ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD DE MÁLAGA

PROYECTO FIN DE CARRERA

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE GESTIÓN DE LA DEMANDA ELÉCTRICA EN VIVIENDAS INTELIGENTES

INGENIERÍA DE TELECOMUNICACIÓN

MÁLAGA, 2009 **MANUEL CASTILLO CAGIGAL**

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD DE MÁLAGA

Titulación: Ingeniería de Telecomunicación

Reunido el tribunal examinador en el día de la fecha, constituido por

D. _____

D. _____

D. _____

para juzgar el Proyecto Fin de Carrera titulado:

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE GESTIÓN DE LA DEMANDA ELÉCTRICA EN VIVIENDAS INTELIGENTES

del alumno D. Manuel Castillo Cagigal
dirigido por D. Alfredo García Lopera

ACORDÓ POR _____ OTORGAR LA CALIFICACIÓN DE _____

Y, para que conste, se extiende firmada por los componentes del tribunal, la presente diligencia

Málaga, a _____ de _____ de _____

El/La Presidente/a | El/La Vocal | El/La Secretario/a

Fdo: _____ Fdo: _____ Fdo: _____

ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD DE MÁLAGA

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE GESTIÓN DE LA DEMANDA ELÉCTRICA EN VIVIENDAS INTELIGENTES

REALIZADO POR:

D. Manuel Castillo Cagigal

DIRIGIDO POR:

D. Alfredo García Lopera

DEPARTAMENTO DE: Tecnología Electrónica

TITULACIÓN: Ingeniería de Telecomunicación

PALABRAS CLAVE: Gestión de la Demanda Eléctrica, Generación Distribuida Fotovoltaica, Sistema de Control.

RESUMEN: El presente proyecto desarrolla un sistema de control encargado de la Gestión de la Demanda Eléctrica en un prototipo de vivienda solar. La vivienda incorpora una fuente de generación fotovoltaica, baterías, una conexión a la red eléctrica y un sistema domótico para controlar los electrodomésticos. El sistema de control desarrollado permite aumentar la eficiencia energética de la vivienda planificando las tareas demandadas por el usuario, gracias a un aprovechamiento inteligente de la energía solar.

Málaga, Septiembre 2009

A mis padres Manuel Castillo y Encarna Cagigal,
y a mi hermana Blanca Castillo.

Agradecimientos

En estas líneas me gustaría, además de agradecer a todas las personas que me han ayudado durante mi carrera, hacer mención de la situación en la que se ha desarrollado este Proyecto Fin de Carrera.

Debido a mi interés por las tecnologías orientadas a la edificación y las energías renovables, busqué un proyecto fin de carrera relacionado con el tema. Durante el verano de 2008 me puse en contacto con Estefanía Caamaño, directora del proyecto GeDELOS-FV, y ella me propuso realizar mi PFC en colaboración con este. Ya que este proyecto pertenece al Instituto de Energía Solar de la UPM, me trasladé a Madrid donde conocí a Alvaro Gutierrez, mi tutor, del departamento de Tecnologías Aplicadas a las Telecomunicaciones de la UPM. En la UMA me puse en contacto con Alfredo García, el cual se ofreció a dirigir mi proyecto desde Málaga y ayudarme en todos los trámites referente a la ETSIT. Esta es la situación en la que he desarrollado mi Proyecto Fin de Carrera y he de mencionar que por ambas partes, tanto desde la UPM como de la UMA, he recibido un gran apoyo que me gustaría agradecer.

Quiero agradecer especialmente a Alvaro Gutierrez Martín, mi tutor desde la UPM, su apoyo y ayuda durante el desarrollo de este proyecto, sobre todo en esos momentos en que coincidió con la finalización de su tesis doctoral que fueron bastante duros.

Gracias a Alfredo García Lopera, mi tutor en la UMA, por su apoyo en todo momento en la realización de este proyecto y su ayuda a la hora de facilitarme la especial situación en la que me encontraba.

Gracias a Felix Monasterio Huelin, doctor en el departamento TEAT, por la ayuda y consejo que me siempre me ha ofrecido en el desarrollo de este proyecto.

Gracias a Estefanía Caamaño Martín, por ayudarme a encontrar este PFC y permitirme colaborar en el proyecto GeDELOS-FV, donde he aprendido mucho y me ha permitido trabajar en algo que realmente me gusta. También me gustaría agradecer su ayuda en el desarrollo de este PFC a Daniel Masa, doctorando en el IES.

Finalmente, quiero agradecer a mi familia todo el apoyo que siempre me han dado durante la realización de este PFC, en toda la carrera y en todos los momentos, tanto buenos como malos. Como no, no puedo finalizar estas líneas sin agradecer el apoyo de todos mis amigos tanto en Málaga como en Madrid que siempre han estado ahí cuando han hecho falta.

Muchas gracias a todos.

Índice general

Índice de figuras

Índice de tablas

Lista de Acrónimos

API: Application Programming Interface (interfaz de programación de aplicaciones).

GeDELOS-FV: Gestión de la Demanda ELéctrica dOméstica con tecnología Solar FotoVoltaica.

IES: Instituto de Energía Solar.

PLC: Power Line Communications (comunicaciones mediante cableado eléctrico).

SOAP: Simple Object Access Protocol (protocolo simple de acceso a objetos).

WSDL: Web Services Description Language (lenguaje de descripción de servicios web).

Capítulo 1

Introducción y Objetivos

1.1. Motivación

El presente proyecto está motivado por el desarrollo de un sistema de control para la Gestión de la Demanda Eléctrica con Generación de Energía Fotovoltaica. El sistema se ha instalado en una vivienda dotada de un sistema domótico que nos permite controlar diferentes cargas, principalmente electrodomésticos, en la vivienda.

El objetivo de este sistema es conseguir la mayor eficiencia energética, tanto mediante el máximo aprovechamiento de la energía fotovoltaica generada, como la reducción del consumo de energía de la red eléctrica. Se pretende que este consumo se produzca en las franjas horarias menos perjudiciales para la red eléctrica y con menor coste para el usuario.

Se pretende facilitar al usuario información sobre el balance energético de la vivienda. De tal manera que el usuario pueda conocer el comportamiento energético que tiene su vivienda, observando tanto la energía generada, como el efecto que tienen las diferentes cargas en el consumo eléctrico a diferentes horas del día. Esta información permite al usuario conocer que comportamientos son más eficientes energéticamente, y cuales son deseables de evitar.

1.2. Encuadre

El problema energético es un problema a nivel global, siendo Europa, y especialmente España, una de las zonas más sensibles debido a su alta dependencia de terceros para suplir sus necesidades. La búsqueda de soluciones es una de las principales preocupaciones en todos los países industrializados y son numerosas las maneras de responder a este problema. Existen dos líneas principales a seguir: cambiar la forma de generar la energía por recursos existentes en el mismo país (normalmente con el uso de energías renovables) y modificar el consumo, ya sea con su reducción o con una gestión apropiada de este.

La Gestión de la Demanda Eléctrica (Pérez et al., 2005) se perfila como una de las principales líneas de desarrollo (Paper, 2002) a la hora de mejorar la eficiencia energética. En las siguientes secciones veremos detalladamente el entorno actual donde se enmarca este Proyecto Fin de Carrera, el cual se encuentra dentro de un proyecto de I+D de plan nacional, GeDELOS-FV.

1.2.1. La demanda eléctrica en España

El análisis de la evolución de la demanda de energía en España en los últimos 20 años revela crecimientos continuos desde 1993 (véase Figura 1.1), con tasas de crecimiento promedio anual en el periodo 1996-2006 del 5 % (S.G.P.E., 2007), notablemente superiores al crecimiento del PIB (3,8 % en el mismo periodo) e indicativas de una baja intensidad energética de nuestra economía.

Por otra parte, del análisis de la evolución anual de la demanda en el sistema eléctrico peninsular, se puede constatar cómo los distintos usos eléctricos determinan las variaciones mensuales de consumo, siendo máximos durante los meses de diciembre y enero en invierno (puntas de invierno), y junio y julio en verano (puntas de verano) (Tecnalia, 2007). A parte de la evolución de la demanda eléctrica durante el año, hay que tener en cuenta la evolución de la demanda diaria, que presenta fuertes oscilaciones dependientes de la temperatura ambiente, la actividad laboral y la evolución de la actividad económica (R.E.E, 1998). Las variaciones del perfil de carga diario, aún

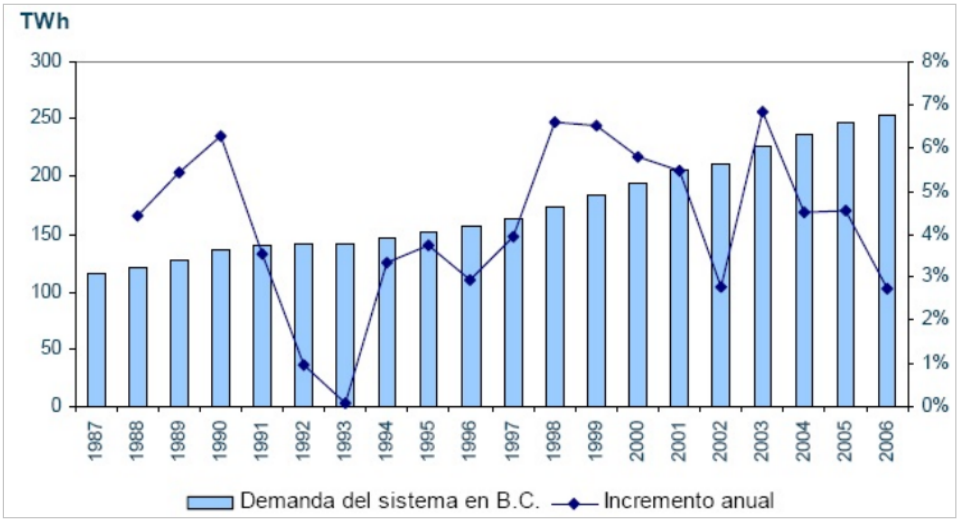


Figura 1.1: Crecimiento interanual de la demanda eléctrica.

siendo en principio aleatorias, pueden ser explicadas en gran parte por modelos basados en los patrones de consumo de los principales sectores de consumidores, los cuales unidos presentan un patrón general de la curva de carga, tal y como se puede observar en la Figura 1.2.

En lo que respecta a la cobertura de la demanda en el sistema peninsular, cabe destacar la contribución de la generación acogida al denominado "Régimen Especial" (producción procedente de energías renovables y cogeneración), lo que confirma el crecimiento continuado experimentado por este tipo de generación en los últimos años (R.E.E, 2006).

A diferencia de otros productos energéticos, la electricidad no es un producto almacenable a gran escala, por lo que el suministro requiere un equilibrio instantáneo entre la generación y la demanda. Como consecuencia, las infraestructuras de generación y red necesarias se dimensionan para asegurar el suministro eléctrico en las horas de máxima demanda. Las elevadas tasas de crecimiento del consumo en los últimos años han generado situaciones comprometidas para el sistema eléctrico. Ello obliga a incrementar cada año la capacidad del sistema para poder atender a la demanda en todo momento, si bien una parte importante de la generación sólo se

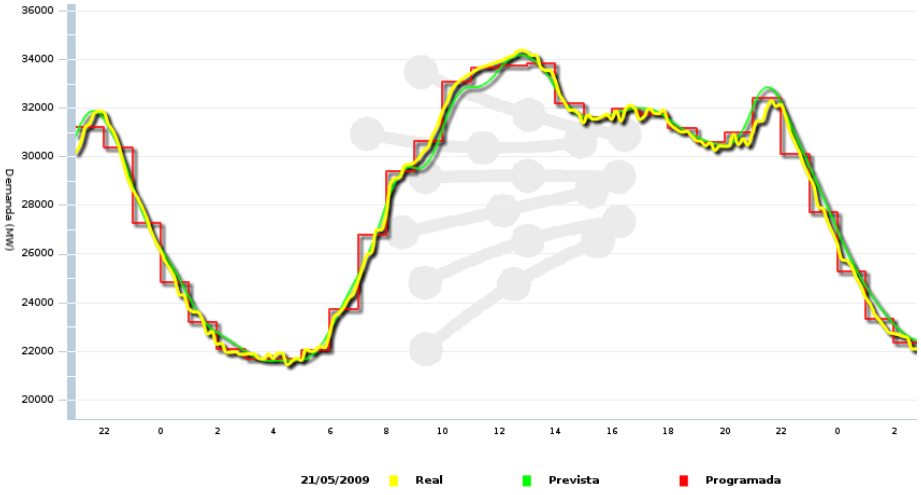


Figura 1.2: Curva de carga del sistema eléctrico para el 21/5/2009.

utiliza unas pocas horas, con el consiguiente coste económico, social y medioambiental (Imaz, 2007). Las consecuencias de este hecho, se traducen en variaciones importantes del precio de la electricidad, que en un mismo día puede variar en un factor de 5:1 entre horas de mayor y menor demanda.

Por otra parte, el sistema eléctrico español se enfrenta a importantes desequilibrios regionales derivados de un modelo de generación fuertemente centralizado. Esto implica que la electricidad se genera a grandes distancias de los puntos de consumo. Así, en zonas donde existe mucha demanda y poca generación pueden existir problemas de congestión de la red debido a la saturación de las líneas de transporte y distribución. Esto conlleva riesgos para la seguridad y calidad del consumo, además de las pérdidas de energía que provoca el transporte de electricidad (en torno al 9 % en España)(Valentí, 2007).

De las diversas alternativas existentes para evitar los riesgos que pueden causar estos desequilibrios, hasta ahora la más empleada por los gestores de la red eléctrica han sido trabajos de mejora en el sistema eléctrico. Otras medidas identificadas, que guardan una estrecha relación con el presente proyecto son:

- **La incorporación de Generación Distribuida cerca de los lugares de**

consumo que, además de reducir la congestión de las líneas, permita disminuir las pérdidas eléctricas en el sistema debidas al transporte y distribución.

- **Una gestión eficaz de la Demanda Eléctrica** que permita disminuir los grandes picos de consumo eléctrico, de forma que los recursos del sistema eléctrico se utilicen con mayor eficiencia. En este sentido, una de las características más determinantes de las redes eléctricas del futuro será la capacidad de los consumidores de desempeñar un papel activo en la cadena del suministro eléctrico.

Respecto a la demanda eléctrica anual de los hogares españoles, esta ha experimentado un aumento sustancial en las dos últimas décadas, pasando de una tasa de crecimiento anual promedio del 1,3 % en el periodo 1989-1996 a un 5,7 % en el periodo 1998-2006 (D.R.E.D., 2007) como resultado del aumento de la población, un mayor equipamiento de los hogares, la escasa proporción en la adquisición de electrodomésticos eficientes y los bajos precios de la electricidad.

1.2.2. Generación fotovoltaica distribuida

Históricamente, los sistemas de suministro eléctrico se han diseñado siguiendo un esquema de integración vertical, sin embargo en los últimos años estamos asistiendo al crecimiento de la Generación Distribuida (GD, pequeños generadores dispersos por las redes eléctricas) para satisfacer las necesidades energéticas de un mundo cada vez más interconectado (Degner et al., 2006). Las ventajas de la Generación Distribuida puede verse desde diferentes perspectivas:

- Perspectiva estratégica: Ante la elevada dependencia de muchos países de combustibles fósiles para la generación eléctrica y el envejecimiento de las infraestructuras de transporte y distribución.

- Perspectiva económica: Debido a la liberalización de los mercados eléctricos y nuevos desarrollos tecnológicos.

- Perspectiva medioambiental: Debido a las emisiones de gases de efecto invernadero y otro contaminantes derivadas del uso de combustibles fósiles y su influencia en el cambio climático.

Los beneficios que producen la GD con Energía Fotovoltaica alcanzan un mayor valor si tiene lugar en las zonas donde se produce la demanda eléctrica y especialmente en áreas urbanizadas (Linder, 2007), donde se aprovecha la capacidad de integración arquitectónica que tienen los módulos fotovoltaicos. Los principales beneficios (Groppi, 2002) son:

- En términos instantáneos: reducción de pérdidas de transporte y distribución, mejoras en la calidad y continuidad del suministro en horas de alta demanda eléctrica y reducción de impactos medioambientales.

- A medio y largo plazo: reducción de la capacidad adicional necesaria para atender las puntas de consumo y distribución de inversiones futuras necesarias para aumentar la capacidad de las redes, sobre todo en baja tensión.

Desde el punto de vista técnico, la experiencia con tecnologías de GD que han alcanzado elevados niveles de penetración en las redes, muestra que es necesario disponer de mecanismos de control para que la GD contribuya a la estabilidad del sistema y de forma especial en el caso de que ocurrieran fallos o problemas de suministro (Caamaño et al., 2007).

En lo que respecta a la energía solar fotovoltaica, en los últimos años la tecnología de inversores ha evolucionado hasta el punto de poder ofrecer las soluciones que proporcionan beneficios para las redes eléctricas tales como:

- **Mejoras de la calidad de suministro:** a través del funcionamiento de los inversores a modo de filtros activos para reducir la distorsión armónica presente en las redes (Bruendlinger et al., 2007).

- **Regulación del factor de potencia, potencia reactiva y control de los niveles de tensión:** Inversores dotados de almacenamiento de energía y un

sistema de control adecuados son capaces de producir o absorber potencia reactiva, compensando el exceso o falta de potencia reactiva en la red y contribuyendo así al control del nivel de tensión (Wakao et al., 2005).

- **Control de simetría de fase:** Los inversores trifásicos equipados con almacenamiento de energía puede mejorar la calidad de la red inyectando corrientes de fase asimétricas que permitan ecualizar las fases de la red.

- **Estabilización de la red y funcionamiento en isla intencionado:** Los inversores con sistema de almacenamiento y un sistema de control adecuado son capaces de controlar activamente el funcionamiento del inversor de forma que contribuya a: estabilizar la tensión y frecuencia de red, mantener un funcionamiento en isla controlado (aislando la instalación de la red), apoyar el funcionamiento de aplicaciones específicas.

Los sistemas fotovoltaicos integrados en viviendas pueden ser de varios tipos:

- Sistemas aislados: Son propio de lugares remotos a la que no llega la red de suministro eléctrico. Constan de un generador fotovoltaico, una batería, un regulador de carga y mecanismo de protección. En el caso de que se quieran emplear cargas en AC se incluye un inversor. Es habitual complementar la instalación con un pequeño generador diésel.

- Sistemas con conexión a red: Es el tipo de sistema más habitual en viviendas en España y resto de Europa. La red eléctrica de suministro es de fácil acceso y la instalación fotovoltaica se conecta directamente a ella. El sistema fotovoltaico esta compuesto por el generador, inversor de conexión a red y elementos de medida y protección.

- Sistemas híbridos: Están conectados a la red de suministro eléctrico al mismo tiempo que constan de algún elemento que permite almacenar energía, generalmente baterías.

De especial interés para la gestión de la demanda son los sistemas híbridos. Combinan las ventajas de la GD con la posibilidad de funcionar en modo autónomo, bien debido a un fallo de red, bien como decisión del usuario. Dentro de los sistemas híbridos es posible encontrar dos topologías distintas: sistemas de acoplo en AC y sistemas de acoplo en DC. Los sistemas de acoplo en AC son más eficientes energéticamente, su diseño es más modular y escalable y permiten la conexión de mayor potencia fotovoltaica. Este último será el utilizado en este Proyecto Fin de Carrera.

1.2.3. Tarificación eléctrica en España

El suministro de energía eléctrica en España se encuentra en un proceso de liberalización iniciado el año 1997, el cual ha supuesto modificaciones sustanciales en las opciones de contratación por parte de los consumidores (B.O.E.).

Para el presente proyecto tendremos en cuenta principalmente la información actual sobre la discriminación horaria (véase Figura 1.3). Existen diferentes tarifas dependiendo de la hora del día, que serán tenidas en cuentan por el sistema de control para minimizar los costes derivados del consumo eléctrico.

Periodo horario	Duración
Horas punta	4 horas/día
	Zona 1: Invierno: 18-22; Verano: 11-15
Horas llano	12 horas/día
	Zona 1: Invierno: 8-18+22-24; Verano: 8-11+15-24
Horas valle	8 horas/día
	Zona 3: Invierno: 0-8; Verano: 0-8

Figura 1.3: Horarios de tarificación.

1.2.4. Sistemas de Control

La Ingeniería de Control tiene una larga historia (Bennett, 1979), actualmente está presente en muchos campos tanto de la ciencia como de la ingeniería permitiendo

mejorar la monitorización y el control de los sistemas donde es utilizada.

La convergencia del control, la comunicación y la computación (Graham and Kumar, 2003) se plantea en la actualidad como una integración de las tecnologías desarrolladas en cada campo específico, en el que se toma en consideración la integración con el entorno (Vasilakos and Pedriyzc, 2006). Esta convergencia señala la importancia de las arquitecturas frente a los algoritmos concretos de resolución de tareas, cuestión que ya había sido señalada desde los comienzos de las teorías de máquinas inteligentes (Valavanis and Saridis, 1992).

La mayoría de los recientes estudios domóticos (Romero, 2004) toman en consideración la totalidad de la vivienda, pero casi siempre exclusivamente en términos de comunicación de datos, dejando de lado tanto los aspectos de control como los de las fuentes de energía.

Sin embargo en recientes trabajos (Liang and Du, 2008) se presenta un sistema de control inteligente resolviendo de manera integral el problema de la calefacción, ventilación y aire acondicionado, tomando en consideración el problema de reducción de la potencia eléctrica.

1.2.5. GeDELOS-FV

El presente Proyecto Fin de Carrera, se encuentra enmarcado dentro de un proyecto de plan Nacional, GeDELOS-FV (Gestión de la Demanda ELéctrica dOméstica con tecnología Solar Fotovoltaica) que pretende explorar las posibilidades de gestionar la demanda eléctrica doméstica mediante tecnologías de generación distribuida y sistemas de control, donde el sistema es capaz de modificar la curva de demanda en función de condicionantes de tipo interno y externos. El proyecto pretende implementar un sistema de control de cargas de la vivienda teniendo en cuenta información propia de la casa y la proveniente de fuentes externas, consiguiendo con todo ello una optimización en el consumo eléctrico de la vivienda sin que exista una pérdida de funcionalidad.

Además del sistema de control, la vivienda dispone de un sistema fotovoltaico

encargado de los siguientes objetivos: i)supervisión del funcionamiento del sistema fotovoltaico, ii)estimación de la producción eléctrica y iii)control del funcionamiento de la instalación fotovoltaica.

En la Figura 1.4 se muestra un esquema de la estructura completa de GeDELOS-FV. El proyecto se ha desarrollado en la vivienda "MagicBox", la cual dispone de todo el equipamiento necesario, para el control y monitorización.

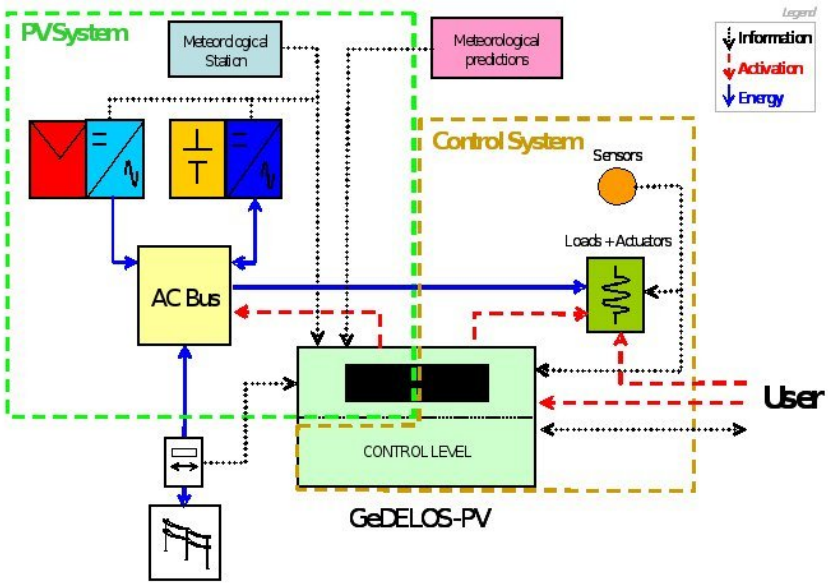


Figura 1.4: Estructura de GeDELOS-FV.

1.2.6. Magicbox

En otoño del 2005 se celebró en Washington D.C. la segunda edición del concurso internacional Solar Decathlon (Moon et al., 2005) patrocinado por el Departamento de Energía de los Estados Unidos. Solar Decathlon es un concurso dirigido a universidades que persigue difundir la posibilidad de conciliar las buenas prácticas arquitectónicas con un uso racional de la energía, a través del aprovechamiento de la energía solar tanto pasiva como activamente y del uso de tecnologías eficientes

actualmente disponibles.

El prototipo "MagicBox" (ver Figura 1.5) constituye una propuesta novedosa que combina la aplicación de principios de diseño bioclimáticos con la integración de tecnologías solares para la producción de electricidad y agua caliente sanitaria (solar fotovoltaica y solar térmica, respectivamente). El uso de las tecnologías de la información y las comunicaciones para el mantenimiento de las variables de confort constituye el tercer elemento innovador del prototipo del que disponemos. Asimismo, cuenta con un sistema de acondicionamiento novedoso basado en el calor latente y el cambio de estado de sustancias (denominadas "geles de cambio de estado") que se encuentran confinadas bajo el suelo de la casa, las cuales se cargan y descargan de energía mediante corrientes de aire inducidas.

Figura 1.5: Imagen de la fachada sur de "MagicBox".

De especial interés para el presente proyecto es el sistema fotovoltaico diseñado. Este incorpora la tecnología más avanzada de microgeneración, denominada

"acoplamiento en AC", que garantiza el máximo aprovechamiento de la energía solar (generadores fotovoltaicos con distintas inclinaciones para aplanar la curva anual de generación, inversores con seguimiento automático de la máxima potencia del generador), al tiempo que posibilita la incorporación de otras tecnologías de generación auxiliares, en este caso acumuladores electroquímicos y la conexión a red tanto para obtener energía de esta como para verter en ella el excedente. La potencia instalada en la casa es de 8,1 kW de potencia nominal, producida por 77 módulos los cuales ocupan 63 m^2 de cubierta y 9 m^2 de la fachada sur y una capacidad en las baterías de 90 kWh.

En el interior, la vivienda está equipada con una lavadora, secadora, lavavajillas, horno, frigorífico/congelador y campana de extracción todos ellos de bajo consumo y dotados de un sistema de control domótico conectados mediante una red PLC. Gracias a una pasarela, podemos comunicarnos y obtener tanto información de los electrodomésticos, como enviar órdenes básicas de funcionamiento. Aparte de estos electrodomésticos controlables tendremos diferentes cargas que no están dotadas de este sistema como los ordenadores, luces, calefacción, aire acondicionado y vitrocerámica.

1.3. Objetivos del proyecto

El objetivo principal del proyecto es realizar un sistema de control, cuyas entradas sean los datos provenientes del sistema fotovoltaico, la carga de las baterías, la red eléctrica y el sistema domótico de la vivienda, y que realice un planificación de las cargas eléctricas durante el día, dando prioridad a los requisitos del usuario. Con dicha planificación se persigue reducir el consumo eléctrico, no sólo evitando el consumo de la red e intentando que éste lo suplan las fuentes de generación propias, sino que este se produzca cuando sea más conveniente para la red eléctrica, en horas de baja tarificación consiguiendo también una eficiencia económica.

Se pretende también, que tanto la arquitectura del sistema como el software de control sean fácilmente escalable. Esto permitirá en un futuro, el desarrollo de

otras funciones internas, incorporando algoritmos más complejos (previsiones más amplias, sistemas de aprendizaje, etc.), aumentando el número de entradas y salidas y mejorando los subsistemas de la vivienda, con sistemas domóticos más complejos y nuevas funcionalidades.

Para conseguir estos objetivos se ha dividido el proyecto en los siguientes subobjetivos:

- Control del sistema domótico: Conocimiento del sistema y generación de las librerías necesarias para su posterior uso por el software de control.
- Caracterización de las cargas: Medición de los consumos de las cargas y conocimiento de su funcionamiento para pronosticar su efecto energético en la vivienda.
- Obtención de datos externos: Información proveniente del sistema fotovoltaico, baterías y red eléctrica para que puedan ser usadas por el software de control.
- Definición de la arquitectura: Esquematizar el funcionamiento del sistema y definir las tareas de cada parte del software.
- Comprobación de mejora en la eficiencia energética: Analizar los resultados obtenidos por el sistema de control para comprobar la mejora en la eficiencia energética.

En resumen, con el desarrollo del sistema de control pretendemos acercarnos a los objetivos del proyecto GEDELOS-FV. Con esta primera versión del sistema de control, se perfila la arquitectura del sistema y se obtienen los primeros resultados respecto a la eficiencia energética buscada.

1.4. Organización del proyecto

En el Capítulo 1 se ha descrito la motivación y encuadre del proyecto final de carrera dentro de un proyecto del plan nacional más amplio (GeDELOS-FV). Igualmente, se han definido los objetivos a llevar a cabo en el proyecto.

En el Capítulo 2, se desarrolla la arquitectura de control, que se encarga de la previsión de la demanda eléctrica de la vivienda. Esta arquitectura se ha definido como una estructura dividida en dos partes fundamentales, una distribuida y otra centralizada.

En el Capítulo 3, se desarrollan las librerías de comunicación con las diferentes cargas del sistema, así como diferentes funciones de monitorización del mismo.

En el Capítulo 4, caracterizamos las diferentes cargas del sistema. Medimos los diferentes consumos de cada uno de los electrodomésticos en función de los programas a utilizar y las variables de entrada. Estos datos serán utilizados como entradas por el sistema de control, para realizar las previsiones de consumo.

En el Capítulo 5, se desarrolla una aplicación práctica. Se analizan los resultados de diferentes planificaciones reales, llevadas a cabo en la vivienda y observamos el efecto de las cargas en el sistema en diferentes momentos del día.

Finalmente, en el Capítulo 6, se muestran las conclusiones y líneas futuras.

Capítulo 2

Arquitectura del Sistema

En el presente capítulo se describe la arquitectura del sistema, tanto funcional como del software implementado. En la Sección 2.1 se introduce la estructura del hardware, con el objetivo de mostrar el sistema eléctrico de la vivienda y su red de datos.

En la Sección 2.2 se desarrolla la arquitectura del sistema, la cuál se divide en dos partes principales: una distribuida y otra centralizada. Se detallan los pasos necesarios para realizar la planificación de las cargas, que entradas son necesarias del exterior y cómo se procesan cada una de ellas.

Finalmente, en la Sección 2.3 se desarrolla la estructura del software que implementa la arquitectura, así como las principales funciones utilizadas.

2.1. Estructura hardware

Como se ha comentado en el Capítulo 1, el sistema se ha implementado en la vivienda "MagicBox", la cual dispone de todo el hardware necesario para poder llevar a cabo de forma real una gestión de la demanda eléctrica.

2.1.1. Conexionado eléctrico

El conexionado eléctrico de la casa dispone de un circuito independiente para cada electrodoméstico, facilitando el estudio del consumo de los mismos y la localización de posibles fallos en el sistema eléctrico. Al tener una fuente de generación fotovoltaica y

unas baterías de acumulación, el sistema eléctrico de la vivienda incorpora inversores, reguladores, etc. encargados del control eléctrico. (Ver Figura 2.1).

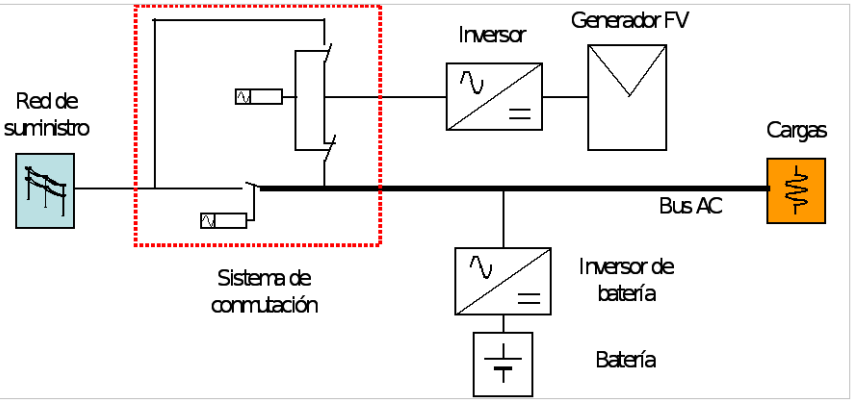


Figura 2.1: Sistema híbrido con acoplo en AC de la vivienda "MagicBox".

2.1.2. Red de datos

La red de datos dispone de dos partes principales. Una es la red PLC donde se encuentran conectados los electrodomésticos controlables, la cuál se desarrolla en el Capítulo 3. Por otro lado, tenemos una red de datos ethernet (Spurgeon, 2000) que comunica todos los PCs de la vivienda.

2.2. Arquitectura del sistema de control

El sistema de control planifica el momento de trabajo de cada uno de los electrodomésticos teniendo en cuenta restricciones temporales del usuario, donde las entradas son los datos del sistema fotovoltaico, el estado de las baterías y el estado de la red, con el objetivo de conseguir la mayor eficiencia energética posible.

Se ha diseñado un sistema de control con una parte distribuida y otra centralizada. La primera planifica las acciones a realizar, mientras que la segunda es la encargada de coordinar y ejecutar los resultados de la parte distribuida. La arquitectura del sistema puede verse en la Figura 2.2.

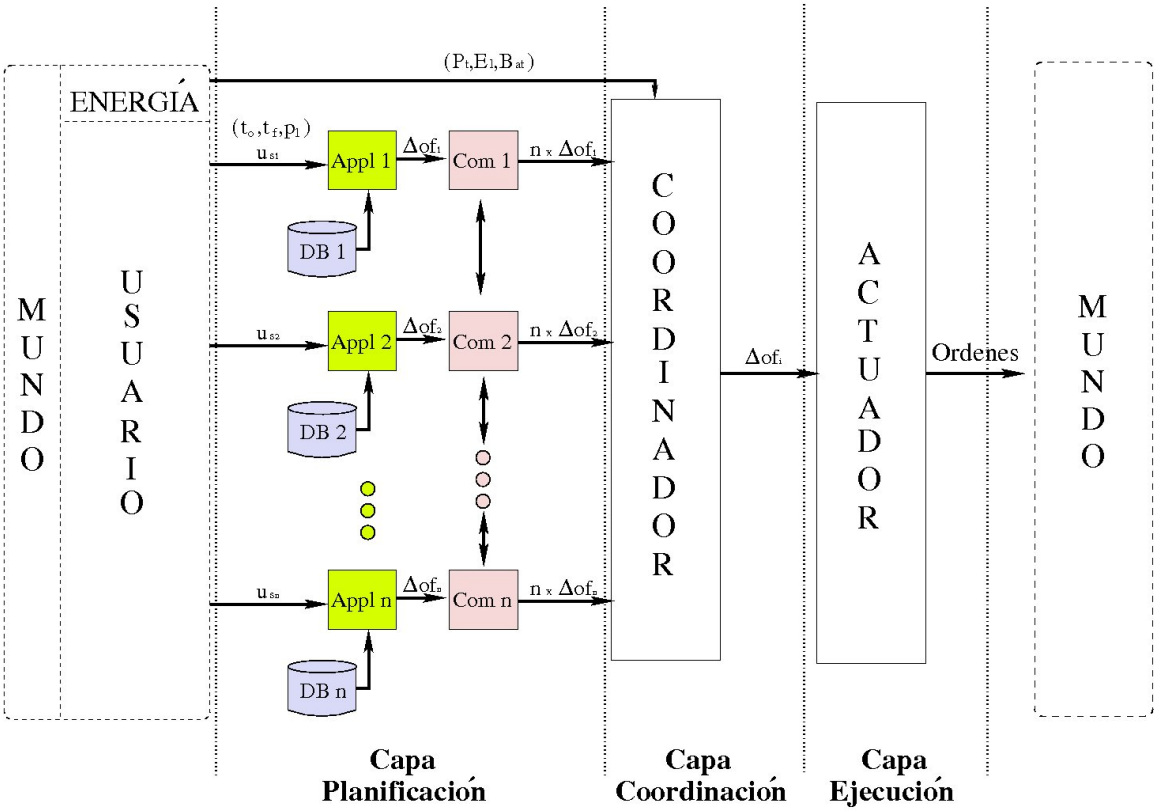


Figura 2.2: Arquitectura del sistema.

2.2.1. Parte distribuida

El sistema de control está compuesto por diferentes subsistemas con características diferentes. Se considera como subsistema al conjunto tanto físico como lógico que representan las cargas controlables (electrodomésticos). En el nivel físico encontramos las cargas eléctricas y en el nivel lógico, los objetos controlados por nuestro software. Dichos objetos contienen información sobre el funcionamiento, consumos y otras variables utilizadas por el sistema de control.

El objetivo del sistema es ejecutar las órdenes del usuario minimizando el consumo de la red eléctrica en favor del uso de la energía fotovoltaica, dando prioridad al usuario, aunque algunas de sus órdenes provoquen una disminución en la eficiencia

energética buscada.

Con todo esto, se plantea la primera parte de la arquitectura como una parte distribuida que sólo tiene información del usuario, y no atiende a necesidades o preferencias energéticas.

Cada subsistema recibe información sobre su tarea a realizar y sus límites temporales definidos por el usuario. Los límites temporales son la hora a la que puede comenzar a trabajar y cuando es necesario que acabe su tarea. Estos son impuestos por el usuario como órdenes de entrada al igual que el tipo de tarea (programa) a realizar por el subsistema. Por ejemplo, activar la lavadora entre las 10 a.m. y 5 p.m., con un programa a 90°C y 1200 rpm.

Una vez planificada su tarea, cada subsistema envía la información de su tarea al resto de subsistemas. De esta manera, cada subsistema planifica y genera un eje de tiempos donde incluye el resto de tareas a realizar por el sistema de control. Un ejemplo gráfico del eje temporal puede verse en la Figura 2.3. Como resultado de este proceso obtenemos una planificación completa por cada subsistema.

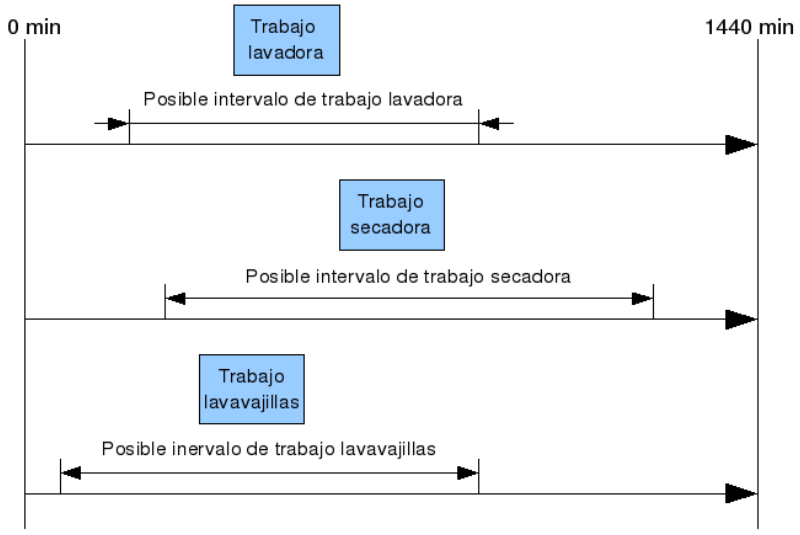


Figura 2.3: Ejemplo de límites temporales.

2.2.2. Parte centralizada

Las diferentes planificaciones generadas en la parte distribuida (una por cada electrodoméstico) se pasan a la parte centralizada. El objetivo de ésta es escoger la planificación más óptima desde el punto de vista energético para ser ejecutada.

En dicha elección, se utiliza la información del sistema fotovoltaico, para prever la producción energética, la tabla de coste del kWh de la red eléctrica proveniente del contador y el estado de carga de baterías en el momento de comienzo de las planificaciones. Toda esta información permite hacer una previsión del balance energético de la vivienda de las próximas 24 horas. Una vez acabada la previsión energética de cada una de las planificaciones, se ejecuta la más óptima.

2.3. Estructura del software

El software que implementa la arquitectura ha sido dividido en tres capas (planificación, coordinación y ejecución), con el objetivo de dividir las diferentes partes del cálculo de la planificación y previsión energéticas.

2.3.1. Capa de planificación

Esta capa pertenece a la arquitectura distribuida del sistema. Esta implementada en dos estructuras software: el planificador, programa que realiza la planificación con los datos que dispone y la estructura de comunicación en anillo, encargada de que todos los subsistemas reciban la información de todas las tareas.

En esta capa, cada subsistema realiza la planificación con los datos de las diferentes órdenes suministradas por el usuario. Estos datos consisten en la tarea a ejecutar con sus diferentes fases y duración y los límites temporales impuestos por el usuario (ver Figura 2.2).

Las tareas a realizar por los diferentes electrodomésticos, son a su vez separables en fases independientes, que pueden ser realizadas en momentos diferentes, dependiendo del electrodoméstico y teniendo en cuenta que la tarea completa finalice dentro de los

límites temporales impuestos por usuario. Estas fases se definen a partir del estudio del funcionamiento de cada electrodoméstico concreto. Por ejemplo, la lavadora queda dividida en tres fases: lavado, aclarado y centrifugado, pudiendo detenerse el funcionamiento de la lavadora al final de cada fase, para posteriormente ser reanudado, sin que se modifique el funcionamiento de la siguiente fase.

Inicialmente cada subsistema genera un eje de tiempos y planifica su tarea dentro del mismo. Posteriormente, cada subsistema envía su información al resto de los subsistemas mediante una comunicación en anillo. Cuando un subsistema recibe una nueva información, la planifica en su eje de tiempos. Este proceso se repite hasta que todos los subsistemas hayan concluido la planificación de todas las tareas impuestas por el usuario.

Un ejemplo real de tres planificaciones diferentes se describe en el Capítulo 5.

2.3.1.1. Planificador

El planificador irá recibiendo la información de las diferentes fases por separado y las irá insertando en la planificación. El siguiente código es la función de actualización de la planificación:

```
int CdishwCom::update(int sender){
   comPhase auxphase;
   int auxtime;
   int auxpos;
   //Si el eje de tiempos esta vaci'o inserta la propia carga.
   if (sheduling.ncomPhases == 0){
      copyshed (sheduling, &myshed);
      return 1;
   }
   //Si los datos recibidos son nuevos inserta la nueva
   //carga en el eje de tiempos.
   if ( searchid(sheduling, sender, 1) == -1){
      for(int i = 0; i < rxshed->ncomPhases; i++){
         //Busca un hueco vacio en la planificacion.
         if (rxshed->comPhase[i].identification != 0){
            auxphase = rxshed->comPhase[i];
            //Localiza un hueco libre suficientemente grande.
            auxtime = searchSpace (sheduling, auxphase);
            if (auxtime == -1)
               //Si no hay espacio en todo el eje de tiempos da error.
```

```
                std::cout << "NO HAY ESPACIO" <<std::endl;
            else{
                //Localizado un hueco valido inserta la nueva carga.
                auxphase.iniTime = auxtime;
                auxphase.endTime = auxtime + auxphase.duration;
                auxpos = position(sheduling, auxtime);
                insertinSpace(sheduling, auxphase, auxpos);
            }
        }
    }
}
```

Hay que tener en cuenta que el primer paso es la inserción en el eje de tiempos de la tarea completa (todas las fases) de uno mismo. Como se puede observar en el algoritmo mostrado, sólo se tendrán en cuenta los datos temporales del programa, por lo que se insertará la carga en el primer hueco disponible sin tener en cuenta valores energéticos.

El planificador tiene la capacidad de planificar las tareas fase por fase, de forma que éstas puedan ser divididas. Actualmente la implementación trabaja con tareas completas, planificándose todas sus fases una de tras de otra.

2.3.1.2. Comunicación en anillo

Recibe la información del anillo y decide que información enviar al próximo subsistema, asegurando que todos los miembros del anillo reciben la información necesaria para realizar la planificación completa. Debemos mencionar, que la información enviada corresponde a tareas completas y no a fases independientes.

A continuación mostramos el código de la comunicación en anillo:

```
int Ccommunication::send(struct Sheduling* input, int sender){
    rxshed = input;
    //Carga inicialmente la propia planificacion.
    if (sheduling.ncomPhases == 0){
        lastshed = &myshed;
        txshed = &myshed;
        return 1;
    }
    //Decide que informacion enviar en el anillo.
```

```
    if ( searchid(sheduling, sender, 1) == -1)
        //Si no se dispone de esa informacion envia la
        //anterior que recibio.
        txshed = lastshed;
    else
        //Si se dispone de esa informacion se la envia
        //al siguiente.
        txshed = rxshed;
    lastshed = rxshed;
}
```

En la Figura 2.4 se puede observar un ejemplo de la comunicación en anillo con tres subsistemas. En dos iteraciones todos los subsistemas tendrán la información de los demás.

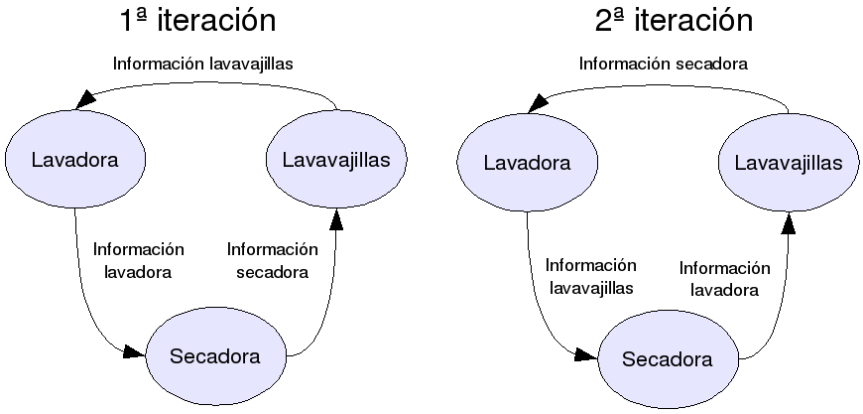


Figura 2.4: Ejemplo de comunicación del anillo.

2.3.2. Capa de coordinación

Esta capa pertenece a la parte centralizada de la arquitectura. Es la encargada de recibir las diferentes planificaciones realizadas por los diferentes subsistemas en la capa de comunicación y decidir cuál de ellas va a ejecutarse finalmente.

Para decidir qué planificación es la más eficiente, la capa de coordinación realiza un balance de energía para cada una de las planificaciones, donde se incluye la información del consumo de las cargas, la previsión de generación fotovoltaica, la

carga de baterías y la información de la red eléctrica. El algoritmo implementado selecciona la planificación con mayor aprovechamiento de energía fotovoltaica. Dicha selección se ha implementado como se muestra en el siguiente código:

```
Sheduling* CSheduler::selection(){
    //Inicializa las variables.
    int output = 0;
    consumption oldconsum;
    oldconsum.solarEn = 0;
    oldconsum.gridEn  = 0;
    oldconsum.price   = 0.0;
    //Analiza una a una las planificaciones.
    for (int i = 0; i < allshed.size ; i++){
        //Evalua la planificacion haciendo el balance
        //de energias.
        ShedEvaluation(allshed.iniShed[i], i);
        std::cout << output << std::endl;
        //Observa si hay una planificacion con mayor
        //consumo fotovoltaico.
        if (Energy.solarEn > oldconsum.solarEn){
            output = i;
            oldconsum = Energy;
        }
        //Vuelve a iniciar la energia a cero para la
        //proxima planificacion
        Energy.solarEn   = 0.0;
        Energy.gridEn    = 0.0;
        Energy.price     = 0.0;
        Energy.batcharge = 1.0;
    }
    //Devuelve la planificacion mas eficiente.
    return allshed.iniShed[output];
}
```

Para calcular el balance energético de una planificación, el coordinador simula minuto a minuto el balance energético de la vivienda teniendo en cuenta su funcionamiento en tiempo real. El coordinador observa la tarea activa y obtiene la potencia instantánea proveniente de la base de datos de los electrodomésticos generada previamente (ver Capítulo 4). Una vez obtenida la potencia consumida por la tarea, el coordinador obtiene la potencia generada por los paneles fotovoltaicos. A partir de ese momento se plantean dos opciones:

- La potencia fotovoltaica generada es mayor que la consumida, por lo que la carga queda cubierta y la potencia sobrante se inyecta en baterías.
- La potencia fotovoltaica es menor que la consumida, en este caso la fotovoltaica solamente cubrirá parte de la potencia necesaria. Con el resto de la potencia requerida se plantean dos opciones:
  - Hay suficiente energía en la batería, la potencia necesaria la entrega la batería.
  - No hay energía suficiente en la batería, será necesario tomar energía de la red para cubrir las necesidades del sistema.

Respecto a la batería hay que tener en cuenta algunos aspectos de su funcionamiento que se explican en el Capítulo 3. Conviene mencionar en este punto, que si la potencia instantánea requerida de la batería es superior a su límite de 5 kW, el resto de la potencia necesaria será tomada de la red, aún cuando haya suficiente energía almacenada.

Una vez finalizado el balance de cargas, se informa al usuario de los diferentes consumos para cada una de las planificaciones, y se informa tanto al usuario como a la capa de ejecución, cual es el que se debe ejecutar.

2.3.3. Capa de ejecución

Esta capa pertenece a la parte centralizada de la arquitectura. La capa ejecución es la encargada de llevar acabo la planificación elegida por la capa de coordinación. Tras recibir esta planificación, se encarga de enviar las órdenes necesarias a cada uno de los electrodomésticos (a las horas planificadas) para ejecutar los programas definidos por el usuario.

2.3.4. Interfaz con el mundo

Es la interfaz entre el usuario y el sistema, será la encargada de preguntar al usuario qué electrodomésticos quiere utilizar, con qué programa (diferentes

parámetros dependiendo del electrodoméstico) y el intervalo temporal deseado.

Tras realizar el balance de energía y obtener una planificación, el sistema ofrece al usuario información sobre la actuación de las cargas y los consumos. En el Capítulo 5 se mostrarán ejemplos de los balances de potencia y energía entregados al usuario para una planificación completa.

2.4. Resumen

En este capítulo se ha descrito el sistema de control desarrollado en este proyecto. El sistema de ha dividido en dos partes fundamentales: una distribuida y otra centralizada.

En la parte distribuida sólo se tienen en cuenta datos referentes a la información temporal, con el objetivo de cumplir los requisitos del usuario y que este no se vea perjudicado por el hecho de utilizar un sistema de control de la demanda. En la parte centralizada se realiza un balance de potencia y energía para poder decidir qué planificación es más eficiente energéticamente. Esta información puede ser consultada por el usuario permitiendo hacer un estudio del sistema energético de la vivienda y permitirle conocer mejor las características energéticas de la misma.

Finalmente, se han descrito las diferentes capas de la arquitectura (planificación, coordinación y ejecución). Estas implementan el funcionamiento global y realizan las funciones utilizadas tanto para planificar las tareas como para evaluar el balance energético.

Capítulo 3

Sistema de Comunicación

El principal componente de la arquitectura del sistema es la comunicación con los diferentes dispositivos de la vivienda. Dichos dispositivos son:

- **Cargas:** Son los elementos de consumo de la vivienda, siendo los electrodomésticos los únicos dispositivos controlables.
- **Sistema fotovoltaico:** Mediante la comunicación con el sistema fotovoltaico dispondremos de la información necesaria sobre la generación energética que éste produce.
- **Contador:** Es el encargado de obtener medidas energéticas globales de toda la vivienda, además de la información de la red eléctrica.
- **Baterías:** Son los elementos acumuladores de la vivienda.

Para la comunicación con los diferentes sistemas, se han desarrollado unas librerías en C++ (Aguilar, 2002), donde cada dispositivo es un objeto de la clase.

En este capítulo explicaremos las librerías y clases desarrolladas así como la información que manejan.

3.1. Comunicación con las cargas

La comunicación con las cargas se realizará con el sistema domótico de Siemens "server@Home", siendo un sistema privado desarrollado por esta empresa y que

actualmente se ha dejado de comercializar. El sistema se compone de diferentes electrodomésticos: lavadora, secadora, frigorífico/congelador, horno, lavavajillas y campana. Todos estos dispositivos incorporan modems PLC (Power Line Communication) (Huidobro, 2003) que se comunican con un PLINT (Power Line Interface) a través de la red eléctrica de la casa. El sistema de control se comunica con una pasarela que accede directamente al PLINT, y la cual usaremos como un servidor web con las APIs (Application Programming Inteface) necesarias para usar en nuestro programa (Fournier, 1999). La arquitectura de este sistema se muestra en la Figura 3.1.

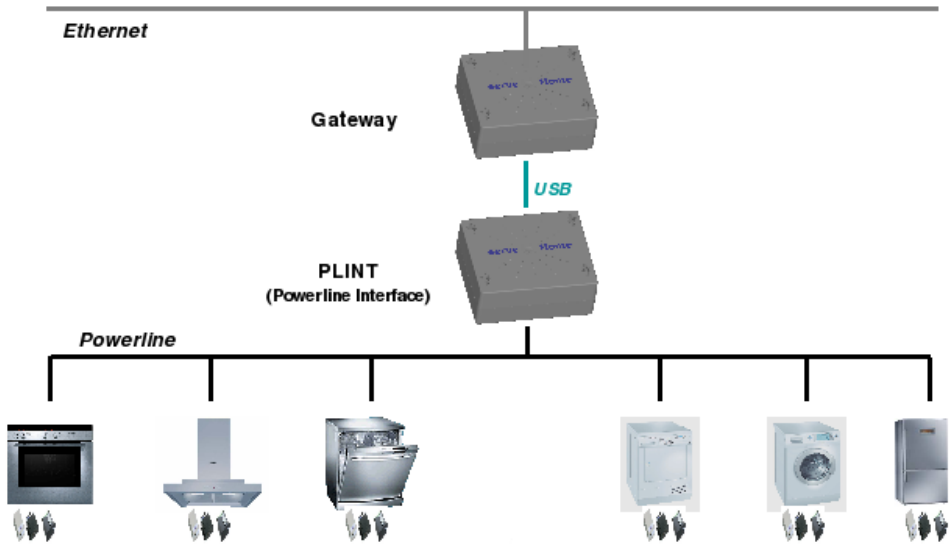


Figura 3.1: Arquitectura de la red domótica.

Las diferentes APIs nos permiten tanto obtener datos de los electrodomésticos como enviar una serie de órdenes básicas. Además, dispone de diferentes eventos que nos informan de sucesos que ocurren durante el funcionamiento de los electrodomésticos. Todas las APIs se encuentran en una serie de librerías dentro del programa principal.

Sobre estas APIs, hemos desarrollado las librerías de comunicación que nos simplifican el control de los electrodomésticos, así como el acceso a información

relacionada con el consumo energético. Queremos resaltar, que los datos de consumo (energía y potencia) de los electrodomésticos, no son accesibles desde los mismos, al no encontrarse disponibles en las APIs. Por lo tanto, hemos tenido que caracterizar cada uno de los electrodomésticos (ver Capítulo 4) para incorporar su modelo al sistema de control.

En los siguientes apartados explicamos la creación de las librerías encargadas de comunicarse con el servidor y posteriormente, por encima de estas, las librerías utilizadas directamente en nuestro sistema. En la Figura 3.2 se muestra la estructura de las librerías de comunicación.

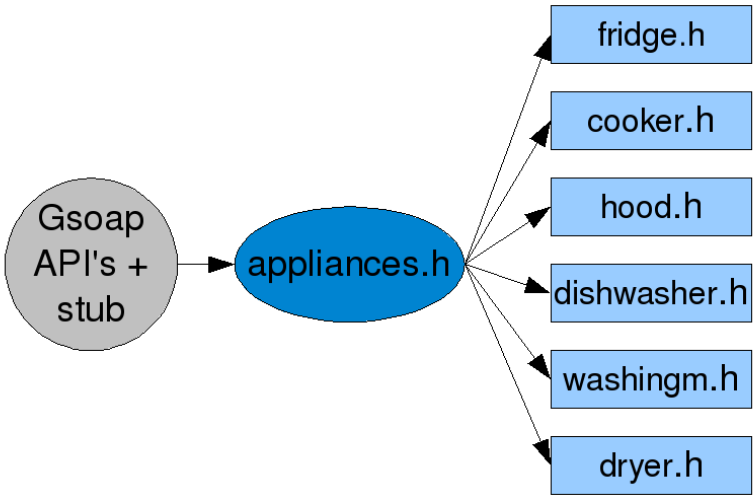


Figura 3.2: Estructura de librerías de comunicación con los electrodomésticos.

3.1.1. Implementación de las librerías de comunicación

Como hemos comentado anteriormente, las librerías de comunicación desarrolladas nos permiten comunicarnos directamente con la pasarela y enviar las órdenes necesarias. Estos comandos pueden ser tanto lectura de datos como acciones a realizar por los electrodomésticos.

Primero debemos implementar las librerías con el "stub"[1] necesario para la comunicación y una clase que contenga todas las APIs de la pasarela. Para el desarrollo de estas librerías hemos utilizado la herramienta *gSoap*, la cual nos facilita la obtención de las APIs de un servidor, en nuestro caso la pasarela, y la creación del "stub". *GSoap* se comunica directamente con un servidor donde se encuentra la interfaz WSDL, obtiene las estructuras de datos y el formato de las funciones para comunicarnos con los electrodomésticos. El funcionamiento de esta herramienta puede verse de forma resumida en el Apéndice A y con más detalle en su manual oficial (van Engelen, 2008).

El primer paso a realizar para el uso de esta herramienta es pedir la estructura de datos y las funciones de acceso al servidor. Esto se realiza mediante la instrucción "wsdl2h" a la que se le suministra la dirección de la pasarela:

```
wsdl2h -o webserver.h http://<IP>:8080/sah-ws
```

En el fichero "webserver.h"podemos ver las estructuras y funciones del sistema. Sin embargo, éstas no pueden ser utilizadas aún por nuestro programa ya que es necesario realizar una compilación cruzada mediante "soapcpp2". Una vez compilado, se generan las librerías de comunicación que nos permitirán tomar control de la pasarela para obtener datos y enviar órdenes a los electrodomésticos.

Aparte de obtener la información de los electrodomésticos, podemos modificar los tipos de datos que nos entregan las funciones y con los que trabajarán las estructuras. Debido a que *gSoap* es un sistema genérico, no siempre los tipos de datos que nos devuelve son los adecuados. Por lo tanto, deberemos modificarlos antes de compilar las librerías finales para trabajar de una forma más cómoda.

En nuestro caso y a modo de ejemplo, se ha modificado el tipo de dato "xsd_anyType*"por una cadena de caracteres "std::string" en el fichero "webserver.h", ya que entran en conflicto a la hora de leerlo nuestro programa:

```
xsd_anyType* ------>std::string
```

[1] Un stub es un procedimiento de biblioteca en una comunicación cliente/servidor. Stub del cliente: representa al procedimiento servidor en el espacio de direcciones del cliente.

Otra modificación necesaria es la dirección de comunicación, es decir la dirección de la pasarela. En nuestro caso, la dirección generada no es correcta y deberemos cambiarla, en este caso el cambio deberá producirse en dos lugares, dentro del fichero "webserver.h":

```
http://testkueche.homeip.net:8888/sah-ws --->http://<IP>:8080/sah-ws
```

Una vez que tenemos configurado el fichero "webserver.h", podemos pasar a compilarlo con la herramienta "soapcpp2" y obtendremos las librerías a introducir en nuestro programa junto con los archivos de configuración. Para realizar dicha acción, utilizaremos la siguiente instrucción:

```
soapcpp2 -i -C -I/home/control/gsoap/gsoap-2.7/gsoap/import webserver.h
```

La dirección que aparece en la llamada al compilador es donde se encuentra el fichero "stlvector.h" en nuestro equipo, para ver el motivo de esto se puede recurrir al manual de *gSoap* (van Engelen, 2008).

Las librerías que tenemos ahora, las necesarias por el motor de gSoap y el fichero de configuración "soapdefs.h"(generado por nosotros para evitar problemas en la compilación con las "autotools"[2] (Ray, 2000)) se incorporarán al directorio de nuestro sistema. Dichas librerías y ficheros de comunicación, se encuentran en los siguientes archivos:

- SAHBinding.nsmap
- soapC.cpp
- soapH.h
- soapdefs.h
- soapSAHBindingProxy.h

[2]Autotools es un conjunto de herramientas producido por el proyecto GNU. Estas herramientas están diseñadas para ayudar a crear paquetes de código fuente portable a varios sistemas Unix.

- soapSAHBindingProxy.cpp
- soapStub.h
- stdsoap2.h
- stdsoap2.cpp

En nuestro programa sólo tendremos que incluir "soapSAHBindingProxy.h", para tener acceso a los electrodomésticos. Al incluir esta librería, podemos declarar el objeto "SAHBindingProxy service". Este objeto es el encargado de comunicarse con la pasarela y actúa de puente entre nuestro código y la pasarela.

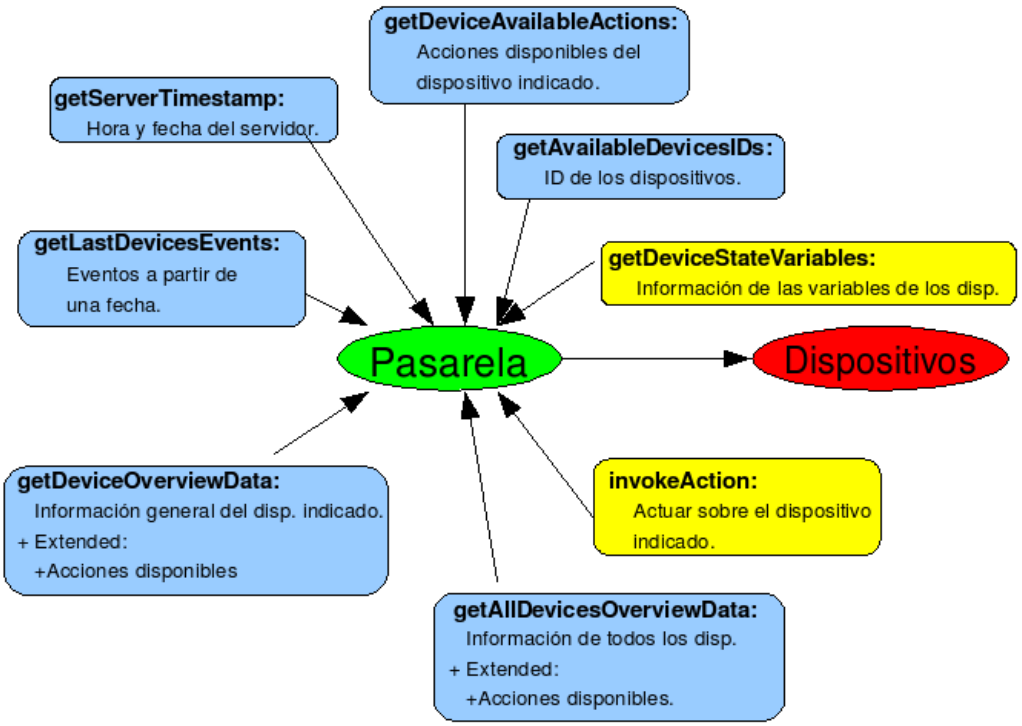


Figura 3.3: APIs de la pasarela.

Las diferentes APIs existentes se muestran en la Figura 3.3. Durante el funcionamiento de nuestro programa usaremos básicamente dos funciones: `getDevice`

`StateVariables`, que nos entrega el valor de las variables de los electrodomésticos e `invokeAction`, la cual nos permite enviar órdenes a los electrodomésticos. La función `getLastDeviceEvents` también podrá entregarnos información en tiempo real sobre eventos ocurridos en los electrodomésticos, pero ésta no tendrá mucha relevancia en un estudio energético de los mismos ya que están orientados a informar al usuario sobre posibles problemas de mantenimiento (por ejemplo si hay o no detergente en el lavavajillas). El resto de funciones contendrán información sobre el sistema en general y se utilizarán para la implementación del programa.

3.1.2. Estructuras de datos

Los datos que intercambiaremos con la pasarela dependen de las APIs. Como ya hemos explicado en el apartado anterior, las APIs son introducidas en la clase generada por *gSoap*, que incluyen una serie de funciones con diferentes estructuras de datos como parámetros de entrada y salida.

Cada electrodoméstico tiene una serie de variables especificas, que debemos pedir a la pasarela mediante la instrucción: `getDeviceStateVariable(param1, param2, respuesta)`. Los parámetros de entrada serán:

- *param1*: el ID del dispositivo.
- *param2*: un array de cadenas de caracteres, cada una de estas cadenas con el nombre de la variable que deseamos leer.

Como resultado esta API nos devolverá una estructura de datos de la siguiente forma:

```
struct ns1__getDeviceStateVariablesResponse{ //Estructura de la respuesta.
public:
        ArrayOfDeviceStateVariable *return_;
};
class SOAP_CMAC ArrayOfDeviceStateVariable : public xsd__anyType {
public:
        ns1__DeviceStateVariable **__ptr;
        int __size;
};
```

```
class SOAP_CMAC ns1__DeviceStateVariable : public xsd__anyType {
public:
        std::string variablename;
        std::string value;
};
```

Las diferentes variables generales de los electrodomésticos se muestran en la Tabla 3.1, mientras que las específicas de cada electrodoméstico las vemos en las tablas 3.2-3.7.

Nombre	Tipo	Descripción
remoteEnabled	Entero	Estado del control remoto
deviceState	Entero	Estado actual del dispositivo
deviceStatus	Cadena	Información sobre el estado actual del dispositivo
deviceType	Cadena	Tipo de dispositivo

Tabla 3.1: Variables generales.

Nombre	Tipo	Descripción
targetTemperatureKF	Entero	Temperatura seleccionada del frigorífico
targetTemperatureGF	Entero	Temperatura seleccionada del congelador
KFCompartment	Bool	Muestra si el frigorífico esta activo
GFCompartment	Bool	Muestra si el congelador esta activo
functionKF	Entero	Función seleccionada del frigorífico
functionGF	Entero	Función seleccionada del congelador
superKF	Bool	Modo “supercooling.ªctivo o no
superGF	Bool	Modo “superfreezing.ªctivo o no

Tabla 3.2: Variables del Frigorífico/congelador.

Nombre	Tipo	Descripción
washingParameters	Entero	Programa de lavado seleccionado
currentPhase	Entero	Fase actual
targetTemperature	Entero	Valor de la temperatura seleccionada
Spin	Entero	Valor de las revoluciones seleccionadas
finishTime	Entero	Momento de finalización del programa

Tabla 3.3: Variables de la lavadora.

Nombre	Tipo	Descripción
washingParameters	Entero	Programa de lavado seleccionado
currentPhase	Entero	Fase actual
remainingTime	Entero	Tiempo restante del programa
startTime	Entero	Momento de comienzo del programa

Tabla 3.4: Variables del lavavajillas.

Nombre	Tipo	Descripción
washingParameters	Entero	Programa de lavado seleccionado
currentPhase	Entero	Fase actual
remainingTime	Entero	Tiempo restante del programa
startTime	Entero	Momento de comienzo del programa

Tabla 3.5: Variables de la secadora.

Nombre	Tipo	Descripción
heatingMode	Entero	Modo de horneado
targetTemperature	Entero	Temperatura seleccionada
endTime	Entero	Momento del final del programa
durationTime	Entero	Duración del programa
childLock	Bool	Muestra si el bloqueo esta activo o no

Tabla 3.6: Variables del horno.

Nombre	Tipo	Descripción
fanSpeed	Entero	Velocidad del extractor
lightOnOffDim	Entero	Intensidad de la luz

Tabla 3.7: Variables de la campana.

Para realizar diferentes acciones en los electrodomésticos deberemos utilizar la función `invokeAction` con la siguiente declaración:

```
virtual int invokeAction(std::string deviceID, std::string actionName,
                         ArrayOfString *parameters,
                         struct ns1__invokeActionResponse &_param_6);
```

- *deviceID*: el ID del electrodoméstico que queremos usar.
- *actionName*: el nombre de la acción (o función) que queremos realizar.
- **parameters:* los parámetros que necesita la función que vamos a realizar.
- *&_param_6*: la respuesta de la función `invokeAction`, está vacía de momento.

La llamada a esta función depende de cada electrodoméstico y de cada acción en concreto, lo que hace difícil trabajar directamente con ella. Por ello y para el paso de las variables de los electrodomésticos a variables de nuestros objetos se han implementado diferentes librerías, que se detallan a continuación.

3.1.3. Librerías de los eletrodomésticos

Las librerías de los electrodomésticos se han desarrollado para simplificar su control y no depender directamente de las funciones de *gSoap*. De esta manera, al llamarlas desde nuestro programa, no tenemos que preocuparnos de posibles cambios de tipo de datos o desbordamientos de memoria. Además, el uso de *gSoap* es transparente para el usuario, quien sólo tiene interés en el control de los electrodomésticos.

Hemos desarrollado una librería por cada uno de los electrodomésticos, que heredan de la clase “Cappliances”, la cual se encuentra en el fichero “appliances.h”.

Esta librería contiene funciones generales usadas por todos los electrodomésticos tales como las de cambios de tipos o la declaración del objeto encargado de la comunicación. La Figura 3.2 muestra un diagrama software de las librerías de los electrodomésticos y la declaración de la clase general se detalla a continuación:

```
#include <soapSAHBindingProxy.h>  //Libreria de comunicacion.

class Cappliances {
    public:
        //Constructor
        Cappliances();

        //Variables
        SAHBindingProxy service;  // Objeto para la comunicacion.
        int remoteEnabled;        // Informacion general de los
        int deviceState;          // dispositivos.
        int deviceStatus;         //
        int working;              //
        int finished;             //
        int timediff;             // Error de tiempo entre el
                                  // servidor y nuestro equipo.

        //Funciones
        int  toint  (string& input);
        bool tobool (string& input);
        void tochar (int input1, char *input2);
        void tochain(string& input1, char *input2);
        int  events (char *device, int timeev, ns1__MessageEvent& output);
};
```

Las variables globales de los electrodomésticos contienen información común a todos ellos en relación a su estado de funcionamiento. Estas variables son usadas para una posterior coordinación de los electrodomésticos. En las siguientes secciones se detallan las diferentes clases de control de cada uno de los electrodomésticos.

3.1.3.1. Frigorífico y congelador

La clase frigorífico/congelador se ha desarrollado de acuerdo a la siguiente declaración:

```
class Cfridge : public Cappliances {
    public:
```

```
        Cfridge();
        ~Cfridge();

        //Variables
        int tempKF;
        int tempGF;
        int functionKF;
        int functionGF;
        bool KFComp;
        bool GFComp;
        bool superKF;
        bool superGF;

        //Funciones
        int update(void);
        int coolingtemp (int input);
        int freezingtemp(int input);
        int supercooling(bool input);
        int superfreezing(bool input);
        float readPower(int min);
};
```

Las variables de este dispositivo contienen información sobre la temperatura actual del frigorífico (*coolingtemp*) y del congelador (*freezingtemp*), si están activados los compartimentos o si están activados los modos "supercooling" y/o "superfreezing". A continuación se describen cada una de sus funciones:

- **int update(void):** Realiza la actualización de las variables, pide el estado actual de todas las variables del frigorífico a la pasarela y las actualiza en nuestro objeto. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **int coolingtemp(int input):** Establece la temperatura del frigorífico. El rango de temperatura pertenece al intervalo [2ºC, 11ºC] con saltos de 1ºC. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **int freezingtemp(int input):** Establece la temperatura del congelador. El rango de temperatura pertenece al intervalo [-26ºC, -18ºC] con saltos de 1ºC.

Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **int supercooling(bool input):** Conecta o desconecta ('true' o 'false' como variable de entrada) el modo "supercooling". Dicho modo pone el frigorífico a 2ºC a la máxima velocidad posible. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **int superfreezing(bool input):** Conecta o desconecta ('true' o 'false' como variable de entrada) el modo "superfreezing". Dicho modo pone el congelador a -28ºC a la máxima velocidad posible. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **float readPower(int min):** Esta función nos entrega la potencia instantánea de consumo en un minuto determinado. Dicha información proviene de la base de datos generada a partir de las medidas realizadas (ver Capítulo 4). Es usada para realizar las predicciones de consumo (ver Capítulo 5).

3.1.3.2. Lavadora

La clase lavadora se ha desarrollado de acuerdo a la siguiente declaración:

```
class Cwashingm : public Cappliances {
    public:
        Cwashingm();
        ~Cwashingm();

        //Variables
        int washparam;
        int currentphase;
        int temp;
        int spin;
        int finishtime;

        //Funciones
        int update(void);
        int start(void);
        int pause(void);
        int stop (void);
        int temperature(int input);
```

```
        int program(int input);
        int setspin(int input);
        float readPower(int min);
};
```

La lavadora contiene como información el programa de lavado a realizar (*washparam*), la fase actual dentro del programa de lavado (*currentphase*), la temperatura a la que está preparado el programa (*temp*), las revoluciones del centrifugado del programa (*spin*) y el tiempo final (*finishtime*).

A continuación se describen cada una de sus funciones:

- **int update(void):** Realiza la actualización de las variables, pide el estado actual de todas las variables de la lavadora a la pasarela y las actualiza en nuestro objeto. Como variables están tanto las propias de la lavadora antes mencionadas como las generales de los electrodomésticos. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **int start(void):** Comienza el lavado con el programa (*washparam*), temperatura (*temp*) y revoluciones (*spin*) indicados previamente con las funciones `program`, `temperature` y `setspin` respectivamente. En el caso de que el lavado haya sido interrumpido con la función `pause`, esta función reanudará el lavado a partir de la fase en la que se encontraba. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **int pause(void):** Detiene el lavado temporalmente hasta que lo reanudemos con la función `start` o lo cancelemos con la función `stop`. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **int stop(void):** Cancela el lavado actual. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **int temperature(int input):** Indica la temperatura a la que deberá trabajar en el próximo lavado (no se podrá modificar la temperatura durante el lavado). Las posibles temperaturas son: lavado en frío, 30°C, 40°C, 50°C, 60°C, 70°C,

80ºC y 90ºC. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **int program(int input):** Indica el programa de lavado al que se ejecutará. Existen 18 programas posibles desde el 0 hasta el 17. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **int setspin(int input):** Indica las revoluciones a las cuales se realizará el centrifugado. Los posibles valores son: 400 rpm, 500 rpm, 600 rpm, 700 rpm, 800 rpm, 900 rpm, 1000 rpm, 1100 rpm, 1200 rpm, 1300 rpm, 1400 rpm, 1500 rpm y 1600 rpm. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **float readPower(int min):** Esta función nos entrega la potencia instantánea de consumo en un minuto determinado. Dicha información proviene de la base de datos generada a partir de las medidas realizadas (ver Capítulo 4). Es usada para realizar las predicciones de consumo (ver Capítulo 5).

Para el correcto funcionamiento de la lavadora es necesario tener en cuenta las siguientes consideraciones:

- Es necesario poner la lavadora en modo "@" cada vez que se quiere iniciar un programa nuevo.

- Si se para un programa con la función `stop` saldremos del modo "@" y será necesario volver a activarlo.

- En caso de que se abra la puerta también se saldrá del modo "@".

- El tiempo de finalización (*finishtime*) será respecto al reloj de la lavadora, no del servidor.

3.1.3.3. Lavavajillas

La clase del lavavajillas se ha desarrollado conforme a la siguiente declaración:

```
class Cdishwasher : public Cappliances {
    public:
        Cdishwasher ();
        ~Cdishwasher();

        //Variables
        int washparam;
        int currentphase;
        int remtime;
        int starttime;

        //Funciones
        int update(void);
        int program(int input);
        int alarmoff(void);
        int start(void);
        int stop(void);
        float readPower(int min);
};
```

El lavavajillas contendrá la información del programa de lavado a realizar o realizándose (*washparam*), la fase actual del lavado (*currentphase*), tiempo que queda para finalizar el programa (*remtime*) y momento del comienzo del programa (*starttime*).

Las funciones son las siguientes:

- **int update(void):** Realiza la actualización de las variables, pide el estado actual de todas las variables del lavavajillas a la pasarela y las actualiza en nuestro objeto. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **int program(int input):** Indica el programa a realizar por el lavavajillas, hay 9 posibles desde el 1 al 9. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **int alarmoff(void):** Desactiva la alarma sonora del lavavajillas que suena cuando el programa ha finalizado. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **int start(void):** Comienza el lavado con el programa anteriormente indicado con la función `program`. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **int stop(void):** Cancela el lavado actual. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **float readPower(int min):** Esta función nos entrega la potencia instantánea de consumo en un minuto determinado. Dicha información proviene de la base de datos generada a partir de las medidas realizadas (ver Capítulo 4). Es usada para realizar las predicciones de consumo (ver Capítulo 5).

Para el correcto funcionamiento del lavavajillas es necesario tener en cuenta las siguientes declaraciones:

- Cuando finalice un programa o lo cancelemos con la función `stop` saldremos del modo "@", por lo que deberemos volver a activarlo manualmente.

- El parámetro tiempo de comienzo (*stratTime*) se corresponde con los minutos del servidor cuando se configuró el programa.

3.1.3.4. Secadora

La clase de la secadora se ha desarrollado conforme a la siguiente declaración:

```
class Cdryer : public Cappliances {
    public:
        Cdryer();
        ~Cdryer();

        //Variables
        int washparam;
        int currentphase;
        int remtime;
        int starttime;
        int spin;

        //Funciones
        int update(void);
        int start (void);
```

```
        int pause (void);
        int stop (void);
        int setspin (int input);
        float readPower(int min);
};
```

Las variables de la secadora contiene información sobre el programa de secado a realizar (*washparam*), la fase actual (*currentphase*), el tiempo restante de secado (*remtime*), el momento de comienzo del programa (*starttime*) y las revoluciones a la que trabajará la secadora (*spin*).

Las funciones son las siguientes:

- **int update(void):** Realiza la actualización de las variables, pide el estado actual de todas las variables de la secadora a la pasarela y las actualiza en nuestro objeto. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **int start(void):** Comienza el secado con el programa (*washparam*) y las revoluciones (*spin*), indicadas previamente con la función `setspin`. En el caso de que el secado haya sido interrumpido con la función `pause`, esta función reanuda el secado a partir de la fase en la que se encontraba. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **int pause(void):** Detiene el secado temporalmente hasta que lo reanudemos con la función `start` o lo cancelemos con la función `stop`. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **int stop(void):** Cancela el secado actual. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **int setspin(int input):** Indica las revoluciones a las cuales se realizará el secado. Los posibles valores son: 800 rpm, 1000 rpm, 1200 rpm, 1400 rpm y 1600 rpm. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **float readPower(int min):** Esta función nos entrega la potencia instantánea de consumo en un minuto determinado. Dicha información proviene de la base de datos generada a partir de las medidas realizadas (ver Capítulo 4). Es usada para realizar las predicciones de consumo (ver Capítulo 5).

Sobre el funcionamiento de la secadora:

- El programa que queramos se deberá introducir de forma manual, para posteriormente pasar al modo "@".
- La variable *starttime* se corresponde con los minutos del servidor cuando se configuró el programa.
- Al finalizar o cancelar un programa saldremos del modo "@".

3.1.3.5. Horno

La clase del horno se ha desarrollado conforme a la siguiente declaración:

```
class Ccooker : public Cappliances{
    public:
        Ccooker();
        ~Ccooker();

        //Variables
        int  heatingmode;
        int  temp;
        int  endtime;
        int  durtime;
        bool childlock;

        //Funciones
        int update(void);
        int reset(void);
        int normalheating(int inputmode, int inputtemp);
        int completeheating(int inputmode, int inputtemp, int inputdur,
                            int inputend);
        float readPower(int min);
};
```

Las variables del horno contiene información sobre el modo de trabajo seleccionado (*heatingmode*), la temperatura de trabajo (*temp*), momento de finalización del

programa (*endtime*), duración del programa (*durtime*) y si esta activado o no el bloqueo de niños (*childlock*).

Las funciones son las siguientes:

- **int update(void):** Realiza la actualización de las variables, pide el estado actual de todas las variables del horno a la pasarela y las actualiza en nuestro objeto. Como variables están tanto las propias del horno como las generales de los electrodomésticos. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **int reset(void):** Reinicia el horno terminando el trabajo actual y volviendo al modo de espera. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **int normalheating(int inputmode, int inputtemp):** Como primer parámetro de la función se indica el modo en el que queremos trabajar. Existen 22 modos, definidos del 0 al 21. El segundo parámetro indica la temperatura a la que queremos trabajar con un rango de 30ºC a 300ºC con saltos de 1ºC. Al pasar estos parámetros al horno, comenzará a trabajar hasta que le indiquemos que se detenga con la función `reset`. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **int completeheating(int inputmode, int inputtemp, int inputdur, int inputend):** Como primer parámetro de la función se indica el modo en el que queremos trabajar. Existen 22 modos, definidos del 0 al 21. El segundo parámetro indica la temperatura a la que queremos trabajar con un rango de 30ºC a 300ºC con saltos de 1ºC, el tercer parámetro indica la duración en minutos que queremos que dure el trabajo y el cuarto parámetro indica el momento que queremos que finalice el trabajo. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **float readPower(int min):** Esta función nos entrega la potencia instantánea de consumo en un minuto determinado. Dicha información proviene de la base de datos generada a partir de las medidas realizadas (ver Capítulo 4). Es usada para realizar las predicciones de consumo (ver Capítulo 5).

3.1.3.6. Campana

La clase de la campana se ha desarrollado conforme a la siguiente declaración:

```
class Chood : public Cappliances {
    public:
        Chood ();
        ~Chood();

        //Variables
        int fanspeed;
        int light;

        //Funciones
        int update(void);
        int setlight(int input);
        int setfan(int input);
        int stop(void);
        float readPower(int min);
};
```

Las variables de la campana contienen información sobre la velocidad a la que esta funcionando el ventilador (*fanspeed*) y la intensidad actual de la luz (*light*).

Las funciones son las siguientes:

- **int update(void):** Realiza la actualización de las variables, pide el estado actual de todas las variables de la campana a la pasarela y las actualiza en nuestro objeto. Como variables están tanto las propias de la campana como las generales de los electrodomésticos. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **int setlight(int input):** Indica la intensidad de luz de la campana mediante un valor entre 0 y 15, a intervalos de 1. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **int setfan(int input):** Indica la intensidad del extractor con valores entre 0 y 4 con intervalos de 1. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **int stop(void):** Apaga la luz de la campana y detiene el extractor simultáneamente. Esta función nos devolverá '1' en el caso de una transmisión correcta y '0' en el caso contrario.

- **float readPower(int min):** Esta función nos entrega la potencia instantánea de consumo en un minuto determinado. Dicha información proviene de la base de datos generada a partir de las medidas realizadas (ver Capítulo 4). Es usada para realizar las predicciones de consumo (ver Capítulo 5).

3.1.4. Datos de consumo

Actualmente la obtención de datos de consumo se realiza leyendo los archivos donde se encuentran las medidas realizadas para el calibrado y modelado de los electrodomésticos.

El formato de estos archivos es de filas y columnas en ficheros de texto, donde las columnas están separadas por espacios y las filas por caracteres de fin de línea. Las filas representan el eje de tiempos con una separación de muestreo de 10 segundos. Para un mayor detalle sobre la obtención de estas medidas, referenciamos al lector al Capítulo 4. Las columnas contienen información eléctrica del electrodoméstico, tal y como la tensión, corriente y potencia consumida en un momento determinado del trabajo. El valor que nos interesa es la potencia consumida. Dicha potencia la leemos mediante la función `float readPower (int min)`, a la que podemos preguntar el valor de la potencia en un minuto concreto. La función nos devolverá su valor en un minuto real.

3.2. Comunicación con el sistema fotovoltaico

La información del sistema fotovoltaico la obtenemos a partir de unos archivos, que contienen la generación del día anterior, a la espera de que las previsiones estén finalizadas.

El fichero "solarinfo.h" contiene la clase "CSolar", con funciones necesarias para la obtención tanto de la lectura de datos de los archivos de los días anteriores como los datos en tiempo real que necesitamos para la supervisión.

3.3. Comunicación con el contador

El contador nos permitirá controlar en tiempo real la potencia consumida por toda la instalación, el fichero "counter.h" contiene la clase "CCounter" con las funciones necesarias para la obtención tanto de los datos de consumo de nuestra instalación como información de tarificación y control de la red eléctrica.

3.4. Comunicación con las baterías

La información necesaria de las baterías será la carga disponible en el momento de realizar las previsiones, para posteriormente prever su dinámica en el transcurso del día. Esta carga inicial la leeremos directamente de los inversores que controlan las baterías.

El fichero "battery.h" contiene la clase "CBattery", donde se encuentran las funciones necesarias para la lectura de datos de la batería. Las características de funcionamiento de las batería se muestra en la Tabla 3.8.

Potencia máxima entregable	5 kW
Potencia máxima inyectable	5 kW
Carga máxima	90 kWh
Carga mínima recomendable	50 % (45 kWh)

Tabla 3.8: Características de funcionamiento de la batería.

3.5. Resumen

En este capítulo hemos desarrollado las librerías y las clases utilizadas en nuestro programa para la comunicación con el mundo, tanto para obtener datos como para mandar órdenes a los electrodomésticos.

Se ha explicado detalladamente la comunicación con las cargas, ya que es la que más se ha desarrollado en este Proyecto Fin de Carrera. Las librerías se han dividido en dos grupos: el primero encargado de la comunicación con el servidor, con funciones de más bajo nivel y posibles desbordamientos de datos y el segundo que se encarga de simplificar esta comunicación, protege de posibles errores a la hora de introducir o leer datos y facilita la tarea al usuario.

Capítulo 4

Caracterización de las cargas

Las cargas controlables son los electrodomésticos de Siemens mencionados en el Capítulo 3. Los fabricantes de electrodomésticos suelen entregar muy poca información sobre el consumo de estos, más bien entregan información relativa a la seguridad de la red como pueden ser corrientes y tensiones aceptadas y máximas, y algún indicador de consumo como la potencia máxima o el tipo de electrodoméstico (A+, A, B, etc.).

Para poder desarrollar un programa que planifique y prevea su consumo en los diferentes modos o situaciones de trabajo es necesario obtener las curvas de consumo para los diferentes casos. Por ello, se ha realizado una caracterización del consumo de las cargas realizando medidas con un vatímetro. De todos los electrodomésticos se han tomado muestras de los consumos cada 10 segundos durante toda la duración del programa en diferentes situaciones.

4.1. Herramientas utilizadas

4.1.1. Sistema de conexión

Al tener que medir la tensión y la corriente con el vatímetro, hemos realizado un circuito eléctrico que nos permita trabajar directamente con el cable de alimentación de los electrodomésticos de forma que tengamos una parte del cableado abierto para insertar la sonda de corriente y otra toma en paralelo para insertar la sonda de tensión

(ver Figura 4.1).

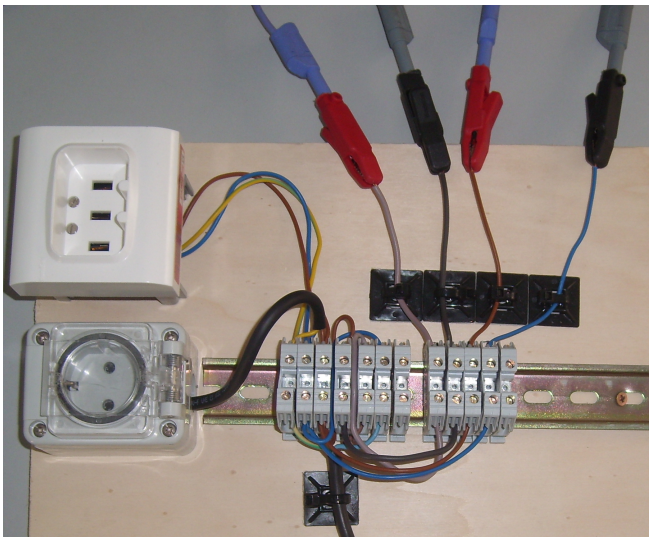


Figura 4.1: Cableado de medida.

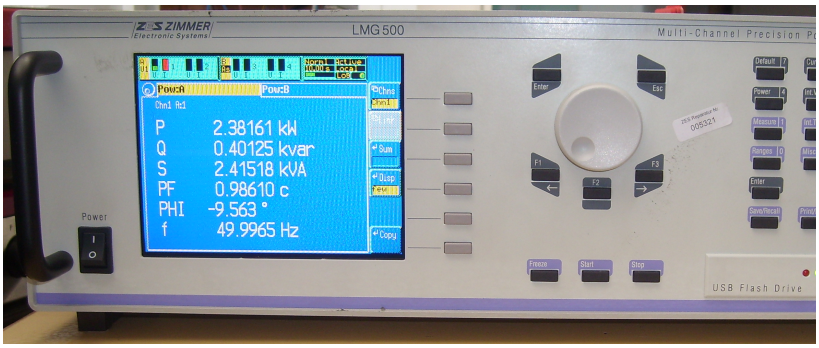
4.1.2. Vatímetro

Las medidas se han realizado con el vatímetro LMG500 de ZES Zimmer Electronic Systems, este puede verse en la Figura 4.2.

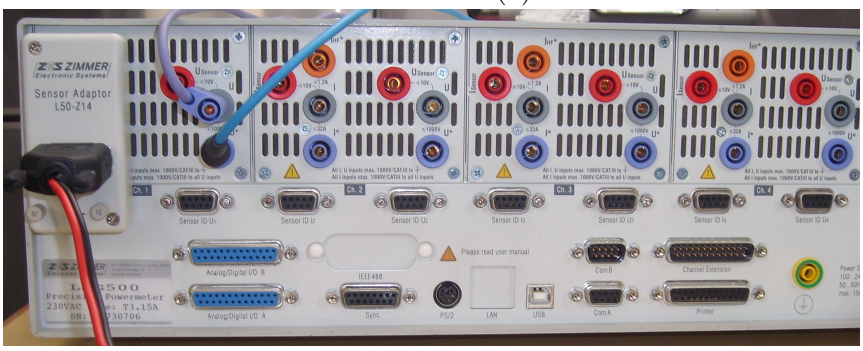
El vatímetro dispone de 8 canales con un rango de frecuencia de 0 a 10 Mhz. El rango de tensiones es de 3 V a 1000 V y el de corriente de 20 mA a 32 A con una precisión del 0.03 %. Esto nos permite medir tanto cada uno de los electrodomésticos independientemente como el consumo total de la vivienda.

Es posible monitorizar y almacenar toda la información tanto de la señal principal como de los armónicos. La información del vatímetro se puede obtener bien incorporando un dispositivo de almacenamiento USB, donde se guardan las medidas cada intervalo de muestras, o bien mediante la comunicación con un portátil.

Otro detalle importante es la capacidad del vatímetro para medir potencia reactiva, por lo que podremos observar el efecto que los diferentes sistema provocan en la calidad de la señal de la red.



(a)



(b)

Figura 4.2: Imagen del vatímetro (a) frontal y (b) posterior.

4.2. Medidas de consumo

Se han realizado medidas con cada uno de los electrodomésticos. Estas han sido introducidas posteriormente en el programa de control para el cálculo de las predicciones.

La duración de las medidas depende de cada electrodoméstico, ya que sus comportamientos son muy diferentes. A continuación se describe como se han realizado las medidas en cada uno de ellos.

4.2.1. Frigorífico/congelador

El consumo de un frigorífico y/o congelador depende principalmente de su compresor, el cual actuará para transferir el calor que se encuentra dentro de la cámara al exterior. El momento en que tienen que trabajar los compresores lo marcará un termostato, el cual, intentará mantener la temperatura dentro de la cámara lo más próxima a la que se haya indicado.

Ya que solamente podemos controlar la temperatura que queremos mantener y no cuando van a trabajar los compresores, la medida de este electrodoméstico tiene un carácter informativo. Sin embargo, su consumo instantáneo no puede ser previsto por el sistema de control.

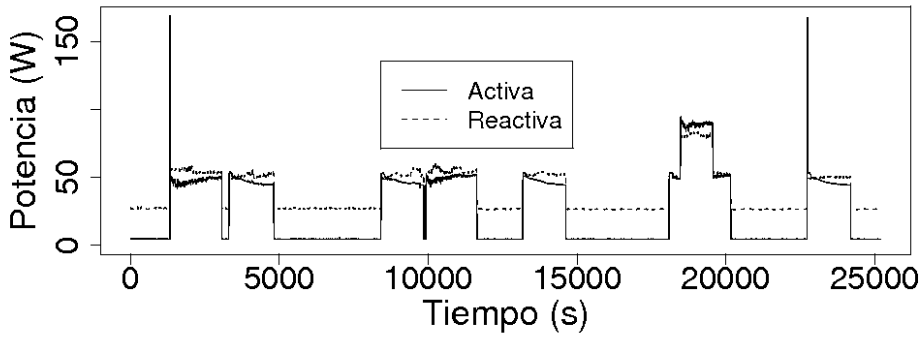


Figura 4.3: Potencia instantánea frigorífico/congelador. Temperatura de frigorífico: 5ºC, tempetatura de congelador: -22ºC.

Se han realizado diferentes medidas durante 7 horas para distintos valores de temperatura del congelador y el frigorífico. La Figura 4.3 muestra el consumo del mismo para una temperatura de frigorífico de 5ºC y una de congelador de -22ºC. Como puede observarse en la Figura 4.3 los dos compresores son independientes ya que cada uno trabaja cuando es necesario bajar la temperatura de la cámara. Observamos que alrededor del segundo 18.000, ambos compresores entran en funcionamiento durante el mismo intervalo de tiempo, por lo que podremos calcular la potencia instantánea

máxima. Igualmente observamos que la potencia reactiva es bastante alta en relación con la activa, lo que implica que nos introduce distorsión de fase en la red eléctrica.

4.2.2. Lavadora

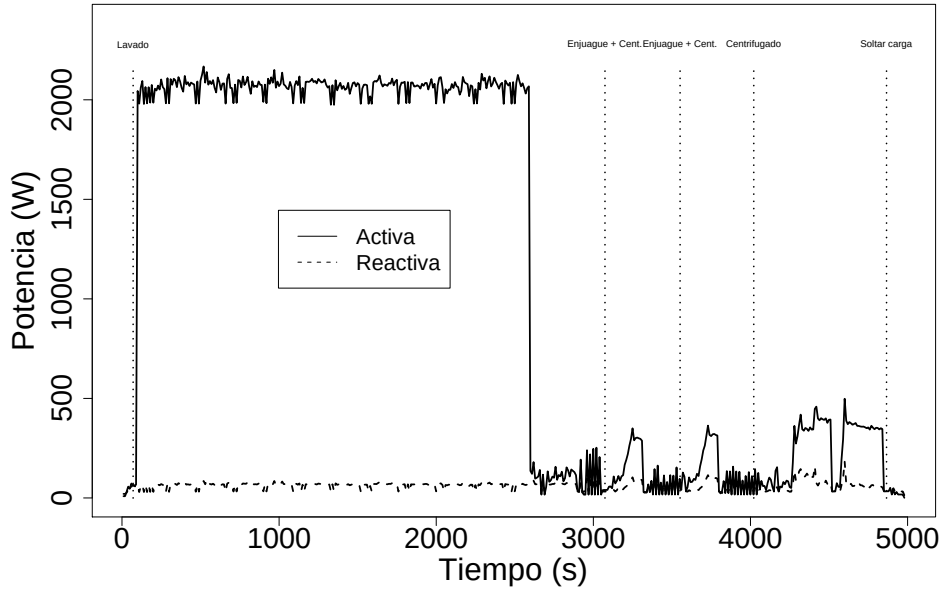
El consumo de la lavadora depende de la temperatura, las revoluciones y la carga (cantidad de ropa). Se ha observado que la mayor parte del consumo depende principalmente de la temperatura de lavado.

Un factor a tener en cuenta en las previsiones de consumo y en el control automático de la lavadora es que el programa total se puede dividir en 3 fases independiente tal y como se explicó en el Capítulo 2.

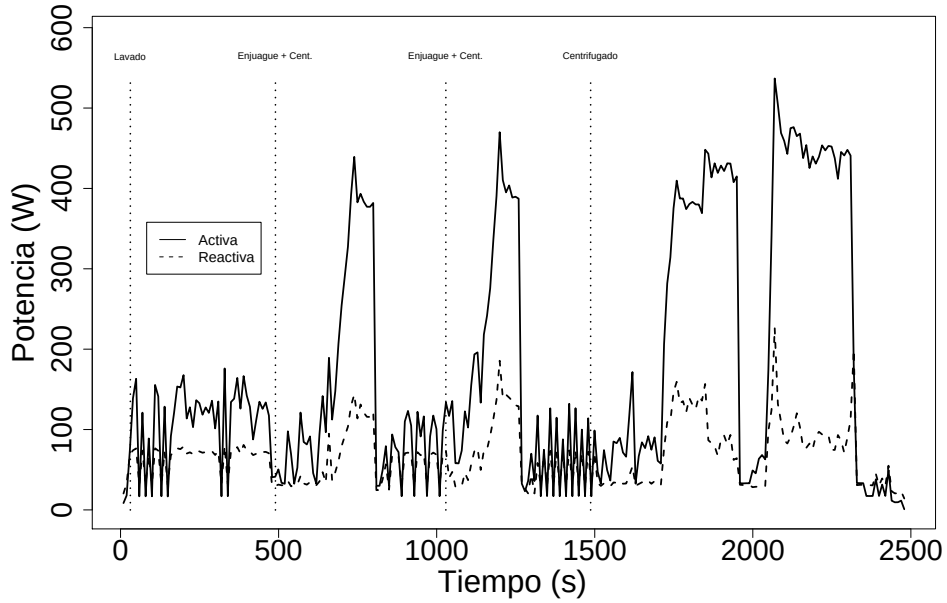
Considerando una carga media y unas revoluciones fijas de 1200 rpm se ha variado la temperatura del agua para ver cómo afecta al consumo. En la Figura 4.4 mostramos el consumo en potencia instantánea para un programa a 90ºC y para un lavado en frío.

Como podemos observar, la duración de la primera fase de la lavadora depende directamente de la temperatura a la que se realiza el lavado sin modificarse la potencia instantánea. Por lo tanto, el consumo energético será lineal con el tiempo. Las otras dos fases no se verán modificadas por la temperatura, por lo que la energía consumida en estas tampoco.

Como puede verse en la Figura 4.5, el mayor consumo de energía se produce en la fase de lavado, con el calentamiento del agua. Por lo tanto, como mencionamos anteriormente la temperatura será el principal parámetro que condicione el consumo de la lavadora.

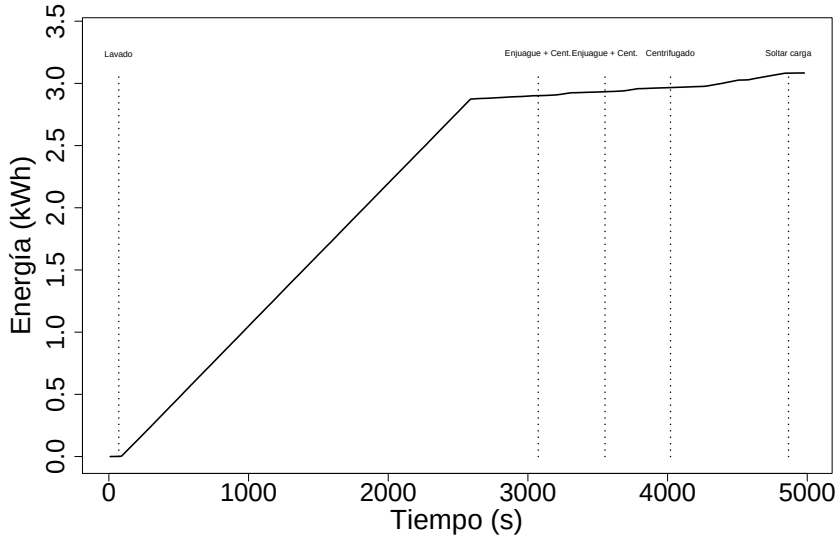


(a)

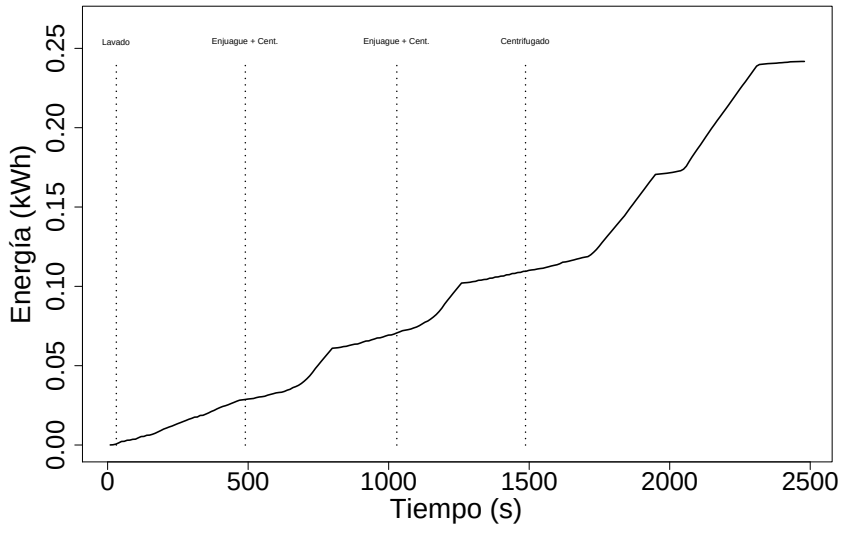


(b)

Figura 4.4: Potencia instantánea de la lavadora: (a) temperatura de lavado: 90°C, revoluciones: 1200 rpm y (b) temperatura de lavado: frío, revoluciones: 1200 rpm.



(a)



(b)

Figura 4.5: Consumo energético de la lavadora: (a) temperatura de lavado: 90°C, revoluciones: 1200 rpm y (b) temperatura de lavado: frío, revoluciones: 1200 rpm.

Las Figuras 4.4b y 4.6 muestran un lavado en frío a media carga pero a diferentes revoluciones, 1200 rpm y 1600 rpm respectivamente. Podemos observar que existe una primera fase muy parecidas, así como en los centrifugados intermedios tampoco observamos prácticamente ninguna diferencia. Sin embargo en la última fase observamos diferencias en la potencia instantánea. También existe un pequeño incremento en la duración del centrifugado final.

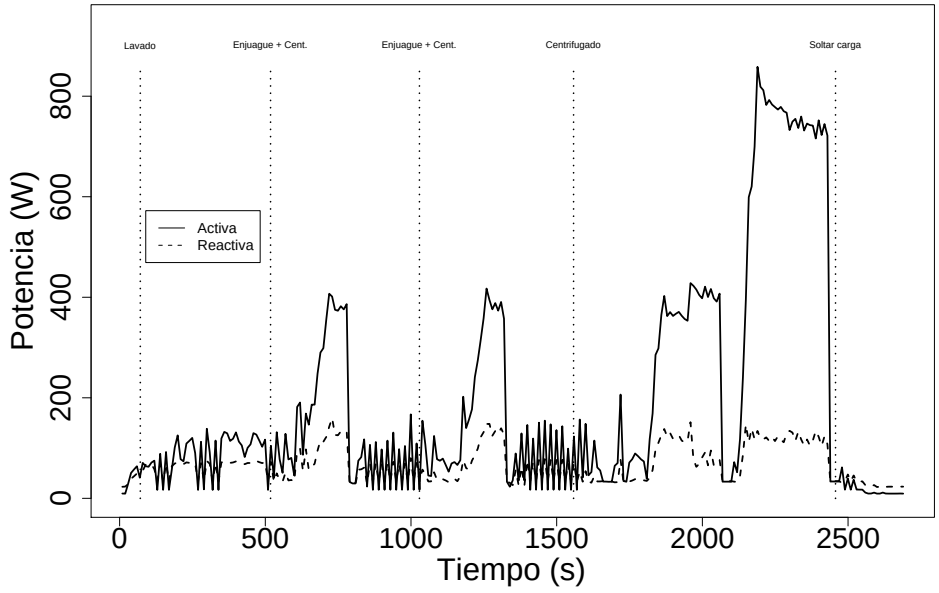
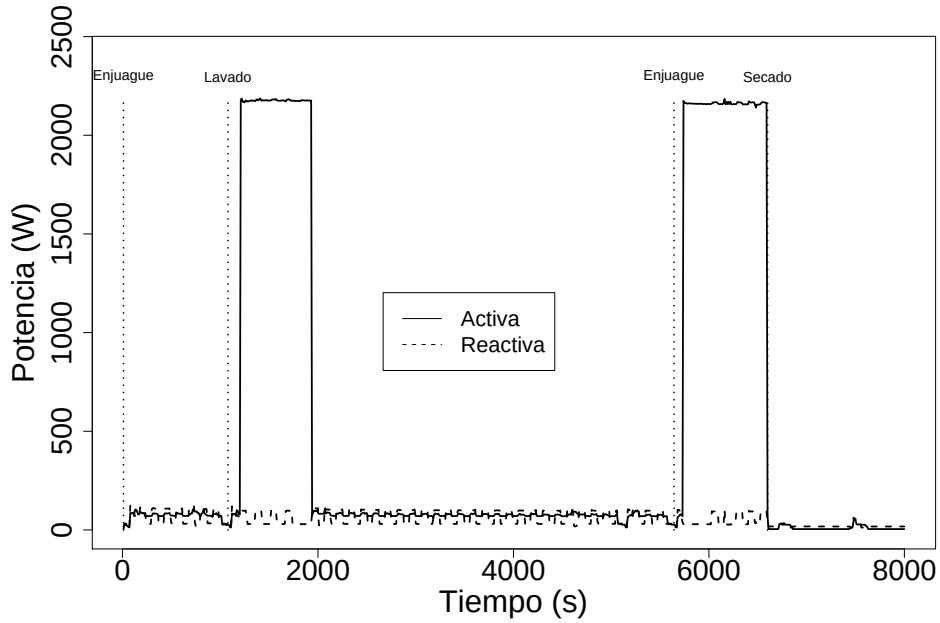
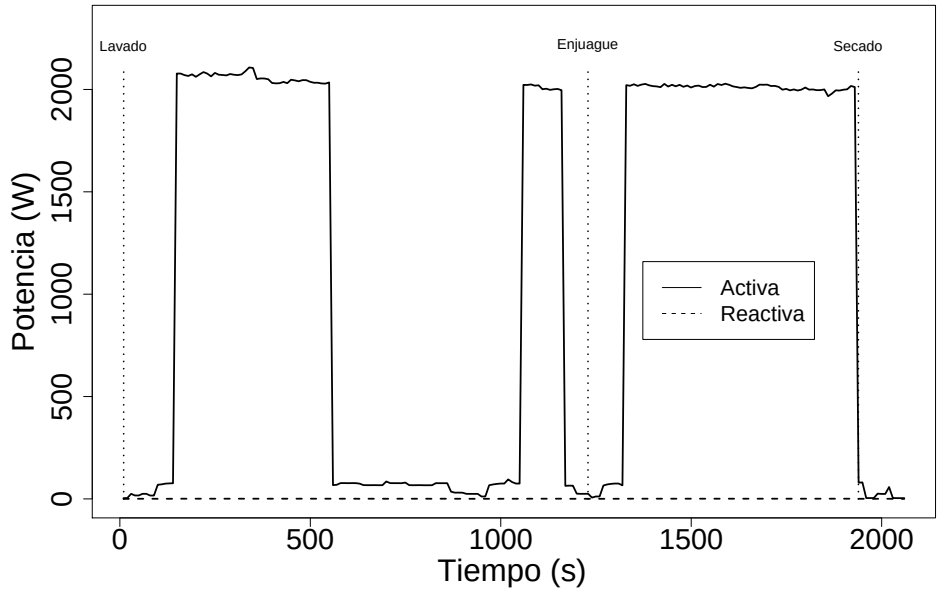


Figura 4.6: Potencia instantánea de la lavadora, temperatura de lavado: frío, revoluciones: 1600 rpm.



(a)



(b)

Figura 4.7: Potencia instantánea del lavavajillas: (a) lavado normal y (b) lavado rápido.

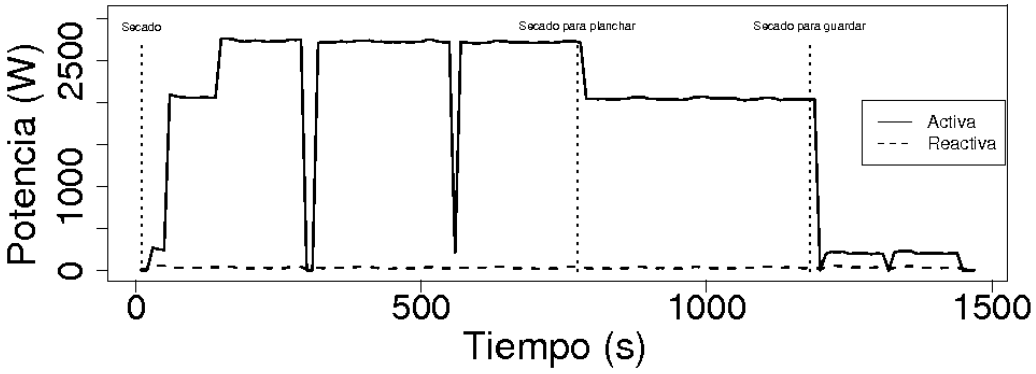
4.2.3. Lavavajillas

Del lavavajillas solamente podemos seleccionar el programa de lavado, por lo que las diferentes curvas de consumo dependerán exclusivamente de este parámetro. En la figura 4.7a se muestra un lavado normal, mientras que la Figura 4.7b corresponde a un lavado rápido.

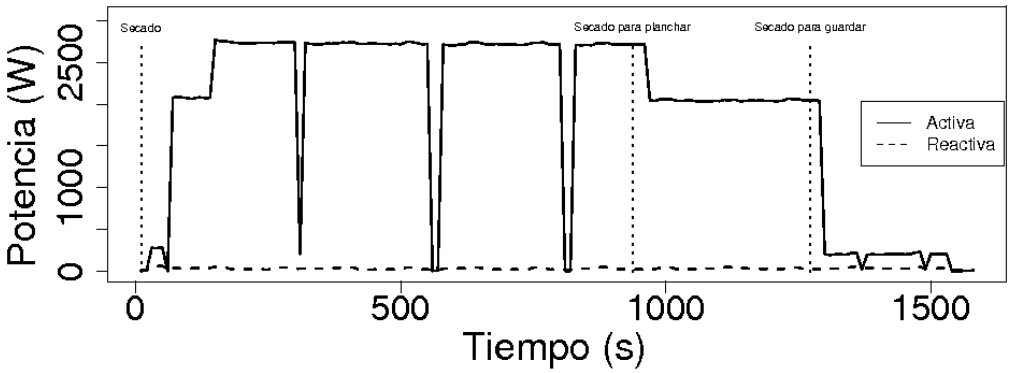
4.2.4. Secadora

El control remoto de la secadora nos permite cambiar las revoluciones de trabajo aunque el programa tiene que ser indicado de forma manual. Por lo tanto, todas las medidas se han realizado con el programa rápido, en el cual variamos las revoluciones de secado. En las figuras 4.8a y 4.8b podemos ver la potencia consumida por la secadora a las revoluciones de 800 rpm y 1600 rpm respectivamente.

Podemos observar como el consumo de la secadora y la duración del programa no varía linealmente en función de las revoluciones. En las figuras 4.9a y 4.9b se muestra la energía consumida por la secadora trabajando a 800 rpm y 1600 rpm respectivamente. Como se puede observar tendremos un mayor consumo a 1600 rpm.

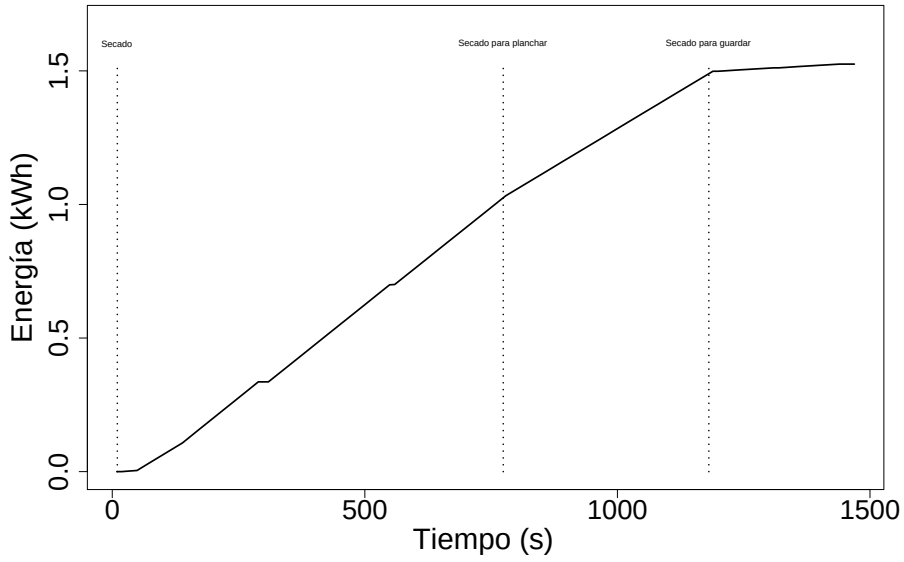


(a)

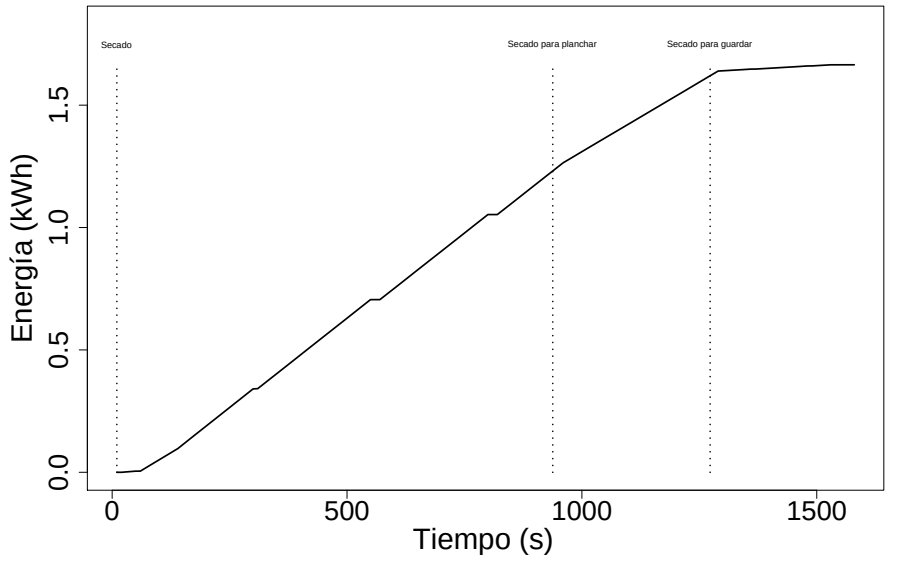


(b)

Figura 4.8: Potencia instantánea de la secadora: (a) revoluciones: 800 rpm y (b) revoluciones: 1600 rpm.



(a)



(b)

Figura 4.9: Consumo energético de la secadora: (a) revoluciones: 800 rpm y (b) revoluciones: 1600 rpm.

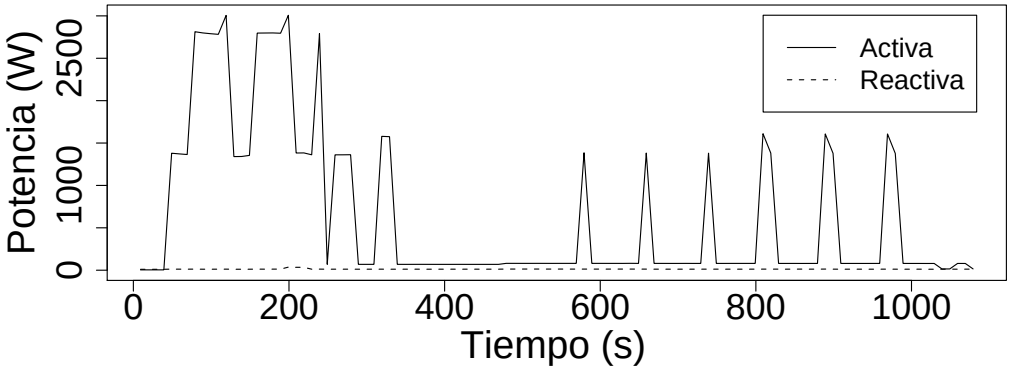
4.2.5. Horno

El consumo del horno se ha medido en condiciones de trabajo normales, es decir un horneado normal (sin gril ni ventilador) y con la temperatura ambiente como temperatura de inicio, es decir, sin precalentar.

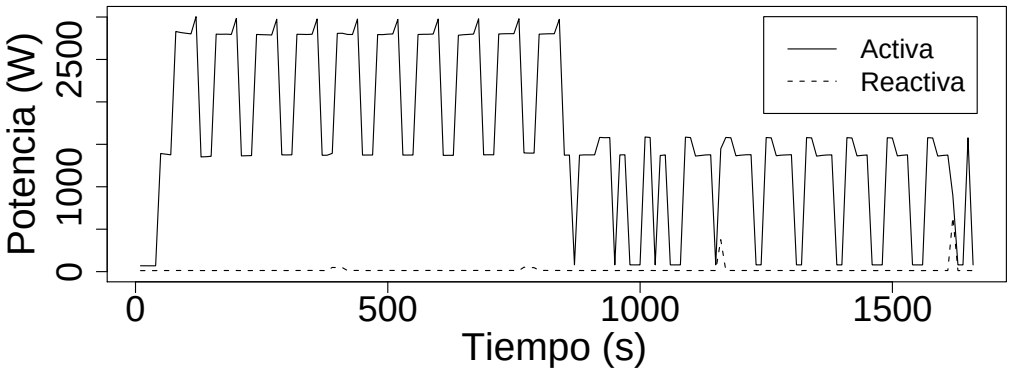
Se distinguen básicamente dos fases: una de calentamiento, en la que se intenta conseguir la temperatura deseada y otra de mantenimiento en la que se mantiene esta temperatura.

El consumo del horno dependerá básicamente de la temperatura a la que queremos hornear. La Figura 4.10a representa un consumo para una temperatura de 100ºC y la Figura 4.10b representa un consumo para una temperatura de 300ºC.

Se puede observar como en la fase de calentamiento se alcanza la máxima potencia del horno (unos 2500 W), donde su funcionamiento tiene una cierta periodicidad. En la fase de mantenimiento, el funcionamiento depende mucho de la temperatura deseada y de la temperatura exterior.



(a)



(b)

Figura 4.10: Potencia instantánea del horno: (a) temperatura: 100ºC y (b) temperatura: 300ºC.

4.2.6. Campana

El consumo de la campana depende de dos elementos: el extractor de la campana del cual se puede controlar su potencia en 4 niveles y la luz de la campana dividida en 15 niveles de intensidad.

Se ha medido el consumo independientemente de cada uno de los elementos ya que el funcionamiento de uno no se ve modificado por el del otro. En la Figura 4.11 puede verse el consumo del extractor de la campana en los 4 niveles de intensidad posibles, primero nos encontramos con el consumo que produce un cambio escalonado en los diferentes niveles del extractor y posteriormente el consumo de cada nivel de potencia independientemente.

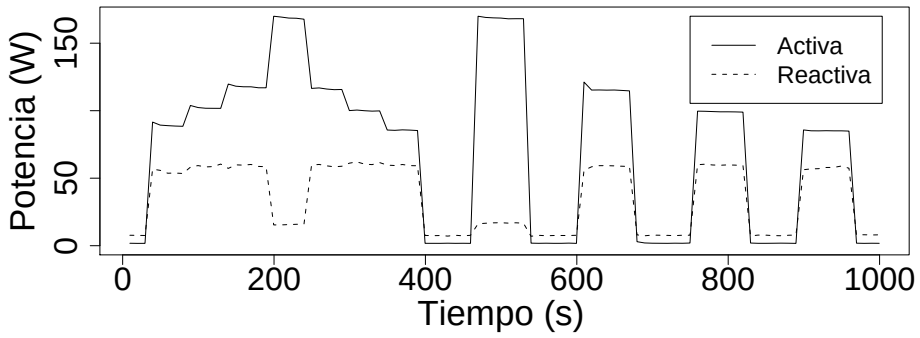


Figura 4.11: Potencia instantánea del extractor. Muestra de todos los niveles de potencia.

En la Figura 4.12 vemos el consumo de la luz cuando se enciende y apaga a 3 niveles diferentes de intensidad que serían 5, 10 y 15. Observamos que el consumo no es lineal con la intensidad de la luz.

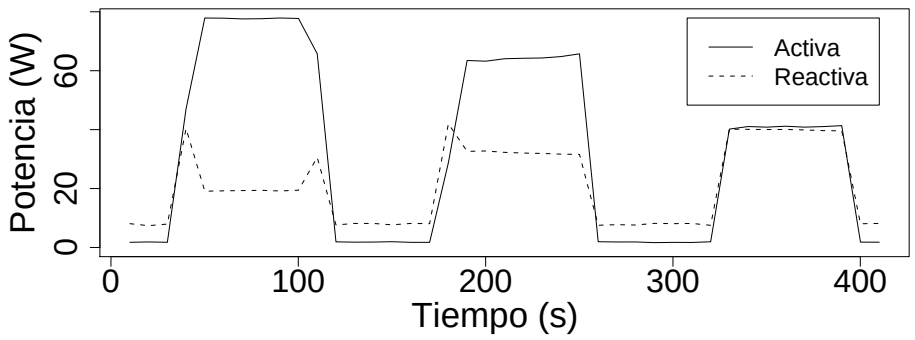


Figura 4.12: Potencia instantánea de la lampara de la campana. Intensidad de luz: 5, 10 y 15.

4.3. Resumen

En este capítulo hemos caracterizado el consumo producido por los diferentes electrodomésticos (las cargas de nuestro sistema). Con esta información, se han deducido algunas características del funcionamiento de los electrodomésticos que nos ayudarán a la hora de trabajar con ellos e implementar los algoritmos de control.

Los valores, a partir de los cuales se han generado las diferentes gráficas, son utilizados por nuestro programa para prever el consumo de las diferentes cargas en una planificación global y con ello poder estimar como afectarán energéticamente a la vivienda (Ver Capítulo 2).

Capítulo 5

Aplicación práctica

En el presente capítulo implementamos una situación real de funcionamiento en la vivienda.

En la Sección 5.1, planteamos una situación con una serie de tareas concretas impuestas por el usuario. Se mostrarán los resultados del sistema de control, tanto para las planificaciones desarrolladas como para el balance energético calculado por éste. Posteriormente observamos el consumo real medido en dicha situación. En la Sección 5.2 se comparan los resultados de los balances energéticos de las diferentes planificaciones.

Finalmente, en la Sección 5.3 se compara un estudio de la eficiencia energética conseguida por el sistema de control con respecto a situaciones no controladas.

5.1. Medidas del comportamiento

En esta sección se muestran los resultados del funcionamiento del sistema de control, tanto en la realización de planificaciones como en el balance de potencia y energía de las mismas.

Para poder mostrar con detalle los resultados del funcionamiento del sistema de control planteamos una situación concreta. En la Tabla 5.1 se observan las diferentes tareas impuestas por el usuario, junto a sus límites temporales, de la situación a desarrollar.

Límite de tiempos lavadora	600 - 1440 (min)
Parámetros de lavado	Temperatura: 60^{o}C, Revoluciones: 1200 rpm
Límite de tiempos secadora	600 - 1440 (min)
Parámetros de secado	Secado rápido, Revoluciones: 1200 rpm
Límite de tiempos lavavajillas	600 - 1440 (min)
Parámetros de lavado	Lavado rápido

Tabla 5.1: Parámetros iniciales de las tareas.

En la Tabla 5.2 se observan los principales parámetros energéticos de nuestra prueba.

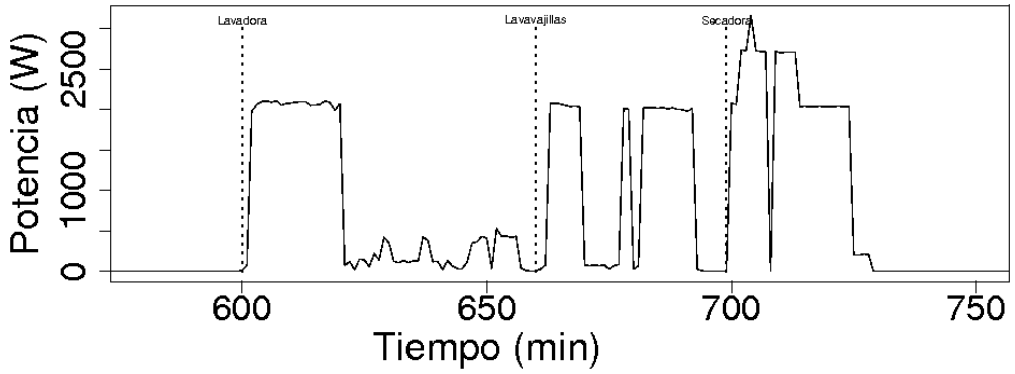
Energía fotovoltaica disponible	7.095 kWh
Potencia pico fotovoltaica	1122.0 W
Carga inicial de baterías	0.0 kWh (vacía)
Consumo total de las cargas	2.437 kWh

Tabla 5.2: Parámetros energéticos iniciales.

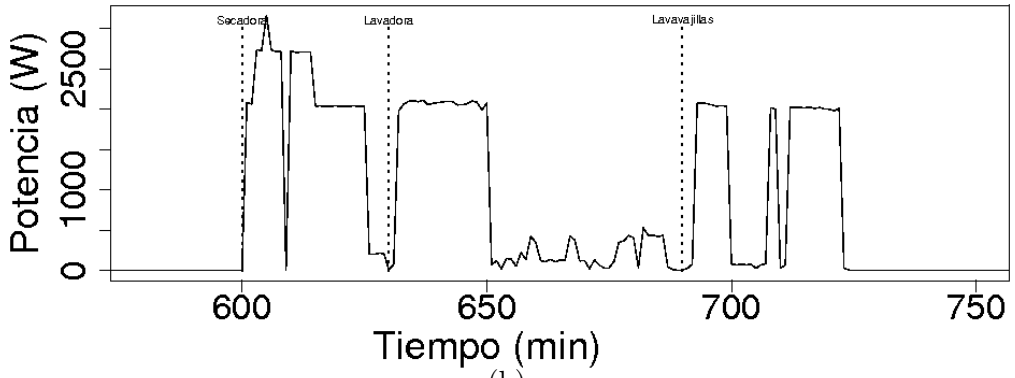
5.1.1. Resultado del sistema de control

Tras ser introducidas las diferentes tareas por el usuario, junto a sus límites temporales, la capa de planificación (ver Capítulo 2) realizará una planificación de todas las tareas por cada uno de los electrodomésticos.

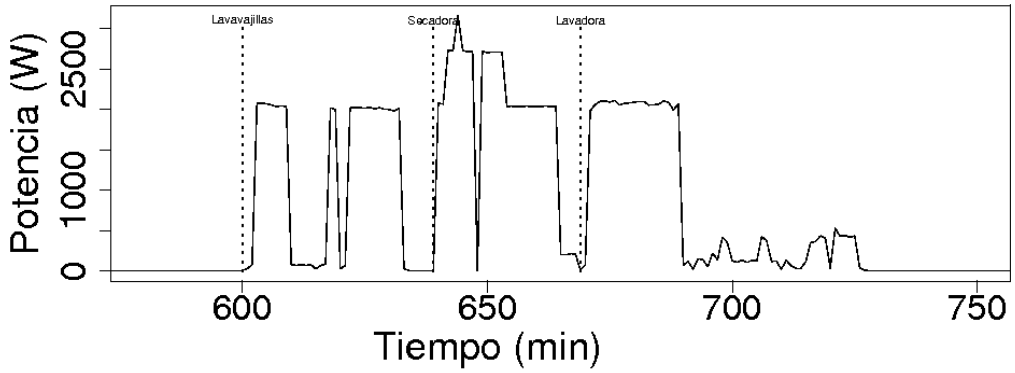
La capa de coordinación recibe las diferentes planificaciones y obtiene el consumo instantáneo de cada una de ellas. En la Figura 5.1 se observan las tres planificaciones realizadas junto a su consumo de potencia instantáneo. Una vez la capa de coordinación tiene la información del consumo de las tres planificaciones, pasa a combinarlas con la generación fotovoltaica como se muestra en la Figura 5.2.



(a)



(b)



(c)

Figura 5.1: Potencia consumida en la planificación de (a) la lavadora, (b) la secadora y (c) el lavavajillas.

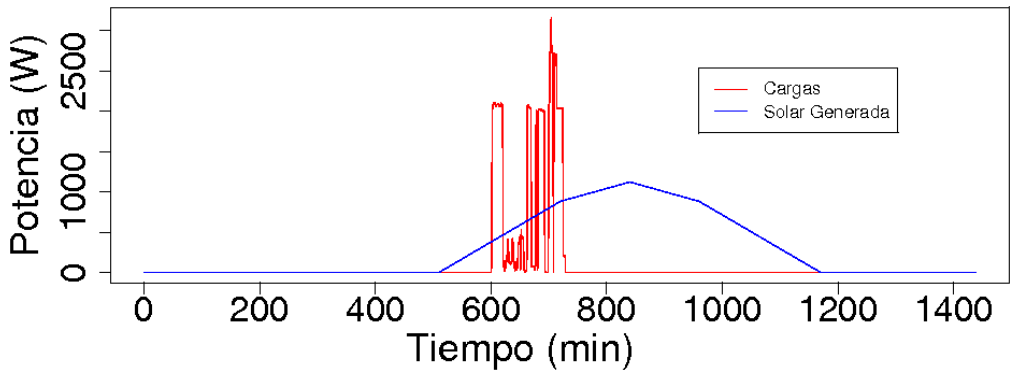


Figura 5.2: Potencia consumida por las cargas y generada por el sistema fotovoltaico para un día de trabajo (0-1440 min).

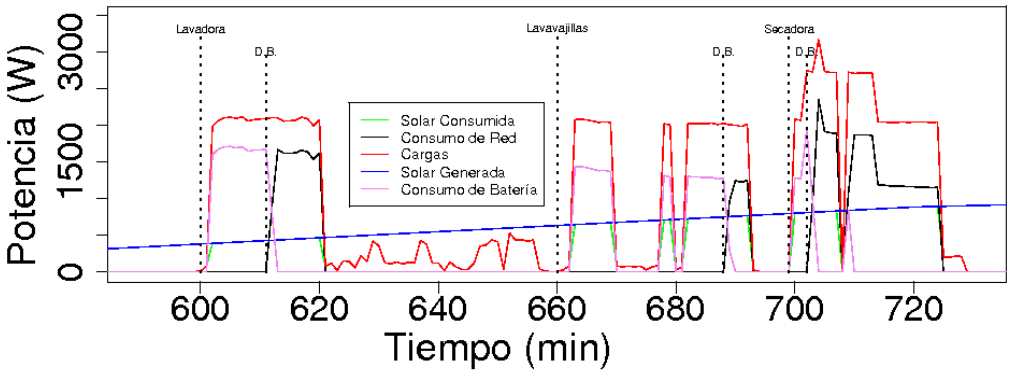
El coordinador pasa a calcular minuto a minuto el balance de potencia, mostrando desde donde se está proporcionando la potencia requerida por las cargas. El balance de potencia de la planificación realizada por la lavadora puede verse en la Figura 5.3a. El marcador D.B. (Descarga de Baterías) indica el momento en el cual no queda energía almacenada en las baterías, como veremos más adelante.

A la vez que se calcula el balance en potencia, se obtiene el consumo energético, integrando la potencia. Con ello se obtiene el balance energético de la planificación. Por ejemplo, el balance energético de la lavadora se muestra en la Figura 5.3b.

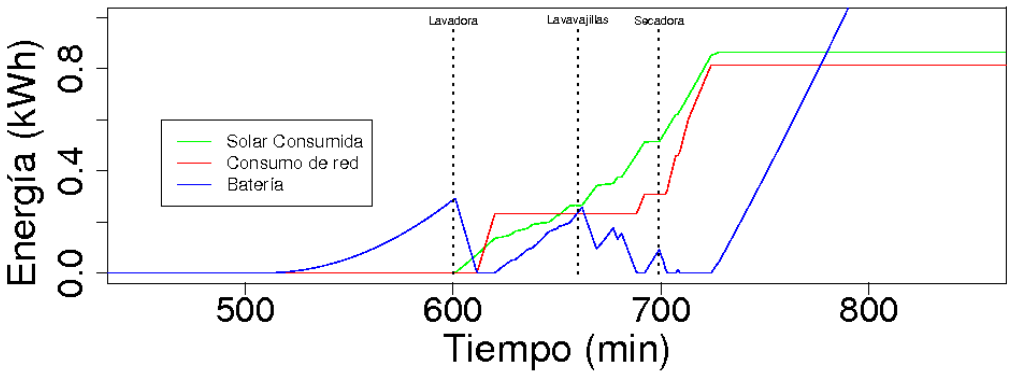
En el balance de potencia, vemos cómo siempre que tengamos potencia fotovoltaica ésta se destinará a cubrir el consumo de las cargas. Cuando el consumo sea mayor que la potencia fotovoltaica generada, la curva de consumo fotovoltaico corresponderá a la de generación solar y cuando el consumo sea menor corresponderá a la del consumo de las cargas. El excedente de energía fotovoltaica será almacenado en las baterías. En el balance de energía de la planificación de la lavadora, observamos que las baterías se cargan con el excedente energético (Figura 5.3b).

En el caso de que la potencia fotovoltaica no sea suficiente, se intentará obtener esta energía de baterías, por ello, hay situaciones el las que la potencia demandada no proviene ni del sistema fotovoltaico ni de la red, en cambio, podemos ver como

desciende la energía almacenada en baterías. En la situación en la que las baterías están descargadas y no tenemos energía fotovoltaica, tomaremos la energía de la red eléctrica (marcador D.B. en la Figura 5.3a). En este caso aumentará la energía consumida de la red, como puede verse en el balance de energía.



(a)

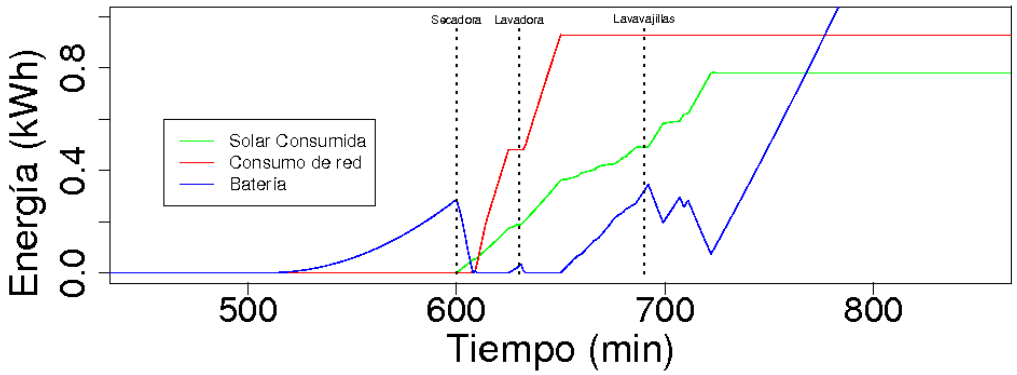


(b)

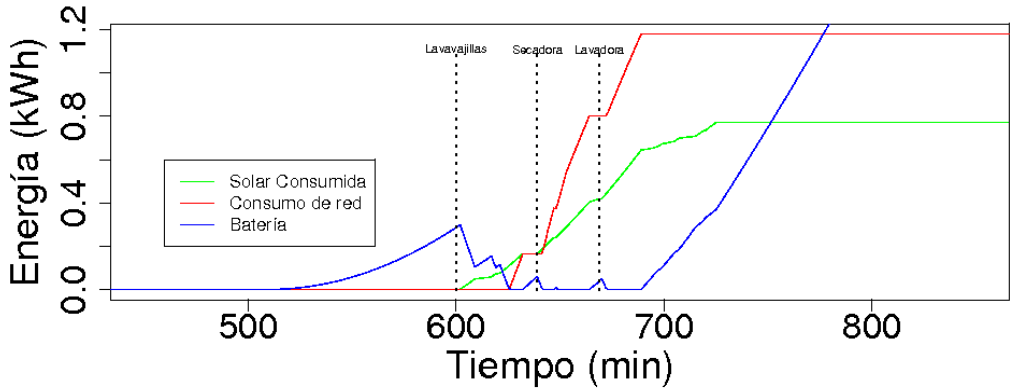
Figura 5.3: Planificación realizada por la lavadora: (a) balance de potencia (b) balance de energía.

La capa de coordinación realiza el balance de potencia y energía para las tres

planificaciones, repitiendo los cálculos anteriormente explicados para la lavadora en las planificaciones de la secadora y el lavavajillas. En las figuras 5.4a y 5.4b se observan los balances energéticos realizados sobre la planificación de la secadora y el lavavajillas respectivamente.



(a)



(b)

Figura 5.4: Balance de energía: (a) planificación secadora (b) planificación lavavajillas.

El sistema de control decide ejecutar la planificación de la lavadora, por ser la planificación con mayor aprovechamiento de energía solar fotovoltaica.

5.1.2. Pruebas reales

Las planificaciones se realizan para posteriormente ser llevadas a cabo por la capa de ejecución. En la Figura 5.5 podemos observar el consumo de la vivienda cuando se realiza la planificación realizada por la lavadora.

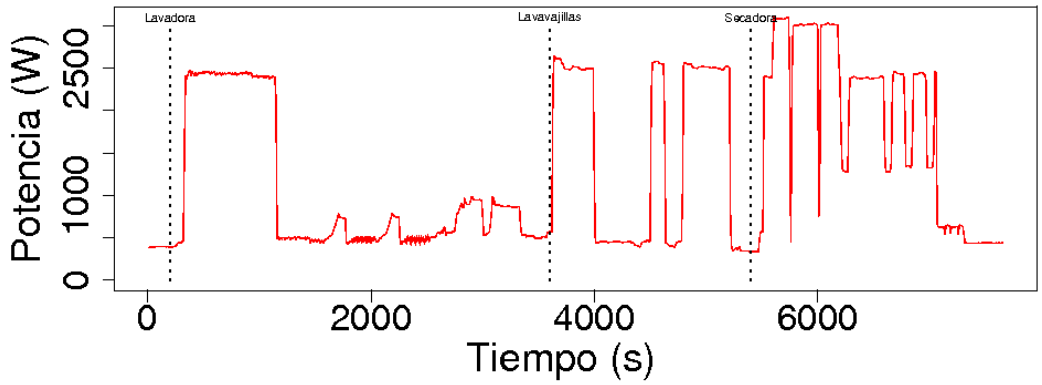


Figura 5.5: Curva de potencia instantánea consumida por la vivienda.

Puede verse el consumo de las cargas junto a un consumo constante que siempre está activo, correspondiente a equipos electrónicos, ordenadores, luces e incluso los compresores del frigorífico.

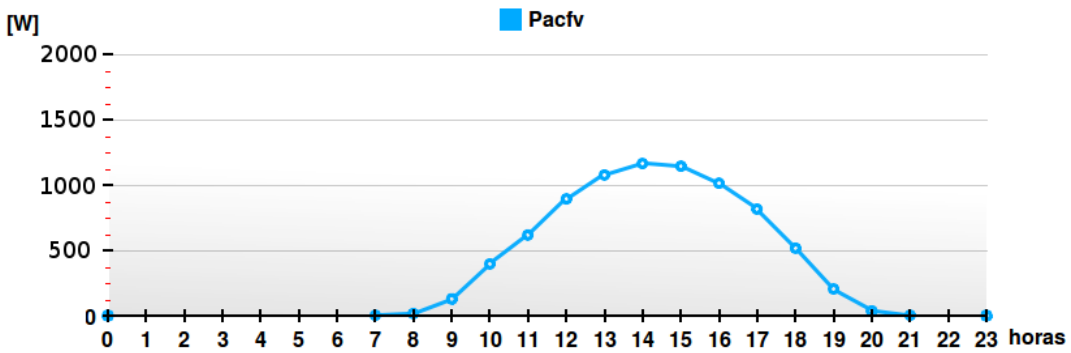


Figura 5.6: Curva de generación solar fotovoltaica.

Finalmente, en la Figura 5.6 se muestra la generación fotovoltaica real tomada como base para realizar la simulación.

5.2. Comparaciones de consumo

En esta sección se compararán los consumos de las diferentes planificaciones. Las comparaciones hacen referencia a los datos calculados por la capa de coordinación en la Sección 5.1.

En la Tabla 5.3 vemos los datos de energía finales de las tres planificaciones, corresponden al valor de la gráfica de energía en el minuto 1440 (24:00 h).

Energía	Lavadora	Secadora	Lavavajillas
Solar	0.865926 kWh	0.781763 kWh	0.773263 kWh
Red	0.81334 kWh	0.929429 kWh	1.18115 kWh
Carga de baterías	5.47133 kWh	5.58742 kWh	5.83914 kWh

Tabla 5.3: Consumo energético final.

En la Tabla 5.4 se observa el porcentaje de aumento de consumo de fotovoltaica y red respecto al caso de menor consumo. También se puede observar la carga final de baterías respecto a la situación de menos carga.

Energía	Lavadora	Secadora	Lavavajillas
Solar	11.98 %	1.09 %	0.0 %
Red	0.0 %	14.27 %	45.2 %
Carga de baterías	0.0 %	2.12 %	6.72 %

Tabla 5.4: Relación de consumos entre planificaciones.

En la Figura 5.7 mostramos un diagrama de los consumos de las diferentes planificaciones. En la planificación de la lavadora tendremos un mayor aprovechamiento de la energía fotovoltaica generada, reduciendo el consumo de red. En

cambio, la planificación del lavavajillas provoca un mayor consumo de la red con el consiguiente aumento del gasto eléctrico. Por ello, el sistema de control elige ejecutar la planificación de la lavadora.

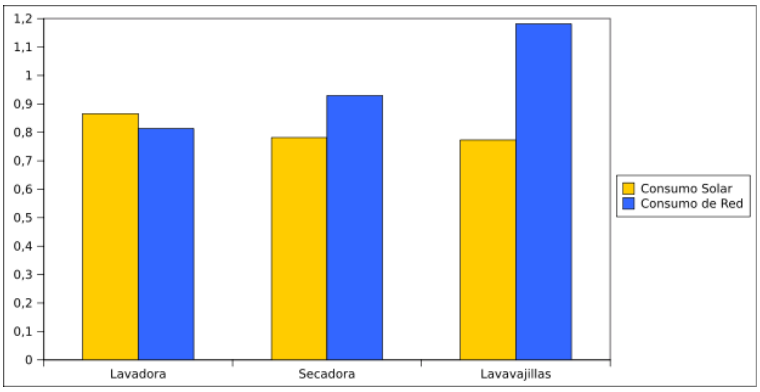


Figura 5.7: Comparación de consumo de energía solar y de red.

Otro factor a tener en cuenta es el estado final de las baterías. En la Figura 5.8 vemos el estado de carga de baterías al final de las tres planificaciones. En este caso, el lavavajillas es el que consigue un mayor nivel de carga. Esto se debe a que, al haber tomado menos energía de los paneles fotovoltaicos, esta energía es almacenada en las baterías. Téngase en cuenta que, para la elección de la planificación, no se considera prioritario el nivel de carga de las baterías.

5.3. Eficiencia energética

La eficiencia depende del parámetro que se desee optimizar. En nuestro caso nos interesa la eficiencia energética y hemos considerado la energía fotovoltaica consumida como referente.

Como se ve en la Tabla 5.2, la energía total generada por el sistema fotovoltaico es de 7.095 kWh y la energía total demandada por las cargas es de 2.437 kWh. En la Tabla 5.5, se muestran los porcentajes de la energía cubierta por la generación solar y la red eléctrica del total de energía demandada. Además puede observarse el

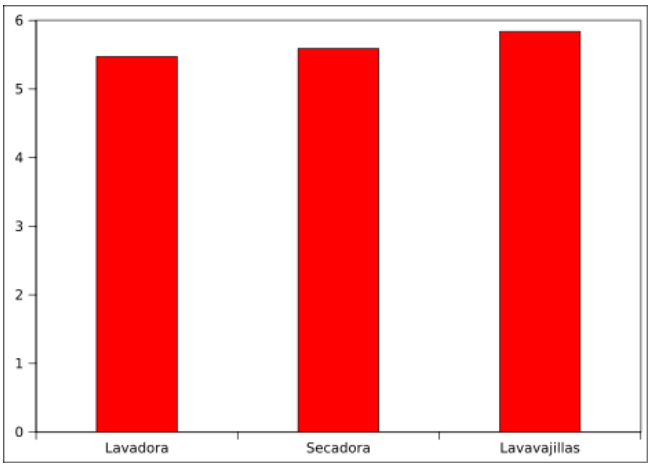


Figura 5.8: Comparación de carga de baterías.

porcentaje de energía fotovoltaica almacenada en baterías respecto al total generado.

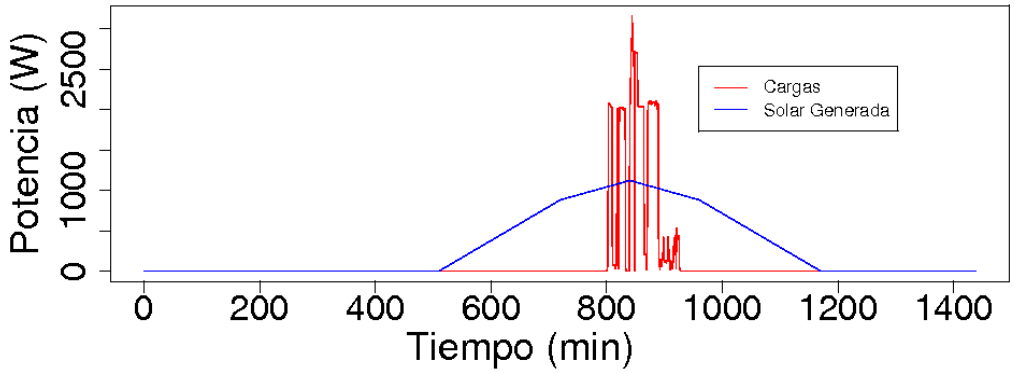
Energía	Lavadora	Secadora	lavavajillas
Solar	35.5 %	32.0 %	31.07 %
Red	33.37 %	38.13 %	48.46 %
Carga de baterías	77.11 %	78.75 %	82.29 %

Tabla 5.5: Porcentajes de consumo energético.

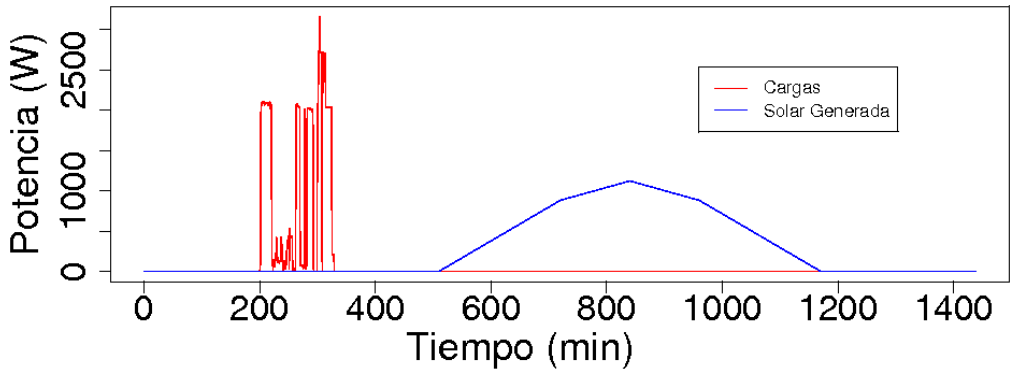
El sistema de control elige la mejor de las planificaciones, cubriendo el 35.5 % de la demanda energética con energía fotovoltaica. En el caso peor se cubre el 31.07 % de la demanda energética con energía fotovoltaica. La eficiencia del sistema de control en esta situación es del 4.43 %, que sería la diferencia entre el caso seleccionado y el caso peor.

Con el ejemplo planteado, hemos desarrollado una situación en la que el consumo se produce en las primeras horas de la mañana, donde la generación fotovoltaica no ha llegado a su máximo (ver Figura 5.2). Con el objetivo de mostrar la eficiencia energética que se puede alcanzar en diferentes momentos del día planteamos dos nuevas situaciones. En la Figura 5.9a se observa una situación donde el consumo se

planifica para las horas de máxima generación fotovoltaica (caso mejor). En la Figura 5.9b se observa una situación donde el consumo se planifica para las horas nocturnas (caso peor). En ambas figuras la curva de consumo de las cargas corresponde a la elegida por el sistema de control en esas situaciones.



(a)



(b)

Figura 5.9: Potencia consumida por las cargas y generada por el sistema fotovoltaico en 24h: (a) consumo al mediodía (b) consumo nocturno.

El balance energético final de las tres situaciones se presenta en la Tabla 5.6. En

la Tabla 5.7 se muestran los porcentajes de la energía cubierta por la generación solar y la red eléctrica del total de energía demandada. Además puede verse el porcentaje de energía fotovoltaica almacenada en baterías respecto al total generado.

Energía	Caso peor	Situación intermedia	Caso mejor
Solar	0.0 kWh	0.865926 kWh	1.2849 kWh
Red	2.437 kWh	0.81334 kWh	0.0 kWh
Carga de baterías	7.095 kWh	5.47133 kWh	4.65799 kWh

Tabla 5.6: Balance energético para diferentes momentos del día.

Energía	Caso peor	Situación intermedia	Caso mejor
Solar	0.0 %	35.5 %	52.72 %
Red	100 %	33.37 %	0.0 %
Carga de baterías	100 %	77.11 %	65.65 %

Tabla 5.7: Porcentajes de consumo energético.

En el caso mejor se puede llegar a cubrir directamente el 52.72 % de la energía demandada. Con la ayuda de las baterías se consigue eliminar el consumo de red por lo que permite una situación de plena autonomía.

Aunque las situaciones son impuestas por el usuario, el sistema de control informa del balance energético de cada situación al mismo. Por lo tanto, es este el encargado de decidir el momento de trabajo de las diferentes tareas, seleccionando los intervalos de trabajo, y sobre el que recae la última palabra para la gestión de la demanda.

5.4. Resumen

En este capítulo hemos mostrado los datos con los que trabaja la capa de coordinación y el interés que tienen a la hora de prever consumos futuros.

A la hora de comparar las diferentes planificaciones realizadas por los tres subsistemas (lavadora, secadora y lavavajillas), se ha comprobado que el hecho de cambiar el orden de éstas, provoca una mejora en la eficiencia energética de la vivienda. Comparando situaciones con las cargas distribuidas en diferentes momentos del día, el balance energético es totalmente diferente, incluso pudiéndose llegar a conseguirse situaciones de autonomía con respecto a la red eléctrica.

Capítulo 6

Conclusiones y líneas futuras

6.1. Conclusiones

En este Proyecto Fin de Carrera se ha desarrollado un sistema de control para la Gestión de la Demanda Eléctrica con Generación Fotovoltaica en el sector residencial. Cada subsistema ejecuta un algoritmo de planificación, que permite que cada electrodoméstico obtenga información de las tareas a realizar y las planifique en un eje temporal. Un coordinador a la salida del sistema distribuido, tras realizar un balance en potencia y energía, selecciona del conjunto de las planificaciones, la más eficiente energéticamente.

Centrándonos en los objetivos planteados al inicio del proyecto, podemos concluir que se han alcanzado todos y cada uno de ellos:

- Control del sistema domótico: Se han desarrollado las librerías necesarias para el control de los electrodomésticos. El mayor problema encontrado ha sido que las APIs de la pasarela no estaban bien definidas en el manual de uso, debido a que los equipos son experimentales, y hubo que recurrir a la ingeniería inversa para conseguir comunicar algunas órdenes a los electrodomésticos. Se han desarrollado las librerías necesarias para poder obtener el estado y enviar órdenes a los electrodomésticos. Para el desarrollo de las librerías se ha hecho uso de la herramienta *gSoap* (ver Apéndice A). Comentar que dado el carácter experimental de los electrodomésticos, no todas las funciones necesarias han

sido implementadas, por no existir en el firmware de los mismos.

- Caracterización de las cargas: La tarea de caracterización de las cargas nos ha permitido conocer el consumo de los electrodoméstico, para ser insertado en una base de datos y ser usados por el sistema de control. Para la caracterización no sólo se ha tenido en cuenta el consumo eléctrico, sino también el comportamiento de los electrodomésticos a nivel funcional y el tiempo que necesitaban para realizar sus diferentes fases.

- Obtención de datos externos: El sistema fotovoltaico nos envía información de días anteriores pero aún estamos a la espera de recibir las previsiones de generación de días futuros a partir de previsiones meteorológicas. Respecto a las baterías, obtenemos el valor de la carga de las mismas en el momento de comenzar la previsión. Sin embargo, sobre la información de la Red Eléctrica estamos a la espera de que se defina, por parte de las compañías, como se va a informar a los usuarios sobre las nuevas tarifas con discriminación horaria.

- Definición de la arquitectura: La arquitectura desarrollada consta de dos partes principales, distribuida y centralizada. La parte distribuida se encarga de planificar, mientras que la parte centralizada realiza el balance de energía y potencia, selecciona la planificación más eficiente energéticamente y la ejecuta.

- Comprobación de mejora en la eficiencia energética: La capa de coordinación es la encargada de realizar el balance de potencia y energía de las diferentes planificaciones. El sistema de control selecciona la planificación de mayor eficiencia energética entre las establecidas por los diferentes subsistemas.

En conclusión, el desarrollo de este proyecto nos ha permitido demostrar que la Gestión de la Demanda Eléctrica provoca una mejora considerable en la eficiencia energética de una vivienda cuyas cargas de trabajo sean programables temporalmente.

6.2. Líneas futuras

Como se mencionó en el Capítulo 1, el sistema de control se encuentra dentro del proyecto GeDELOS-FV. Este proyecto dispone de un año más de trabajo en el que se seguirá desarrollando el sistema de control hacia una versión totalmente distribuida. Se buscará conseguir una mayor eficiencia energética con la mejora de las funciones encargadas de realizar la planificación, no teniendo en cuenta tan sólo parámetros temporales en la primera parte de la arquitectura, sino también información energética.

Un futuro desarrollo del sistema de control debe tener en cuenta un número ilimitado de cargas de cualquier naturaleza. Con esto, el sistema podría ser integrado en grandes consumidores eléctricos que dispongan de un elevado número de dispositivos, como puede ser el sector servicios (hoteles, hospitales, edificios del gobierno, etc.) e incluso incorporarse a la industria.

Otra línea de desarrollo es la coordinación de diferentes sistemas de control permitiendo que varias viviendas compartan recursos. El objetivo debe ser que el balance energético del conjunto sea el más eficiente posible buscando un mejor aprovechamiento de la energía solar entre todas las viviendas.

La incorporación de funciones de optimización más avanzadas se perfila como una de las líneas de desarrollo futuras más interesantes. Utilizando técnicas como redes neuronales, algoritmos genéticos, lógica borrosa, etc. (Escolano et al., 2003), se puede dotar al sistema de una mayor inteligencia haciendo que la vivienda aprenda a regular su consumo y hacerlo más eficiente, siempre dándole prioridad a la comodidad del usuario.

6.3. Contribuciones

Este PFC describe investigaciones originales llevadas a cabo por el autor. Parte del contenido de este PFC esta basado en artículos que, durante el desarrollo del proyecto, el autor junto a un numero de colaboradores han publicado. Dichas publicaciones se

detallan a continuación:

- Manuel Castillo, Álvaro Gutiérrez, Félix Monasterio-Huelin, Daniel Masa, Estefanía Caamaño, Javier Jiménez-Leube y Jorge Porro. Sistema de Control Distribuido para la Gestión de la Demanda en el Sector Residencial. *I Congreso de Generación Distribuida.*

- Estefanía Caamaño-Martín, Daniel Masa, Álvaro Gutiérrez, Félix Monasterio-Huelin, Javier Jiménez-Leube, Jorge Porro y Manuel Castillo. Optimización del Uso de un Sistema Fotovoltaico Mediante Gestión Activa de la Demanda. *I Congreso de Generación Distribuida.*

- Estefanía Caamaño, Daniel Masa, Álvaro Gutiérrez, Félix Monasterio-Huelin, Javier Jiménez-Leube, Jorge Porro and Manuel Castillo. Optimizing PV use through active demand side management. *24th European Photovoltaic Solar Energy Conference.*

Apéndice A

Guía Rápida gSOAP

A.1. Introducción

La herramienta *gSoap* realiza una conexión de lenguaje entre el protocolo SOAP/XML y el lenguaje de programación C/C++ facilitando considerablemente el desarrollo de servicios web y aplicaciones cliente/servidor en estos lenguajes.

SOAP (siglas de Simple Object Access Protocol) es un protocolo para el intercambio de estrucutras de datos en la implementación de servicios web. Este intercambio de datos depende de XML (eXtensible Markup Language) como formato de mensaje. SOAP puede formar la capa básica de una pila de un protocolo de servicio web, proveyendo un fragmentado de mensajes básico a partir del cual puede formarse el servico web.

Muchas herramientas para C/C++ usan SOAP para implementar los servicios web y ofrecen APIs que requieren el uso de librerías de clases para estructuras de datos específicas de SOAP, lo que nos obliga a adaptar la lógica de la aplicación a estas librerías. En cambio *gSoap* provee a C/C++ las APIs de SOAP de forma transparente ocultando al usuario detalles irrelevantes de las especificaciones de SOAP.

El objetivo de esta guía es mostrar los pasos básicos de esta herramienta, desde el principio hasta conseguir las librerías con las clases de los objetos que posteriormente nos servirán para hacer peticiones al servidor deseado. Veremos también como compilar nuestros programas con *gSoap* y una explicación de las librerías que nos

crea de forma que podamos enteder como comunicarnos con el servidor. Aunque los pasos indicados y la información son bastante genéricos hay que resaltar que están orientados a la versión de *gSoap* 2.7.11 y en el lenguaje C++.

A.2. Como usar gSoap

A.2.1. Obtener gSoap

gSoap es una herramienta de software libre la cual podremos encontrar en su página oficial junto a su manual (http://www.cs.fsu.edu/ engelen/soap.html). *gSoap* está disponible en los sistemas operativos Linux, Windows y MaxOS.

Desde linux podemos descargar el paquete desde los repositorios, o bien obtener el código que encontraremos en la página antes citada.

La herramienta *gSoap* es autocontenida por lo que no hará falta descargar ningún paquete más, a no ser que queramos usar OpenSSL debiendo obtener las librerías correspondientes.

A.2.2. Herramientas a usar

Una vez instalado *gSoap* tendremos dos herramientas básicas con las cuales generaremos todas las librerías:

- wsdl2h
- soapcpp2

WSDL son las siglas de Web Services Description Language, un formato XML que se utiliza para describir servicios web, actualmente se usa la versión 2.0. WSDL describe la interfaz pública a los servicios Web. Está basado en XML y describe la forma de comunicación, es decir, los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo. Las operaciones y mensajes que soporta se describen en abstracto y se ligan después al protocolo concreto de red y al formato del mensaje. En nuestro caso wsdl2h (programa

cliente) se conecta al servidor especificado, lee el WSDL para determinar que funciones están disponibles en el servidor y los tipos de datos con los que trabaja. Todo esto nos lo entregará en una librería la cual usaremos posteriormente con el compilador soapcpp2 para crear las librerías de comunciaciones para nuestro programa C/C++.

A.2.3. Generación de las librerías

En este apartado explicaremos rápidamente como conseguir las librerías para usar en nuestro porgrama. Posteriormente hablaremos de los posibles cambios en estas librerías y de su estructura.

Estos pasos se explican a modo de ejemplo para un servidor que tenemos en casa para el control domótico de electrodomésticos. En la página web oficial podremos realizar múltiples ejemplos los cuales tambien vienen explicados paso a paso.

Lo primero que debemos hacer es pedir las funciones y las estructuras de datos al servidor que tiene esta información en formato WSDL:

```
wsdl2h -o salida.h http://<IP>:8080/sah-ws
```

Esta URL será donde se encuentre la información del formato WSDL en el servidor. Esta instrucción genera la librería con las declaraciones de las APIs y el tipo de dato de sus parámetros. Por defecto generará el código suponiendo que se usará C++, para producir una aplicación en C puro se utiliza la opción '-c':

```
wsdl2h -c -o salida.h http://<IP>:8080/sah-ws
```

La salida de esta instrucción (en nuestro caso salida.h) tiene forma de librería en C y como ya se ha mencionado contiene ya las APIs del servidor como si fueran funciones en C, pero todavía no hemos generado los stubs para las APIs de C/C++. Para hacer esto ejecutamos el compliador soapcpp2:

```
soapcpp2 -i -C -I(import) salida.h
```

Como vemos en el ejemplo tenemos un parámetro que es (import) esto deberemos cambiarlo por la dirección donde se encuentre la librería stlvector.h para soportar vectores STL. Esta librería está dentro del directorio donde hayamos instalado *gSoap*:

P.ej: `-I/home/control/gsoap/gsoap-2.7/gsoap/import`

La opción -i indica que queremos objetos C++ proxy y servidor que incluye el cliente. La opción -C indica que sólo queremos los archivos del lado del cliente. Como puede verse, el último parámetro (salida.h) será la salida de la herramienta wsdl2h (la cual puede ser modificada).

Ahora ya tendremos en nuestro directorio todo el código necesario para hacer peticiones al servidor, incluidas las tablas XML. Este código debe incluirse en nuestro programa como veremos más adelante.

A.2.4. Uso en nuestro programa

Como ya hemos comentado, tras ejecutar el compilador soapccpp2 obtenemos las librerías necesarias para que nuestro porgrama pueda comunicarse con el servidor, mediante las funciones del objeto encargado de realizar la comunicación.

Lo primero que tendremos que hacer es incluir las librerías en nuestro porgrama:

```
#include "soap(nombre\_servicio)Proxy.h"

#include "(nombre\_servicio).nsmap"
```

El nombre del servicio dependerá del servidor y lo genera automáticamente soapcpp2, un ejemplo en un servidor el cual su servicio se llamara SAHBinding:

```
#include "soapSAHBindingProxy.h"

#include "SAHBinding.nsmap"
```

La primera de las librerías es la que contiene la clase con la cual crearemos el objeto encargado de realizar la comuniación con el servidor. La segunda hace referencia al mapa de espacios de nombres utilizado. A continuación mostramos un ejemplo de programa que usa estas librerías para comunciarse:

```
main() {
   //Objeto encargado de comunicarse con el servidor.
   SAHBindingProxy servicio;
   //Estructura para recibir los datos.
   ns1__getAvailableDevicesIDsResponse resp;
   int numdisp;
   //Sin problemas.
   if (servicio.getAvailableDevicesIDs(resp)== SOAP_OK){
      numdisp = resp.return_->__size;
      std::cout << "Tenemos "<< numdisp << " elementos." << std::endl;
   else
      //Informe de error.
      servicio.soap_stream_fault(std::cerr);
   return 0;
}
```

Lo primero que hace el programa es crear el objeto servicio (puede usarse cualquier nombre). Este objeto viene de la clase (nombre_servicio) Proxy la cual se encuentra en la librería antes mencionada. Esta es la relación que tiene nuestro programa con SOAP, ya que toda comunicación son llamadas a funciones de esta clase quedando oculto para nosotros toda lo referente a la comunicación. Posteriormente, creamos una variable del tipo "ns1__getAvailableDeviceIDsResponse" la cuál será una estructura de datos creada por *gSoap* para recibir la información preveniente de nuestra función. Posteriormente en la Sección A.3 veremos como ver estas estructura y como modificarlas.

Realmente las funciones creadas por *gSoap* nos devolverán la información en el último de los parámetros de la función de la clase de comunicaciones ya que el dato que nos devuelve directamente la función es un informe sobre el estado de la conexión. En nuestro caso, si no ha habido ningún problema en la comunicación seguiremos adelante y trabajaremos con la información que nos ha devuelto el servidor dentro de la estructura que nosotros tenemos en la variable 'resp'. En caso de error, *gSoap* nos imprimirá por pantalla un informe del error. Los tipos de errores no serán tratados en esta guía rápida.

A.2.5. Compilación

Una vez tenemos nuestro programa acabado debemos incluir el código fuente del motor de *gSoap* en la compilación. Este código se encuentra en el archivo "stdsoap2.cpp"que encontraremos junto a "stdsoap2.h.[en] el directorio raíz de *gSoap*. Una vez copiemos estos archivo en el entorno de trabajo, podremos proceder a la compilación. A continuación se muestra un ejemplo de la llamada al compilador:

```
g++ -o IDdisp IDdisp.cpp soapC.cpp stdsoap2.cpp soapSAHBindingProxy.cpp
```

Cada archivo tendrá su utilidad:

- El código de nuestro programa (IDdisp.cpp en este ejemplo).
- La implementación de las funciones usadas (soapC.cpp y soapSAHBinding-Proxy.cpp).
- El código del motor del entorno de *gSoap* (stdsoap2.cpp).

A.3. Librerías

Esta sección pretende comentar los archivos generados por *gSoap* y como poder ver en ellos las APIs generadas y las estructuras de datos con las que trabajará nuestro programa. Además se mostrará como modificar los tipos de datos que manejará nuestro programa.

- Salida de wsdl2h:

Este fichero, que hemos llamado "salida.h", sólo hace falta para compilar el resto con la herramienta `soapcpp2`. En este fichero pueden verse las APIs de nuestro servidor y las estructuras de datos con las que vamos a trabajar. Al ser el archivo intermedio entre la petición del protocolo WSDL y la compilación final de nuetras librerías, nos permite facilmente modificar los tipos de datos con los que vamos a trabajar.

Dentro de este archivo, lo primero que nos encontramos en una cabecera expliacando como se ha creado esta librería e información sobre el siguiente paso de compilación. Despues viene las diferentes secciones.

La sección "Schema Typesçontiene información sobre las estructuras de datos que se van a utilizar. Es importante repasar estas estructuras porque serán las que usemos en nuestros programas y es muy común cometer errores de compilación debido a lo enrevesadas que pueden llegar a ser las relaciones entre estas estructuras. Como se ha comentado anteriormente, ya que aún no se han compilado las librerías que usaremos directamente en nuestros programas, podemos realizar cambios en estas estructuras de datos de forma que cuando se reciban los datos en XML y los pasemos a C/C++ podemos definir a que tipo. Aquí podemos ver un ejemplo:

Código inicial:

```
/// "urn:sah":DeviceStateVariable is a complexType.
class ns1__DeviceStateVariable : public xsd\__\_anyType {
    public:
/// Element variablename of type xs:string.
    std::string       variablename   1; ///< Required element.
/// Element value of type xs:anyType.
    xsd__anyType*   value          1;   ///< Required element.
};
```

Código modificado:

```
/// "urn:sah":DeviceStateVariable is a complexType.
class ns1__DeviceStateVariable : public xsd\__\_anyType {
    public:
/// Element variablename of type xs:string.
    std::string       variablename   1; ///< Required element.
/// Element value of type xs:anyType.
    std::string       value          1; ///< Required element.
};
```

Como podemos ver se ha pasado del tipo `xsd__anyType*` a `std::string` de forma que cuando usemos la estructura `ns1__DeviceStateVariable` y queramos ver el contenido de la variable value podremos tratarla como una cadena.

Por ejemplo, podríamos hacer también esto de un entero a una cadena, de forma que en vez de recibir el valor numérico *gSoap* nos pasaría los datos a cadena de caracteres. El problema de esto, es que podemos provocar serios problemas a la

hora de trabajar con los datos como podría ser pasar caracteres a enteros. Por eso recomendamos que cualquier cambio se haga estando muy seguro de lo que se esta haciendo.

Más adelante llegamos a la sección "Services", en la cual podemos ver las funciones disponibles en nuestro servidor es decir las APIs. Aquí podemos ver tambien la URL a donde se va a realizar la conexión con el servidor, esta URL pude modificarse en caso de que por ejemplo cambiemos el IP del servidor (NOTA: puede ocurrir que cuando se crea esta librería no se escriba automáticamente bien la URL, por lo que conviene comprobar que esta URL es corecta). Posteriormente se definen todas las APIs. También están algunas definiciones de estructuras de datos como son las estructuras de las respuestas del servidor.

Este archivo, aunque sólo se usará para la generación de las demás librerías, permite tener una buena idéa de como funciona el servidor y de los servicios que nos va a poder proporcionar.

- soap(nombre_servicio)Proxy.h:

Esta librería, la cual debe ser llamada directamente por nuestro programa, es la que contiene la clase del objeto de comunicaciones que despues usaremos durante todo nuestro programa para comunicarnos con el servidor. Se puede cambiar el nombre de esta clase pero deberemos hacerlo tambien en el archivo soap(nombre_servicio)Proxy.cpp, ya que es donde estan declaradas las funciones de la clase.

Primero encontramos funciones propias del la comunicación con SOAP, las cuales no se estudiarán en esta guía. Después de estas funciones nos econtramos con las APIs de nuestro servidor. Aquí podemos ver claramente las estructuras de datos que se van a usar como parámetros de las diferentes funciones.

- soapStub.h:

Se encuentran definidas detalladamente las clases y estructuras de datos utilizadas para la comunicación con el servidor.

- soapH.h:

Funciones internas de *gSoap* usadas para la comunicación.

- stdsoap2.h:

Librería con el motor del entorno de *gSoap*.

Bibliografía

Aguilar, L. J. (2002). *Programación en C++: Algoritmos, estructuras de datos y objetos*. Mc GrawHill, Madrid, España.

Bennett, S. (1979). *A History of Control Engineering 1800-1930*. London: Peregrinus on behalf of the Institution of Electrical Engineers, London, UK.

B.O.E. Ley 54/1997, de 27 noviembre, del Sector Eléctrico. Boletín Oficial del Estado núm. 285, del 28 de noviembre de 1997.

Bruendlinger, R., Bletterie, B., and Mayr, C. (2007). *PV-Wechselrichter als aktive Filter zur Vermessung der Netzqualität - Was Können moderne Geräte leisten?* Proceeding 22 Symposium Photovoltaische Solarenergie, Staffelstein, Germany.

Caamaño, E., Suna, D., Thornycroft, J., Cobben, S., Elswijk, M., Gaiddon, B., Erge, T., and Laukamp, H. (2007). *Utilities experience and perception of PV distributed generation*. E.U.

Degner, T., Schmid, J., and Strauss, P. (2006). *Distributed generation with high penetration of renewable energy sources*. DISPOWER, Kassel, Germany.

D.R.E.D. (2007). El consumo eléctrico en el mercado peninsular. informes de los años 1998 a 2006. Technical Report CNE PA004/08, Comisión Nacional de la Energía, España.

Escolano, F., Cazorla, M. A., Alfonso, M. I., Colomina, O., and Lozano, M. A. (2003). *Inteligencia artificial: modelos, técnicas y áreas de aplicación*. Thomson, Alicante, España.

Fournier, R. (1999). *A methodology for client/server and web application development.* Yourdon Press, USA.

Graham, S. and Kumar, P. R. (2003). *The Convergence of Control, Communication and Computation?* Lecture Notes in Computer Science, Volume 2775, Springer-Verlag, Berlin, Germany.

Groppi, F. (2002). Grid connected photovoltaic power systems: power value and capacity value of pv systems. Technical Report IEA PVPS T5-11: 2002, CESI, Italy.

Huidobro, J. M. (2003). *Tecnología avanzada de telecomunicaciones.* Paraninfo, Madrid, España.

Imaz, L. (2007). *La contribución de la gestión de la demanda ala eficiencia del sistema eléctrico.* Jornadas Internacionales de Equipos Eléctricos, Bilbao, España.

Liang, J. and Du, R. (2008). *Design of intelligent comfort control system with human learning and minimun power control strategies.* Energy Conversion and Management, Hong Kong, China.

Linder, S. (2007). *Solar Urban Planning - PV in Urban Environment: Germany.* Jornadas Aplicación de la Energía Solar Fotovoltaica desde el Urbanismo, E.T.S. Arquitectura U.P.M., Madrid, España.

Moon, S., Nahan, R., Warner, C., and Wassmer, M. (2005). Solar decathlon 2005: The event in review. Technical Report DOE/GO-102006-2328, National Renewable Energy Laboratory, U.S.A.

Paper, G. (2002). Towards a eu strategy for security of energy supply. Technical Report (2002)321, CEC, Brussels, Belgium.

Pérez, J. I., Sánchez, L. J., and Pardo, M. (2005). *La gestión de la demanda de electricidad.* Ed. Fundación Alternativas, Madrid, España.

Ray, J. (2000). *Guía esencial Linux.* Prentice Hall, USA.

R.E.E (1998). *Atlas de la demanda eléctrica en España. Proyecto INDEL.* Ed. Red eléctrica de España, España.

R.E.E (2006). *El sistema electrico español 2006.* Ed. Red eléctrica de España, España.

Romero, C. (2004). *Domótica e Inmótica. Vivienda y Edificios Inteligentes.* Ra - Ma, Madrid, España.

S.G.P.E. (2007). Planificación de los sectores de electricidad y gas 2008-2016. Technical report, Ministerio de Industria, Turismo y Comercio, España.

Spurgeon, C. (2000). *Ethernet: the definitive guide.* O'Reilly, USA.

Tecnalia, L. (2007). *Guia básica de la gestión de la demanda eléctrica.* Fundación de la energía de la comunidad de Madrid, Madrid, España.

Valavanis, K. P. and Saridis, G. N. (1992). *Intelligent Robotic Systems: Theory, Design and Applications.* Kluwer Academic Publishers, USA.

Valentí, J. I. F. (2007). *Eficiencia energética en redes eléctricas desde la perpectiva de las empresas distribuidoras.* Jornadas de la Cátedra Endesa Red de Innovación Energética, Barcelona, España.

van Engelen, R. (2008). *gSOAP 2.7.11 User Guide.* Florida State University, USA. http://www.cs.fsu.edu/ engelen/soap.html.

Vasilakos, A. and Pedriyzc, W. (2006). *Ambient Intelligence, wireless networking and ubiquitous computing.* Artech House Publishers, USA.

Wakao, S., Mikami, T., and Tamura, F. (2005). *Voltage control in small-scale disperser PV power generation system of networked composition.* Proceedings 20th European Photovoltaic Solar Energy Conference and Exhibition, Barcelona, España.