Universidad de la República
Facultad de Ingeniería

Controlador de energía domiciliario para una Red Eléctrica Inteligente

Memoria de proyecto presentada a la Facultad de Ingeniería de la Universidad de la República por

Gonzalo Belcredi, Pablo Modernell, Nicolás Sosa

en cumplimiento parcial de los requerimientos para la obtención del título de Ingeniero Electricista.

Tutor
Dr. Ing. Leonardo Steinfeld ..... Universidad de la República

Co-Tutor
Dr. Ing. Fernando Silveira ..... Universidad de la República

Tribunal
Dr. Ing. Alvaro Giusto ..... Universidad de la República
Dr. Ing. Mario Vignolo ..... Universidad de la República
Dr. Ing. Leonardo Steinfeld ..... Universidad de la República

Montevideo
martes 9 junio, 2015

Controlador de energía domiciliario para una Red Eléctrica Inteligente, Gonzalo Belcredi, Pablo Modernell, Nicolás Sosa.

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 211 páginas.
Compilada el martes 9 junio, 2015.
http://iie.fing.edu.uy/

Agradecimientos

Queremos agradecer a todas las personas que de alguna forma colaboraron con la concepción, desarrollo y ejecución de este proyecto. A *Alejandro Gutierrez* del IMFIA por el aporte en las ideas originales que nos llevaron hacia la temática de Redes Eléctricas Inteligentes, a *Fernando Silveira* y *Martín Randall* por su apoyo en la definición del proyecto. A *Julián Oreggioni* por su disponibilidad para evacuar consultas sobre aspectos de diseño de circuitos y sus recomendaciones para mejoras en el diseño, a *Javier Schandy* que participó en la construcción de los circuitos y *Daniel Slomovitz* por sus valiosos comentarios y recomendaciones sobre el proceso de medición.

También agradecemos a la Fundación Julio Ricaldoni de la Facultad de Ingeniería y al CII de la ANII por la financiación del proyecto a través de la convocatoria de apoyo a proyectos de fin de grado, así como al Presidente de UTE *Gonzalo Casaravilla* por su interés en el proyecto.

Agradecemos a *Gustavo Fernández* y *Oscar Talmon* de UTE por recibirnos y ayudarnos a comprender un poco más sobre la temática de Redes Eléctricas Inteligentes en el Uruguay, así como a todas las personas que dedicaron amablemente parte de su tiempo para facilitarnos la comprensión de diversos aspectos relacionados con nuestro proyecto.

Hacemos una especial mención a nuestro tutor *Leonardo Steinfeld* por su apoyo constante durante todo el proyecto, tanto en aspectos técnicos como de gestión.

Queremos también destacar y agradecer profundamente el apoyo de nuestras familias a lo largo de todo el proyecto.

Esta página ha sido intencionalmente dejada en blanco.

Resumen

Las transformaciones en la matriz energética que incluyen la incorporación de energías renovables y la micro generación imponen nuevos desafíos técnicos en la distribución y en el uso de la energía con el fin de aprovechar eficientemente los recursos energéticos. Como consecuencia del vertiginoso desarrollo de las tecnologías de la información, se ha incrementado el estudio de formas inteligentes de gestión de la red eléctrica para enfrentar los nuevos desafíos, dando lugar a una tendencia que se conoce como *Smart Grid*.

Para lograr una optimización de la red eléctrica es necesario contar por un lado con un sistema de gestión eléctrica a nivel nacional, que involucre a todos los componentes de la misma y envíe información de tiempo real a los usuarios, de forma que puedan optimizar su consumo.

Por otro lado se necesita un controlador de energía local, e.g. a nivel domiciliario, que pueda recibir esta información, procesarla de forma conveniente y en última instancia realizar comandos de control sobre los distintos electrodomésticos. Para este propósito se implementó una red de comunicación inalámbrica entre los dispositivos a monitorear y el controlador de energía, un circuito que reporta el consumo de los electrodomésticos y otro que es capaz de comandar el encendido y apagado de los mismos. Además se proporciona una interfaz web para que el usuario pueda visualizar el consumo y enviar directivas hacia los electrodomésticos. Se implementó también un algoritmo de control básico para un termotanque que toma en consideración la tarifa dinámica proporcionada por el distribuidor de energía así como las características de consumo del hogar.

El paradigma de desarrollo está fuertemente inspirado en la utilización de hardware y software de código abierto, por lo que se implementó una biblioteca de funciones de forma de favorecer la colaboración de investigadores y entusiastas, por ejemplo a través de la introducción de mejoras en los algoritmos de optimización. En el mismo sentido se espera impulsar la colaboración a nivel de hardware con la incorporación de nuevos sensores y actuadores.

La plataforma cumple globalmente con los objetivos planteados inicialmente, a través de la realización de reportes de consumo de energía y el comando a distancia de los electrodomésticos vía web, así como de la aplicación en tiempo real de algoritmos de optimización.

Esta página ha sido intencionalmente dejada en blanco.

Tabla de contenidos

Tabla de contenidos

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 1

Introducción

1.1. Presentación del problema

Uruguay ha encarado en los últimos años transformaciones importantes en su matriz energética con la ampliación de la potencia instalada, la incorporación de energías renovables y la interconexión eléctrica con Brasil.

El desarrollo de las telecomunicaciones y de las tecnologías de la información ha disparado el estudio de formas inteligentes de gestión de la red eléctrica. Un término ampliamente conocido es el de *Smart Grid*, que hace referencia a la integración de un sistema de comunicación seguro y bi-direccional entre todos los actores involucrados en la generación, distribución y consumo de energía eléctrica [9]. Este sistema tiene como meta la mejora en la gestión de las redes eléctricas, permitiendo un aprovechamiento eficiente de los recursos energéticos y facilitando la incorporación de energías renovables.

Las inversiones en fuentes energéticas como la eólica o la fotovoltaica, en las que factores climáticos imponen máximos de generación en determinados períodos del día, ponen de manifiesto la necesidad de una mejor sincronización entre la generación y el consumo de la energía. Para lograr un máximo aprovechamiento de los recursos, es necesario implementar medios de almacenamiento de energía y métodos para la gestión de la demanda.

La investigación en torno a las posibilidades de pronosticar la disponibilidad de los recursos energéticos a corto plazo es fundamental para posibilitar la sincronización mencionada anteriormente. Además se necesita una plataforma local en cada hogar o industria, que conozca en tiempo real las condiciones de generación, de forma de poder realizar un control de consumo a nivel del usuario final. Esto último es lo que se conoce como gestión de la demanda.

Algunas empresas de suministro eléctrico establecen tarifas diferenciadas según el horario de consumo, que buscan una demanda más equilibrada de los recursos energéticos a lo largo del día. Pero una vez establecida la tarifa, el control del consumo queda bajo total responsabilidad del usuario, quien no cuenta información adecuada sobre el consumo eléctrico de cada dispositivo, de forma de poder ser totalmente consciente de las posibilidades de ahorro. Además, el usuario nor-

malmente no dispone de métodos efectivos para controlar automáticamente sus electrodomésticos.

Un ejemplo de posibilidad de ahorro mediante la elección conveniente de los períodos de encendido está dado por el termotanque. Este representa cerca del 37 % del consumo eléctrico de los hogares (ver [17]) y por ser un dispositivo acumulador de energía posibilita adelantar o retrasar su puesta en funcionamiento de forma de evitar consumo en horas de baja disponibilidad eléctrica.

Para lograr una optimización de la Red Eléctrica es necesario contar por un lado con un sistema de gestión energética a nivel general para un conjunto de consumidores y generadores, que involucre a todos los componentes de la red y envíe información de tiempo real a los usuarios permitiéndoles gestionar de forma inteligente su consumo. Por otro lado se necesita una plataforma local (por ejemplo a nivel domiciliario) que pueda decodificar esta información, procesarla de forma conveniente y de ser necesario llevar a cabo medidas de control de consumo en los dispositivos del hogar.

El sistema de gestión energética (gestión general de toda la red eléctrica) manejaría varias entradas de información, como ser los niveles de generación de energía en las diferentes usinas, el costo de dicha generación, el pronóstico de demanda, etc., con el fin de realizar una gestión completa de la red mediante el envío de mensajes e información a los diferentes actores del sistema (tanto generadores como consumidores).

En el caso de la plataforma local (enfoque por consumidor individual) uno de los desafíos está en el diseño de un sistema que permita comunicarse y recibir información e instrucciones de parte del Sistema de Gestión Energética. Además, otro desafío está en optimizar el consumo a nivel del usuario final independientemente de la disponibilidad de información externa. Se piensa entonces en un sistema de optimización que tenga autoridad y capacidad de control sobre las cargas del usuario final (por ejemplo, electrodomésticos).

El diseño del sistema de gestión energética a nivel general depende de una plataforma aún no disponible en el Uruguay. Los organismos de generación y administración de la energía eléctrica (UTE y ADME) no cuentan aún con un sistema de información y comunicación capaz de suministrar en tiempo real todos los datos que se usarían como entradas de un sistema *Smart Grid*. Sin embargo, el diseño e implementación de un controlador local para el hogar podría suponer en sí mismo un gran aporte para lograr optimizar los consumos de cada domicilio y constituiría un paso importante en la construcción gradual de un sistema de distribución y consumo cada vez más inteligente. Es por este motivo que se decidió enfocar el proyecto en el desarrollo de tal controlador, constituyendo su diseño, implementación y evaluación el primer objetivo de este trabajo. Para tener en cuenta la integración del controlador con un eventual sistema de gestión de energía a nivel general se hicieron hipótesis acerca de las posibles características y métodos de intercambios de información que este sistema podría ofrecer. Principalmente se evaluó la posibilidad de que dicho sistema interactúe con los usuarios enviando información del precio de la energía. Se consideró también la posibilidad de envío de comandos o directivas para actuar directamente sobre los dispositivos

del usuario final.

1.2. Definición del proyecto

El proyecto consiste en el diseño, desarrollo e implementación de una plataforma abierta de optimización de consumo de energía eléctrica a nivel domiciliario (Home Energy Management System). Esta plataforma está constituida por un controlador central y dispositivos periféricos, a los que se hará referencia mediante el término *nodos*, capaces de tomar mediciones y actuar sobre las diferentes cargas (electrodomésticos).

El controlador es capaz de optimizar el consumo eléctrico, manteniendo el confort para los usuarios y proporcionando información mediante una interfaz web para facilitar y motivar un uso eficiente de la energía.

El diseño cuenta con diferentes bloques de hardware, que miden el consumo y realizan el control de los dispositivos eléctricos. Estos bloques se comunican e interconectan de forma de permitir la ejecución de algoritmos de control adecuados.

El sistema proporciona además una API (Application Programing Interface), que permite la implementación y pruebas de algoritmos de optimización de consumo desarrollados por terceros.

A modo de evaluación del sistema y ejemplo de utilización, se implementa y ensaya un algoritmo de optimización de consumo para un termotanque.

El sistema pretende ser una plataforma para pruebas de distintas estrategias de control y optimización de consumo, sin ser el estudio y desarrollo de estas últimas un objetivo central del proyecto.

1.3. Objetivos y Alcance

1.3.1. Objetivo general

Desarrollar una plataforma de monitoreo y control de consumo eléctrico domiciliario a nivel de usuario final de un sistema de gestión de energía eléctrica de tipo *Smart Grid*.

1.3.2. Objetivos Específicos

- Diseñar y construir un Módulo Controlador. Este módulo será capaz de realizar el control del sistema, enviando los comandos y analizando la información recibida por los demás módulos. Será el encargado de centralizar la comunicación con un sistema de gestión general de la red *Smart Grid* (el sistema de gestión general de la red eléctrica está fuera del alcance de este proyecto).

- Proporcionar una API que facilite la implementación y pruebas de algoritmos

de optimización energética desarrollados por terceros[1].

- Diseñar e implementar un algoritmo de control de consumo para un Módulo de Carga, analizando el perfil de operación de al menos un tipo de dispositivo o electrodoméstico, de forma de obtener un método de control y operación adecuado al mismo. Con esto se pretende probar que la plataforma desarrollada comanda satisfactoriamente los dispositivos conectados a ella.
- Diseñar e implementar el sistema de comunicación entre los *nodos*, seleccionando la tecnología más adecuada para la aplicación.
- Construir un Módulo de Medición de consumo, capaz de monitorear y reportar la potencia consumida por un dispositivo o una línea de carga (por ejemplo, una línea asociada a la iluminación o a un grupo de electrodomésticos).
- Construir un Módulo de Carga, capaz de operar un dispositivo o electrodoméstico, controlando de esta forma la potencia consumida por el mismo.

1.3.3. Alcance

- Se implementa una interface web de usuario para visualización de consumo, y para encender y apagar dispositivos de forma remota.
- El diseño de HW es en base a integración de módulos comerciales ya desarrollados, y por ser un prototipo no estará orientado a la comercialización. El objetivo del HW del proyecto es implementar una prueba de concepto, por lo cual se busca obtener un desempeño funcional básico, no necesariamente conforme a las normas y estándares de la industria.
- Los aspectos relacionados con la seguridad hacia los usuarios y protección de datos y equipos NO cumplen necesariamente los requerimientos de un producto final y listo para comercializar.
- No se desarrolló ningún componente del sistema de gestión general de la red *Smart Grid*. Se hizo foco solamente en los dispositivos que constituyen un consumidor (Ej. Hogar, comercio o industria). Sí se contempló la integración entre el sistema local diseñado y un eventual sistema de gestión de energía a nivel nacional o regional, inexistente al momento en el Uruguay.
- Fueron implementados dos Módulos de Medición y comando de carga con capacidad de comunicación remota con el controlador central domiciliario. Para algunas pruebas se incorporaron módulos de comunicación adicionales

[1]Este objetivo no estaba incluido en el alcance original del proyecto. Sin embargo, durante el desarrollo del mismo, al constatar que este campo de trabajo representa un área de profundo interés para la investigación en el futuro próximo a nivel nacional e internacional, por iniciativa propia del grupo de trabajo se decidió ampliar el alcance favoreciendo el trabajo de futuros investigadores.

enviando datos ficticios al controlador, simulando así una sistema de monitoreo y administración de energía domiciliario con mayor cantidad de puntos de medición y control.

1.4. Estructura general del documento

Este capítulo presentó una breve descripción del proyecto especificando claramente sus objetivos y alcances, y se brindó un panorama general del contexto y del problema que se intenta resolver.

En el Capítulo 2 se presenta un marco contextual más amplio, incluyendo motivaciones, desafíos y estado del arte del área de trabajo. También se presenta en dicho capítulo un resumen de los conceptos más relevantes de la bibliografía consultada en una primer etapa de revisión, y se citan algunos trabajos que fueron utilizados a lo largo del proyecto como referencia para la elección de componentes y diseño del sistema.

Luego de la contextualización e introducción general al problema a resolver, se presenta en el Capítulo 3 una descripción global del sistema con explicaciones generales de los distintos bloques de hardware y software diseñados, mostrando los requerimientos que determinarán las tecnologías de diseño.

Los siguientes tres capítulos describen detalladamente el diseño de los componentes del sistema. En el Capítulo 4 se presentan las tecnologías de comunicación utilizadas fundamentando la elección de los protocolos y explicando detalladamente cómo se implementó la comunicación en las plataformas seleccionadas. El Capítulo 5 detalla la forma en que se diseñaron los *nodos* que comandan a los electrodomésticos y reportan las mediciones, se especifica el diseño e implementación de los circuitos de medición y carga, y se presenta el software desarrollado para el microcontrolador seleccionado. Luego el Capítulo 6 explica las consideraciones al momento de diseñar el controlador, justificando la elección del lenguaje de programación y detallando la implementación de la API desarrollada. Este capítulo es de particular interés para quien quiera hacer uso de las funcionalidades de la API para ejecutar algoritmos de optimización de consumo.

Para completar la especificación del diseño del controlador, el Capítulo 7 se concentra en la combinación de los distintos componentes diseñados para implementar un sistema de optimización, monitoreo y comando de dispositivos a través de una interfaz web. Se presenta un ejemplo práctico de cómo utilizar las funcionalidades de la API desarrollada, llevando a la práctica un algoritmo de optimización simple para un termotanque.

En el Capítulo 8 se describe el ensayo y evaluación del desempeño de la plataforma en conjunto, y de cada componente en particular, indicando las principales observaciones acerca de los resultados obtenidos.

Finalmente, en el Capítulo 9 se presentan las conclusiones generales del proyecto y se indican cuales son, a criterio de los autores, las líneas de trabajo futuras más interesantes así como las mejoras necesarias para el sistema implementado.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 2

Revisión bibliográfica y estado del arte

2.1. Introducción

Desde los inicios de la generación y distribución de energía eléctrica a gran escala en el siglo XIX el mundo ha cambiado y las tecnologías han ido superándose en forma constante. Sin embargo, las redes eléctricas no han acompañado esta tendencia y siguen siendo en general muy similares a las de hace un siglo atrás.

Es por ello necesario repensar los paradigmas de generación y distribución de energía eléctrica de acuerdo a una nueva realidad, persiguiendo objetivos como el de lograr una red más segura, menos centralizada, y una matriz energética diversificada y basada fundamentalmente en fuentes de energía renovables.

Por otra parte el aumento constante del consumo eléctrico obliga al desarrollo de estrategias de optimización de las redes, que permitan sacar el máximo provecho de las instalaciones y capacidad energética de la matriz. Algunas categorías de estas estrategias son *Demand-side Management*, *Demand Response* y *Energy Management System*.

Tales estrategias, tendientes a buscar un consumo más eficiente, equilibrado en el tiempo y planificado, son difícilmente realizables en el marco que imponen las infraestructuras y tecnologías actuales. Por estos motivos avanzar hacia redes eléctricas más inteligentes es una necesidad y un desafío.

Existen hoy en día nuevas tecnologías de comunicación y procesamiento de datos que posibilitan la implementación de sistemas eléctricos inteligentes. Algunas de estas tecnologías son revisadas en este capítulo, evaluadas en capítulos posteriores y aplicadas en el presente proyecto.

2.2. Contexto y motivación

2.2.1. Contexto histórico

Los sistemas de generación y distribución de energía eléctrica han crecido en forma exponencial desde sus inicios, debido en gran parte a las demandas del mercado. Sin embargo, este crecimiento se ha dado en volumen sin estar acom-

pañado por una clara actualización de los paradigmas de diseño utilizados. Las primeras redes de distribución eléctrica tuvieron lugar en la década de 1880 en EEUU. En aquel entonces se utilizó energía DC, sin embargo poco tiempo más tarde se presentó la necesidad de migrar los sistemas a corriente alterna para facilitar la distribución. Esta necesidad surgió porque desde los inicios las redes de distribución comenzaron a crecer a un ritmo muy alto debido a la gran demanda de clientes. Esta creciente demanda, y en consecuencia el crecimiento de la red, se dio fundamentalmente por las condiciones económicas del mercado y el modelo de negocios propuesto, en el cual la energía eléctrica era administrada en forma monopólica, convirtiéndola en un negocio sumamente rentable cuyos costos disminuían con el crecimiento de la cantidad de instalaciones. Esto desencadenó un crecimiento exponencial que duró décadas.

Hoy en día la situación es diferente y los requerimientos y características que deberían cumplir las redes actuales, no son los mismos que los de un siglo atrás.

En sus inicios las características del sistema de generación y distribución de energía eléctrica eran, entre otras:

- Negocio monopólico
- Control centralizado
- Flujo de energía unidireccional
- Crecimiento exponencial

En la actualidad la situación es distinta, al menos en países en los cuales existe un alto nivel de acceso a la energía eléctrica. Revisando las características anteriores, puede verse como los requerimientos de la red de hoy van en dirección contraria. A continuación se repasan algunos aspectos en línea con esta observación.

Para empezar, el negocio no es monopólico, o al menos no necesariamente; si bien existe en el Uruguay una empresa pública encargada de administrar la generación y distribución de energía eléctrica a nivel nacional, hay también diferentes empresas privadas que participan del mercado generando y vendiendo energía a la administración.

Hoy en día es posible adquirir equipos de generación eólica, solar, etc. y utilizarlos como fuente de autoabastecimiento. Esto lleva a suponer que el control de sistemas de la envergadura de los actuales, merecería una arquitectura distribuida, a los efectos de mejorar la seguridad y disponibilidad de los servicios.

El flujo de energía ya no es unidireccional. Existen en el Uruguay empresas que generan energía en forma particular para sustentar su consumo, y venden el excedente al ente encargado de la administración de la energía a nivel nacional.

Actualmente el crecimiento de la red se encuentra estancado, o al menos el ritmo de crecimiento ha disminuido significativamente. Sin embargo, el consumo es cada vez mayor y también lo son las exigencias de los clientes en cuanto a calidad de servicio, disponibilidad, seguridad, etc. De modo que el foco deja de ser el crecimiento exponencial en cantidad de suscriptores, y pasa a concentrarse sobre nuevos desafíos como son la eficiencia, la confiabilidad y la robustez del sistema.

Los motivos por los cuales las exigencias de la red actual son tan distintos a los de un siglo atrás son varios, y no se pretende hacer un repaso exhaustivo de los mismos. Algunos de los motivos principales tienen que ver con aspectos económicos y de sustentabilidad. Existen también motivos políticos o estratégicos para algunos países, que apuntan a tener una red más segura y menos vulnerable a ataques. Por otra parte hay también aspectos técnicos que han cambiado, como la necesidad de integrar a la red fuentes de energía cuya explotación no era viable en el pasado.

Todas estos elementos, que definen requerimientos de la red actual diametralmente opuestos a los de antaño, hacen pensar que las redes de hoy en día deberían ser muy diferentes a las de las primeras épocas. Sin embargo, sorprendentemente esto no es así. Las redes de hoy son conceptualmente muy similares a las de principios del siglo XX. Por supuesto que los avances de la tecnología y la electrónica han permitido mejorar algunos aspectos, como la capacidad de monitoreo y análisis de datos, o las proyecciones de consumo. Sin embargo, esto no es suficiente para compensar el crecimiento en volumen de las instalaciones y los mayores desafíos que esto conlleva. Si bien los avances tecnológicos han mejorado algunas características particulares de la red, el diseño de las mismas se sigue basando en los paradigmas establecidos un siglo atrás.

La situación es aún peor teniendo en cuenta que los sistemas de generación y distribución han crecido exponencialmente durante décadas, basados en modelos que fueron desarrollados en su momento para una escala mucho menor. Jesse Sherer hizo una aseveración[1] respecto del sistema eléctrico estadounidense, que puede perfectamente aplicarse a muchos otros:

> La red de EEUU de hoy es un producto de la historia, más que del planeamiento

Esta es una realidad presente en la mayoría de los países con alto nivel de electrificación. Haciendo aún más énfasis en este punto, resulta muy ilustrativa una conocida comparación anecdótica entre dos situaciones completamente diferentes como son la telefonía y las redes eléctricas. En la comparación se imagina la situación en que tanto Edison como Bell pudieran visitar el presente y observar la evolución de las redes eléctricas y la telefonía. Se dice entonces que Bell no comprendería casi nada acerca del estado actual de las comunicaciones. La mayoría de los conceptos y las tecnologías utilizadas le resultaría ajenas. Sin embargo, Edison seguramente reconocería la gran mayoría de las tecnologías utilizadas para la generación y distribución de energía [20].

Steven Collier[2] desarrolla una teoría muy interesante acerca de cómo las redes eléctricas son la excepción a las reglas de Moore y Wright; mientras que en todos los campos tecnológicos los dispositivos son cada vez más pequeños y el costo del hardware es cada vez menor, en lo que respecta a los sistemas de generación y distribución eléctrica esta tendencia no se respeta. Cada vez se construyen plantas más grandes y más caras. Los sensores se mejoran en cuanto a precisión y

[1] http://large.stanford.edu/courses/2010/ph240/sherer1/, Mayo 2015

[2] http://smartgrid.ieee.org/resources/ieee-smart-grid-webinars/past-ieee-smart-grid-webinars, Mayo 2015.

prestaciones, pero su tamaño es el mismo y el costo es aún mayor. Las líneas de transmisión son cada vez más grandes y costosas. La conclusión del expositor es que debido a circunstancias históricas o coyunturales sobre las que no se pretende indagar en el presente trabajo, en lo que respecta a la energía eléctrica no ha habido innovaciones rupturistas a lo largo del tiempo, y estos cambios de paradigma son ahora necesarios y urgentes.

Todos estos factores, en el contexto de un mundo en el que las telecomunicaciones e internet se han masificado, promueven el surgimiento de una tendencia de innovación que se conoce comúnmente con el nombre de Smart Grid (o Red Eléctrica Inteligente).

En el contexto de la Red Eléctrica Inteligente (REI), existen avances en numerosos campos de aplicación, que serán revisados brevemente en el presente capítulo. En cada uno de estos campos, la actividad actual en investigación y desarrollo es enorme. Esto presenta un desafío en lo que respecta al trabajo en el presente proyecto, ya que si bien existe conocimiento y experiencia a nivel internacional sobre los cuales apoyarse, es más grande el terreno inexplorado.

Motivaciones para avanzar hacia una red eléctrica más inteligente

El incremento sostenido de la demanda de energía y la necesidad de incorporar energías limpias a gran escala (ver Fig. 2.1) son algunas de las motivaciones para realizar cambios sustantivos en el sistema eléctrico.

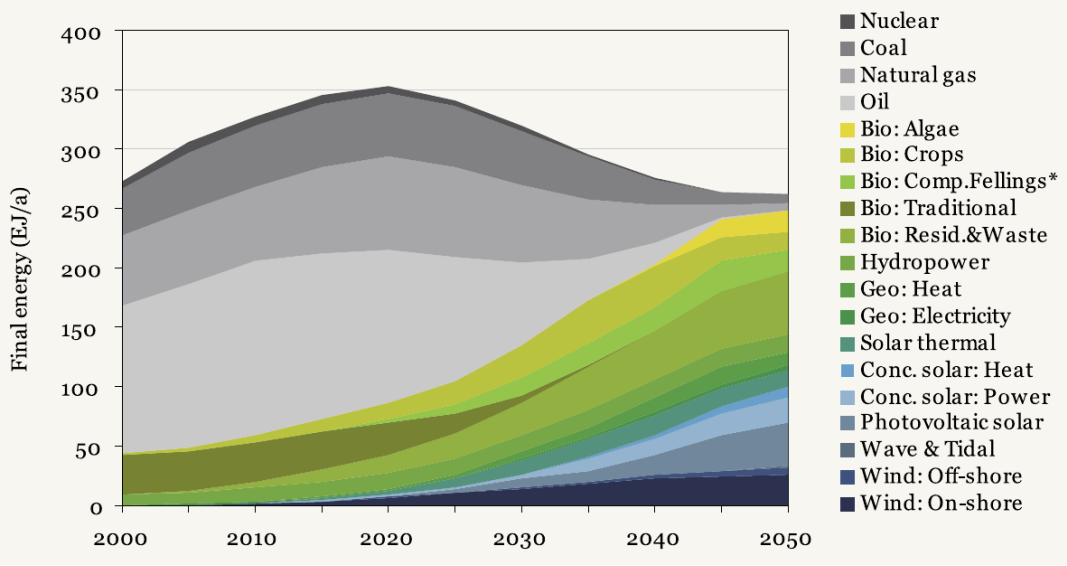


Figura 2.1: Pronóstico de la generación mundial de energía por fuente. Se aprecia una disminución sustantiva de la utilización de combustibles fósiles y por otra parte la proliferación de la energía solar, eólica, biomasa e hidráulica WFF Energy Report, 2010.

No obstante, no son sólo las proyecciones de futuro las que preocupan, sino también que el desempeño de la mayoría de las redes eléctricas actuales presenta importantes ineficiencias, Farhangi [27].

Uno de los antecedentes directos de las Redes Eléctricas Inteligentes son las iniciativas de control de demanda - o *Demand Response* en inglés- que como se detalla en la sección 2.3.4 tienen por objetivo suavizar la curva de demanda, en particular en las horas pico, para obtener niveles de consumo mejor distribuidos a lo largo del tiempo y en las diferentes regiones abastecidas.

Estas estrategias son difícilmente implementables sobre las redes eléctricas clásicas, donde no se cuenta con las herramientas de monitoreo y control adecuadas. La existencia de una red eléctrica más inteligente sería una factor determinante que facilitaría la implementación de este tipo estrategias.

En lo que respecta a la generación de energía, con una participación creciente de las energías renovables y de la microgeneración cambia el paradigma tradicional en el que hay pocas centrales generadoras, y comienza a pensarse en generación automatizada y distribuida. Este nuevo paradigma es viable con redes inteligentes que incorporan el uso de canales bi-direccionales de energía y comunicación.

La generación distribuida de energía tiene una relevancia creciente en función de la proliferación de la microgeneración (10kW o menos): paneles fotovoltaicos, micro generadores eólicos, cogeneradores, etc. Este tipo de generación permite reducir la pérdida de energía al acortar la distancia entre la fuente y el consumidor final, disminuye la vulnerabilidad a las posibles fallas de un generador, y reduce los costos del sistema de distribución al tener un menor flujo de energía transportada, Janaka [29]. Todos estos beneficios, suponen como contraparte la necesidad de contar con redes tecnológicamente superiores a las actuales, que permitan la integración segura y eficiente de los diferentes actores distribuidos.

En resumen, la situación actual a nivel global impone la necesidad de un cambio de paradigma en la concepción y diseño de los sistemas de generación y distribución de energía, y para que estos cambios sean posibles, es necesario agregar un mayor nivel de inteligencia a la red y a los subsistemas que la conforman.

2.3. Conceptos básicos de redes eléctricas y Smart Grid

2.3.1. Redes eléctricas (Grids)

Una red eléctrica es una gran infraestructura interconectada para la entrega de electricidad desde las plantas de energía hacia los usuarios finales, Deng et al., [21]. Se compone de 3 componentes principales: generación, transmisión y distribución.

- *Generación.* Este componente se encarga de proveer de energía eléctrica al sistema, generalmente de forma unidireccional a partir de unas pocas centrales por intermedio de generadores electromagnéticos alimentados principalmente por motores de combustión (a base de combustibles fósiles y no fósiles) o turbinas (hidroeléctricas, eólicas).

- *Transmisión.* La energía generada en las centrales se vuelca a alto voltaje en la red de transmisión donde recorre grandes distancias hasta llegar a las subestaciones de distribución.

- *Distribución.* A la salida de las subestaciones, la energía entra en la red de distribución y se encamina a los puntos de servicio. Al llegar al punto de servicio se reduce el voltaje al valor requerido por el mismo.

2.3.2. Redes eléctricas inteligentes (Smart Grids)

Una *red eléctrica inteligente* (REI, o Smart Grid en inglés), convierte la red eléctrica tradicional en una red bi-direccional tanto de energía como de información. Una REI corresponde a la integración de un sistema de comunicación seguro y bi-direccional entre todos los nodos involucrados en la generación, distribución y consumo de energía eléctrica (IEEE, [9]).

2.3.3. Micro y Macro Grids

Descentralizar la generación permite realizar una división de la red en componentes más pequeños, de forma de aislar zonas ante caídas en la red general. En este sentido, se definen las microgrids como redes interconectadas de sistemas eléctricos distribuidos (cargas y fuentes) que logran funcionar independientemente de la red eléctrica (macrogrid), Farhangi [27]. La microgrid puede aislarse de la macrogrid de acuerdo a las necesidades, en particular cuando los parámetros de calidad de la macrogrid se encuentran por debajo de cierto umbral preestablecido, Lasseter [32].

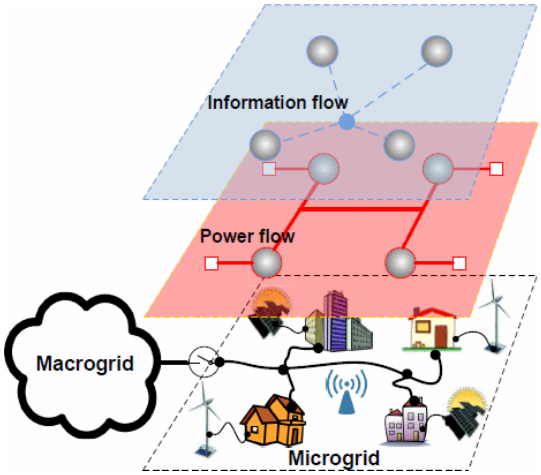


Figura 2.2: Ejemplo de microgrid, [26]. La capa inferior muestra la estructura física de la microgrid, compuesta por edificios, generadores (solares y eólicos) y un punto de acceso inalámbrico. La siguiente capa muestra el flujo que se intercambia por intermedio del tendido eléctrico, y la capa superior representa el intercambio de información entre los componentes del sistema.

2.3.4. Demand Response

Debido a que las pérdidas en las líneas de tensión crecen cuadráticamente con la corriente transportada, una de las formas para lograr un uso más eficiente de la red es mediante el aplanamiento de la curva de demanda, de forma de disminuir el tiempo en el que se transportan grandes flujos de energía.

Adicionalmente, la creciente participación de la energía eólica y solar, de naturaleza estocástica, incrementa la incertidumbre en el pronóstico de la disponibilidad energética, por lo que se debe disponer de una fuente de respaldo que pueda acompasar la generación instantánea con la demand, Varaiya et al., [45]. Esto puede representar un costo importante al contar con plantas subutilizadas y cuya puesta en funcionamiento no es inmediata, por lo que hay grandes esfuerzos por implementar sistemas de gestión de demanda (Demand-side Management, DSM).

Debido a lo anterior, la distribuidora de energía se encuentra motivada a promover una modificación del comportamiento del consumidor, utilizando tarifas dinámicas de forma que los picos de consumo sean recortados en perfiles de demanda suavizados, Fang et al., [26].

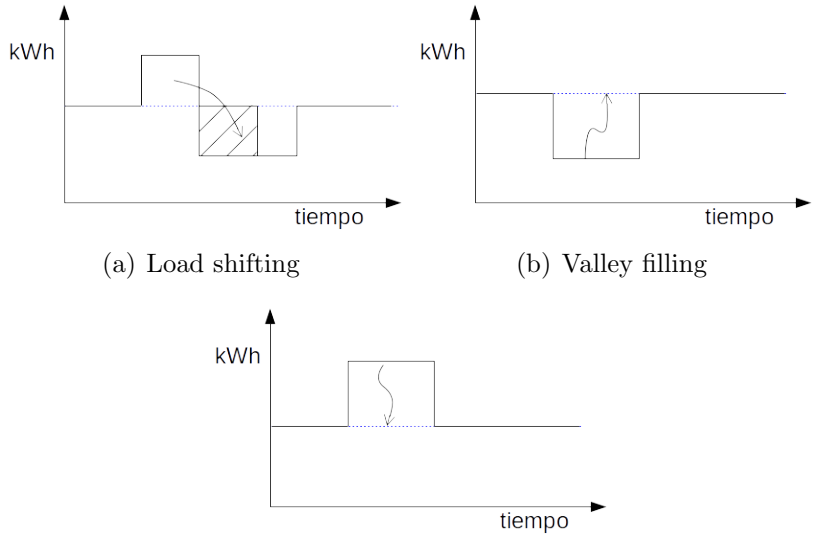
En este sentido, y según la definición de Law et al., [33], *Demand Response* se refiere a "una tarifa o un programa establecido para incentivar cambios en el uso eléctrico por parte de usuarios finales en respuesta a cambios en el precio de la electricidad a lo largo del tiempo, o a través de incentivos económicos diseñados para inducir un menor consumo de energía en momentos de altos precios en el mercado eléctrico o cuando la fiabilidad de la red se encuentra comprometida."

De acuerdo a Law et al., el aplanamiento de la demanda puede realizarse de tres modos distintos: recorte de picos (*peak clipping*), relleno de valles (*valley filling*) y corrimiento de consumo (*load shifting*):

- **Peak clipping**. Se trata de limitar el consumo de energía en momentos de pico, con posible afectación al confort del usuario.

- **Valley filling**. En momentos de alta disponibilidad eléctrica se estimula el consumo de carga de acumuladores térmicos, baterías recargables, autos eléctricos, etc.

- **Load shifting**. Con este método se desplaza temporalmente la actividad de las cargas, sin afectar a priori el confort del usuario.



(a) Load shifting (b) Valley filling

(c) Peak clipping

Figura 2.3: Métodos de aplanamiento de la demanda.

El último paso para implementar DSM es poder contar con un sistema de gestión de energía a nivel de usuario final que reciba información de los distribuidores de energía y gestione las cargas locales.

2.3.5. Energy Management Systems

Los sistemas de gestión de energía (EMS) son sistemas informáticos utilizados por los administradores de la red eléctrica o proveedores de energía para controlar, gestionar y monitorear el desempeño de la red y la matriz de generación y distribución en general. Los EMS, integrados con sistemas de automatización del hogar, juegan un papel fundamental en el control del consumo eléctrico, ya que ponen a disposición del usuario una infraestructura que le permite optimizar el consumo de energía eléctrica [13].

2.3.6. Home Energy Management Systems

Los Home Energy Management Systems son sistemas de optimización de consumo a nivel domiciliario. Típicamente intercambian información con sistemas de gestión global de la red eléctrica (EMS). El objetivo de estos sistemas es monitorear y optimizar el consumo de energía en un domicilio en base a la información recibida por parte del EMS. Por lo general esta información se basa fundamentalmente en datos acerca del precio actual de la energía, o un pronóstico del mismo.

Tal como se explicó en el capítulo 1 el objetivo del presente proyecto es justamente diseñar e implementar una plataforma abierta HEMS, por lo cual este concepto será abordado y estudiado a lo largo de todo el trabajo.

2.4. Tecnologías de comunicación

Cuando se habla de sistemas y tecnologías de comunicación para Smart Grid, existe un gran nivel de acuerdo en diferenciar tres tipos de áreas de aplicación diferentes en base al tipo de información que se desea transmitir y el alcance de la misma. En la bibliografía revisada, la gran mayoría de los autores maneja los siguientes conceptos:

Home Area Network (HAN): Se trata de una red local a nivel doméstico, de la cual podrían formar parte dispositivos de medición (smart metering), dispositivos de control y equipos de uso final.

Neighberhood Area Network (NAN): Es una red que vincula varias HAN, y dentro de la cual puede existir un gateway hacia el sistema general de gestión de la red. La existencia o no de una NAN como tal, diferenciada de la red global, dependerá de la arquitectura definida para el sistema y los paradigmas tomados como referencia. Una posible función de la NAN podría ser la de permitir el intercambio de datos entre domicilios cercanos en un vecindario, facilitando el comercio interno de energía entre ellos y simplificando las tareas vinculadas a gestión de la demanda por parte del sistema global, mediante el análisis intermedio de los datos a nivel barrial.

Wide Area Network (WAN): Sería la red global, que abarcaría toda la extensión del sistema de generación y distribución eléctrica.

Estas definiciones son importantes para comprender las tendencias en cuanto a las tecnologías utilizadas para el intercambio de datos, ya que cada uno de estos niveles de aplicación tiene requerimientos diferentes y singulares, y por ende existen sistemas de comunicación idóneos para un tipo de subred y no para otro. En el libro *Smart Grid - Technology and Applications*, J. Ekanayake [29] realiza un resumen de las diferentes tecnologías utilizadas en cada nivel de aplicación. Luego de una vasta revisión bibliográfica se constata que el resumen es de hecho muy acertado, ya que todos los casos de aplicación encontrados utilizan tecnologías de comunicación incluidas en dicho resumen. Los autores Fan et al., [25], realizan una síntesis similar, enriqueciendo el resumen anterior. Tomado como referencia estas fuentes, se realiza el siguiente resumen de idoneidad y popularidad de sistemas de comunicación para cada una de las áreas de aplicación.

Poniendo foco en el proyecto que nos ocupa, existen dos tipos de sistemas de comunicación de interés. El primero correspondería a una HAN mediante la cual el controlador pueda comunicarse con los dispositivos periféricos. El segundo, se trata de una NAN/WAN a la que el controlador se conectará para llegar hasta el eventual sistema de gestión general de la red. A priori, luego de una primera revisión bibliográfica, se destaca el hecho de que ZigBee parece ser la tecnología más popular en el caso de la HAN, aunque 6LoWPAN es una variante más reciente que podría tomar fuerza rápidamente dentro de las comunicaciones inalámbricas.

Red	Tecnología de comunicación
HAN	ZigBee, 6LoWPAN, BlueTooth, Wi-Fi, Ethernet, PLC, BPL
NAN	ZigBee, Wi-Fi, Metro, Ethernet, PLC,BPL, DSL, EDGE, HSPA, UMTS, GSM, LTE, WiMax, Frame Relay
WAN	Ethernet, microwave, WiMax, 3G/LTE, fibra óptica, MPLS, Frame Relay

Tabla 2.1: Tecnologías utilizadas para las diferentes áreas de aplicación de redes en Smart Grid.

En el caso de la WAN, no existe una tendencia tan clara, y aquí entran en juego las características geográficas y coyunturales de cada región. En el caso de Uruguay, a priori parecería razonable la conformación de una WAN utilizando tecnologías IP sobre fibra óptica, dada masificación de este medio físico en los últimos dos años. Debe quedar claro que estas son simplemente observaciones. En el capítulo 4 se presenta un estudio e investigación exhaustiva de estos temas y recién en esa instancia se arriba a conclusiones determinantes, las cuales serían prematuras en esta etapa de revisión.

2.5. Síntesis del capítulo

En el presente capítulo se brindó un marco general acerca del tema que nos ocupa, abarcando el contexto histórico, motivaciones y estado del arte. Se presentaron además algunos conceptos importantes y un resumen de la revisión bibliográfica realizada al comienzo del proyecto, donde se puede ver un enfoque general y tendencias en Smart Grid.

Capítulo 3

Diseño conceptual

La plataforma abierta HEMS presentada en este trabajo está compuesta por un controlador central y *nodos* encargados de tomar mediciones y de encender o apagar los electrodomésticos. Además de una interfaz de usuario web, ofrece una biblioteca de funciones básicas (API) para facilitar la integración de cualquier algoritmo de control. Mientras los *nodos* se comunican con el controlador central mediante una red local HAN (*Home Area Network*), este último intercambia información con un sistema de gestión general de la red que al no existir aún en Uruguay será modelado según ciertas hipótesis.

El objetivo de este capítulo es presentar un primer diseño conceptual de los componentes de la plataforma, los mecanismos de comunicación entre ellos, y la forma en que la misma interactúa con el sistema general de gestión de la red eléctrica.

3.1. Descripción general del sistema

3.1.1. Modelo del problema y sus actores

En el Capítulo 1 se presentaron algunas características generales del esquema en el cual se enmarca el proyecto. Se explicó allí que por un lado se toma como hipótesis la existencia de un agente externo, administrador y proveedor de la energía eléctrica, y por otra parte se considera al consumidor particular, en nuestro caso de carácter domiciliario. En la Fig. 3.1 se observa un esquema conceptual de este escenario.

Para lograr una red más inteligente, se necesita que tanto los proveedores o administradores de la energía, como también los consumidores, sean cada vez más inteligentes. En el caso del ente administrador y proveedor de energía, su estudio queda por fuera del alcance del presente trabajo. Sin embargo, se trabajará sobre la base de algunas hipótesis acerca de este actor.

El caso del consumidor final de carácter domiciliario concentra la mayor atención de este trabajo. A la hora de trabajar a este nivel, se considera a cada usuario final como una célula independiente, que únicamente interactúa puertas afuera con

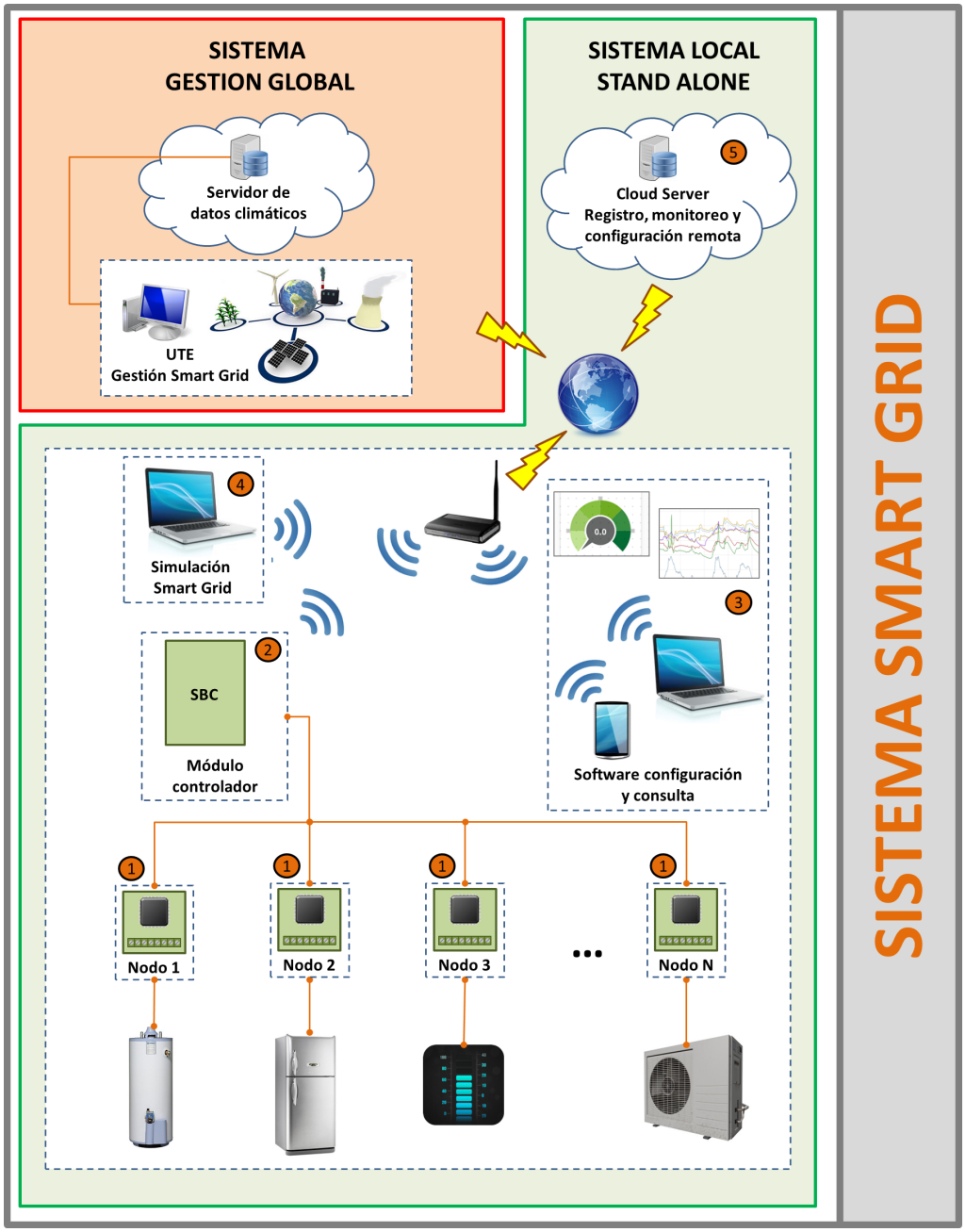


Figura 3.1: Diagrama general del sistema.

el sistema de gestión general de energía administrado por la empresa de suministro.

Con el objetivo de facilitar la lectura, de aquí en más se usará el término *Energy Management System* (EMS), para hacer referencia al sistema de gestión de energía a nivel nacional y *Home Energy Management System* (HEMS) para hacer referencia al sistema de optimización local domiciliario.

Al optimizar el consumo a nivel del usuario domiciliario, se busca disminuir sus gastos por concepto de energía sin tener en cuenta necesariamente lo que suceda en el resto de la red o lo que hagan los demás consumidores. Desde la perspectiva de una red eléctrica inteligente, el comportamiento del resto de los consumidores sí tendrá influencia sobre las decisiones óptimas de cada consumidor particular, pero dicha influencia se verá reflejada a través de la información intercambiada con la empresa de suministro, ya sea a través de variaciones en la tarifa o del envío de solicitudes o directivas. Sin embargo a los efectos del diseño, el sistema local ubicado en cada hogar y encargado de la optimización para ese consumidor particular, se comportará como un sistema cerrado a excepción de la comunicación con el supuesto sistema EMS. Es importante precisar este detalle ya que permite el estudio del sistema a nivel domiciliario sin necesidad de tener en cuenta cualquier otro factor o actor externo involucrado en la red o el mercado eléctrico.

3.1.2. Modelo e hipótesis acerca del sistema EMS

A los efectos del desarrollo de soluciones orientadas a los paradigmas de REI (Redes Eléctricas Inteligentes, o Smart Grids), se hace la hipótesis de que existe un sistema EMS de gestión de la red que cuenta con la información y herramientas necesarias para brindarle a los consumidores todos aquellos datos que podrían ayudarlo a bajar los costos de su consumo eléctrico en el marco de una estrategia de control de la demanda o *demand response*.

En Uruguay, este rol podría ser asumido por UTE o ADME. Sin embargo, en la actualidad este tipo de estrategias no están implementadas en este país y es por este motivo que se proponen ciertas hipótesis acerca del tipo de información que el EMS podría poner a disposición de los usuarios, así como también acerca de los medios a través de los cuales dicha información sería provista. Se supone entonces que la información de la que se dispondrá por parte del sistema EMS será:

- Pronóstico de tarifa para las próximas 24hs, actualizado una vez al día a una hora determinada.
- Actualización de dicho pronóstico para la hora actual al comienzo de la misma, dentro de un cierto intervalo de confianza con respecto al pronóstico diario.

La información mencionada es de fundamental importancia para lograr un alto nivel de optimización del consumo, tanto a nivel local del usuario como a nivel global de la red (Bloustein [18]). Se han implementado numerosos programas que logran optimizar el consumo de un domicilio en base a estos datos (Barbose et al., [16]). Por tal motivo cualquier sistema de optimización de energía domiciliario debe prever la posibilidad de comunicarse con el EMS y acceder a ellas.

Si bien se contempla la posibilidad de recibir comandos para ciertos electrodomésticos desde el EMS con intención de actuar directamente sobre estos, no se realizarán mayores hipótesis acerca de cómo podría implementarse este envío de directivas, ya que su definición debe estar necesariamente ligada a modelos comerciales y de calidad de servicio que no son objeto de estudio del presente trabajo.

Simplemente a modo de ejemplo, el consumidor podría optar por un contrato en el cual el proveedor de energía pudiera solicitar mediante una directiva la reducción del consumo instantáneo a un mínimo pautado, una cantidad máxima de veces al año previamente acordada.

A pesar de no implementarse la funcionalidad de aceptar directivas del EMS para actuar directamente sobre los electrodomésticos, el diseño del controlador brinda herramientas para poder adicionar esta funcionalidad fácilmente.

En cuanto a la comunicación entre el EMS y el HEMS, dado el alto nivel de acceso a internet existente en el Uruguay [1], y que este acceso es proporcionado por un ente estatal, resulta razonable considerar que la información sobre la tarifa será brindada al usuario a través de Internet. Ya existen experiencias en otros países en relación a este método de comunicación. Por ejemplo, la empresa ComEd, proveedora de energía eléctrica en el estado de Illinois, US, ofrece un portal con la información de precio dinámico de la energía [2].

A los efectos del diseño del sistema HEMS se considera que el mismo podrá acceder a la información de tarifa dinámica (Real Time Pricing, RTP) a través de un servidor web. Dado que tal recurso aún no existe en Uruguay, se simula el método de intercambio de datos descrito a través de un un servidor propio.

3.1.3. Arquitectura básica del sistema HEMS

En esta sección se aborda el diseño conceptual del sistema de control a nivel domiciliario, considerándolo como una célula independiente dentro de la red, cuya única interacción con agentes externos será a través del intercambio de información RTP con el EMS.

El objetivo del HEMS es realizar una optimización local del consumo de energía eléctrica a nivel del domicilio, entendiendo esta optimización tanto en términos absolutos de la cantidad de energía como del costo económico asociado en función de la hora del día en que se efectúa dicho consumo. Esta optimización se realiza mediante un algoritmo que tiene como entradas la información de RTP, los datos de consumo actual de los electrodomésticos y los requerimientos de confort del usuario. La salida del algoritmo consiste en órdenes de comando hacia los distintos electrodomésticos ya sea para encenderlos o apagarlos.

Entradas del sistema:

- Datos de precio dinámico provenientes del EMS.

[1] http://www.antel.com.uy/antel/antel-en-el-mundo/uruguay-el-pais-de-la-fibra-optica-al-hogar, Mayo 2015.

[2] https://rrtp.comed.com/live-prices/, Mayo 2015.

- Datos de consumo actual de los electrodomésticos.
- Datos de configuración del usuario, en base a hábitos y preferencias de confort.

Salidas del sistema:

- Comandos de control de los electrodomésticos.

Tal como se mencionó en capítulos previos, el enfoque del proyecto no es el de implementar un HEMS cerrado, con un algoritmo de optimización predefinido, sino lograr una plataforma abierta y versátil, capaz de facilitar la implementación de algoritmos que trabajen en base a las entradas y salidas de información previamente definidas.

En lo que respecta a los datos de hábitos y confort del usuario, los mismos deberán ser tenidos en cuenta al momento de programar los algoritmos de optimización y control, mediante el uso de las funcionalidades provistas por la API.

Tanto para la obtención de los datos de consumo actual de los electrodomésticos, como para la ejecución de comandos para controlarlos, se precisa contar con algún mecanismo de comunicación con los mismos. Existe en la actualidad una tendencia tecnológica conocida como *Internet of Things* o *Internet de las Cosas* (de aquí en más IoT) según la cual se pronostica que en el futuro una gran parte de los objetos, en particular los electrodomésticos típicos, contarán de serie con conectividad a Internet. Los electrodomésticos que cuentan con una interfaz de comunicación a través de la cual pueden ofrecer sus datos de funcionamiento y consumo, y recibir instrucciones, set points o directivas de comando, se conocen como *Smart Appliances*. En la actualidad esa realidad es aún lejana; la mayoría de los electrodomésticos típicos, y particularmente los que presentan mayores consumos en un domicilio, están lejos de ofrecer algún nivel de inteligencia o interfaces de comunicación. Para que esto no sea un obstáculo para la implementación de sistemas de optimización domiciliario, es necesario agregarle al HEMS ciertos módulos de conexión con dichos electrodomésticos, de modo de convertirlos de alguna forma en *Smart Appliances*, o al menos lograr acceder a información acerca de sus estados y consumos, y obtener la posibilidad de comandarlos en forma remota. De aquí en más se utiliza el término como *nodos* para hacer referencia a tales módulos de conexión con los electrodomésticos.

En la Fig. 3.2 se muestra el esquema de solución propuesto para la arquitectura del HEMS.

El módulo principal y núcleo del sistema es el *controlador*, donde se reúnen los datos provenientes del EMS y de los electrodomésticos. Estos se procesan mediante un algoritmo de optimización, y se emiten comandos de encendido o apagado hacia los nodos. De aquí en más se hará referencia al controlador como *Home Energy Controller (HEC)*. El HEC es entonces el encargado del procesamiento de los algoritmos de optimización y ofrece una interfaz de usuario de modo tal que éste pueda monitorear el consumo, y controlar los diferentes electrodomésticos.

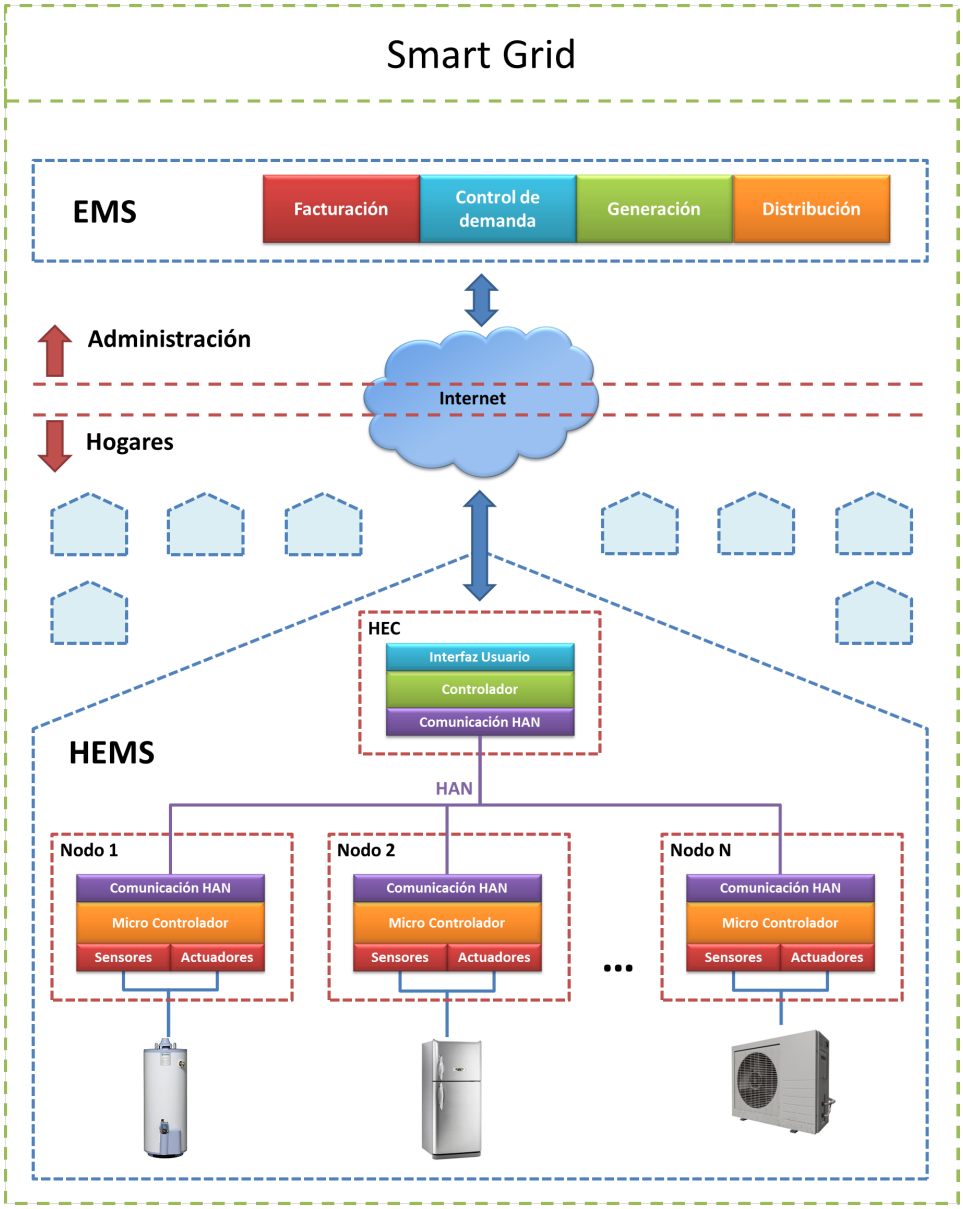


Figura 3.2: Arquitectura general del HEMS.

Por otra parte el sistema cuenta con un conjunto de nodos que le permiten reunir información de consumo y actuar sobre los electrodomésticos. Los nodos están compuestos por sensores que permiten realizar mediciones de las principales variables eléctricas de inteŕes, y cuentan por otro lado con actuadores ON/OFF que permiten encender o apagar los artefactos. Aunque no se contemplan en esta primer versión, se podría incorporar un módulo de control infrarrojo para comandar, por ejemplo, equipos de aire acondicionado. También es posible conectar un nodo a una toma de energía de la que se alimenta más de un equipo, o incluso a la entrada de energía del domicilio. Esto serviría por ejemplo para realizar un

monitoreo del consumo total del hogar. Se prevé incluso la posibilidad de contar con nodos equipados con otro tipo de sensores, como por ejemplo de temperatura o luminosidad, para obtener información del entorno que puede ser valiosa para optimizar el uso de ciertos artefactos. Estos aspectos se abordan con más detalle en secciones posteriores.

Para permitir la interacción con los nodos es necesario contar con un sistema de comunicación entre estos y el HEC. Este sistema de comunicación constituye una red local intra domiciliaria. Este tipo de redes suele categorizarse con el nombre de *Home Area Network* (HAN). Este es el término que se utilizará de aquí en más para hacer referencia a la red de comunicación entre los nodos y el HEC.

Existen diversas tecnologías y topologías posibles para conformar una HAN. Los requerimientos y características de la misma son abordados con mayor detalle en secciones posteriores.

3.2. Sistema de comunicación: Home Area Network

La HAN constituye la plataforma de comunicación entre los nodos conectados a los electrodomésticos y el HEC. Se trata de una red intra domiciliaria, compuesta de un HEC y tantos nodos como electrodomésticos se desee monitorear y controlar.

3.2.1. Análisis de requerimientos de la HAN

El objetivo de la HAN es brindar un canal de comunicación bi-direccional y confiable a través del cual el HEC pueda acceder a los datos de consumo de los electrodomésticos medidos por los nodos, y por otro lado enviar mensajes de comando a los nodos para controlar sus artefactos asociados.

Diversos factores hacen que la topología de la red sea inevitablemente variable. Desde la incorporación de nuevos electrodomésticos, la eventual conexión o desconexión de los ya existentes e incluso debido a la variación en las condiciones de radio, la HAN debe ser versátil y estar preparada para enfrentar una topología dinámica. El sistema de comunicación debe ser capaz de auto configurarse ante cambios topológicos e incluso en el caso más extremo en el que se sufra un corte de energía o el controlador se apague, la HAN debe ser capaz de reiniciarse y reconfigurarse automáticamente una vez normalizada la situación.

En cuanto al número de interlocutores soportados por la HAN además del HEC, esto es, el número de nodos soportados, parece razonable afirmar que el mismo presentará un máximo del orden de las decenas. Este número es completamente arbitrario. Sin embargo parece difícil pensar que un hogar pudiera contar con más de 100 artefactos a controlar en forma independiente.

Se propone un diseño en el cual los nodos tengan la capacidad de sensar y pre procesar las mediciones adquiridas de modo de enviar datos de medidas ya calculadas, y no muestras de señales o datos brutos en grandes cantidades. Bajo esta hipótesis, que se justifica en la Sección 3.3, los requerimientos en cuanto al ancho de banda o flujo de datos a soportar son relativamente bajos. Por otra parte, los comandos enviados por el HEC implican un intercambio reducido de

información hacia cada nodo. Para determinar el orden de magnitud del ancho de banda necesario se considera que el intercambio de datos y comandos entre el HEC y cada nodo se realiza una vez por segundo. Suponiendo 20 bytes de datos por cada intercambio (hipótesis por demás conservadora), en el peor escenario con 100 nodos conectados se necesitaría un ancho de banda de 2 KB/s en términos de throughput.

Otro aspecto a considerar es el de la latencia permitida. Aquí la restricción vendría impuesta por los algoritmos de optimización utilizados, ya que los mismos realizan decisiones lógicas en función de las medidas de los nodos y la información de RTP, y actúan sobre los electrodomésticos activándolos en los momentos óptimos. Si la latencia en la comunicación entre HEC y nodos fuera excesiva, la efectividad de los algoritmos podría verse comprometida. Dado que el HEMS a implementar pretende ser una plataforma abierta que soporte diferentes algoritmos, se realizó una revisión de algunos de ellos (e.g. Xiong et al. [48]) para obtener una idea más rigurosa acerca de estos requerimientos. En base a esta revisión se encontró que latencias del orden de unos segundos son perfectamente aceptables.

En cuanto a los medios físicos para la comunicación, un sistema de comunicación con cableado dedicado presentaría serias desventajas para la aplicación por temas de practicidad para el usuario y costos de instalación. Sin embargo existen varias tecnologías que podrían ser utilizadas para resolver la HAN en cuestión. Dichas tecnologías podrían ser agrupadas en una primera clasificación de la siguiente manera:

- Inalámbricas
- PLC (Power Line Communication)

PLC es una tecnología madura y cada vez más difundida y ha sido aplicada en arquitecturas similares (Salvadori et al. [40]). Sin embargo, presenta la desventaja de solo permitir la incorporación de nodos que se conecten a la red eléctrica. En secciones anteriores se explicó que si bien el objetivo principal de los nodos es el de medir variables eléctricas de consumo, se prevé también la utilización de nodos con sensores de otros tipos. Por ejemplo, en el supuesto de que se desee utilizar un nodo con sensor de temperatura para comandar y optimizar el funcionamiento de un equipo de climatización, o un sensor de nivel de luminosidad para controlar un sistema de iluminación, es muy probable que la mejor ubicación para dichos sensores, y por ende para el nodo, no coincida con la ubicación de algún tomacorriente disponible. En ese caso sería deseable poder utilizar un nodo alimentado a batería, y comunicado con el HEC en forma inalámbrica.

Teniendo en cuenta lo anterior, para este trabajo se consideran las tecnologías inalámbricas como la mejor opción para el diseño del sistema, dada la versatilidad y flexibilidad que ofrecen. Sin embargo, la integración de tecnologías basadas en PLC resulta un tema de interés para trabajo futuro.

Si bien las comunicaciones inalámbricas ofrecen muchas ventajas, también presentan algunos desafíos. En particular se debe considerar el alcance. En caso de utilizar tecnologías que no ofrezcan grandes coberturas, o sufran atenuaciones o

interferencias típicas en un hogar que degraden la performance, debe soportarse el uso de topologías tipo *mesh*, en las que cada interlocutor puede funcionar como repetidor para extender el alcance de la red hasta los interlocutores más lejanos. En el contexto de este trabajo, el objetivo es cubrir las dimensiones de un domicilio típico. Esto es, nodos distribuidos a lo largo y ancho de superficies que pueden llegar a los cientos de metros cuadrados, con paredes entre ellos y la existencia de interferencias de radio típicas de una zona urbana.

3.2.2. Resumen de requerimientos de la HAN

A partir del análisis realizado en párrafos anteriores, se desprenden los siguientes requerimientos básicos para la implementación de la HAN:

- Robustez y estabilidad.
- Topología dinámica y autoconfigurable.
- Bidireccionalidad entre HEC y nodos.
- Número de interlocutores a soportar: HEC + 100 nodos.
- Ancho de banda mínimo: 2 KB/s de troughput.
- Latencia admitida máxima: 1seg.
- Medio de comunicación: Inalámbrico.
- Amplia cobertura, o en su defecto soporte de topologías mesh.

3.3. Los nodos

Una característica esencial de un sistema de control en el contexto de REI, es la capacidad de obtener información del consumo de los electrodomésticos. Esta información será la base para las decisiones que tome el controlador, y además sirve para que el usuario pueda ser consciente del impacto de sus hábitos de consumo, promoviendo un comportamiento más eficiente (Aman et al. [14]).

3.3.1. Descripción de los nodos

Los nodos tienen una doble funcionalidad; mientras recaban información del ambiente (temperatura, humedad, presencia humana) o de una carga (corriente, voltaje, potencia consumida) al mismo tiempo deben poder actuar para comandar estas últimas. Toda la información procesada por los nodos es transmitida al HEC de forma de poder correr los algoritmos de optimización. Dado que se pretende limitar el uso de ancho de banda en la HAN, es necesario que los nodos cuenten con una capacidad mínima de procesamiento de forma de disminuir el volumen de información intercambiada.

Un nodo debe contar entonces con las siguientes funcionalidades:

- Medición
- Procesamiento
- Comunicación
- Comando

Es posible que hayan nodos que no tengan necesidad de sensar y comandar simultáneamente, por ejemplo en el caso de un sensor de temperatura ubicado lejos de una carga y que solo tiene interés a los efectos de recabar información del ambiente. También puede darse el caso donde sólo haya necesidad de realizar control sin necesidad de sensar señales. Sin embargo, las funcionalidades de comunicación y de procesamiento deben estar presentes en cualquier caso.

Por lo descrito anteriormente, el nodo debe concebirse con un conjunto de funcionalidades independientes, distinguiendo tres tipos de nodos: *Medición*, *Comando* y *Medición&Comando*, tal como se muestra en la Fig. 3.3.

Todas las configuraciones comparten el microcontrolador (MCU) encargado de procesar las señales y de gestionar la radio que asegure la comunicación con la HAN. Para la función de sensado es necesario contar en primer lugar con un transductor, que convierta las variables físicas en una señal eléctrica. Además es necesario colocar una etapa de acondicionamiento de la señal de forma de adaptarla a los rangos de funcionamiento del ADC, un filtro antialiasing antes de realizar el muestreo de la señal, y una protección eléctrica por sobretensión en la entrada del MCU.

Un supuesto adicional al momento de diseñar los nodos es que estos deben basarse en una plataforma de bajo costo, de forma de hacer viable su adopción masiva en un futuro.

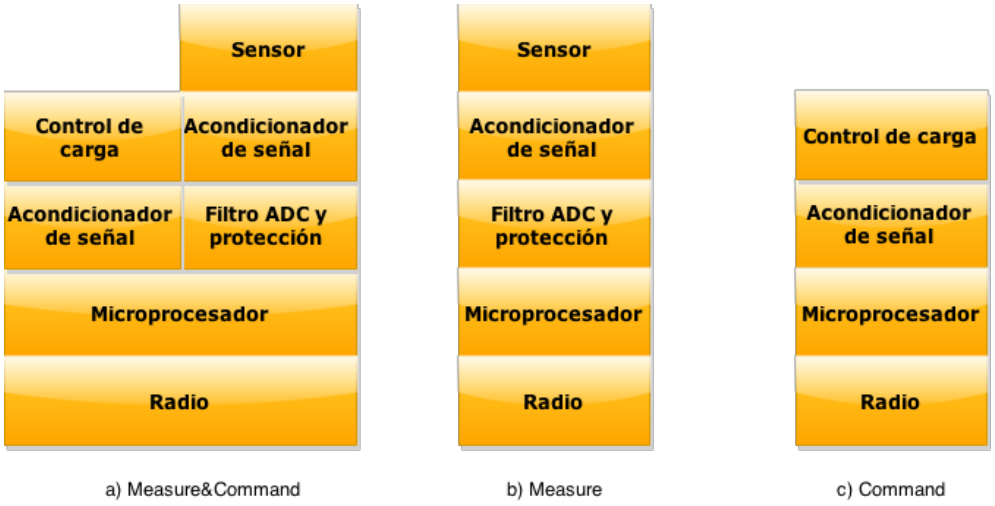


Figura 3.3: configuraciones posibles para los nodos.

Medición

Si bien el diseño del sistema debe admitir la inclusión de numerosos sensores que den cuenta del ambiente y del estado de los electrodomésticos a monitorear y controlar, dado el alcance del actual proyecto, se decide implementar únicamente un sensor que reporte medidas de corriente, voltaje y potencia consumida por las cargas.

Estas medidas permiten conocer el estado de las cargas, posibilitando el monitoreo del consumo, y en base a este, el comando de los electrodomésticos por parte del controlador.

En un primer relevamiento de los electrodomésticos se establece como requerimiento la posibilidad de medir cargas de hasta 2.5kW. Dado que la tensión de alimentación de los electrodomésticos está acotada por el voltaje de la red y asumiendo que $Vred \leq 250$ VAC, el sensor debe ser capaz de medir corrientes de hasta 10 A. Finalmente, se espera acotar en el orden de un 5 % la incertidumbre de la potencia relevada (incluyendo aquí los errores introducidos en la etapa de procesamiento).

Al diseñar la etapa de medición se tiene en cuenta que las señales de salida deben ser debidamente acondicionadas para su posterior procesamiento. El objetivo es realizar el muestreo de forma tal que permita contar con una mínima distorsión en los 10 primeros armónicos de la frecuencia fundamental ($f_{red} = 50Hz$), requiriendo tener una frecuencia de corte de al menos 500Hz. Esto debe tenerse en cuenta a la hora del filtrado de señales y su adquisición. Un diagrama conceptual se presenta en la Fig. 3.4.

Los requerimientos son entonces:

- $Vmax \geq 250$ VAC
- $Imax \geq 10$ A
- Incertidumbre en el orden del 5 %
- Frecuencia de corte de 500Hz

Procesamiento

Al definir las capacidades necesarias para el procesamiento de señales se deben identificar en primer lugar las operaciones requeridas para reportar los valores de voltaje, corriente y potencia (activa y reactiva).

Para estas operaciones es necesario ubicar un compromiso entre capacidad de procesamiento local de la información y el ancho de banda requerido por el sistema de comunicación. Sería posible enviar todas las muestras de corriente y voltaje relevadas por los ADC de forma que sea el propio HEC quien realice el procesamiento de las señales. Sin embargo, se decide realizar en el nodo tareas de pre-procesamiento, de forma de disminuir sensiblemente el volumen de datos intercambiados.

De esta forma, el MCU debe ser capaz de muestrear la señal analógica y realizar las siguientes operaciones:

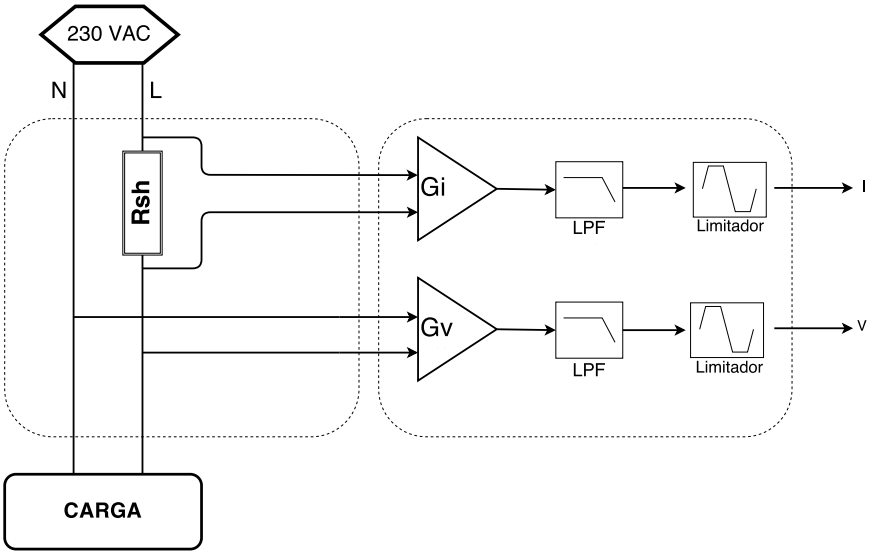


Figura 3.4: Diagrama conceptual medición.

$$I_{rms} = G_{I.}\sqrt{\frac{1}{N}.\sum_{k=1}^{N} I_k^2} \tag{3.1}$$

$$V_{rms} = G_{V}.\sqrt{\frac{1}{N}.\sum_{k=1}^{N} V_k^2} \tag{3.2}$$

La potencia activa y reactiva se calcula de la siguiente manera:

$$P = \frac{1}{N}.\sum_{k=1}^{N} V_{rms}(k).I_{rms}(k), \tag{3.3}$$

$$Q = \frac{1}{N}.\sum_{k=1}^{N} V_{rms}(k).I_{rms}(k - N/4) \tag{3.4}$$

Finalmente, el nodo debe poder comunicarse con la HAN a través de una radio gestionada por el MCU, que soporte el protocolo de comunicación implementado.

Comando

A los efectos de implementar algoritmos de optimización de energía es necesario poder actuar sobre los electrodomésticos del hogar. De forma de poder testear el

correcto funcionamiento de la plataforma, se decide implementar un comando de control ON/OFF accionado a través de un relé. El control del relé es gestionado por el MCU a través de uno de los puertos de salidas digitales.

En futuros trabajos se podrían incluir otros métodos de control, por ejemplo: regulación de la intensidad de dispositivos luminosos, transmisores IR para controlar dispositivos de aire acondicionado, etc. En el alcance actual del proyecto se diseña un nodo de comando capaz de activar y desactivar cargas de hasta $2,5kW$.

La integración de los distintos componentes del sistema se esquematiza en la Fig. 3.5.

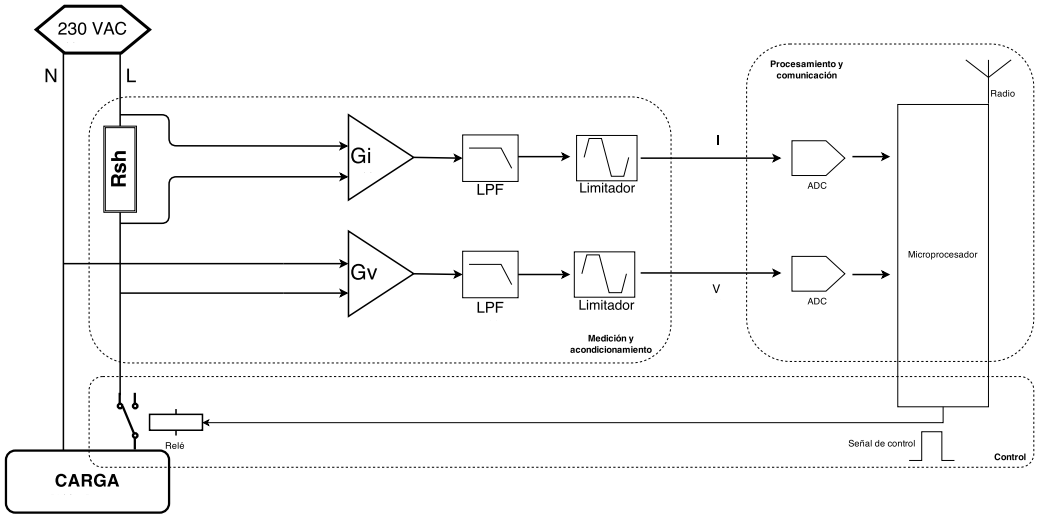


Figura 3.5: Esquema conceptual de los nodos.

3.4. Home Energy Controller

3.4.1. Descripción general del controlador

El controlador central (HEC) es el componente encargado de gestionar la comunicación con todos los nodos del sistema, recibir información del EMS y manejar la interfaz con el usuario.

El HEC es el componente donde se ejecutarán los algoritmos de optimización de consumo, se centralizará en él todo el manejo de la información obtenida de los sensores (medidores) y el comando de los dispositivos.

Debido a que los nodos son diseñados en base a dispositivos de bajos recursos, no es viable asignar más tareas a estos, y se decide centralizar el procesamiento de los datos en un único componente. Esto significa que los nodos no son capaces de enviar sus medidas directamente a la interfaz de usuario y por ende una eventual falla en el HEC dejaría inutilizable al sistema, siendo este un punto a analizar en cuanto a las posibles mejoras futuras que se mencionan al final del documento. Sin embargo, siendo el objetivo del diseño un hogar típico, en el cual se espera que la

cantidad de electrodomésticos o líneas de carga esté en el orden de las decenas, se entiende que resulta un compromiso adecuado entre robustez y simplicidad.

Además de proporcionar un sistema básico de monitoreo, comando y optimización de consumo, el principal objetivo del diseño del HEC es la implementación de una API (Application Programming Interface). Dicha API se plantea como una abstracción de software que eventualmente permitiría a distintos grupos de investigación implementar y probar algoritmos de optimización de consumo.

La Fig. 3.6 muestra un diagrama general de las funciones del HEC. Aquí se aprecian los tres módulos de software que conforman la API abierta y el lugar que ocupa el módulo de optimización, haciendo uso de esta.

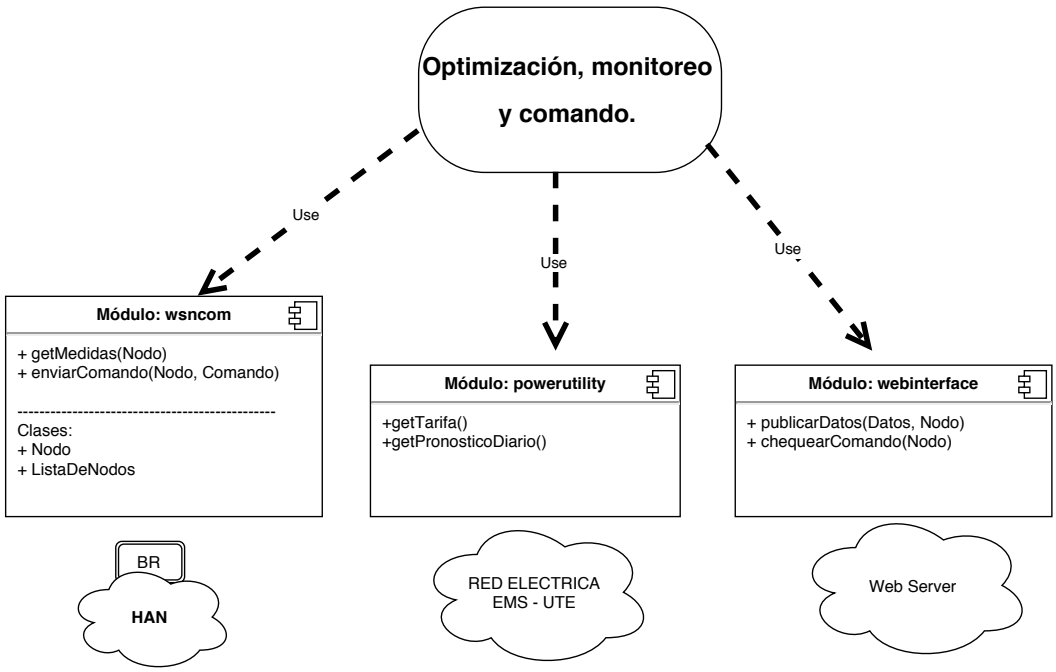


Figura 3.6: Diagrama funcional del HEC.

En el diagrama también se muestran los tres elementos del sistema general; la HAN, la interfaz web y el EMS de la empresa distribuidora de energía, representando la interacción de estos con el HEC. Se aprecia cómo los módulos de software diseñados ofrecen interfaces para la interacción con cada uno de estos tres elementos.

En el Capítulo 6 se explicará el diseño detallado de cada uno de los tres módulos, para luego describir cómo se implementó este diseño en un lenguaje de programación particular.

Para la implementación de la interfaz de usuario se propone utilizar soluciones de código abierto. Se requiere que los usuarios puedan visualizar a través de la web las medidas actuales e históricas reportadas por todos los dispositivos.

3.4.2. Optimización, monitoreo y control usando API diseñada

Se propone un módulo de optimización básico capaz de monitorear el estado de los electrodomésticos conectados, reportando su consumo mediante la interfaz web de usuario. Además, este módulo permite al usuario interactuar con los dispositivos, encendiendo y apagando los mismos a través de la interfaz web.

Se propone evaluar la plataforma implementando el algoritmo de optimización propuesto por Pengwei Du et al., en [22].

El módulo de optimización diseñado se encarga de ejecutar tres tareas de forma simultánea:

- Optimización del consumo de un termotanque.
- Reporte de mediciones a la interfaz web.
- Comando de dispositivos a pedido del usuario.

Para ello mantiene una lista con referencias a todos los dispositivos conectados. En las siguientes subsecciones se describe brevemente el diseño de cada una de estas tareas ejecutadas por el HEC.

Reporte de mediciones

Para realizar el reporte de mediciones, el HEC debe recorrer la lista de nodos, y para cada uno de ellos consultar la medida. Luego de obtenida la medida, esta es publicada en la interfaz web para que quede disponible para visualización por parte del usuario. De esta forma se obtiene información en tiempo real del consumo de todos los nodos.

Comando remoto de dispositivos

A pesar de que el comando manual de los dispositivos en forma remota no estaba contemplado dentro del alcance del proyecto, se considera interesante brindar esta funcionalidad al usuario. Para ello se propone incorporar una opción de comando de los nodos con botones en la interfaz web.

Optimización del consumo de un termotanque

Para evaluar la plataforma se propone implementar el algoritmo de optimización del termotanque propuesto por Pengwei Du et al., en [22]. Dado que el usuario podrá comandar el termotanque en forma manual, se debe incorporar un criterio de prioridades con respecto al algoritmo. El criterio para establecer esta política de control tendrá como prioritarios los comandos manuales del usuario por sobre los comandos ejecutados en forma automática como resultado del algoritmo.

3.5. Síntesis del capítulo

En el presente capítulo se presentó una descripción general del modelo del problema y de la arquitectura de la solución propuesta.

Se propuso trabajar sobre el escenario de una REI compuesta por un sistema de administración de la red eléctrica y la energía y un universo de consumidores domiciliarios equipados con un sistema de optimización local.

Si bien el sistema EMS no existe en el Uruguay al momento de la realización de este trabajo, se realizaron supuestos y se ofreció un modelo hipotético para el mismo, en el cual una de las hipótesis fundamentales es que este ofrece un pronóstico diario del precio de la energía para las siguientes 24 hs, así como el precio actualizado en tiempo real (hora a hora).

En el caso de los consumidores, se considera que cada uno de ellos se comporta como una célula independiente de las demás, que sólo interactúa puertas afuera del domicilio con el EMS.

Se presentó también el esquema conceptual de la solución propuesta HEMS, donde se describen los diferentes módulos que lo componen. Dicho sistema está compuesto por un controlador principal (HEC) y por nodos que implementan la conexión con los electrodomésticos para realizar mediciones y enviar comandos de encendido y apagado. El HEC se comunica con los nodos a través de una red HAN inalámbrica.

El HEC se encarga de comunicarse con el EMS para recibir datos de RTP (Real Time Pricing). A su vez ejecuta algoritmos de optimización a partir de los cuales controla el encendido y apagado de los electrodomésticos en base a su estado y consumo actual, y tomando en cuenta la información de RTP recibida. Su configuración se realiza a través de una interfaz de usuario y brinda un conjunto de bibliotecas de software con directivas simples disponibles para el desarrollador, para facilitar la implementación algoritmos de optimización.

Capítulo 4

Diseño detallado e implementación de la red HAN

4.1. Introducción

En el Capítulo 3 se presentó el diseño conceptual del sistema HEMS, constituido por un conjunto de nodos asociados a los electrodomésticos, un controlador central (HEC) y una plataforma de comunicación para interconectarlos (HAN). La red HAN es entonces la plataforma de comunicación entre el HEC y los nodos del sistema. La misma debe ofrecer un mecanismo de comunicación bidireccional entre cada uno de los nodos y el HEC.

En el presente capítulo se aborda el diseño detallado e implementación de la red HAN, incluyendo detalles de elección de tecnologías, definición de protocolos y desarrollo de software de comunicación. Al finalizar el presente capítulo, el lector conocerá por completo el stack de comunicaciones que conforma la HAN, y la forma en que fueron implementadas las funcionalidades de comunicación a nivel de programación en los nodos.

4.2. Análisis y elección de tecnologías

Tal como fue mencionado en el Capítulo 3, las tasas de transferencia de datos entre los nodos y el controlador son bajas, por lo que no resulta necesario adoptar tecnologías que permitan un gran ancho de banda. Por otro lado, sí es necesario minimizar el consumo, dar soporte a topologías tipo mesh y trabajar con stacks de comunicación cuyas implementaciones requieran poca memoria RAM y ROM. Esto hace que las LR-WPANs (Low-Rate Wireless Personal Area Networks) sean consideradas como la tecnología más adecuada, dejando de lado otras muy conocidas como Wi-Fi o Bluetooth.

En los últimos años se han desarrollado múltiples protocolos y tecnologías de comunicación inalámbrica, y hay disponibles estudios sobre cómo estas tecnologías pueden ser utilizadas en aplicaciones de Smart Grid. El trabajo de Carles Gomez y Joseph Paradells [28] presenta diferentes opciones de tecnologías para el tipo

de aplicación que se propone implementar en el presente trabajo, comparando: Zigbee, 6LoWPAN, Z-Wave, INSTEON y Wavenis. Dentro de esta lista, Zigbee y 6LoWPAN, que utilizan el estándar IEEE 802.15.4 para la capa física, se destacan por ser las únicas con especificaciones de protocolo abiertas. Este último factor hizo que la decisión se acotara entre Zigbee y 6LoWPAN.

4.2.1. Discusión y elección de tecnologías y protocolos

Zigbee ha sido ampliamente utilizada durante los últimos años y cuenta con un buen soporte a nivel industrial y comercial (Peizhong Yi et al. [39]).

6LoWPAN por su parte es el resultado de un grupo de trabajo del IETF [1]. Se trata de una iniciativa para llevar el protocolo IP al mundo de los dispositivos de bajos recursos y sistemas embebidos utilizando IEEE 802.15.4 como protocolo de capa física y enlace. 6LoWPAN implementa una capa de adaptación entre IPv6 e IEEE 802.15.4 que permite la transmisión de paquetes IP sobre IEEE 802.15.4 mediante compresión de encabezado y fragmentación de paquetes. Sin esta capa de adaptación esto no sería posible debido a que el *throughput* ofrecido por IEEE 802.15.4 no es suficiente para transportar paquetes IP estándar.

Tras una comparación de tecnologías se observa que tanto 6LoWPAN como ZigBee satisfacen los requerimientos básicos de la aplicación. En particular:

- Soporte para topologías tipo mesh.
- Gran compatibilidad de hardware (solo se requiere interface de radio IEEE 802.15.4).

Sin embargo, 6LoWPAN cuenta con algunas ventajas importantes sobre ZigBee, tales como:

- Soporte nativo de IPv6, sin necesidad de conversión de protocolos para la conexión a Internet.
- Su especificación abierta (IETF RFC 6282) no tiene requerimientos de licencia.
- Existencia de implementaciones open-source sumamente livianas tanto en términos de procesamiento como memoria de carga, tales como uIP [23] o las ofrecidas por OpenWSN[2]. Estas implementaciones son compatibles con el sistema operativo Contiki OS[3].

En base a esto, IPv6 sobre 6LoWPAN fue el protocolo elegido como plataforma para el sistema de comunicación HAN.

[1] https://datatracker.ietf.org/wg/6lowpan/charter/, Mayo 2015.

[2] https://openwsn.atlassian.net/wiki/pages/viewpage.action?pageId=688187, Mayo 2015.

[3] En secciones posteriores se profundizará sobre este sistema operativo.

Sin embargo la sola elección de 6LoWPAN no resuelve el problema y los requerimientos de la HAN. Esta capa de adaptación debe ser complementada con un conjunto de protocolos y tecnologías adicionales para lograr un stack que contemple todos los requerimientos de la aplicación a implementar. En particular es necesario definir un protocolo de ruteo que trabaje sobre 6LoWPAN y posibilite la implementación de topologías mesh, y un protocolo a nivel de aplicación que simplifique el intercambio de mensajes a nivel de usuario.

En cuanto al protocolo de ruteo, *RPL (IPv6 Routing Protocol for Low-Power and Lossy Networks)* se presenta como la alternativa de ruteo y meshing más adoptada sobre 6LoWPAN. RPL es un estándar de la IETF diseñado especialmente para redes de sensores (*low power and lossy networks* o LLN) [46] que ya ha sido usado, probado y evaluado en el contexto de Smart Grid y en aplicaciones similares a las del presente trabajo [12, 15, 44]. Se trata de un protocolo flexible en el cual la topología de red es automáticamente armada por los nodos, utilizando distintos tipos de métricas como cantidad de saltos o indicadores de calidad de enlaces. Para la presente aplicación se elige RPL como protocolo de ruteo para lograr topologías mesh debido a que ha sido ampliamente probado y evaluado exitosamente.

Para utilizar cada uno de estos protocolos es necesario contar con una implementación abierta de los mismos o de lo contrario desarrollarla. La forma más sencilla de integrar implementaciones de stacks para microcontroladores sin tener que desarrollarlos es utilizar plataformas de software que ofrezcan servicios similares a los de un sistema operativo para sistemas embebidos. Algunas de estas plataformas son conocidas como *Real Time Operating System* (RTOS).

El conjunto de protocolos anteriormente descritos se encuentran implementados y soportados por la plataforma de software Contiki OS. Para lograr aprovechar una gran variedad de desarrollos ofrecidos por la comunidad Contiki, se optó por la utilización de esta plataforma de software para la implementación de los nodos. Contar con esta ventaja permite ahorrar tiempo de desarrollo en áreas accesorias del proyecto, ganando tiempo para poner foco en las áreas de interés fundamental.

Elegir Contiki como sistema operativo impone algunas restricciones en la elección del hardware y de los microcontroladores. Existe un universo limitado de motes[4] soportados por Contiki. Portar Contiki a un mote o a un microcontrolador no soportado es posible, pero requeriría esfuerzos que no van en línea con los objetivos del proyecto. Por tal motivo, a partir de la elección de Contiki, se realiza la elección del hardware a utilizar.

El conjunto de motes soportados por Contiki y que a su vez cumplan con los requerimientos de hardware de la HAN se reduce a aquellos que incorporen un transceptor de RF IEEE 802.15.4 y cuenten con recursos suficientes para utilizar los stacks de comunicación. Adicionalmente se valora la capacidad de simulación en Cooja [5], que permite simular una aplicación y realizar un debug completo de las

[4]En la jerga de las redes de sensores se utiliza comúnmente el término "mote" para hacer referencia a nodos sensores. Esto es, nodos que participan en una red de sensores y que poseen ciertas capacidades de procesamiento, almacenamiento y reporte de datos (http://en.wikipedia.org/wiki/Sensor_node, Mayo 2015).

[5]Esta herramienta se describe en el Apéndice A.

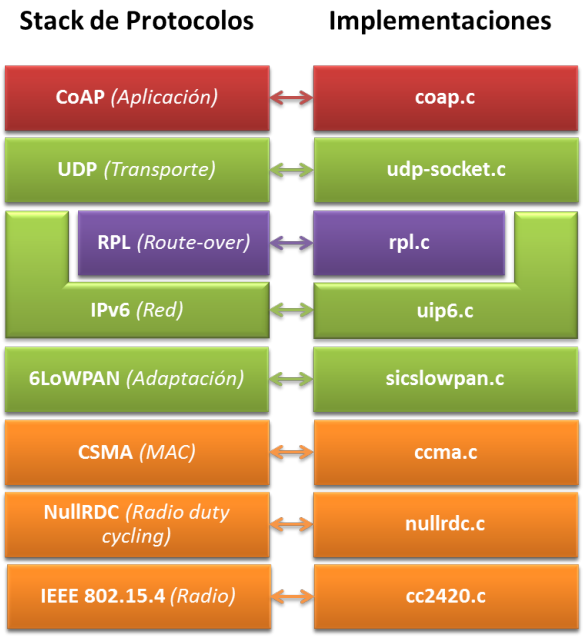


Figura 4.1: Stack de protocolos de la HAN.

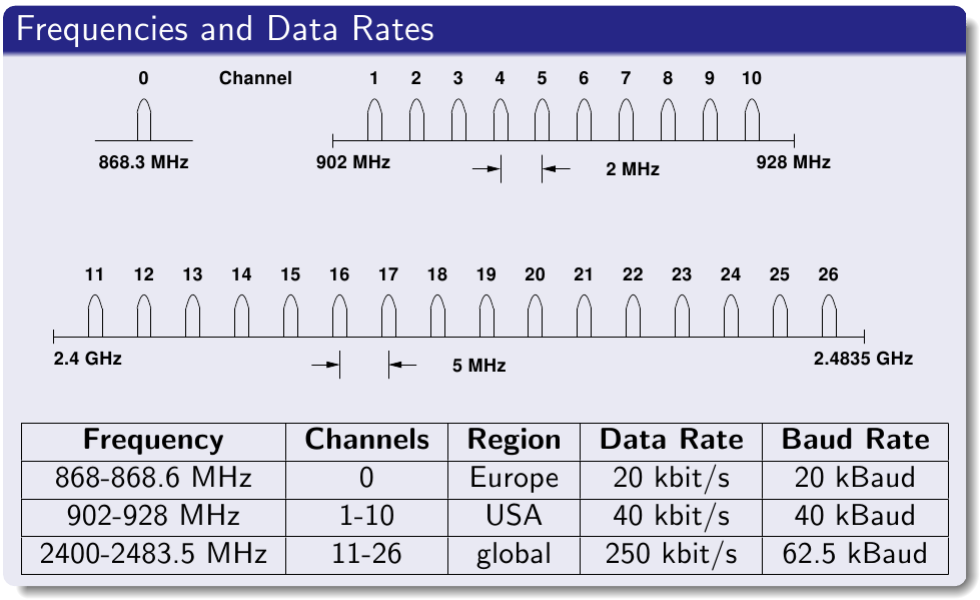
funcionalidades de comunicación. En función de lo anterior la elección se reduce a las siguientes plataformas: wismote, Z1, MICAz y TmoteSky.

En lo que respecta a la HAN, cualquiera de estas plataformas cumpliría en principio los requerimientos. La elección final se realiza en base a los requerimientos de las funcionalidades de medición y control y se presenta en el Capítulo 5.

En lo que refiere a protocolos de capa de aplicación, el soporte nativo de IPv6 de 6LoWPAN permite el uso de protocolos basados en la arquitectura REST (Representational State Transfer), utilizando un modelo cliente-servidor. En años recientes el grupo de trabajo CoRE (Constrained Restful Enviroments) de la IETF, ha desarrollado un protocolo denominado CoAP (Bormann et al. [19]) del cual existe una implementación para Contiki OS. El protocolo CoAP ofrece una arquitectura simple y flexible, ideal para este tipo de aplicaciones, por lo cual es elegido como protocolo de aplicación para la HAN.

4.2.2. Resumen de tecnologías a utilizar

A modo de resumen, se presenta en la Fig. 4.1 un esquema del stack de protocolos completo en base al cual se implementará la red de comunicaciones HAN.



Frequency	Channels	Region	Data Rate	Baud Rate
868-868.6 MHz	0	Europe	20 kbit/s	20 kBaud
902-928 MHz	1-10	USA	40 kbit/s	40 kBaud
2400-2483.5 MHz	11-26	global	250 kbit/s	62.5 kBaud

Figura 4.2: Datos característicos de IEEE 802.15.4 [41].

4.3. Descripción técnica de las tecnologías y stacks a utilizar

En esta sección se presenta una primera introducción a las características fundamentales de cada uno de los protocolos de comunicación seleccionados para la implementación de la HAN. Por simplicidad, no se realizará un estudio exhaustivo de todo el vasto conjunto de funcionalidades de cada uno de los protocolos utilizados, sino únicamente de aquellas que son significativas para el proyecto en cuestión, y cuya implementación se encuentra disponible en Contiki.

Teniendo en cuenta el stack de comunicaciones e implementaciones presentado anteriormente y resumido en la Fig. 4.1, se ofrece a continuación un repaso más detallado del funcionamiento de cada una de estas capas.

4.3.1. IEEE 802.15.4 (Capa física y acceso al medio)

Descripción de IEEE 802.15.4

IEEE 802.15.4 es un estándar de la IEEE que define las capas físicas y de control de acceso al medio [6]. Fue diseñado para aplicaciones HAN, con baja tasa de datos y a través de medio inalámbrico. En la Fig. 4.2 se muestran algunos datos característicos del protocolo.

IEEE 802.15.4 soporta diferentes bandas de frecuencia. Las bandas de frecuencia más bajas implican un menor ancho de banda, pero a su vez permiten mayores distancias.

[6] http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf, Mayo 2015.

Existen dos métodos de funcionamiento posibles: “Beacon mode” y “Beaconless mode”. El primero utiliza un método de división de tiempo (TDMA) a los efectos de economizar el envío de mensajes de radio mediante un sistema coordinado de turnos de emisión de paquetes por parte de los diferentes nodos, basándose en la utilización de un nodo especial llamado *Beacon* que funciona como coordinador de la red.

Existen dos tipos diferentes de nodos o interlocutores: *Full function device* (FFD) y *Reduced function device* (RFD). Los primeros implementan todo el protocolo. Son capaces de funcionar en cualquier topología y pueden ser configurados para funcionar como coordinadores de la red (nodo Beacon). Los segundos, por el contrario, solo cuentan con una implementación del protocolo más simple y liviana, y sus capacidades están limitadas a las de un nodo *leaf* u *hoja*. Teniendo en cuenta esto, vale la pena señalar además que se soportan dos topologías diferentes; *Peer-to-Peer* y *Estrella*. En cualquiera de las dos se necesita al menos un FFD para cumplir el rol de *PAN Coordinator*.

En lo que respecta al control de acceso al medio (MAC), el mismo se realiza utilizando el método CSMA/CA.

Implementación de IEEE 802.15.4 en Contiki

Contiki ofrece stacks compatibles con el estándar IEEE 802.15.4, que implementan la capa de acceso al medio para la radio. Esto hace que IEEE 802.15.4 pueda ser fácilmente utilizado sobre esta plataforma de software. Sin embargo, no todas las características definidas por el estándar se encuentran implementadas. En particular los stacks de Contiki no soportan el uso del modo “Beacon”, por lo cual no existen los *PAN Coordinators* y todos los nodos tienen igual jerarquía sobre IEEE 802.15.4 (no se diferencia entre RFD y FFD). En lugar de la coordinación mediante un Bacon, se utiliza una estrategia de RDC diferente, basada en ciclos asíncronos.

4.3.2. 6LoWPAN y su implementación en Contiki OS

Introducción

6LoWPAN es un protocolo estándar, resultado del trabajo de un grupo del IEFT [7]. Es uno de los protocolos de comunicación predilectos de Contiki. Para trabajar con 6LoWPAN, la comunidad de Contiki desarrolló un completo stack que implementa cada una de las capas requeridas. No obstante, dicha implementación no incluye todas las características definidas por 6LoWPAN, sino una selección de las mismas.

A continuación se realiza un repaso de 6LoWPAN, siempre teniendo como referencia la implementación de Contiki, que es en definitiva la que se se ha seleccionado para la implementación de la HAN.

[7] https://datatracker.ietf.org/wg/6lowpan/charter/, Mayo 2015.

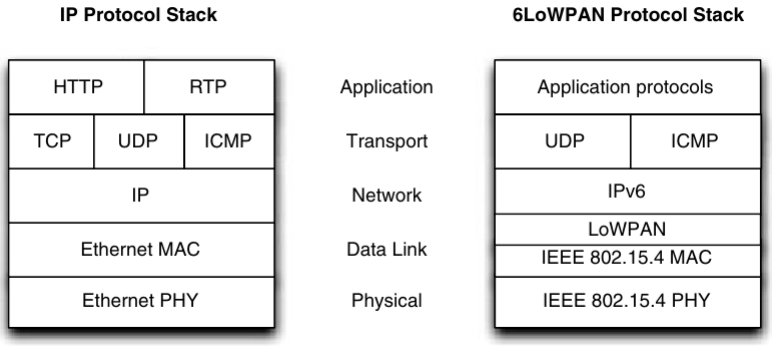


Figura 4.3: Stacks IP y 6LoWPAN [43].

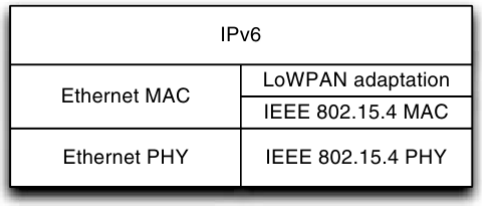


Figura 4.4: 6LoWPAN Border Router [43].

Estructura básica de 6LoWPAN

En el libro "6LoWPAN: The Wireless Embedded Internet" de Zach Shelby y Carsten Bormann [43] se presentan dos esquemas sumamente ilustrativos que ayudan a obtener un panorama general de la pila de protocolos en los distintos niveles del modelo OSI. En la Fig. 4.3 se observa una comparación entre un stack 6LoWPAN típico y el conocido stack TCP/IP, mientras en la Fig. 4.4 se muestra el esquema de lo que se conoce como *Border Router*, dispositivo capaz de rutear desde una red 6LoWPAN hacia una red estándar TCP/IP.

Cabe destacar que la estructura del stack de 6LoWPAN presentada no es única, ya que es posible utilizar 6LoWPAN con otros protocolos de capa física y acceso al medio aparte de IEEE 802.15.4 [43]. La capa de transporte tampoco es única, ya que se puede utilizar tanto UDP como TCP. Sin embargo el esquema presentado corresponde a la configuración típica utilizada en gran parte de las aplicaciones y es la estructura utilizada en el presente trabajo.

Capa de Adaptación 6LoWPAN e IPv6

Una de las ventajas más distintivas de 6LoWPAN es que ha sido diseñado desde sus inicios con internet como referencia. Esto es, la conectividad IP es una característica intrínseca del protocolo, y a diferencia de muchos otros no necesita de gateways para lograr acceso a una red IP. Tomando el paradigma IoT como uno de los orientadores para el diseño, 6LoWPAN fue concebido desde el comienzo

en base a IPv6, que en un futuro no lejano habrá reemplazado por completo al conocido IPv4.

Las direcciones en IPv6 tienen una longitud de 128 bits. En dispositivos de bajos recursos de procesamiento y memoria, resulta excesivo destinar esta cantidad de información a una simple dirección. Por otra parte, con esta longitud de direcciones los mensajes de radio se hacen muy extensos y el *overhead* crece afectando la eficiencia de las redes inalámbricas. Por este motivo, 6LoWPAN utiliza estrategias de compresión de direcciones, realizando un mapeo entre las direcciones de capa de enlace (MAC) y red (IP), ganando eficiencia en forma drástica.

La estrategia de compresión consiste básicamente en asumir que cierta parte de las direcciones (prefijo) resulta siempre conocida y no cambia dentro de una *Personal Area Network* (PAN), por lo cual no es necesario incluirlas en todos los mensajes. A la hora de enviar un mensaje hacia afuera de la PAN, por ejemplo a través de internet, se necesita un router capaz de reestructurar las direcciones volviendo al formato IPv6 estándar. Esta tarea queda a cargo de un dispositivo que recibe el nombre de *Border Router* (Fig. 4.4). El Border Router es capaz de comprender direcciones IPv6 en su versión completa, por lo cual en muchas aplicaciones se trata de un dispositivo más potente o al menos con menores requerimientos en términos de ahorro de energía o recursos. En la presente aplicación sin embargo, se utiliza el mismo modelo de mote para la implementación de nodos y border-router, tal como se verá en secciones posteriores.

Además de la compresión de direcciones, 6LoWPAN utiliza estrategias de fragmentación de mensajes para lograr una comunicación efectiva y eficiente. La fragmentación es una característica fundamental del protocolo, ya que los protocolos de capa de enlace utilizados (típicamente IEEE 802.15.4) muchas veces no soportan mensajes de la longitud necesaria para enviar los paquetes en una red convencional IPv6. Tanto las funciones de compresión de direcciones como las de fragmentación de mensajes son realizadas por la llamada *Capa de Adaptación 6LoWPAN*, que es la parte sustancial del protocolo. En la Fig. 4.3 dicha capa se muestra ubicada entre la capa de enlace y la capa de red.

Capa de transporte sobre 6LoWPAN e IPv6

Al igual que el protocolo 6LoWPAN e IPv6, la capa de transporte se encuentra implementada en Contiki. En este caso se ofrecen las opciones de UDP y TCP.

TCP es un protocolo orientado a conexión. Esto significa que brinda mayores garantías en términos de control de recepción de los mensajes con respecto a UDP, que es un protocolo *best effort*. El costo que se paga al utilizar TCP está relacionado con la complejidad del protocolo, los recursos que insume, y el *overhead* que agrega a los paquetes transmitidos.

En Contiki y sus stacks, tanto UDP como TCP están implementados y disponibles para su utilización. TCP se encuentra disponible en una versión reducida en relación a la versión convencional utilizada en redes de computadoras. En sistemas embebidos y microcontroladores participantes de redes PAN, sobre todo inalámbricas, no es común utilizar TCP [43]. La opción más elegida es UDP por sus escasas exigencias en términos de recursos.

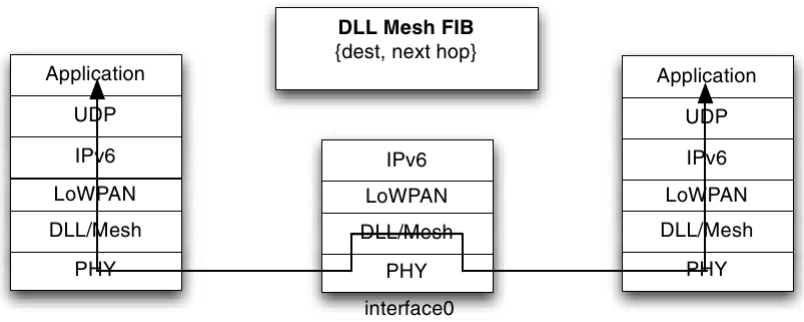


Figura 4.5: 6LoWPAN meshing: Mesh-under [43].

4.3.3. RPL: Mecanismo de ruteo y meshing

Tipos de ruteo

Al trabajar con dispositivos de recursos limitados y sobre redes HAN a través de medios inalámbricos tales como IEEE 802.15.4, resulta de particular importancia conocer las posibilidades de implementar arquitecturas de red del tipo mesh en las cuales todos los nodos, o al menos un conjunto de ellos, puedan ser capaces de actuar como repetidores de mensajes, con el objetivo de extender el alcance de la red más allá de la distancia máxima impuesta por el enlace de radio punto a punto.

Resulta oportuno identificar dos tipos diferentes de ruteo; por un lado existe el ruteo hacia afuera de la HAN, el cual es realizado por el Border Router. Por otro lado existe el ruteo intra-HAN, necesario para lograr arquitecturas de meshing mediante repetición de mensajes.

Este último tipo de ruteo puede realizarse de distintas formas e incluso a diferentes niveles del stack. Una opción sería efectuarlo a nivel IP. Otra, sería realizarlo a nivel MAC. En el contexto de 6LoWPAN existen dos alternativas diferentes para lograr mecanismos de meshing. Estas reciben los nombres de *Mesh-under* y *Route-over* respectivamente. En la Fig. 4.5 se observa un posible esquema de implementación del mecanismo mesh-under. El mecanismo Route-over se ilustra en la Fig. 4.6.

En el caso de Mesh-under los mecanismos de ruteo mesh pueden implementarse en la capa de adaptación 6LoWPAN, en la capa de enlace, o en en el límite entre ambas. Dentro de este paradigma existen diferentes protocolos que pueden ser tomados como ejemplo, de los cuales el más importante es ISA-100, que consiste en un protocolo mesh para comunicación basada en IEEE 802.15.4 especialmente orientado a aplicaciones industriales.

Los stacks de Contiki no implementan mecanismos de Mesh-under, sino que apuntan a la implementación de mecanismos mesh mediante estrategias de Route-over[8].

Los mecanismos de Route-over se implementan en la frontera entre la capa

[8] http://contiki.sourceforge.net/docs/2.6/a01794.html, Mayo 2015.

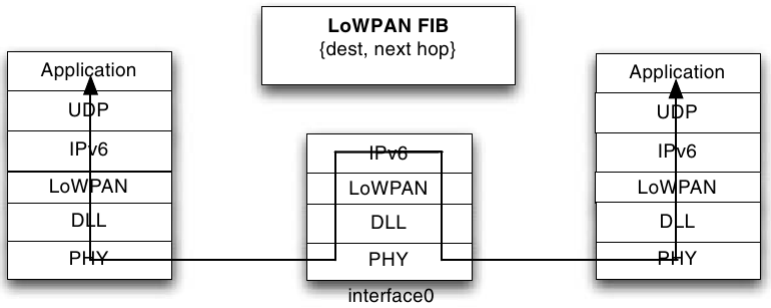


Figura 4.6: 6LoWPAN meshing: Route-over [43].

de red y la de transporte. Si bien existen diferentes protocolos o mecanismos de Route-over, Contiki y su stack implementan uno en particular que ha ganado gran aceptación por sus prestaciones. Se trata del protocolo RPL, ya introducido en la sección 4.2.

RPL y sus mecanismos

RPL fue definido por un grupo de trabajo de IETF denominado *Routing over Low-power and Lossy networks* (ROLL)[9]. Contiki implementa una versión de este protocolo que si bien no contiene todas las funcionalidades previstas en la definición, brinda los mecanismos y funciones principales con la ventaja adicional de ser una implementación liviana y eficiente. La implementación RPL de Contiki se denomina *ContikiRPL* [44].

Existen varias publicaciones de trabajos dedicados a evaluar el desempeño del protocolo sobre Contiki, que muestran resultados exitosos en diversos escenarios [12, 15, 44].

RPL es un protocolo de ruteo basado en vectores de distancia. Consiste en conectar los diferentes nodos utilizando una estructura de tipo árbol denominada *Destination Oriented Directed Acyclic Graph* (DODAG). En la Fig. 4.7 se muestra un ejemplo de este tipo de estructura.

El nodo ubicado en la parte superior del gráfico se denomina *DODAG Root*. Se dice que los datos dirigidos hacia el Root viajan "hacia arriba", mientras que los datos provenientes del Root van "hacia abajo".

La estructura jerárquica de tipo árbol se construye asignando diferentes *rangos* a cada nodo, siendo este rango una función de la distancia lógica desde el nodo hasta Root. El Root tiene rango unitario y el rango del resto de los nodos será mayor cuanto más alejados del Root se encuentren. Es importante aclarar que cuando se habla del rango como función de la distancia, se hace referencia a la distancia lógica, como concepto general vinculado al costo de transmisión y no como distancia física. Es decir, un nodo podría estar a unos pocos centímetros del Root, y sin embargo tener un rango muy alto, por ejemplo por estar prácticamente sin

[9] https://datatracker.ietf.org/wg/roll/documents/, Mayo 2015.

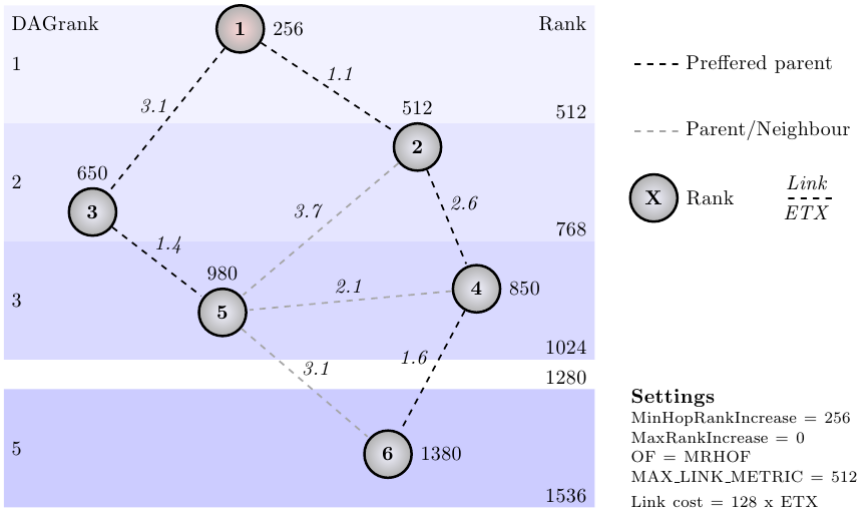


Figura 4.7: RPL: esquema de estructura DODAG [30].

batería y contar con una potencia de radio muy baja, o por existir un medio físico atenuante entre ambos interlocutores. De la misma forma, un nodo más alejado en términos de distancia física podría contar con un rango menor debido, por ejemplo, a su mayor potencia de radio o a una trayectoria de comunicación favorable.

La forma práctica en que la distancia lógica, y en consecuencia el rango de cada nodo se calculan, no es única sino que está dada por la *Objective function* (OF) utilizada, que es una opción del protocolo. Hoy en día existen dos OF fundamentales definidas en el marco de la IETF:

OF0 [RFC 6552]: Es una función sencilla basada exclusivamente en *hop-count*. Es decir, la distancia se mide en saltos de nodo a nodo. En este caso se priorizan rutas con menor cantidad de saltos, aunque estos sean de baja calidad, por sobre rutas con mayor cantidad de saltos de buena calidad [11].

MRHOF (Minimum Rank with Hysteresis Objective Function) [RFC 6719]: Se trata de una función más elaborada, que básicamente cuenta con tres opciones o reglas de distancia [10]:

- Hop-count: Similar a OF0.
- Latencia: Tiempo de respuesta de la trayectoria de comunicación.
- ETX (Expected transmissions): La variable de medida de distancia es la cantidad de transmisiones que se espera serán necesarias antes de realizar exitosamente el envío de un mensaje.

En base a la OF elegida cada nodo calcula la distancia que lo separa de los demás, y mediante un mecanismo de propagación que se explicará en párrafos posteriores, la HAN acaba por organizarse de acuerdo a una estructura análoga a un árbol genealógico, donde el Root es el primer ancestro y padre de sus descendientes

(el resto de los nodos). Los nodos más cercanos al Root serán sus descendientes directos, pero serán a su vez padres de otros nodos más alejados.

RPL soporta cuatro modos diferentes de funcionamiento:

1. Sin rutas hacia abajo almacenadas en los nodos: solo sirve cuando la comunicación es en un solo sentido, es decir, los nodos reportan información al Root.

2. Modo sin almacenamiento de rutas: se permite el tráfico hacia abajo pero los nodos no almacenan rutas, sino que el Root debe acompañar el mensaje con la especificación completa de la ruta a seguir para alcanzar el destino.

3. Modo con almacenamiento, sin multicast: se permite el tráfico hacia arriba y hacia abajo, y cada nodo debe almacenar las rutas unicast.

4. Modo con almacenamiento y multicast: idem anterior, pero con el agregado de que cada nodo debe almacenar no solo rutas unicast sino también multicast.

ContikiRPL implementa solo los modos 1, 3 y 4. El modo de funcionamiento se define en el software del Root, y cuando un nodo se une a un DODAG coordinado por dicho Root debe respetar y cumplir con su modo de funcionamiento. Si un nodo no cuenta con capacidades o recursos suficientes para ejecutar dicho modo, puede unirse en modo "hoja", en los extremos del DODAG.

Para introducir los mecanismos mediante los cuales se construyen los grafos y las rutas, es necesario conocer los tres tipos de mensaje fundamentales definidos por el protocolo:

- **DODAG Information Object (DIO)**: son usados para formar, mantener y descubrir un DODAG, y contienen información acerca de este que le permite a los nodos descubrir instancias y seleccionar a los padres favoritos.

- **DODAG Information Solicitation (DIS)**: son usados por cualquier nodo para solicitar información a nodos vecinos. Esto es, una petición de envío de DIO. Es disparado por un nodo durante el start-up o cuando no ha recibido un DIO luego de cierto tiempo.

- **DODAG Destination Advertisement Object (DAO)**: Son usados para propagar información de destino en dirección arriba, para completar las tablas de ruteo de nodos ancestros.

Continuando con la analogía del árbol genealógico, el fundamento del mecanismo mediante el cual se forma un DODAG consiste en la capacidad de cada nodo de evaluar "padres candidatos", elegir el favorito, adoptarlo como padre y reportarle la información que tiene del resto de la red. El proceso se da mediante un mecanismo de propagación comenzando por el Root y su entorno, y avanzando hacia abajo en la estructura.

El primer paso para la construcción de un DODAG se da cuando un nodo configurado como Root despierta y comienza a enviar DIOs en forma de broadcast. Los nodos más cercanos (en el alcance directo del Root) recibirán los mensajes del Root y los procesarán del mismo modo que procesarían cualquier otro DIO, con la particularidad de que por el rango declarado en dicho DIO, los nodos elegirán al Root como padre. Esta decisión será obvia, pues el Root habrá declarado rango 1 (no podría haber un padre más adecuado). Una vez que los nodos cercanos eligieron al Root como padre, se lo comunicarán mediante un DAO y comenzarán a enviar sus propios DIOs hacia la HAN. El proceso continúa del mismo modo en forma sucesiva, y a medida que los diferentes nodos van colectando mayor información (nuevas rutas) irán comunicándola a sus ancestros o superiores mediante mensajes DAO.

Una vez que el DODAG está armado cada nodo tendrá una tabla de rutas con la información de los nodos que están por debajo en la escala jerárquica. Luego, el mecanismo de ruteo es sencillo: cuando un paquete debe ser enviado "hacia arriba", el nodo lo envía a su padre favorito. Este hará lo propio con el suyo, y así sucesivamente hasta llegar al Root. En el caso del ruteo "hacia abajo", cuando un nodo recibe un paquete chequea si el destinatario se encuentra registrado en su tabla de ruteo (información de nodos de menor jerarquía). En caso de que así sea, reenviará el paquete por la ruta correspondiente. En caso de no tener información del destinatario, enviará el paquete "hacia arriba" a su padre favorito para que este resuelva mediante el mismo mecanismo.

4.3.4. Capa de Aplicación: CoAP

Introducción

Hasta ahora se ha realizado un repaso sobre los stacks de protocolos que permiten el intercambio de mensajes entre nodos a lo largo de una HAN, e incluso el acceso desde y hacia estos nodos por parte de interlocutores externos a través de Internet. Sin embargo, para que el intercambio de mensajes se convierta en intercambio de información significativa, se necesita recurrir a mecanismos o protocolos de más alto nivel, situados en las capas superiores. En este sentido existen diferentes alternativas.

La primera alternativa consiste en elaborar estos mecanismos de intercambio a nivel de programa de usuario. Esto es, hacer uso de las API, funciones y servicios brindados por capas inferiores, y programar rutinas que generen y gestionen el intercambio de información entre interlocutores de la red. Esta estrategia tiene la ventaja de que permitiría obtener eventualmente el mayor grado de eficiencia, al estar programada en forma específicamente dedicada al tipo de datos e información que se desea intercambiar. A su vez, permite al programador tener un control absoluto sobre las características de este intercambio.

Otra estrategia es la de utilizar mecanismos estándar de intercambio. Esto es, protocolos de capa de aplicación abiertos, no propietarios, que ofrezcan servicios de más alto nivel al programador de la aplicación final. En este sentido y en el contexto de IoT, existe una clara tendencia a utilizar protocolos abiertos basados

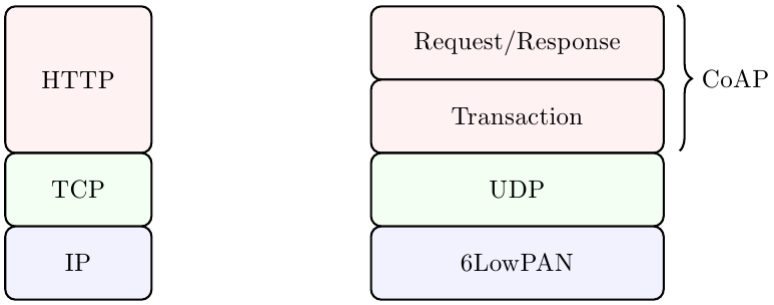


Figura 4.8: Stacks HTTP y CoAP [30].

en arquitecturas similares a las de HTTP. Esta línea tiene por un lado la ventaja de representar una opción estandarizada, lo que brinda mayor versatilidad y compatibilidad entre sistemas. Por otro lado permite al programador ahorrar esfuerzos de desarrollo, al contar con bibliotecas ya implementadas, testeadas y depuradas. En el contexto de este proyecto se optó por este último camino, esto es, el de utilizar protocolos estándar y abiertos, y dentro de este marco se halló que CoAP representa una opción sumamente interesante.

CoAP y su funcionamiento

CoAP basa su funcionamiento en la arquitectura REST, en la cual la información se ofrece en forma de *recursos* disponibles en un servidor, y las aplicaciones intercambian datos mediante el uso de *Métodos* como PUT, POST, GET y DELETE, los cuales son ampliamente conocidos en el universo HTTP. Para cada recurso hay típicamente un servidor y uno o varios clientes que se suscriben a él. Esto permite que cada aplicación reciba sólo información de los recursos que le interesan.

La falta de soporte de mensajes multicast y el gran overhead impuesto por HTTP, hacen que este protocolo en su versión estándar no sea apto para su utilización en sistemas de recursos restringidos [30]. Con el objetivo de mejorar estos aspectos y adaptar HTTP para su uso en dicho tipo de sistemas, el grupo de trabajo CoRE de IETF desarrolló el protocolo llamado “Constrained Application Protocol”, mejor conocido como CoAP (Bormann et al. [19]).

CoAP implementa un subconjunto de las funcionalidades disponibles en HTTP y a su vez agrega algunas características especiales que lo hacen ideal para el uso en sistemas de recursos limitados.

Típicamente CoAP funciona sobre UDP. Sin embargo, el protocolo implementa funciones de retransmisión para suplir aquellas características de control de mensajes que se pierden al prescindir de TCP. Esto se representa en la Fig. 4.8, que incluye una comparación con la estructura típica de HTTP.

CoAP presenta algunas particularidades que lo distinguen de HTTP y lo hacen apto para el funcionamiento sobre sistemas embebidos o de recursos restringidos. Algunas de estas particularidades tienen que ver con los tipos de recursos disponi-

bles. Existen en CoAP diferentes tipos de recursos, con distintas características. En particular existen recursos "observables" y "no observables". Los primeros admiten una "suscripción" por parte de un cliente, que recibirá la información representada por dicho recurso periódicamente. Los recursos no observables en cambio, no actualizan su contenido automáticamente, sino bajo solicitud.

CoAP brinda también las funciones *Block* y *Discovery*. La primera permite que la información de un determinado recurso sea enviada en forma fragmentada en el caso de ser muy extensa. La segunda, permite a los diferentes nodos de una red reconocerse y tomar conocimiento de los recursos ofrecidos por los demás. Además, implementa entre otros los métodos GET y PUT, que permiten consultar o escribir sobre un recurso en el momento en que son ejecutadas.

Para facilitar las tareas de debug e interpretación de mensajes por parte del usuario, se cuenta con una herramienta sumamente útil denominada COPPER. COPPER es un plugin desarrollado para el navegador Firefox, que implementa el protocolo CoAP e incorpora una interfaz web gráfica a través de la cual el usuario puede interactuar con clientes y servidores CoAP. Esta herramienta se describe con mayor detalle en el Apéndice A.

4.4. Diseño y configuración de parámetros de comunicación

En la presente sección se recapitulan los conceptos expuestos en la sección anterior, pero siempre poniendo foco en los aspectos prácticos del diseño de la solución. Se resumen todas las características, plataformas, protocolos y tecnologías elegidas para el sistema y luego se especifican los detalles de diseño en cuanto a topología, parámetros, características y funcionalidades utilizadas, etc.

En la Fig. 4.9 se muestra la arquitectura final del diseño de la HAN. Tanto los nodos como el Border Router se encuentran implementados sobre *motes* cuyas características de hardware serán abordadas en el capítulo siguiente. El software de cada uno de estos nodos y el Border Router está programado sobre Contiki OS, y las funcionalidades de comunicación se alcanzan gracias a la utilización de los stacks disponibles antes mencionados.

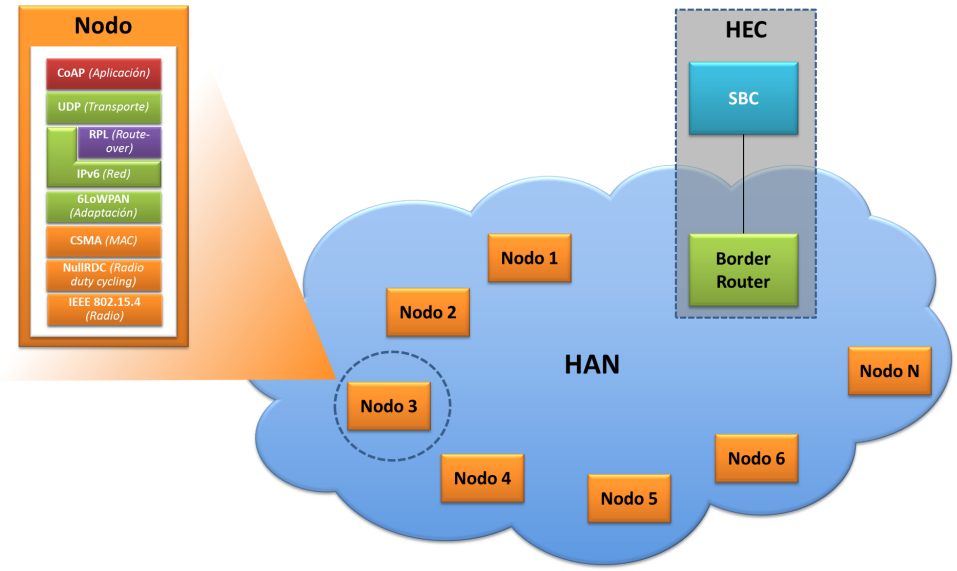


Figura 4.9: Arquitectura de la HAN.

4.4.1. Resumen de parámetros y configuraciones por protocolo

A continuación se repasan los protocolos que componen el stack de comunicación de la HAN, y para cada uno de ellos se establecen parámetros, opciones o configuraciones específicas.

IEEE 802.15.4

- **Banda de frecuencia**: Dado que en esta aplicación las distancias no son un requerimiento exigente, y teniendo en cuenta que existe una mayor oferta de hardware para para la banda 2.4GHz que para las bandas sub-GHz, se opta por trabajar en la banda de 2.4GHz.
- **RDC**: La funcionalidad RDC es opcional, y no es utilizada en el presente trabajo por insumir una considerable cantidad de recursos de memoria RAM y ROM del microcontrolador de los nodos. Para lograr un mejor aprovechamiento de estos recursos, se opta por eliminar el RDC, asumiendo el costo consecuente de un mayor consumo de energía, que en el caso de la aplicación no resulta crítico. Para evitar el uso de RDC, se debe utilizar el driver *nullrdc* en Contiki.

6LoWPAN - IP - UDP

- **Capa de transporte**: Se propone utilizar UDP y realizar el control de transporte de mensajes en capa de aplicación (a través de CoAP).
- **Border Router**: Se propone utilizar un Border Router capaz de brindar acceso a la HAN desde el HEC. Dado que el HEC deberá a su vez estar co-

nectado a internet (típicamente conexión LAN a módem ADSL) se propone utilizar conexión serial al Border Router utilizando un puerto LAN virtual. La forma más sencilla de implementar este Border Router es utilizando un mote del mismo modelo usado para la implementación de los nodos, y aprovechar su puerto USB para la conexión. Se brindarán mayores detalles sobre este punto en capítulos posteriores.

RPL

ContikiRPL implementa MRHOF, y su variable de evaluación por defecto es ETX. En la bibliografía de evaluación mencionada se obtienen resultados exitosos con esta configuración y por tal motivo es elegida como base para el diseño de la HAN.

CoAP

Se propone la utilización de dos tipos de recurso diferentes que serán ofrecidos por cada nodo. Uno será utilizado para el reporte de las medidas hacia el HEC, a través del Border Router, y el otro servirá para que el HEC envíe los comandos hacia cada nodo:

- res-mediciones-SG: Recurso *observable* y no escribible (soporta método GET).
- res-comandos-SG: Recurso escribible (soporta métodos GET, PUT y POST).

Configuración general de stacks en Contiki

En la Tabla 4.1 se resumen los parámetros básicos de configuración de los diferentes protocolos del stack a utilizar.

4.5. Implementación y ensayos preliminares HAN

En la presente sección se presenta la implementación de la red HAN en base a las funcionalidades definidas y el diseño propuesto para el stack. Para facilitar la lectura y evitar una extensión excesiva del capítulo, se brindan aquí los lineamientos más importantes del desarrollo e implementación, sin entrar en los detalles más específicos. En este sentido, no se presenta aquí una descripción detallada de directorios, código fuente programado, formato de representación de variables y mensajes, parámetros específicos de Contiki, etc. No obstante, si el lector tuviera interés en conocer este tipo de detalles, los mismos pueden ser consultados en el Apéndice A ("Programación Contiki").

A los efectos de explicar los detalles de implementación, se hace referencia a los motes Zolertia Z1, que resultan ser los motes elegidos para la implementación de los nodos y el Border Router. La discusión y elección de esta plataforma de hardware se explica en el Capítulo 5.

		Nodos	**Border Router**
CoAP	Recurso res-mediciones-SG	Servidor	NO (HEC Cliente)
	Recurso res-comando-SG	Servidor	NO (HEC Cliente)
Transporte	TCP	No	No
	UDP	Si	Si
RPL	Root	No	Si
	Objective function	MRHOF (ETX)	MRHOF (ETX)
	Modo de funcionamiento	4 (rutea)	4 (rutea)
IPv6			
6LoWPAN			
IEEE 802.15.4	Frecuencia	2.4GHz	2.4GHz
	Tipo de dispositivo	FFD	FFD
	Acceso al medio	CSMA	CSMA
Hardware	Modelo Mote	Zolertia Z1	Zolertia Z1

Tabla 4.1: Configuración HAN.

Se hará referencia en la presente sección a algunas herramientas de programación y simulación en el contexto de sistemas embebidos y redes de sensores utilizando Contiki. Entre ellas se mencionan *Instant Contiki*, *Cooja* y *COPPER*. No es necesario conocer estas herramientas para comprender los contenidos aquí brindados, pero si el lector quisiera leer acerca de las mismas, algunas detalles al respecto se brindan en el Apéndice A.

4.5.1. Diseños de referencia y punto de partida

Una ventaja importante de trabajar con Contiki es que además de contar con stacks de comunicación ya desarrollados y testeados, se dispone de un interesante conjunto de códigos de ejemplo. Se trata de aplicaciones típicas que cumplen funcionalidades diversas y sirven como punto de partida o referencia para el programador. Tanto el software de comunicación de los nodos como el del Border Router fueron desarrollados en base a los siguientes ejemplos:

- *RPL Border Router (rpl-border-router.c)*: este ejemplo sirvió como base para el desarrollo del Border Router. Brinda una aplicación que detecta automáticamente los vecinos RPL y forma una estructura DODAG (red mesh). Además de cumplir con las funciones de configuración automática de la red, brinda acceso a la HAN a través de un puerto serial. Es decir que conectando un PC mediante puerto USB al mote que corre este programa, se tiene acceso a todos los nodos gracias a sus funciones de gateway. El RPL Border

Router es además sumamente útil a la hora de realizar un debug a nivel RPL, ya que ofrece al usuario una sencilla página web donde refleja todas sus rutas y vecinos detectados.

- *Er Rest Example*: este ejemplo sirvió como base para el software de comunicación de los nodos a través de recursos CoAP. Ofrece ejemplos de código fuente tanto para un servidor REST (*er-example-server.c*) como para un cliente. Además, brinda una serie de recursos típicos que pueden resultar útiles para una gran variedad de aplicaciones. Este ejemplo también ofrece una simulación en Cooja[10] previamente configurada, donde se implementa una estructura cliente-servidor.

Los programas desarrollados a partir de los estos ejemplos constituyen el motor de la plataforma de comunicación. Sin embargo, el contenido de los mensajes a nivel de aplicación está dado por los diferentes *recursos CoAP* utilizados. Como parte del ejemplo *Er Rest Example* se ofrece un conjunto de unos 35 recursos, de los cuales se tomaron como referencia dos, que fueron adaptados para obtener los dos recursos diseñados para el sistema, a saber, *res-mediciones-SG* y *res-comando-SG*:

- *res-push.c (obs)*: se trata de un recurso observable, que permite realizar una tarea periódica y enviar la actualización de la información a los clientes suscritos cada un período de tiempo configurable.

- *res-leds.c (put)*: se trata de un ejemplo típico para el uso de un recurso a través de la función PUT. En este caso particular se activan los leds del mote de acuerdo al valor de un parámetro incluido en el mensaje de consigna enviado desde el cliente.

Por más detalles acerca de estos ejemplo, el lector puede consultar el Apéndice A.3.1.

4.5.2. Programación a partir de los ejemplos disponibles

Tomando como referencia los ejemplos mencionados, se programó el software tanto para los nodos como para el Border Router. En el Apéndice A.2.1 se ofrecen detalles acerca de la estructura de los programas desarrollados, incluyendo códigos fuente y directorios configurados.

En el caso del Border Router se utilizó el ejemplo *RPL Border Router* prácticamente sin cambios a nivel de código. Sin embargo se realizaron varios cambios a nivel de parámetros de funcionamiento de forma de configurar el stack de comunicaciones utilizado de acuerdo al stack propuesto en este trabajo.

El software de comunicación de los nodos, en particular las funcionalidades vinculadas al protocolo CoAP, fue programado utilizando como referencia el ejemplo *Er Rest Example* y particularmente el programa *er-example-server.c*. En este caso no solo se realizaron cambios a nivel de configuración sino que el código también

[10] Ver Apéndice "Programación Contiki" A.

fue adaptado a la aplicación del nodo. Finalmente, los recursos *res-mediciones-SG* y *res-comando-SG* propuestos en el diseño del sistema de comunicación entre los nodos y el HEC fueron programados tomando como inspiración los ejemplos *res-push.c* y *res-leds.c* anteriormente explicados.

Descripción de los recursos CoAP desarrollados

A continuación se brinda una breve descripción de los recursos CoAP desarrollados para los nodos. Por detalles acerca de formato de representación y codificación de variables consultar el Apéndice A.3.2. En el mismo apéndice, en la Sección A.3.5, se brinda una descripción detallada del código implementado, explicado la forma en las que se ejecutan las diferentes funciones.

Recurso res-mediciones-SG.c

Se trata de un recurso *observable*. Su contenido es actualizado y enviado automáticamente a todos los clientes suscritos cada un tiempo configurable, que ha sido seteado en 5 segundos. Además del modo OBSERVE, también acepta la directiva GET en caso de que el cliente quiera consultarlo por iniciativa propia en un momento dado.

Este recurso ofrece como respuesta un conjunto de 6 valores, representando las siguientes variables de interés:

- Tensión RMS: medida de la tensión reportada por el nodo.
- Corriente RMS: medida de la corriente reportada por el nodo.
- Potencia Activa: medida de la potencia activa reportada por el nodo.
- Potencia Reactiva: medida de la potencia reactiva reportada por el nodo.
- Estado de relés: reporte del estado actual de la carga (activado/desactivado) enviado por los nodos al HEC.
- Tensión de referencia de señales: variable auxiliar del sistema que monitorea la tensión continua de referencia utilizada durante el acondicionamiento de las señales. Solo tiene utilidad a nivel de monitoreo de estado de la electrónica.

Debe tenerse en cuenta que a esta altura del documento las dos últimas variables reportadas, y particularmente la última (tensión de referencia), pueden tener poco sentido para el lector. Sin embargo su utilidad será aclarada en detalle en el Capítulo 5.

Recurso res-comando-SG.c

Se trata de un recurso que permite el uso de los métodos PUT y POST. Esto significa que el cliente puede escribir los valores de este recurso. El objetivo de este recurso es permitir al HEC escribir valores de comando para la activación o desactivación del electrodoméstico asociado al nodo. Cuando el HEC escribe un

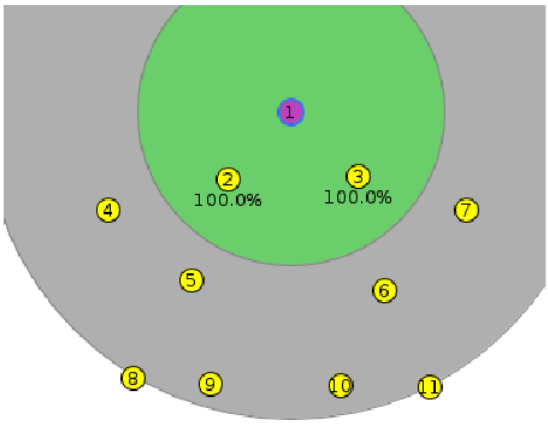


Figura 4.10: Simulación RPL en Cooja.

nuevo valor en este recurso, el programa se interrumpe y se llama inmediatamente al proceso encargado del comando de los relés, activándolos o desactivándolos de acuerdo al valor del parámetro recibido.

4.5.3. Simulaciones en Cooja y ensayos preliminares

Previo a probar el programa directamente en los motes, se realizaron algunas pruebas simuladas utilizando la herramienta de simulación "Cooja". Se construyó una simulación armando una arquitectura formada por un border-router y diez nodos, utilizando los programas descritos en la sección anterior. La arquitectura se armó sobre motes Z1 simulados. La distribución de los diferentes nodos puede verse en la Fig. 4.10.

El mote violeta identificado con "1" representa al Border Router y los restantes en amarillo corresponden a los nodos. El círculo central verde indica el área de cobertura de radio del border router y puede tomarse como referencia para tener una noción del área de cobertura de radio de los demás nodos. Como puede verse, la distribución de nodos en el espacio no se realizó en forma aleatoria, sino que se siguió un patrón geométrico claro. Si bien esto no es lo que sucederá en la realidad, el contar con una geometría sencilla y clara facilita el debug a nivel de RPL, ya que hace que los algoritmos de elección de rutas y asignación de rangos sean mucho más predecibles y fáciles de interpretar.

En base a simulaciones se concluyó que no era posible utilizar el stack completo con las configuraciones originales de Contiki, ya que no se contaba con recursos suficientes de ROM y RAM en el mote. Esto llevaba en algunos casos a errores de compilación y en algunas oportunidades provocaba comportamientos erráticos a nivel de RPL al no poder alojar correctamente en RAM la información acerca de rutas y vecinos. Estos comportamientos se observaron también al probar la red con motes reales.

Con el fin de economizar en recursos de ROM se eliminó la funcionalidad RDC

de la radio, pasando a utilizar el driver de radio *NullRDC* en lugar de *ContikiRDC*. Esto no representa mayores problemas, ya que no se pierde nada desde el punto de vista funcional. El cambio sólo tiene impacto en el consumo de energía del mote, que se ve incrementado. Esta situación no es en absoluto crítica en esta aplicación dado que los motes disponen de energía permanente tomada desde la red como se explica en el Capítulo 5. Este detalle podría mejorarse con facilidad, sencillamente utilizando un mote con mayor capacidad de memoria ROM.

Para economizar en uso de RAM se modificaron algunos parámetros de configuración vinculados a la cantidad de vecinos admitidos y cantidad de rutas RPL almacenadas por cada nodo. Se observó que reduciendo los buffers de almacenamiento utilizados por cada stack se logra un importante ahorro en memoria RAM, que resolvió los problemas descritos en párrafos anteriores. Los parámetros modificados a nivel de Contiki con sus respectivos valores originales y finales pueden consultarse en el Apéndice A.3.3.

Para evaluar la HAN en la totalidad de su stack se utilizó la herramienta COPPER, interactuando con los nodos simulados a través de CoAP. Mediante la mencionada herramienta se constató que todos los nodos ofrecen las medidas en el formato previsto y también aceptan los valores de comando para los relés.

Una vez probada la HAN a nivel de simulación, se procedió a cargar los programas en motes reales para evaluar el verdadero desempeño de la red. Para la prueba se dispuso de un total de 6 motes Zolertia Z1, configurando un Border Router y 5 nodos. Se realizaron pruebas con distintas ubicaciones de los nodos, conformando diferentes topologías a lo largo de distintas habitaciones de un apartamento. Se observó un comportamiento correcto confirmando los resultados obtenidos a nivel de simulaciones. En el Capítulo 8 se presentan ensayos más rigurosos y con resultados más precisos en lo que respecta al desempeño y latencia de los mensajes de la red HAN funcionado en forma integrada con el resto del sistema HEMS en su versión final.

En el Apéndice A.3.4 se puede consultar el resumen de los parámetros básicos de funcionamiento del stack a nivel de Contiki.

4.6. Síntesis del capítulo

Se presentó una breve recapitulación de los requerimientos y características básicas de la HAN, que surgieron como resultado de la discusión expuesta en el Capítulo 3.

En base a los requerimientos y características previamente definidos, se realizó un análisis de tecnologías disponibles. Se evaluaron las ventajas y desventajas de cada una, y se explicó la elección de tecnologías que servirán como base para el diseño. Se eligió utilizar IPv6/6LoWPAN sobre IEEE 802.15.4 (comunicación inalámbrica) como base del stack de comunicación, y CoAP como protocolo de aplicación.

Una vez elegidas las tecnologías y protocolos, se presentó una explicación introductoria del funcionamiento de cada uno de ellos, manteniendo el foco en los aspectos prácticos y en aquellas funcionalidades que están directamente vinculadas con la aplicación en cuestión.

Finalmente se propuso un diseño detallado de la HAN, especificando modos de funcionamiento, opciones o parámetros para cada uno de los protocolos.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 5

Diseño detallado e implementación de los Nodos

5.1. Introducción

Tal como se mencionó en el Capítulo 3, los nodos son los encargados de realizar mediciones de las variables eléctricas de consumo de los electrodomésticos y al mismo tiempo de comandar el encendido y apagado de los mismos en base a las instrucciones recibidas de parte del HEC. En el presente capítulo se introducen las plataformas y tecnologías elegidas como base para la implementación, y se abordan los detalles de diseño de hardware, adquisición y procesamiento de señales, métodos de medición empleados, y comando ON/OFF de las cargas. Luego se presenta la implementación de los circuitos y el software de procesamiento de señales y comando embebido en el MCU de los nodos.

5.2. Elección de Hardware y breve descripción

En esta sección se da cuenta de la selección de hardware realizada para la implementación del sistema, de acuerdo a lo detallado en el capítulo anterior.

5.2.1. Módulo de procesamiento y comunicación

De acuerdo a las especificaciones de la HAN, el universo de soluciones de hardware para la comunicación son las plataformas: wismote, Z1, MICAz y TmoteSky. De ellas se destaca la plataforma Z1 de Zolertia que es compatible con un gran número de sensores y actuadores de la marca Phidgets. Cuenta además con un microprocesador MSP430F2617 con 92KB de ROM y 8KB de RAM. Esta memoria se entiende suficiente para realizar un pre-procesamiento de las medidas de forma de disminuir el volumen de datos intercambiado, mejorando por consiguiente la performance de la red.

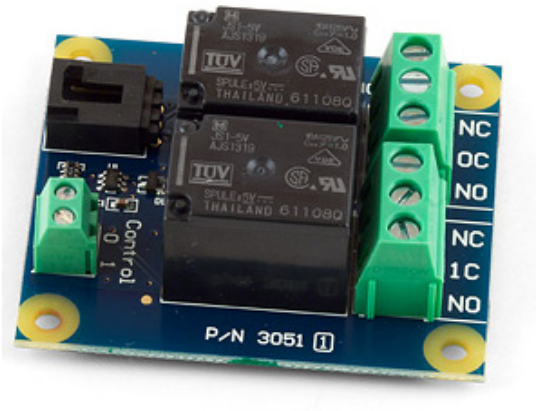
Los motes Z1 son módulos diseñados para facilitar el desarrollo de aplicaciones de *Wireless Sensor Network* (WSN). En la sección 5.3 se explica el diseño completo

de los nodos en base a las características explicadas en la descripción detallada del dispositivo que se encuentra en el Apéndice B.1.1.

5.2.2. Módulo de comando

En función de la elección de los motes Z1, se realizó una búsqueda de relés de la marca Phidgets compatibles con esta plataforma y que tuvieran capacidad para comandar cargas de al menos 10A. Luego de este relevamiento se optó por adquirir el dispositivo *Dual Relay Board (modelo: 3051_1)* que cuenta con dos relés electromecánicos que soportan hasta 20 contactos por minuto, con valores máximos de voltaje y corriente de carga de hasta 240VAC y 10A respectivamente.

El "Phidget 3051_1 Dual Relay Board" (Fig. 5.1a) consiste en una placa compuesta por dos relés electromecánicos de idénticas características denominados R0 y R1 y cuyas bobinas son comandadas por dos entradas digitales disponibles mediante borneras. La placa cuenta con una etapa transistorizada que separa las entradas lógicas del circuito de las bobinas funcionando como "drive" del relé y asegurando una alta impedancia a la entrada. Esto hace que las entradas lógicas sean compatibles con salidas lógicas estándar, particularmente con las del Z1 y su MSP430. En la Fig. 5.1b se incluyen algunas especificaciones básicas de esta placa de relés.



(a) 3051 Phidget Dual Relay Board.

Electrical Properties

Dielectric Strength	1.5 kV AC
Contact Resistance Max	100 mΩ
Load Voltage Max (DC)	100 V DC
Load Voltage Max (AC)	240 V AC
Load Current Min	100 mA
Load Current Max (DC)	5 A
Load Current Max (AC)	10 A
Turn-on Time Max	10 ms
Turn-off Time Max	10 ms

Board

Current Consumption Min	14 mA
Current Consumption Max	180 mA
Supply Voltage Min	3.3 V DC
Supply Voltage Max	12 V DC

(b) Datos técnicos.

Figura 5.1

Las especificaciones más importantes a tener en cuenta a la hora del diseño tienen que ver con la alimentación de la placa. La misma debe ser alimentada con una tensión entre 3.3VDC y 12 VDC, y la fuente deberá ser capaz de proporcionar 180mA en el peor caso.

5.2.3. Módulo de medición

Para la medición de tensión y corriente, dentro del universo de sensores Phidgets no existe una solución capaz de relevar ambas medidas simultáneamente, y

no se logró identificar ningún otro conjunto de sensores listos para conectar y utilizar que cumplan con todos requerimientos en cuanto a precio y características. Intentando acotar un eventual desarrollo de hardware propio, se identificó el dispositivo *KillAWatt P4400* de la empresa P3 International, un medidor de energía ampliamente comercializado, de bajo costo y con una exactitud declarada de 0,2 %. Este producto calcula Vrms, Irms, potencia activa-reactiva y factor de potencia. Como ventaja adicional, este dispositivo es frecuentemente modificado por entusiastas en distintos proyectos de domótica, por lo que se dispone de varios circuitos esquemáticos que facilitan la comprensión de su funcionamiento. La intención es poder utilizar parte del dispositivo y adaptarlo en función de los requerimientos del sistema, particularmente al ADC del MSP430F2617 y al voltaje de la red (el KillAWatt está especificado para redes de 110VAC).

Como se detalla en la Fig. 5.2, el *KillAWatt P4400*[1] se compone básicamente de cuatro bloques: un bloque de potencia (representado con "1" en el esquema), un bloque de medición y acondicionamiento ("16a", "11" y "16" según esquema), un bloque de muestreo y procesamiento ("12","13","14" y "17" en el esquema) y un bloque de alimentación ("18" en el esquema).

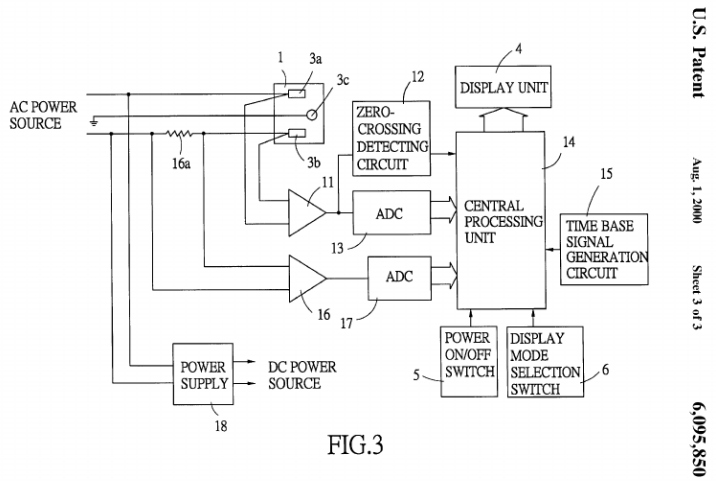


Figura 5.2: Esquema descriptivo de la patente US6095850.

Dado que la patente no presenta información detallada del diseño, para completar la identificación de los componentes se consultó material de terceros[2] [3] y se llevó a cabo una inspección visual del producto. Con toda la información recabada

[1]Protegida en USA bajo la patente US6095850, presentada en 1998. Como las patentes tienen aplicación territorial, y al no tener registro en Uruguay, no hay ninguna limitación en cuanto a la modificación del diseño así como su eventual producción en el territorio nacional.

[2]https://reindeerflotilla.wordpress.com/2011/08/23/the-kill-a-watt-not-all-versions-are-created-equal/

[3]http://www.mikesmicromania.com/2013/03/kill-watt-circuit-analysis.html

se logró obtener el esquemático general del dispositivo que se muestra en las Figs. 5.3 y 5.4.

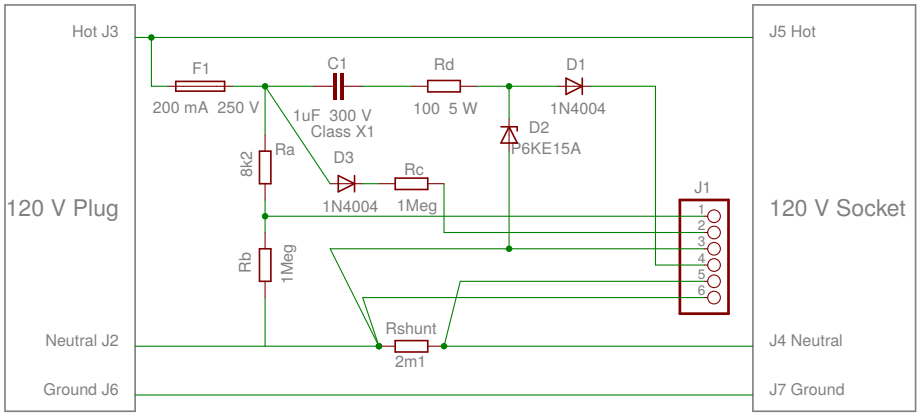


Figura 5.3: Placa de potencia.

La placa de potencia es la primer etapa del sistema y se ubica entre la carga a medir, en adelante Z, y la red eléctrica. Se observa en primer lugar que el circuito cuenta con un fusible de protección de forma de evitar picos de corriente en la placa (F1).

La medida de corriente se implementa mediante el shunt R_{shunt}, ubicado entre uno de los terminales de la carga y el neutro de la red. Las señales de corriente están dadas por:

$$\boxed{V_{i1} = \left(\frac{R_{20}}{R_{29}}\right) R_{shunt}.I_Z + V_{REF} = G_{i1}.I_Z + V_{REF}} \tag{5.1}$$

$$\boxed{V_{i2} = \left(\frac{R_{20}R_{15}}{R_{29}R_9}\right) R_{shunt}.I_Z + V_{REF} = G_{i2}.I_Z + V_{REF}} \tag{5.2}$$

La señal de voltaje está dada por:

$$\boxed{V_v = \left[\frac{(1 + R_4/R_{26})R_bR_4}{(R_a + R_b)(R_{26} + R_b + R_a/\!/R_b)}\right] V_{red} + V_{REF}} \tag{5.3}$$

Un detalle a destacar es que el voltaje de referencia del circuito se toma del terminal Neutro del tomacorrientes (diseño americano). En casos como el de la red uruguaya, donde el Neutro no está puesto a tierra, las señales están montadas sobre altos valores de tensión (típicamente valor de fase). En otras palabras, la referencia de la electrónica tiene una diferencia de potencial importante con respecto a la tierra de la red. Esto requiere extremar las precauciones al manipular el circuito.

La alimentación del bloque de acondicionamiento y procesamiento proviene de la propia red eléctrica, mediante un rectificador con filtro que luego es estabilizado por el diodo D1 de 9,1V. La alimentación de los amplificadores operacionales se

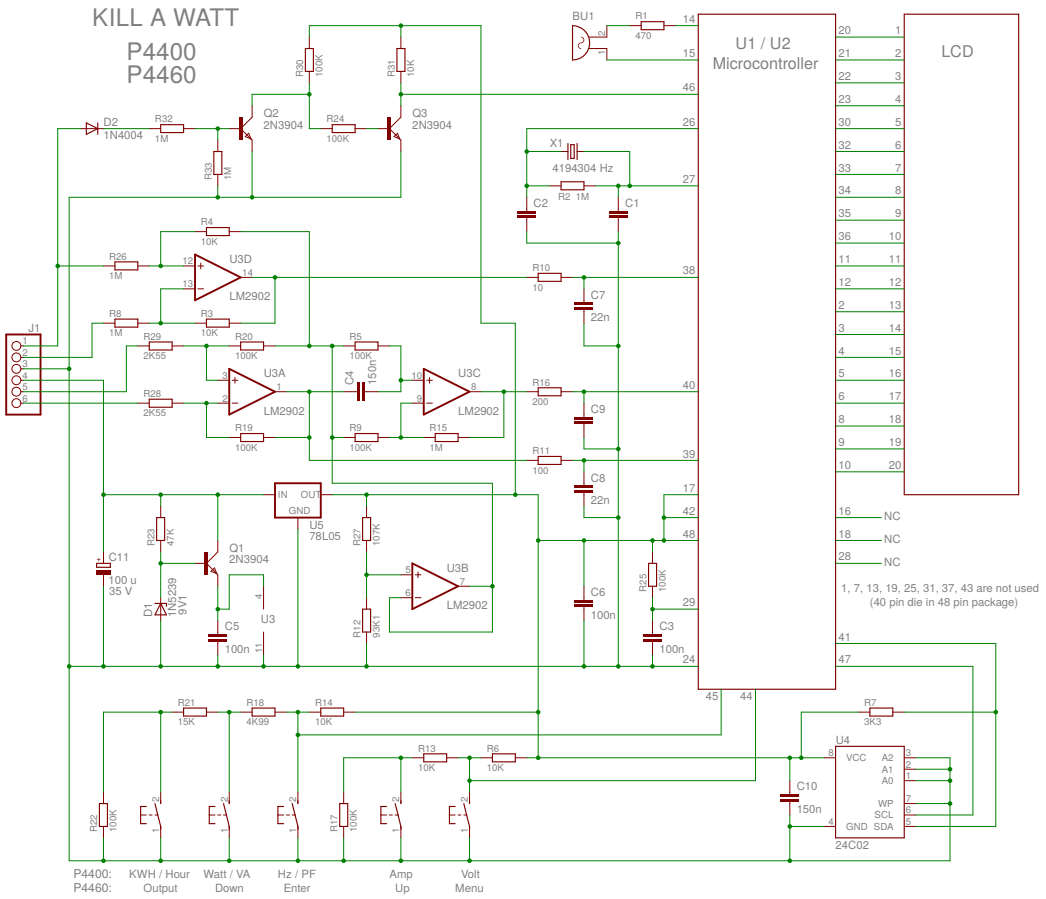


Figura 5.4: Placa de acondicionamiento y procesamiento.

toma de los bornes del capacitor C5. En este bloque también se genera el voltaje de referencia V_{REF} que será el componente de continua de las señales de salida a muestrear.

Para finalizar, el bloque de procesamiento cuenta con un microprocesador que tiene como entradas las señales antes mencionadas e incluye un detector de cruces por cero así como un display para mostrar los cálculos realizados.

Nota: una descripción más detallada del dispositivo se encuentra en el Apéndice B.1.2.

5.3. Integración de Hardware

El diseño general de los nodos consiste en la integración de los bloques antes mencionados con las adaptaciones, modificaciones o mejoras necesarias en cada caso. En la presente sección se explicará el proceso de diseño llevado a cabo para integrar los diferentes bloques de hardware previamente descritos.

5.3.1. Esquema general de funcionamiento

En la Fig. 5.5 se muestra un diagrama de los diferentes bloques de hardware que conforman los nodos.

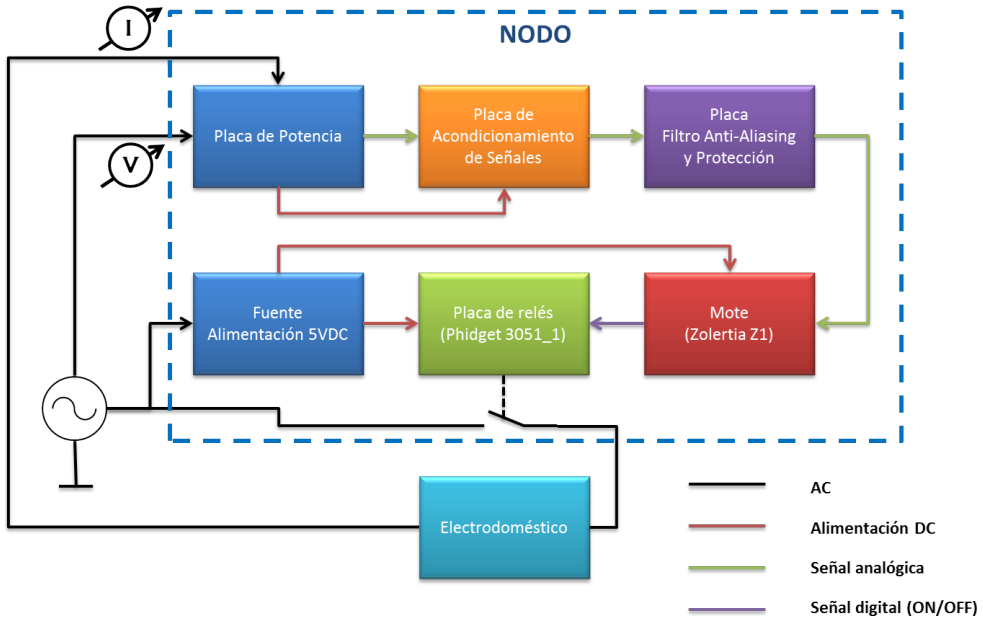


Figura 5.5: Diagrama de bloques detallado de los nodos.

La Placa de potencia corresponde a la etapa de potencia del KillAWatt (Fig. 5.3), con cambios en los valores de alguno de sus componentes, que serán explicados más adelante. Su función principal es la de realizar una primera atenuación en los niveles de tensión sensada y convertir la corriente consumida por el electrodoméstico a una señal de tensión. Estas señales son luego acondicionadas en la placa de acondicionamiento, cuya electrónica es alimentada por la placa de potencia.

La placa de acondicionamiento es la encargada de convertir las señales recibidas de la placa de potencia a rangos adecuados para su muestreo mediante los ADCs. Su diseño está fuertemente inspirado en la etapa de acondicionamiento del KillAWatt. Sin embargo en este caso se realizaron cambios significativos con el objetivo de lograr una buena integración con el Z1.

La placa de Filtro Anti-Aliasing y Protección fue diseñada desde cero y su función es la de realizar un filtrado pasabajos de las señales a muestrear con el fin de reducir el efecto de *aliasing*. A su vez, esta placa incorpora mecanismos de protección de los ADC, limitando la amplitud de las señales a muestrear.

El mote Z1 es el encargado del muestreo y procesamiento de las señales, el comando de la placa de relés mediante salidas digitales, y la comunicación con la HAN. El mote es utilizado tal cual se suministra por el fabricante, sin realizar ningún cambio o agregados a su diseño.

La placa de comando de carga es el dispositivo Phidget 3051. Se utiliza tal cual es suministrada por el fabricante sin cambios en su diseño. Su función es la

de abrir y cerrar el circuito de alimentación de la carga en base a la señal recibida en sus entradas digitales. Esta señal proviene de salidas digitales del Z1.

Dado que la placa de potencia, cuyo diseño parte del KillAWatt, no cuenta con potencia suficiente para alimentar todo el conjunto de bloques, es necesario recurrir a una fuente auxiliar. Esta consiste en una fuente con aislación galvánica (transformador), rectificador de onda completa y etapa de filtrado. Tanto el Z1 como la placa de comando de carga son alimentados mediante esta fuente.

5.3.2. Diseño de alimentacion de Placa de Comando y Z1

Como se explica en el Apéndice B.1.1, el mote Z1 puede ser alimentado tanto en 3VDC como en 5VDC a través de su puerto USB. La placa de comando por su parte debe alimentarse en el rango de 3,3VDC a 12VDC. Con el objetivo de simplificar el diseño y uniformizar valores de tensión se elige entonces fijar la alimentación de ambos en 5VDC.

Los 5VDC se obtienen de una fuente auxiliar 100-240 VAC/ 5VDC, que no puede ser conmutada. Por el contrario, debe utilizar un transformador de modo de evitar ruido de alta frecuencia que introduciría la fuente conmutada, y así evitar afectar la medida además de asegurar aislación galvánica. Esto último es necesario para la integración de las placas de Potencia y Acondicionamiento tal como se explica más adelante.

El el caso del mote Z1, se propone realizar la alimentación a través del puerto USB con un conector micro USB. Si bien esto tiene la desventaja de volver inaccesible el puerto durante el funcionamiento del nodo, mejora la robustez del diseño, protegiendo el MCU de los espurios de la red, al contar el puerto USB con una etapa de aislación mediante bobinas.

En cuanto a la potencia de la fuente, esta debe ser dimensionada teniendo en cuenta los consumos máximos de la placa de comando y el Z1. En el caso de la placa de comando, se observó en secciones anteriores que el consumo máximo es de 180mA (Fig. 5.1). Para obtener una estimación del consumo máximo del Z1 se sumaron los consumos máximos declarados para cada uno de los bloques que lo componen. Estos consumos pueden consultarse en [8], y su suma es de 160 mA en el peor caso. El máximo consumo para la fuente es entonces:

$$Isup_{MAX} = Irelay_{MAX} + Imote_{MAX} = 180mA + 160mA = 340mA \tag{5.4}$$

De aquí que la potencia será:

$$Psup_{MAX} = 5V.340mA = 1,7W \tag{5.5}$$

Se propone entonces el uso de una fuente comercial estándar por motivos de simplicidad. A continuación se resumen las especificaciones de la misma.

Especificaciones Fuente de Alimentación 5VDC

- Tensión entrada: 100-240 VAC

- Tensión salida: 5 VDC
- Potencia > 1,7 W
- Aislado galvánicamente

5.3.3. Diseño de integración de Z1 con la Placa de Comando

La placa de comando cuenta con dos relés R0 y R1, gestionados por dos entradas lógicas independientes. Si bien a los efectos funcionales del diseño del nodo sólo se necesita un relé para comandar la carga, se prevé la conexión de ambos relés a salidas lógicas del Z1 de forma de tener comando sobre los dos. Esto puede permitir en el futuro realizar algún tipo de comando más complejo y brinda un fácil repuesto en una eventual avería de un relé. Arbitrariamente se designa el relé R0 como el principal y R1 como auxiliar.

Al estar ambos bloques alimentados con la misma fuente, la referencia de 0VDC (Vss) es la misma para ambos, por lo cual las salidas digitales del Z1 tienen la misma referencia que las entradas digitales de la placa de comando. Para asegurar el correcto funcionamiento de las señales digitales, es necesario asegurar que los niveles lógicos (VL) y (VH) de las salidas del Z1 son compatibles con los de las entradas de la placa de comando.

De acuerdo a las hojas de datos del MSP430 [6], en el caso de las salidas del Z1, el nivel bajo se encuentra entre Vss y Vss+0,6 y el nivel alto está en el rango de Vcc-0,6 a Vcc, con Vss=0V y Vcc igual a 3V para una alimentación de 5VDC a través del puerto USB del mote Z1.

En el caso de la placa de comando, el fabricante no declara los rangos de tensión admisible en las puertas lógicas, por lo cual la compatibilidad con las salidas del Z1 se comprobó en forma empírica confirmando que los niveles lógicos son compatibles con la alimentación elegida.

El otro aspecto a definir es la elección de los pines GPIO del Z1 a utilizar como salidas lógicas para el comando de los relés. Tal como se describe en las hojas de datos del Z1 [8] el mote dispone de un puerto dedicado en forma casi exclusiva a pines que pueden ser configurados y utilizados como GPIO. Se trata del puerto JP1C. Sin embargo, algunos pines disponibles en el puerto JP1A (puerto de acceso a los ADCs), descrito en el Apéndice B.1.1, pueden ser también configurados como GPIO. Este es el caso de los pines P6.5 y P6.6 del MCU, correspondientes a los pines 10 y 8 del puerto JP1A respectivamente. Estos pines son las entradas a los ADC 5 y 6, pero pueden ser configurados como salidas digitales. Por simplicidad de diseño, para utilizar un solo puerto del mote, se propone utilizar estos dos pines como las salidas digitales para el comando de los relays, con la asignación de señales mostrada en la Tabla 5.1.

Pin Z1	Pin MCU	Terminal Comando	Descripción
JP1A - 10	P6.5	Control - 0	Control del relé R0 (Relé principal)
JP1A - 8	P6.6	Control - 1	Control del relé R1 (Relé auxiliar)

Tabla 5.1: Asignación de salidas digitales para el comando de relés.

5.3.4. Diseño de placas de potencia, acondicionamiento y filtrado de señales

Luego del análisis del circuito del KillAWatt y dado que se desconoce la lógica del microcontrolador se decide reutilizar solamente las etapas de potencia, alimentación y acondicionamiento de señales introduciendo algunas modificaciones:

- Ajuste del voltaje de referencia.
- Ajuste de la señal corriente.
- Ajuste de ganancia de tensión para medir hasta 250VAC.
- Eliminación de uno de los canales de medición de corriente.

Ajuste del voltaje de referencia

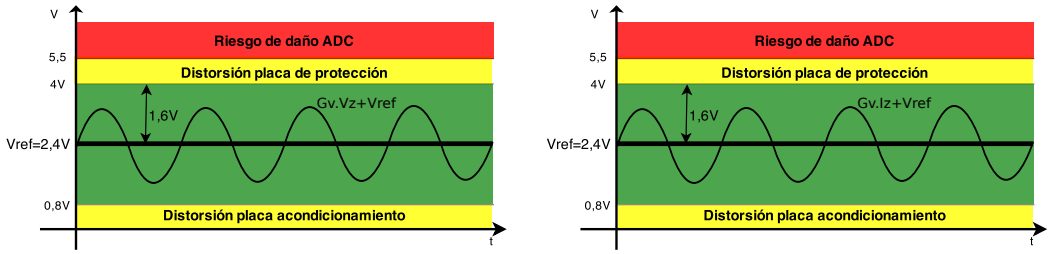
El voltaje de referencia V_{REF} es la componente de continua de las señales de salida a muestrear. Esto es, las señales a muestrear se encuentran montadas sobre la continua V_{REF}.

Es preciso definir el nuevo rango de voltaje a la salida a los efectos de poder fijar correctamente los niveles de continua y las ganancias de las señales. Los requerimientos en este sentido vienen dados por las características y rangos de funcionamiento de los ADC en conjunto con sus correspondientes etapas de filtrado anti-aliasing y protección.

Se decide utilizar la opción de *referencia interna* como voltaje de referencia en los ADC a los efectos de simplificar el diseño. El voltaje de referencia interno puede fijarse en $1,5V$ o 2,5V. Para no perder rango en la adquisición se utiliza $Vr_+ = 2,5V$. C, Existen dos ADC del Z1 que cuentan con un divisor resistivo previo a la entrada de señal del MCU. La utilización de tal divisor permite aumentar el rango de señal efectivo admitido a la entrada del mote por un factor de $5/3$[4]. En este escenario, el voltaje máximo a la salida de la placa de filtrado y protección es de $\frac{5}{3}Vr_+ = 4,17V$. Sin embargo, y como será explicado más adelante, debido a distorsiones que introducen los diodos de esta etapa se acotará la salida de la placa de potencia y acondicionamiento hasta los 4V.

El rango de salida tiene también limitaciones por abajo, producto de los amplificadores operacionales en la etapa de acondicionamiento. Mediciones realizadas

[4]Por detalles consultar Apéndice B.1.1



(a) Señal de voltaje.

(b) Señal de corriente.

Figura 5.6: Excursión de voltaje a la salida.

durante el ensayo de la placa indican que existe distorsión para tensiones inferiores a 0,8V. Por lo visto anteriormente el rango de voltaje de señal (tanto para señal de tensión como de corriente) a la salida de esta placa se fija en:

$$0,8V \leq v_V = v_v + V_{REF} \leq 4V$$

$$0,8V \leq v_I = v_i + V_{REF} \leq 4V$$

Para maximizar la excursión de salida se toma entonces,

$$V_{REF} = \frac{4V + 0,8V}{2} = 2,4V$$

A partir de la ecuación B.10 se obtienen nuevos valores de R_{12} y R_{27}. Se elige $R_{27} = 68k\Omega$ y $R_{12} = 63k\Omega$.

Como se esquematiza en la Fig. 5.6, la excursión de voltaje a la salida es de $1,6V$.

Tal como será explicado en secciones posteriores, para realizar los cálculos en la etapa de procesamiento es necesario conocer el valor de continua de las señales (V_{REF}) a nivel del software. Una posibilidad es medir este valor en laboratorio en una etapa inicial de calibración e incorporarlo como parámetro constante en el programa. Sin embargo, esto podría introducir errores debidos a corrimientos en el voltaje causados por ejemplo por cambios en la temperatura de los componentes. Por este motivo se diseña el circuito previendo la adquisición de V_{REF} por parte del ADC. Tal como será explicado en la Sección 5.5 este valor será utilizado a nivel de software para implementar una estrategia de auto adaptación ante pequeñas variaciones en el valor de tensión de V_{REF}.

Ajuste de la señal de corriente

En los requerimientos del medidor de corriente fue establecido que se deben poder medir corrientes de hasta al menos 10A. Dado que el KillAWatt soporta corrientes de hasta 15A, el fondo de escala del medidor se establece en 15A.

A los efectos de economizar recursos de ROM y RAM se opta por eliminar uno de los canales de corriente. Contar con dos etapas de amplificación tiene por objetivo una disminución de la relación señal a ruido de cuantización (SQNR) para corrientes pequeñas pero no disminuye la relación señal a ruido (SNR) de la señal, puesto que el ruido eléctrico se amplifica junto con la misma. Dado que la resolución de los ADC es relativamente alta (12 bits), la amplitud de la señal a muestrear, y por ende el SQNR, no presenta una gran limitación. Por el contrario, los recursos de ROM del MCU sí resultan escasos, lo cual justifica este compromiso. Como se dijo anteriormente, esta opción presenta como desventaja el incremento en los errores de cuantización para señales pequeñas. Los cálculos realizados de SQNR para corrientes pequeñas indican que este impacto no es significativo. Para un ADC de B Bits, con margen dinámico X_m y amplitud de la señal de entrada al ADC X_p, el SQNR está dado por la ecuación descrita en [37]:

$$SQNR = 10\log_{10}\left(\frac{\sigma_x^2}{\sigma_e^2}\right) = 6,02B + 10,8 - 20\log_{10}\left(\frac{X_m}{X_p/\sqrt{2}}\right) \tag{5.6}$$

Para una corriente de entrada de 100mA que se corresponde con una señal de entrada al ADC de amplitud $X_p = 9,6mV$, considerando el margen dinámico $X_m = 2,5V$, se calculó una relación señal a ruido de cuantización (SQNR) de 31dB si se implementa una sola etapa de amplificación. A fondo de escala (corrientes de 15A), el SQNR aumenta a 75dB.

Para maximizar la excursión de salida, la ganancia de la etapa de amplificación de la señal de corriente debe ser:

$$G_I = \frac{1,6V}{15\sqrt{2}A} = 7,54 \times 10^{-2}\Omega \tag{5.7}$$

De donde,

$$\frac{R_{20}}{R_{29}} = \frac{G_I}{R_{shunt}} = 35,91$$

La ganancia queda entonces fijada utilizando los valores $R_{29} = 2,4k\Omega$ y $R_{20} = 84,5k\Omega$, lo que da la siguiente ganancia esperada:

$$G_I = 7,39 \times 10^{-2}\Omega$$

Dado que se eliminó la segunda etapa de amplificación de la señal de corriente, en el nuevo diseño no se incluye el amplificador operacional U3C.

Ajuste de la señal de voltaje

Una de las principales modificaciones al diseño del KillAWatt consiste en adaptarlo para posibilitar lecturas de hasta 250VAC, dado que el valor máximo admitido por el KillAWatt es de 125VAC. Para ello es necesario introducir modificaciones en la ganancia de la tensión para que el voltaje de red se mapee en el rango de conversión de la placa de protección y filtro.

Dado que no es de interés utilizar la etapa de rectificación de media onda de la tensión de entrada descrita en la Apéndice B.1.2, se decide eliminar el diodo D3 de la placa de potencia.

A los efectos de ubicar la señal de voltaje dentro del rango de salida de la placa de acondicionamiento se debe fijar la ganancia de tal modo que la tensión máxima admitida en la carga ($250\sqrt{2}V$) sea mapeada en el límite superior del rango de operación, esto es:

$$G_V = \frac{1,6V}{250\sqrt{2}} = 4,53 \times 10^{-3} \tag{5.8}$$

Para modificar la ganancia al valor deseado, a partir de la ecuación B.9, se elige $R_3 = 76k\Omega$.

Resumen del rediseño del circuito

Las modificaciones al diseño del KillAWatt se pueden apreciar en las Figs. 5.7 y 5.8, y se detallan a continuación:

- Modificación de resistencias: $R_{27} = 68k\Omega, R_{12} = 63k\Omega, R_{29} = 2,4k\Omega, R_{20} = 84,5k\Omega, R_3 = 76k\Omega$
- Eliminación del Diodo D3
- Eliminación del Amplificador Operacional U3C

Ganancias y valores de continua esperados:

- $G_I = 7,39 \times 10^{-2}\Omega$
- $G_V = 4,53 \times 10^{-3}V/V$
- $V_{CC} = 8,4V$
- $V_{REF} = 2,4V$

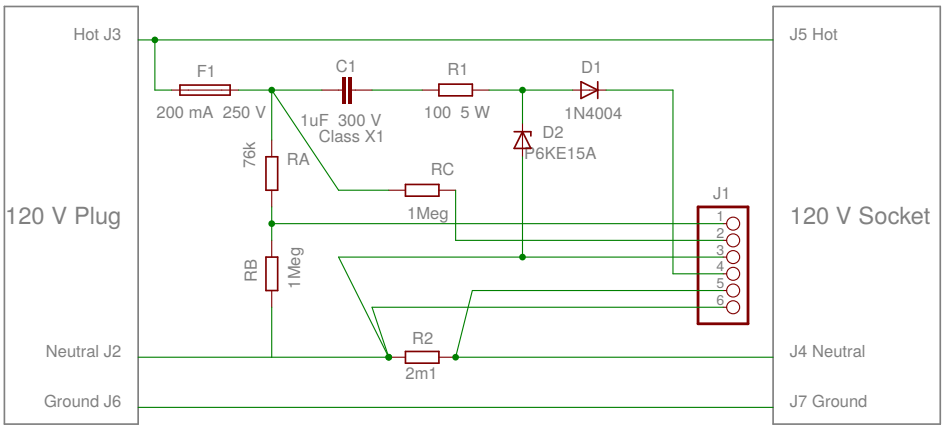


Figura 5.7: Nuevo circuito de potencia.

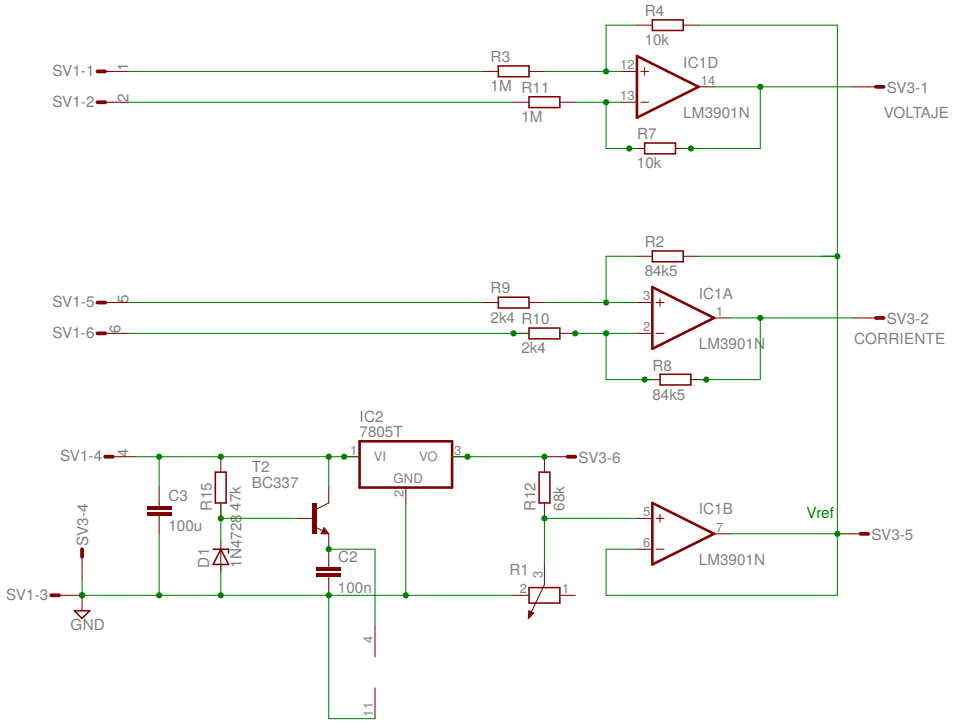


Figura 5.8: Nuevo circuito de acondicionamiento.

Muestreo y Anti-Aliasing

A la hora de realizar el muestreo de la señal con el Z1 uno de los parámetros más importantes a definir es la frecuencia de muestreo. La misma debe ser suficiente para obtener una representación de la señal que contemple los armónicos que se desean medir. En base a la frecuencia de muestreo elegida, se debe incluir una etapa de filtrado previo al muestreo de la señal de modo de evitar o disminuir el efecto del fenómeno de *aliasing*.

Si bien existen filtros complejos que permiten lograr mejor selectividad, por razones de simplicidad se decidió utilizar un filtro RC de primer orden como el que se muestra en la Fig. 5.9. Esta decisión se basó también en numerosos diseños de referencia consultados, que se citan e incluyen en la discusión presentada en el Apéndice B.1.3. La elección de la frecuencia de muestreo se eligió en base a un compromiso entre distorsión introducida por debajo de la frecuencia de corte (banda de interés) y la atenuación lograda en la zona de *aliasing*. El diseño de este filtro constituye un punto mejorable del circuito. El lector puede consultar la discusión completa del diseño del filtro, incluyendo la explicación de los compromisos asumidos y posibles opciones de mejora futura en el Apéndice B.1.3.

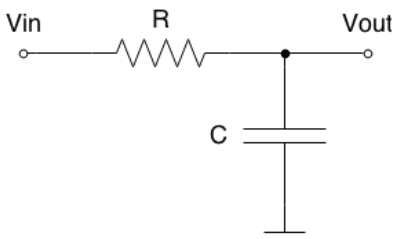


Figura 5.9: Esquema del filtro RC.

Tras la discusión antes mencionada se resolvió utilizar una frecuencia de muestreo de 8KHz, e implementar un filtro con las siguientes especificaciones:

- $R = 1k\Omega$
- $C = 33nF$

La frecuencia de corte del filtro es entonces:

$$fc = \frac{1}{2.\pi.RC} = 4{,}82kHz \tag{5.9}$$

Protección de los ADC

La Placa de Acondicionamiento de Señales fue diseñada con un rango de voltaje a la salida $[0,8; 4]VDC$. Sin embargo, ante señales de entrada a dicha placa que excedan el rango previsto, o fallas en la electrónica de la misma, por ejemplo en las etapas de ganancia, no es posible asegurar que las señales a la salida de esta placa no excederán los márgenes de seguridad. De hecho, el límite superior a la salida vendría dado por la excursión superior de los operacionales, que está muy por encima de los 5VDC fijados como límite. Por este motivo, es necesario incluir en el diseño algun sistema de protección que resguarde la electrónica de los ADC limitando los niveles de señal en sus entradas.

En el Apéndice B.1.4 se explica con detalle este punto y se presenta una descripción y discusión de diferentes métodos que podrian utilizarse para lograr este objetivo. A los efectos del presente diseño se utilizan diodos zener en la configuración de estabilizador de tensión de modo de evitar que el nivel de la señal exceda los valores de umbral. Para limitar el nivel de las señales al umbral propuesto de 5VDC se propone utilizar diodos de 4,7V. En la Fig. 5.10 se muestra esta configuración.

La resistencia R sirve para proteger al diodo en caso de que la tensión supere el umbral de ruptura y el mismo comience a conducir. Las especificaciones del circuito de protección se definen en el Apéndice B.1.4 y se indican a continuación:

- Diodo zener: Vz=4,7V (P=1W)
- Resistencia: 300Ω

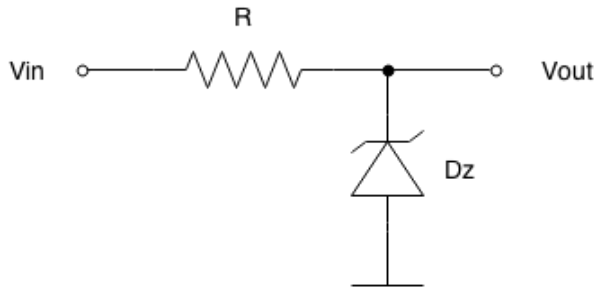


Figura 5.10: Protección con diodos zener como limitadores de voltaje.

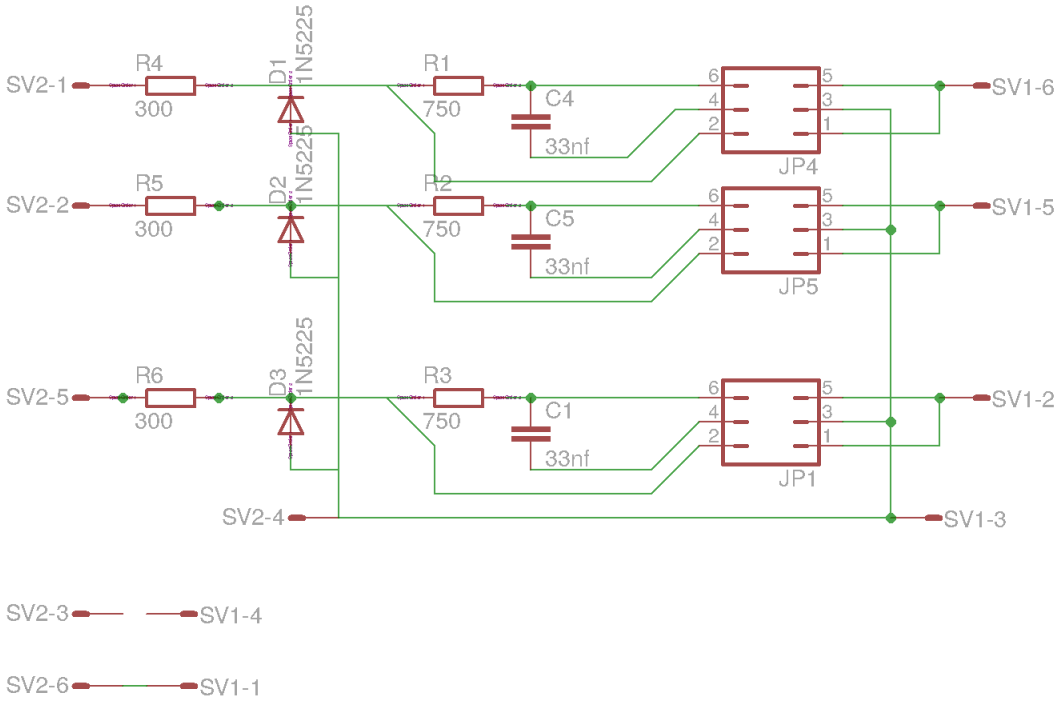


Figura 5.11: Esquemático del circuito del filtro anti-aliasing y protección de ADC.

Resumen del diseño de la Placa de Filtrado y Protección

A partir de lo visto anteriormente, se integran las funcionalidades de filtrado y protección utilizando la configuración mostrada en la Fig. 5.11, donde la resistencia calculada para el filtro se divide en dos resistencias de valores 300Ω y 750Ω.

5.4. Implementación de hardware

En esta sección se detallará la implementación de hardware a partir del diseño presentado en las secciones anteriores.

5.4.1. Placa de potencia

En la Fig. 5.12 se muestran las modificaciones realizadas sobre la placa de potencia del KillAWatt, que consistieron en:

- $R_3 = 76k\Omega$
- Eliminación del Diodo D3

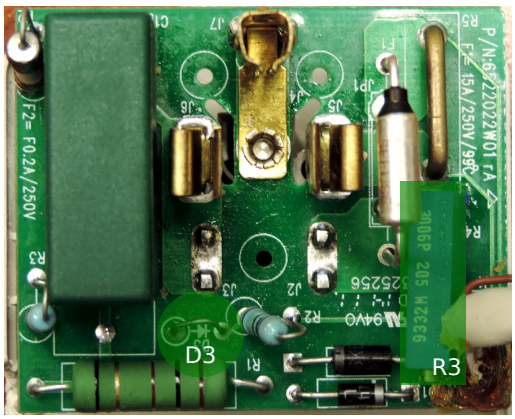
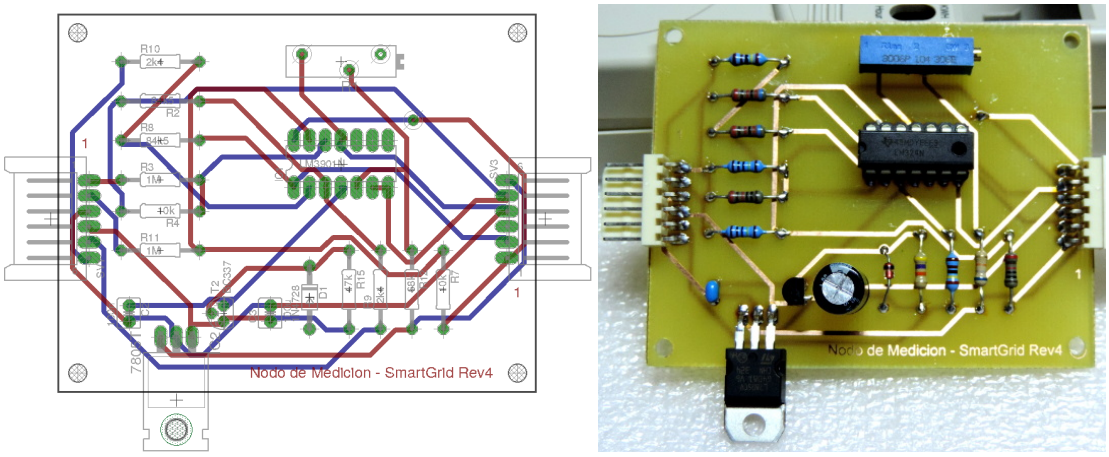


Figura 5.12: Modificaciones realizadas a la placa de potencia del KillAWatt.

Para evitar la rectificación de media onda de la señal de voltaje, se reemplaza el Diodo D3 por un puente conductor. Para el cambio de ganancia de la tensión, en lugar de colocar el valor de resistencia nominal calculado ($76k\Omega$), se coloca un preset de forma de permitir un reajuste de la ganancia en caso de necesidad.

5.4.2. Placa de medición y acondicionamiento

Debido a que esta parte de la placa del KillAWatt está implementada con tecnología de montaje superficial (SMT por su sigla en inglés), por la dificultad encontrada en modificar satisfactoriamente el circuito se decide realizar una placa propia sobre la base del esquemático de la Fig. 5.8. El diseño del PCB de doble capa se muestra en la Fig. 5.13. En la Tabla B.2 del Apéndice B se detalla la lista de componentes.



(a) Diseño de PCB. (b) Placa fabricada.

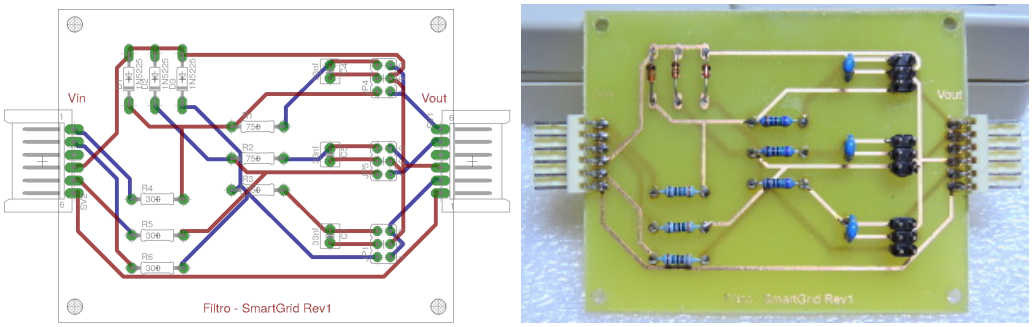
Figura 5.13: Desarrollo de la placa de medición y acondicionamiento.

5.4.3. Placa de filtrado y protección

Se diseñó un PCB para el filtrado y protección de las señales de voltaje y de corriente. Tal como se explicó en secciones previas, se incluye en el diseño la señal de referencia V_{REF} de modo que se encuentre disponible a la entrada del mote para ser muestreada y luego utilizar su valor para realizar un ajuste automático del valor de referencia mediante el software.

Puesto que las señales de voltaje y de corriente son afectadas por el filtrado de la etapa anti aliasing, se aplica esta misma etapa también a V_{REF}. Esto se hace no solo para eliminar posibles ruidos de alta frecuencia en dicha señal, sino también para que tanto V_{REF} como las señales de corriente y tensión, y en particular sus respectivas componentes de continua, sufran exactamente la misma atenuación durante esta etapa. En caso contrario, la señal V_{REF} dejaría de ser representativa del nivel de continua de las señales tras la etapa de filtrado. El filtro afecta la ganancia de las señales, aún en banda pasante, por lo que la resistencia variable de la etapa de potencia puede ajustarse de forma que a la salida de la placa de filtrado y protección alcance el máximo valor del rango sin distorsión (4V) para tensiones de 250VAC.

Finalmente para poder evaluar los efectos del filtro, se colocan jumpers de forma de poder habilitar o deshabilitar sencillamente la etapa de filtrado. Con este diseño la habilitación del filtro es independiente de la etapa de protección que siempre se encuentra activa.



(a) Diseño de PCB. (b) Placa fabricada.

Figura 5.14: Desarrollo de la placa de filtrado y protección.

En la Fig. 5.14 se muestra el diseño PCB y una fotografía de la placa de protección y filtrado implementada. En la Tabla B.3 de Apéndice B se listan los componentes utilizados.

5.4.4. Integración de Hardware

El diseño modular permite conectar fácilmente los distintos componentes del sistema, como se muestra en el esquema de conexionado de la Fig. 5.15.

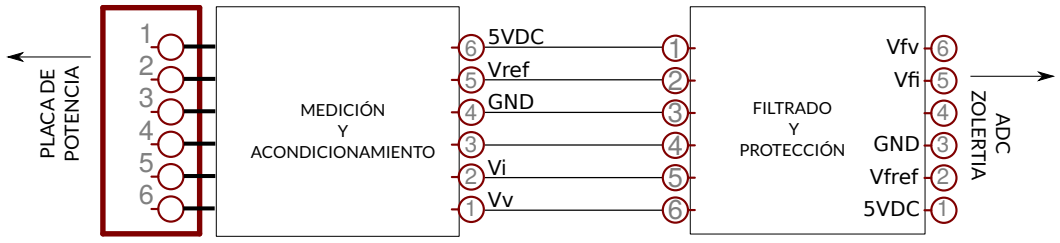


Figura 5.15: Diagrama de conexiones y referencia de pines.

En la Fig. 5.16 se muestra la incorporación de los módulos de comando (relé Phidget) y el módulo de procesamiento y comunicación (mote Z1). El mote se conecta por un lado a las salidas de la placa de filtrado y protección V_{fV},V_{fI},V_{fREF} y GND. Por otra parte se conecta con las 2 entradas de control y de alimentación del módulo de comando.

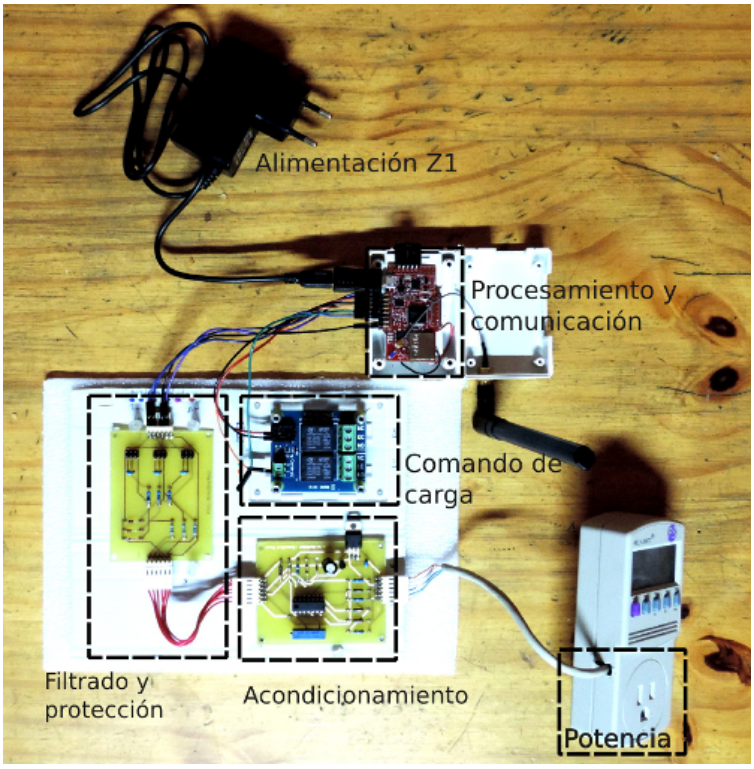


Figura 5.16: Placa de medicion y acondicionamiento KillAWatt en tecnología SMT

Como fuente de alimentación para el mote Z1 y la placa de comando se utilizó un cargador de celular estándar con transformador (aislación galvánica) de acuerdo a lo definido en la Sección 5.3.2. Los datos característicos de la fuente son:

- INPUT: 100/240 VAC, 50/60Hz
- OUTPUT: +5V DC / 1,5A

5.5. Implementación de software de los Nodos

En el Capítulo 4 se explicaron las funcionalidades y programas vinculados con la comunicación HAN. En la presente sección se aborda el resto del software de los nodos, específicamente las funciones de muestreo, medición y control de la placa de comando. Para facilitar la lectura y evitar una extensión excesiva del capítulo, se brindan aquí los lineamientos más importantes del desarrollo e implementación, sin entrar en los detalles más específicos. En este sentido, no se presenta aquí una descripción detallada de directorios, código fuente programado, formato de representación de variables, parámetros específicos de Contiki, etc. No obstante, si el lector tuviera interés en conocer este tipo de detalles, los mismos pueden ser consultados en el Apéndice “Programación Contiki” A.

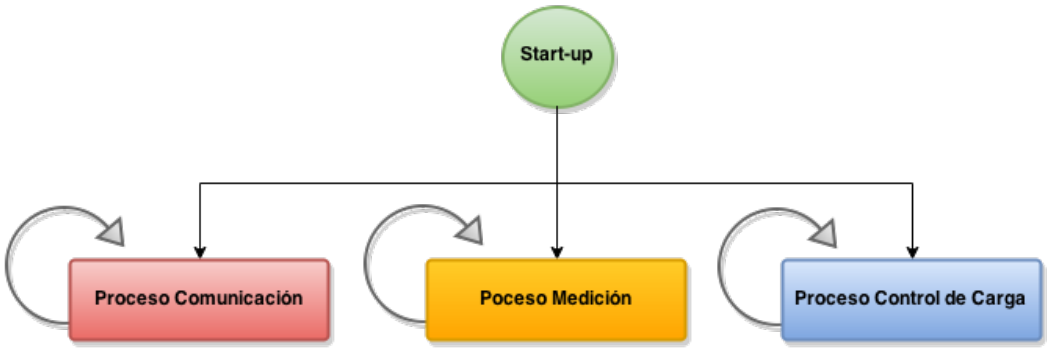


Figura 5.17: Diagrama de flujo de los hilos de procesamiento principales del software de los nodos.

El programa principal de los nodos (implementado en el fuente *er-server-SG.c*, está compuesto por tres procesos diferentes a nivel de Contiki[5]. Estos son:

- *PROCESS_THREAD(comunicacion, ev, data)*: Se encarga de gestionar la comunicación de acuerdo a lo explicado en el Capítulo 4.
- *PROCESS_THREAD(medicion, ev, data)*: En este protothread[6] se concentran las operaciones de muestreo y procesamiento de mediciones.
- *PROCESS_THREAD(control_carga, ev, data)*: Encargado de manejar los pines de salidas digitales del mote para encender o apagar un dispositivo a través de la Placa de Relés.

En la Fig. 5.17 se muestra un esquema simplificado de los tres hilos principales de procesamiento. Los procesos son gestionados por Contiki. Todas las funcionalidades de soporte propias del sistema operativo así como la ejecución de los stacks de comunicación son dejadas de lado en esta descripción.

El proceso de comunicación simplemente se encarga de iniciar los servicios de comunicación durante el start-up. Dichos servicios fueron descritos en el Capítulo 4. En las siguientes secciones se pondrá atención sobre los procesos de medición y comando de carga.

5.5.1. Proceso de medición

Este proceso se encarga de gestionar la utilización de los ADC, realizar el muestreo y procesar las muestras para obtener las mediciones deseadas. Tal como se explicó en secciones anteriores, se realiza el muestreo de 3 señales provenientes de la Placa de Acondicionamiento de Señales. Estas son *Vfv, Vfi* y *Vref*.

De acuerdo a lo explicado en la Sección 5.3.4 el muestreo se realiza a 8KHz. Debido a que las capacidades del mote son limitadas, especialmente en términos

[5]Por información acerca de procesos en Contiki consultar https://github.com/contiki-os/contiki/wiki/Processes, Mayo 2015.

[6]https://github.com/contiki-os/contiki/wiki/Processes#Protothreads, Mayo 2015.

de RAM (8KB), el número de muestras a adquirir y almacenar no puede ser excesivo. De hecho se comprobó empíricamente que no se podían almacenar más de 250 muestras de las tres señales sin comprometer el desempeño de otras funcionalidades. Por tal motivo, a la frecuencia de muestreo propuesta no es posible adquirir varios ciclos de las señales en un barrido, ya que a 50Hz de frecuencia de red se necesitarían 160 muestras para almacenar un ciclo. Por otro lado, realizar el procesamiento y cálculo de las mediciones en base a la adquisición de un único ciclo puede llevar a grandes errores por causa de efectos espurios.

Para resolver este problema se utilizó una estrategia intermedia; para cada señal se realiza el muestreo de un ciclo completo y luego el cálculo correspondiente de la medida, repitiendo este procedimiento cuatro veces. El resultado final se obtiene mediante el promedio de las cuatro medidas parciales. Más específicamente, los muestreos realizados son de un ciclo y cuarto de duración, ya que se utiliza un cuarto de ciclo extra para el cálculo de la potencia reactiva Q en base a la fórmula (3.4) introducida en el Capítulo 3.

En la Fig. 5.18 se muestra un diagrama de flujo simplificado de este proceso. Los parámetros indicados entre corchetes hacen referencia a parámetros seteables en el software de los nodos. En forma predeterminada TIEMPO_ENTRE_REPORTES equivale a 4 segundos y TIEMPO_ENTRE_MEDIDAS equivale a 0,5 segundos. Se observa que luego de terminado un muestreo, las muestras se pre-procesan obteniendo un resultado parcial que luego será promediado con el resto.

Puesto que el cálculo de las medidas requiere de Vfref, este valor se adquiere en un primer lugar. Se monitorea el valor de Vfref para quitar el valor de continua a las señales de tensión y corriente para realizar un correcto procesamiento. El objetivo de esto es lograr una suerte de auto sintonía que mejora la precisión de la medida en caso de una eventual desviación del valor de Vref en la electrónica, disminuyendo el error con respecto a la situación más sencilla en la cual el Vref se considera una constante preconfigurada en el software.

Para realizar el muestreo se utiliza un tipo de temporizador especial ofrecido por Contiki, que permite medir tiempos sumamente cortos y cuyo funcionamiento se basa en interrupciones, por lo cual tiene prioridad sobre los procesos generales. Esta familia de temporizadores se denomina *rtimer*[7]. Se utiliza entonces un rtimer para medir los tiempos entre la adquisición de cada muestra. La resolución del rtimer en el MCU utilizado es de 1/32768 segundos, por lo cual la frecuencia de muestreo deberá corresponder a un valor divisor de 32768. Este valor se configura en 8192Hz, esto es, fs=8,192KHz.

Para poder utilizar los ADC correctamente los mismos son configurados por software al comienzo del programa (mediante el código ubicado en los archivos *z1-ADCs-SG.h* y *z1-ADCs-SG.c*). Entre otras cosas se configura la referencia interna en 2.5V definida en la Sección 5.3.4.

Por más detalles acerca de la implementación de las funciones de medición y configuración de los ADC a nivel de código, el lector puede consultar el Apéndice "Programación Contiki" A.4.

En la Fig. 5.19 se grafica una adquisición realizada por un nodo. Las señales

[7]https://github.com/contiki-os/contiki/wiki#Internals, Mayo 2015.

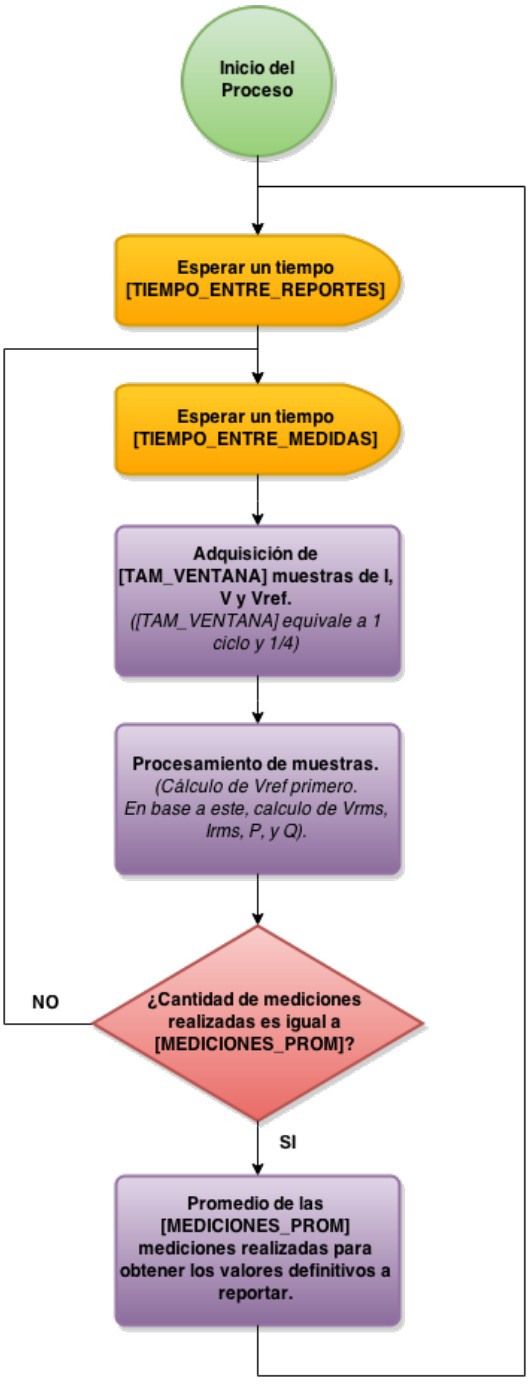


Figura 5.18: Diagrama de flujo del proceso de medición, parte del software de los Nodos.

fueron re-escaladas para que resulten comparables en magnitud, por lo cual la figura solo debe ser evaluada desde un punto de vista cualitativo. En la misma se muestran las señales de tensión, corriente, y también la corriente desfasada un cuarto de ciclo.

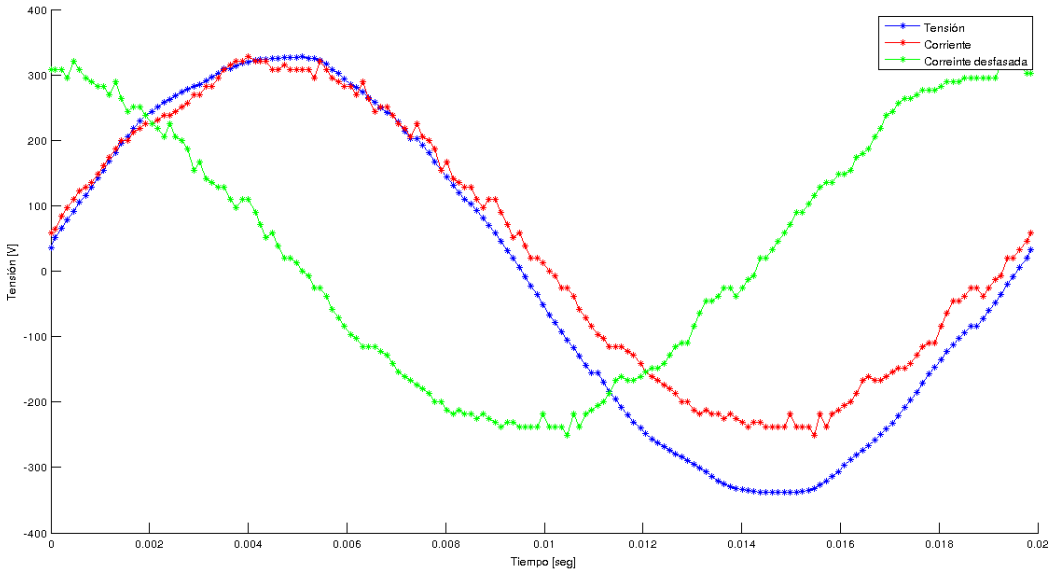


Figura 5.19: Ejemplo de adquisición de señales realizada con un nodo y graficadas con Matlab.

5.5.2. Proceso de control de carga

El proceso encargado del control ON/OFF de los relés de la placa de comando es sencillo. El proceso permanece en stand-by a la espera de un eventual mensaje de comando de parte del HEC recibido a través de CoAP. El recurso CoAP encargado del comando de los relés (*res-comando-SG.c*) ejecuta una interrupción llamando al proceso de control de carga del programa principal apenas recibe un mensaje de comando. Luego el proceso evalúa este mensaje y de acuerdo a la instrucción recibida activa o desactiva los relés mediante salidas digitales del MCU.

En la Fig. 5.20 se muestra un diagrama de flujo de este proceso. Se incluye un último paso accesorio mediante el cual se guarda en memoria una representación del estado actual de los relés para ser reportada al HEC, y a la vez se encienden o apagan dos LEDs del mote utilizados como indicadores testigo.

Por más detalles acerca de la implementación de proceso de comando de carga a nivel de código, el lector puede consultar el Apéndice “Programación Contiki” A.4.

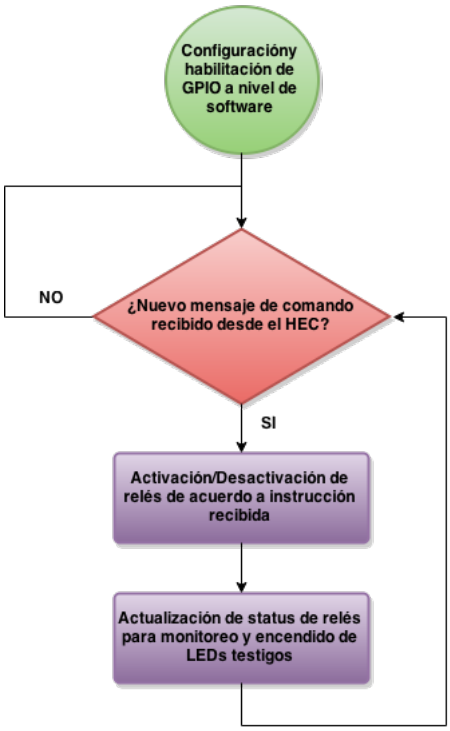


Figura 5.20: Diagrama de flujo del proceso de comando de carga, parte del software de los nodos.

5.6. Síntesis del capítulo

A partir del modelo y análisis de requerimientos de los nodos expuesto en el Capítulo 3 se presentó en este capítulo el diseño e implementación de los nodos. En particular se realizó una evaluación, discusión y elección de tecnologías utilizadas como base para la implementación, dentro de las cuales se encuentran la placa de relés Phidget, el medidor de energía KillAWatt y el mote Zolertia Z1, con la plataforma de software Contiki OS que fue elegida en el Capítulo 4.

El hardware de los nodos se diseñó en base a la integración de los componentes de hardware antes mencionados, con la incorporación de un filtro anti-aliasing y un circuito de protección. En el caso de la adquisición de señales se implementó un diseño similar al del KillAWatt, con algunas modificaciones y agregados.

En base a los diseños realizados y la integración de los módulos mencionados se presentó la implementación del nodo, incluyendo el diseño del montaje electrónico.

Finalmente se explicaron las características fundamentales del software embebido en el MCU de los nodos, programado para realizar el procesamiento de las señales, la comunicación con el HEC y el comando de los electrodomésticos.

Capítulo 6

Diseño detallado e implementación del Controlador (HEC)

6.1. Introducción

El HEC constituye el núcleo de la plataforma HEMS propuesta ya que es el encargado de centralizar y procesar la información recibida desde los nodos, ejecutar los algoritmos de optimización y en base a ellos emitir mensajes de comando hacia los electrodomésticos.

Dado que se propone en el presente trabajo la implementación de una plataforma abierta que permita la utilización de diferentes algoritmos de optimización, el HEC brinda una biblioteca de funciones (API) que sirve como interfaz de programación de forma de poder utilizar las funcionalidades básicas de la plataforma, tales como comunicación con los nodos de la HAN o la comunicación con el EMS. De esta forma la API permite aprovechar las funciones básicas de la plataforma HEMS y ponerlas a disposición de un algoritmo definido por el desarrollador que hace uso de la API.

En el presente capítulo se presenta el diseño detallado y la implementación del controlador (HEC), describiendo el hardware, el software, la API desarrollada y mostrando cómo fueron implementadas las distintas funcionalidades.

6.2. Plataforma de Hardware para el HEC

Para lograr un diseño de bajo costo y dimensiones reducidas, se decidió implementar el HEC en un Raspberry Pi [1], modelo B+ [2]:

- SoC Broadcom BCM2835
- CPU ARM1176JZF-S, 700MHz

[1] http://www.raspberrypi.org/, Mayo 2015.
[2] https://www.adafruit.com/datasheets/pi-specs.pdf, Mayo 2015.

- 512MB de RAM
- 10/100 Mbit/s Ethernet
- Alimentación en 5V DC con microUSB

A pesar de haber utilizado este equipo, el diseño de la plataforma y del software del HEC es independiente de este hardware particular. El HEC puede ser portado sin ninguna modificación a un PC convencional.

Existen varias distribuciones de GNU/Linux desarrolladas para este dispositivo, entre las que se seleccionó Raspbian. Sobre esta distribución de Linux pueden ejecutarse intérpretes Python, y este será el principal requerimiento de cualquier plataforma en la que se desee instalar el software del HEC desarrollado y utilizar la API para implementar algoritmos de optimización de consumo.

6.3. Border Router

Tal como se explicó en capítulos anteriores, el Border Router (gateway de acceso a la HAN por parte del HEC) se implementa utilizando un Mote Zolertia Z1 idéntico a los utilizados para los nodos, con el programa de *rpl-border-router* ofrecido como ejemplo de aplicación por la plataforma Contiki. Por detalles acerca de este dispositivo y su funcionamiento consultar el Capítulo 4.

Para conectar el mote que actúa como Border Router al HEC se crea una interfaz de red virtual en el sistema operativo del HEC. El mote se conecta a un puerto USB del HEC y luego se utiliza el Makefile incluido en la aplicación ejemplo provista por Instant Contiki que se encarga de crear la interfaz de red que servirá como punto de entrada a la red HAN. En el Apéndice C se repite esta explicación detallando las líneas de código específicas para realizar la conexión.

6.4. Plataforma de Software basada en Python

Python es un lenguaje interpretado, multiplataforma, con tipado dinámico y orientado a objetos. Cuenta con una activa y extensa comunidad de desarrolladores [42], y una excelente documentación en línea de la web oficial[3]. Dos buenas referencias para comenzar familiarizarse con este lenguaje son los libros de Gonzalez [24] y Lutz [34].

En secciones posteriores se describirán los diferentes módulos de software implementados en Python. Para ello se introducen en esta sección algunos conceptos fundamentales, de forma de facilitar la comprensión al lector. Mayores detalles se brindan en el Apéndice C. Sin embargo, esta documentación no pretende ser una guía sobre Python. Se asume que el lector cuenta con conocimientos básicos de programación y nociones de diseño orientado a objetos. Además se brindarán en forma complementaria algunas referencias de consulta donde profundizar los conceptos abordados.

[3]https://docs.python.org/3/, Mayo 2015.

Las principales características que hacen a Python el lenguaje elegido para el diseño son su implementación open-source y posibilidad de rápido aprendizaje, en parte debido a su clara sintaxis. Este último punto lo hace ideal para implementar módulos con el objetivo de que sean utilizados y eventualmente adaptados o mejorados por terceros.

Si bien está fuertemente orientado a objetos, es multiparadigma, por lo que se puede utilizar para programación funcional y estructurada, y cuenta con funcionalidades para el manejo de excepciones.

A las ventajas ya mencionadas, se suma una amplia colección de bibliotecas. Como contrapartida a la flexibilidad y portabilidad que ofrecen los lenguajes interpretados, hay que tener en cuenta la eventual desventaja de su performance comparado con lenguajes compilados. En este sentido es oportuno precisar que se trata en realidad de un lenguaje semi-interpretado, ya que el código se puede traducir a bytecodes que luego son ejecutados por el intérprete. Existen varias estrategias para optimizar la velocidad de ejecución. Entre ellas está la posibilidad de crear extensiones y módulos compilados, por ejemplo en lenguaje C, para funciones o módulos de software críticos que requieran todo el potencial del hardware subyacente [31] [36]. Es decir, existe la posibilidad de por ejemplo escribir módulos en el lenguaje C y luego llamar a estas bibliotecas desde Python, integrando así las funcionalidades ofrecidas.

Varios autores, por ejemplo Oliphant [36], destacan a Python como un lenguaje apropiado para el entorno científico y actividad de investigación. Este fue otro punto que favoreció la elección de este lenguaje de programación.

6.4.1. Implementación de CoAP

Una de las funcionalidades a brindar por parte de la API a desarrollar es la comunicación con los nodos (Medición/Comando). Tal como se explicó en el Capítulo 4 el protocolo de capa de aplicación utilizado para esto es CoAP.

Esto significa que un punto clave para confirmar la elección de Python consiste en la evaluación de las distintas implementaciones existentes de CoAP para este lenguaje. Entre las diferentes opciones en distintos lenguajes, se destacan dos opciones en Python:

- *txThings*: Basada en Twisted, framework para programación orientada a eventos. Orientada a Python 2.
- *aiocoap*: Implementación basada en la anterior (txThings), que utiliza un nuevo módulo llamado *asyncio* para I/O asíncrona, introducido en Python 3.4.

Como el diseño de software no requiere soporte de componentes que necesiten compatibilidad hacia atrás con Python 2, la utilización de los nuevos componentes de Python 3 que marcarán la tendencia futura resultó particularmente atractiva. Además, dado que *aiocoap* cuenta con una mejor documentación en su web oficial, se definió su elección como implementación de CoAP a utilizar en el presente trabajo.

La elección de trabajar con Python 3.4, utilizando el módulo *asyncio* marca cuáles serán las posibilidades para el manejo de código concurrente en las que se enfocará el diseño de la API. Esto implica que al implementar un algoritmo de optimización junto a estrategias de monitoreo y control, tarea que naturalmente traerá el requerimiento de ejecutar varias acciones simultáneamente, se contará principalmente con dos herramientas: *Threads* y *Corrutinas*.

En el Apéndice C se brinda un breve resumen acerca del manejo de la concurrencia en Python utilizando las mencionadas herramientas. Si bien no se profundiza al respecto en esta sección, cabe aclarar que si una corrutina es llamada directamente no se obtiene ningún resultado, sino un objeto de tipo *generador* sobre el que hay que iterar para obtener los resultados. Quien se encarga de realizar esta iteración, además de coordinar la ejecución de las corrutinas, es una entidad llamada *Event Loop*. Cada thread tiene un event loop asociado y en el Capítulo 7 se muestra detalladamente un ejemplo de cómo agendar las corrutinas en el event loop correspondiente.

6.5. Diseño e implementación de la API

En esta sección se profundiza en el diseño de los módulos de software de la API desarrollada, que se describió en la sección 3.4.1, detallando las distintas funciones y clases que estos proporcionan al desarrollador describiendo los elementos más relevantes de la implementación. En el Apéndice C se realiza una descripción más detalladas de los elementos de la API.

En la Fig. 6.1 se presenta la estructura general del HEC, mostrando los módulos desarrollados.

6.5.1. Comunicación con la HAN

El módulo de comunicación con la HAN, denominado *wsncommunication*, ofrece las funciones básicas necesarias para obtener las mediciones desde los nodos y enviar comandos hacia ellos. Adicionalmente ofrece funciones que permiten conocer cuáles son los nodos accesibles en la red HAN, observar su estado y definir sus atributos. Para ello se definen ciertas funciones globales del módulo y un conjunto de clases que representan los nodos del sistema y la lista de nodos accesibles; SgNode y SgNList respectivamente.

Se incluyen funciones globales para las siguientes tareas:

- Localizar las direcciones IP accesibles en la red.
- Mantener un historial de todas las medidas reportadas.
- Obtener las Mediciones realizadas por un nodo.
- Enviar comandos.

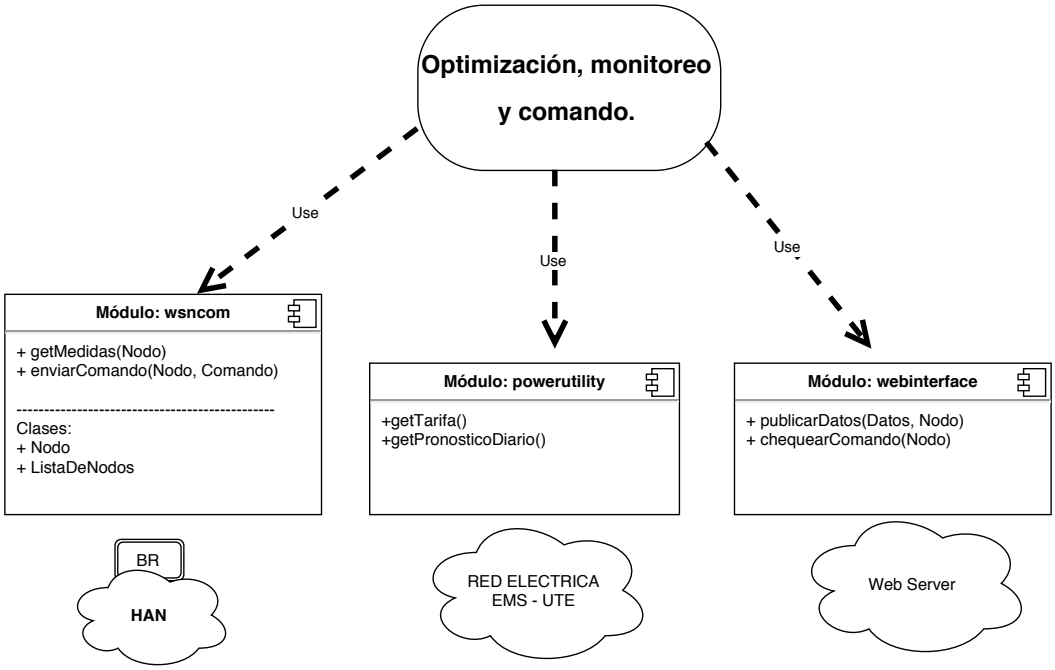


Figura 6.1: Diagrama general del HEC.

Búsqueda de nodos accesibles: función getReachable()

Esta función realiza una verificación de los nodos de la red, devolviendo una lista con las direcciones accesibles.

La estrategia de búsqueda de nodos utiliza el módulo Python *os* para acceder al comando *ping6* del sistema operativo. Este enfoque no es el óptimo, estando esta funcionalidad en el listado de las mejoras futuras del sistema. La implementación de otras estrategias (e.g. implementación de un servidor CoAP en el Border Router que devuelva el listado de nodos) es posible y mejorarían la performance del sistema en escenarios con varios nodos conectados, sobre todo si existe la necesidad de una frecuente actualización de la lista de nodos. Sin embargo, el diseño actual encapsula esta tarea dentro de una única función, por lo cual un cambio en esta implementación es posible sin la necesidad de realizar modificaciones en otras partes del código.

Esta forma de implementación de la búsqueda de nodos hace que en esta versión del sistema sea necesario asumir conocido el prefijo de la subred IPv6, y que las direcciones de los nodos utilizados sean las más bajas del rango. Sin mayor pérdida de generalidad se eligió “aaaa::c30c:0:0:0/64” como prefijo, de acuerdo a los rangos de direcciones definidos en el Capítulo 4.

Para realizar la búsqueda se recorren todas las direcciones válidas dentro de la subred, enviando 3 mensajes ping. Si el nodo responde a los mensajes se agrega su dirección en la lista a devolver.

Historial de medidas reportadas: función writeMeasureLog(data, nodeID)

Para facilitar el acceso a un historial con todas las medidas reportadas se mantiene un archivo local y se define la función *writeMeasureLog*. Esta función escribe entradas en el log de medidas, agregando una marca de tiempo correspondiente al momento en que se registran las mismas. Para ello recibe como argumentos una lista con los datos a escribir y una identificación del nodo en un string llamado *nodeID*.

Recepción de mediciones de los nodos: corrutina getMeasure(node)

Esta corrutina implementa una de las principales funcionalidades del módulo y del sistema, que es la obtención de las mediciones reportadas por un nodo. Recibe como parámetro un objeto de la clase *SgNode* que encapsula la información relacionada a un nodo, y que se describe más adelante en este capítulo.

Luego de recibida, la respuesta es procesada y se escribe una entrada en el log de medidas utilizando *writeMeasureLog()*.

Asociada a la corrutina *getMeasure(node)* se definió la función auxiliar *call_get Measure(node)* que permite acceder a la medida de un nodo sin necesidad de tener en cuenta como llamar adecuadamente a las corrutinas. Internamente se accede al event loop y se agenda la corrutina.

Comandos hacia los nodos: corrutina putCommand(node, comm)

La corrutina *putCommand(node, comm)* permite enviar comandos a los nodos. Recibe como parámetro un objeto del tipo *SgNode* representando al nodo que se quiere comandar y un string con el comando a enviar a través de CoAP.

Para implementarla también se utilizan los servicios ofrecidos por el módulo *aiocoap*. Se abre una conexión, se configura el mensaje a enviar y luego se queda a la espera de que el mismo sea recibido.

En caso de que el mensaje sea recibido satisfactoriamente, se debe actualizar el atributo del nodo que representa a su estado para registrar el estado de funcionamiento.

También se define una función auxiliar, llamada *call_putCommand(node, comm)*, para permitir enviar comandos sin necesidad de preocuparse por manipular adecuadamente el event loop.

Representación de nodos y listas de nodos

Para sintetizar toda la información relacionada a un nodo incluyendo su estado, atributos y capacidades se diseña una clase llamada "SgNode". Los objetos de esta clase son capaces de almacenar la siguiente información:

- *Nombre*
- *Prioridad*: Indicador de posibilidad/conveniencia de actuar sobre este dispositivo. Ya sea por motivos de confort del usuario como por posibilidades técnicas de los electrodomésticos.

- *Tipo*: Parámetro que indica el tipo de nodo en cuanto a sus capacidades (Medición, Comando, Medición y Comando, etc.).
- *Dirección IP*
- *Última vez que fue visto*
- *Última medida reportada*
- *Estado del nodo*: El objeto de la clase SGNode debe recordar el estado asociado a su nodo, indicando si está encendido o apagado.

Además de representar las características de los nodos conectados, el diseño del módulo contempla la necesidad de obtener un fácil acceso al listado de nodos conectados al sistema. Para ello se diseña la clase SgNList que representa una lista de todos los nodos del sistema. Ofrece también funcionalidades para actualizar la lista, buscar nodos para fijar sus propiedades y operar sobre ellos.

Esta clase cuenta con los siguientes miembros, que representan el estado de la lista:

- *Lista de nodos*
- *Ultima actualización*
- *Dirección IP del border router*

Además, las listas de nodos cuentan con los siguientes métodos que permiten realizar actualizaciones y buscar nodos:

- *update()*: Este método permite actualizar la lista, buscando nuevos nodos que pueden haberse incorporado a la red recientemente. Dentro de este método, se usa de forma auxiliar la función global del módulo *getReachable()* para consultar qué nodos son accesibles.

  Una forma de independizar los detalles de cómo se realizan ciertas funcionalidades en el software, para que eventuales modificaciones no impacten sobre el código que utiliza esas bibliotecas de funciones, es ocultar la implementación (information hidding [4]) y definir claramente la forma de acceder a los datos y funciones. Para poder ocultar la implementación particular del acceso a los nodos, y la forma de detectarlos, se diseña de modo tal que el usuario cree una lista SgNList, y luego llame al método update cada vez que requiera actualizarla.

- *clean()*: Realiza una limpieza de la lista, eliminando los nodos de los que no se registra actividad reciente. El parámetro a observar es el atributo que indica la última detección de los nodos, y el criterio está basado en un parámetro que se le puede pasar a la función, correspondiente a la cantidad de segundos de antigüedad máxima admitida. Por defecto se fijó este parámetro en el equivalente a 1 día.

[4] http://en.wikipedia.org/wiki/Information_hiding, Mayo 2015.

- *findNode()*: Esta función busca a un nodo con determinada dirección IP (incluída como parámetro en la llamada) dentro de la lista, modifica su nombre y tipo según lo especificado al llamarla, y devuelve una referencia al nodo encontrado.

  Al momento de configurar la red y querer asignar parámetros y características a cada nodo, es necesario poder verificar que los mismos se encuentran presentes y accesibles. En caso afirmativo se modifica los atributos del nodo localizado, de acuerdo a lo especificado al llamar al método, y se devuelve una referencia al nodo correspondiente.

  Si ningún nodo de la lista tiene la dirección IP buscada, se realiza una actualización de la misma y se vuelve a buscar. Para manejar la situación en la que un nodo no es localizado se lanza una excepción del tipo *NodeNotFoundError*, clase que se describe más adelante en este capítulo. Así el usuario que llama a esta función tiene una forma de atender esta situación usando el manejo estándar de excepciones de Python.

6.5.2. Interfaz web de usuario

Para poder interactuar con los usuarios del sistema, el HEC debe ofrecer una interfaz. Se propone la implementación de una interfaz web basada fundamentalmente en herramientas open source gratuitas, que permiten disminuir los tiempos de desarrollo y costos asociados.

Si bien se utilizan algunos servicios secundarios que no son completamente open-source, el núcleo central de la interfaz sí lo es, estando todo el código accesible y disponible para instalarse en un servidor propio. En esta primer implementación particular, y por motivos de agilizar el desarrollo, la interfaz web se encuentra alojada en servidores de terceros que ofrecen el servicio de hosting. Esto implica que si el usuario pierde conexión a internet no podrá interactuar con la interfaz, pero por otra parte ofrece la posibilidad de comando remoto de los dispositivos desde cualquier lugar del mundo, siendo esto una funcionalidad particularmente atractiva para los usuarios.

La interfaz web está desarrollada en base a un componente central open-source que mantiene la base de datos, y además se utilizaron otras dos plataformas web que ofrecen servicios gratuitos. En las siguientes subsecciones se describirán brevemente las tres herramientas utilizadas, para luego presentar las funcionalidades del módulo de software desarrollado para interactuar con la interfaz web.

EmonCMS

EmonCMS[5] es una aplicación web open source que permite procesar, almacenar y visualizar información acerca de consumo eléctrico. La plataforma es parte del proyecto OpenEnergyMonitor[6] y constituye el núcleo de la interfaz web ya que

[5] https://github.com/emoncms/emoncms/blob/master/readme.md, Mayo 2015.
[6] http://openenergymonitor.org/emon/, Mayo 2015.

es aquí donde se almacenan las diferentes variables de interés, se mantiene un historial de mediciones, y se muestran al usuario mediante paneles gráficos llamados *dashboards*.

Para la implementación de este componente de la interfaz de usuario se utilizaron algunos resultados del trabajo realizado por Nakasone, Seré y Modernell, [35].

Esta plataforma permite definir diferentes variables sobre las cuales se pueden escribir o actualizar datos en forma periódica. Los datos son guardados por el servidor de la aplicación y quedan disponibles para ser consultados por el usuario.

Estas variables que almacenan series de datos pueden llevar el registro de un valor tal como es recibido, o pueden también procesar la información realizando operaciones sobre los datos antes de almacenarlos. Por ejemplo, se podría tener una variable que almacene el valor de la potencia aparente en base a los valores recibidos de "Tensión Vrms" y "Corriente RMS" procesando estos valores mediante la operación $Vrms.Irms$ y almacenando el resultado.

Para cada nodo conectado al sistema se definieron las siguientes variables que almacenan los datos correspondientes:

- Vrms *(valor de tensión)*
- Irms *(valor de corriente)*
- P *(valor de potencia activa)*
- Q *(valor de potencia reactiva)*
- S *(valor de potencia aparente)*
- FP *(valor del factor de potencia)*
- Relay_Status *(representación de status de los relays)*

EmonCMS brinda interfaces de programación con comandos básicos a través de los cuales un servidor o cliente remoto puede enviar información y leer o modificar valores de las variables almacenadas. Al crear una cuenta en EmonCMS el usuario (programador) obtiene una URL y un conjunto de claves denominadas *API Keys* que conforman las rutas de acceso de información para realizar este tipo de intercambios.

Para intercambiar datos con la interfaz se debe acceder al servidor a través de HTTP enviando comandos con un formato predefinido por las API mencionadas e incluyendo la *API Key* de seguridad. Existen diferentes formatos en los que los comandos y la información intercambiada puede ser representada. En el presente trabajo se decidió utilizar comandos en formato *JSON*[7]. *JSON* es un formato ligero para el intercambio de información que tiene la ventaja de ser fácilmente legible por humanos. Se trata de un formato de texto basado en JavaScript, pero que utiliza convenciones simples que resultan familiares en casi cualquier lenguaje de programación.

[7]http://json.org/, Mayo 2015.

Figura 6.2: Capturas de pantalla de algunos Dashboards en EmonCMS.

En base a este formato se utilizaron dos comandos básicos; uno para escribir valores (actualización de variables desde el HEC) y otro para leer valores almacenados en la base de datos de la interfaz. La lectura de valores se implementó para verificar si existen solicitudes de comando de electrodomésticos por parte del usuario. Estas instrucciones de lectura y escritura de variables son utilizadas por el software del HEC para el intercambio de información con la plataforma que maneja la interfaz web.

Para facilitar la lectura de la información por parte del usuario se configuraron algunos paneles gráficos, y en la Fig. 6.2 se muestran algunas capturas de pantalla de los mismos.

Weebly y Freeboard

Se utilizaron las herramientas proporcionadas por Weebly [8] y Freeboard [9] para proporcionar funcionalidades complementarias como son la posibilidad de tener una página web principal con menús de acceso a las distintas funcionalidades de la plataforma, y además para permitir el uso de botones de comando remoto de los electrodomésticos.

Freeboard es una plataforma online para el desarrollo y utilización de paneles gráficos de visualización de información. En su versión más básica, que permite únicamente desarrollos abiertos, la plataforma es gratuita.

Esta plataforma permite mostrar información mediante gráficos animados y diagramas de tendencia. Soporta el formato *JSON* para el intercambio de infor-

[8] http://www.weebly.com, Mayo 2015.
[9] https://freeboard.io/, Mayo 2015.

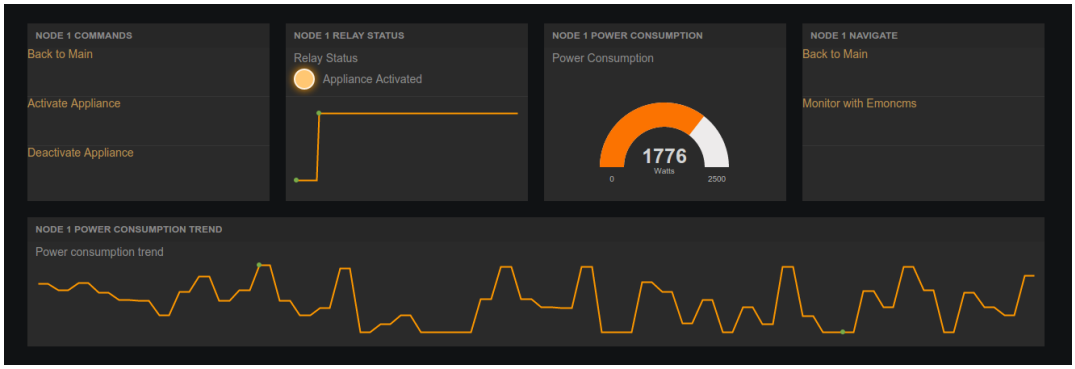


Figura 6.3: Capturas de pantalla de un Dashboard en Freeboard.

mación con servidores o clientes externos a través de HTTP, y por lo tanto se puede integrar fácilmente con la base de datos de EmonCMS.

Una de las ventajas que ofrece Freeboard es que sus interfaces gráficas están optimizadas para Smart Phones. Por este motivo se decidió utilizar esta herramienta para complementar las opciones de visualización ofrecidas por EmonCMS, brindando una versión más sencilla y adecuada a dispositivos móviles, y fundamentalmente incluyendo una interfaz con comandos para actuar manualmente sobre los nodos y por ende sobre los electrodomésticos conectados.

Para lograr la funcionalidad de comando se utilizan botones que brindan acceso a URLs compuestas por el comando de escritura en variables de la base de datos de EmonCMS. Este valor puede ser consultado por el HEC para determinar si el usuario solicitó actuar sobre algún dispositivo, en función de la estrategia de monitoreo y comando implementada.

En la Fig. 6.3 se muestra una captura de pantalla del panel de monitoreo y comando para uno de los nodos. El panel gráfico incluye un indicador de potencia actual, una gráfica de tendencia de la potencia y del estado de la carga (ON/OFF), y lo más importante, un link de comando para activar y desactivar el nodo, encendiendo o apagando el electrodoméstico.

Para tener una presentación gráfica de la interfaz más amigable con el usuario se utilizaron los servicios gratuitos de *Weebly*. Mediante esta herramienta se proporciona una página principal desde la que se puede obtener acceso a los datos del proyecto y los diferentes menús, incluyendo medición y comando, con los links correspondientes a las plataformas y paneles gráficos antes mencionados.

En la Fig. 6.4 se muestran algunas capturas de pantalla de esta interfaz de inicio.

Interacción entre el HEC y la interfaz web: módulo webinterface

Para facilitar la comunicación entre el HEC y la interfaz web previamente descrita se diseña el módulo de la API llamado *webinterface*. Este módulo de manejo de la interfaz web proporciona funciones globales que implementan las siguientes funcionalidades:

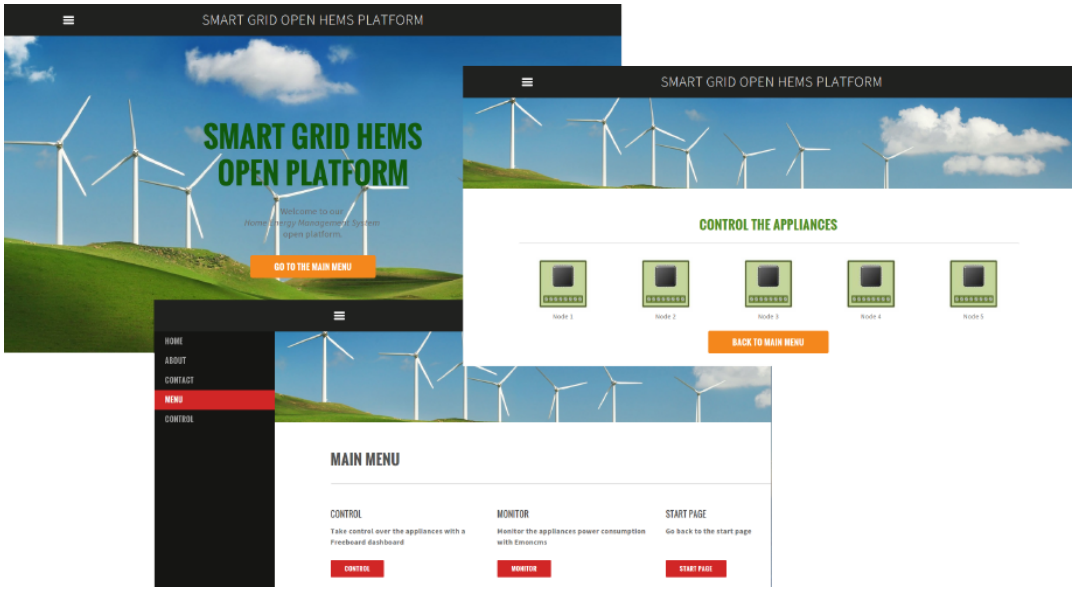


Figura 6.4: Capturas de pantalla del menú principal de la interfaz de usuario.

- *Publicar datos en la web:* Mediante una función global se permite publicar en la interfaz web las medidas reportadas por un nodo. Para ello simplemente se abre una conexión HTTP con el servidor web, para luego hacer un GET con una cadena de texto que contiene los datos en formato JSON.
- *Chequear si un usuario solicita accionar un dispositivo:* Para simplificar el diseño, y permitir al HEC un control total y centralizado de todos los dispositivos sin necesidad de atender directamente las solicitudes del usuario, se diseña de modo que este último interactúe con la interfaz web. Es luego el HEC quien define el momento de consulta a esta base de datos, según los criterios de diseño, para actuar o no de acuerdo a lo solicitado por el usuario. [10]
- *Mantener un historial de los datos publicados:* Se cuenta con una función que escribe una entrada en el log de la interfaz, agregando una marca de tiempo correspondiente al momento en que se hace el registro. El log se basa en un archivo local, y sirve para mantener un historial de toda la información que fue enviada al servidor web.

6.5.3. Interacción con el EMS de la empresa de suministro de energía

Si bien no existe actualmente en el Uruguay un EMS capaz de proveer información de tarifa dinámica (RTP), el módulo *powerutilitycom* diseñado simula la

[10] En el capítulo 7, que presenta el diseño de un sistema básico pero completo de monitoreo, comando y optimización, se presenta un ejemplo donde el controlador chequea constantemente si el usuario requiere realizar alguna acción sobre los dispositivos.

interacción con la empresa de suministro de energía eléctrica. Para ello se mantiene un archivo local en el HEC con información del último pronóstico de tarifa horaria para todo el día, y se ofrece la posibilidad de actualizar este archivo de forma remota, simulando la conexión con el EMS.

Este módulo implementa funciones para proporcionar las siguientes funcionalidades:

- *Pronóstico diario del precio de la energía*: Obtiene el pronóstico de tarifa horaria para todo el día, accediendo al archivo local almacenado en el HEC.
- *Consulta del precio real para la hora actual*: Proporciona el precio real actual de la energía mediante una simulación. Para ello se adiciona una señal aleatoria gaussiana al precio estimado para la hora correspondiente.
- *Actualización del pronóstico de precio diario*: Permite actualizar de forma remota el pronóstico diario para el precio de la energía eléctrica mediante conexión con un servidor web.

6.5.4. Manejo de errores

Para menejar los errores en tiempo de ejecución se utiliza el sistema de excepciones de Python. Este sistema permite cambiar el flujo de ejecución de un programa ante la ocurrencia de un error o situación anómala, proporcionando expresiones estándar para manejar estas situaciones. El libro de Lutz [34] es una buena referencia en caso de que el lector quiera consultar los conceptos básicos asociados a las excepciones en programación.

Para definir tipos de errores específicos, que representen situaciones anómalas en la ejecución del código de optimización que utiliza la API diseñada, se crearon clases que heredan de *Exception*. De esta forma el usuario de la API puede realizar las llamadas a las distintas funciones dentro de bloques *try-except* y así manejar de forma conveniente los errores en tiempo de ejecución. (Por ejemplo, en caso de querer definir parámetros de un nodo que no se encuentra accesible.)

Las clases definidas guardan información acerca del contexto en el cual se produjo el error, tanto de la acción que se estaba realizando como de un posible error previo responsable de producir la excepción.

- *Errores en el acceso a la interfaz web: clase InternetError*: Se definió esta clase en el módulo de comunicación con la interfaz web para representar errores al momento de intentar acceder a la misma. Este tipo de excepción es lanzada cuando se produce un error al intentar escribir un dato en la interfaz, o al chequear un comando de usuario.
- *Errores de comunicación a nivel de CoAP: clase CoapError*: Con esta clase, definida en el módulo de comunicación con la HAN, se representan los errores ocurridos al momento de intentar acceder a un nodo a través de CoAP. Ya sea al solicitar un reporte de medición o al intentar enviar un comando para actuar sobre un dispositivo.

- *Errores en el acceso a la HAN: clase HanAccessError*: Esta clase, también definida en el módulo de comunicación con la HAN, representa un error al intentar acceder a algúna dirección IP para verificar conectividad. Está pensada en base a la implementación actual de la búsqueda de nodos activos en la red y en este caso se tiene como atributo, además del error previo, la dirección IP a la que se estaba intentando acceder al momento del error.
- *Error de nodo no localizado: clase NodeNotFoundError*: Se definió esta clase en el módulo de comunicación con la HAN, para apoyar la funcionalidad proporcionada por el método *findNode()*. Con este tipo de excepción se representa la situación en que determinado nodo que estaba siendo buscado no se encuentra presente en la lista actual de nodos. Quien llama a la función *findNode()* debe capturar y atender este tipo de excepciones.

6.6. Síntesis del capítulo

A partir de los requerimientos preliminares introducidos en el Capítulo 3, se presentó en este capítulo el diseño de la arquitectura general del software del HEC, definiendo los distintos módulos y funcionalidades que lo componen.

Luego de fundamentar la elección de Python como lenguaje de programación y Raspberry Pi como plataforma de hardware, se desarrollaron los detalles de diseño de cada uno de los módulos definidos en la API, incluyendo la interfaz de usuario que fué implementada en base a servicios web de la plataforma open source EmonCMS.

Al finalizar este capítulo el lector conoce la implementación del HEC y la API brindada por el mismo. En base a esto y a lo explicado en el Capítulo 7 que presenta un ejemplo práctico de aplicación de la API, debería ser capaz de utilizar el HEC y la plataforma HEMS para el ensayo y ejecución de algoritmos de optimización desarrollados en Python.

Capítulo 7

Ejemplo de aplicación de la plataforma HEMS

7.1. Introducción

Contando con módulos de hardware diseñados para interactuar con los electrodomésticos (nodos), un sistema de comunicación (HAN), un controlado (HEC) y una API de software para desarrollar estrategias de control, se implementa un módulo de software que integra todo el sistema y le da sentido a la plataforma HEMS.

El principal objetivo del presente capítulo es brindar al lector un ejemplo concreto del uso de la API desarrollada para lograr una plataforma de monitoreo, comando de dispositivos y optimización en base a un algoritmo específico.

7.2. Implementación y ensayos HEC

El objetivo de esta sección es poner a prueba la plataforma HEMS implementada, y en particular el controlador HEC, con la API desarrollada y detallada en el Capítulo 6. Para ello se utilizan los servicios proporcionados por la API como base para implementar una aplicación completa de optimización, monitoreo y comando de dispositivos, incluyendo las funcionalidades de interfaz de usuario web.

A los efectos de estos ensayos se implementará un algoritmo para optimización de termotanque, que servirá como ejemplo para evaluar las capacidades de la plataforma.

El software del HEC deberá realizar continuamente y de forma simultánea las siguientes tres tareas:

- *Optimización*: Comandar el estado del termotanque de acuerdo a lo determinado por el algoritmo de optimización.
- *Medición*: Actualizar la base de datos de la interfaz web con los reportes de mediciones de todos los nodos conectados.

- *Comando*: Chequear si el usuario del sistema solicita encender o apagar un dispositivo en forma manual remota desde la interfaz web, y llevar a cabo la acción solicitada.

Para implementar estas tareas se emplearán las funcionalidades de threading y programación asíncrona de Python, utilizando varios hilos de ejecución y corrutinas.

El módulo principal del HEC, llamado *controladorSG2015*, define las siguientes tres corrutinas que luego serán examinadas en detalle:

- *calefonLoop()*: Se encarga de ejecutar la tarea de comandar el termotanque para optimizar su consumo eléctrico.
- *loopMedidas()*: Es responsable de mantener actualizada la base de datos con las mediciones.
- *checkCommand()*: Realiza consultas a la interfaz web para saber si el usuario requiere encender o apagar un dispositivo, y ejecuta el comando correspondiente.

La función *main* del mencionado módulo principal (*controladorSG2015*) crea dos hilos de ejecución (threads), uno para chequear los comandos con la corrutina *checkCommand()* y otro para el loop de medidas implementado con la corrutina *loopMedidas()*. Posteriormente se verifica que el termotanque esté presente en la lista de nodos, y en caso afirmativo se crea un tercer thread para comenzar a ejecutar el algoritmo de optimización.

Para el manejo de los nodos conectados al sistema se utiliza una variable global de la clase SgNList, como se describe en C.3.3 y cuyos detalles se pueden consultar en el Apéndice C.

```
1 list = wsncommunication.SgNList()
```

A continuación se explica cómo están implementadas las tres corrutinas principales, para luego volver sobre la función *main* y examinar cómo son agendadas las corrutinas en cada hilo de ejecución.

7.2.1. Optimización del consumo: corrutina calefonLoop()

Para realizar la optimización de consumo energético del termotanque se implementó un módulo llamado *controlcalefon*. En este módulo se implementa la primer etapa del algoritmo desarrollado por Du, Pengwei y Lu, Ning [22].

El primer punto a destacar de la implementación de la tarea de optimización, es que será programada como una corrutina. Esto implica que en los momentos en que la tarea se encuentre a la espera de completar una comunicación por la red, no bloqueará el thread que la esté ejecutando. Por otra parte, para iniciar la tarea es necesario agendarla en el scheduler correspondiente al thread. Estas funcionalidades son ofrecidas por el módulo *asyncio*.

Para definir la corrutina, se utiliza el decorador *@asyncio.coroutine*[1]:

```
@asyncio.coroutine
def checkCommandLoop():
```

En la primer línea de la corrutina se utiliza el módulo *controlcalefon* para obtener una lista del estado en que debe estar el calefón en cada minuto del día.

```
comandosOpt = controlCalefon.run()
```

Debido a que solo se implementa parcialmente el algoritmo de optimización de Du, Pengwei y Lu, Ning [22], se puede apreciar como el control resulta ser en lazo abierto, sin tener ninguna realimentación respecto a lo que realmente sucede con el consumo de agua caliente. Este enfoque es suficiente para el objetivo de la aplicación, que está diseñada para probar la API y verificar que el HEC ejecuta los comandos cuando son requeridos.

Luego la corrutina entra en un loop infinito dentro del cual, para cada minuto, se enciende o apaga el calefón en función del estado indicado por la lista de estados óptimos del termotanque. Para ejecutar el comando indicado en *comandosOpt* se tiene en cuenta además del estado actual del termotanque, el último comando recibido desde la interfaz web de usuario.

El criterio de prioridad utilizado entre los comandos del usuario (a través de la interfaz web) y el loop de optimización, fue el siguiente: si un usuario indica que el termotanque debe estar encendido se acepta esta decisión y se ignora lo estipulado por el algoritmo de control, y si el usuario indica que el termotanque debe estar apagado se utiliza el algoritmo de optimización. Este criterio es arbitrario, y pueden implementarse estrategias de optimización con prioridades diferentes.

Para cada recorrido del loop, una vez determinado si se debe actuar sobre el termotanque, se llama a la corrutina correspondiente para enviar el comando. Por ejemplo:

```
if (comandosOpt[min]==0) and
        (calefon._status == 1) and
         (lastCheckCalefon==0):
    yield from wsncommunication.putCommand(
            calefon,"comando_relays=0")
```

Implementación del algoritmo de optimización: módulo controlcalefon

Para la optimización del consumo de termotanque se implementó parcialmente el algoritmo propuesto por Pengwei Du en [22]. El algoritmo tiene como objetivo construir un vector de estados de control del termotanque (ON-OFF) en función de los parámetros de su modelado físico, el consumo de agua caliente pronosticado e información RTP (Real Time Pricing). La optimización queda sujeta a restricciones de confort, determinadas por el usuario a través de la definición de una banda de temperatura (Tmin, Tmax).

El termotanque se caracteriza por los siguientes parámetros: C (capacitancia térmica), R (resistencia térmica), Q (Potencia kW) y V (volumen en lts). La estimación de la temperatura de un termotanque es estudiada por Liam Paull, et.

[1] Por mayores detalles acerca de la definición de corrutina se puede consultar el Apéndice C.1.2

al en [38], y está dada por los parámetros anteriormente descritos, la temperatura ambiente T_{amb}, la temperatura de entrada del agua fría T_{af}, el flujo de agua que atraviesa el termotanque W_D, el calor específico del agua c_p y la superficie del termotanque A, a través de las siguientes ecuaciones:

$$C = \rho.c_p V, \quad H = \rho.c_p, \quad G = A/R \tag{7.1}$$

$$\tau = \frac{C}{G + H.WD} \tag{7.2}$$

$$K = \frac{G.T_{amb} + H.W_D.T_{af} + Q}{G + H.W_D} \tag{7.3}$$

$$T(t) = T_{ini}.e^{-t/\tau} + K(1 - e^{-t/\tau}) \tag{7.4}$$

El algoritmo estima (o setea) en primer lugar el volumen de agua caliente a ser utilizado en el hogar para las siguientes 24hs. Esto fija la cantidad de energía necesaria para poder llevar esta masa de agua desde la temperatura de agua fría a la temperatura deseada. Luego, en función de los distintos parámetros se determina la cantidad de horas t_{ON} que debe estar encendido el termotanque para transferir esa energía.

En un segundo lugar se toma la función de costo $F(t)$ provista por el EMS o un pronóstico y se reordena esta función de forma creciente, en adelante $F_c(t)$. Se define un precio de referencia P_r dado por $P_r = F_c(t_{ON})$.

El vector de control de termotanque, u se establece de la siguiente manera:

$$\begin{cases} u(t) = 1 \text{ si } F(t) < P_r \\ u(t) = 0 \text{ en otro caso.} \end{cases} \tag{7.5}$$

Sin embargo deben realizarse ajustes al vector u de forma de mantener la temperatura del termotanque en la zona de confort definida por el usuario. En momentos donde la temperatura se encuentra fuera de esta banda, se implementa un mecanismo de saturación mediante una secuencia particular en el vector u dada por el modelo del termotanque.

La segunda etapa del algoritmo no fue implementada. Consiste en el ajuste en tiempo real del vector de control u, en función del pronóstico actualizado de consumo de agua caliente y de la nueva información RTP. En la Fig. 7.1 se muestra el funcionamiento del sistema, con la simulación del consumo de agua caliente en un termotanque.

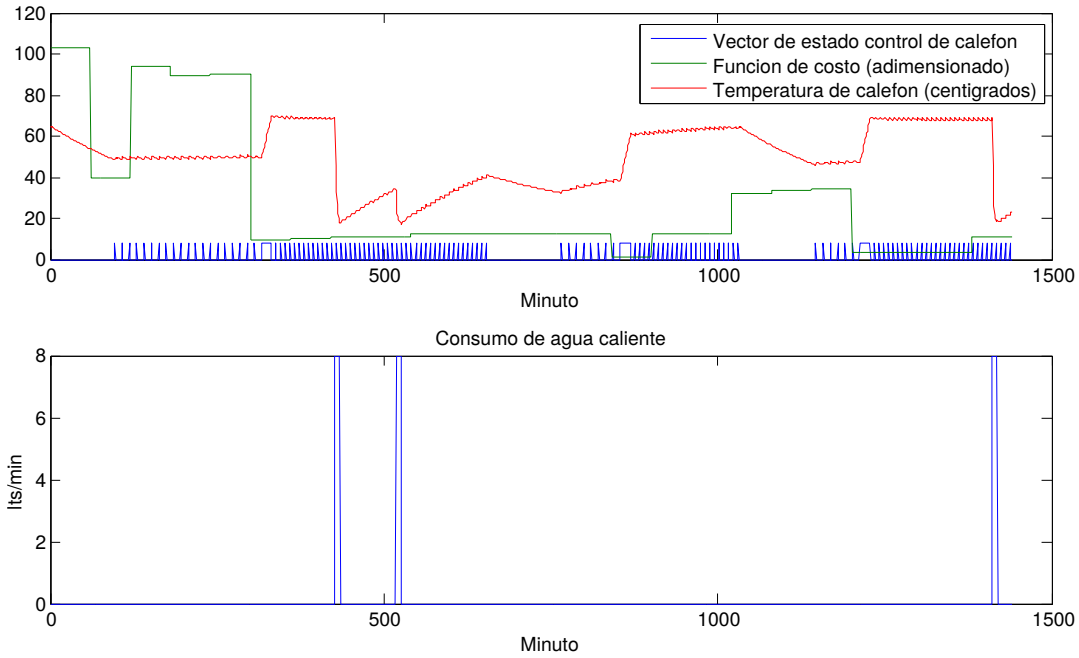


Figura 7.1: Simulación de consumo de agua caliente, algoritmo y temperatura del termotanque. Los parámetros del calefón simulado son $Qc = 1500W$, $C = 5{,}67 \times 10^4$ y $G = 4{,}5$.

7.2.2. Reporte de mediciones: corrutina loopMedidas()

Esta corrutina inicia un loop de medidas para todos los nodos conectados al sistema. Recorre la lista de nodos (que son objetos de la clase SgNode 6.5.1), y para todos los que tengan capacidad de medición se solicita la medida actual:

```
1 med = yield from wsncommunication.getMeasure(n)
```

Posteriormente se publica en el servidor web la medida del nodo usando la función *webinterface.pushData(n.lastMeasure,n)*.

7.2.3. Comando remoto de dispositivos: corrutina checkCommand()

La corrutina *checkCommand()* inicia un loop en el que se consulta para cada dispositivo si el usuario intenta actuar sobre él. Esta verificación de los comandos se realiza con la función *webinterface.checkCommand(node)*, para todos los nodos con capacidad de ser comandados. Una vez evaluado el comando recibido desde la interfaz web, se actúa sobre el nodo usando la corrutina correspondiente, por ejemplo:

```
1 yield from wsncommunication.putCommand(node,"comando_ON/OFF")
```

En caso de que el parámetro *comando_ON/OFF* recibido desde la interfaz web corresponda al comando “Apagar”, antes de enviarlo al nodo correspondiente utilizando esta corrutina se verifica que el dispositivo asociado no sea el termotanque

(cuyo consumo se está optimizando) de modo de respetar el criterio definido en la Sec. 7.2.1.

La corrutina de optimización debe conocer cuál fue el último comando solicitado por el usuario para el termotanque, y así saber si debe hacer optimización automática o si el usuario quiere dejarlo siempre encendido. Para ello, cada vez que se consulta el último comando enviado al termotanque se actualiza una variable global que es utilizada por la corrutina de optimización para saber si el usuario quiere controlar el termotanque en modo manual.

7.2.4. Coordinación de las corrutinas: función main()

A continuación se detalla cómo se utilizaron las funcionalidades ofrecidas por los módulos de Python *threading* y *asyncio* para ejecutar simultáneamente las tres corrutinas implementadas. En el Apéndice C.1 se ofrece una breve descripción de estos módulos.

Se crearn tres hilos de ejecución (threads), contando cada uno de ellos con un *Event Loop* [2] que funcionará como scheduler de las corrutinas asociadas al thread. Para este diseño particular solamente habrá una corrutina ejecutándose en cada en cada *Event Loop*, pero podrían agregarse más en caso de que la aplicación a implementar lo requiera y así aprovechar el potencial ofrecido por las corrutinas.

Para crear los hilos de ejecución se utiliza la clase *Thread* del módulo *threading*. Entre los argumentos que se le debe pasar al constructor de esta clase está el *target*, que representa a la tarea que será invocada al iniciar el thread. Otro argumento del constructor es *args* con una *tupla* [3] conteniendo los argumentos para el target del thread.

Tres funciones auxiliares se definieron para manejar la forma de asignar un *Event Loop* a cada thread. Estas funciones reciben como argumento el *Event Loop* que asignarán a su thread, e inician la ejecución de la corrutina correspondiente. Se nombró a estas funciones auxiliares de la siguiente forma:

- *call_calefonLoop()*
- *call_loopMedidas()*
- *call_checkCommand()*

A modo de ejemplo se muestra el código de la función auxiliar *call_checkCommand()*, donde se utiliza la función *asyncio.set_event_loop(loop)* para asociar al contexto actual (thread) el loop recibido como argumento. La estructura de las otras dos funciones auxiliares es completamente análoga.

```
1 def call_checkCommandLoop(loop):
2     asyncio.set_event_loop(loop)
3     loop.run_until_complete(checkCommandLoop())
```

Volviendo a la función *main*, las siguientes líneas de código se encargan de crear el thread de los comandos web e iniciar su ejecución.

[2] ver C.1.2

[3] Las tuplas son listas inmutables de objetos, ver [24].

```
    loop = asyncio.get_event_loop()
    check = threading.Thread(target=call_checkCommandLoop,
            args=(loop,))
    check.start()
```

En este caso lo que se hizo fue acceder al *Event Loop* del contexto principal, usando *asyncio.get_event_loop()*, y pasarlo como argumento al nuevo thread creado. Luego se inicia la ejecución del thread invocando al método *start()*.

Para crear el thread encargado de realizar las mediciones y reportar a la interfaz web se sigue una estructura muy similar:

```
    loop2 = asyncio.new_event_loop()
    medidas = threading.Thread(target=call_loopMedidas,
             args=(loop2,))
    medidas.start()
```

Cabe notar que en este caso se crea un nuevo *event loop* para ser asignado al segundo thread, donde se realizarán las medidas, usando *asyncio.new_event_loop()*.

Estando ya en ejecución los threads de medición y comando a través de la web, se inicia el loop de optimización de consumo del termotanque. Previamente se verifica que en la lista de nodos se encuentra conectado el termotanque, utilizando para ello al método *findNode()* proporcionado por la clase SgNList. Se asume conocida la dirección IP del termotanque.

```
    try:
        calefon = list.findNode("Calefon","aaaa::c30c:0:0:1",
                "Measure&Command")
    except wsncommunication.NodeNotFoundError:
        pass
    else:
        loop3 = asyncio.new_event_loop()
        ctrlcalef = threading.Thread(target=call_calefonLoop,
                args=(loop3,calefon))
        ctrlcalef.start()
```

Es oportuno observar cómo en este caso, además de un nuevo *Event Loop*, se pasa como argumento una referencia al termotanque que se desea optimizar.

7.3. Síntesis del capítulo

En este capítulo se presentó una implementación completa de un sistema de optimización, comando de dispositivos y monitoreo de consumo, que hace uso de los elementos diseñados y descritos en los capítulos previos.

Se detalla la implementación de un algoritmo de control utilizando la API diseñada en el Capítulo 6, explicando con un ejemplo básico cómo debe utilizarse la biblioteca de funciones y describiendo una forma de agendar corrutinas en distintos threads para ejecutar tareas de forma concurrente.

A través de este ejemplo de aplicación se completó una descripción global de la utilización de la API desarrollada.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 8

Ensayos y evaluación del sistema

8.1. Introducción

La plataforma HEMS se compone de un HEC y nodos comunicados a través de una HAN de acuerdo a los diseños e implementaciones presentados en capítulos previos. El HEC ofrece una biblioteca de funciones en forma de API para facilitar la integración de algoritmos de control y optimización de energía eléctrica a la plataforma.

A lo largo de los capítulos de diseño e implementación de cada uno de los módulos que componen la plataforma HEMS se presentaron algunos ensayos y observaciones preliminares, sin embargo no se ha presentado aún un ensayo y evaluación general de desempeño del sistema completo. En el presente capítulo se describen entonces todos los experimentos y pruebas realizadas, y a partir de ellas las observaciones y conclusiones a las que se arribó. Se incluye también una breve evaluación respecto del algoritmo de optimización introducido como caso de aplicación en el Capítulo 7.

8.2. Ensayos y calibración de nodo

8.2.1. Placa de acondicionamiento de señales

En un primer lugar se realizaron medidas de las tensiones de continua del circuito, que se muestran a continuación:

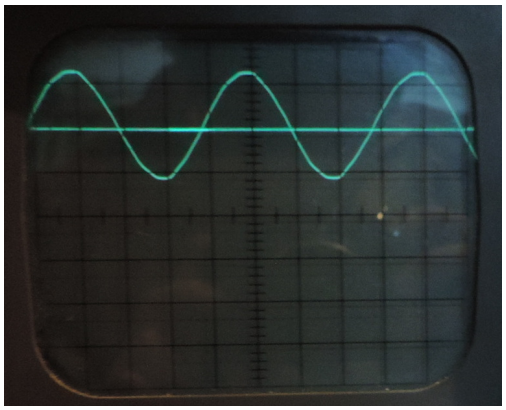
Señal	Esperado	Real
V_{CC}	8,4V	8,3V
V_{REF}	2,4V	2,4V
V_{43}	14,9V	15,2V

Tabla 8.1: Valores de continua esperados y medidos.

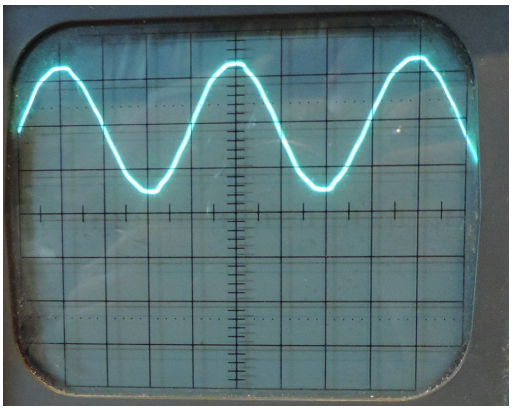
Todos estos valores se aproximan a los valores teóricos y simulados (ver la

Tabla 8.1), por lo que el circuito se comporta de la manera esperada en sus niveles de continua.

Las señales V_{fv} y V_{fi} fueron observadas en un osciloscopio mientras se tomaban lecturas de un medidor de potencia sometido al mismo voltaje de Red y a la misma intensidad de carga, como se muestra en la Fig. 8.1 y la Fig. 8.2.

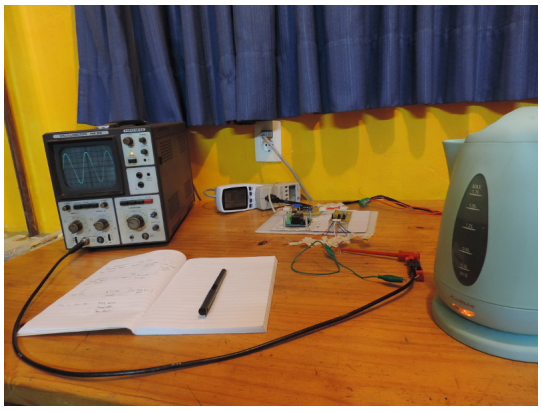


(a) Señales de voltaje y corriente, ambas montadas en Vref. Dado que no hay carga activada la amplitud de la señal de corriente es nula. Parámetros del osciloscopio: 1V div, 5ms div, modo DC.

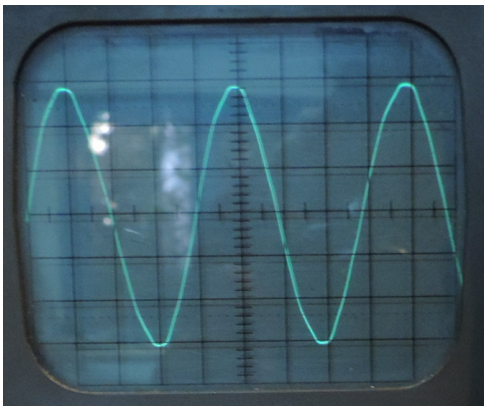


(b) La señal de voltaje tiene 2,94Vpp, el voltaje en el medidor patrón indica 330V, por lo que $G_V = 4,45 \times 10^{-3} V/V$. Parámetros del osciloscopio: 1V div, 5ms div, modo DC.

Figura 8.1: Ensayo de la señal de voltaje.



(a) Se coloca una caldera eléctrica para el ensayo de corriente de carga.



(b) La señal de corriente tiene 1,15Vpp, la corriente en el medidor patrón indica 8,78A, por lo que $G_I = 6,55 \times 10^{-2} VV$.. Parámetros del osciloscopio: 20mv div, 5ms div, modo AC.

Figura 8.2: Ensayo de la señal de corriente.

En cuanto a esta primera etapa de ganancias, la correspondiente a la señal

de voltaje se aproxima a la estimada inicialmente (con un error relativo de un 1,7 %). En el caso de la ganancia de la señal de corriente, se aprecia una diferencia significativa con respecto a lo esperado (cercana al 12 %). A la hora de analizar las razones de tal diferencia, se constató que:

Existieron errores a la hora de la implementación de la placa, a saber: se utilizaron dos resistencias (R_9 y R_{10}) al 5 % de tolerancia en la etapa de ganancia de corriente cuando en realidad se especificaron tolerancias al 1 % durante el diseño para minimizar el desapareo entre las mismas. Por otra parte, en el caso del otro par de resistencias del mismo amplificador operacional, se utilizaron componentes al 1 % pero con valores distintos a los del diseño (se montaron resistencias de $62k\Omega$ en lugar de $84,5k\Omega$ para R_2 y R_8). Esta última diferencia implica una nueva ganancia de transducción teórica de $5,37 \times 10^{-2}\Omega$, sin embargo a partir de los ensayos se obtuvo una ganancia de transducción real de $6,5 \times 10^{-2}\Omega$. Se atribuye la diferencia al posible desapareo introducido por las resistencias al 5 % y a una posible desviación en el valor de R_{shunt}, ya que no existe ninguna garantía de que su valor sea el que figura en los esquemáticos ($2,1m\Omega$), puesto que esta información no es publicada por el fabricante y el componente puede variar en los distintos procesos de producción. Además este componente no puede ser medido sin afectar el circuito en forma irreversible.

Si bien se constatan diferencias en las ganancias, esto no resulta crítico, puesto que es fácilmente ajustable al momento de la calibración de software en los propios nodos. Sin embargo, más allá de esta calibración, vale notar que una menor ganancia en la señal de corriente implica una mayor señal a ruido de cuantización (SQNR), afectando principalmente a las señales de corrientes pequeñas.

8.2.2. Placa de filtrado

El objetivo de este ensayo es comprobar que se satisfacen los criterios de diseño del filtro. Los resultados esperados son:

- Polo en $f_{-3dB} = \frac{1}{2.\pi.RC} = 4593Hz$. Considerando la incertidumbre de los componentes: resistencia al 1 % y capacitor al 5 %, se espera encontrar el polo en el rango de [4331,4883]Hz.

- Ganancia unitaria hasta los 500Hz.

En el experimento se inyectó una señal sinusoidal provista por un generador de señales de aproximadamente 1,2Vpp con un offset de 1,8V. Las señales de entrada y salida se visualizaron en el osciloscopio para relevar la respuesta en frecuencia. Los resultados se muestran en la Tabla 8.2.

f (Hz)	Vipp	Vopp	$\frac{Vopp}{Vipp}$
20	1,2	1,2	1,00
50	1,24	1,22	0,98
100	1,24	1,22	0,98
250	1,22	1,22	1,00
400	1,24	1,22	0,98
500	1,22	1,2	0,98
700	1,22	1,2	0,98
1000	1,22	1,18	0,97
2000	1,24	1,1	0,89
3000	1,2	1	0,83
4000	1,2	0,88	0,73
4500	1,2	0,84	0,70

f (Hz)	Vipp	Vopp	$\frac{Vopp}{Vipp}$
5000	1,22	0,78	0,64
5500	1,22	0,74	0,61
6000	1,2	0,7	0,58
8000	1,2	0,58	0,48
12000	1,19	0,42	0,35
20000	1,19	0,27	0,23
40000	1,2	0,14	0,12
80000	1,2	0,08	0,07
120000	1,17	0,046	0,04
300000	1,19	0,019	0,02
1000000	1,17	0,004	0,00

Tabla 8.2: Ensayo de respuesta en frecuencia.

En la Fig. 8.3 se grafica la respuesta en frecuencia del filtro. Se observa que el polo se ubica entre los 4000Hz y los 4500Hz. Para obtener una ubicación más precisa del polo se realizó una interpolación lineal entre las dos muestras inmediatas consecutivas, que arroja como resultado una frecuencia de 4380Hz. Se concluye entonces que el polo se encuentra dentro del rango esperado de [4331,4883]Hz.

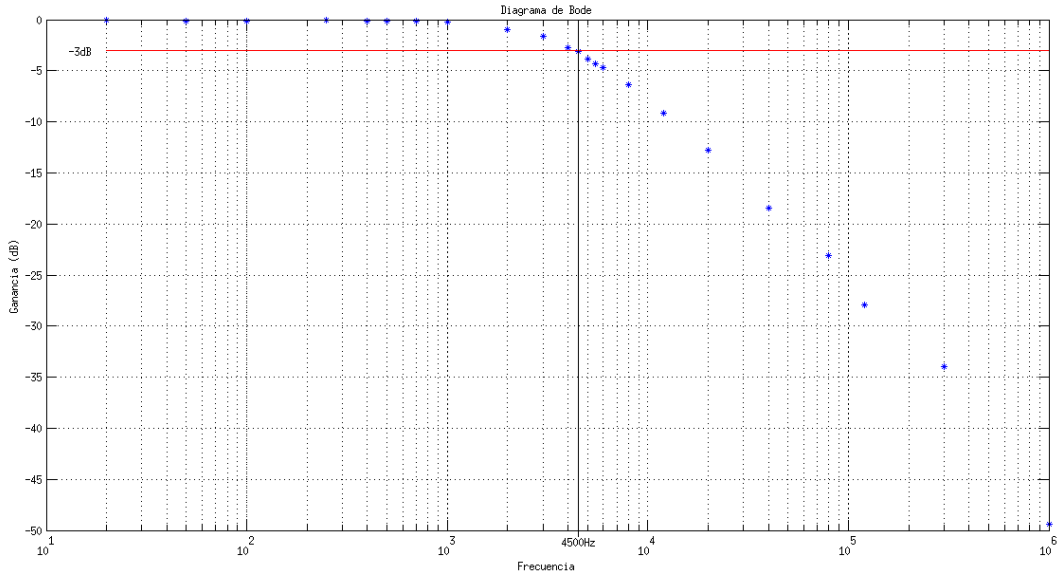


Figura 8.3: Respuesta en frecuencia del filtro.

Los resultados también muestran una respuesta plana en ganancia unitaria hasta una frecuencia cercana de 1kHz (con una desviación en la ganancia de menos del 2 %) por lo que se concluye que el filtro se comporta como era esperado.

Potencia	Descripcion	Grupo
40W	Bombilla incandescente	CALIBRACIÓN
60W	Bombilla incandescente	CALIBRACIÓN
100W	Bombilla incandescente	CALIBRACIÓN
200W	Bombilla incandescente	CALIBRACIÓN
720W	Tostadora	CALIBRACIÓN
750W	Cafetera	CALIBRACIÓN
950W	Tostadora + Bombilla	CALIBRACIÓN
2000W	Caldera eléctrica 1	CALIBRACIÓN
40W	Bombilla incandescente	ENSAYO
60W	Bombilla incandescente	ENSAYO
100W	Bombilla incandescente	ENSAYO
200W	Bombilla incandescente	ENSAYO
450W	Enceradora	ENSAYO
650W	Sandwichera	ENSAYO
850W	Sandwichera + Bombilla	ENSAYO
1850W	Caldera eléctrica 2	ENSAYO

Tabla 8.3: Lista de electrodomésticos escogidos para las etapas de calibración y ensayo.

8.2.3. Calibración y ensayo de mediciones

Calibración de mediciones

Luego de la obtención de las medidas a través del ADC, se realizan ajustes a la ganancia de señales a nivel de software. El ADC recibe 3/5 de la salida de la placa de filtrado y protección. Por lo que se tiene que:

$$V_{RMS} = \frac{5}{3.G_V}.V_{fv} = 374,1497.V_{fv} \tag{8.1}$$

$$I_{RMS} = \frac{5}{3.G_I}.V_{fi} = 25,4493\Omega^{-1}.V_{fi} \tag{8.2}$$

A partir de estas nuevas ganancias, se realizó una etapa de calibración y una de ensayo. La primera consistió en comparar distintas medidas contra el medidor de consumo modelo ICO-EM-AU de la empresa Ecoworthy, que cuenta con una exactitud del 3 % y reporta corriente, voltaje, potencia activa, reactiva y factor de potencia.

Para ello se formaron dos grupos de electrodomésticos; uno para calibración y otro para ensayo, de forma de poder independizar ambas etapas. Estos grupos se muestran en la Tabla 8.3.

En las Tablas 8.4, 8.5, 8.6 y 8.7 se muestran los resultados de los ensayos.

Potencia	Descripción	IRMS ICO	IRMS	Error relativo
40W	Bombilla incandescente	0,20	0,15	23,08 %
60W	Bombilla incandescente	0,26	0,19	26,64 %
100W	Bombilla incandescente	0,49	0,34	30,89 %
200W	Bombilla incandescente	1,00	0,72	28,22 %
720W	Tostadora	3,43	2,58	24,80 %
750W	Cafetera	3,38	2,54	24,85 %
950W	Tostadora + Bombilla	4,42	3,35	24,21 %
2000W	Caldera eléctrica 1	8,78	6,67	24,06 %

Tabla 8.4: Ensayo de IRMS, valores en A.

Potencia	Descripción	VRMS ICO	VRMS	Error relativo
40W	Bombilla incandescente	237,70	240,00	0,97 %
60W	Bombilla incandescente	237,40	240,00	1,10 %
100W	Bombilla incandescente	237,60	240,00	1,01 %
200W	Bombilla incandescente	237,10	240,00	1,22 %
720W	Tostadora	233,90	237,00	1,33 %
750W	Cafetera	235,60	239,00	1,44 %
950W	Tostadora + Bombilla	233,40	237,00	1,54 %
2000W	Caldera eléctrica 1	228,70	234,00	1,18 %

Tabla 8.5: Ensayo de V_{RMS}, valores en V.

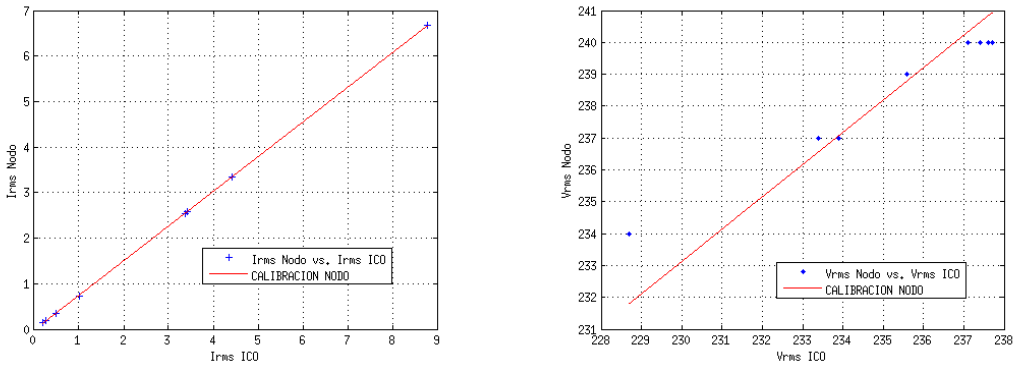
Potencia	Descripción	P ICO	P	Error relativo
40W	Bombilla incandescente	46,40	35,60	23,28 %
60W	Bombilla incandescente	61,50	47,60	22,60 %
100W	Bombilla incandescente	116,70	90,50	22,45 %
200W	Bombilla incandescente	238,30	185,00	22,37 %
720W	Tostadora	802,00	624,00	22,19 %
750W	Cafetera	797,10	616,00	22,72 %
950W	Tostadora + Bombilla	1040,00	806,00	22,50 %
2000W	Caldera eléctrica 1	2011,00	1573,00	21,78 %

Tabla 8.6: Ensayo de potencia activa, valores en W.

Potencia	Descripción	FP ICO	FP	Error relativo
40W	Bombilla incandescente	1,00	0,77	23,20 %
60W	Bombilla incandescente	1,00	0,77	22,58 %
100W	Bombilla incandescente	0,99	0,77	21,80 %
200W	Bombilla incandescente	0,99	0,78	21,42 %
720W	Tostadora	0,98	0,78	20,66 %
750W	Cafetera	1,00	0,77	22,64 %
950W	Tostadora + Bombilla	0,98	0,78	20,28 %
2000W	Caldera eléctrica 1	0,98	0,78	20,09 %

Tabla 8.7: Ensayo de Factor de Potencia.

Para calibrar el sistema se procedió a un ajuste lineal de las señales de voltaje y corriente a través de mínimos cuadrados en Matlab, como se puede apreciar en la Fig. 8.4. El ajuste lineal está dado por las ecuaciones (8.3) y (8.4).



(a) I_{RMS} (b) V_{RMS}

Figura 8.4: Ajuste lineal de las medidas.

$$V_{RMS} = 1,014.V_{RMS_{ICO}} \text{ con ECM=1,006 y } R^2\text{=0,7909.} \tag{8.3}$$

$$I_{RMS} = 0,7613.I_{RMS_{ICO}} - 0,02256A \text{ con ECM=0,01684 y } R^2\text{=0,9999.} \tag{8.4}$$

Finalmente esto da lugar a las transferencias del sistema:

$$\boxed{V_{RMS} = 368,9840.V_{fv}} \tag{8.5}$$

$$\boxed{I_{RMS} = 33,4287\Omega^{-1}.V_{fi} + 2,9634 \times 10^{-2}A} \tag{8.6}$$

Potencia	Descripción	IRMS ICO	IRMS	Error relativo
40W	Bombilla incandescente	0,19	0,32	64,95 %
60W	Bombilla incandescente	0,26	0,37	43,41 %
100W	Bombilla incandescente	0,49	0,57	16,33 %
200W	Bombilla incandescente	1,00	1,07	7,00 %
450W	Enceradora	1,87	2,07	10,70 %
650W	Sandwichera	3,27	3,35	2,35 %
850W	Sandwichera + Bombilla	4,24	4,31	1,65 %
1850W	Caldera eléctrica 2	9,04	9,14	1,13 %

Tabla 8.8: Ensayo final de I_{RMS}, valores en A.

Potencia	Descripción	VRMS ICO	VRMS	Error relativo
40W	Bombilla incandescente	240,00	239,00	0,42 %
60W	Bombilla incandescente	240,00	239,00	0,42 %
100W	Bombilla incandescente	236,50	235,00	0,63 %
200W	Bombilla incandescente	235,40	234,00	0,59 %
450W	Enceradora	233,90	233,00	0,38 %
650W	Sandwichera	232,60	232,00	0,26 %
850W	Sandwichera + Bombilla	232,20	231,00	0,52 %
1850W	Caldera eléctrica 2	228,40	229,00	0,26 %

Tabla 8.9: Ensayo final de V_{RMS}, valores en V.

Ensayo de mediciones

Luego de realizar la calibración y obtener las transferencias finales del sistema se procedió al ensayo el otro grupo de electrodomésticos. Los resultados se muestran en las Tablas 8.8, 8.9, 8.10 y 8.11.

La calibración del sistema permitió disminuir los errores relativos en las distintas mediciones. En el caso de la corriente el error fué importante, fundamentalmente para las corrientes más pequeñas (menores a 1A). Esto es consistente con la observación acerca de que los errores de cuantización que aumentan a medida que disminuye el nivel de las señales. Para las cargas más importantes el error disminuye a menos del 5 %.

El voltaje es la medida en la que se obtuvo el mejor comportamiento, con un error relativo en las observaciones realizadas de menos del 1 %. En el caso de la potencia, evidentemente el error introducido por la corriente ha impactado en su desempeño, aunque los errores obtenidos en esta etapa son de todas maneras menores al 4 % en relación con el medidor ICO.

Potencia	Descripción	P ICO	P	Error relativo
40W	Bombilla incandescente	46,70	47,40	1,50 %
60W	Bombilla incandescente	62,20	63,30	1,77 %
100W	Bombilla incandescente	116,50	119,00	2,15 %
200W	Bombilla incandescente	234,40	243,00	3,67 %
450W	Enceradora	380,50	383,00	0,66 %
650W	Sandwichera	763,30	790,00	3,50 %
850W	Sandwichera + Bombilla	982,50	1015,00	3,31 %
1850W	Caldera eléctrica 2	2063,00	2140,00	3,73 %

Tabla 8.10: Ensayo final de la potencia activa, valores en W.

Potencia	Descripción	FP ICO	FP	Error relativo
40W	Bombilla incandescente	1,00	0,62	38,02 %
60W	Bombilla incandescente	1,00	0,72	28,42 %
100W	Bombilla incandescente	0,99	0,89	10,26 %
200W	Bombilla incandescente	1,00	0,97	2,95 %
450W	Enceradora	0,85	0,79	6,58 %
650W	Sandwichera	0,99	1,02	2,67 %
850W	Sandwichera + Bombilla	0,99	1,02	2,98 %
1850W	Caldera eléctrica 2	0,99	1,02	3,28 %

Tabla 8.11: Ensayo final de Factor de Potencia.

8.2.4. Placa de comando de carga

En una primera instancia se probó la placa de comando de carga en forma aislada del resto del sistema. Para ello se le alimentó con la fuente auxiliar de 5VDC descrita en el Capítulo 5 y se inyectaron señales con valor de 3VDC en los terminales de control para confirmar la correcta activación y desactivación de cada uno de los dos relés que la componen. Las señales de 3VDC fueron generadas con una fuente auxiliar de laboratorio.

Una vez probado el funcionamiento básico de la placa, se procedió a la integración con el Z1, alimentando ambos bloques con la fuente auxiliar de 5VDC y conectando las salidas digitales del Z1 a los terminales de comando de los relés según se explicó en el Capítulo 5. Por otra parte se utilizó otro mote Z1 como border router conectado a un ordenador mediante el cual se accedió a los recursos de CoAP del nodo utilizando COPPER. Mediante esta herramienta se enviaron comandos en forma manual hacia el nodo y se observó que el mismo activaba y desactivaba los relés en forma correcta de acuerdo a los comandos enviados. Se observó un retardo prácticamente imperceptible entre el envío del comando y la operación de los relés. De este modo la prueba de la placa de comando y su

interacción con el Z1 se consideró exitosa.

8.2.5. Ensayo general del nodo

Una vez probado cada uno de los bloques de hardware del nodo y realizadas las calibraciones correspondientes para la etapa de medición, se procedió a montar un nodo completo integrando todos los bloques y se ensayó su desempeño. El nodo se integró entonces con los siguientes elementos implementados según se explicó en el Capítulo 5 y de acuerdo a la configuración mostrada en la Fig. 5.16 de dicho capítulo:

- Mote Z1 con su software correspondiente
- Placa de potencia
- Placa de acondicionamiento
- Placa de filtrado y protección
- Placa de relés
- Fuente auxiliar de 5VDC.

Ensayos realizados

Para ensayar las funcionalidades del nodo en su conjunto se interactuó con el mismo mediante la herramiento COPPER, utilizando las funciones de comunicación del nodo y accediendo a sus recursos CoAP para observar sus reportes y realizar comandos manuales. Para ello se utilizó un mote Z1 auxiliar oficiando de border router y conectado a un ordenador. Sobre este esquema de ensayo se realizaron los siguientes pasos y pruebas:

1. En una primera instancia se puso en funcionamiento el nodo sin estar conectado a ninguna carga. En esta situación se accedió a sus datos mediante COPPER comprobando así que las funciones de comunicación y procesamiento de señales se encontraban funcionando correctamente, ya que se pudo acceder exitosamente a los recursos y las medidas reportadas eran coherentes con los resultados obtenidos durante la etapa de calibración.

2. Para probar el funcionamiento de los relays se enviaron comandos de activación en forma manual mediante el recurso correspondiente utilizando COPPER. Se comprobó que los relays se activaban en forma correcta de acuerdo a lo esperado, con delays imperceptibles a nivel de la comunicación.

3. Como paso siguiente se conectó una carga de prueba y se evaluó el comportamiento del nodo con carga. Para ello se utilizó una lámpara incandescente de 40W. Accediendo a las mediciones nuevamente a través de COPPER se confirmó que las medidas reportadas seguían siendo correctas.

4. Se procedió a comandar la carga conectada, activándola y desactivándola mediante el envío manual de comandos CoAP a través de COPPER. Se observó que en este escenario los relays funcionaron correctamente, encendiendo y apagando la carga de la forma esperada.

5. Finalmente, se repitieron los ensayos 3 y 4 con una carga de mayor potencia (2KW). El nodo reportó nuevamente mediciones correctas y ante el primer envío de comando activó los relays de acuerdo a lo esperado. Sin embargo, tras repetir este ensayo en repetidas ocasiones se observó que el nodo dejó de responder, quedando inaccesible a la red HAN incluso a nivel IP (confirmado vía ping). En esta situación, si bien el nodo no se encontraba accesible desde el punto de vista de la red, sus funciones de procesamiento a nivel de software parecían ejecutarse en forma correcta de acuerdo al comportamiento observado en sus LEDs.

6. A partir de lo observado en 5) se procedió a resetear el mote Z1 y se observó que la comunicación y todas las funciones se restablecieron. Sin embargo, al repetir el paso 5) nuevamente, el mismo problema volvió a surgir luego de una serie de operaciones de comando exitosas.

Observaciones y conclusiones

Se observó que las fallas se daban únicamente con carga conectada, en particular con las cargas de potencias mayores y que en la mayoría de los casos la falla ocurría al momento del switcheo de la carga, ya sea en el encendido o apagado por medio del relay. Sin embargo, se observó un alto nivel de aleatoriedad en la falla, y no fué posible identificar y caracterizar las situaciones que la producían.

A partir de una revisión bibliográfica en torno a las características de la falla se elabora la hipótesis de que el problema se de debe a una alta susceptibilidad del sistema al ruido eléctrico de la red. En publicaciones tales como [47] y [2] entre otras se presenta información acerca de problemas típicos en sistemas con MCU causados por ruido eléctrico. Los síntomas presentados en tales documentos coinciden exactamente con los experimentados durante los ensayos descritos anteriormente. Según estos documentos, los MCU son normalmente muy susceptibles al ruido eléctrico y en particular al ruido de tierras, que entre otros problemas genera errores en el contador de programa y stack de ejecución, tornando el programa impredecible o llevándolo a un estado inestable. El hecho de que al reiniciar el nodo (reset del Z1) el mismo comienza a operar sin problemas de inmediato refuerza esta hipótesis y coincide con los comportamientos típicos descritos en los documentos citados.

Debido a que esta falla se presentó en la última etapa de ensayos sobre el final del proyecto, no fué posible por motivos de cronograma realizar mejoras y nuevos experimentos para confirmar esta hipótesis. Sin embargo se considera que la misma está fuertemente respaldada por las observaciones.

Para confirmar la hipótesis e intentar solucionar el problema se podrían implementar circuitos de aislación y protección contra ruido como los presentados

en [47]. Sin embargo por los motivos de cronograma mencionados y teniendo en cuenta que el hardware no es un aspecto central del proyecto, dicha actividad quedará propuesta como trabajo futuro.

Síntesis de resultados

En síntesis, el nodo en su versión completa presenta el comportamiento esperado de acuerdo al diseño en todos sus aspectos funcionales. Sin embargo, presenta un grado de inestabilidad y vulnerabilidad que resulta inaceptable, y deberá ser resuelto en diseños posteriores.

8.3. Integración y ensayos globales del sistema

Una vez ensayados todos los bloques de la plataforma HEMS por separado se procedió a integrarlos para evaluar el sistema en su conjunto. Para ello se montó la plataforma completa incluyendo HEC y nodos comunicados a través de la HAN. En síntesis la plataforma ensayada se montó en base a los siguientes componentes:

- HEC, incluyendo software de optimización de termotanque basado en utilización de la API desarrollada.
- 1 nodo con funcionalidad completa (comunicación + medición + comando).
- 4 nodos únicamente con funcionalidad de comunicación.
- Interfaz de usuario en plataforma web.

Los 4 nodos con funcionalidad restringida consistieron únicamente en motes Z1 corriendo el programa completo de los nodos pero sin las placas hardware necesarias para las funciones de comando y medición. Si bien estos motes no brindan las funcionalidades completas de un nodo, resultan útiles para la evaluación de varios aspectos de la plataforma entre los cuales se encuentran el desempeño de la HAN, comunicación con el HEC y reportes hacia interfaz de usuario. Estos nodos aceptan comandos y reportan mediciones de la misma forma que el nodo completo, solo que los valores reportados no corresponden a medidas reales, y la activación de los relés se manifiesta únicamente en el encendido y apagado de los LEDs testigos programados en los motes.

El objetivo de este ensayo consiste en evaluar todas las funcionalidades de la plataforma desde una perspectiva global y de integración, evaluando diferentes casos de uso.

Evaluación de interfaz de usuario y sus funcionalidades

Con el sistema completo en funcionamiento se accedió a la interfaz de usuario web y se activaron los comandos para cada uno de los nodos. Se observó que los comandos funcionaron con éxito en todos los casos, aunque con un retardo del orden de los 10 segundos entre el envío y confirmación de activación o desactivación

Figura 8.5: Visualización del historial ofrecida por la interfaz web de usuario.

mediante el indicador gráfico de estado de carga. Las únicas excepciones se dieron al utilizar el nodo completo con cargas conectadas, pero en todos los casos en los que hubo error el mismo se debió a la falla del nodo descrita en la sección 8.2.5 y no a problemas de comunicación con la interfaz de usuario. Con el resto de los nodos no se detectó ninguna falla de comandos.

El ensayo de los comandos se realizó incluso utilizando diferentes teléfonos móviles conectados a internet a través de la red 3G, emulando la situación de ejemplo en la que un usuario quisiera encender el calefón en forma manual remota mientras viaja de regreso al hogar. En todos los casos la operación fue exitosa y pudo realizarse con facilidad gracias a que la plataforma web utilizada (Freeboard) se encuentra gráficamente optimizada para smartphones.

Luego, se dejó el sistema en funcionamiento durante varias horas y se accedió a la interfaz para observar los datos y tendencias históricas de reportes almacenados. Se observó que todos los datos fueron almacenados correctamente. Sin embargo este ensayo se realizó desconectando el nodo completo de la red eléctrica debido a su inestabilidad, por lo cual los datos almacenados no fueron representativos de medidas reales. No obstante, el ensayo sirvió para confirmar que el reporte e historización funciona bien desde el punto de vista del software de los nodos, HEC e interfaz, así como también en lo que refiere a la HAN.

La Fig. 8.5 muestra un ejemplo de la visualización del historial de consumo disponible para el usuario.

Evaluación del algoritmo de optimización implementado

El objetivo de este ensayo fue evaluar el funcionamiento del algoritmo de optimización implementado y por ende también las funcionalidades brindadas por la plataforma disponibles mediante la API desarrollada. Se hizo frente a una limitación importante que consiste en la inestabilidad de la electrónica y el MCU del nodo en su versión completa descrita en la Sección 8.2.5. Esta limitación impidió poner en funcionamiento el nodo completo comandando un termotanque durante períodos de tiempo suficientes como para ensayar el algoritmo. Para eludir este obstáculo se decidió probar el comportamiento del algoritmo sin utilizar el nodo conectado a la red y a la carga. Para ello se puso a funcionar la plataforma

con el nodo completo desconectado de la red eléctrica y se evaluó el resultado del algoritmo observando la activación y desactivación de los relés y los LEDs testigos del mote. Se realizó en el HEC un log de los comandos y reportes de estado de relé y se los contrastó contra la salida teórica del algoritmo, llegando a la conclusión de que el mismo se ejecutó correctamente de acuerdo a lo esperado.

Si bien el ensayo realizado fue parcial, ya que no se realizó el control eléctrico sobre el termotanque, los resultados permiten concluir que la plataforma brindó las funcionalidades suficientes para su correcta ejecución.

No se pudo demostrar con mediciones objetivas que el algoritmo ensayado realmente mejora la eficiencia del calefón ya que por el problema mencionado no se pudieron monitorear los consumos. Sin embargo, el desarrollo del algoritmo no era un objetivo en sí, sino una forma de evaluar la plataforma HEMS.

Evaluación de delay en mediciones

Con el objetivo de evaluar el desempeño general de la red en lo que refiere al retardo para el reporte de los datos, se midió el tiempo que tarda el sistema desde que un usuario de la API intenta acceder a las medidas de un nodo hasta que esta información es recibida y correctamente registrada en el log de mediciones.

Los tiempos dependen de la estrategia particular de medición implementada, ya que el desarrollador podrá coordinar grupos de nodos en distintas corrutinas de medidas (ver C.1.2) en función de las prioridades de los dispositivos asociados. Por ejemplo, si existen ciertos dispositivos de los que se requiere obtener reportes con mayor prioridad y no es deseable retardar el acceso a esta información por esperar las medidas de otros nodos, se debe combinar adecuadamente la utilización de las funcionalidades de la API para que la espera por un nodo no bloquee el reporte de los nodos prioritarios.

Para el ensayo realizado se utilizó el esquema de implementación detallado anteriormente en la Sección 7.2 en el cual todos los nodos a medir tienen la misma prioridad y además el loop de medición espera por cada nodo sin pasar al siguiente hasta no recibir una respuesta. Esto significa que si un nodo no responde el loop quedaría en espera indefinidamente. Los tiempos obtenidos podrían ser eventualmente mejorados con una implementación que resuelva el *polling* utilizando otra estrategia.

En el diseño del experimento se utiliza la marca de tiempo que por defecto se agrega en cada entrada del log de mediciones, según se describe en la Sección C.3.1, y se agrega una marca de tiempo adicional que indica el momento en que el usuario de la API decide solicitar la medida de un nodo:

```
tini = time.time()
med = yield from wsncommunicationtest.getMeasure(n, tini)
```

Además se modificó la corrutina de medición para que acepte este nuevo argumento y lo agregue al final de los datos cada vez que escribe en el log:

```
  #Se acepta nuevo argumento para recibir el tiempo inicial:
@asyncio.coroutine
def getMeasure(node, tini):
    #Modificación en linea que escribe el registro en el log:
writeMeasureLog(sal+[tini],str(node.ip))
```

Implementadas estas pequeñas modificaciones al código, se inició el sistema con 5 nodos distribuidos en distintas habitaciones de un domicilio y se lo dejó funcionando durante una noche.

Luego se midieron los tiempos entre los instantes indicados por las marcas de tiempo, obteniendo los resultados indicados en la Tabla 8.12

	Promedio	**Desviación estandar**	**Mínimo**	**Máximo**
Global	0.41	1.02	0.09	43.67
Nodo 1	0.45	1.49	0.1	35.03
Nodo 2	0.44	0.51	0.14	6.8
Nodo 3	0.32	0.41	0.1	9.04
Nodo 4	0.62	1.69	0.09	43.67
Nodo 5	0.34	0.63	0.1	16.2

Tabla 8.12: Tiempos de demora entre solicitud de una medida y escritura del log correspondiente.

Se puede apreciar que el tiempo promedio es inferior a medio segundo y que la amplia mayoría de los datos presentan una demora inferior a un segundo. Esto se puede ver más claramente en los histogramas de la Fig. 8.6 que aparece sobre el final del capítulo.

En base a este ensayo preliminar, que deberá ser profundizado en trabajos futuros, se puede concluir que los tiempos de demora de la red cumplen con los requerimientos básicos propuestos en el Capítulo 3.2.1.

8.4. Síntesis del capítulo

En el presente capítulo se realizó un ensayo y evaluación de desempeño de la plataforma HEMS y cada uno de sus módulos, incluyendo el algoritmo de optimización del termotanque implementado como ejemplo de caso de uso de la plataforma.

Se describieron los ensayos realizados al sistema, comenzando por los asociados a los componentes particulares y luego pasando a las pruebas globales del sistema en su conjunto.

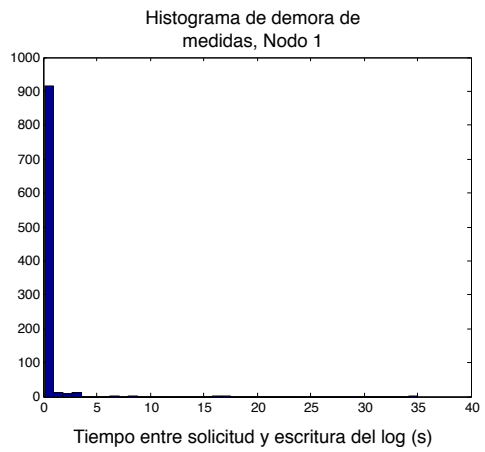
A partir de los ensayos realizados se presentaron los resultados obtenidos y se realizaron observaciones y apreciaciones respecto de los mismos. Se presentaron también las conclusiones específicas a las que se arribó tras el análisis.

Los resultados indicaron que el sistema funcionó en general de acuerdo a lo esperado. Sin embargo, en el caso de la evaluación de la electrónica de los nodos de medición y comando se encontró que los mismos presentan una gran inestabilidad debida a la vulnerabilidad del MCU al ruido eléctrico de la red. El diseño de las placas de medición y comando no resultó adecuado en este sentido ya que no protege al MCU de este tipo de ruido. No obstante, en todos los demás aspectos las pruebas fueron exitosas. En particular se observó que el error en las medidas se encuentra dentro de los márgenes esperados.

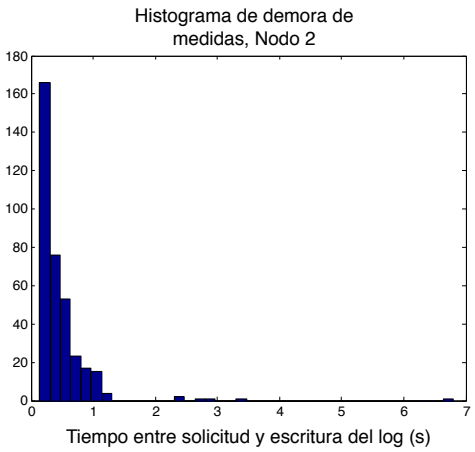
En lo que respecta a la HAN y el stack de protocolos utilizados para la comunicación entre los nodos y el HEC, la misma presentó un comportamiento satisfactorio, mostrándose robusta y estable.

De la misma forma, la interfaz de usuario y sus funcionalidades fueron probadas con éxito. En particular se observaron correctamente los indicadores gráficos online, las tendencias de variables historizadas, y se probaron los comandos manuales, incluso utilizando un smartphone.

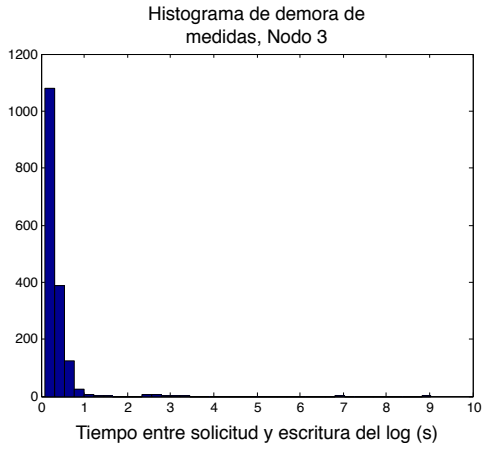
En cuanto al software del HEC, esto es la API y el algoritmo de optimización implementado como caso de uso, se confirmó que la biblioteca de funciones desarrollada es fácilmente utilizable para la integración de un algoritmo de optimización, y las funciones primitivas brindadas ofrecieron las funcionalidades buscadas en forma exitosa. No obstante, debido al problema de inestabilidad en la electrónica de los nodos, no se pudo confirmar fehacientemente que el algoritmo probado optimiza el consumo de un termotanque real.



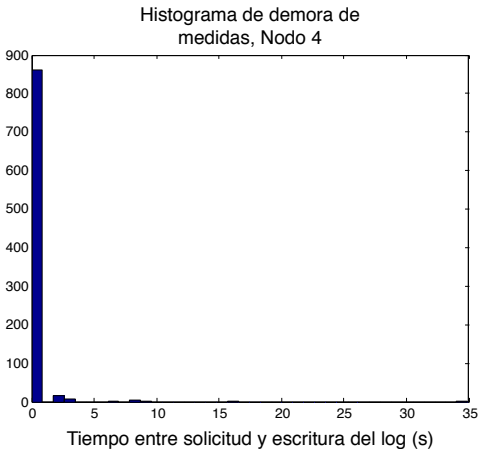
(a) Nodo 1



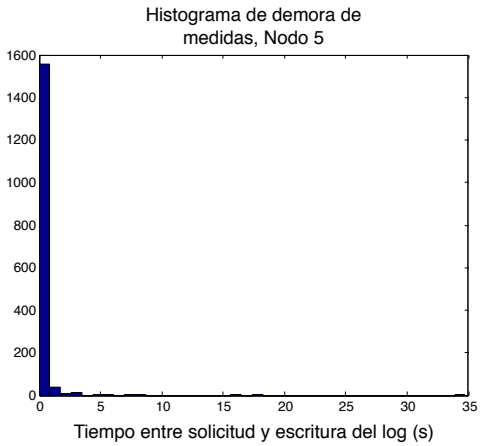
(b) Nodo 2



(c) Nodo 3



(d) Nodo 4



(e) Nodo 5

Figura 8.6: Tiempo de demora entre solicitud de medida y obtención de datos para cada Nodo.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 9

Conclusiones y líneas de trabajo futuro

Tras los ensayos realizados y en base a los resultados obtenidos se puede afirmar que para cada uno de los módulos de la plataforma propuesta, incluyendo hardware y software, se lograron las funcionalidades buscadas. Desde este punto de vista es posible concluir que se cumplieron todos los objetivos del proyecto en forma exitosa. No obstante, es oportuno destacar que en el caso del hardware de los nodos de medición y comando, no se logró una versión estable del diseño de la electrónica. En este sentido, se hubiera esperado obtener mejores resultados. La inestabilidad de estos componentes de hardware limitó las pruebas y ensayos globales del sistema y la plataforma. En particular este problema hizo que no fuera posible ensayar y evaluar en un escenario real el algoritmo de optimización de termotanque implementado en el Capítulo 7, por lo cual su evaluación y las conclusiones respecto de los resultados de optimización se basaron en ensayos parciales y simulaciones.

Uno de los aspectos más exitosos del proyecto fue el desarrollo de la API, que convirtió al HEC y los nodos en una plataforma HEMS abierta y facilitadora de futuras investigaciones y pruebas de algoritmos. Este enfoque no formaba parte de los planes iniciales, sin embargo se entendió que podía constituir un aporte significativo en pos de facilitar iniciativas de investigación en torno al tema.

Esta API desarrollada proporciona una herramienta para que un desarrollador de algoritmos pueda poner a prueba su trabajo sin necesidad de preocuparse por los aspectos de implementación de los demás componentes y módulos del sistema. Luego de aprender los conceptos básicos del lenguaje Python, y de estudiar el funcionamiento del conjunto de bibliotecas estándar, se pudo comprobar que su clara sintaxis facilita el rápido aprendizaje y este era uno de los objetivos buscados con la elección del lenguaje de diseño. Si bien hay muchos aspectos a mejorar, sobre todo teniendo en cuenta la poca experiencia que se tenía al comenzar el desarrollo, se considera haber logrado un diseño modular que brinda independencia ante posibles cambios en la implementación.

Otra área del proyecto que merece ser destacada es la implementación de la HAN utilizando un stack conformado fundamentalmente por IEEE 802.15.4, 6LoWPAN, IPv6, RPL y CoAP. Lo primero que debe decirse es que la tecnología inalámbrica de radio utilizada resultó satisfactoria y se considera adecuada para

aplicaciones de este tipo de acuerdo a los resultados obtenidos y los desempeños observados. Dentro de esta familia de comunicaciones inalámbricas ZigBee se presentaba como la opción con mayor presencia y soporte comercial, sin embargo los resultados mostraron que la 6LoWPAN y RPL junto a IPv6 representan una alternativa sumamente interesante. En el caso de RPL los resultados preliminares en cuanto al desempeño del ruteo y conformación de redes mesh fueron altamente satisfactorios, si bien sería deseable la realización sistemática de una mayor cantidad de pruebas. La elección de CoAP como protocolo de aplicación resultó adecuada, y si bien su comprensión y dominio insumió un tiempo considerable, el tiempo de desarrollo se redujo al disminuir las tareas de manejo de paquetes de comunicación a nivel del programa de usuario.

En lo que respecta al área del software embebido, Contiki OS se mostró como una opción muy interesante sobre todo debido al amplio conjunto de stacks de comunicación implementados de los que se dispone al utilizar este sistema operativo. Además cuenta con herramientas de simulación y una gran cantidad de ejemplos de código que sirven como punto de partida para iniciar el desarrollo. Comprender el funcionamiento de este sistema operativo requirió un tiempo considerable, pero el balance final fue positivo ya que una vez dominado, al menos a un primer nivel básico, el tiempo de desarrollo y programación se vió reducido.

En cuanto a los motes Z1 de Zolertia, HW seleccionado para la implementación de los nodos y el border router, si bien el producto se desempeñó adecuadamente hubo algunos inconvenientes. El principal problema fue que no se contó con una versión estable del GNU Compiler Collection (gcc) que funcionara correctamente con el MCU incluido en el mote, y debido a este problema nunca se logró aprovechar la totalidad de la memoria ROM. Esto generó algunos problemas a la hora de la programación y obligó a recortar algunas funcionalidades, como por ejemplo prescindir del RDC en la radio. Otro motivo que hace considerar que el Z1 puede no ser una opción adecuada es el tipo de ADCs que su MCU ofrece. Tal como se describió en el Capítulo 5 y en el Apéndice B el MSP430 del Z1 no cuenta con ADCs del tipo sigma-delta, que a partir de los diseños de referencia allí estudiados y citados se presentan como la mejor opción para este tipo de aplicaciones. Haber detectado este problema a tiempo hubiera permitiendo obtener un mejor desempeño del sistema en términos de distorsión por aliasing, haciendo suficiente el uso del sencillo filtro RC propuesto.

Otro módulo de hardware utilizado fue la placa de relés Phidget. Este componente funcionó correctamente pero su integración al mote Z1 no fue tan sencilla como era de esperarse sobre todo debido a los ya mencionados problemas de ruido eléctrico, que se vieron agravados al activar o desactivar cargas de alto consumo. Una primer conclusión es que esta placa no constituye una buena opción. Tras los resultados obtenidos se considera que hubiera sido mejor optar por relés de estado sólido.

En lo que respecta a los desarrollos de hardware propios, si bien nunca fueron considerados como centrales o foco del proyecto, debe decirse que constituyeron el punto más problemático y donde el éxito en el cumplimiento de los objetivos se ve relativizado. Es decir, si bien los objetivos se cumplen, se esperaban resul-

tados mucho mejores desde el punto de vista de la confiabilidad y la robustez. La inestabilidad de la electrónica y la susceptibilidad ante ruido eléctrico de la red, manifestado en comportamientos inestables de MCU, hacen que si bien los circuitos funcionan de acuerdo al diseño, los mismos no sean utilizables en la práctica para un sistema real. El circuito de la placa de medición debe rediseñarse a los efectos de limitar al máximo el ruido eléctrico propagado hacia el microcontrolador, a través de mecanismos de aislación. Teniendo en cuenta que el circuito de medición fue inspirado en un diseño de referencia con origen en EEUU, país cuyo sistema de generación y distribución de energía eléctrica es diferente al de Uruguay, es necesario realizar las adaptaciones correspondientes para contemplar por ejemplo el hecho de no contar con un neutro aterrado.

Respecto de la placa de filtrado y protección, además de las consideraciones hechas anteriormente sobre la influencia del tipo de ADC en los requerimientos de filtrado, sería deseable tener mejores niveles de protección implementando por ejemplo un circuito del tipo *voltage clamp* según se describe en el Apéndice B.1.4.

No obstante las observaciones realizadas en los párrafos anteriores, resulta oportuno destacar que en lo que respecta a la calidad de las mediciones obtenidas, dejando de lado aspectos de estabilidad y robustez, los resultados obtenidos fueron satisfactorios. Estos resultados se deben no solo a la electrónica, sino al enfoque utilizado en el programa de procesamiento de señales que resultó ser exitoso a pesar haber presentado grandes desafíos debidos a las limitaciones del mote en términos de capacidad de RAM y ROM, que obligaron a maximizar esfuerzos para lograr eficiencia en el software y utilizar métodos alternativos al no ser posible el almacenamiento de muestras de más de un ciclo.

9.1. Verificación de objetivos Específicos

La verificación del cumplimiento de las metas planteadas al comienzo del proyecto en base a los criterios de éxito definidos en la propuesta inicial (Sección 1.3), constituye un enfoque objetivo para la evaluación del proyecto y el desempeño del grupo. En este sentido, se presentan a continuación los objetivos específicos del proyecto, discutiendo y evaluando en cada caso el cumplimiento o no de los mismos y el grado de éxito alcanzado.

1. *Construir un Módulo de Carga, capaz de operar un dispositivo o electrodoméstico, controlando de esta forma la potencia consumida por el mismo.*

   Este objetivo se cumplió dado que se logró implementar un circuito de control integrando soluciones comerciales (un mote Z1 y una placa de relés) y que dicha implementación resultó capaz de comandar cargas de hasta 10A en forma ON/OFF. Sin embargo, desde un punto de vista más riguroso, cabe destacar que la elección del módulo de comando no resultó óptima, ya que se observó que los relés electromecánicos favorecían un incremento en el ruido eléctrico, en desmedro de la estabilidad del sistema, haciendo que el comportamiento del mote Z1 resulte inestable e impidiendo su utilización

durante períodos de tiempo prolongados. En síntesis, se logró la funcionalidad buscada pero la implementación no resultó robusta ni confiable.

2. *Construir un Módulo de Medición de consumo, capaz de monitorear y reportar la potencia consumida por un dispositivo o una línea de carga (por ejemplo, una línea asociada a la iluminación o a un grupo de electrodomésticos).*

   Este objetivo fue cumplido con éxito. Se logró diseñar e implementar un nodo de medición en base a diseños de referencia adaptados e integración de hardware. Los ensayos mostraron que se lograron mediciones con un error cercano al 5 % en los rangos de interés, lo cual se considera como éxito de acuerdo a los criterios estipulados inicialmente.

   Como aspecto mejorable, al igual que en el caso del nodo de control de carga, cabe señalar que la versión de implementación lograda introduce ruido eléctrico en el sistema contribuyendo a un comportamiento inestable del mote. Si bien el desarrollo de hardware no estaba en el foco central del proyecto, y en particular se dejaron fuera del alcance los aspectos de seguridad para el usuario y equipos, por su importancia en un futuro desarrollo cabe mencionar que este es un punto importante a mejorar.

3. *Diseñar y construir un Módulo Controlador. Este módulo será capaz de realizar el control del sistema, enviando los comandos y analizando la información recibida por los demás módulos. Será el encargado de centralizar la comunicación con un sistema global Smart Grid (el sistema global está fuera del alcance de este proyecto).*

   Este objetivo se cumplió en su totalidad con un alto grado de éxito según los criterios previamente establecidos. Durante el desarrollo del proyecto se detectó la oportunidad de realizar un aporte particular y se decidió ampliar el alcance y avanzar hacia un objetivo más ambicioso y general, que consistió en diseñar e implementar no solo un controlador (HEC) si no una plataforma de software compuesta por bibliotecas de funciones (API) permitiendo que el sistema HEMS original se convierta en una plataforma abierta y escalable.

   Desde el punto de vista de las funcionalidades logradas se destaca la API con su evaluación y demostración mediante un caso particular de sumo interés como lo es la optimización del consumo del termotanque. Se destaca además la interfaz de usuario gráfica sobre plataforma web, que permite no sólo mostrar las medidas reportadas al usuario si no también enviar comandos a los electrodomésticos en forma remota.

4. *Diseñar e implementar un algoritmo de control de consumo para un Nodo de Carga, analizando el perfil de operación de al menos un tipo de dispositivo o electrodoméstico, de forma de obtener un método de control y operación adecuado al mismo. Con esto se pretende probar que la plataforma desarrollada comanda satisfactoriamente los dispositivos conectados a ella.*

Teniendo en cuenta que el estudio de algoritmos de optimización es un tema suficientemente profundo como para dedicarle un proyecto entero, y que existen muchos algoritmos propuestos a nivel internacional, se priorizaron otros aspectos del proyecto que constituyen un aporte más valioso. En este sentido se dió mayor importancia al desarrollo de la API y se buscó implementar un algoritmo simple con el objetivo principal de probarla. Esta implementación sirvió para confirmar que el sistema es capaz de comandar los dispositivos de acuerdo a lo determinado por el algoritmo, de forma concurrente con las tareas de reporte de medición y ejecución de comandos manuales del usuario. Si bien no se lograron mediciones objetivas de ahorro energético, es razonable esperar que en el marco de una tarifa dinámica este tipo de implementación reduzca el costo para el usuario.

5. *Diseñar e implementar el sistema de comunicación entre los nodos y el HEC, seleccionando la tecnología más adecuada para la aplicación.*

   Se logró implementar un sistema de comunicación robusto en base a tecnologías probadas y abiertas. Luego de un análisis y estudio de las distintas alternativas, el principal reto consistió en comprender cómo funcionan las tecnologías seleccionadas y en particular aprender a usar las implementaciones existentes de las mismas. Se comprobó que Contiki OS es una plataforma sumamente valiosa ya que permitió reducir drásticamente los tiempos de desarrollo al ofrecer implementaciones abiertas de los stacks elegidos.

9.2. Apreciaciones generales

Para lograr los objetivos del proyecto se requirió comprender de forma global los conceptos relacionados a las redes eléctricas en lo que refiere al nuevo paradigma que implican las Redes Eléctricas Inteligentes y realizar un análisis de un gran conjunto de tecnologías posibles para diseñar una solución al problema planteado.

Además de resultar una oportunidad de gran aprendizaje para los autores del proyecto, se considera que el mismo aborda una temática de creciente interés y puede servir como punto de partida y propiciar trabajos futuros que intenten mejorar los aspectos que no fueron implementados de forma totalmente satisfactoria, así como complementar el desarrollo con nuevas funcionalidades.

Dejando de lado por un momento la evaluación objetiva de los resultados, se considera oportuno destacar que el proyecto presentó una instancia de aprendizaje a la vez desafiante y motivadora. Se abarcaron áreas de diversa índole, como diseño y desarrollo de hardware, programación de sistemas embebidos y la utilización de conceptos avanzados de programación concurrente. Muchas de estas áreas se presentaban sumamente desafiantes debido a la poca experiencia previa de los integrantes del grupo en dichos campos. A modo de ejemplo, el proyecto requirió aprender dos nuevos lenguajes de programación, tales como C y Phyton. De la misma forma se incursionó en la investigación y utilización de protocolos de comunicación para redes inalámbricas de sensores, integrando estas con Internet y plataformas web.

Desde una perspectiva general, cabe destacar y valorar todo el proceso más allá de los resultados concretos obtenidos, que de por sí dejan sumamente satisfecho al grupo. Se considera haber abarcado una amplia variedad de temáticas y conceptos, logrando articular un conjunto de tecnologías e integrarlas en una única solución para satisfacer requerimientos concretos.

9.3. Líneas de trabajo futuro

A continuación se intentará especificar cuáles son, según el criterio de los autores, los aspectos más interesantes para abordar en trabajos futuros. Algunos de los puntos destacados añadirán funcionalidades interesantes al sistema o presentan mejoras deseables, y otros se podrían identificar como necesarios para que la plataforma sea realmente útil para el desarrollo y ejecución de algoritmos de optimización.

9.3.1. Mejoras en el circuito de medición

Para que la plataforma tenga una utilidad real y genere un aporte sustantivo es necesario rediseñar el circuito de medición. Los principales aspectos a tener en cuenta son los siguientes:

- Rediseñar el circuito teniendo en cuenta la necesidad de aislar el microcontrolador de las distintas fuentes de ruido.
- Aumentar la precisión de medición, en particular para la corriente en los rangos más pequeños. Esto podría lograrse utilizando múltiples etapas de amplificación para diferentes resoluciones (en el Capítulo 5.3.4 se realizó un análisis de los motivos por los que se excluyó esta alternativa en el diseño actual).
- Mejorar la etapa de filtrado y muestreo de la señal. Se propone considerar la posibilidad de utilizar conversores ADC sigma-delta conservando filtros RC sencillos.
- Tener en cuenta aspectos de seguridad para el usuario y el equipamiento.

9.3.2. Mejoras en la capacidad de comandar dispositivos

Para poder incorporar estrategias de optimización más complejas puede ser necesario implementar nodos con capacidad de comando más avanzada que el básico ON/OFF incorporado actualmente. Esto podría servir por ejemplo para comandar sistemas de climatización con aire acondicionado controlando los set point y potencia de los equipos.

Algunas mejoras deseables:

- Para lograr un diseño más robusto de los nodos, es necesario minimizar el ruido introducido por el módulo de comando. En un análisis preliminar se detectó que el encendido y apagado de los dispositivos influye sobre la estabilidad del microcontrolador, y si bien se logró mejorar esto añadiendo componentes en paralelo a los relés actuales para disminuir los efectos no deseados introducidos por las bobinas, se propone estudiar la posibilidad de utilizar relés de estado sólido.
- Incorporación de módulos de control infrarrojo para dispositivos que permitan este tipo de control (típicamente equipos de aire acondicionado).

9.3.3. Mejoras en el sistema de comunicación y la API desarrollada

Las mejoras en la API diseñada facilitarán el desarrollo de pruebas de algoritmos de optimización, y un aspecto clave para la viabilidad la plataforma es el sistema de comunicación implementado. Además de ser necesarias más evaluaciones sistemáticas y exhaustivas del sistema de comunicación, que no fueron realizadas debido a los plazos del proyecto, otros puntos de mejora deseables son:

- Integración de PLC (Power Line Communication) con las tecnologías de radio ya implementadas. Existen algunos trabajos que abordan esta temática y podría resultar en una mejora considerable del sistema [40].
- Estudio de implementaciones de Border Router más fácilmente configurables.
- Mejoras en la implementación de la búsqueda de nodos disponibles en la red. La API actual implementa esto de una forma ampliamente mejorable, pero se diseñó pensando en que los cambios en esta implementación no afecten a los algoritmos que la utilicen.
- Autodetección de las funcionalidades y tipos de mensajes que puede aceptar cada nodo. Actualmente se trabaja con recursos fijos previamente establecidos en todos los nodos, y sería deseable que cada nuevo nodo pueda publicar esta información y así aportar más flexibilidad al diseño.

9.4. Gestión del proyecto

Un aspecto muy importante de la ejecución del proyecto es el de la gestión. En el Apéndice D se presentan detalles acerca de los diferentes aspectos de la gestión del proyecto, entre los cuales se destacan los flujos contables, análisis de dedicación y cumplimiento de plazos. Sin embargo resulta oportuno dedicar unos párrafos en esta sección para presentar un breve resumen de las valoraciones más importantes al respecto.

En lo que respecta al tiempo de dedicación al proyecto, en relación a la asignación de recursos, los resultados han sido buenos, ya que el monto de horas de

empleadas se mantuvo razonablemente ajustado a la planificación inicial. El sobrecosto en horas-hombre se ubicó por debajo del 20 %, y si bien en el último mes hubo una esperada sobrecarga de tareas para mejorar la documentación, la distribución del esfuerzo durante el proyecto estuvo de acuerdo a lo planificado.

Otro aspecto importante tiene que ver con los plazos de entrega y cumplimiento de entregables pautados durante la planificación del proyecto. En este sentido la valoración es correcta, ya que se cumplió con los entregables previstos para cada uno de los dos hitos pautados. En lo que refiere a la extensión total del proyecto, es oportuno puntualizar que el plazo no se cumplió en forma estricta, ya que fue necesario solicitar una prórroga de un mes. Sin embargo, la extensión en el plazo estuvo debidamente justificada particularmente por el hecho de que se decidió presentar un paper a la conferencia 2015 ISGT-LA[1] mostrando el trabajo realizado durante el proyecto. Esto insumió un tiempo importante pero se consideró que podría significar un aporte interesante.

Finalmente, en lo que refiere a la gestión de los recursos económicos, que facilitó el desarrollo del proyecto en toda su extensión y alcance, cabe mencionar que se logró la financiación por parte de la ANII y la CII gracias a haber tenido la iniciativa para la postulación del proyecto. Una vez conseguida la financiación, la correcta gestión permitió adquirir todos los componentes necesarios y finalizar el proyecto con los recursos previamente asignados.

[1] http://isgtla.org/, Mayo 2015.

Apéndice A

Programación Contiki

A.1. Herramientas de programación en Contiki

A.1.1. Versión Contiki y GCC

Estas fueron las versiones utilizadas para la programación y compilación de microcontroladores:

- Versión Contiki: Contiki 2.7
- Versión GCC: 4.6.3

A.1.2. Instant Contiki

Contiki ofrece una herramienta denominada Instant Contiki[1], que consiste en una máquina virtual con el software preinstalado para programar y testear programas en Contiki. Como del software Contiki, se ofrecen bibliotecas con ejemplos de referencia. Por motivos de simplicidad, esta fue la herramienta utilizada a la hora de programar microcontroladores utilizando Contiki.

A.1.3. Cooja

Una de las ventajas de Contiki es que cuenta con una herramienta integral de simulación y debug, incluyendo la posibilidad de simular redes de comunicación incluso a nivel de radio. Esta herramienta se denomina Cooja[2]. y viene instalada en Instant Contiki, y fue utilizada para realizar una primer evaluación de la HAN y su diseño.

Cooja permite guardar simulaciones, incluyendo la arquitectura de una red de motes configurada. Las simulaciones en Cooja se guardan como archivos de extensión .csc.

[1] http://www.contiki-os.org/start.html#install-instant-contiki, Mayo 2015

[2] https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja, Mayo 2015

A.1.4. COPPER

COPPER es un plugin desarrollado para el navegador Firefox[3], que implementa el protocolo CoAP e incorpora una interfaz web gráfica a través de la cual el usuario puede interactuar con servidores CoAP (en particular con los nodos) descubriendo, observando y escribiendo recursos. También permite ejecutar funciones de diagnóstico. Este plugin resulta sumamente práctico a la hora de realizar ensayos y debug sobre sistemas de comunicación basados en este protocolo. En la figura A.1 se muestra una captura de pantalla de COPPER, donde se observa su aspecto y algunas de sus características y funciones.

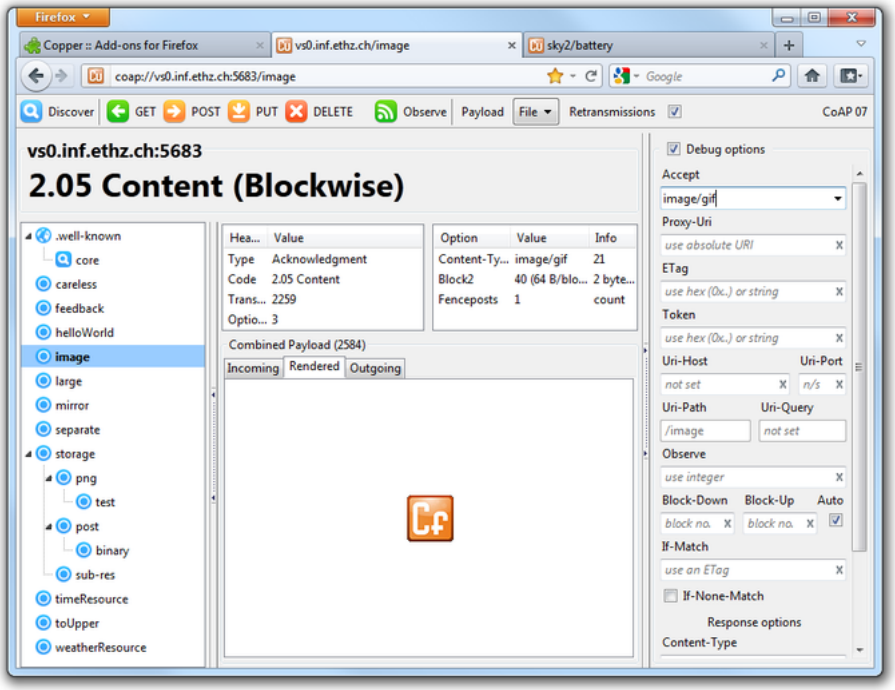


Figura A.1: COPPER Firefox Plugin

A.2. Generalidades acerca del software de los nodos y el Border Router

A lo largo de esta sección, siempre que se haga referencia a rutas de ubicación de archivos de programas de Contiki, las mismas estarán referenciadas al sistema de archivos por defecto de Instant Contiki versión 2.7.

[3]https://addons.mozilla.org/En-us/firefox/addon/copper-270430/, Mayo 2015

A.2.1. Estructura general de los programas

Se creó un directorio denominado "smart-grid" dentro del directorio de ejemplos de Contiki para comenzar a trabajar sobre una copia de los ejemplos originales. El software de los nodos fue ubicado en:

.../examples/smart-grid/er-rest-example

Se compone de los siguientes archivos fuente:

- *er-server-SG.c*: Programa principal.
- *z1-ADCs-SG.c*: Manejo de ADC's.
- *z1-ADCs-SG.h*: Configuración ADC's.
- *variables_globales.h*: Variables globales del programa.
- *project-conf.h*: Configuración de Contiki y stacks de comunicación.

El código correspondiente a los recursos CoAP se encuentra en:

.../examples/smart-grid/er-rest-example/resources

Este se compone de los siguiente archivos:

- *res-mediciones-SG.c*: Recurso CoAP para reporte de medidas
- *res-comando-SG.c* Recurso CoAP para comando de relés

El software del Border Router, se ubica en:

.../examples/smart-grid/rpl-border-router

Este se compone de los siguientes archivos fuente:

- *border-router-SG.c*: Programa principal
- *project-conf.h*: Configuración de Contiki y stacks de comunicación

A.3. Software comunicación HAN (nodos y Border Router)

A.3.1. Diseños de referencia y punto de partida

Una ventaja importante de trabajar con Contiki es que además de contar con stacks de comunicación ya desarrollados y testeados, se dispone de un interesante conjunto de ejemplos de código. Se trata de aplicaciones típicas que cumplen

funcionalidades diversas y sirven como punto de partida o referencia para el programador.

Dos ejemplos de código que resultaron sumamente oportunos para el proyecto por implementar funcionalidades similares a las buscadas son:

- RPL Border Router: Home/contiki/examples/rpl-border-router
- Er Rest Example: Home/contiki/examples/er-rest-example

El ejemplo RPL border router brinda una aplicación que detecta automáticamente los vecinos RPL y forma una estructura DODAG (red mesh). Además de cumplir con las funciones de configuración automática de la red, brinda acceso a la HAN a través de un puerto serial. Es decir que conectando un PC mediante puerto serial al mote que corre este programa, se tiene acceso a todos los nodos gracias a sus funciones de gateway. Una forma fácil de hacer esto es crear una tarjeta de red virtual en el PC mapeada al puerto serie. Una vez hecho esto, la PC tendrá acceso IP a cada uno de los nodos.

El rpl-border-router es además sumamente útil a la hora de realizar un debug a nivel RPL. Una de las razones de esto es que ofrece al usuario una sencilla página web donde refleja todas sus rutas y vecinos detectados. El código fuente del rpl-border-router se encuentra en:

Home/contiki/examples/rpl-border-router/border-router.c

El ejemplo *Er Rest Example* brinda un punto de partida sumamente interesante para trabajar con CoAP, ya que ofrece ejemplos de código fuente tanto para un servidor REST como para un cliente. Además, brinda una serie de recursos típicos que pueden resultar útiles para una gran variedad de aplicaciones. Este ejemplo ofrece además una simulación en Cooja [4]previamente configurada, donde se implementa una estructura cliente-servidor.

El ejemplo *Er Rest Example* ofrece los siguiente códigos fuente básicos y simulaciones Cooja:

- Home/contiki/examples/er-rest-example/er-example-client.c
- Home/contiki/examples/er-rest-example/er-example-server.c
- Home/contiki/examples/er-rest-example/er-server-client.csc

Existen varios recursos CoAP disponibles como ejemplos. Todos ellos se ofrecen como parte del ejemplo “er-rest-example”, en la ubicación:

Home/contiki/examples/er-rest-example/resources

Dentro de este directorio se encuentra unos 35 recursos disponibles. De estos ejemplos, se tomaron como referencia dos, que fueron adaptados para obtener los dos recursos diseñados para el sistema:

[4]Ver Anexo A.

- *res-push.c (obs)*: Se trata de un recurso observable, que permite realizar una tarea periódica y enviar la actualización de la información a los clientes suscritos cada un período de tiempo configurado.
- *res-leds.c (put)*: Se trata de un ejemplo típico para el uso de un recurso a través de la función PUT. En este caso particular se activan los leds del mote de acuerdo al valor de un parámetro incluido en el mensaje de consigna enviado desde el cliente.

A.3.2. Programación a partir de los ejemplos disponibles

En el caso del Border Router, los cambios a nivel de parámetros de funcionamiento respecto al ejemplo rpl-border-router se reflejan en el archivo project-conf.h.

En el caso del software de los nodos, el uso de los recursos en CoAP está inspirado en el ejemplo er-example-server.c, sin embargo el programa es original.

Los recursos res-mediciones-SG.c y res-comando-SG.c fueron programados en base a los ejemplos res-push.c y res-leds.c. A continuación se brinda una descripción de dichos recursos.

Descripción de los recursos CoAP desarrollados

res-mediciones-SG.c:

Se trata de un recurso *observable*. Su contenido es actualizado y enviado automáticamente a todos los clientes suscritos cada un tiempo configurable, que ha sido seteado en 5 segundos. Además del modo OBSERVE, también acepta la directiva GET en caso de que el cliente quiera consultarlo por iniciativa propia en un momento dado.

Este recurso ofrece como respuesta un conjunto de 6 valores enteros separados por comas. En la Tabla A.1 se muestra cada uno de estos valores con sus respectivos formatos y unidades.

Campo	**Valor reportado**	**Unidades**	**Descripción**
1	[datos.Vrms x 10]	V x 10	Tensión RMS
2	[datos.Irms x 1000]	mA	Corriente RMS
3	[datos.P x 10]	W x 10	Potencia Activa
4	[datos.Q x 10]	VAR x 10	Potencia Reactiva
5	status_relays		Estado de los relays (00, 0F, F0 o FF)
6	[datos.Vref x 1000]	mV	Tensión de referencia placa acondicionadora

Tabla A.1: Variables reportadas por el recurso res-mediciones-SG.

NOTA: Debe tenerse en cuenta que en CoAP los valores son enviados en formato de texto.

El significado de las variables *status_relays* y *datos.Vref* adquirirá sentido al leer el Capítulo 5. En el caso del estado de los relays, la variable status_relays está compuesta por un byte con la codificación mostrada en la Tabla A.2.

Valor status_relays (Hex)	**Estado Relay 1**	**Estado Relay 0**
00	Apagado	Apagado
0F	Apagado	Encendido
F0	Encendido	Apagado
FF	Encendido	Encendido

Tabla A.2: Codificación de estado de relays mediante variable status_relays.

res-comando-SG.c:

Se trata de un recurso que permite el uso de los métodos PUT y POST. Esto significa que el cliente puede escribir los valores de este recurso. Cuando un cliente (particularmente el HEC) escribe un nuevo valor en este recurso, el programa se interrumpe y se llama inmediatamente al proceso encargado del comando de los relés, activándolos o desactivándolos de acuerdo al valor del parámetro escrito por el cliente.

En la Tabla A.3 se describen los comandos aceptados y la función de cada uno de ellos. En la columna "Comando" se señala el contenido exacto a nivel de throughput del mensaje CoAP.

Comando (string)	**Comando Relay 1**	**Comando Relay 0**
comando_relays = 0	Apagar	Apagar
comando_relays = 1	Apagar	Encender
comando_relays = 2	Encender	Apagar
comando_relays = 3	Encender	Encender

Tabla A.3: Comando de relays a través de CoAP mediante variable comando_relays.

A.3.3. Configuración definitiva de parámetros

En la Tabla A.4 se incluyen los parámetros de configuración de la comunicación a nivel de Contiki incluyendo sus valores originales y los nuevos valores configurados.

A.3.4. Resumen de resultados y configuración final

En la Tabla A.5 se resumen los parámetros básicos de funcionamiento del stack a nivel de Contiki.

Parámetro Contiki	**Valor original**	**NODOS-Nuevo Valor**	**BORDER ROUTER/Nuevo Valor**
COLLECT_NBR_TABLE_CONF_MAX_NEIGHBORS	32	4	10
NBR_TABLE_CONF_MAX_NEIGHBORS	20	4	4
UIP_CONF_MAX_ROUTES	20	4	10
QUEUEBUF_CONF_NUM	8	4	8
NETSTACK_CONF_RDC	contikimac_driver	nullrdc_driver	nullrdc_driver

Tabla A.4: Parámetros de configuración a nivel de Contiki

A.3.5. Descripción del código

A continuación se brinda una breve descripción del código C programado en los nodos para la implementación de las funcionalidades de comunicación descritas.

En la sección A.2.1 se explicó la forma en que el programa se divide en diferentes archivos de código fuente. En la presente sección se explicarán los detalles de implementación de aquellas partes del programa que ejecuten funciones de comunicación. A saber:
.../examples/smart-grid/er-rest-example:

- *er-server-SG.c*: Programa principal
- *variables_globales.h*: Variables globales del programa
- *project-conf.h*: Configuración de Contiki y stacks de comunicación

.../examples/smart-grid/er-rest-example/resources:

- *res-mediciones-SG.c*: Recurso CoAP para reporte de medidas
- *res-comando-SG.c* Recurso CoAP para comando de relés

.../examples/smart-grid/rpl-border-router:

- *border-router-SG.c*: Programa principal
- *project-conf.h*: Configuración de Contiki y stacks de comunicación

Programa principal

El programa principal de los nodos se encuentra en el fuente er-server-SG.c. Este programa se encarga del manejo de los recursos CoAP a partir de la activación del motor REST. Esto se hace mediante la siguiente instrucción:

```
1 /* Initialize the REST engine. */
2 rest_init_engine();
```

Todo el resto del stack es manejado automáticamente por Contiki en base a los parámetros de configuración especificados en project-conf.h que ya fueron mencionados en secciones anteriores.

Al ejecutar la directiva anterior, el programa ejecutará todos aquellos recursos que hayan sido previamente declarados y activados. Para que la declaración y activación funcionen correctamente, todos los fuentes de los recursos deben estar ubicados en el subdirectorio “resources”. La declaración y activación de los recursos se hace de la siguiente forma:

```
1 extern resource_t res_mediciones_SG;
2 extern resource_t res_comando_SG;
3
4 rest_activate_resource(&res_mediciones_SG, "Reportes/Mediciones");
5 rest_activate_resource(&res_comando_SG, "Comandos/Relays");
```

Mediante las instrucciones anteriores se está declarando y activando los recursos *res-mediciones-SG* y *res-comando-SG*. Para ambos se especifica una ruta de acceso a cada recurso. Esta ruta debe ser especificada por el cliente a la hora de acceder a los mismos.

Hecho esto, ambos recursos quedan activados, funcionando y disponibles para cualquier cliente en la HAN. Sin embargo, es necesario programar cada uno de estos recursos para que ofrezcan la información y funcionalidad deseada. A continuación se explica la implementación de cada uno de ellos.

Recurso Mediciones

Tal como se explicó en secciones anteriores, el recurso *res_mediciones_SG* es un recurso *observable* que ofrece como respuesta un conjunto de 6 valores enteros separados por comas.

En el archivo *res-mediciones-SG.c* se encuentra declarado y programado este recurso. La declaración se realiza como se muestra a continuación, definiendo el nombre y los parámetros básicos del recurso.

```
PERIODIC_RESOURCE(res_mediciones_SG,
                  "title=\"Reporte de mediciones\";obs",
                  res_get_handler,
                  NULL,
                  NULL,
                  NULL,
                  5 * CLOCK_SECOND,
                  res_periodic_handler);
```

Para definir el tipo de funcionalidades que el recurso ofrecerá, se deben configurar diferentes *handlers* que responden a distintos eventos o solicitudes. En este caso lo más importante es definir la función *res_get_handler* que será la que determine la respuesta del recurso ante una llamada GET u OBSERVE. A continuación se muestran los principales fragmentos de esta función.

```
static void
res_get_handler(void *request,
void *response,
uint8_t *buffer,
uint16_t preferred_size,
int32_t *offset){

REST.set_response_payload(response,
buffer, snprintf((char *)buffer,
preferred_size,
"%d,%d,%d,%d,%d,%d",
(int)(datos.Vrms*10),
(int)(datos.Irms*1000),
(int)(datos.p*10),
(int)(datos.q*10),
status_relays,
(int)(datos.Vref*1000)));
}
```

Mediante esta función *REST.set_response_payload* se define el contenido de la respuesta del recurso, incluyendo el valor de las variables *datos.Vrms, datos.Irms, datos.p, datos.q, status_relays, datos.Vref.*

Recurso Comando de Relés

Al igual que en el caso anterior, este recurso es declarado y configurado en su correspondiente archivo fuente, en este caso *res-comando-SG.c*. A continuación se

muestra la definición del recurso.

```
RESOURCE(res_comando_SG,
            "title=\"Comando\";rt=\"Comando de relays\"",
            NULL,
            res_post_put_handler,
            res_post_put_handler,
            NULL);
```

A diferencia del caso anterior en el cual el recurso ofrecía información ante solicitudes del tipo GET u OBSERVE, la función de este recurso es la de recibir información mediante los métodos POST o PUT ejecutados por un cliente. Por tal motivo lo más importante a la hora de la programación de este recurso es definir la función *res_post_put_handler*. A continuación se muestra la programación de esta función:

```
static void
res_post_put_handler(void *request,
void *response,
uint8_t *buffer,
uint16_t preferred_size,
int32_t *offset){
 const char *recibido = NULL;

 REST.get_post_variable(request, "comando_relays", &recibido);
 PRINTF("Recibido comando relays: %s\n", recibido);
 PRINTF("\n%d\n",*recibido);

 //Recibo un caracter a trav\'{e}s de CoAP y lo codifico de la siguiente forma:
 if (*recibido == '0'){
     comando_relays = 0x00;
     process_poll(&control_carga);
 }
 else if (*recibido == '1'){
     comando_relays = 0x0F;
     process_poll(&control_carga);
 }
 else if (*recibido == '2'){
     comando_relays = 0xF0;
     process_poll(&control_carga);
 }
 else if (*recibido == '3'){
     comando_relays = 0xFF;
     process_poll(&control_carga);
 }
 else{
     PRINTF("Error - Comando incorrecto");
 }

}
```

Mediante la función anterior se recibe la información enviada por un cliente en formato de texto, la misma se aloja en un área de memoria referenciada por el puntero *recibido*, y luego se decodifica para interpretarla como un conjunto de 4 posibles comandos para activar o desactivar los dos relés[5] disponibles en cada Nodo.

La información decodificada se almacena en la variable *comando_relays* que es luego utilizada por el programa principal para activar o desactivar los relés.

Apenas se decodifica el mensaje se llama a la rutina de manejo de relés mediante una interrupción ejecutando *process_poll(&control_carga)* para que dicha rutina realice la acción de comando correspondiente en forma inmediata.

[5] Ver capítulo 5

Programa del Border Router

No se realizaron cambios a nivel de este programa respecto del programa ejemplo *border-router.c* mencionado en secciones anteriores. Únicamente se modificaron los parámetros en su correspondiente *project-conf.h* de acuerdo a lo explicado en la sección A.3.3.

A.4. Software de los nodos; medición y comando

A.4.1. Proceso de medición

En la presente sección se brindan detalles acerca del programa que ejecuta las funciones de la medición. Se recomienda la previa lectura de la descripción general del programa brindada en el Capítulo "Diseño detallado e implementación de los nodos" 5.5.

A continuación se muestra un fragmento de la función utilizada para realizar el muestreo. La sintaxis para utilizar los *rtimers* no es sencilla, pero el principio general del funcionamiento sí lo es: la funcion *muestreo* es la encargada de tomar las muestras de las señales. Lo que se hace mediante la función *rtimer_set()* al comienzo del código es *agendar* una interrupción por tiempo a ejecutarse dentro un tiempo equivalente al período de muestreo (RTIMER_NOW() + PERIODO_MUESTREO). Luego se realiza la lectura de los sensores, capturando el valor de tensión presente en los ADC y guardando las muestras de cada una de las tres señales en sus registros correspondientes. Es conveniente observar que cuando se agenda la próxima interrupción, se indica que la misma debe ser atendida mediante la función *muestreo*, que es la función desde donde se está programando la interrupción. Lo que sucede es que la primera vez que se llama a la función *muestreo* (primera muestra) la invocación se hace desde el cuerpo del proceso principal (proceso *medicion*). Luego, una vez capturada la primera muestra, es la propia función *muestreo* la que se auto-agenda para atender futuras interrupciones (i.e. tomar futuras muestras). Este proceso se repite hasta llegar a la última muestra (condición *muestra* < *TAM_VENTANA-1*), situación en la cual la función en lugar de reprogramar una nueva interrupción ejecuta una llamada al proceso principal, permitiéndole continuar con su ejecución *process_poll(&medicion)*.

```
1  static char muestreo(struct rtimer *rt, void* ptr){
2      if (muestra < TAM_VENTANA-1){
3          uint8_t ret;
4          ret = rtimer_set(
5  &t_muestreo,
6  RTIMER_NOW()+PERIODO_MUESTREO,
7  1,
8  (void (*)(struct rtimer *, void *))muestreo, NULL);
9          if(ret){ PRINT_STD("Error Timer: %u\n", ret); }
10     }
11     else {
12         process_poll(&medicion);
13     }
14     //Se realiza la lectura de los diferentes sensores.
15 //Esto es, se toman las muestras:
16     buff_C[muestra] = sensores.value(SENSOR_CORRIENTE);
17     buff_V[muestra] = sensores.value(SENSOR_VOLTAJE);
18     buff_Vref[muestra] = sensores.value(SENSOR_VREF);
19     muestra++; //Incremento la cuenta de muestra actual
20     return 1;
21 }
```

Para poder utilizar los ADC correctamente los mismos son configurados mediante el código ubicado en los archivos *z1-ADCs-SG.h* y *z1-ADCs-SG.c*. Aquí se configura entre otras cosas la referencia interna en 2.5V definida en la sección 5.3.4. A continuación se muestran los parámetros de funcionamiento de los ADC en su configuración final.

```
static void
sensors_activate(uint8_t type)
{
uint8_t pre = adc_on;

adc_on |= type;

if(pre == 0 && adc_on > 0) {
    P6DIR = 0xff;
    P6OUT = 0x00;
    P6SEL |= 0x0b; // bit 3 + 1 + 0

    // if nothing was started before, start up the ADC system
    // Set up the ADC.
// Setup ADC12, ref., sampling time
    ADC12CTL0 = REF2_5V + SHT0_5 + SHT1_5 + MSC;
    // Use sampling timer, repeat-sequenc-of-channels
    ADC12CTL1 = SHP + CONSEQ_3 + CSTARTADD_0;
    // convert up to MEM4
    ADC12MCTL4 |= EOS;

    ADC12CTL0 |= ADC12ON + REFON;
    ADC12CTL0 |= ENC;         // enable conversion
    ADC12CTL0 |= ADC12SC;         // sample & convert
}
```

A.4.2. Comando de carga

A continuación se muestra un fragmento del código del proceso encargado del comando de los relés. Se observa que lo primero que se hace durante la ejecución inicial (start-up del nodo) es configurar y habilitar las salidas digitales del MCU utilizadas para el comando. Luego, se entra en un loop donde se espera la llamada externa indicando que se recibió un mensaje de comando, y finalmente si esto ocurre se procesa el mensaje recibido en la variable *comando_relays* y se comandan los relés en base a este.

```
PROCESS_BEGIN();

//Habilito salidas para ambos relays
P6SEL &= ~(1 << P6_DO_RELAY_0);
P6SEL &= ~(1 << P6_DO_RELAY_1);
P6DIR |= (1 << P6_DO_RELAY_0);
P6DIR |= (1 << P6_DO_RELAY_1);

//Arranco con ambos relays apagados
P6OUT &= ~(1 << P6_DO_RELAY_0);
P6OUT &= ~(1 << P6_DO_RELAY_1);
status_relays = 0x00;

while(1) {
    //Llamada desde recurso CoAP indicando que se recibi\'{o} un comando
     PROCESS_WAIT_UNTIL(ev == PROCESS_EVENT_POLL);
     ev = PROCESS_EVENT_NONE;

     //COMANDO DE RELAY 0
     if ((comando_relays & 0x0F) > 0){
         P6OUT |= (1 << P6_DO_RELAY_0); //Enciendo relay 0
         status_relays |= 0x0F; //Actualizo estado: relay 0 encendido
     }
     else{
         P6OUT &= ~(1 << P6_DO_RELAY_0); //Apago relay 0
         status_relays &= 0xF0; //Actualizo estado: relay 0 apagado
     }

     //COMANDO DE RELAY 1
     if ((comando_relays & 0xF0) > 0){
         P6OUT |= (1 << P6_DO_RELAY_1); //Enciendo relay 1
```

```
32          status_relays |= 0xF0; //Actualizo estado: relay 1 encendido
33      }
34      else{
35          P6OUT &= ~(1 << P6_DO_RELAY_1); //Apago relay 1
36          status_relays &= 0x0F; //Actualizo estado: relay 1 apagado
37      }
38
39      //Monitoreo estado y se\~nalizo con LEDs
40      LED AZUL = ESTADO R0 | LED VERDE = ESTADO R1
41      if ((status_relays & 0x0F) > 0){
42          leds_on(LEDS_BLUE);
43      }
44      else{
45          leds_off(LEDS_BLUE);
46      }
47      if ((status_relays & 0xF0) > 0){
48          leds_on(LEDS_GREEN);
49      }
50      else{
51          leds_off(LEDS_GREEN);
52      }
53
54  } //End while
55  PROCESS_END();
56  }
```

Capa		**Nodos**	**Border Router**
CoAP	Recurso re-mediciones-SG	Servidor	No implementa
	Recurso res-comando-SG	Servidor	No implementa
Transporte	TCP	No	No
	UDP	Si	Si
RPL	Root	No	SÅ
	Objective function	MRHOF (ETX)	MRHOF (ETX)
	Modo de funcionamiento	4 (rutea)	4 (rutea)
	Max rutas	4	10
IPv6	Direcciones	aaaa:0:0:0:c30c:0:0:000n	aaaa:0:0:0:c30c:0:0:000n
6LoWPAN	Max. Cant. Vecinos	4	10
IEEE 802.15.4	Frecuencia	2.4GHz	2.4GHz
	Tipo de dispositivo	FFD	FFD
	RDC	NO	NO
	Acceso al medio	CSMA	CSMA
Hardware	Modelo Mote	Zolertia Z1	Zolertia Z1

Tabla A.5: Parámetros de configuración del stack de comunicación en Contiki.

Apéndice B

Detalles de diseño e implementación de los nodos

B.1. Descripción de Hardware

B.1.1. Z1

Los motes Z1 (Fig. B.1) son módulos diseñados para facilitar el desarrollo de aplicaciones de *Wireless Sensor Network* (WSN). Se pondrá foco únicamente en aquellas características del sistema que están estrechamente ligadas al diseño de la aplicación en cuestión.

Figura B.1: Fotografía del mote Z1 de Zolertia [8].

En la Figura B.2 se observa un diagrama de bloques del mote Z1. De los bloques que componen el sistema, los más importantes para el diseño son los siguientes:

- **Microcontrolador (MCU)**. Es el núcleo del módulo y se encarga de almacenar y procesar el programa de usuario, en particular Contiki y sus funciones de comunicación.

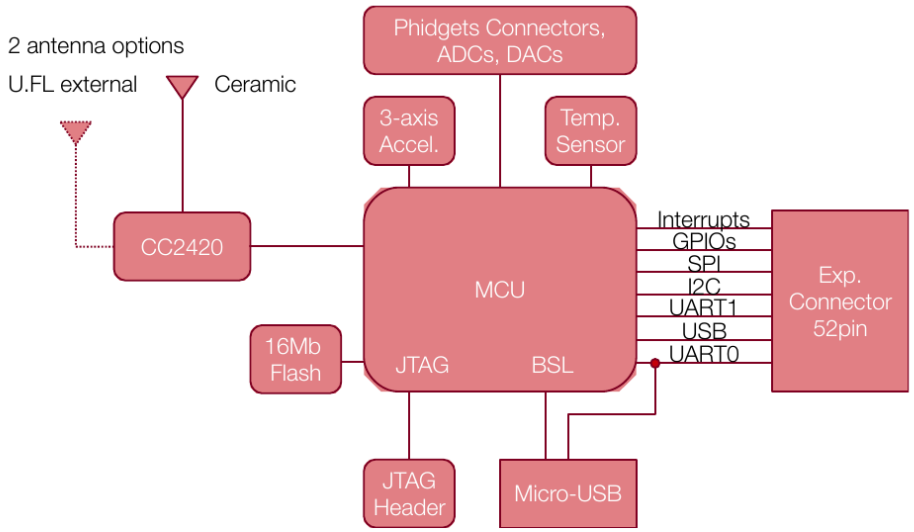


Figura B.2: Diagrama en bloques del Zolertia Z1 [8].

- **ADCś y DACś**. El microcontrolador cuenta con una serie de *Analog to Digital Converter* (ADC) y *Digital to Analog Converter* (DAC) disponibles. El mote Z1 ofrece interfaces para estos puertos incluyendo conectores del tipo *phidget* [8] para facilitar la conexión.
- **Radio IEEE 802.15.4 2.4 GHz (CC2420)**. Este módulo es el que brinda conectividad con la HAN a nivel de radio.
- **Puerto MicroUSB**. Z1 ofrece un puerto micro USB que facilita el desarrollo al proveer una interfaz estándar de comunicación con PC para programación y debug.

Todos estos módulos vienen integrados en el mote Z1, simplificando al usuario las tareas de integración.

La configuración de comunicación a través de la radio utilizando IEEE 802.15.4 ha sido explicada en el Capítulo 4 por lo cual no será abordada aquí. Se pondrá foco en cambio sobre el microprocesador y sus componentes, prestando particular atención a los ADCs, DACs y *General Purpose Input Output* (GPIO), que brindará la interfaz de hardware para poder interactuar con los sensores y actuadores de los nodos.

El microprocesador utilizado en los motes Z1 es el MSP430F2617 de la familia MSP430 de Texas Instruments. Las características principales de este microprocesador son las siguientes:

- Rango de alimentación 1.8 V DC to 3.6 V DC.
- 16-Bit Arquitectura RISC, 62.5-ns Tiempo de Ciclo de Instrucción.
- Conversores analógico/digital tipo SAR con referencia interna, 12-Bit de resolución, con funciones Sample-and-Hold, y Autoscan.

- Conversores digital/analógico Dual 12-Bit con sincronización.
- Memoria Flash: 92KB + 256B. Memoria RAM: 8KB.

A la hora de integrar los motes Z1 con el resto del hardware que conforma los nodos (sensores y actuadores) los puntos críticos a estudiar son la alimentación, los ADCs y los pines de salida lógica (GPIO), ya que en estos casos es necesario asegurar la compatibilidad de los distintos componentes en lo que refiere a tensiones, impedancias y rangos de señal.

Alimentación del Z1 y el MCU

El Z1 prevé dos fuentes de alimentación estándar:

- 2 baterías AAA o fuente de 3VDC
- Alimentación por USB 5VDC

En el caso de alimentar el mote a través del USB, el puerto genera internamente un nivel de alimentación de 3VDC para el resto de los bloques. Por esto es indistinto para el MCU y los demás bloques si se utiliza alimentación de 3VDC o 5VDC, ya que estos siempre recibirán 3VDC. Una diferencia de usar alimentación por USB es que permite utilizar ciertos pines del mote para sensores que trabajen con nivel de tensión de 5VDC.

Funcionamiento de los ADC's y pines de salidas lógicas

Las salidas lógicas no presentan un gran desafío a nivel de integración ya que su funcionamiento es muy simple; son capaces de mantener en el pin correspondiente una señal lógica alta o baja, de valor Vss o Vcc respectivamente [6]. El detalle más importante a tener en cuenta a la hora de su utilización es que estos pines no son capaces de suministrar grandes valores de corriente (la tensión cae drásticamente para corrientes en el orden de 20mA [6]), por lo cual para utilizar estas señales en otro dispositivo, este último debe tener *entradas lógicas* de alta impedancia de modo de no drenar corriente en exceso de los pines de salida del MCU.

Las características de los ADCs hacen que resulte interesante dedicar algunos párrafos para describir sus modos de funcionamiento, detallando los parámetros más relevantes para la aplicación que se está implementando. En la Fig. B.3 se muestra un diagrama en bloques de los ADC, incluyendo los diferentes parámetros de funcionamiento, algunos de los cuales serán explicados en los próximos párrafos.

Se dispone de 8 ADCs (A0 .. A8), pudiendo algunos de ellos ser configurados para trabajar como DAC o incluso como GPIO como funcionalidad auxiliar, pero a los efectos introducir su funcionamiento para esta aplicación se trata a cada uno de ellos por igual en esta sección y solo se describe el funcionamiento en modo ADC.

El primer parámetro de funcionamiento de los ADC a tener en cuenta son los *Absolute Maximun Ratings*, que son los niveles máximos de tensión que se pueden presentar en los pines de señal sin dañar el MCU. Estos son:

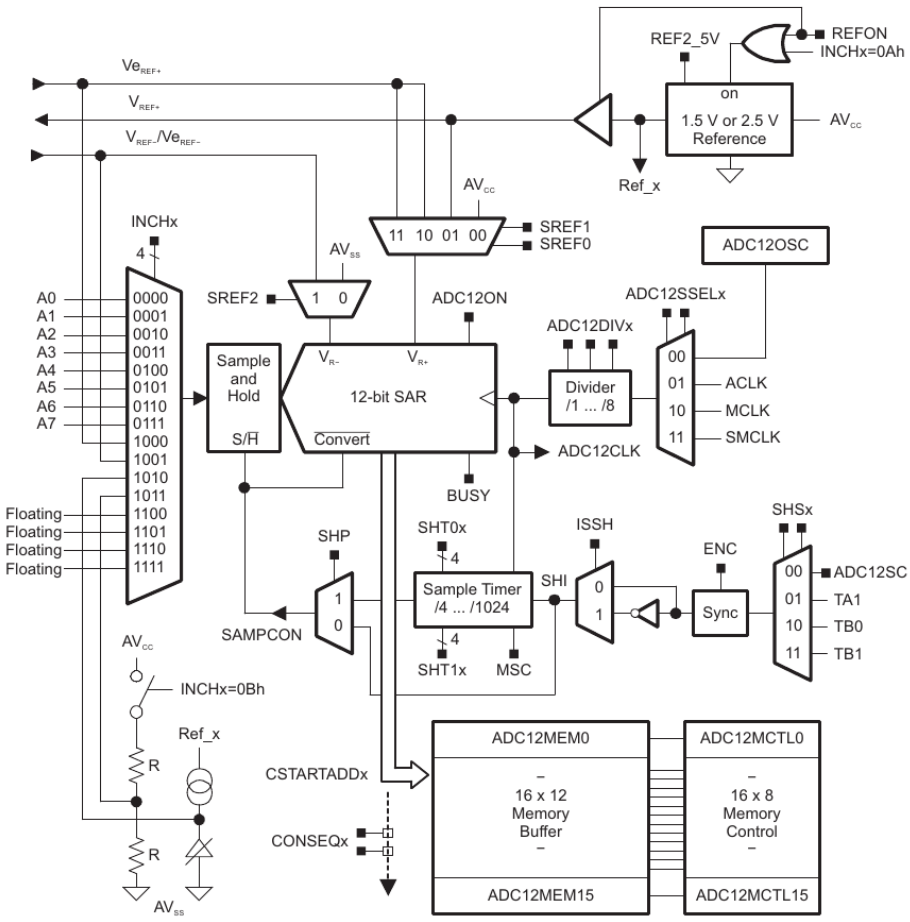


Figura B.3: Diagrama en bloques de los ADC del MCU [7].

- Mínima tensión aplicada a cualquier pin (en particular ADCs): -0.3V
- Máxima tensión aplicada a cualquier pin (en particular ADCs): Vcc + 0.3V

Otro parámetro de funcionamiento importante es la referencia; para convertir las señales analógicas y representarlas en formato digital, el ADC necesita una referencia de tensión contra la cual comparar el nivel de las señales. El valor de cada muestra se calcula mediante la ecuación (B.1), donde V_{IN} es el nivel de la señal y V_{R+} y V_{R-} son los límites superior e inferior de la conversión. El ADC posee una resolución de 12 bits y por lo tanto podrá representar cada muestra con un conjunto de 4096 valores diferentes (entre 0 y 4095). V_{R+} y V_{R-} definen el rango de conversión. A cualquier nivel de señal que esté por encima o por debajo de este rango, se le asignará el valor 4095 ó 0 respectivamente [7].

$$N_{ADC} = 4095.\frac{V_{IN} - V_{R-}}{V_{R+} - V_{R-}} \tag{B.1}$$

Estos valores de referencia consisten en valores reales de tensión, que deben estar presentes en el MCU a la hora de la conversión para poder realizar las compa-

raciones en forma electrónica. Existen diferentes formas de generar estas tensiones de referencia, pudiendo ser fijadas internamente por el MCU, o ser tomadas de una fuente externa. Esta selección se realiza por software mediante los parámetros SREF0 y SREF1 y SREF2 mostrados en el esquema.

A la hora de utilizar una referencia interna existen dos opciones; 1,5V y 2,5V. La activación y selección del nivel de referencia interna se hace mediante los parámetros *REFON* y *REF*2_5V respectivamente. En la Tabla B.1 se resumen las opciones mencionadas. En la Sec. 5.3.4 se explica la opción elegida para la aplicación.

SREF2	**SREF1**	**SREF0**	**Rango conversión ([Vr- .. Vr+])**	**Comentario**
0	0	0	[Vss .. Vcc]	
0	0	1	[Vss .. Ref_X]	Ref_X puede ser 1,5 o 2,5V dependiendo de REF2_5
0	1	0	[Vss .. Veref+]	Veref+ es una referencia externa
0	1	1	[Vss .. Veref+]]	Veref+ es una referencia externa
1	0	0	[Veref- .. Vcc]	Veref- es una referencia externa
1	0	1	[Veref- .. Ref_X]	Ref_X puede ser 1,5 o 2,5V dependiendo de REF2_5
1	1	0	[Veref- .. Veref+]	Veref- y Veref+ son referencias externas
1	1	1	[Veref- .. Veref+]]	Veref- y Veref+ son referencias externas

Tabla B.1: Parámetros de configuración de ADC.

En la Figura B.4 se muestran las variables antes mencionadas. El área de conversión se muestra en verde, delimitada por las referencias superior e inferior, y en amarillo se muestran los niveles fuera del área de conversión pero admitidos por el ADC. Las áreas rojas muestran aquellos niveles de tensión en los cuales hay riesgos de dañar el ADC.

Hasta ahora se ha descrito el funcionamiento de los ADC del MCU. Sin embargo, hay que tener en cuenta que al trabajar con el mote Z1 algunos pines de los ADC no están directamente disponibles, si no que el Z1 implementa etapas de pre-acondicionamiento de señal que deben ser tenidas en cuenta.

En la Figura B.5 se muestra un esquema del circuito del puerto de ADCs en el Z1. Este puerto se denomina JP1A. Los terminales ADC del MCU se representan con las etiquetas “AD 0” a “AD 7”. Se observa entonces que el “AD 0” y el “AD 3”, a diferencia de los demás, no se encuentran disponibles directamente en el puerto, sino que las señales inyectadas a la entrada del mismo (pines 5 y 15 de

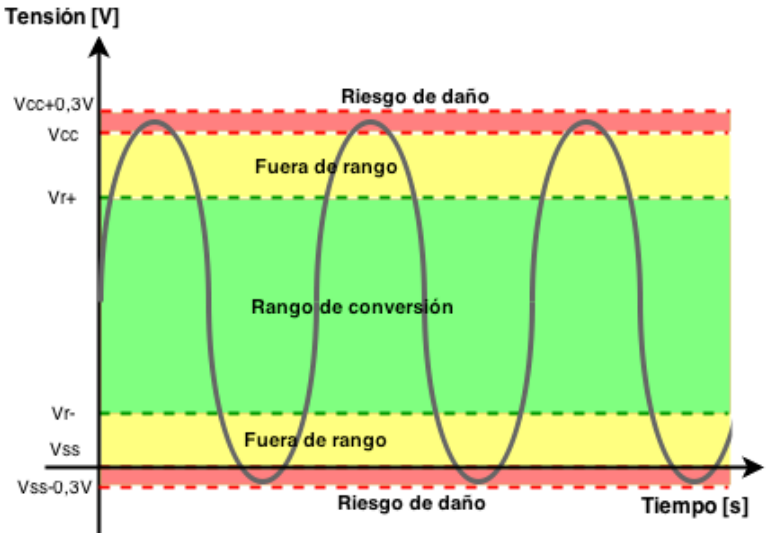


Figura B.4: Rangos de funcionamiento de los ADC.

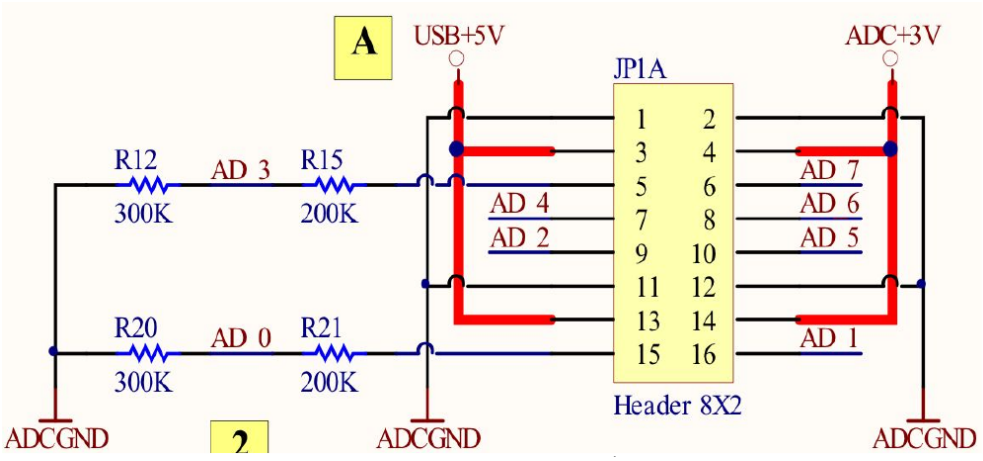


Figura B.5: Puerto JP1A del Zolertia Z1 [8].

JP1A) pasan primero por un divisor resistivo. En ambos casos, el divisor resistivo atenúa el nivel de la señal en un factor de 3/5 como se muestra en las ecuaciones (B.2) y (B.3):

$$V_{AD0} = \frac{R_{20}}{R_{20} + R_{21}}.V_{15} = \frac{3.V_{15}}{5} \tag{B.2}$$

$$V_{AD3} = \frac{R_{12}}{R_{12} + R_{15}}.V_{5} = \frac{3.V_{5}}{5} \tag{B.3}$$

La razón por la que Z1 implementa estos divisores resistivos es para permitir el uso de sensores con señales en el rango de 0 a 5V (para Vcc = 3V). Es decir, utilizando los divisores resistivos se logra ampliar el rango efectivo de señal admitido a la entrada del mote por un factor de 5/3 con respecto al rango del ADC

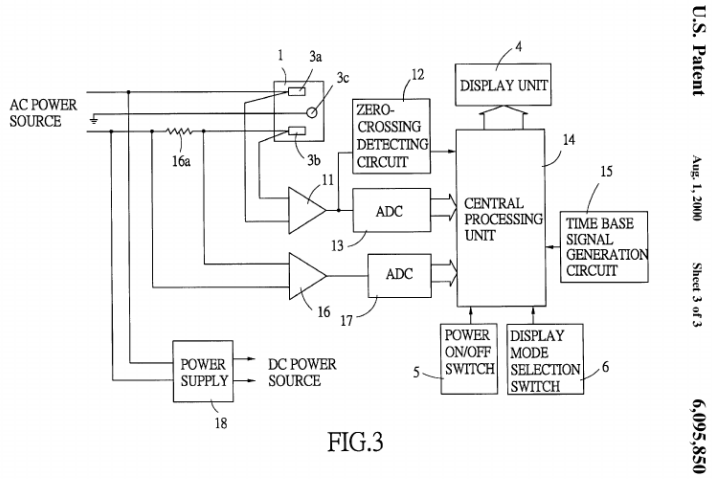


Figura B.6: Esquema descriptivo de la patente US6095850.

del MCU. Concretamente, si el MCU se encuentra alimentado con Vcc=3V, los puertos 5 y 15 de JP1A permiten inyectar señales de hasta 5V sin dañar el ADC. Esta posibilidad de utilizar un rango ampliado es muy práctica a la hora de utilizar algunos sensores phidget, y por eso Z1 dispone de dos puertos para tal fin. Como se verá más adelante, esta configuración será aprovechada a la hora del diseño.

Los parámetros de funcionamiento mencionados brindan ciertas libertades pero a la vez imponen restricciones en cuanto a las características de la señal de entrada. Esto debe ser tenido en cuenta en el diseño de las etapas de señal previas al muestreo, de modo de asegurar rangos válidos de señal y el funcionamiento seguro de los ADC. En la sección 5.3 se explica con detalle el diseño completo de los nodos en base a las características anteriormente explicadas.

B.1.2. KillAWatt

Como se detalla en la Figura B.6, el *KillAWatt P4400*[1] se compone básicamente de cuatro bloques: un bloque de potencia (representado con "1" en el esquema), un bloque de medición y acondicionamiento ("16a", "11" y "16" según esquema), un bloque de muestreo y procesamiento ("12","13","14" y "17" en el esquema) y un bloque de alimentación ("18" en el esquema).

Dado que la patente no presenta información detallada del diseño, para completar la identificación de los componentes se consultó material de terceros[2] [3] y se

[1]Protegida en USA bajo la patente US6095850, presentada en 1998. Como las patentes tienen aplicación territorial, y al no tener registro en Uruguay, no hay ninguna limitación en cuanto a la modificación del diseño así como su eventual producción en el territorio nacional.

[2]https://reindeerflotilla.wordpress.com/2011/08/23/the-kill-a-watt-not-all-versions-are-created-equal/

[3]http://www.mikesmicromania.com/2013/03/kill-watt-circuit-analysis.html

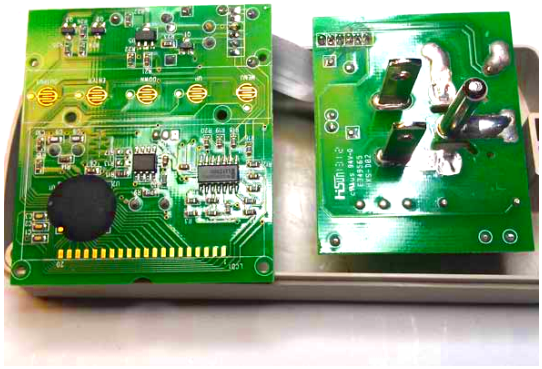


Figura B.7: KillAWatt P4400.

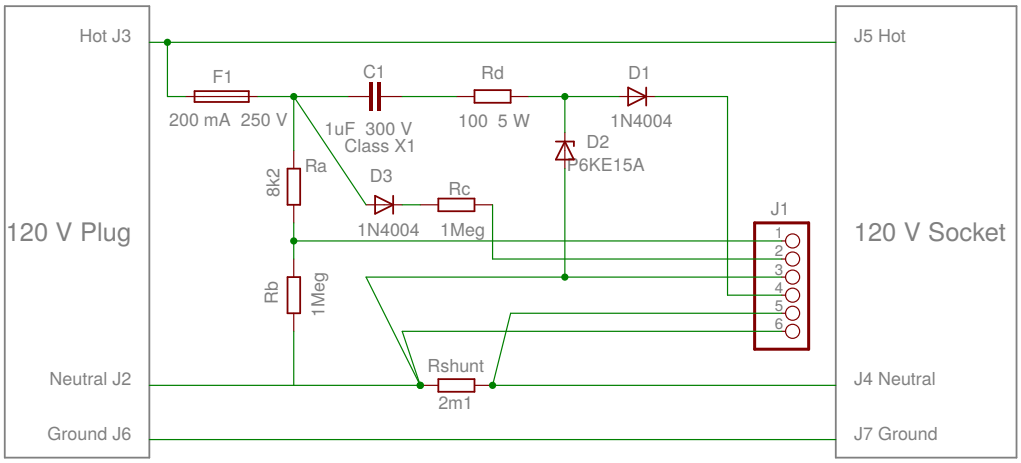


Figura B.8: Placa de potencia.

llevó a cabo una inspección visual del producto. Con toda la información recabada se logró obtener el esquemático general del dispositivo que se muestra en las Figs. B.8 B.9.

La placa de potencia es la primer etapa del sistema y se ubica entre la carga a medir, en adelante Z, y la red eléctrica. Se observa en primer lugar que el circuito cuenta con un fusible de protección de forma de evitar picos de corriente en la placa (F1).

La medida de corriente se implementa mediante el shunt R_{shunt}, ubicado entre uno de los terminales de la carga y el neutro de la red. La corriente consumida por la carga es proporcional al voltaje entre los bornes de esta resistencia (5,6). Por la Ley de Ohm se tiene que:

$$I_Z = V_{56}/R_{shunt} \tag{B.4}$$

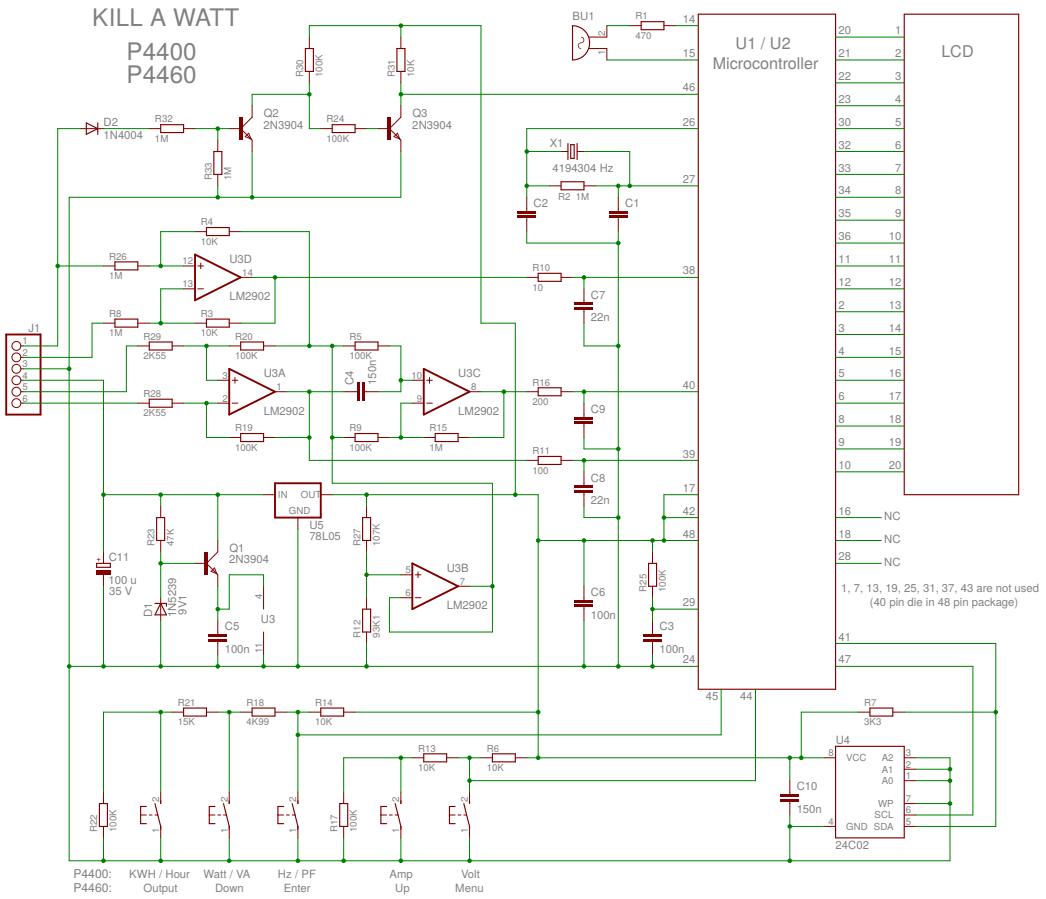


Figura B.9: Placa de acondicionamiento y procesamiento.

El voltaje en bornes de la resistencia shunt es amplificado en dos etapas de la siguiente manera:

$$V_{U3A1} = \left(\frac{R_{20}}{R_{29}}\right) V_{56} + V_{REF} \tag{B.5}$$

$$V_{U3C1} = \left(\frac{R_{20}.R_{15}}{R_{29}.R_9}\right) V_{56} + V_{REF} \tag{B.6}$$

De donde se obtiene:

$$\boxed{V_{i1} = \left(\frac{R_{20}}{R_{29}}\right) R_{shunt}.I_Z + V_{REF} = G_{i1}.I_Z + V_{REF}} \tag{B.7}$$

$$\boxed{V_{i2} = \left(\frac{R_{20}R_{15}}{R_{29}R_9}\right) R_{shunt}.I_Z + V_{REF} = G_{i2}.I_Z + V_{REF}} \tag{B.8}$$

Estas dos estapas dan lugar a dos canales de medición de corriente con distintas resoluciones, cuya elección se realizaría en timpo real (durante el procesamiento

de señales) en función de su amplitud. Esto es de forma de lograr un menor error de cuantización para señales pequeñas.

El voltaje de la red pasa por un atenuador resistivo y luego por un rectificador de media onda, como se muestra en la Figura B.10. Esta tensión es atenuada nuevamente por el amplificador operacional U3D. La señal de voltaje a la salida del mismo está dada por:

$$\boxed{V_v = \left[\frac{(1 + R_4/R_{26})R_b R_4}{(R_a + R_b)(R_{26} + R_b + R_a /\!/ R_b)}\right] V_{red} + V_{REF}} \tag{B.9}$$

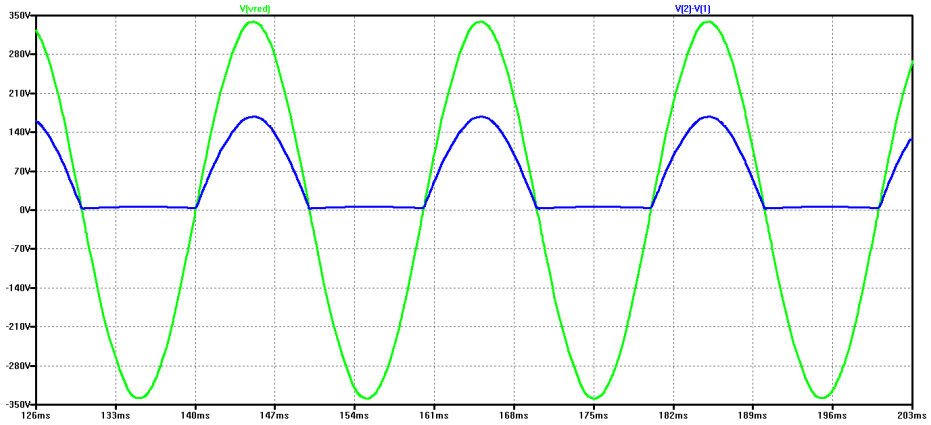


Figura B.10: Simulación de V_{red} y V_{21} en LTSpice.

Un detalle a destacar es que el voltaje de referencia del circuito se toma del terminal Neutro del tomacorrientes (diseño americano). En casos como el de la red uruguaya, donde el Neutro no está puesto a tierra, las señales están montadas sobre altos valores de tensión (típicamente valor de fase). En otras palabras, la referencia de la electrónica tiene una diferencia de potencial importante con respecto a la tierra de la red. Esto requiere extremar las precauciones al manipular el circuito.

La alimentación del bloque de acondicionamiento y procesamiento proviene de la propia red eléctrica, mediante un rectificador con filtro que luego es estabilizado por el diodo D1 de 9,1V. La alimentación de los amplificadores operacionales se toma de los bornes del capacitor C5. En este bloque también se genera el voltaje de referencia V_{REF} que será el componente de continua de las señales de salida a muestrear. El voltaje de alimentación y el voltaje de referencia están dados por:

$$\boxed{VCC = V_{D1} - V_{BE,Q1} \simeq 8{,}4V}$$

$$\boxed{V_{REF} = 5V.\frac{R_{12}}{R_{12} + R_{27}} \simeq 2{,}33V} \tag{B.10}$$

Para finalizar, el bloque de procesamiento incluye un microprocesador que tiene como entradas las señales antes mencionadas, un detector de cruces por cero así como un display para mostrar los cálculos realizados.

B.1.3. Filtro anti-aliasing

En el capítulo 3 se definió que las mediciones debían contemplar información de la señal incluyendo hasta el décimo armónico. En el caso de señales de 50Hz (frecuencia de la red eléctrica en el Uruguay) esto significa que el muestreo debe abarcar frecuencias de hasta 500Hz.

Según el Teorema de Muestreo de Nyquist, una señal de banda limitada puede ser reconstruida completamente si la frecuencia de muestreo es al menos el doble de su frecuencia máxima. Es decir que en este caso se debe utilizar una frecuencia de muestreo de al menos 1KHz. Por otra parte, para evitar el fenómeno de aliasing o solapamiento se debe asegurar que el ancho de banda de la señal analógica a muestrear sea acotado, siendo la frecuencia máxima de dicha señal menor que la mitad de la frecuencia de muestreo definida.

Sin embargo, la elección de la frecuencia de muestreo no resulta inmediata; si bien realizar un muestreo excesivo puede ser poco eficiente en términos de recursos, muchas veces realizar un sobremuestreo (muestrear a una frecuencia mayor a la de Nyquist) puede ser útil o conveniente para evitar la necesidad de implementar filtros anti-aliasing complejos. En la teoría, utilizando la frecuencia de Nyquist como frecuencia de muestreo y un filtro pasabajos ideal (selectividad infinita) con corte a la mitad de dicha frecuencia como filtro antialiasing funcionaría como método de muestreo. Sin embargo esto no es realizable en la práctica. A la hora de diseãr un filtro siempre existen compromisos entre selectividad, complejidad y distorsión en banda pasante (tanto en amplitud como en fase).

Por ejemplo, en el caso de la presente aplicación, elegir una frecuencia de muestreo de 1KHz implicaría la necesidad de contar con un filtro ideal con frecuencia de corte en 500Hz de modo de obtener medidas del décimo armónico sin distorsión. El no contar con un filtro ideal implicaría un compromiso; o bien se asegura un muy buen nivel de atenuación sobre los 500Hz, o bien se asegura una distorsión y atenuación muy bajas en la banda de 0 a 500Hz. Ambos no se pueden lograr simultáneamente.

Existen diferentes tipos de filtros tanto pasivos como activos, con distintas complejidades y características. Cuanto mayor sea la selectividad buscada, mayor será la complejidad requerida. Uno de los filtros pasabajos más sencillos es el conocido como “Filtro RC”, constituido simplemente por un resistor y un condensador en serie. Este filtro no ofrece una gran selectividad, pero resulta atractivo por la sencillez de su implementación.

Para contar con referencias al respecto se estudiaron algunos ejemplos de aplicación similares al presente sistema de medición. Algunos de los diseños consultados son [1, 3–5]. En todos ellos se utilizan filtros anti-aliasing de primer orden de tipo RC. Dado que el diseño de hardware está lejos de ser un objetivo o área de enfoque del presente trabajo, se opta en este sentido por adoptar el criterio de los diseños consultados y utilizar filtros RC.

Antes de pasar al diseño detallado del filtro, es conveniente repasar el comportamiento cualitativo del filtro RC y su configuración. En la Figura B.11 se muestra el circuito del filtro RC propuesto.

La transferencia del filtro viene dada por:

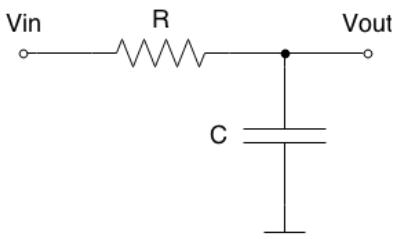


Figura B.11: Esquema del filtro RC.

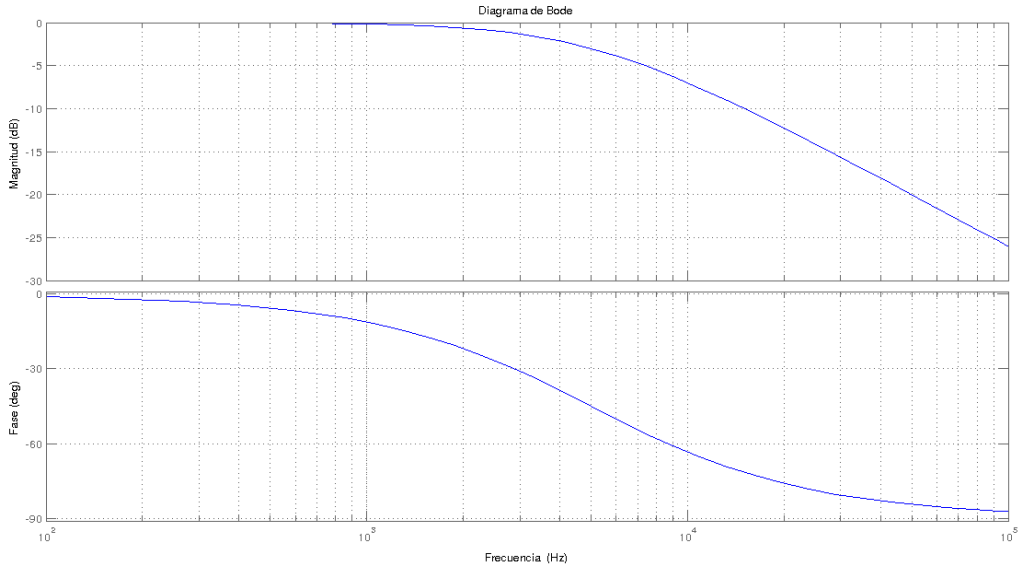


Figura B.12: Diagrama de Bode para el filtro RC con $fc = 5KHz$.

$$V_{out}(s) = \frac{1/Cs}{R + 1/Cs} V_{in}(s) = \frac{1}{1 + RCs} V_{in}(s) \tag{B.11}$$

La frecuencia de corte, que viene dada por la caída de -3dB en la ganancia, se calcula como:

$$fc = \frac{1}{2.\pi.R.C} \tag{B.12}$$

En la Figura B.12 se muestra un diagrama de bode del filtro RC con una frecuencia de corte de 5KHz.

Al diseñar el filtro RC hay que tener en cuenta algunos detalles a la hora de fijar la frecuencia de corte. Por un lado se debe asegurar una atenuación suficiente para cualquier frecuencia por arriba de fs/2, donde fs corresponde a la frecuencia de muestreo del ADC. Si la atenuación en esta banda no es suficiente, el sistema será susceptible a sufrir errores o distorsión por solapamiento. Por otra parte se

debe asegurar que en la banda de interés (0-500Hz) el filtro no introduzca distorsión significativa ni en amplitud ni en fase.

En el caso de un filtro RC la distorsión en amplitud vendría dada por el nivel de atenuación en banda pasante y sus efectos resultan evidentes para la aplicación en cuestión. En el caso de la distorsión de fase, la misma consiste en el desfasaje que el filtro introduce en la banda pasante. El desfasaje en sí no es un gran problema para las mediciones a realizar por los nodos siempre y cuando el mismotenga un comportamiento muy similar para la tensión y para la corriente. Si existiera un desapareamiento entre el desfasaje introducido para la tensión y la corriente, esto sería equivalente a introducir un desfasaje entre ellos y significaría una distorsión en la medida de la potencia.

Las tecnologías actuales de fabricación de condensadores impiden alcanzar tolerancias del orden del 1 % como en el caso de las resistencias, por lo cual resulta difícil asegurar que no existirá tal desapareamiento, sobre todo en regiones de la respuesta donde el desfasaje introducido es mayor (cerca de la frecuencia de corte). Una forma de evitar los efectos de esta distorsión es trabajar en el rango de frecuencias en el cual la misma no resulta significativa, esto es lejos de la frecuencia de corte.

Si se busca entonces que la banda de interés (0-500Hz) se encuentre en la zona plana de la respuesta en amplitud y fase del filtro RC, se puede fijar como criterio que la frecuencia de corte del mismo debe estar ubicada al menos una década por encima de la máxima frecuencia de interés. Siguiendo este criterio se fija la frecuencia de corte del filtro en 5KHz.

fc=5KHz

Ahora bien, al fijar la frecuencia de corte del filtro es necesario volver a poner atención en la frecuencia de muestreo elegida. Evidentemente la frecuencia de 1KHz que en un principio pareció suficiente en términos ideales según el Teorema de Nyquist ya no resulta adecuada, ya que como mínimo la misma deberá pasar a ser 10KHz (el doble de la frecuencia de corte del filtro). La realidad es que 10 KHz tampoco resulta suficiente dado que el filtro utilizado está lejos de ser un filtro ideal con selectividad infinita, sino que presenta en cambio una pendiente de atenuación de apenas 20dB/dec en zonas por encima de la frecuencia de corte. El filtro RC ofrece una atenuación razonable de -20dB recién una década por encima de la frecuencia de corte, o sea por encima de 10KHz. Teniendo en cuenta esto, la frecuencia de muestreo debería estar por encima de 20KHz.

En este punto surge una limitación importante del diseño; el MCU del Z1 no tiene capacidad suficiente para muestrear a velocidades tan altas. Incluso aunque los ADC lo permitieran, la cantidad de muestras a almacenar y a procesar serían demasiadas para las capacidades de RAM y procesamiento del MCU. En base a experimentación y ensayos con el Z1 se estableció que la frecuencia de muestreo podría ubicarse como máximo en el entorno de 8KHz.

Al estudiar los diseños de referencia citados anteriormente para ver de qué forma se resolvía este problema, se observó que si bien se utilizan MCUs de capacidades similares, la estrategia de sobre-muestreo se realiza a nivel de los ADC. Típicamente se utilizan ADC de tipo sigma-delta, que por sus características intrínsecas

de principio de funcionamiento, permiten realizar el muestreo utilizando una frecuencia de muestreo alta, pero almacenando sin embargo un número de muestras reducido. Debido a esta característica se dice que este tipo de ADC presenta un "filtro anti-aliasing intrínseco".

En el caso de los Z1 se dispone de ADC de 12-bits del tipo SAR (aproximaciones sucesivas). Este tipo de ADC no cuenta con las ventajas antes descritas, no permitiendo realizar un sobremuestreo que brinde la posibilidad de utilizar filtros de poca precisión y selectividad.

Teniendo en cuenta el foco del proyecto, considerando que no se busca una precisión exigente en las medidas y teniendo en cuenta que no se consideró dentro del alcance el diseño e implementación de hardware, se resolvió utilizar el diseño expuesto, pero usando una frecuencia de muestreo de 8KHz, y asumiendo los errores que puedan sufrirse por causa del solapamiento, que evidentemente no serán despreciables. Específicamente se utiliza un filtro RC con:

$R = 1k\Omega$; $C = 33nF$

La frecuencia de corte del filtro es entonces:

$$fc = \frac{1}{2.\pi.RC} = 4{,}82kHz \tag{B.13}$$

Tal como se dijo anteriormente, al ser la frecuencia de muestreo igual a 8kHz, puede preverse que se sufrirá una distorsión de aliasing significativa.

En cuanto a la distorsión introducida por el filtro en banda pasante, teniendo en cuenta que la zona de trabajo de 0-500Hz se encuentra aproximadamente una década por debajo de la frecuencia de corte del filtro, se prevé que la distorsión introducida por el filtro en dicha banda no sea significativa.

B.1.4. Protección de ADC

En la Sección B.1 se explicó que la tensión aplicada a las entradas ADC del MCU deben estar limitadas dentro de un rango de seguridad para evitar daños en la electrónica. Los límites de tensión eran [Vss-0,3V .. Vcc+0,3V]. Dado que la alimentación del mote se realiza en 5VDC a través del puerto USB, la alimentación del MCU será con Vss=0VDC y Vcc=3VDC. Esto significa que cualquier señal aplicada a un ADC no debe estar por fuera del rango [-0,3VDC .. 3,3VDC]. Dado que tal como se explicó en secciones anteriores se opta por utilizar puertos ADC equipados con un divisor resistivo a la entrada, que permite ampliar el rango de señal a la entrada del mote, este rango de seguridad [-0,3VDC .. 3,3VDC] a la entrada del MCU equivale a un rango de seguridad efectivo a la entrada del mote de [-0,5VDC .. 5,5VDC]. Este nuevo rango surge de multiplicar el rango del MCU por 5/3, que corresponde al valor introducido por el divisor resistivo.

Con el objetivo de asegurar el correcto funcionamiento de los ADC y evitar posibles daños, se decide tomar un rango de seguridad aún más conservador, fijandolo en:

Rango de seguridad de señales a la entrada del mote: [0 VDC .. 5VDC].

Observando el diseño de la Placa de Acondicionamiento de Señales, resulta sencillo ver que el límite inferior no presenta riesgos, ya que no hay posibilidades

de que dicha placa genere niveles menores a cero en la tensión de salida de señales. De hecho, la mínima tensión a la salida viene dada por la excursión de salida de los amplificadores operacionales, que será siempre mayor a cero y se ubica en el entorno de 0,8V para los valores de alimentación del diseño.

El límite superior de las señales a la salida de la Placa de Acondicionamiento de Señales, sí presenta en cambio riesgos a considerar. Tal como se explicó en secciones anteriores la misma fue diseñada con un rango de trabajo de señales de [0,8VDC .. 4VDC]. Sin embargo, ante señales de entrada a dicha placa que excedan el rango previsto, o fallas en la electrónica de la misma, por ejemplo en las etapas de ganancia, no es posible asegurar que las señales a la salida de esta placa no excederán los márgenes de seguridad. De hecho, el límite superior a la salida vendría dado por la excursión superior de los operacionales, que está muy por encima de los 5VDC fijados como límite. Por este motivo, es necesario incluir en el diseño algun sistema de protección que resguarde la electrónica de los ADC limitando los niveles de señal en sus entradas.

Uno de los métodos más típicos para realizar este tipo de protección consiste en utilizar diodos en configuración *voltage clamp*. Para ello se utilizan diodos ubicados entre la señal a limitar y una tensión de referencia, que será justamente el umbral al cual se desea limitar la señal. En la Figura B.13[4] se muestra un esquema sencillo de esta configuración.

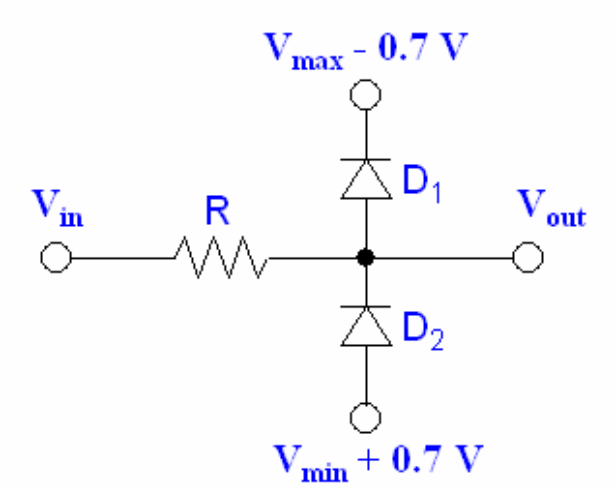


Figura B.13: Protección de los ADC con diodos en configuración voltage clamp.

Tal como se observa en la figura, contar con una señal de referencia de tensión de umbral es un requerimiento para implementar este mecanismo de protección. Para los valores propuestos se debería contar en este caso con una señal ubicada apenas por encima de 4VDC, que es el límite superior del rango de funcionamiento. En el caso de la Placa de Acondicionamiento de Señales dicha señal de referencia

[4]http://hades.mech.northwestern.edu/index.php/Diodes_and_TransistorsVoltage$_C lamp$, Mayo 2015.

no se encontraba disponible a la hora del diseño del circuito de muestreo, por lo cual se buscó una alternativa diferente para el diseño de la protección.

En lugar de utilizar la configuración previamente descrita, se utilizan diodos zener en la configuración de estabilizador de tensión de modo de evitar que el nivel de la señal exceda los valores de umbral. Para limitar el nivel de las señales al umbral propuesto de 5VDC se propone utilizar diodos de 4,7V. En la Figura B.14 se muestra esta configuración.

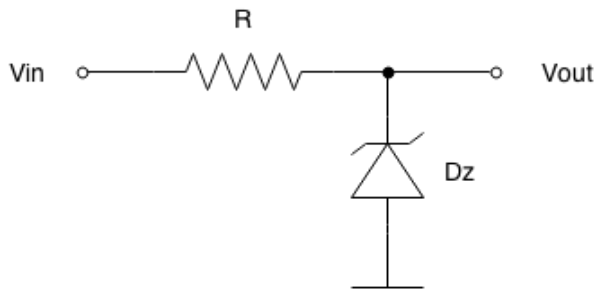


Figura B.14: Protección con diodos zener como limitadores de voltaje.

En caso de que la tensión en Vout supere el nivel de ruptura del diodo, el mismo comenzará a conducir y habrá un incremento en la caída de tensión en R, manteniendo el nivel en Vout dentro de los límites permitidos. Sin embargo, hay que tener cuidado en la elección del valor de R. Para un diodo de 1W de potencia, la corriente máxima admitida por el diodo será de:

$$Iz_{max} = \frac{Pz_{max}}{Vz} = \frac{1\text{W}}{4{,}7\text{V}} = 0,213\text{A} \tag{B.14}$$

La resistencia R sirve para proteger al diodo en caso de que la tensión supere el umbral de ruptura y el mismo comience a conducir. Según lo explicado en secciones anteriores, Vin podría llegar en este caso a unos 9V, que corresponde a la excursión de los amplificadores operacionales. Tomando un caso aún más conservador y pensando que Vin pudiera llegar a 10V, se tiene que en ese caso la corriente a través de la resistencia y el diodo será:

$$I_R = Iz = \frac{Vin - Vz}{R} = \frac{10\text{V} - 4,7\text{V}}{R} < 0,213\text{A} \tag{B.15}$$

Entonces:

$$R > \frac{10\text{V} - 4,7\text{V}}{0,213\text{A}} = 24\Omega \tag{B.16}$$

Este valor es el mínimo que se puede elegir para la resistencia. Sin embargo no es necesariamente el óptimo. Para valores bajos de la resistencia el nivel de corte del diodo se ubicara por encima de su nivel de ruptura nominal y la curva de corte será muy lenta. En cambio para valores altos de R el se asegura un corte efectivo

para niveles en el entorno del Vz, sufriendo como contrapartida un gran nivel de distorsión (atenuación no lineal) en los niveles más altos de la zona de trabajo. Este comportamiento fue ensayado y caracterizado para diferentes tipos de diodos y se concluyó que el mejor compromiso se obtiene utilizando:

- Diodo zener: Vz=4,7V P=1W
- Resistencia: 300Ω

B.1.5. Lista de componentes de placas diseñadas

Componente	**Descripción**	**Parámetro**
C2	Capacitor Cerámico	100nF
C3	Capacitor Electrolítico	100uF
D1	Diodo	1N4728
IC1	Amplificador Operacional	LM3901N
IC2	Regulador de tensión 5V	7805T
R1	Resistencia variable	0..100k (63k)
R2	Resistencia	84k5
R3	Resistencia	1M
R4	Resistencia	10k
R7	Resistencia	10k
R8	Resistencia	84k5
R9	Resistencia	2k4
R10	Resistencia	2k4
R11	Resistencia	1M
R12	Resistencia	68k
R15	Resistencia	47k
SV1	Conector	N/A
SV3	Conector	N/A
T2	Transistor BJT	BC337

Tabla B.2: Lista de componentes de placa de medición y acondicionamiento.

Componente	**Descripción**	**Parámetro**
C1	Capacitor cerámico	33nf
C4	Capacitor cerámico	33nf
C5	Capacitor cerámico	33nf
D1	Diodo zener	4,7V
D2	Diodo zener	4,7V
D3	Diodo zener	4,7V
JP1	Jumper	N/A
JP4	Jumper	N/A
JP5	Jumper	N/A
R1	Resistencia	750
R2	Resistencia	750
R3	Resistencia	750
R4	Resistencia	300
R5	Resistencia	300
R6	Resistencia	300
SV1	Conector 6 pines	N/A
SV2	Conector 6 pines	N/A

Tabla B.3: Lista de componentes de placa de filtrado y protección.

Apéndice C

Detalles de implementación de API y programación concurrente

C.1. Manejo de concurrencia: Threads y Corrutinas

Al hablar de *concurrencia* se hace referencia a la capacidad del software de ejecutar varias subrutinas simultáneamente en un mismo procesador. Existen varios enfoques para lograr este objetivo. Se ofrece aquí una breve introducción a algunas de las funcionalidades de Python para programacion concurrente utilizadas en el desarrollo del HEC. No se pretende cubrir completamente estos temas, que superan ampliamente el alcance de este proyecto. Sin embargo se intenta orientar al lector sobre los conceptos básicos para lograr utilizar la API desarrollada.

Dos de las principales funcionalidades de Python para programacion concurrente son los *threads* y la programación asíncrona mediante *corrutinas*.

C.1.1. Threads

Un thread, o hilo de ejecución, es la unidad básica de utilización de CPU. La principal implementación en Python es en el módulo *threading*, que permite escribir código con múltiples hilos de ejecución. De esta forma se mejora la performance ya que si un thread tiene que esperar por algún motivo, los demás continuarán ejecutándose.

El módulo *threading* proporciona una interfaz para trabajar con el módulo de bajo nivel *_thread*, y tiene funciones para sincronizar la ejecución entre los distintos hilos. Hay que destacar que el *scheduling*, es decir la coordinación entre los distintos threads decidiendo cuál se ejecuta en cada instante, es responsabilidad del sistema operativo. Si bien existen mecanismos para bloquear un thread y hacer que un conjunto de instrucciones se ejecuten sin interrupción, como por ejemplo con las funcionalidades de la clase *threading.Lock*, el programador no tiene un modo fácil de saber en qué momento se cambia de un thread a otro.

La documentación online de Python explica muy bien todas las funcionalidades de este módulo, y es una referencia básica para comenzar a utilizarlo.

C.1.2. Corrutinas: módulo asyncio

Otra forma de implementar la concurrencia, es mediante programación asíncrona con corrutinas. Una corrutina es una generalización de una subrutina o función, en la cual se permiten varios puntos de salida donde la ejecución se detiene en espera de un resultado (e.g. en espera de una respuesta a una petición a través de la red). En este caso el modo de trabajo entre las distintas subrutinas que intentan ejecutarse simultáneamente es cooperativo, donde cada corrutina cede el control de la ejecución a las demás cuando debe esperar por determinado resultado.

Para comprender el principio básico de funcionamiento hay que tener en mente que ciertas operaciones pueden bloquear la ejecución de un programa. Por ejemplo, cuando se intenta acceder a un recurso CoAP de algún servidor, el programa quedaría eventualmente detenido hasta que se obtuviera una respuesta. La programación asíncrona trata de aumentar el rendimiento implementando un agente capaz de ordenar la ejecución de las corrutinas. Cuando una corrutina debe esperar por un resultado cede el control nuevamente a este agente, que permite entonces que la siguiente corrutina se ejecute. Cuando el resultado por el que la primer corrutina quedó esperando está listo, la ejecución se reanuda recordando en qué punto había quedado suspendida.

El módulo *asyncio*, introducido en la última versión de Python (3.4), es un intento de unificar toda la programación asíncrona en Python y trabaja con tres elementos:

- *Corrutinas*: Son generadores de Python que pueden suspender su ejecución en espera de cierto procesamiento externo, y retornar al mismo punto donde se detuvo una vez que este procesamiento externo devuelve un resultado.

- *Event Loops*: Son una abstracción a las funciones de manejo de *polling* de eventos del sistema operativo. El sistema cuenta con recursos para monitorear cuando se cumplen determinadas condiciones, llamando a una función que atiende al evento detectado. Uno de los módulos que implementan este tipo de abstracción es *asyncio*. Sin embargo no es el único, ya que existen también otros módulos que lo hacen, como *Tornado* o *Twisted*. *Asyncio* permite contar con un scheduler que se encargue de coordinar la ejecución de las corrutinas.

- *Futures y Tasks*: La clase *asyncio.Future* y su subclase *asyncio.Task* representan el resultado de una operación que todavía no ha concluido. Cuando una corrutina suspende su ejecución, lo hace en espera de un objeto de la clase *asyncio.Future*.

Al definir y utilizar corrutinas en Python con *asyncio* lo primero que se debe tener en cuenta es que estas son funciones generadoras decoradas con *@asyncio.coroutine*. En caso de no estar familiarizado con estos conceptos se pueden encontrar buenas explicaciones en las referencias Gonzalez [24] y Lutz [34].

Un generador es una función que en lugar de devolver un único resultado devuelve un conjunto de resultados intermedios, permitiendo acceder a estos sin

necesidad de esperar que se terminen de computar todos. Utiliza la expresión *yield*, y *yield from* a partir de Python 3.3. Un decorador es una función especial que recibe como argumento una función, y devuelve otra función (tener en cuenta que las funciones en Python son objetos). De esta forma los decoradores añaden funcionalidades a una determinada función.

La observación práctica respecto a las corrutinas, es que si son llamadas directamente no se obtiene ningún resultado, sino un objeto *generador* sobre el que hay que iterar para obtener los resultados. Quien se encarga de realizar esta iteración, además de coordinar la ejecución de las corrutinas, es el *Event Loop*. Cada thread tiene un event loop asociado y en el Capítulo 7 se muestra detalladamente un ejemplo de cómo agendar las corrutinas en el event loop correspondiente.

C.2. Conexión del Border Router

Tal como se explicó en capítulos anteriores el Border Router (gateway de acceso a la HAN por parte del HEC) se implementa utilizando un Mote Zolertia Z1 idéntico a los utilizados para los nodos, con el programa de *rpl-border-router* ofrecido como ejemplo de aplicación por la plataforma Contiki. Por detalles acerca de este dispositivo y su funcionamiento consultar el Capítulo 4.

Para conectar el mote que actúa como Border Router al HEC se crea una interfaz de red virtual en el sistema operativo (Raspbian) del HEC. El mote se conecta a un puerto USB del HEC y luego se utiliza el Makefile incluido en la aplicación ejemplo provista por Instant Contiki que se encarga de crear la interfaz de red que servirá como punto de entrada a la red HAN.

Se ejecuta el siguiente comando en el directorio del border router:

```
1 make connect-router
```

Que simplemente hace lo siguiente para crear una interfaz virtual y dar acceso al border router:

```
1 ...
2 connect-router:    $(CONTIKI)/tools/tunslip6
3    sudo $(CONTIKI)/tools/tunslip6 $(PREFIX)
4 ...
```

C.3. Implementación de comunicación con la HAN

En la presente sección se explica cómo fueron implementadas en Python las funcionalidades del módulo de comunicación con la HAN, denominado *hancommunication*, añadiendo mayor nivel de detalle respecto a lo presentado en el Capítulo 6.

C.3.1. Estructura general y funciones globales

En la Figura C.1 se muestra un diagrama de la implementación en Python del módulo *hancommunication*.

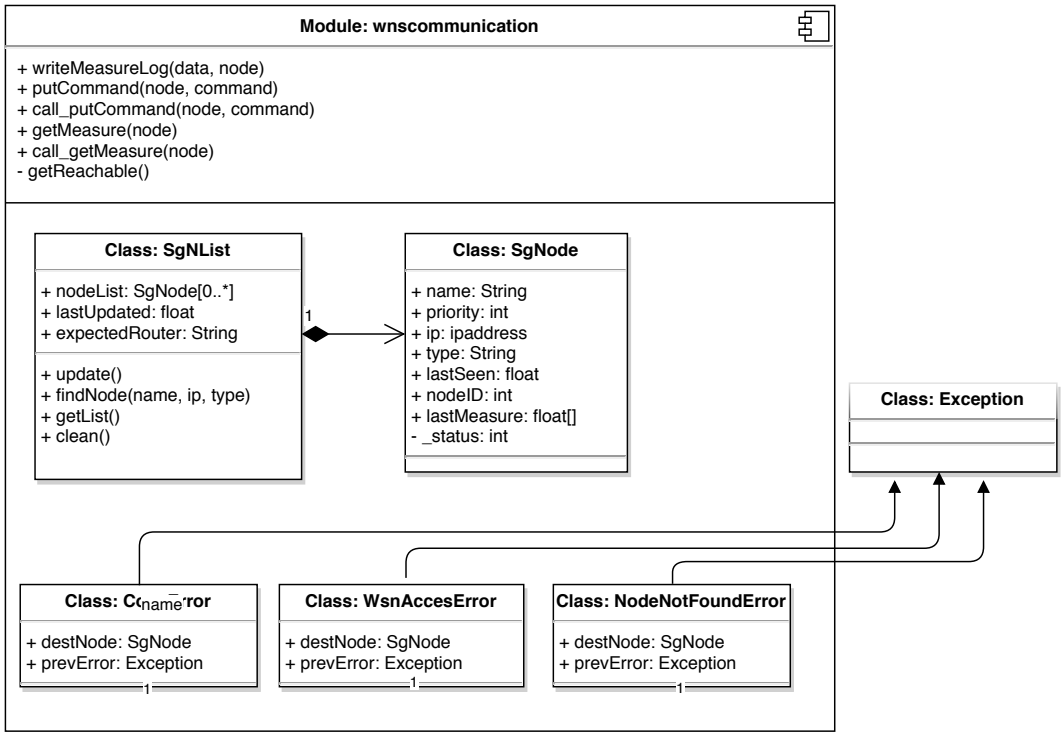


Figura C.1: Implementación del módulo de comunicación con la HAN.

Se detalla a continuación la implementación de las funciones globales de este módulo.

Búsqueda de nodos accesibles: función getReachable()

Esta función realiza una verificación de los nodos de la red, devolviendo una lista con objetos *ipaddress.IPv6Address*, correspondientes a las direcciones accesibles.

La estrategia de búsqueda de nodos utiliza el módulo Python *os* para acceder al comando *ping6* del sistema operativo. Este enfoque no es el óptimo, estando esta funcionalidad en el listado de las mejoras futuras del sistema. La implementación de otras estrategias (e.g. implementación de un servidor CoAP en el Border Router que devuelva el listado de nodos) es posible y mejorarían la performance del sistema en escenarios con varios nodos conectados, sobre todo si existe la necesidad de una frecuente actualización de la lista de nodos. Sin embargo, el diseño actual encapsula esta tarea dentro de una única función, por lo cual un cambio en esta implementación es posible sin la necesidad de realizar modificaciones en otras partes del código.

Esta forma de implementación de la búsqueda de nodos hace que en esta versión del sistema sea necesario asumir conocido el prefijo de la subred IPv6, y que las direcciones de los nodos utilizados sean las más bajas del rango. Sin mayor pérdida

de generalidad se eligió “aaaa::c30c:0:0:0/64” como prefijo, de acuerdo a los rangos de direcciones definidos en el capítulo 4.

Para gestionar los errores durante la ejecución del programa se utilizan las funcionalidades estándar de manejo de excepciones provistas por Python. Un primer posible error a considerar se produce en el contexto de la función *getReachable()*, en caso de haber un problema al intentar generar la lista de nodos. Para manejar esta situación se define una clase denominada *WsnAccessError*, que hereda de *Exception*, y que será explicada más adelante en este mismo capítulo.

Para realizar la búsqueda se recorren todas las direcciones válidas dentro de la subred, enviando 3 mensajes ping. Si el nodo responde a los mensajes se agrega el objeto *ipaddress.IPv6Address* a la lista de nodos a devolver.

Historial de medidas reportadas: función writeMeasureLog(data, nodeID)

Para facilitar el acceso a un historial con todas las medidas reportadas se mantiene un archivo llamado *'measureLog.csv'* y se define la función *writeMeasureLog*. Esta función escribe entradas en el log de medidas, agregando una marca de tiempo correspondiente al momento en que se registran las mismas. Para ello recibe como argumentos una lista con los datos a escribir y una identificación del nodo en un string llamado *nodeID*. Si bien en principio se espera recibir un string con la dirección IP, se podría utilizar otro tipo de identificador a estos efectos.

Recepción de mediciones de los nodos: corrutina getMeasure(node)

Esta corrutina implementa una de las principales funcionalidades del módulo y del sistema, que es la obtención de las mediciones reportadas por un nodo. Recibe como parámetro un objeto de la clase *SgNode* que encapsula la información relacionada a un nodo, y que se describe más adelante en este capítulo.

Para la implementación se utilizan directamente los servicios ofrecidos por el módulo *aiocoap*. Se crea una conexión utilizando la clase *Context* para luego configurar el tipo de mensaje y quedar a la espera de una respuesta.

Luego de recibida, la respuesta es procesada y se escribe una entrada en el log de medidas utilizando *writeMeasureLog()*.

Un detalle fundamental para evitar problemas en la comunicación luego de tener al sistema en funcionamiento por cierto período de tiempo es cerrar las conexiones abiertas. Esto no es considerado en los ejemplos de la documentación del módulo *aiocoap*, pero investigando dentro de la clase Context se observa que existe un método llamado *shutdown()*.

Para asegurar que la conexión es cerrada bajo cualquier circunstancia, se agrega la siguiente línea en un bloque *finally*:

```
1    finally:
2        yield from protocol.shutdown()
```

Todo error generado al momento de intentar acceder a una medida de un nodo es relanzado y representado en una clase llamada *CoapError*, que será descrita más adelante en este capítulo.

Asociada a la corrutina *getMeasure(node)* se definió la función auxiliar *call_getMeasure(node)* que permite acceder a la medida de un nodo sin necesidad de tener en cuenta como llamar adecuadamente a las corrutinas. Internamente se accede al event loop y se agenda la corrutina. Este es un primer ejemplo de cómo se debe invocar los servicios de una corrutina:

```
1    loop = asyncio.get_event_loop()
2    m = loop.run_until_complete(getMeasure(node))
```

Se observa que lo único que hace esta función auxiliar es obtener una referencia al event loop del thread en ejecución y comenzar la ejecución de la corrutina.

Comandos hacia los nodos: corrutina putCommand(node, comm)

La corrutina *putCommand(node, comm)* permite enviar comandos a los nodos. Recibe como parámetro un objeto del tipo *SgNode* representando al nodo que se quiere comandar y un string con el comando a enviar a través de CoAP.

Para implementarla también se utilizan los servicios ofrecidos por el módulo *aiocoap*, y el esquema es muy similar; Se abre una conexión utilizando la clase Context, se configura el mensaje a enviar y luego se queda a la espera de que el mismo sea recibido.

```
1        context = yield from Context.create_client_context()
2        payload = comm.encode()
3        request = Message(code=PUT, payload=payload)
4        request.opt.uri_host = str(node.ip)
5        request.opt.uri_path = ("Comandos","Relays")
6        response = yield from context.request(request).response
```

En caso de que el mensaje sea recibido satisfactoriamente, se debe actualizar el atributo *_status* del nodo. Esto se implementa en el bloque *else*:

```
1    else:
2        node._status = int("3" in comm or "2" in comm or "1" in comm)
```

Al igual que en la corrutina para realizar mediciones, todo error es relanzado utilizando la clase *CoapError*.

También se define una función auxiliar, llamada *call_putCommand(node, comm)*, para permitir enviar comandos sin necesidad de preocuparse por manipular adecuadamente el event loop.

C.3.2. Representación de los nodos: clase SgNode

La clase SgNode, que encapsula toda la información relacionada a un nodo incluyendo su estado, características y capacidades, se implementó en Python con los siguientes atributos de clase:

- *name*: Un string que representa e identifica al nodo.
- *priority*: Indicador del tipo de la posibilidad/conveniencia de actuar sobre este dispositivo. Ya sea por motivos de confort del usuario, como por posibilidades técnicas de los electrodomésticos, es necesario codificar qué tipo de acciones se puede realizar. Todos los nodos son creados con este atributo en el valor 1, que significa máxima prioridad.

- *type*: String que indica las capacidades del nodo, y permite verificar si tiene sentido solicitar medidas o enviar comandos hacia el mismo. Por defecto el constructor (función a la que se llama automáticamente al momento de crear un objeto, y se encarga de inicializar sus atributos) pone este atributo en *Dummy*, y si bien el usuario de la API podría fijar arbitrariamente criterios para los distintos tipos, se diseñó pensando en los siguientes valores: *Measure, MeasureCommand, Command* y *BorderRouter*.

- *ip*: Para el manejo de las direcciones IP se hace uso del módulo de Python *ipaddress*. La documentación web describe detalladamente cómo utilizar todas las funcionalidades que ofrece.

- *lastSeen*: El valor de un *float* utilizado como etiqueta temporal, que indica la última vez que fue visto el nodo. Se obtiene utilizando el módulo Python *time*.

- *lastMeasure*: La última medida realizada se guarda en una lista que el constructor de SgNode inicializa vacía.

- *nodeID*: Es un *int* que representa al nodo, y se obtiene a partir de la dirección IP.

- *_status*: Esta variable, inicializada en un entero con valor 0 por el constructor, representa el estado del nodo.

C.3.3. Representación de listas de nodos: clase SgNList

En esta sección se detalla cómo fueron implementadas en Python las funcionalidades ofrecidas por la clase *SgNList*, que representa listas de nodos conectados al sistema.

Los atributos de esta clase son:

- *_nl*: El listado con los nodos (SgNode) accesibles se almacena en este atributo, que el constructor inicializa como una lista vacía, para luego llamar al método *update()* descrito más adelante en esta misma sección de la documentación.

- *lastUpdated*: El indicador del tiempo desde la última actualización de la lista se almacena como un float creado por el módulo *time*, de forma similar al atributo *lastMeasure* de la clase SgNode. Notar que el constructor inicializa este valor en 0, y es el método *update* el que asigna el valor correspondiente.

- *expectedRouter*: Al crear la lista de nodos, se asigna en este String la dirección del nodo que actuará como border router. Esto será utilizado para configurarlo al momento de detectarlo en la lista de nodos.

Actualización de la lista: método update()

Una forma de independizar los detalles de cómo se realizan ciertas funcionalidades en el software, para que eventuales modificaciones no impacten sobre el código que utiliza esas bibliotecas de funciones, es ocultar la implementación (information hidding [1]) y definir claramente la forma de acceder a los datos y funciones. Para poder ocultar la implementación particular del acceso a los nodos, y la forma de detectarlos, se diseña de modo tal que el usuario cree una lista SgNList, y luego llame a su método update cada vez que requiera actualizarla. Por ejemplo:

```
1  list = hancommunication.SgNList()
2  list.update()
```

El objetivo de esto es poder mejorar la forma en que se actualiza la lista de nodos, en búsqueda de más eficiencia y confiabilidad, sin impactar en los algoritmos de optimización escritos con la API.

La función *update()* obtiene un listado de las direcciones válidas mediante la función global *getReachable()* descrita anteriormente, y luego recorre el listado agregando nuevos nodos. Al detectar una dirección que no estaba anteriormente en la lista, se llama al constructor de *SgNode* con la nueva dirección como parámetro, para crear un nuevo nodo. Si la dirección corresponde a un nodo que ya estaba presente en la lista, solamente se actualiza su atributo *lastSeen* con el tiempo actual.

La primera vez que se ejecuta la función, o siempre que la lista esté vacía, se intenta localizar al border router según el valor del atributo *expectedRouter*.

Búsqueda de un nodo en la HAN: método findNode()

Al momento de configurar la red y querer asignar parámetros y características a cada nodo, es necesario poder verificar que los mismos se encuentran presentes y accesibles.

Con este objetivo la función *findNode(name, dirip)* verifica si en la lista de nodos detectados hay uno con la IP indicada como parámetro. En caso afirmativo modifica sus atributos seteando el nombre indicado en el primer parámetro y devuelve una referencia al nodo correspondiente. Como tercer parámetro acepta de forma opcional el tipo de nodo que se está buscando, para modificar el atributo *type* al momento de localizarlo.

Si ningún nodo de la lista tiene la dirección IP buscada, se realiza una actualización de la misma y se vuelve a buscar. Esto se repite por defecto 2 veces, y como cuarto parámetro se acepta la cantidad de veces que debe actualizar la lista de nodos antes de concluir que el dispositivo buscado no está accesible.

Para manejar la situación en la que un nodo no es localizado se lanza una excepción del tipo *NodeNotFoundError*, clase que se describe más adelante en este capítulo. Así el usuario que llama a esta función tiene una forma de atender esta situación usando el manejo estándar de excepciones de Python.

[1] http://en.wikipedia.org/wiki/Information_hiding

Ocultando la implementación de las listas: método getList()

Un atributo que la totalidad de los usuarios de la clase *SgNList* utilizarán es el *_nl*, debido a que es la razón de ser de esta clase. Tal como se describió anteriormente, este atributo es una lista de objetos de clase SgNode.

Si en el futuro se quisiera hacer una modificación a la implementación de la lista de nodos, todo el código que utiliza la API se vería afectado, debido a que todos utilizarán una lista de nodos. Para lograr una mayor robustez se agrega la función *getList()* para acceder a la lista de cada objeto SgNList. Si bien en Python no hay atributos privados, y de hecho los usuarios pueden acceder directamente a la lista, se sigue la convención de agregar un guión bajo en su nombre para indicar que el atributo debería ser utilizado con precaución.

Limpieza de la lista de nodos: método clean(maxAge)

Además de verificar la incorporación de nuevos nodos a la red, y actualizar la información que confirma la presencia de los detectados previamente, puede resultar de gran utilidad tener un mecanismo para la limpieza de la lista. La función *clean()* se encarga de recorrer la lista, y eliminar de la misma a los nodos que hace tiempo no son detectados.

El parámetro a observar es el atributo *lastSeen* de los nodos, y el criterio está basado en un parámetro que se le puede pasar a la función, correspondiente a la cantidad de segundos de antigüedad máxima admitida. Por defecto se fijó este parámetro en el equivalente a 1 día.

Auxiliar para Debugging: método __DEBUGPrintNodeList()

Esta función auxiliar resultó muy útil durante el desarrollo del HEC, y tiene aplicación general a la hora de implementar algoritmos con él. Por ello se mantuvo a __DEBUGPrintNodeList(), función cuyo único objetivo es imprimir en pantalla la lista actual de los nodos.

C.3.4. Errores de comunicación a nivel de CoAP: clase CoapError

Con esta clase, que hereda de *Exception*, se representan los errores ocurridos al momento de intentar acceder a un nodo a través de CoAP. Tiene dos atributos que son:

- *destNode*: Aquí se almacena una referencia al nodo con el que se estaba intentando interactuar.
- *prevError*: Guarda una referencia a la excepción que generó el error. La clase *CoapError* está pensada para relanzar excepciones dando información útil sobre el contexto de la misma, y por lo tanto resulta de utilidad tener una referencia a la excepción original.

C.3.5. Errores en el acceso a la HAN: clase HanAccessError

Esta clase, que también hereda de *Exception*, representa un error al intentar acceder a algún nodo de la HAN para verificar conectividad. Está pensada en base a la implementación actual de la búsqueda de nodos activos en la red y en este caso se tiene como atributo, además del error previo, la dirección IP a la que se estaba intentando acceder al momento del error.

C.3.6. Error de nodo no localizado: clase NodeNotFoundError

Apoyando la funcionalidad proporcionada por el método *findNode()*, se definió la clase *NodeNotFoundError*. Con este tipo de excepción se representa la situación en que determinado nodo que estaba siendo buscado no se encuentra presente en la red. Quien llama a la función *findNode()* debe capturar y atender este tipo de excepciones.

C.4. Implementación de la interfaz web

Para poder interactuar con los usuarios del sistema, el HEC debe ofrecer una interfaz. Se propone la implementación de una interfaz web basada fundamentalmente en herramientas open source gratuitas, que permiten disminuir los tiempos de desarrollo y costos asociados.

Si bien se utilizan algunos servicios secundarios que no son completamente open-source, el núcleo central de la interfaz sí lo es, estando todo el código accesible y disponible para instalarse en un servidor propio. En esta primer implementación particular, y por motivos de agilizar el desarrollo, la interfaz web se encuentra alojada en servidores de terceros que ofrecen el servicio de hosting. Esto implica que si el usuario pierde conexión a internet no podrá interactuar con la interfaz, pero por otra parte ofrece la posibilidad de comando remoto de los dispositivos desde cualquier lugar del mundo, siendo esto una funcionalidad particularmente atractiva para los usuarios.

A continuación se describen brevemente las plataformas utilizadas para el desarrollo de la interfaz web y luego se describen los módulos de software desarrollados en el HEC para implementar la comunicación con dicha interfaz.

C.4.1. Estructura de la interfaz web

La interfaz web está desarrollada en base a un componente central open-source que mantiene la base de datos, y además se utilizaron otras dos plataformas web que ofrecen servicios gratuitos. Estos tres componentes son:

- EmonCMS[2]: Constituye el núcleo de la plataforma ya que es aquí donde se almacenan las diferentes variables de interés, se mantiene un historial

[2]https://github.com/emoncms/emoncms/blob/master/readme.md, Mayo 2015.

de mediciones, y se muestran al usuario mediante paneles gráficos llamados *dashboards*.

- Weebly[3]: Se utiliza para desarrollar una interfaz web principal (página web de inicio), con los diferentes menús de acceso a las distintas funcionalidades de la plataforma.

- Freeboard[4]: Mediante esta herramienta se implementa una interfaz gráfica complementaria a la de EmonCMS, fundamentalmente para permitir el uso de botones para el comando remoto de los electrodomésticos.

A continuación se describe brevemente cada una de estas tres herramientas y las funcionalidades desarrolladas en base a ellas.

C.4.2. Núcleo de la interfaz utilizando EmonCMS

EmonCMS es una aplicación web open source que permite procesar, almacenar y visualizar información acerca de consumo eléctrico. La plataforma es parte del proyecto OpenEnergyMonitor[5].

Esta plataforma permite definir diferentes variables sobre las cuales se pueden escribir o actualizar valores en forma periódica. Los valores son guardados por el servidor de la aplicación y quedan disponibles para ser consultados por el usuario.

En EmonCMS cada variable o fuente de información se denomina *Input*. Una vez declarada una *Input* se puede crear un log o registro de información a partir de ella. Este log se denomina *Feed*. De alguna manera puede considerarse entonces que las *Inputs* son las variables de tiempo real, o fuentes de información que son actualizadas desde algún dispositivo externo (en este proyecto sería el HEC) y los *Feeds* son variables históricas capaces de almacenar las tendencias de las *Inputs*. Un *Feed* puede llevar el registro del valor de una *Input* tal como el mismo es recibido, o puede también almacenar información pre-procesada a partir de los valores de una *Input*. Por ejemplo, se podría tener un *Feed* que almacene el valor de la potencia aparente en base a los valores recibidos en los *Input* "Tensión Vrms" y "Corriente RMS" procesando estos valores mediante la operación $Vrms * Irms$ y almacenando el resultado en el *Feed* "Potencia Aparente".

A partir de lo explicado anteriormente se debe notar que la interfaz entre esta plataforma web y los dispositivos donde la información es generada (HEC) está dada por la configuración de las *Inputs* y por la forma en que estas son actualizadas. Es decir, la forma en que la plataforma recibe información actualizando las *Inputs*.

Para esta aplicación particular se configuraron 5 conjuntos idénticos de *Inputs*, cada uno correspondiente a un nodo (nodos 1 a 5). Por cada nodo se crearon entonces las siguientes *Inputs*:

- Vrms *(valor de tensión)*

[3]http://www.weebly.com, Mayo 2015.
[4]https://freeboard.io/, Mayo 2015.
[5]http://openenergymonitor.org/emon/, Mayo 2015.

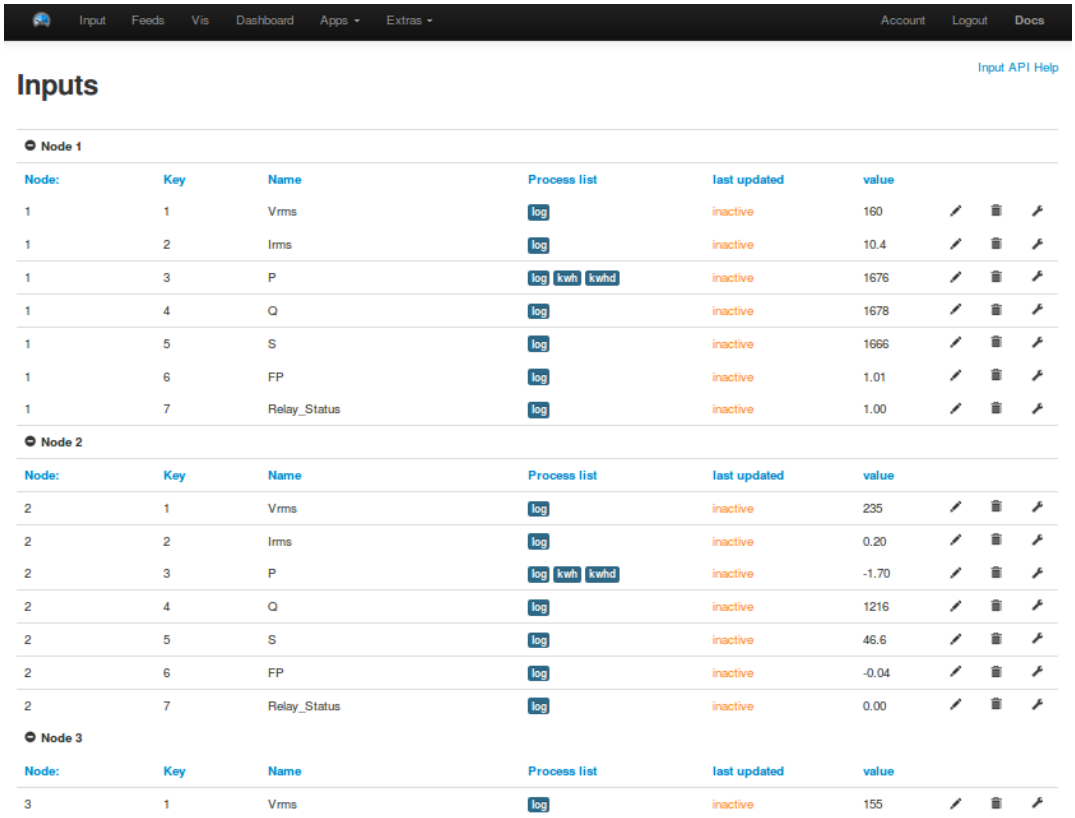
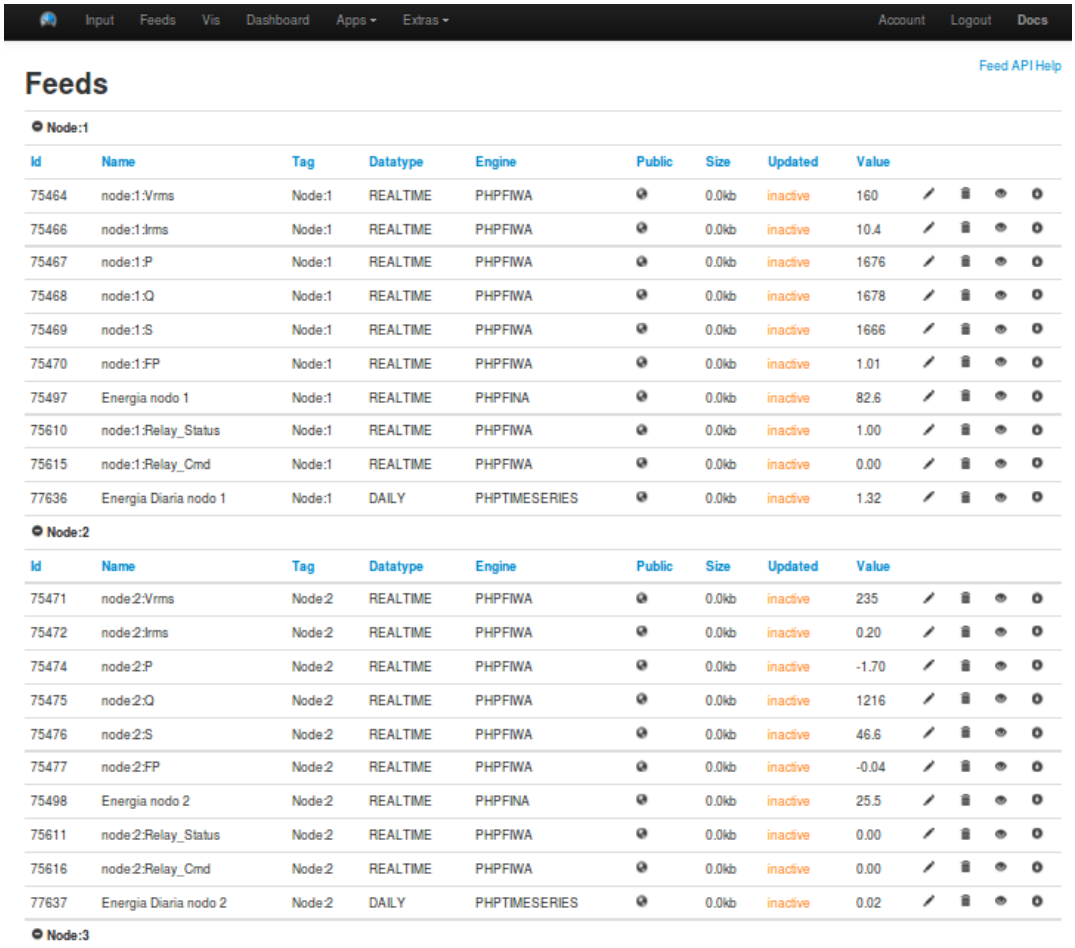


Figura C.2: Configuración de Inputs en EmonCMS.

- Irms *(valor de corriente)*
- P *(valor de potencia activa)*
- Q *(valor de potencia reactiva)*
- S *(valor de potencia aparente)*
- FP *(valor del factor de potencia)*
- Relay_Status *(representación de status de los relays)*

En la Figura C.2 se muestra una captura de pantalla de dicha configuracion. Para cada *Input* se configuraron diferentes *Feeds* que se observan bajo la columna "Process List". Al realizar dicha configuración, se generaron 5 conjuntos de *Feeds* (uno por cada nodo) con los siguientes logs:

- Vrms *(Log de la tensión)*
- Irms *(Log de la corriente)*
- P *(Log de la potencia activa)*



Input Feeds Vis Dashboard Apps Extras Account Logout Docs

Feed API Help

Feeds

Node:1

Id	Name	Tag	Datatype	Engine	Public	Size	Updated	Value
75464	node:1:Vrms	Node:1	REALTIME	PHPFIWA		0.0kb	inactive	160
75466	node:1:Irms	Node:1	REALTIME	PHPFIWA		0.0kb	inactive	10.4
75467	node:1:P	Node:1	REALTIME	PHPFIWA		0.0kb	inactive	1676
75468	node:1:Q	Node:1	REALTIME	PHPFIWA		0.0kb	inactive	1678
75469	node:1:S	Node:1	REALTIME	PHPFIWA		0.0kb	inactive	1666
75470	node:1:FP	Node:1	REALTIME	PHPFIWA		0.0kb	inactive	1.01
75497	Energia nodo 1	Node:1	REALTIME	PHPFINA		0.0kb	inactive	82.6
75610	node:1:Relay_Status	Node:1	REALTIME	PHPFIWA		0.0kb	inactive	1.00
75615	node:1:Relay_Cmd	Node:1	REALTIME	PHPFIWA		0.0kb	inactive	0.00
77636	Energia Diaria nodo 1	Node:1	DAILY	PHPTIMESERIES		0.0kb	inactive	1.32

Node:2

Id	Name	Tag	Datatype	Engine	Public	Size	Updated	Value
75471	node:2:Vrms	Node:2	REALTIME	PHPFIWA		0.0kb	inactive	235
75472	node:2:Irms	Node:2	REALTIME	PHPFIWA		0.0kb	inactive	0.20
75474	node:2:P	Node:2	REALTIME	PHPFIWA		0.0kb	inactive	-1.70
75475	node:2:Q	Node:2	REALTIME	PHPFIWA		0.0kb	inactive	1216
75476	node:2:S	Node:2	REALTIME	PHPFIWA		0.0kb	inactive	46.6
75477	node:2:FP	Node:2	REALTIME	PHPFIWA		0.0kb	inactive	-0.04
75498	Energia nodo 2	Node:2	REALTIME	PHPFINA		0.0kb	inactive	25.5
75611	node:2:Relay_Status	Node:2	REALTIME	PHPFIWA		0.0kb	inactive	0.00
75616	node:2:Relay_Cmd	Node:2	REALTIME	PHPFIWA		0.0kb	inactive	0.00
77637	Energia Diaria nodo 2	Node:2	DAILY	PHPTIMESERIES		0.0kb	inactive	0.02

Node:3

Figura C.3: Configuración de Inputs en EmonCMS.

- Q *(Log de la potencia reactiva)*
- S *(Log de la potencia aparente)*
- FP *(Log del factor de potencia)*
- Energia *(Log de energía, calculado a partir de la potencia en el tiempo)*
- Relay_Status *(Log de status de los relays)*
- Relay_CMD *(Variable utilizada para realizar el comando de los relays)*
- Energía_Diaria

En la Figura C.3 se muestra una captura de pantalla de esta configuración.

EmonCMS brinda dos interfaces de programación denominadas *Input API* y *Feed API* respectivamente, con comandos básicos a través de los cuales un servidor o cliente remoto puede enviar información hacia las *Inputs* y leer o modificar valores de los *Feeds*. Al crear una cuenta en EmonCMS el usuario (programador) obtiene

una URL y un conjunto de claves denominadas *API Keys* que conforman las rutas de acceso de información para realizar este tipo de intercambios.

Para actualizar una *Input* se debe acceder (método GET en http) a la URL correspondiente enviando un comando predefinido en un formato determinado por las API mencionadas e incluyendo la *API Key* de seguridad.

Existen diferentes formatos en los que los comandos y la información intercambiada puede ser representada. En el presente trabajo se decidió utilizar comandos en formato *JSON*[6]. *JSON* es un formato ligero para el intercambio de información que tiene la ventaja de ser fácilmente legible por humanos. Se trata de un formato de texto basado en JavaScript, pero que utiliza convenciones simples que resultan familiares en casi cualquier lenguaje de programación.

En base a este formato se utilizaron dos comandos básicos; uno para escribir valores en las *Inputs* (actualización de variables desde el HEC) y otro para leer valores de los *Feeds*. Esto último se hizo en particular para leer los valores de comando manual de los electrodomésticos introducidos por el usuario mediante la interfaz gráfica. A continuación se muestra la estructura de estos comandos http:

```
post_command =
"/input/bulk.json?data=[[0,
{node},
{Vrms},
{Irms},
{P},
{Q},
{S},
{FP},
{status}]]&apikey="+input_apikey

get_command =
"/feed/value.json?id={ID}&apikey="+feed_apikey
```

En el caso de *post_command* se utiliza un formato en forma de estructura donde se envía el valor del nodo y todos los valores asociados a sus variables. Luego se incluye un *apikey* necesario para la autenticación del mensaje.

En el caso de *get_command* se identifica el *Feed* a acceder mediante su ID. Luego se incluye el *apikey* de autenticación.

Estas instrucciones son utilizadas por el software del HEC para el intercambio de información con la plataforma que maneja la interfaz web. Los detalles acerca del rol del HEC se abordan en secciones posteriores.

Para facilitar la lectura de la información por parte del usuario se configuraron algunos paneles gráficos o *dashboards*. En la Figura C.4 se muestran algunas capturas de estos paneles.

C.4.3. Interfaz gráfica para comando remoto utilizando Freeboard.io

Freeboard es una plataforma online para el desarrollo y ejecución de paneles gráficos de visualización de información (*Dashboads*). En su versión más básica, que permite únicamente desarrollos abiertos, la plataforma es gratuita.

Esta plataforma permite mostrar información mediante gráficos animados y diagramas de tendencia. Soporta entre otros el formato *JSON* para el intercambio

[6]http://json.org/, Mayo 2015

Figura C.4: Capturas de pantalla de algunos Dashboards en EmonCMS.

de información con servidores o clientes externos a través de HTTP, y por lo tanto se puede integrar fácilmente con la base de datos de EmonCMS.

Una de las ventajas que ofrece Freeboard es que sus interfaces gráficas están optimizadas para Smart Phones. Por este motivo se decidió utilizar esta herramienta para complementar los *dashoboards* desarrollados en EmonCMS brindando una versión más sencilla de los mismos y fundamentalmente incluyendo una interfaz con comandos para actuar manualmente sobre los nodos y por ende sobre los electrodomésticos conectados.

Para lograr la funcionalidad de comando se utilizaron los *Feeds* "Relay_CMD" mencionados en párrafos anteriores. Mediante botones que brindan acceso a URLs compuestas por el comando de escritura de los *Feeds*, se modifica el valor de estas variables. Este valor puede ser consultado por el HEC para determinar si el usuario solicitó actuar sobre algún dispositivo, en función de la estrategia de monitoreo y comando implementada.

En la Figura C.5 se muestra una captura de pantalla del panel de monitoreo y comando para uno de los nodos. El panel gráfico incluye un indicador de potencia actual, una gráfica de tendencia de la potencia y del estado de la carga (ON/OFF), y lo más importante, un link de comando para activar y desactivar el nodo, encendiendo o apagando el electrodoméstico.

Los botones de *Encendido* y *Apagado* no son otra cosa que un link a una URL con el comando de escritura de los *Feeds* "Relay_CMD", incluyendo el valor a escribir (0 o 1) en formato *JSON*. A continuación se muestra la URL de comando, donde los campos ID, VALOR y APIKEY corresponden al identificador de la variable, valor a escribir, y clave de autenticación respectivamente:

```
1 http://emoncms.org/feed/insert.json?id=[ID]&value=[VALOR]&apikey=[APIKEY]\
```

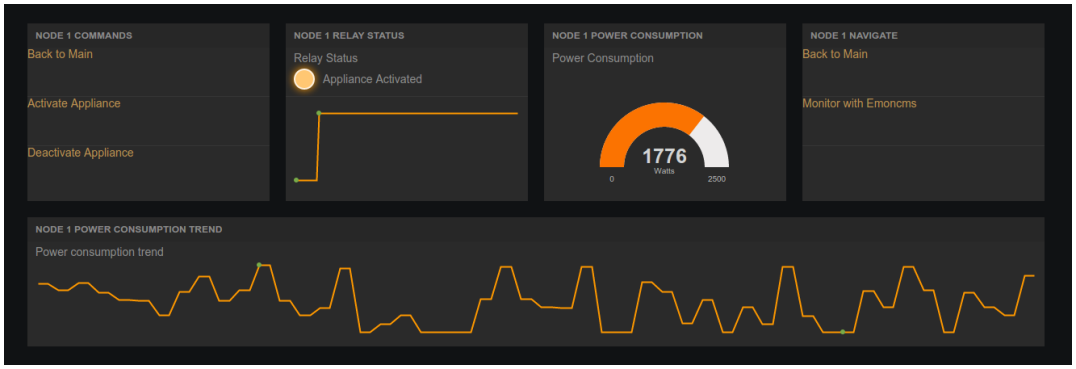


Figura C.5: Capturas de pantalla de un Dashboard en Freeboard.

C.4.4. Página web principal basada en Weebly

Weebly es una plataforma de edición y desarrollo de páginas web que incluye servicios de hosting y administración de dominios. Si bien los servicios en general son pagos, existen algunas opciones básicas gratuitas.

La versión gratuita de desarrollo tiene limitaciones a nivel gráfico y funcional, como por ejemplo la no disponibilidad de utilizar acceso a la web con usuario y contraseña. Esta plataforma brinda además subdominios gratuitos bajo el dominio *weebly.com*.

A los efectos de este proyecto, se utilizaron las versiones gratuitas de esta plataforma con el objetivo de desarrollar una pantalla principal o *Home Page* y así lograr una plataforma más amigable para el usuario. Mediante esta página se puede obtener acceso a los datos del proyecto y los diferentes menús, incluyendo medición y comando, con los links correspondientes a las plataformas y paneles gráficos antes mencionados.

En la Figura C.6 se muestran algunas capturas de pantalla de esta interfaz de inicio.

C.4.5. Interacción entre el HEC y la interfaz web: módulo webinterface

Para facilitar la comunicación entre el HEC y la interfaz web previamente descrita se diseña el módulo *webinterface*. La Figura C.7 se muestra una representación de la implementación del módulo *webinterface*, incluyendo las funciones globales ofrecidas y clases definidas.

A continuación se describe la implementación de las distintas funcionalidades de este módulo, así como las características más relevantes de la base de datos e interfaz web utilizada.

Historial con los datos publicados: función writeInterfaceLog(data, node)

La función *writeInterfaceLog(data, node)* escribe una entrada en el log de la interfaz, agregando una marca de tiempo correspondiente al momento en que se

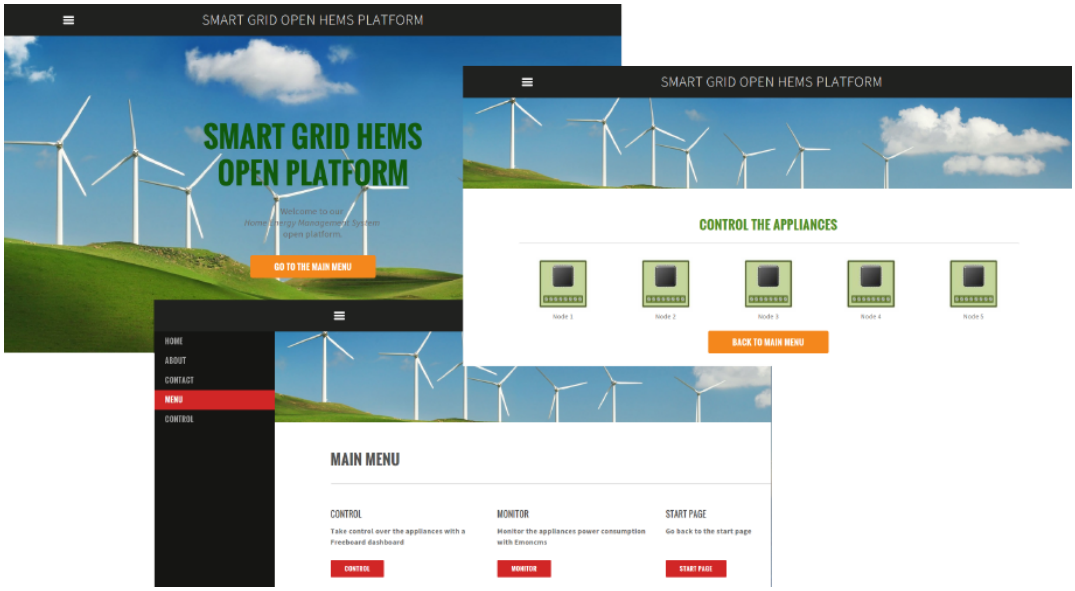


Figura C.6: Capturas de pantalla del menú principal de la interfaz de usuario.

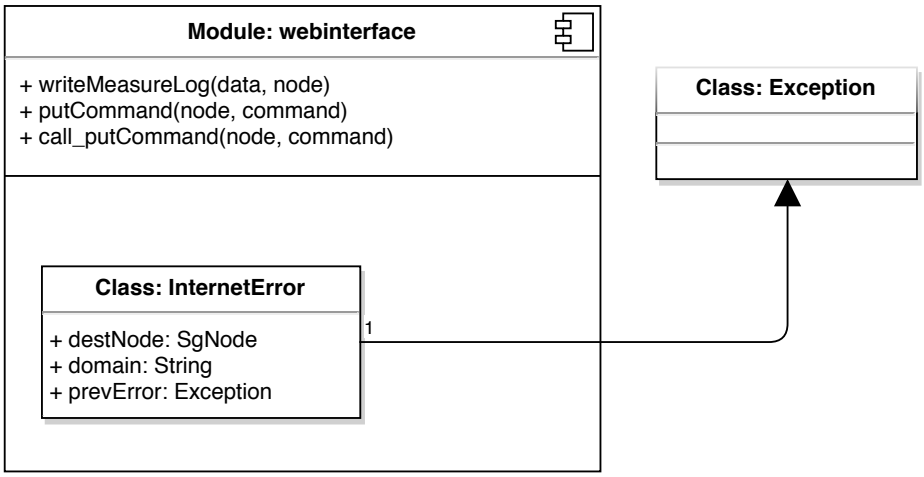


Figura C.7: Diagrama general del módulo de manejo de la interfaz web.

hace el registro. El log se mantiene en el archivo 'interfaceLog.csv', y sirve para mantener un historial de toda la información que fue enviada al servidor web.

Recibe como parámetros una lista con los datos que fueron publicados, y una referencia al objeto SgNode que representa al nodo involucrado.

Publicando datos en la interfaz web: función pushData(data, node)

Permite publicar en la interfaz web las medidas reportadas por un nodo. Para ello simplemente abre una conexión HTTP con el servidor web, para luego hacer un GET con un *string* que contiene los datos en formato JSON:

```
input_apikey = "68643f450badfb4fb82d61197af28b67"
       post_command =
"/input/bulk.json?data
=[[0,{nodeID},{Vrms},{Irms},{P},{Q},{S},{FP},{status}]]
&apikey="+input_apikey
    conn = http.client.HTTPConnection("emoncms.org")
           conn.request("GET",
post_command.format('''Formato adecuado para los datos'''))
       post_response = conn.getresponse()
```

Una vez publicados los datos en la interfaz de forma satisfactoria, se escribe la entrada en el log correspondiente, usando la función *writeInterfaceLog(data,node)*. En caso producirse algún error mientras se intenta publicar los datos, se relanza una excepción de la clase *InternetError*, que se describe más adelante en esta sección.

Comando remoto de los dispositivos: función checkCommand(node)

Para simplificar el diseño, y permitir al HEC un control total y centralizado de todos los dispositivos sin necesidad de atender directamente las solicitudes del usuario, de diseña de modo que este último interactúe con la interfaz web. Es luego el HEC quien define el momento de consulta a esta base de datos, según los criterios de diseño, para actuar o no de acuerdo a lo solicitado por el usuario.

En el capítulo 7, donde se diseña un sistema básico pero completo de monitoreo, comando y optimización, se presenta un ejemplo donde el controlador chequea constantemente si el usuario requiere realizar alguna acción sobre los dispositivos.

La función *checkCommand(node)* proporciona un mecanismo para interactuar con la interfaz, consultando si el usuario solicitó encender o apagar un dispositivo. Recibe como parámetro el objeto de clase SgNode que representa al nodo, y devuelve el comando que actualmente está almacenado en la base de datos.

Errores en el acceso a la interfaz: clase InternetError

Para representar errores al momento de intentar acceder a la interfaz, se definió la clase *InternetError*, que hereda de Exception. Este tipo de excepción es lanzada al momento de producirse un error al intentar escribir un dato en la interfaz, o al chequear un comando de usuario. Define los siguientes atributos:

- *domain* Es un String que indica el domino al que se estaba intentando acceder.
- *destNode* Guarda una referencia al nodo relacionado con la solicitud de interacción con la interfaz.
- *prevError* Al igual que las demás clases de Excepción definidas, están pensadas para relanzarse cuando se produce algún error dentro de una función. Por ello es útil mantener una referencia al error original, información que se almacena en este atributo.

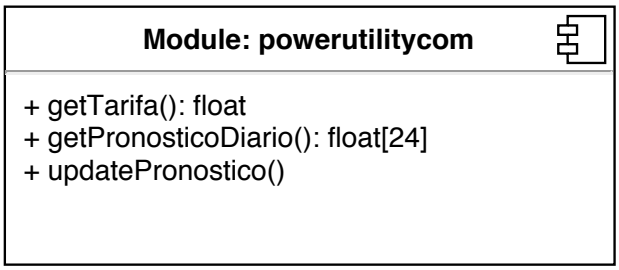


Figura C.8: Módulo de comunicación con EMS.

C.5. Implementación de comunicación con EMS

El módulo *powerutilitycom* simula la interacción con la empresa de suministro eléctrico. Como todavía no existe en el Uruguay un EMS que sea capaz de enviar comandos y proporcionar información RTP, es necesario implementar este módulo de la forma más general posible.

En la Figura C.8 se muestra un diagrama del módulo, donde se detallan las funciones que ofrece.

C.5.1. Obtención del pronostico diario de la tarifa: función getPronosticoDiario()

Esta función devuelve una lista con el precio estimado para la energía en cada hora del día. Para implementar esta funcionalidad, se mantiene un archivo local llamado "tarifaEstimada.csv", y la función *getPronosticoDiario()* lee este archivo y devuelve los valores correspondientes.

C.5.2. Actualización de la tarifa diária: función updatePronostico()

La función *updatePronostico()* actualiza el archivo que mantiene el pronóstico diario del precio de la energía, desde un servidor web. Permite de este modo cambiar el pronóstico de forma remota.

Es en esta función donde se espera que sea necesario cambiar la implementación al momento en que la empresa de suministro comience a enviar información sobre el pronóstico diario de la tarifa. Si bien no hay nada de esto definido, se espera que los cambios necesarios en la implementación no impacten en el usuario de la API.

C.5.3. Precio real actual: función getTarifa()

La función *getTarifa()* permite conocer el precio real de la energía en la hora actual. Actualmente esto se implementa simulando un valor aleatorio del precio, basado en el pronóstico para la hora correspondiente. Esto se hace con las bibliotecas estándar de Python:

```
1     try:
2         tariff = getPronosticoDiario()
3         p = tariff[time.gmtime(time.time()).tm_hour]
4 ...
5     else:
6         return random.gauss(p,0.7)
```

Apéndice D

Gestión del proyecto

D.1. Flujo Contable

En la Fig. D.1 se muestra una gráfica del saldo contable a lo largo del proyecto. La gráfica corresponde al presupuesto de equipamiento y materiales. Se comienza con un saldo negativo hasta el cobro del primer desembolso por parte de la ANII, y se logra finalizar el proyecto con un saldo positivo a nivel de este rubro.

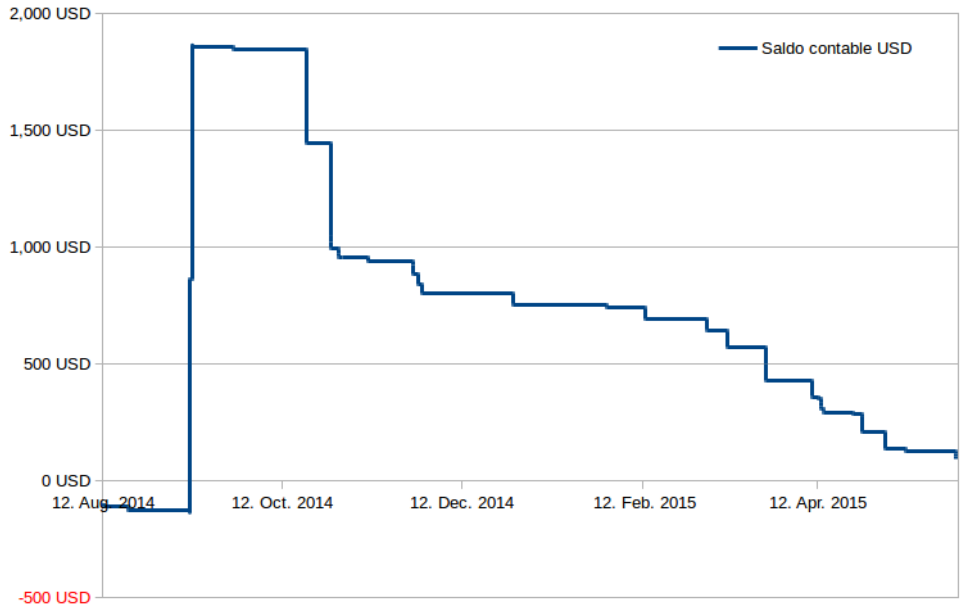


Figura D.1: Saldo contable a lo largo del proyecto.

D.2. Dedicación

D.2.1. Estimación inicial de horas-hombre

En la Fig. D.2 se muestra la estimación inicial de horas-hombre por tarea realizada al inicio del proyecto. Se observa que en el total se considera la asignación de 1598 horas-hombre para este proyecto.

SMART GRID			
Tareas	**HH**	**Subtareas**	**HH**
Estudio de estado del arte Smart Grid	**60**	Investigación	45
		Documentación	15
Nodo de carga	**145**	Diseño	50
		Desarrollo e implementación	40
		Software	40
		Documentación	15
Nodo de medición	**145**	Diseño	50
		Desarrollo e implementación	40
		Software	40
		Documentación	15
Controlador	**465**	Estudio de requerimientos del controlador	10
		Elección de micro controlador	60
		Diseño	30
		Desarrollo e implementación	50
		Desarrollo de aplicación	300
		Documentación	15
Sistema de comunicación	**275**	Estudio del estado del arte	40
		Definición topología de sestema de comunicación	20
		Integración de HW	45
		Implementación HW interfaz de comunicación	20
		Software	75
		Ensayo sistema de comunicación	60
		Documentación	15
Armado de perfil de control	**115**	Estudio comportamiento de carga	30
		Gestión de compra o usufructo de carga para pruebas	5
		Definición de atributos de la carga	10
		Desarrollo de algoritmo de control	60
		Documentación	10
Ensayo global del sistema	**105**	Ensayo	90
		Documentación	15
Documentación final	**60**		60
Hito 1 AVANCE	**2**		2
Hito 2 AVANCE	**2**		2
Hito 3 DEFENSA	**4**		4
Seguimiento y gestión de proyecto	**70**		70
Buffer	**150**		150
Total HH	**1598**		

Figura D.2: Estimación inicial de horas-hombre por tarea.

D.2.2. Nivel de dedicación horaria a lo largo del proyecto

En la Fig. D.3 se muestra la evolución de la dedicación horaria de cada uno de los integrantes y del grupo en general a lo largo de todo el proyecto. Se aprecia cómo al comienzo la dedicación fue irregular, y en la etapa final la misma se

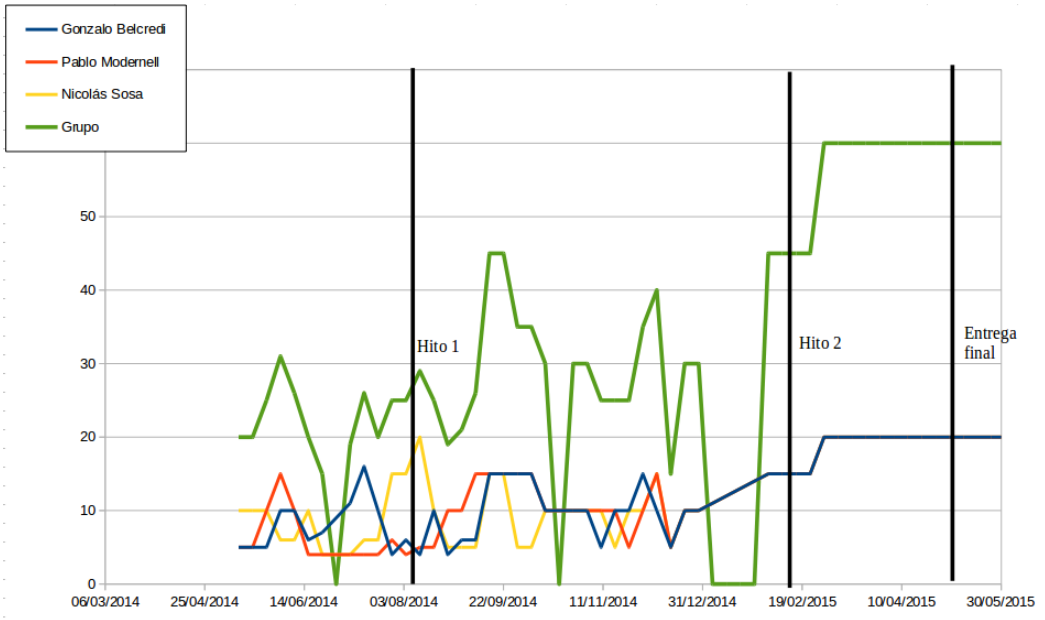
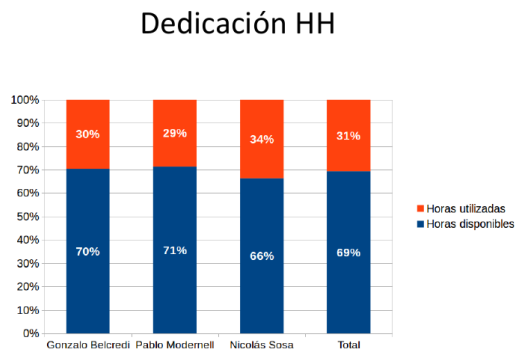
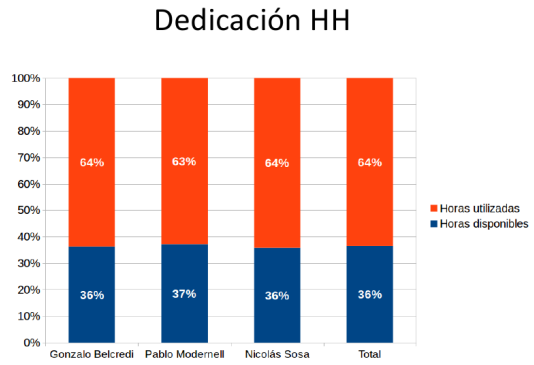


Figura D.3: Nivel de dedicación horaria a lo largo del proyecto, medido en horas semanales.



(a) Hito 1.



(b) Hito 2.

Figura D.4: Dedicación de horas-hombre (hitos 1 y 2).

incrementó notoriamente. No obstante, esto estaba en parte planificado desde el inicio, ya que se anticipaba que los integrantes podrían dedicar mayor tiempo al proyecto durante el año 2015.

D.2.3. Utilización de horas-hombre a lo largo del proyecto

En la Fig. D.4 se muestra el saldo de horas disponible al momento del Hito 1 y el Hito 2 respectivamente.

Al finalizar el proyecto la cantidad de horas-hombre totales dedicadas ascendió a:

Horas totales dedicadas: 1887 Horas-hombre.

Teniendo en cuenta que inicialmente se asignaron al proyecto 1598 horas, el sobrecosto asumido en horas dedicadas asciende a:

Sobre-costo en horas-hombre: 289 Horas-hombre.

Lo cual significa un 18 % del total estimado inicialmente.

D.3. Ritmo de avance a lo largo del proyecto

En las Figs. D.5 y D.6 se presentan dos gráficas que reflejan el nivel de avance del proyecto a los momentos de los Hitos 1 y 2 respectivamente. En ambas figuras, el peso de cada tarea con respecto al total corresponde a la estimación realizada al momento del análisis, esto es, al momento del Hito 1 y 2 respectivamente.

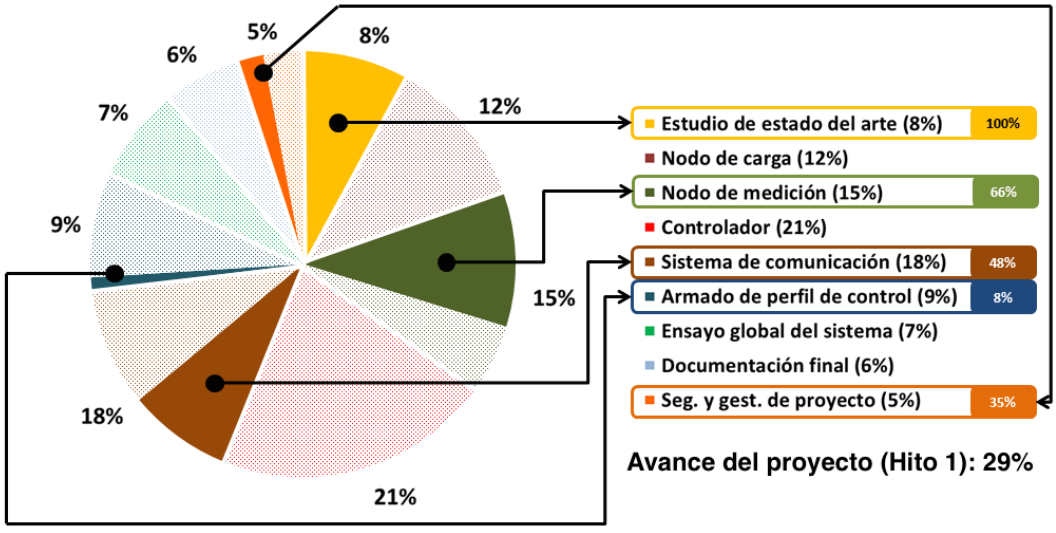


Figura D.5: Avances al momento del Hito 1.

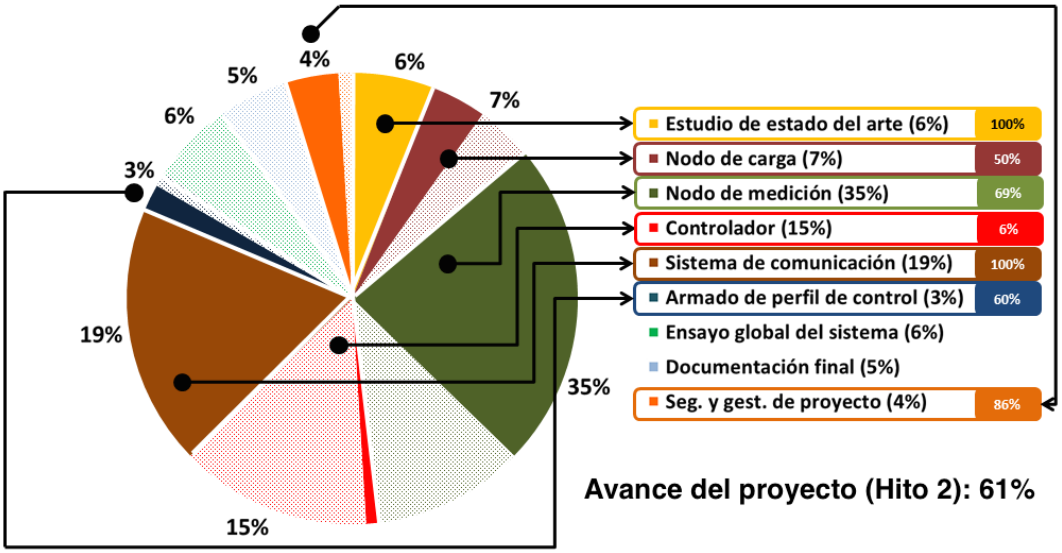


Figura D.6: Avances al momento del Hito 2.

Esta página ha sido intencionalmente dejada en blanco.

Referencias

[1] *A Low Cost Tamper-Resistant Energy Meter Based on the ADE7761 with Missing Neutral Function.*

[2] *DESIGNING WITH MICROCONTROLLERS IN NOISY ENVIRONMENTS.*

[3] *Implementation of a Single-Phase Electronic Watt-Hour Meter Using the MSP430F6736(A) (Rev. D).*

[4] Implementing An Electronic Watt-Hour Meter With MSP430FE42x(A).

[5] *Kinetis M One-phase power meter Design Reference Manual.*

[6] *MSP430F241x, MSP430F261x Mixed Signal Microcontroller (Rev. K).*

[7] *MSP430x2xx Family User's Guide (Rev. J).*

[8] *Zolertia Z1 Datasheet.*

[9] IEEE Guide for Smart Grid Interoperability of Energy Technology and Information Technology Operation with the Electric Power System (EPS), End-Use Applications, and Loads. Technical report, 2011.

[10] Omprakash G. ¡¿. The Minimum Rank with Hysteresis Objective Function. Technical Report 6719, RFC Editor, Fremont, CA, USA, September 2012.

[11] Pascal T. ¡¿. Objective Function Zero for the Routing Protocol for Low-Power and Lossy Networks (RPL). Technical Report 6552, RFC Editor, Fremont, CA, USA, March 2012.

[12] Hazrat Ali. A performance evaluation of rpl in contiki. Master's thesis, Swedish Institute of Computer Science (SICS), Sweden, School of Computing, Blekinge Institute of Technology, SE – 371 79 Karlskrona, Sweden, 2012.

[13] S. Aman, Y. Simmhan, and V. K. Prasanna. Energy management systems: state of the art and emerging trends. *Communications Magazine, IEEE*, 51(1):114–119, January 2013.

[14] S. Aman, Y. Simmhan, and V.K. Prasanna. Energy management systems: state of the art and emerging trends. *Communications Magazine, IEEE*, 51(1):114–119, January 2013.

[15] Emilio Ancillotti, Raffaele Bruno, and Marco Conti. The role of the RPL routing protocol for smart grid communications. *Communications Magazine, IEEE*, 51(1):75–83, January 2013.

[16] Galen L. Barbose, Charles A. Goldman, and Bernie Neenan. A Survey of Utility Experience with Real Time Pricing. Technical report, LBNL, 2004.

[17] Fundación Bariloche and MIEM DNE. Estudios de base para el diseño de estrategias y políticas energéticas: relevamiento de consumos de energía sectoriales en términos de energía útil a nivel nacional. INFORME DEL SECTOR RESIDENCIAL. Technical report, December 2008.

[18] E. Bloustein. Assessment of customer response to real time pricing. Technical report, Rutgers-The State University of New Jersey, 2005.

[19] Carsten Bormann, Angelo P. Castellani, and Zach Shelby. CoAP: An application protocol for billions of tiny internet nodes. *Internet Computing, IEEE*, 16(2):62–67, March 2012.

[20] Litos S. Communication. the SMART GRID:an introduction.

[21] Ruilong Deng, Zaiyue Yang, Mo-Yuen Chow, and Jiming Chen. A Survey on Demand Response in Smart Grids: Mathematical Models and Approaches.

[22] Pengwei Du and Ning Lu. Appliance Commitment for Household Load Scheduling. *Smart Grid, IEEE Transactions on*, 2(2):411–419, June 2011.

[23] Adam Dunkels. Full tcp/ip for 8-bit architectures. In *Proceedings of the 1st international conference on Mobile systems, applications and services*, pages 85–98. ACM, 2003.

[24] Raúl González Duque. Python para todos. 2011.

[25] Zhong Fan, P. Kulkarni, S. Gormus, C. Efthymiou, G. Kalogridis, M. Sooriyabandara, Ziming Zhu, S. Lambotharan, and Woon H. Chin. Smart Grid Communications: Overview of Research Challenges, Solutions, and Standardization Activities. *Communications Surveys & Tutorials, IEEE*, 15(1):21–38, 2013.

[26] Xi Fang, Satyajayant Misra, Guoliang Xue, and Dejun Yang. Smart Grid — The New and Improved Power Grid: A Survey. *Communications Surveys & Tutorials, IEEE*, 14(4):944–980, 2012.

[27] H. Farhangi. The path of the smart grid. *Power and Energy Magazine, IEEE*, 8(1):18–28, January 2010.

[28] Carles Gomez and Josep Paradells. Wireless home automation networks: A survey of architectures and technologies. *Communications Magazine, IEEE*, 48(6):92–101, June 2010.

[29] Nick J. Janaka Ekanayake. *Smart Grid: Technology and Applications*. Wiley, 2012.

[30] Dag B. Johan Westö. An Overview of Enabling Technologies for THE INTERNET OF THINGS. Technical report, Novia Univesity of Applied Sciences, 2014.

[31] Li Jun and Li Ling. Comparative research on python speed optimization strategies. In *Intelligent Computing and Integrated Systems (ICISS), 2010 International Conference on*, pages 57–59, Oct 2010.

[32] R. H. Lasseter. Smart Distribution: Coupled Microgrids. *Proceedings of the IEEE*, 99(6):1074–1082, June 2011.

[33] Yee W. Law, Tansu Alpcan, Victor C. S. Lee, Anthony Lo, Slaven Marusic, and Marimuthu Palaniswami. Demand Response Architectures and Load Management Algorithms for Energy-Efficient Power Grids: A Survey. In *Knowledge, Information and Creativity Support Systems (KICSS), 2012 Seventh International Conference on*, pages 134–141. IEEE, November 2012.

[34] Mark Lutz. *Learning python*. .ºReilly Media, Inc.", 2013.

[35] K. Nakasone, A. Seré, and P. Modernell. Home Energy Metering: Red de medición de consumo eléctrico. Technical report, Facultad de Ingeniería, Universidad de la República., 2014. Proyecto de fin de curso de la asignatura Sistemas embebidos para tiempo real.

[36] Travis E. Oliphant. Python for scientific computing. *Computing in Science Engineering*, 9(3):10–20, May 2007.

[37] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing (3rd Edition) (Prentice-Hall Signal Processing Series)*. Prentice Hall, 3 edition, August 2009.

[38] Liam Paull, Howard Li, and Liuchen Chang. A novel domestic electric water heater model for a multi-objective demand side management program. *Electric Power Systems Research*, 80(12):1446–1451, December 2010.

[39] Abiodun Iwayemi Peizhong Yi and Chi Zhou. Building automation networks for smart grids. *International Journal of Digital Multimedia Broadcasting*, 2011(2), April 2011.

[40] F. Salvadori, C.S. Gehrke, A.C. de Oliveira, M. de Campos, and P.S. Sausen. Smart grid infrastructure using a hybrid network architecture. *Smart Grid, IEEE Transactions on*, 4(3):1630–1639, Sept 2013.

[41] Anuj Sehgal. A Practical Introduction to 6LoWPANProgramming IPv6 Wireless Sensor Networks with Contiki. Technical report, Jacobs University Bremen, Germany, 2010.

[42] C. Severance. Guido van rossum: The modern era of python. *Computer*, 48(3):8–10, Mar 2015.

[43] Zach Shelby and Carsten Bormann. *6LoWPAN: The Wireless Embedded Internet*. Wiley Publishing, 2010.

[44] Nicolas Tsiftes, Joakim Eriksson, and Adam Dunkels. Low-power wireless IPv6 routing with ContikiRPL. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, IPSN '10, pages 406–407, New York, NY, USA, 2010. ACM.

[45] Pravin P. Varaiya, Felix F. Wu, and Janusz W. Bialek. Smart Operation of Smart Grid: Risk-Limiting Dispatch. *Proceedings of the IEEE*, 99(1):40–57, January 2011.

[46] Tim W. RPL: IPv6 routing protocol for Low-Power and lossy networks. Technical Report 6550, RFC Editor, Fremont, CA, USA, March 2012.

[47] Tom Williamson. Designing Microcontroller Systems for Electrically Noisy Environments. Technical report, 1993.

[48] Gang Xiong, Chen Chen, S. Kishore, and A. Yener. Smart (in-home) power scheduling for demand response on the smart grid. In *Innovative Smart Grid Technologies (ISGT), 2011 IEEE PES*, pages 1–7. IEEE, January 2011.

Índice de tablas

Esta página ha sido intencionalmente dejada en blanco.

Índice de figuras

Esta página ha sido intencionalmente dejada en blanco.

Contenido del CD

Plan de Proyecto

Presentaciones de los Hitos

Paper presentado al ISGT-LA 2015

Software

- Nodos
- HEC

Hardware

- Archivo fuente de los PCB
- Esquemáticos

Esta es la última página.
Compilado el martes 9 junio, 2015.
http://iie.fing.edu.uy/