión de los módulos de potabilización, controlando la puesta en marcha y parada de cada módulo, y dejando que estos realicen sus actividades concretas de potabilización.

4.3.5 Arquitectura de Software

Para realizar este sistema, se han contemplado los siguientes componentes de software:

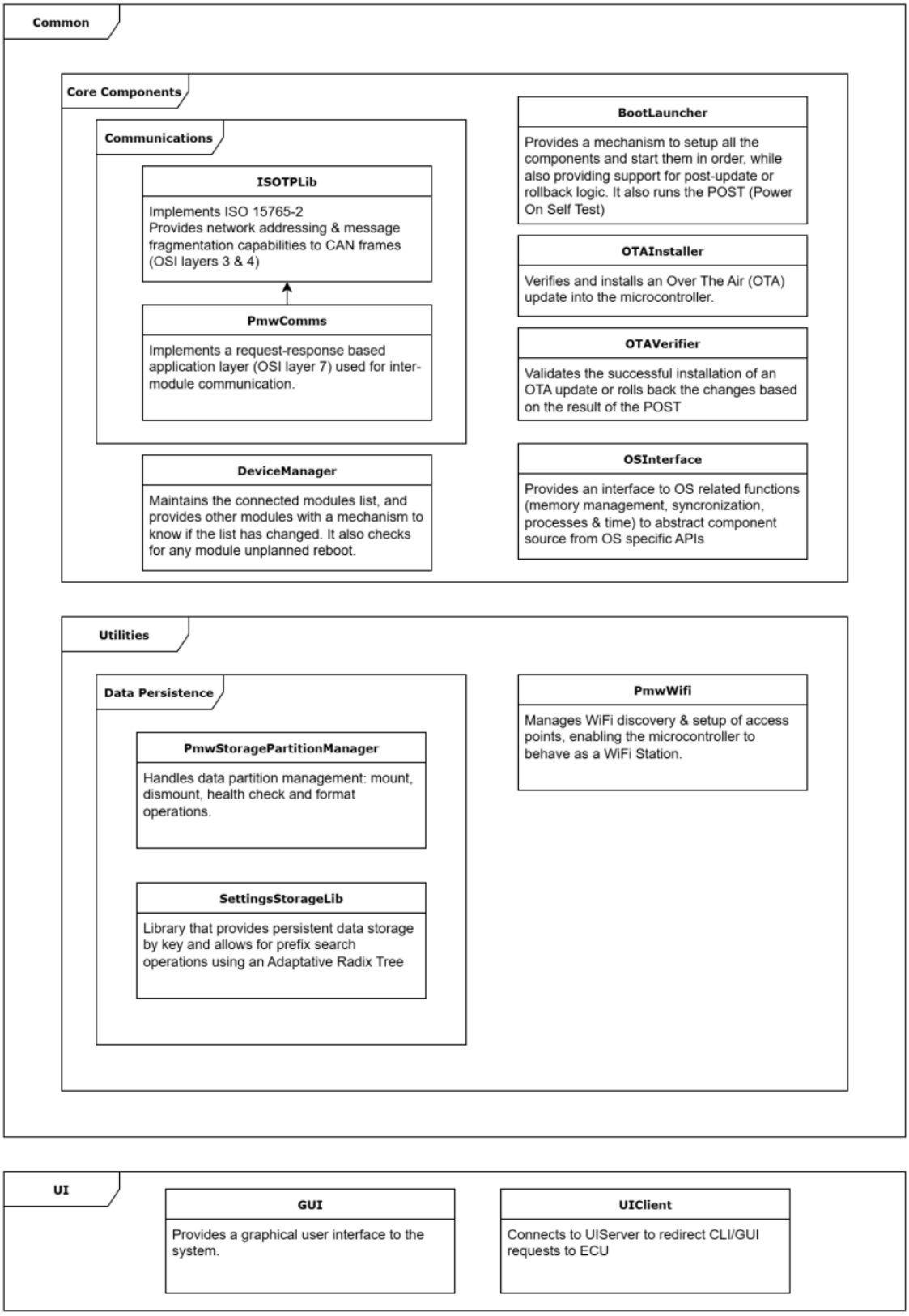

Ilustración 4: Componentes software del proyecto 1, Common y UI

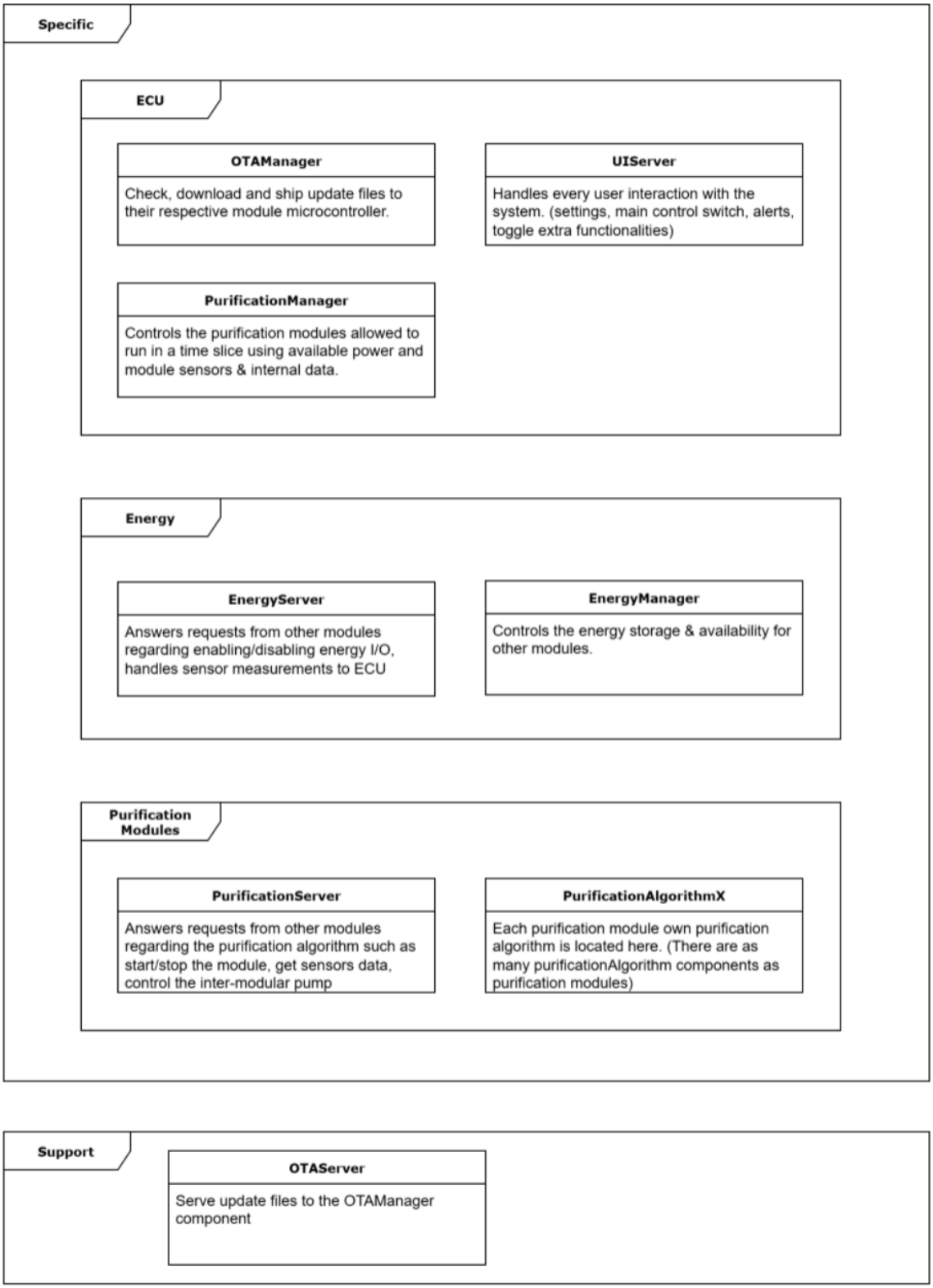

Ilustración 5: Componentes software del proyecto 2, Specific y Support

Estos componentes están agrupados según su función y localización.

Los componentes de la sección *Common* agrupan todo el software embebido en todos los microcontroladores del sistema e incluyen los sistemas de comunicaciones, gestión de arranque, librerías de sistema operativo y persistencia de datos.

Los componentes de la sección *Specific* contienen los sistemas específicos a cada módulo, ECU, energía y potabilización.

En la sección *Support*, se encuentra el servidor de actualizaciones de software inalámbricas del sistema, y en un futuro se incluirá en esa sección cualquier desarrollo de software que no se ejecute en los microcontroladores del sistema.

Existe una sección más, que se encuentra pendiente de validación por parte del equipo de desarrollo del proyecto: *UI*, que contiene la interfaz de usuario y se encuentra pendiente de concretar el hardware a utilizar para la interfaz, para poder decidir si utilizar un microcontrolador específico para la interfaz, o incluirlo dentro de ECU.

5 Diseño y Desarrollo de la Unidad de Control

5.1 Introducción

La ECU utilizará un microcontrolador (MCU) de la familia ESP32 aún por decidir, no obstante, las capacidades generales del modelo escogido son:

- ≥4MB de memoria flash
- ≥4MB de memoria RAM
- Posibilidad de conexión a una red Wifi (Modo Station)
- Capacidad de actualización del software de manera remota (OTA)
- Capacidad de conectar un transmisor CAN
- Capacidad de ejecutar FreeRTOS

Para desarrollar el software necesario, no se necesita conocer el modelo exacto, ya que Espressif pone a disposición de los desarrolladores un framework compatible con gran parte de su gama de MCUs llamado Espressif IoT Development Framework (ESP-IDF). [29]

Este framework permite desarrollar aplicaciones y librerías en C/C++ que serán ejecutadas por el sistema operativo en tiempo real FreeRTOS. Además, ofrece numerosas librerías para facilitar el desarrollo de estas, como logging, Wifi, pila TCP/IP, OTA, CAN entre otras.

Aun así, el objetivo de desarrollo de software es generar código lo más desacoplado posible entre sí, pero también con el hardware donde se ejecute. Por ello se pueden distinguir dos tipos de componentes, los independientes del hardware, es decir, aquellos que se pueden compilar en cualquier arquitectura utilizando un compilador compatible, como G++, y los dependientes de librerías del framework, que solo pueden ser compilados en plataformas compatibles con ESP-IDF.

De todos los componentes software utilizados en este proyecto (Ilustración 4, Ilustración 5), para desarrollar la ECU son necesarios los siguientes:

1. OSInterface
2. PmwWifi
3. PmwStoragePartitionManager
4. SettingsStorageLib
5. ISOTPLib
6. PmwComms
7. DeviceManager
8. BootLauncher
9. OTAInstaller
10. OTAVerifier
11. OTAManager
12. PurificationManager
13. UIServer

Por la cantidad y complejidad de este desarrollo, no se han podido diseñar y desarrollar todos los componentes en el marco de este Trabajo Fin de Grado, por lo que se describirá el estado actual de estos componentes.

5.2 Componentes

5.2.1 OSInterface

5.2.1.1 Diseño

Este componente software está compuesto por una interfaz que proporciona funcionalidad dependiente del sistema operativo (memoria dinámica, sincronización, logging y operaciones relacionadas con el tiempo) y sus dos implementaciones: una diseñada para ejecutarse con ESP-IDF, y la otra diseñada para funcionar en Linux.

Este componente es clave para independizar el resto de los componentes, del sistema operativo utilizado, lo que permite construir y probar componentes fuera de un microcontrolador real o emulado, y hacerlo directamente en la máquina de desarrollo.

Este componente contiene las siguientes funcionalidades:

- Conseguir el número de milisegundos transcurridos desde el inicio del microcontrolador.
- Suspender la ejecución del programa durante el número de milisegundos especificados.
- Asignar memoria dinámica.
- Liberar memoria dinámica.
- Ejecutar una función en un hilo nuevo.
- Crear un mutex nuevo.
- Reservar y liberar un mutex.
- Crear un semáforo binario nuevo.
- Reservar y liberar un semáforo binario.
- Escribir un mensaje con un nivel y etiqueta concretos en el registro de logs.
- Modificar el nivel de log mínimo a guardar para una etiqueta concreta.

5.2.1.2 Desarrollo

El componente contiene una interfaz, de mismo nombre que el componente (OSInterface), que se utiliza en otros componentes, junto con las interfaces de mutex y semáforo binario. El desarrollo se completa con dos implementaciones de las interfaces, permitiendo la ejecución de componentes que utilicen OSInterface en Linux y ESP-IDF (FreeRTOS).

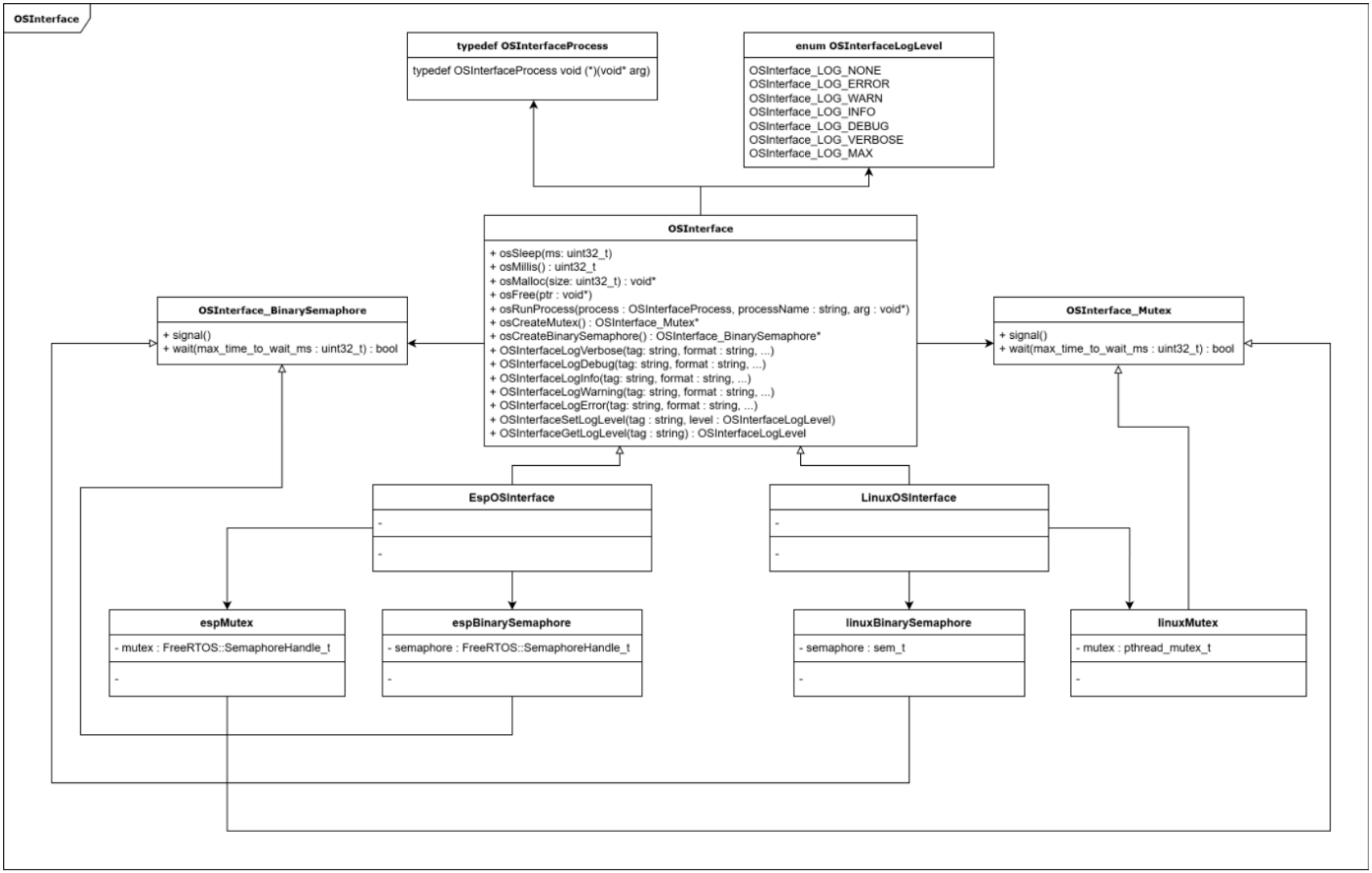

Ilustración 6: Diagrama de clases de OSInterface

Para garantizar que ambas implementaciones poseen el mismo comportamiento, se han desarrollado una serie de test unitarios que se ejecutan en ambas plataformas.

5.2.2 PmwWifi

Este componente depende de ESP-IDF para funcionar.

5.2.2.1 Diseño

Este componente se utiliza para realizar la búsqueda y conexión de redes Wifi, con ello, el objetivo es poder tener una conexión a internet que permita la búsqueda e instalación de actualizaciones de software vía OTA.

Su diseño es muy sencillo, ya que ESP-IDF dispone de una librería para el control del módulo wifi de sus microcontroladores. Solamente nos interesa poder buscar redes cercanas, y poder conectarnos a ellas introduciendo el SSID y la clave de acceso correspondientes, y el objetivo es abstraer toda la configuración pertinente de la librería del framework, para que el usuario de este componente no tenga que configurar nada para poder conectarse a una red.

5.2.2.2 Desarrollo

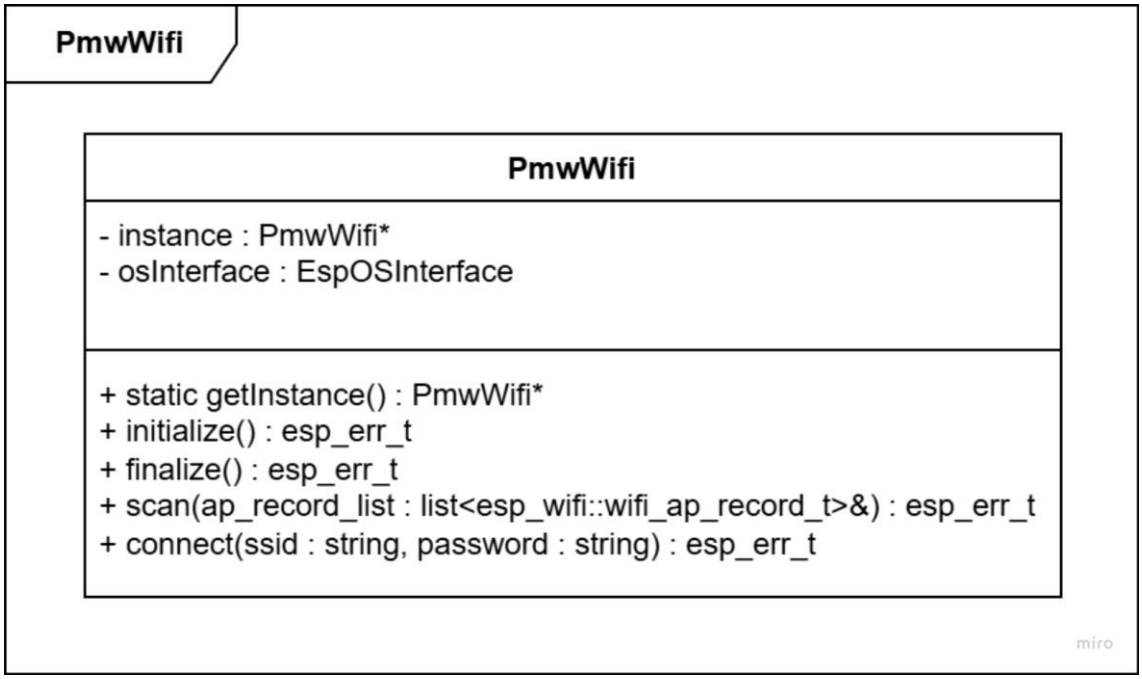

Ilustración 7:Diagrama de clases de PMW_Wifi

Este componente está compuesto por una clase singleton, el constructor es privado, y para acceder al único objeto se utiliza el método getInstance.

Una vez se accede al objeto, el usuario puede configurar e iniciar el modem Wifi, parar y liberar todos los recursos asociados, escanear las inmediaciones en busca de redes disponibles y conectarse a una red.

Una vez realizada la conexión, de forma automática se asigna una dirección IP al microcontrolador, que podrá utilizar las funciones de red de la pila TCP/IP proporcionada por el fabricante.

5.2.3 PmwStoragePartitionManager

Este componente depende de ESP-IDF para funcionar.

5.2.3.1 Diseño

Para el correcto funcionamiento del sistema, es necesario disponer de una partición de datos, donde se puedan guardar recursos necesarios para la ejecución del sistema, como archivos de configuración, recursos gráficos, archivos de localización, etc.

Como la memoria flash del microcontrolador se encuentra en una serie de chips conectados por SPI (SPI NOR flash), para guardar la información se ha decidido utilizar el sistema de archivos de código abierto SPIFFS (SPI Flash File System), diseñado para su uso en dispositivos con bajos recursos, con soporte para wear levelling, resistencia frente a fallos de energía en escritura y capacidad de restauración de integridad. [35]

ESP-IDF posee soporte para integrar este sistema de archivos en sus microcontroladores con facilidad.

Este componente se divide en dos partes:

1. La configuración de CMake y Kconfig:
   Siguiendo la documentación de desarrollo de ESP-IDF, se puede configurar desde Kconfig (el sistema de configuración de funcionalidades a compilar de ESP-IDF, basado en el programa del mismo nombre utilizado para configurar la compilación del kernel de Linux), una lista de particiones en las que dividir la memoria del microcontrolador.

   Esto es muy útil para asignar varias particiones de código ejecutable para poder realizar actualizaciones OTA, pero también permite asignar otras particiones a datos.

   Después, desde el archivo de CMake del componente, se puede rellenar la partición de datos a partir de los archivos guardados dentro de un directorio especifico de la máquina de desarrollo. CMake se encargará de formatear la partición, escribir los datos en ella y combinarla con la imagen de memoria que se cargará en el microcontrolador.

2. El código fuente del microcontrolador:
   Desde el lado del microcontrolador, para poder acceder a los datos almacenados se debe de montar la partición. ESP-IDF proporciona librerías que se encargan de la mayor parte de la lógica del montaje, por lo que el trabajo de este componente se relega al manejo de errores y provee una interfaz de alto nivel, con la funcionalidad de montar/desmontar la partición, junto con comprobar y restaurar la integridad y formatear la partición.

5.2.3.2 Desarrollo

Para preparar las particiones, se debe crear un archivo con la siguiente estructura:

```
# Name,   Type, SubType, Offset,  Size, Flags
nvs,      data, nvs,     0x9000,  0x4000,
otadata,  data, ota,     0xd000,  0x2000,
factory,  app,  factory, ,  2M,
ota_0,    app,  ota_0,   , 2M,
ota_1,    app,  ota_1,   , 2M,
storage,  data, spiffs, , 1M,
```

Ilustración 8: Tabla de particiones de ECU, 8MB

En ella se escriben el numero e información de las particiones. El siguiente paso es configurar CMake para generar la partición a partir de los datos guardados en una carpeta especifica:

```
idf_component_register(SRCS "StoragePartitionManager.cpp"
                       INCLUDE_DIRS "include"
                       REQUIRES spiffs)

spiffs_create_partition_image(storage "./storage_spiffs_root" FLASH_IN_PROJECT)
```

Ilustración 9: CMakelists.txt de PmwStoragePartitionManager

CMake asocia la partición a rellenar por el nombre (storage), y crea la imagen de la partición a partir de los contenidos ubicados en "./storage_spiffs_root". Por último, se puede configurar si la imagen de la partición se debe subir al microcontrolador junto con el resto del código.

En cuanto a la implementación del código que monta la partición en el microcontrolador, nos encontramos frente a una clase singleton que abstrae toda la configuración necesaria por las librerías de ESP-IDF para manejar la partición y provee una interfaz de alto nivel.

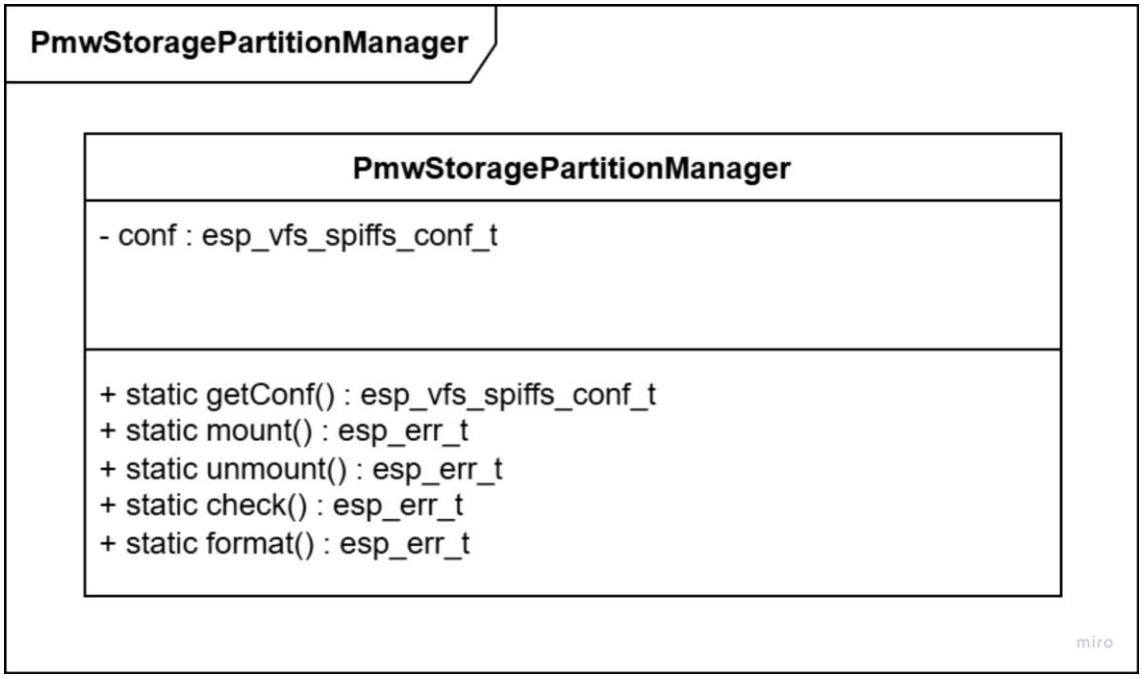

Ilustración 10: Diagrama de clases de PmwStoragePartitionManager

Se proveen funciones para realizar las acciones necesarias con la partición, como montar/desmontar, comprobar y reparar, o formatear la partición. También se permite consultar la configuración interna de la partición.

Dentro de la implementación de la función mount, se ha implementado un sistema de gestión de errores para intentar recuperar la partición si se detecta algún problema con la coherencia del sistema de archivos.

Una vez montada la partición, para acceder a los archivos de esta, se pueden utilizar las funciones estándar de C de manejo de archivos indicando en la ruta el prefijo utilizado para montar la partición ("/storage" por defecto, aunque puede modificarse mediante Kconfig).

5.2.4 SettingsStorageLib

Este componente no depende de librerías de ESP-IDF para funcionar.

5.2.4.1 Diseño

SettingsStorageLib es una librería que tiene como objetivos ser utilizada como sistema de almacenamiento centralizado y persistente de la configuración del sistema. Para ello posee las siguientes capacidades:

- Almacenamiento de pares clave-valor, cuya clave sea una string y cuyo valor sea un entero, un número real o una string.
- Consistencia de la información guardada, al utilizar mecanismos de sincronización.
- Uso eficiente de los recursos del sistema, al utilizar una estructura de datos avanzada (Adaptive Radix Tree [36]).
- Posibilidad de realizar operaciones sobre grupos de claves, utilizando prefijos en las claves para agrupar.
- Posibilidad de restaurar las configuraciones a su valor por defecto por grupos o de forma independiente.
- Persistencia de la configuración en el sistema de archivos.
- Minimización de los errores de desarrolladores, utilizando una interfaz estricta, que separa el registro de nuevas claves, de la modificación de existentes, para evitar que errores de ortografía en las claves pasen desapercibidos.

Un Adaptive Radix Tree (ART) es una optimización de un Radix Tree (RT), es decir una estructura de datos clave-valor cuya clave es un array de bytes y su valor puede ser cualquier estructura, que permite realizar operaciones avanzadas como búsqueda de claves por prefijo de forma eficiente. ART optimiza el uso de recursos de un RT adaptando el tamaño y tipo de nodos y hojas del árbol. El rendimiento teórico de esta estructura se encuentra a la par que un hashmap, pero conservando un orden lexicográfico y permitiendo la búsqueda de claves en *O(N)* siendo N el tamaño de la clave [36].

Se ha decidido utilizar esta estructura de datos para permitir construir una interfaz de usuario en la que exponer la configuración del sistema, el uso de prefijos permite de manera sencilla agrupar configuraciones relacionadas entre ellas, por ejemplo, las claves "/purification/maxWaterPurifiedPerDay" y "/purification/enabled" se pueden agrupar fácilmente en un menú llamado "purification" que cuelgue de la raíz de la interfaz. Esto se puede repetir con múltiples niveles, y para obtener cada submenú, solo hay que iterar sobre el prefijo del menú dentro del ART (en el ejemplo anterior "/purification"). Con esto se evita utilizar estructuras de datos recursivas, como por ejemplo un mapa que pueda guardar como valor otro mapa.

Para poder utilizarse como almacén centralizado, se deben implementar mecanismos de sincronización que eviten condiciones de carrera. El caso que más se adapta a la situación es el problema de los lectores y escritores [37].

Existen varias soluciones generales a este problema, cada una con sus implicaciones. Para esta librería se ha utilizado la solución que da preferencia a los escritores, es decir, se permite que N procesos lean de forma concurrente, pero a la hora de escribir, solo uno puede hacerlo, y para garantizar que no haya inanición, una vez un proceso indica que quiere escribir, se impide a nuevos lectores acceder al recurso compartido. Una vez todos los lectores que estuvieran en el proceso de lectura hayan terminado, se le da paso al escritor, y cuando termina de escribir, se vuelve a permitir la lectura y escritura de otros procesos que hayan sido puestos en espera. No se espera que haya inanición de los procesos lectores ya que en un sistema de configuración es mucho más probable consultar elementos que modificarlos durante la ejecución del programa, por lo que se espera que los tiempos entre escrituras sean suficientemente grandes como para cubrir la demanda de las lecturas.

Además de todo lo anterior, la librería debe poder guardar una copia en memoria persistente de la configuración, garantizando que se pueda restaurar en un futuro tras reiniciarse el sistema.

5.2.4.2 Desarrollo

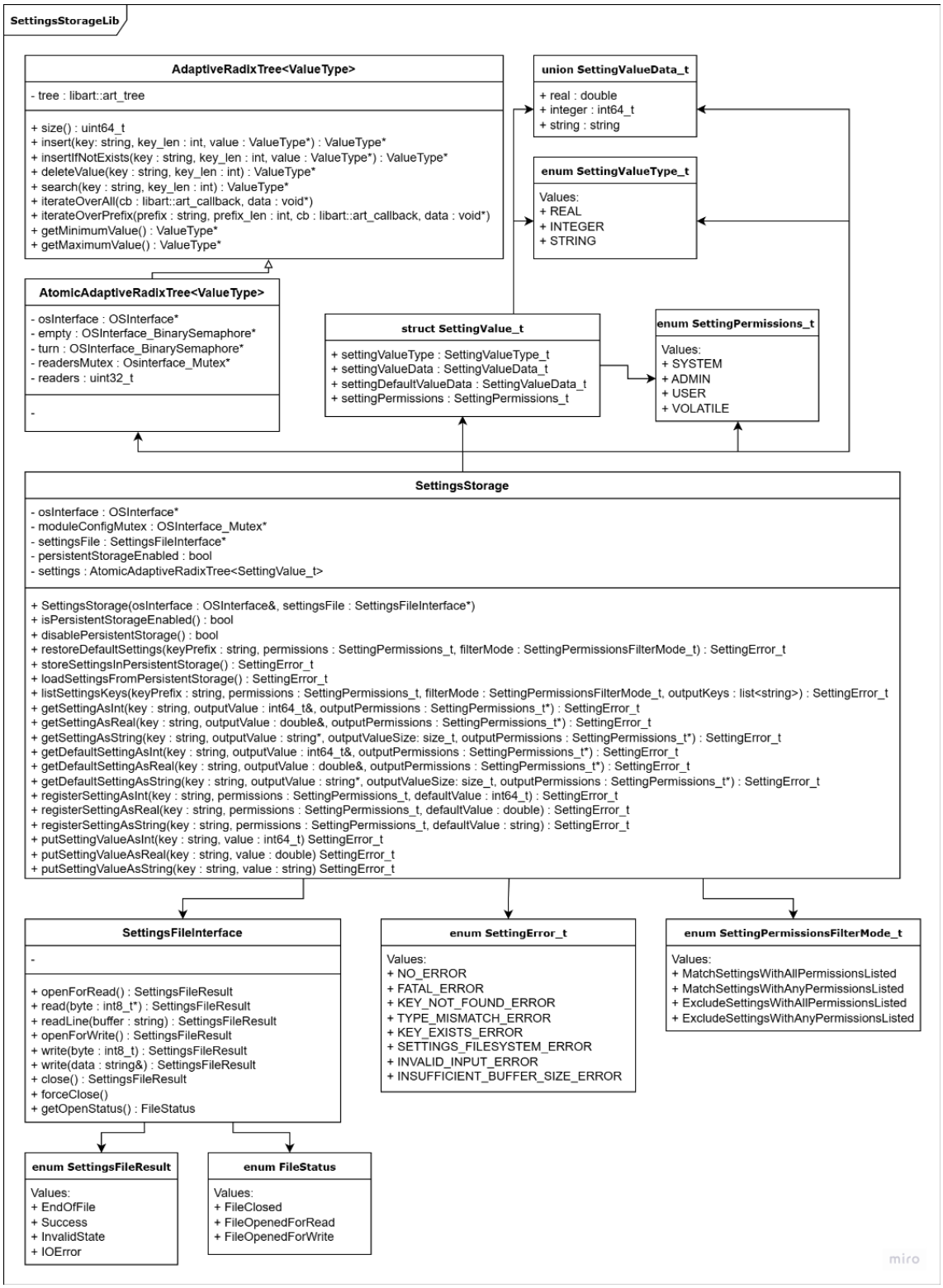

Ilustración 11: Diagrama de clases de SettingsStorageLib

Esta librería utiliza libart [38], una implementación de ART escrita en C99 y que no contiene dependencias, lo que la hace portable y un candidato ideal en proyectos que pueden ser compilados para varias arquitecturas como este.

Sobre esta librería se ha implementado un AdaptiveRadixTree utilizando un template de C++, que permite utilizar libart con un tipado fuerte para evitar errores con el manejo de la memoria.

Para añadir soporte concurrente se ha extendido AdaptiveRadixTree con AtomicAdaptiveRadixTree donde se resuelve el problema de los lectores y escritores.

En cuanto a la implementación del almacén de configuración, se ha creado la clase SettingsStorage que contiene toda la funcionalidad pública de cara a los usuarios.

Para guardar la información, se ha creado un AtomicAdaptiveRadixTree cuyos valores son de tipo SettingValue_t, una estructura que contiene el tipo de dato guardado (real, entero o string), el valor actual, el valor por defecto (para poder restaurar la configuración, como se verá más adelante) y los permisos de visibilidad ese elemento.

La interfaz que exporta SettingsStorage permite registrar un nuevo elemento de configuración, es decir crear un nuevo par de clave y valor, asignando el valor actual y por defecto (ver registerSettingAsXXX). Este paso debe realizarse durante el arranque del sistema, antes de cargar la configuración almacenada en memoria persistente (si existe).

Una vez registrados los elementos de configuración, la librería puede operar con normalidad, proporcionando métodos para leer (getSettingAsXXX), modificar (putSettingAsXXX) o listar claves que utilicen un prefijo determinado (o todas si el prefijo es vacío) (listSettingsKeys). El listado aplica un filtro de permisos, esto es útil para separar configuraciones internas del sistema, de ajustes que solo deba utilizar el administrador del sistema, o aquellos que puedan ser modificados por cualquier usuario.

Por último, se permite la restauración de los ajustes por defecto, guardados en el momento de registro de cada elemento de configuración. Al igual que el listado, la restauración se puede aplicar a elementos o grupos de elementos concretos, basado en los prefijos de las claves. Además, también se pueden aplicar los filtros de visibilidad para solo afectar a los elementos que cumplan con el criterio dentro de un grupo de claves.

Cuando se modifica un elemento de configuración, solamente la copia en memoria volátil se modifica, lo que significa que tras un reinicio los cambios se pierden, para evitar esto, el usuario de la librería puede utilizar storeSettingsInPersistentStorage para guardar una copia no volátil que poder restaurar más tarde (por ejemplo, tras reiniciar y registrar los elementos de configuración) utilizando loadSettingsFromPersistentStorage.

Para garantizar que el sistema puede seguir funcionando incluso tras un fallo al cargar la configuración (por una corrupción de datos), la librería pasa a funcionar en modo no persistente, garantizando que al menos la copia en memoria con los ajustes por defecto (y las modificaciones realizadas en ejecución) puede ser utilizada. También se puede pasar manualmente a este modo utilizando disablePersistentStorage y consultar el modo de funcionamiento con isPersistentStorageEnabled

Para evitar dependencias del sistema operativo, se ha creado una interfaz (SettingsFileInterface) con una serie de funciones relacionadas con la lectura y escritura de archivos, con la intención de que sea la implementación de esta interfaz la que maneje los detalles del manejo del archivo de respaldo de la configuración. Este archivo se serializa en formato TSV y se añade un checksum (CRC32) para garantizar su integridad.

5.2.5 ISOTPLib

Este componente no depende de librerías de ESP-IDF para funcionar.

Este componente implementa parte del estándar ISO 15765-2. Se ha diseñado e implementado debido a la falta de librerías de código abierto que cumplieran los siguientes requisitos:

- Implementar el modo de direccionamiento “Normal Fixed Addressing” tanto físico como funcional.
- Poder enviar y recibir múltiples mensajes a la vez (con la limitación del protocolo de no poder tener mensajes con mismo Network Address Information (N_AI) [34], es decir, mismo origen, destino y tipo de dirección de destino pertenecientes a distintos mensajes.
- No tener que interactuar como usuario con CAN, es decir, que toda la lógica necesaria para el funcionamiento de ISO-TP esté contenida en la librería o en interfaces.

5.2.5.1 Diseño

ISOTPLib tiene como objetivos poder ser usada independientemente del sistema operativo o implementación CAN, poder funcionar en entornos donde no existe la concurrencia de procesos (con la excepción de interrupciones) y cumplir con el resto de los requisitos mencionados previamente.

El funcionamiento de ISOTPLib es el siguiente:

Se configura la librería con la dirección del nodo local (N_SA), se configuran la lista de direcciones funcionales a las que se quiere suscribir, se indica la memoria dinámica máxima que podrá utilizar la librería y se proporcionan los tres callbacks requeridos por ISO-TP: N_USData.FFIndication (notificación de comienzo de la recepción de un mensaje), N_USData.Indication (recepción de un mensaje) y N_USData.Confirm (notificación de fin del envío de un mensaje). El otro servicio proporcionado por ISO-TP N_USData.Request (envío de un mensaje) se puede utilizar llamando a una función.

Para mantener en funcionamiento la librería, se deberá llamar de forma periódica a una función que será la encargada de ejecutar toda la lógica del protocolo. Al hacerse así, en entornos sin multitarea se puede alternar la ejecución de esa función con el resto de la lógica del programa, y en entornos multitarea, ejecutar en un proceso nuevo un bucle infinito donde se llame a esa función periódicamente.

La librería contiene protecciones contra condiciones de carrera, ya que podrá ejecutarse tanto en entornos secuenciales como concurrentes.

Para independizar la librería de la implementación CAN subyacente, el usuario deberá implementar una interfaz con los comportamientos que se esperan por parte de la librería.

5.2.5.2 Desarrollo

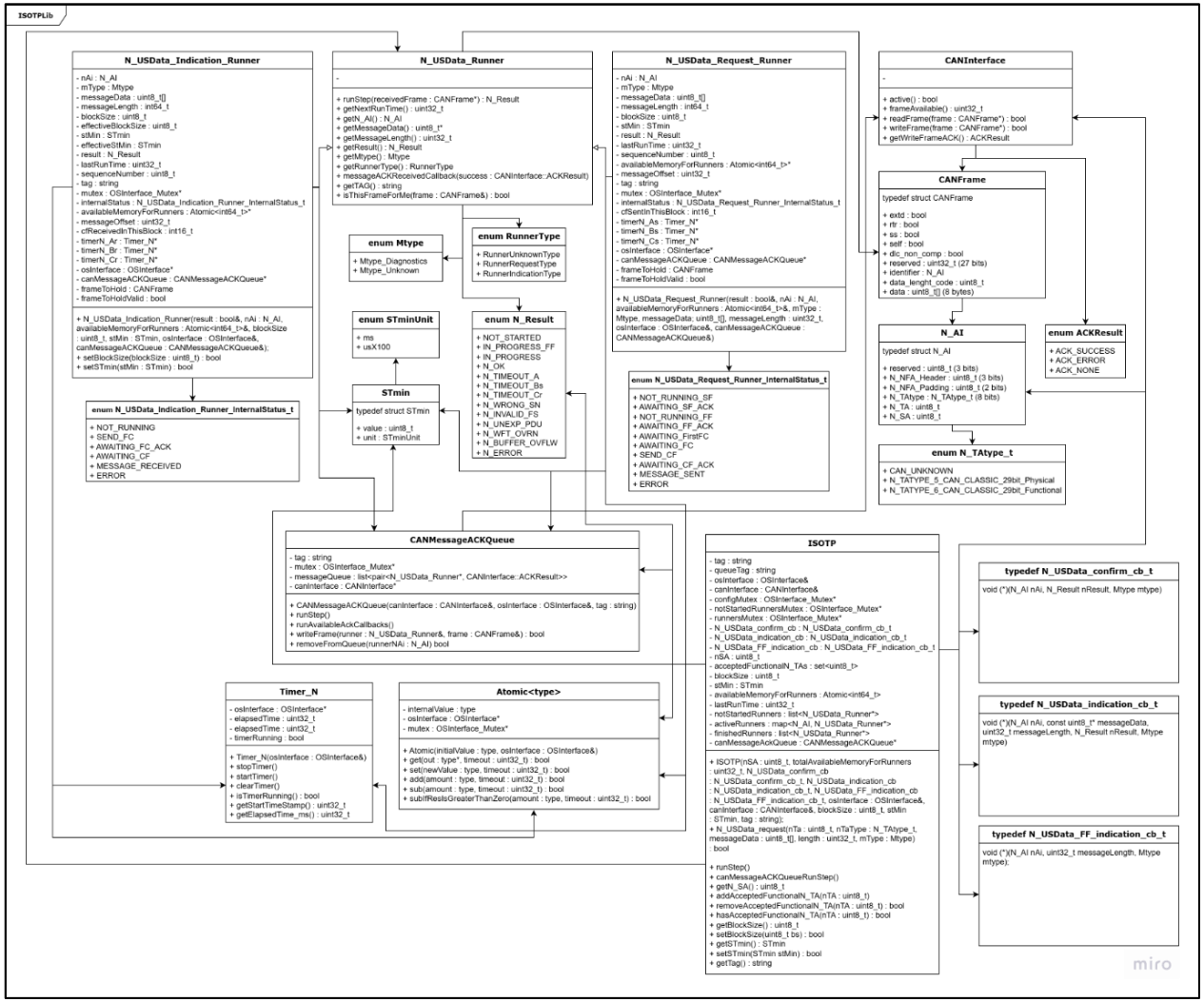

Ilustración 12: Diagrama de clases de ISOTPLib

ISOTPLib está compuesto por la interfaz CANInterface, la clase pública ISOTP, y una serie de clases auxiliares que complementan la funcionalidad.

Al instanciar un objeto de tipo ISOTP, se configura la memoria dinámica máxima que puede utilizarse para ISO-TP, la dirección del nodo ISO-TP (N_SA), los tres callbacks de las notificaciones de envío y recepción de mensajes, la implementación de CANInterface a utilizar, junto a dos parámetros más, el tamaño de bloque (blockSize), que indica cuantas tramas CAN se pueden enviar de forma seguida en mensajes multitrama antes de esperar una trama de la parte receptora, y el tiempo de separación mínimo (STmin), que indica cuanto tiempo debe esperarse entre el envío de dos tramas como mínimo.

Con esta configuración, todos los mensajes de tipo fisico cuya dirección de destino (N_TA) sea la dirección del nodo serán recibidos y puestos a disposición de la capa superior a través del callback correspondiente.

Para poder enviar y recibir mensajes, y teniendo en cuenta que ISO-TP permite el envío de mensajes de una sola trama o multitrama, se debe llamar a dos funciones de forma periódica, idealmente en procesos separados:

- runStep: ejecuta un “paso” en el envío o recepción de los mensajes pendientes.
- canMessageACKQueueRunStep: ejecuta un paso en el sistema que comprueba la confirmación de escritura de las tramas CAN. Esta función es necesaria, y se debe ejecutar lo más seguido posible, porque en muchas implementaciones de controladores CAN, las confirmaciones de escritura se guardan en un registro del circuito especializado que controla el acceso al bus CAN, y, por lo tanto, si no se lee periódicamente, se corre el riesgo de sobrescribir el registro, y perder la confirmación de escritura de ese mensaje.

En una implementación ideal, se debería configurar el microcontrolador para lanzar una interrupción al escribir en el registro del circuito, y llamar a canMessageACKQueueRunStep desde ahí, ya que su objetivo es guardar en una estructura de datos el valor del ACK para utilizarlo más tarde en runStep.

Para enviar un mensaje a otro nodo, se debe utilizar la función N_USData_request con la dirección y el tipo de dirección de envío (N_TA y N_TAtype), el mensaje con su tamaño y el tipo de mensaje (el tipo de mensaje Mtype se utiliza por las capas superiores si se sigue implementando la pila de comunicaciones CAN de la que depende ISO-TP, llamada Diagnostics over CAN o DoCAN. Si no se implementa DoCAN, el tipo de mensaje debe ser siempre Diagnostics).

Además, se pueden configurar durante la ejecución los parámetros STmin y blockSize, y modificar la lista de direcciones funcionales (o multicast) que se deben transmitir a la capa superior.

El funcionamiento interno de ISOTP depende de los objetos de tipo N_USData_Runner. Cada uno de estos objetos representa un mensaje que se encuentra en proceso de envío o recepción, y contienen toda la lógica necesaria para transmitir el mensaje.

Existen dos tipos de N_USData_Runner: N_USData_Request_Runner (envía un mensaje) y N_USData_Indication_Runner (recibe un mensaje).

Cada runner contiene un estado interno, que avanza cada vez que se ejecuta runStep. ISOTP, durante la ejecución de runStep, ejecuta runStep para cada runner si se cumple alguna de las siguientes condiciones: Si ISOTP recibe una trama CAN y la función isThisFrameForMe del runner devuelve true, o si la llamada a getNextRunTime del runner devuelve un timestamp anterior al instante actual. La lógica concreta para comprobar cuando debe ser ejecutado el runner depende del propio runner y funciona acorde a ISO-TP. Los runners señalizan que han terminado de ejecutarse si al ejecutar runStep, devuelven un valor distinto de IN_PROGRESS. Cuando esto ocurre, ISOTP ejecuta el callback correspondiente según el tipo de runner.

El funcionamiento interno de cada runner se describe a continuación:

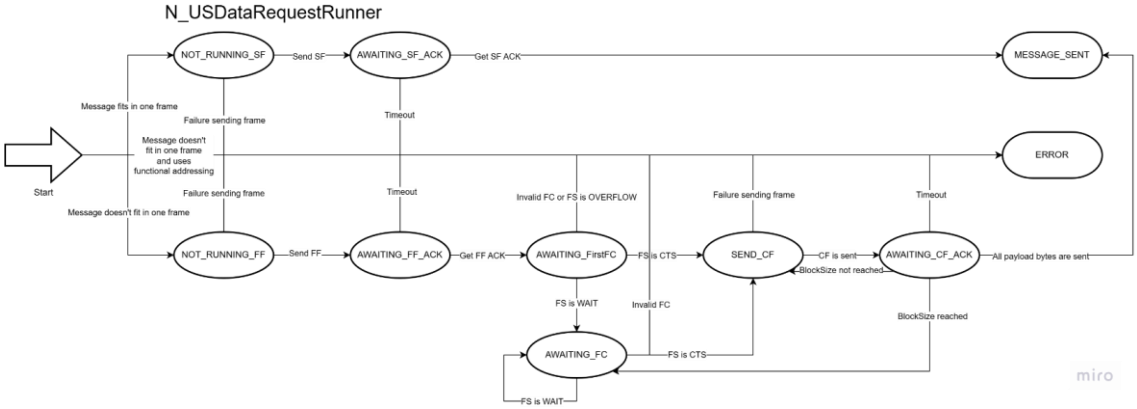

Ilustración 13: Diagrama de estados de un Request Runner

Para un Request Runner, el envío de un mensaje implica procesar el tipo de mensaje en función de su tamaño y modo de direccionamiento primero. Si cabe en un frame, se envía un frame de tipo Single Frame (SF) y cuando se confirme el envío se da por finalizada la transmisión. Si no cabe en un frame, se envía un frame de tipo First Frame (FF) con metadatos del mensaje y los primeros bytes de datos. Una vez confirmado el envío, se espera a recibir un frame de tipo Flow Control (FC) que contiene la información sobre como realizar el envío (STmin y blockSize). Con esa información se envían tantos Consecutive Frame (CF) con datos como indique el blockSize, y se espera a recibir un FC. Esto se repite en bucle hasta completar el envío de los datos.

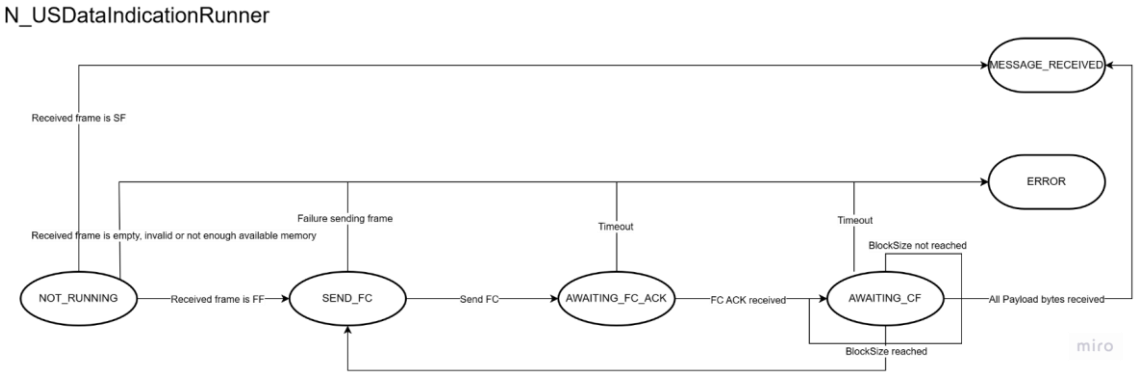

Ilustración 14: Diagrama de estados de un Indication Runner

Para un Indication Runner, al recibir un primer frame, se debe analizar el tipo de este, SF o FF. Si es SF, se decodifica y termina la recepción. Si es FF, se envía un FS con el blockSize y STmin configurados en ISOTPLib y se espera a recibir tantos CF seguidos como indique el blockSize. Después se vuelve a enviar un FC con los valores actuales. Estos pasos se repiten hasta recibir el mensaje completo.

ISOTP mantiene una serie de estructuras de datos en las que guarda los runners que se encuentran en ejecución, los que ya han terminado y los que han sido creados, pero todavía no han empezado a ejecutarse. El paso de una estructura a otra viene dictado por el resultado de ejecutar la función runStep del runner.

Hay pasos de la ejecución de un runner que requieren recibir la confirmación de escritura de una trama, ahí entra en juego la clase CANMessageACKQueue, su objetivo es mantener una cola de runners que han escrito una trama CAN y el valor del ACK correspondiente a cada trama. Durante la ejecución del runStep de ISOTP, se llama a runAvailableAckCallbacks, que ejecuta la función messageACKReceivedCallback de cada runner que tenga el resultado de la confirmación disponible.

La clase Atomic se utiliza para evitar superar el límite de memoria asignada al objeto ISOTP, para ello se mantiene un objeto Atomic que contiene un entero con el número de bytes disponibles para asignar dinámicamente, y utilizando la función subIfResIsGreaterThanZero, cada runner indica cuanta memoria quiere reservar, y si hay suficiente para hacerlo. Esta función, está inspirada por la instrucción test and set de muchas arquitecturas de procesadores. Solo modifica el valor si consigue acceso, o en este caso, si hay suficiente memoria.

Esta memoria se mantiene reservada hasta la destrucción del runner, una vez ejecutado el callback correspondiente.

La clase Timer_N contiene una implementación de un cronometro, y se utiliza para detectar timeouts y actualizar el valor de getNextRunTime de cada runner.

En cuanto a la implementación de la interfaz CANInterface, se necesita implementar las siguientes funciones:

- active: devuelve true si el bus CAN está en funcionamiento.
- frameAvailable: devuelve true si el controlador CAN ha recibido una trama nueva.
- readFrame: lee una trama CAN
- writeFrame: escribe, o pone en cola de escritura una trama CAN
- getWriteFrameACK: devuelve la confirmación de escritura de la última trama si está disponible.

5.2.6 PmwComms

Este componente no depende de librerías de ESP-IDF para funcionar.

5.2.6.1 Diseño

PmwComms se sitúa encima de ISO-TP, en la capa de aplicación del modelo OSI, y proporciona una abstracción de los a los mensajes al proporcionar un esquema cliente-servidor. Se soportan 3 tipos de mensaje: request, response y multicast. Request y multicast son mensajes enviados por clientes y response por servidores.

Para garantizar que un nodo pueda actuar como cliente y servidor a la vez, se utiliza un dispatcher que desvía los mensajes ISO-TP recibidos al destino apropiado, basándose en la cabecera del mensaje.

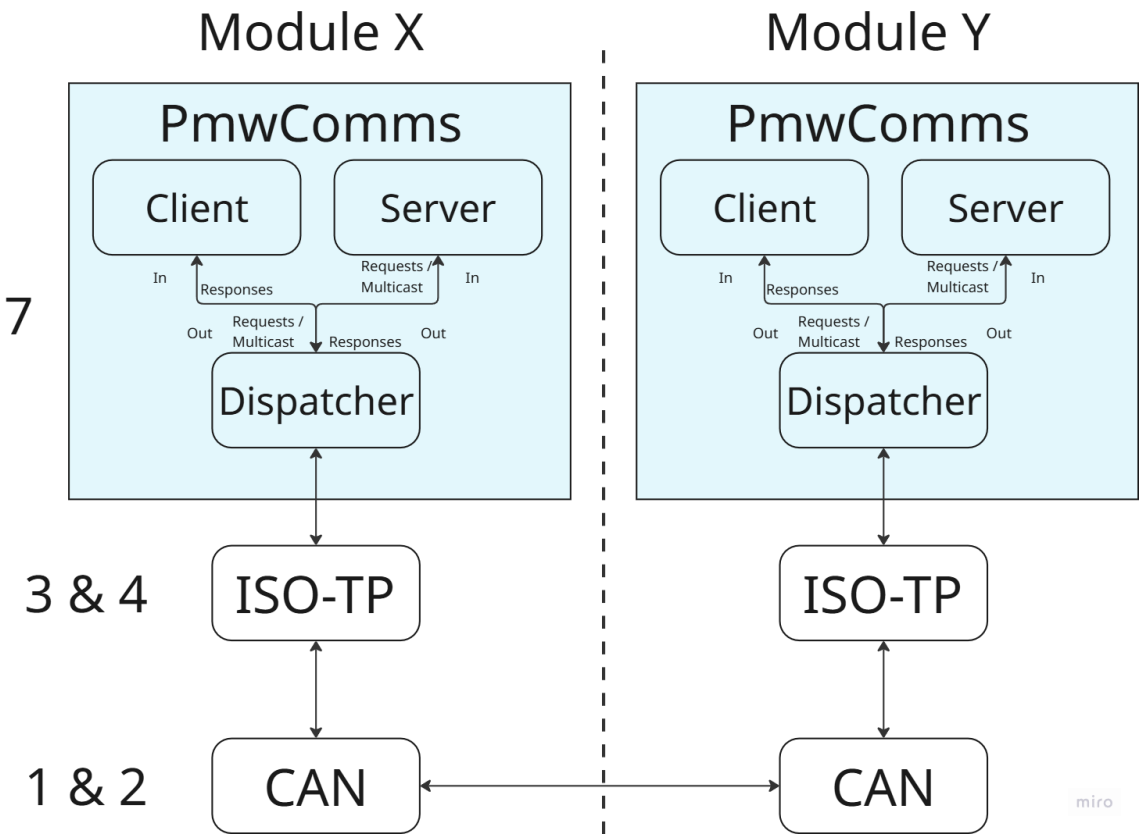

Ilustración 15: Diagrama de la pila de comunicaciones del sistema

Cada tipo de mensaje posee una representación distinta. La diferenciación entre request/response y multicast se realiza comparando los metadatos proporcionados por ISO-TP en busca de un mensaje de tipo funcional, mientras que la diferenciación entre request y response se hace con la RequestFlag.

Los mensajes de tipo request contienen una RequestID que las asocia con una response, y además contienen un OperationID que indica el servicio u operación que se invoca en el servidor.

Los mensajes de tipo response mantienen la RequestID de la request asociada y contienen el StatusCode de la respuesta, además del payload si corresponde.

Los mensajes de tipo multicast tienen un payload máximo de 6 bytes, limitación impuesta por ISO-TP, ya que tiene que caber en una sola trama CAN2.0B de 8 bytes (6 (payload) + 1 (PmwComms header) + 1 (ISO-TP header))

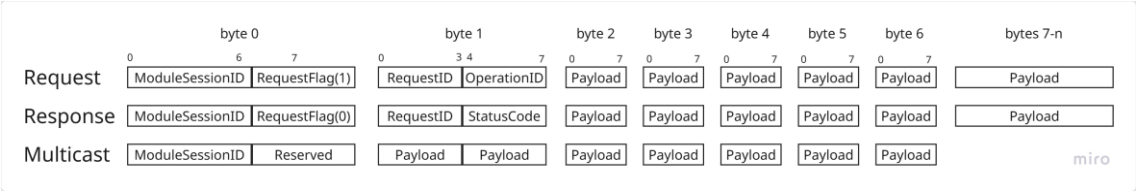

Ilustración 16: Tabla con la forma de cada tipo de mensaje de PmwComms

En el sistema, las direcciones de los módulos son estáticas y preasignadas para cada módulo. Para evitar colisiones con las direcciones de los módulos, PmwComms incorpora un valor generado por cada módulo de forma aleatoria al iniciarse, llamado ModuleSessionID, si se detectan varios mensajes con misma dirección de origen y distinto ModuleSessionID, se lanza un error de comunicaciones. Esta comprobación es válida tanto para comprobar colisiones de direcciones como para detectar reinicios inesperados de los nodos.

5.2.6.2 Desarrollo

Este componente no ha podido ser implementado como parte del Trabajo Fin de Grado, no obstante, su diseño está completo y se presenta a continuación.

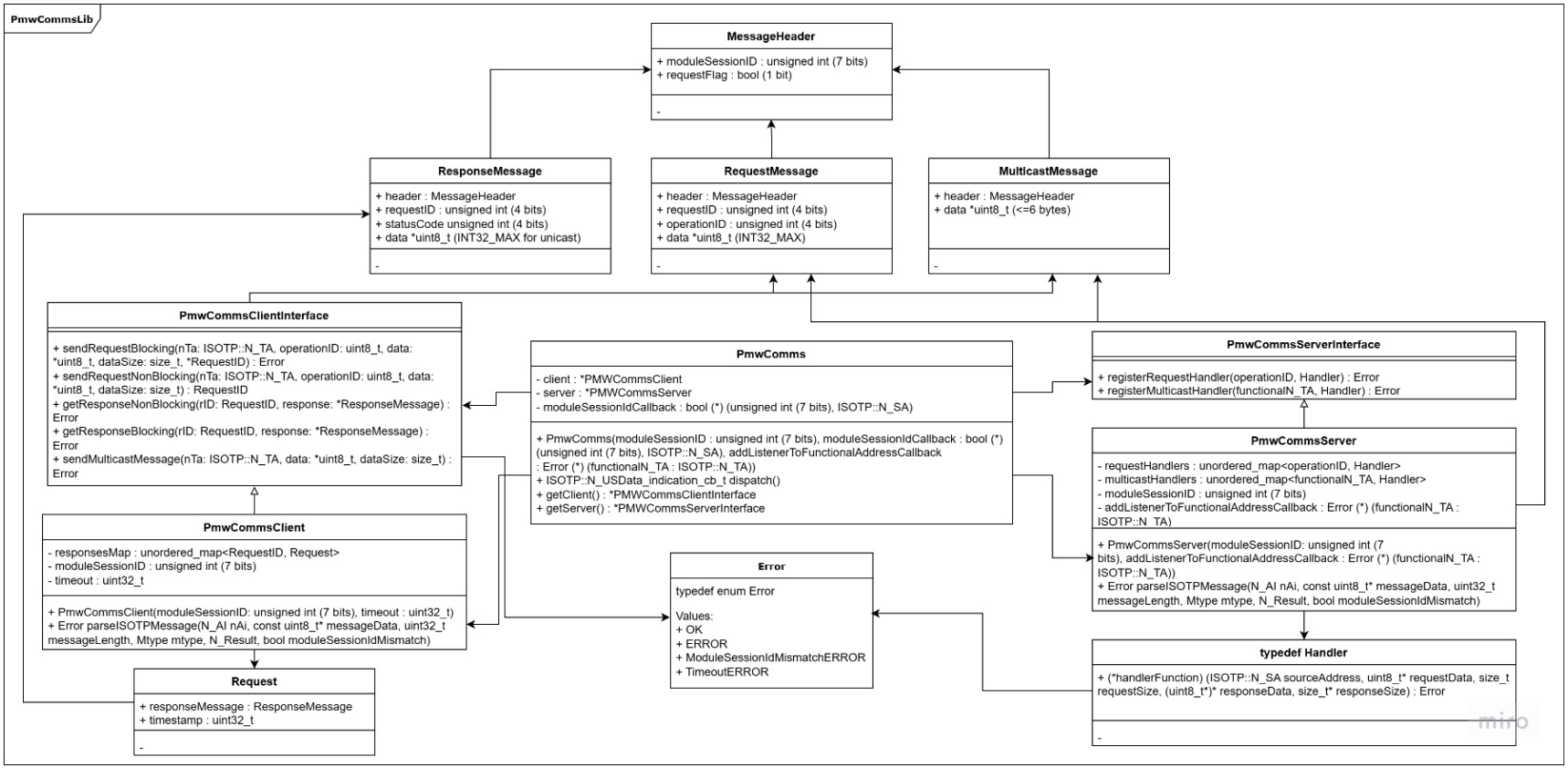

Ilustración 17: Diagrama de clases de PmwComms

Para configurar el componente, se debe construir un objeto de la clase PmwComms indicando el ModuleSessionID del nodo y dos funciones necesarias: la que comprueba que el ModuleSessionID de un mensaje recibido sea válido, y un callback para registrar en ISO-TP direcciones funcionales a las que escuchar.

Una vez construido, se debe registrar la función dispatch en ISO-TP como N_USData.Indication callback.

El objeto de tipo PmwComms pone a disposición un servidor y cliente a través de interfaces que ocultan los métodos privados del componente (métodos públicos a nivel de clase, pero que no deben ser utilizados por el usuario).

El servidor se debe configurar con una serie de handlers, funciones que reciben información de la request, y devuelven una response. Cuando llegue un mensaje cuyo operationID sea igual al del handler registrado, se pasará a esta función el mensaje, y se enviará la respuesta generada. En el caso de mensajes multicast, la respuesta se ignora, ya que, según el protocolo, estos mensajes no van emparejados.

Por otro lado, con el cliente se pueden enviar requests de manera síncrona o asíncrona, y comprobar la recepción de una respuesta de forma asíncrona, o esperar a tener la respuesta (sincrona). También se pueden enviar mensajes multicast.

Cuando se envía una request, se genera una requestID que se utilizará para acceder a la response o consultar el estado de esta.

Cuando un mensaje llega a ISO-TP, se ejecuta la función dispatch de PmwComms, esta se encarga de comprobar que el moduleSessionID del mensaje sea válido y enruta el mensaje al cliente o servidor, en función de la cabecera de este. En cualquier caso, se ejecuta la función parseISOTPMessage del cliente o servidor.

Si el mensaje es para el servidor, se comprueba si es un mensaje multicast o una request, se ejecuta el handler correspondiente, y en el caso de las requests, se genera y envía la response a partir de los datos aportados por el handler.

Si el mensaje es para el cliente (es una response), esta se guardará dentro del cliente hasta que sea consultada o se necesite espacio para guardar una nueva response.

5.2.7 DeviceManager

Este componente no depende de librerías de ESP-IDF para funcionar.

5.2.7.1 Diseño

Para los componentes de PurificationManager, PurificationAlgorithm, PmwComms y OTAManager se necesita conocer información acerca de los módulos conectados, esta información incluye la dirección de red (N_TA), la versión de hardware y software y el moduleSessionID.

Este componente se encarga de mantener la lista de módulos conectados al sistema y su información, guardar la información del módulo local y de responder los mensajes de deviceDiscovery, setModuleSessionIDVerification y getModuleInfo. Además, informa a otros componentes de cambios en los módulos conectados y verifica que un moduleSessionID es válido para una dirección determinada si la verificación está activa.

ECU utiliza una versión modificada de este componente, donde además de responder los mensajes de deviceDiscovery, setModuleSessionIDVerification y getModuleInfo, también se encarga de enviar estos mensajes de forma periódica y guarda más información sobre cada módulo.

Al iniciarse el sistema, se inicializa DeviceManager con la información del módulo local y se registran los handlers de los mensajes en PmwComms.

Después, cualquier modulo interesado en ser avisado cuando se detecte un cambio en los módulos conectados, puede registrarse en DeviceManager.

Además de todo lo anterior, al iniciarse el DeviceManager de ECU, se crea un nuevo proceso donde de forma periodica se envían mensajes de deviceDiscovery. Estos mensajes son de tipo multicast y llegan a todos los módulos, que responden enviando otro mensaje multicast de tipo deviceDiscoveryResponse. Este mensaje contiene el moduleSessionID del módulo que ha respondido.

Al ser la respuesta multicast, todos los módulos la reciben y guardan en su lista de módulos la dirección y moduleSessionID de cada módulo que responde.

En el caso de ECU, donde también interesa conocer más información acerca de los módulos, cuando se procesa un mensaje de deviceDiscoveryResponse, se envía al módulo que ha contestado una request de tipo getModuleInfo, cuya response contiene el resto de la información necesaria.

DeviceManager se encarga de verificar los moduleSessionID de los mensajes recibidos por PmwComms, pero hay situaciones donde esto no es deseable, por ejemplo: tras actualizar el software de un módulo, este se reinicia y su moduleSessionID cambia. No obstante, no es un reinicio no planificado ni un error de direccionamiento, por lo que para evitar que sea tratado como tal, en estas situaciones la verificación se puede desactivar desde ECU, y propagarse al resto de módulos mediante el envío del mensaje multicast setModuleSessionIDVerification. Una vez terminada la actualización, se fuerza el reinicio de todos los módulos y se restaura la verificación.

5.2.7.2 Desarrollo

Este componente no ha podido ser implementado como parte del Trabajo Fin de Grado, no obstante, su diseño está completo y se presenta a continuación.

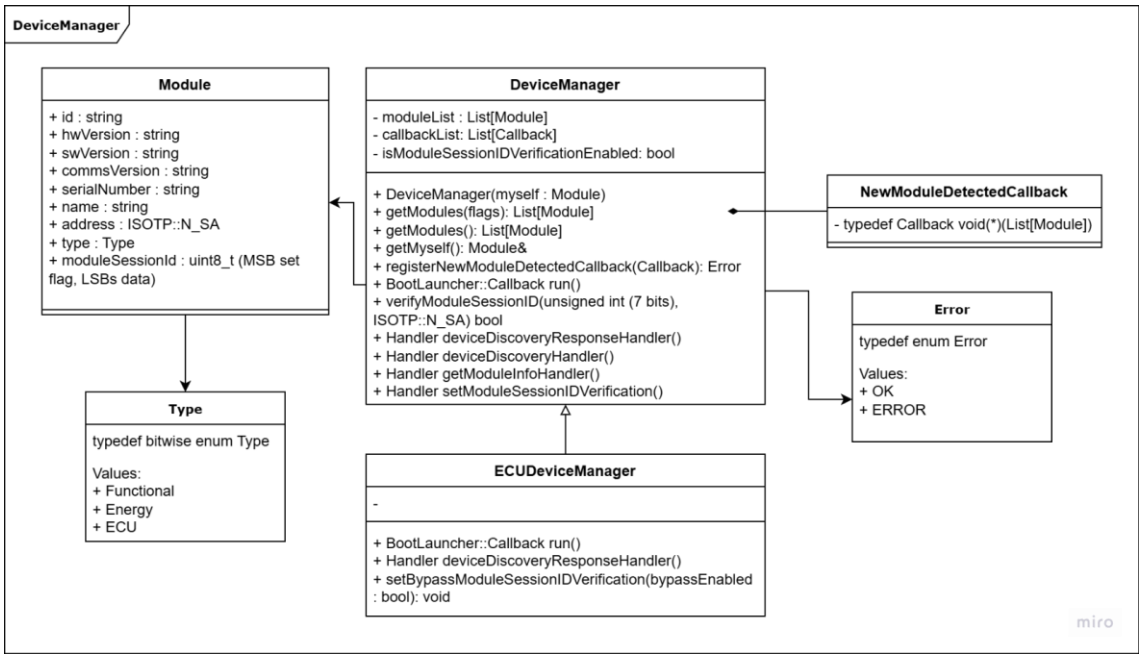

Ilustración 18: Diagrama de clases de DeviceManager

Las dos versiones se implementan en la clase DeviceManager, que contiene la funcionalidad básica y se utiliza en todos los módulos salvo ECU, y ECUDeviceManager, que extiende DeviceManager e incluye la lógica extra necesaria.

Los módulos se guardan en una lista con una peculiaridad, el módulo local ocupa la primera posición de la lista. Esta lista solo se puede modificar mediante la función verifyModuleSessionID, si se encuentra un nuevo módulo que no exista en la lista, o no contenga un moduleSessionID.

La lista de módulos y la información del módulo local pueden ser consultadas mediante a llamadas a las funciones getModules y getMyself. La función getModules admite filtrar por una combinación de tipos de módulo: Functional (Potabilización), Energy y ECU.

DeviceManager está diseñado para ser integrado con BootManager, por lo que contiene un callback que será ejecutado al iniciar el sistema, y que registrará los handlers en PmwComms. En el caso de ECU, además creará un nuevo proceso encargado de enviar periódicamente los mensajes de deviceDiscovery.

ECU puede solicitar más información sobre un módulo nuevo al recibir su deviceDiscoveryResponse enviando desde el handler de este mensaje, una request de tipo getModuleInfo, y registrando la información contenida en la respuesta.

DeviceManager contiene una lista de callbacks que son ejecutados cuando la lista de módulos cambia, esto sirve para notificar a otros componentes del cambio de hardware, y permite modificar sobre la marcha algoritmos como los que gobiernan la potabilización. En cualquier momento de la ejecución del sistema se puede registrar un callback, que se añadirá a los ya existentes y se llamará en el siguiente cambio detectado en la lista de módulos.

ECUDeviceManager contiene una función llamada setModuleSessionIDVerification que permite activar o desactivar la verificación de moduleSessionID, al llamar a esta función, se actualiza el atributo isModuleSessionIDVerificationEnabled y se envía un mensaje multicast de tipo setModuleSessionIDVerification con el valor actual del atributo. Al recibirse este mensaje, el handler setModuleSessionIDVerification actualiza el atributo en su instancia.

5.2.8 BootLauncher

Este componente no depende de librerías de ESP-IDF para funcionar.

5.2.8.1 Diseño

Este componente proporciona un mecanismo para simplificar el proceso de inicio del software de cada microcontrolador y la realización de un Power On Self Test (POST). Permite ejecutar de forma secuencial una serie de acciones de estos tipos: Run, Update o Rollback. El tipo de acción registrado permite ejecutar el callback asociado a la acción en el caso de que durante el proceso de arranque se detecte una de estas situaciones:

- Se ha detectado que se acaba de instalar una actualización del sistema. (Update)
- Se ha detectado que una actualización del sistema ha fallado y se ha restaurado la versión anterior. (Rollback)

Las acciones de tipo Run siempre se ejecutan.

Si durante el proceso de arranque, alguno de los callback de las acciones falla, el proceso de inicio se aborta, y el sistema se reinicia.

Durante el proceso de actualización OTA, una vez actualizado el sistema, este se reinicia para aplicar los cambios. En esta primera ejecución del nuevo software, se deben marcar como validos los cambios, o de lo contrario con el siguiente reinicio el microcontrolador restaurará la versión anterior al iniciarse.

Todos los cambios y pruebas que deban realizarse para comprobar que el nuevo software funciona debe realizarse dentro de las acciones de tipo Update de este componente, y si el arranque es exitoso, entonces se marcará la actualización como válida.

En el caso de que no sea así, el código necesario para deshacer los cambios de la actualización debe ejecutarse como acciones de tipo Rollback.

5.2.8.2 Desarrollo

Este componente no ha podido ser implementado como parte del Trabajo Fin de Grado, no obstante, su diseño está completo y se presenta a continuación.

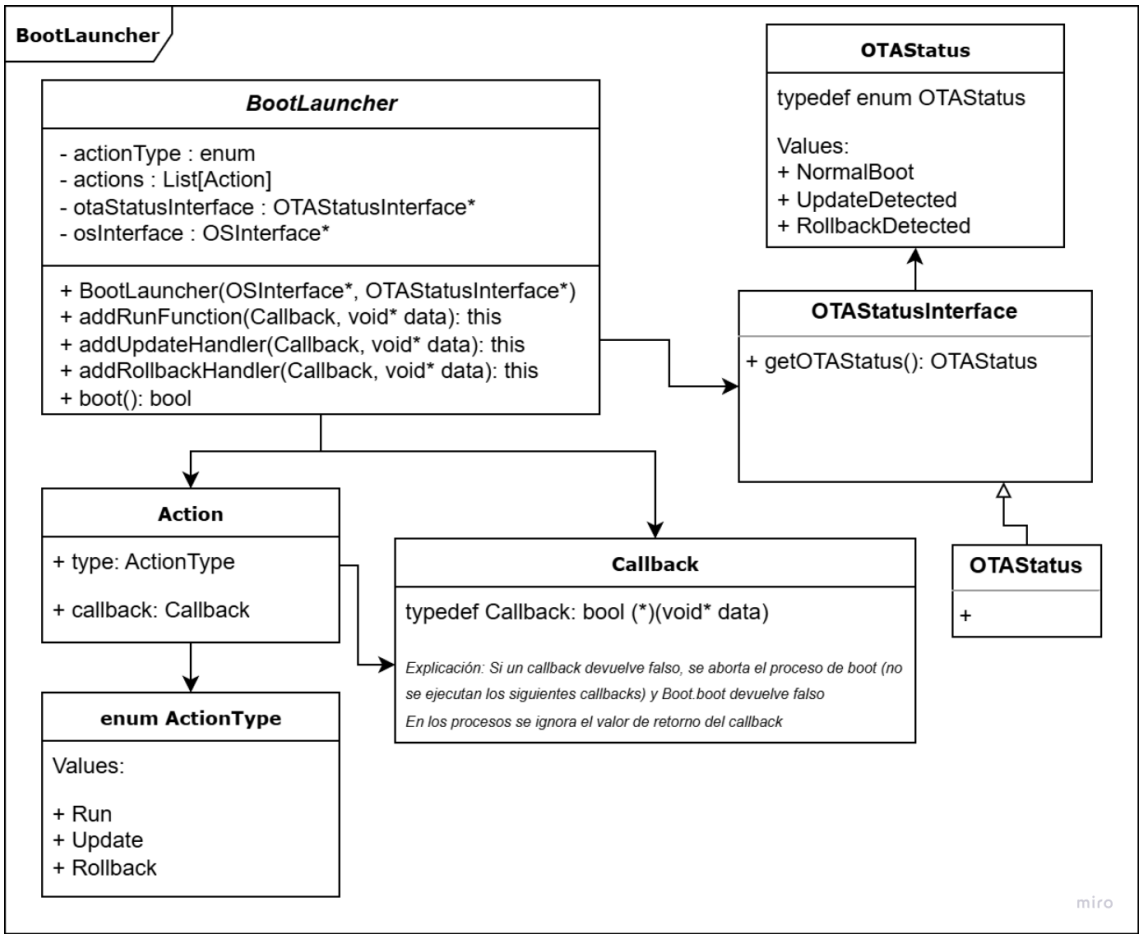

Ilustración 19: Diagrama de clases de BootLauncher

Para utilizar BootLauncher, se debe instanciar un objeto de esta clase utilizando como parámetros las implementaciones de OSInterface y OTAStatusInterface, esta última sirve para desacoplar el componente de funcionalidad dependiente del entorno de ejecución.

Una vez instanciado el objeto, para registrar las acciones se utiliza el patrón de diseño del constructor [39] para añadir cada acción. Estas acciones se realizarán en el orden en el que fueron añadidas, si no se dan las condiciones para que una acción se ejecute (el tipo de acción es distinto del tipo de inicio), esta se saltará.

Por último, una vez configuradas las acciones, ejecturar la función boot realiza todo el proceso de inicio. Si todos los callbacks de las acciones devuelven true, boot devolverá true, si algún callback devuelve false, no se ejecutarán más acciones y se devolverá false.

5.2.9 OTA

Estos componentes dependen de ESP-IDF para funcionar.

5.2.9.1 Diseño

El sistema de actualizaciones inalámbricas permite actualizar los microcontroladores de cualquier módulo del sistema. El archivo de actualización se descarga en ECU, el único módulo que puede tener acceso a internet, y se envía por PmwComms al módulo destino si la actualización no va dirigida a ECU.

Los componentes que realizan las actualizaciones OTA son OTAInstaller, OTAVerifier y OTAManager. OTAManager se ejecuta exclusivamente en ECU y se encarga de comprobar si hay actualizaciones disponibles y descargar y enviar al módulo destino la actualización utilizando PmwComms. OTAInstaller recibe la actualización y la instala en el módulo destino y OTAVerifier se ejecuta en el siguiente reinicio, y tras completar toda la secuencia de inicio de BootLauncher, marca la actualización como correcta y desactiva el mecanismo de rollback, o si algo falla, realiza el rollback automáticamente.

Para ahorrar recursos, la descarga e instalación se realiza de manera simultánea. Para ello, se descarga la actualización en bloques que se envían al módulo destino y se instalan por partes. Cuando se termina de descargar la actualización, se notifica al módulo destino para que termine el proceso de actualización. Esto es posible porque los microcontroladores de todos los módulos poseen una tabla de particiones con al menos dos de ellas dedicadas a OTA [40], esto permite modificar la partición inactiva sin comprometer la integridad del software que se está ejecutando en ese momento. Al reiniciar el microcontrolador, se cambia la partición activa y se inicia el nuevo software en su lugar.

La descarga de las actualizaciones se realiza desde un servidor con la siguiente API REST:

1.1 POST /api/OTA/v1/latest

Check for an available OTA update

Sends the current device information to determine if a new software version is available for download.

REQUEST

REQUEST BODY - application/json

```
{
  moduleID*     string  Unique identifier for the hardware module.
  hwVersion*    string  Hardware version in Semantic Versioning (SemVer) format.
  swVersion*    string  Current software version in SemVer format.
  serialNumber* string  The unique serial number of the device.
}
```

RESPONSE

STATUS CODE - 200: OTA available. The response contains the OTA ID.

RESPONSE MODEL - application/json

```
{
  otaID  string  A unique hash that identifies the OTA update.
}
```

STATUS CODE - 204: No new update is available for this device.

STATUS CODE - 400: Bad request. Required fields are missing or invalid.

Ilustración 20: Descripción de /latest

1.2 GET /api/OTA/v1/info/{otaID}

Get information about a specific OTA

Retrieves the details of an OTA update, such as the new version, description, and changelog, using its ID.

REQUEST

PATH PARAMETERS

NAME	TYPE	DESCRIPTION
*otaID	hash	The unique hash ID of the OTA.

RESPONSE

STATUS CODE - 200: Detailed information of the OTA.

RESPONSE MODEL - application/json

```
{
  swVersion    string  The software version that will be updated to.
  description  string  A brief description of the update.
  changelog    string  A detailed list of changes in this version.
  imageHash    string  The hash (e.g., SHA-256) of the OTA image file for verification.
}
```

Ilustración 21: Descripción de /info

1.3 POST /api/OTA/v1/download

Download the OTA file

Requests the binary file of the OTA update to proceed with the installation on the device.

REQUEST

REQUEST BODY - application/json

```
{
  otaID*  string  The unique hash ID of the OTA to download.
}
```

RESPONSE

STATUS CODE - 200: The OTA binary file.

RESPONSE MODEL - application/octet-stream

```
string
```

The update file (firmware).

STATUS CODE - 404: OTA file for the provided ID was not found.

Ilustración 22: Descripción de /download

5.2.9.2 Desarrollo

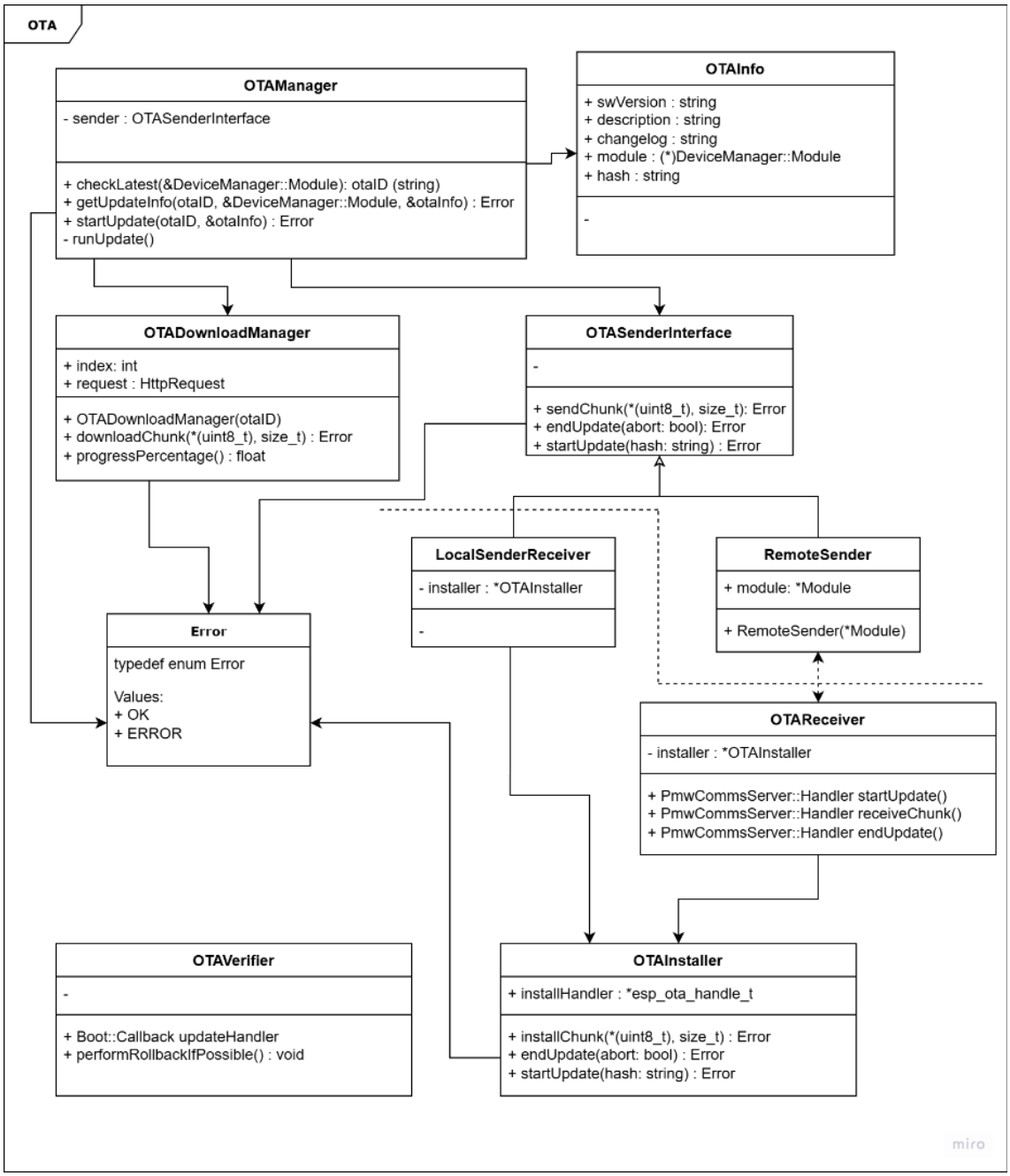

Ilustración 23: Diagrama de clases del sistema de actualizaciones OTA

La comunicación con el servidor de actualizaciones se realiza mediante la clase OTAManager, esta se encarga de recuperar la información de la ultima versión disponible para un módulo y de gestionar el proceso de actualización.

Cuando se ejecuta la función startUpdate, se desactiva la verificación de moduleSessionID y se crean los objectos OTADownloadManager y LocalSenderReceiver (si la actualización es de ECU) o RemoteSender (si la actualización es de cualquier otro módulo). Después ejecuta la función startUpdate del objeto sender para preparar el objeto OTAInstaller del módulo a actualizar.

Después, se ejecuta runUpdate, que realiza las siguientes acciones en bucle hasta terminar la actualización:

1. Descarga un bloque del archivo de actualización
2. Envía el bloque al módulo destino.
3. Este bloque se recibe en el módulo destino y se instala.

Si ocurre un error en alguno de estos pasos, se ejecuta la función endUpdate del sender con la flag de abort activa para parar el proceso de actualización y descartar los cambios. Cuando termina la actualización se ejecuta endUpdate sin la flag de abort, para indicar que se ha terminado correctamente de descargar. OTAInstaller reinicia el microcontrolador tras recibir esta orden.

Tras reiniciarse, se ejecuta BootLauncher con el evento Update y si se ejecuta correctamente el proceso de inicio, OTAVerifier marca la actualización como exitosa. (OTAVerifier se ejecuta desde BootLauncher, siendo la última acción en ejecutarse de la cadena). Si el proceso de inicio falla, se ejecuta la función performRollbackIfPossible, que restaura el sistema al estado anterior.

En el caso de que ocurra alguna excepción durante el proceso de arranque, el gestor de arranque de Espressif se encarga automáticamente de realizar el rollback. [40]

5.2.10 PurificationManager

Este componente gestiona los módulos de potabilización que pueden funcionar a la vez, teniendo en cuenta las restricciones que cada módulo indica, y maximizando la cantidad de módulos activos a la vez y sin exceder la energía disponible en cada momento.

Este componente no ha sido diseñado todavía, por lo que no se puede proporcionar más información al respecto.

5.2.11 UIServer

Este componente gestiona toda la interacción persona – ordenador, independientemente del tipo de interfaz utilizada. Este componente recibe las acciones de una interfaz de usuario, como una interfaz gráfica (GUI) o una interfaz por comandos (CLI), las ejecuta y envía la respuesta a la interfaz de origen.

Este componente no ha sido diseñado todavía, por lo que no se puede proporcionar más información al respecto.

6 Evaluación del sistema

Para evaluar el correcto funcionamiento de los componentes desarrollados, se han realizado una serie de test unitarios y de integración. Para poder mantener la calidad del código implementado, se han utilizado en la medida de lo posible otros programas como linters, analizadores estáticos y de memoria.

6.1 Configuración de Compilación

Todos los componentes del proyecto se compilan utilizando las opciones más estrictas que permite el compilador: -Wall, -Wextra, -Wpedantic y -Werror.

Con ello se minimiza la introducción de fallos en el código y se fuerza a realizar soluciones más elegantes y a escribir código menos ambiguo.

6.2 Test Driven Development

Para garantizar que los test cubren el funcionamiento actual del código, se ha seguido la metodología TDD, que consiste en crear (o modificar) los test necesarios antes de implementar los cambios en el código del producto. Esto permite mantener los tests al día con el código, y permiten que los test documenten el funcionamiento esperado de tu programa.

6.3 Integración Continua

Para aumentar la capacidad de localizar los fallos que puedan introducirse en el código, y localizarlos y corregirlos antes de que supongan un problema, se ha creado un mecanismo de integración continua que automatiza gran parte del trabajo de verificación del funcionamiento del sistema. Para ello se ha utilizado la funcionalidad de GitHub Actions, y su capacidad de ejecutar los tests de cada componente junto con otras herramientas de análisis en cada commit, pull request y merge, minimizando así la necesidad de ejecutar pruebas manuales.

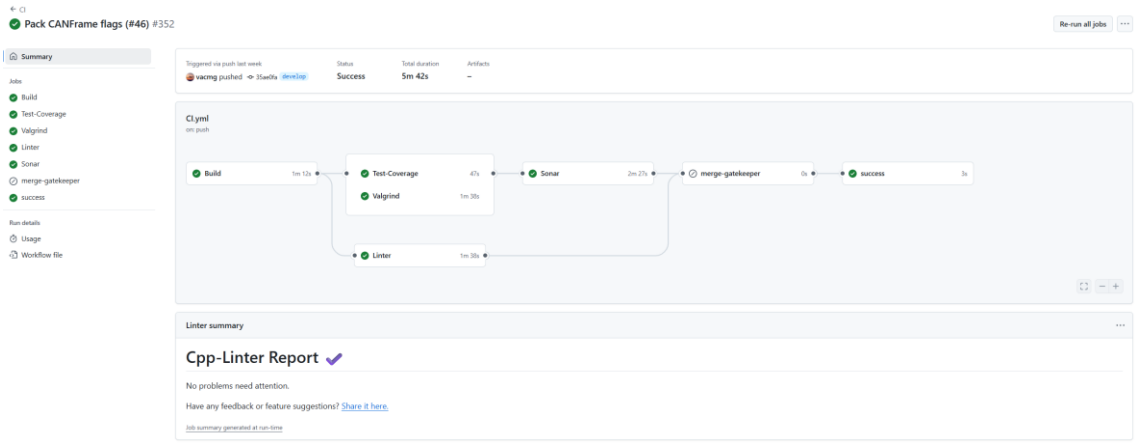

Ilustración 24: Resultado de la ejecución del sistema de integración continua al finalizar una pull request

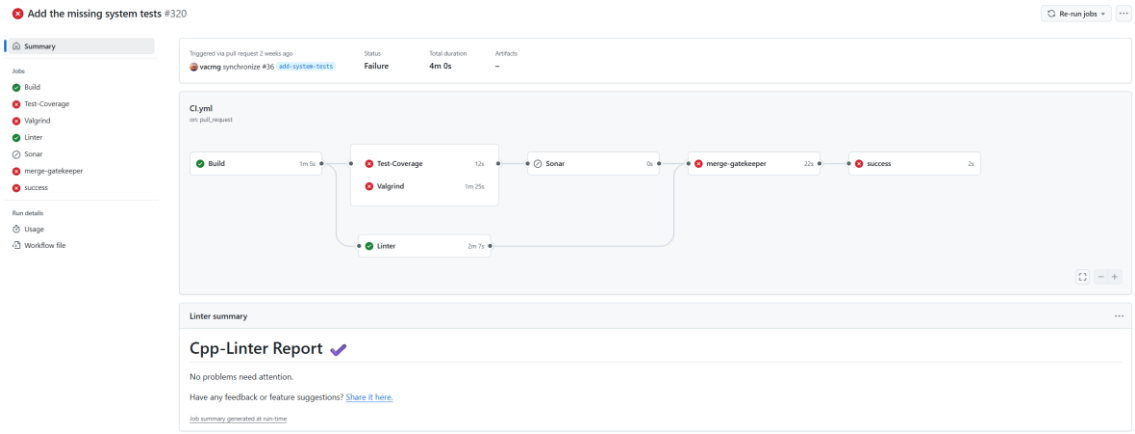

Ilustración 25: Resultado de la ejecución del sistema de integración continua al fallar los test

```
2386  [----------] Global test environment tear-down
2387  [==========] 78 tests from 7 test suites ran. (22756 ms total)
2388  [  PASSED  ] 77 tests.
2389  [  FAILED  ] 1 test, listed below:
2390  [  FAILED  ] LocalCANNetwork.network_getWriteFrameACK_test_success
2391
2392   1 FAILED TEST
2393  Error: Process completed with exit code 1.
```

Ilustración 26: Detalle del error detectado

Como se puede observar en las ilustraciones anteriores, este sistema proporciona al desarrollador de información útil y próxima temporalmente a la introducción de un fallo, lo que permite analizarlo y repararlo de forma más rápida y sencilla.

En el desarrollo de los componentes de ECU, se distinguen dos tipos, aquellos que dependen de ESP-IDF, y aquellos que solo dependen de librerías estándar de C++.

6.3.1 Integración Continua para Componentes Dependientes de ESP-IDF

Estos componentes tienen una gran limitación a la hora de ejecutar test y verificar su funcionamiento, no existe un compilador de arquitectura x86, y no se puede ejecutar un programa compilado para ESP32 de forma nativa. La solución, utilizar un emulador.

ESP-IDF contiene un emulador de la arquitectura ESP32 con capacidad de emular no solo el procesador, sino también ciertos componentes hardware del microcontrolador real.

No obstante, herramientas como SonarQube y Valgrind no funcionan en esta arquitectura por falta de símbolos proporcionados por ESP-IDF o por no poder ejecutarse de forma nativa.

El sistema de CI para componentes dependientes de ESP-IDF consiste en compilar para la arquitectura ESP32 una aplicación de prueba que contiene los tests unitarios y de integración a ejecutar, para después iniciar el emulador y ejecutar los test dentro.

```yaml
on:
    pull_request:
        branches-ignore:
            - '*Test*'
    push:
        branches:
            - 'develop'
            - 'main'

jobs:
    Build-Test:
        runs-on: ubuntu-latest
        env:
            GH_PAT: ${{ secrets.PMW_CI_PAT }}
            GHCR_PAT: ${{ secrets.PMW_GHCR_PAT }}
            GH_USER_NAME: vacmg
            IDF_DOCKER_IMAGE: ghcr.io/vacmg/pmw-esp-idf
            IDF_DOCKER_TAG: release-v5.4

        steps:
            -   name: "Checkout repository"
                uses: actions/checkout@v4

            -   name: Cache Build Environment
                id: CacheBuildEnvironment
                uses: actions/cache@v4
                with:
                    key: "${{ runner.os }}-build-${{ hashFiles('**.cpp', '**.h', '**CMakeLists.txt') }}"
                    path: 'test-project/build'

            -   name: "Docker GitHub Container Registry Login"
                run: echo $GHCR_PAT | docker login ghcr.io -u USERNAME --password-stdin

            -   name: "ESP-IDF CI"
                uses: espressif/esp-idf-ci-action@v1
                with:
                    esp_idf_docker_image: ${{ env.IDF_DOCKER_IMAGE }}
                    esp_idf_version: ${{ env.IDF_DOCKER_TAG }}
                    extra_docker_args: --env GH_PAT=$GH_PAT --env GH_USER_NAME=$GH_USER_NAME
                    target: esp32
                    path: 'test-project'
                    command: |
                        git config --global --replace-all "url.https://$GH_USER_NAME:$GH_PAT@github.com/.insteadOf" ssh://git@github.com: &&
                        git config --global --add "url.https://$GH_USER_NAME:$GH_PAT@github.com/.insteadOf" git@github.com: &&
                        git config --global --add "url.https://$GH_USER_NAME:$GH_PAT@github.com/.insteadOf" https://github.com/ &&

                        idf.py build &&
                        pytest
```

Ilustración 27: Definición de la acción de GitHub de Integración Continua para ESP-IDF

Para ejecutar los test unitarios automáticamente dentro del emulador (QEMU), ESP-IDF proporciona una librería para pytest que simplifica la ejecución de los test a unas pocas líneas de código:

```
1    from pytest_embedded_qemu import QemuDut
2
3    def test_qemu_run_unity(dut: QemuDut) -> None:
4        dut.run_all_single_board_cases()
5        assert len(dut.testsuite.testcases) > 0, "No test cases found"
```

Ilustración 28: Contenido de test_orchestrator.py

6.3.2 Integración Contina para Componentes Independientes de ESP-IDF

Estos componentes no presentan las limitaciones anteriores, y permiten un mayor nivel de análisis.

Estos componentes, además de compilar sus aplicaciones de prueba (donde están contenidos los test del componente) y ejecutarlas, también se ejecuta clang-tidy, un linter que analiza el código de forma estática, clang-format, un programa que comprueba que el código sigue una guía de estilo predefinida, Valgrind, un analizador de memoria que permite detectar memory leaks, use after free y otros fallos relacionados con el manejo de la memoria y SonarQube, un potente analizador estático que genera reportes de seguridad, bugs y cobertura de código por los test.

```yaml
jobs:
    Build:
        runs-on: ubuntu-latest
        env:
            GH_PAT: ${{ secrets.PMW_CI_PAT }}
            GH_USER_NAME: vacmg

        steps:
            -   name: "Import GitHub credentials" # ~/.gitconfig
                run: |
                    git config --global --replace-all "url.https://$GH_USER_NAME:$GH_PAT@github.com/.insteadOf" ssh://git@github.com:
                    git config --global --add "url.https://$GH_USER_NAME:$GH_PAT@github.com/.insteadOf" git@github.com:
                    git config --global --add "url.https://$GH_USER_NAME:$GH_PAT@github.com/.insteadOf" https://github.com/

            -   name: Cat GitHub credentials
                run: cat ~/.gitconfig

            -   name: "Checkout repository"
                uses: actions/checkout@v4

            -   name: Cache Build Environment
                id: CacheBuildEnvironment
                uses: actions/cache@v4
                with:
                    key: "${{ runner.os }}-${{ hashFiles('Source/**', 'Tests/**') }}-build"
                    path: 'build'

            -   name: Configure CMake
                if: steps.CacheBuildEnvironment.outputs.cache-hit == '' # If the cache was not restored
                run: mkdir build && cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_CXX_FLAGS=--coverage -DCMAKE_C_FLAGS=--coverage -S . -B ./build

            -   name: Build Project
                if: steps.CacheBuildEnvironment.outputs.cache-hit == '' # If the cache was not restored
                run: cmake --build ./build --target LinuxOSInterface

            -   name: Build Tests & Dependencies
                if: steps.CacheBuildEnvironment.outputs.cache-hit == '' # If the cache was not restored
                run: cmake --build ./build
```

Ilustración 29: Definición de la etapa "Build" de acción de GitHub de Integración Continua para C++

Para agilizar la ejecución de los trabajos de CI, se utiliza la cache para guardar la configuración y compilaciones antiguas, mejorando la velocidad de compilación al no tener que recompilar todo el código, sino solamente las partes que cambian.

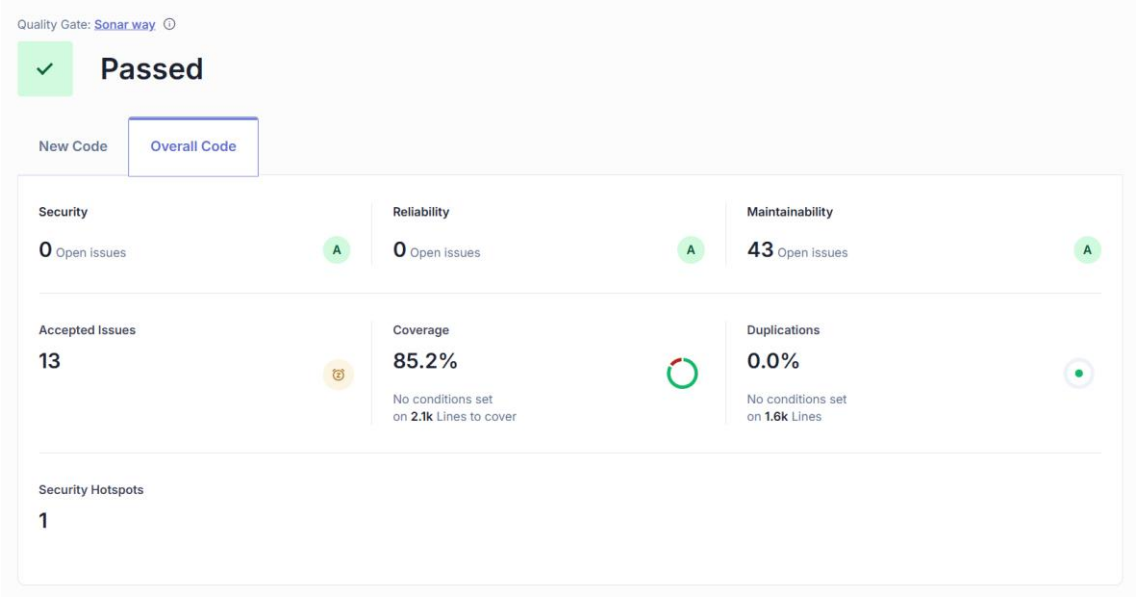

Ilustración 30: Reporte generado por Sonar

SonarQube permite obtener el estado general del código de cada componente, y proporciona un detallado sistema para analizar los problemas y posibles soluciones a errores que detecte esta herramienta.

Ilustración 31: Detalle de los problemas detectados

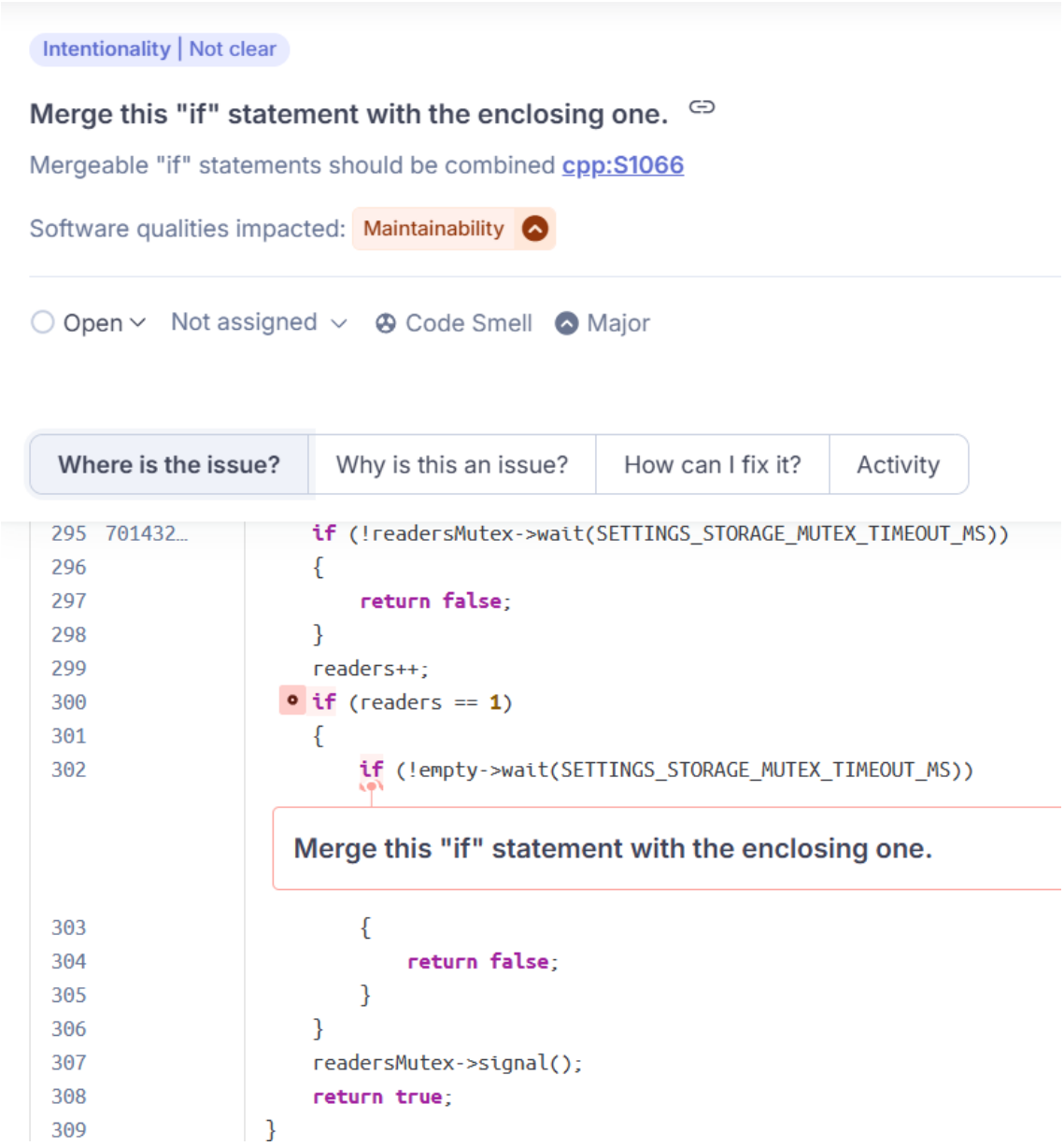

Ilustración 32: Detalle de uno de los problemas detectados

6.4 Técnicas Utilizadas para Realizar los Test

Para comprobar el funcionamiento de componentes que dependen de otros se han realizado varías técnicas:

- Uso de mocks y fakes: Para solucionar dependencias de sistemas hardware, como un bus CAN, se ha optado por construir un fake, es decir una implementación ligera que mantiene la funcionalidad principal (envío y recepción de tramas), pero falsifica detalles del sistema real, como el arbitraje para priorizar el envío de una trama frente a otra.
- Implementaciones para diferentes arquitecturas: La mayoría de los componentes desarrollados dependen del uso de componentes dependientes del sistema operativo, como los mecanismos de sincronización. En este caso se ha optado por construir una interfaz única y utilizar la implementación compatible con el sistema donde se ejecuta el programa. Aunque suponga un mayor esfuerzo mantener dos implementaciones distintas, se consigue poder ejecutar de forma homogénea otros componentes sin depender del uso de emuladores en el proceso de integración continua, lo que agiliza el desarrollo y permite utilizar herramientas de análisis más avanzadas.

7 Análisis de Impacto

7.1 Impacto Personal

En el plano personal, poder desarrollar parte de Purify my Water dentro de este Trabajo Fin de Grado me ha permitido por un lado ganar más experiencia en el diseño y desarrollo de sistemas informáticos, y por otro lado poder dedicarlos a un proyecto con fines humanitarios.

La realización de este trabajo ha permitido realizar grandes avances en el proyecto, hace tan solo un año solamente teníamos un diseño de alto nivel del prototipo que se está desarrollando, y ahora tenemos una gran parte del prototipo totalmente diseñada, tanto en software como en hardware.

También he podido afrontar retos técnicos de gran envergadura, como es el desarrollo de sistemas embebidos, donde el hardware se debe desarrollar a la vez que el software, y garantizar la compatibilidad entre ambos es esencial.

Por último, este trabajo me ha permitido compaginar la finalización de mis estudios académicos de grado, con un proyecto al que he dedicado una parte importante de mi vida, y al que puedo dedicar todo este esfuerzo.

7.2 Impacto Técnico

En cuanto al impacto técnico, una de las grandes innovaciones propuestas en este proyecto ha sido la modularidad llevada al extremo, el sistema se puede simplificar hasta el punto de que solamente se necesitan tres tipos de elementos para construir un sistema de potabilización: la unidad de control (ECU), la unidad de energía y módulos de potabilización que pueden ser conectados y utilizados sin informar previamente a la ECU de su finalidad ni requerir actualizaciones de software ni modificaciones en el sistema.

Por otro lado, este proyecto cubre un espacio que ningún otro sistema ha cubierto antes, un sistema de tamaño medio con altas capacidades de filtración y desinfección que puede operar de forma autónoma y sin supervisión humana, con la excepción de realizar pequeños mantenimientos periódicos, y que minimiza el uso de consumibles para no depender de una cadena de suministro para operar durante un gran periodo de tiempo.

7.3 Impacto Socioeconómico

Este proyecto tiene la capacidad de mejorar la calidad de vida de millones de personas, los usuarios objetivo son personas que viven en zonas rurales y semiurbanas de países en vías de desarrollo, y en las que no existe una infraestructura hídrica y la red eléctrica no es fiable. Si tomamos que cada potabilizadora está diseñada para suministrar agua para consumo a unas 300 personas aproximadamente, y que el sistema está diseñado para ser asequible y fácilmente escalable y replicable, se puede realizar una estimación de lo que instalar 100, o 1000 puede suponer a la población de estas zonas del planeta.

7.3.1 Objetivos de Desarrollo Sostenible

Este proyecto está comprometido con los Objetivos de Desarrollo Sostenible, y generará cuando esté terminado mejoras en las siguientes metas:

7.3.1.1 Meta 1.4

"Para 2030, garantizar que todos los hombres y mujeres, en particular los pobres y los más vulnerables, tengan los mismos derechos a los recursos económicos, así como acceso a los servicios básicos, la propiedad y el control de las tierras y otros bienes, la herencia, los recursos naturales, las nuevas tecnologías y los servicios económicos, incluida la microfinanciación" [41]

El agua potable es un servicio básico que mejora la calidad de vida y reduce los costos asociados a la compra de agua embotellada o tratamientos médicos derivados del consumo de agua insalubre. [17]

7.3.1.2 Meta 3.3

"Para 2030, poner fin a las epidemias del SIDA, la tuberculosis, la malaria y las enfermedades tropicales desatendidas y combatir la hepatitis, las enfermedades transmitidas por el agua y otras enfermedades transmisibles" [41]

Al eliminar las enfermedades causadas por agua no potable (diarrea, cólera, fiebre tifoidea), el sistema mejora significativamente la salud de las comunidades. [17]

7.3.1.3 Meta 3.9

"Para 2030, reducir sustancialmente el número de muertes y enfermedades producidas por productos químicos peligrosos y la contaminación del aire, el agua y el suelo" [41]

Garantizar agua potable reduce la exposición a químicos y contaminantes. [17]

7.3.1.4 Meta 6.1

"De aquí a 2030, lograr el acceso universal y equitativo al agua potable a un precio asequible para todos" [41]

El sistema aborda directamente esta meta al proporcionar agua limpia a comunidades donde este recurso es escaso o inexistente. [17]

7.3.1.5 Meta 6.4

"De aquí a 2030, aumentar considerablemente el uso eficiente de los recursos hídricos en todos los sectores y asegurar la sostenibilidad de la extracción y el abastecimiento de agua dulce para hacer frente a la escasez de agua y reducir considerablemente el número de personas que sufren falta de agua" [41]

Nuestro sistema autónomo optimiza la potabilización y minimiza el desperdicio tanto de consumibles como de agua, gracias al diseño de sus etapas de potabilización. [17]

7.3.1.6 Meta 6.6

"De aquí a 2030, proteger y restablecer los ecosistemas relacionados con el agua, incluidos los bosques, las montañas, los humedales, los ríos, los acuíferos y los lagos" [41]

Nuestro sistema contempla mecanismos para garantizar una extracción sostenible de agua, garantizando que no se pueda superar un máximo de agua potabilizada por día, para evitar que, ante fallos de sensores, el sistema intente extraer toda el agua posible. [17]

8 Conclusiones y Trabajo Futuro

8.1 Objetivos cumplidos

En este Trabajo Fin de Grado, se han completado la mayoría de los objetivos previstos:

O1 (Estudio del problema y el estado del arte) y O2 (Diseño de un sistema de sistema de potabilización de agua a pequeña escala barato, modular y automático) se han completado en su totalidad.

O3 (Diseño de la unidad de control del sistema) y O4 (Desarrollo de varios componentes software de la unidad de control) se han completado parcialmente, porque no ha dado tiempo a diseñar, validar y desarrollar los componentes de PurificationManager y UIServer, ya que quedan todavía decisiones de diseño por tomar con respecto a estos sistemas en conjunto con el resto del equipo de desarrollo del proyecto.

O5 (Validación y Pruebas) se ha completado parcialmente, ya que, al no haber terminado el desarrollo completo de la unidad de control, y no tener listo el software de otros módulos, no se han podido realizar test de sistema, y solamente se han podido realizar algunos test de integración entre componentes, además de los test unitarios de cada componente.

8.2 Trabajo futuro

Las líneas de trabajo futuro son claras:

A corto y medio plazo, se debe continuar desarrollando tanto el hardware como el software del prototipo, para validar tanto la viabilidad de los sistemas de control, como la capacidad de potabilización que posee el sistema completo. Una vez validado se debe crear un proyecto piloto en un entorno real, que sirva para revisar y optimizar el diseño, y realizar las mejoras pertinentes de cara a la producción en masa del sistema.

A largo plazo, el objetivo es conseguir la máxima expansión de este sistema de potabilización, para poder conseguir el mayor impacto social posible. Los retos que encontraremos en esa fase estarán relacionados con la producción en masa del sistema y con la escalabilidad de la producción, para pasar de una producción artesanal, a una industrial.

8.3 Conclusión

Gracias a este Trabajo Fin de Grado, el equipo detrás de Purify my Water se encuentra un paso más cerca de conseguir dejar una huella positiva en el mundo, contribuyendo a mejorar sociedades enteras gracias a nuestro sistema de potabilización modular.

9 Bibliografía

[1] Organización Mundial de la Salud, «Agua para consumo humano,» 13 9 2023. [En línea]. Available: https://www.who.int/es/news-room/fact-sheets/detail/drinking-water. [Último acceso: 4 11 2024].

[2] United Nations, «Objetivo 6: Garantizar la disponibilidad de agua y su gestión sostenible y el saneamiento para todos,» [En línea]. Available: https://www.un.org/sustainabledevelopment/es/water-and-sanitation/. [Último acceso: 04 11 2024].

[3] Portal de la Contratación Pública de la Comunidad de Madrid, «Proyecto y obra de la nueva ETAP en Colmenar Viejo. Fase I,» 19 1 2023. [En línea]. Available: https://contratos-publicos.comunidad.madrid/contrato-publico/proyecto-obra-nueva-etap-colmenar-viejo-fase-i. [Último acceso: 04 11 2024].

[4] A. Burns y A. Wellings, Sistemas de Tiempo Real y Lenguajes de Programación, Addison Wesley, 2002.

[5] B. Úbeda Miñarro, 2009. [En línea]. Available: https://www.um.es/documents/4874468/19345367/ssee-t01.pdf/4ea71f56-2950-4c3f-acbe-e7699e490f4e. [Último acceso: 7 1 2025].

[6] Ministerio de Sanidad España, « Calidad de las aguas - Preguntas más frecuentes,» [En línea]. Available: https://www.sanidad.gob.es/profesionales/saludPublica/saludAmbLaboral/calidadAguas/preguntasFrec.htm. [Último acceso: 08 11 2024].

[7] Naciones Unidas, «Informe de los Objetivos de Desarrollo Sostenible 2019,» 2019.

[8] UN Water, «Water quality and wastewater,» 2018.

[9] J. Carmody, «Water treatment in Developing Countries,» 17 4 2020. [En línea]. Available: https://usfblogs.usfca.edu/sustainability/2020/04/17/water-treatment-in-developing-countries/. [Último acceso: 09 11 2024].

[10] Universidad de Chile, «ecnología de bajo costo permitirá abastecer de agua libre de contaminantes metálicos a zonas rurales del país,» 10 10 2023. [En línea]. Available: https://uchile.cl/noticias/209904/tecnologia-permitira-remover-metales-del-agua-en-zonas-rurales. [Último acceso: 7 11 2024].

[11] C. R. Chinachi Sotalin y B. A. Macas Ochoa, «Aplicación del Método Sodis utilizando Tres Sistemas de Tratamiento de Desinfección de Agua Lluvia en la Comunidad Puerto Santa Ana.,» 2020.

[12] Ecofiltro Europe, «Ecofiltro Process,» [En línea]. Available: https://ecofiltroeurope.com/pages/process. [Último acceso: 7 11 2024].

[13] Fraunhofer IST, «SafeWaterAfrica (Fraunhofer IST),» 30 11 2019. [En línea]. Available: https://www.ist.fraunhofer.de/en/reference-projects/safewaterafrica.html. [Último acceso: 10 11 2024].

[14] European Commission, «CORDIS EU research results,» 16 8 2022. [En línea]. Available: https://cordis.europa.eu/project/id/689925. [Último acceso: 10 11 2024].

[15] Desolenator, «Desolenator Technology,» [En línea]. Available: https://www.desolenator.com/product. [Último acceso: 10 11 2024].

[16] AquaResearch, «H2gO Purifier,» [En línea]. Available: https://aquaresearch.com/product-info/. [Último acceso: 10 11 2024].

[17] Purify my Water, «Acerca del proyecto,» [En línea]. Available: https://purifymywater.org/. [Último acceso: 04 06 2025].

[18] Forbes España, «Forbes,» 2021. [En línea]. Available: https://forbes.es/mejores-creativos/. [Último acceso: 6 1 2025].

[19] Ashoka España y Portugal, «Ashoka Young Changemakers,» [En línea]. Available: https://www.ashoka.org/es-es/program/j%C3%B3venes-changemakers. [Último acceso: 6 1 2025].

[20] Actúa UPM, «Premios a las Mejores Ideas #22ActúaUPM,» [En línea]. Available: https://actuaupm.blogspot.com/2025/05/quieres-saber-si-tu-idea-pasa-la.html. [Último acceso: 23 06 2025].

[21] J. W. Grenning, Test-Driven Development for Embedded C, Pragmatic Bookshelf, 2011.

[22] Github, «CI with GitHub Actions,» [En línea]. Available: https://docs.github.com/es/actions/about-github-actions/about-continuous-integration-with-github-actions. [Último acceso: 7 1 2025].

[23] S. Chacon y B. Straub, Pro Git, Apress.

[24] GitHub, «Acerca de GitHub y Git,» [En línea]. Available: https://docs.github.com/es/get-started/start-your-journey/about-github-and-git. [Último acceso: 7 1 2025].

[25] Clang, «Clang Tools,» [En línea]. Available: https://clang.llvm.org/docs/ClangTools.html. [Último acceso: 7 1 2025].

[26] CMake, «Cmake About,» [En línea]. Available: https://cmake.org/about/. [Último acceso: 7 1 2025].

[27] SonarSource, «SonarQube,» [En línea]. Available: https://www.sonarsource.com/. [Último acceso: 7 1 2025].

[28] Google, «Google Test Github,» [En línea]. Available: https://github.com/google/googletest. [Último acceso: 7 1 2025].

[29] Espressif, «ESP IDF,» [En línea]. Available: https://www.espressif.com/en/products/sdks/esp-idf. [Último acceso: 7 1 2025].

[30] CLion, «CLion,» [En línea]. Available: https://www.jetbrains.com/es-es/clion/. [Último acceso: 7 1 2025].

[31] Miro, «Using Miro,» [En línea]. Available: https://help.miro.com/hc/en-us/categories/360001420434-Using-Miro. [Último acceso: 04 06 2025].

[32] Espressif, «Espressif,» [En línea]. Available: https://www.espressif.com/. [Último acceso: 11 6 2025].

[33] Bosch (via archive.org), «CAN Literature,» 19 8 2014. [En línea]. Available: https://web.archive.org/web/20140819090626/http://www.bosch-semiconductors.de/en/ubk_semiconductors/ip_modules_3/produkttabelle_ip_modules/can_literature_1/can_literature.html. [Último acceso: 11 6 2025].

[34] ISO/TC 22/SC 31, «ISO 15765-2:2024 Road vehicles — Diagnostic communication over Controller Area Network (DoCAN) Part 2: Transport protocol and network layer services,» ISO, 2024.

[35] P. Andersson, «SPIFFS,» [En línea]. Available: https://github.com/pellepl/spiffs. [Último acceso: 17 6 2025].

[36] V. Leis, A. Kemper y T. Neumann, «The Adaptive Radix Tree: ARTful Indexing for Main-Memory Databases,» *ICDE '13: Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013),* pp. 38-49, 2013.

[37] L. Lamport, «Concurrent reading and writing,» *Communications of the ACM,* vol. 20, nº 11, pp. 806-811, 1977.

[38] A. Dadgar, «libart,» [En línea]. Available: https://github.com/armon/libart. [Último acceso: 17 6 2025].

[39] A. Shvets, Dive Into Design Patterns, 2018.

[40] Espressif, «Over The Air Updates (OTA) - ESP32 - v5.4.1,» [En línea]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/ota.html. [Último acceso: 21 6 2025].

[41] United Nations, «Objetivos de desarollo sostenible,» [En línea]. Available: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/. [Último acceso: 22 6 2025].

[42] Samco Tech, «How much does an industrial water treatment system cost,» [En línea]. Available: https://samcotech.com/how-much-does-an-industrial-water-treatment-system-cost/. [Último acceso: 9 11 2024].

10 Anexo

10.1 Repositorios de código del proyecto

10.1.1 ISOTPLib

https://github.com/vacmg/ISOTPLib

10.1.2 SettingsStorageLib

https://github.com/vacmg/SettingsStorageLib

https://github.com/vacmg/SettingsFile

10.1.3 OSInterface

https://github.com/vacmg/OSInterface

https://github.com/vacmg/EspOSInterface

https://github.com/vacmg/LinuxOSInterface

10.1.4 PmwStoragePartitionManager

https://github.com/vacmg/PmwStoragePartitionManager

10.1.5 PmwOTAVerifier

https://github.com/vacmg/PmwOTAVerifier

10.1.6 PmwWifi

https://github.com/vacmg/PmwWifi

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Wed Jul 02 17:38:43 CEST 2025
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)