Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

Máster en Ingeniería Industrial

MÁSTER EN INGENIERÍA INDUSTRIAL

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

TRABAJO FIN DE MÁSTER

INSTALACIÓN DOMÓTICA BASADA EN OPENHAB Y RASPBERRY PI

Autor: D. Ricardo Vega Alonso

Tutor: D. Eduardo Zalama Casanova

Valladolid, 29 de agosto de 2016

Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

Máster en Ingeniería Industrial

MÁSTER EN INGENIERÍA INDUSTRIAL

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

TRABAJO FIN DE MÁSTER

INSTALACIÓN DOMÓTICA BASADA EN OPENHAB Y RASPBERRY PI

Autor: D. Ricardo Vega Alonso

Tutor: D. Eduardo Zalama Casanova

Valladolid, 29 de agosto de 2016

Resumen

El presente trabajo pretende aportar una solución tecnológica viable al cuidado de una población anciana cada vez más numerosa, integrando para ello un sensor que ha sido específicamente diseñado para ser capaz de detectar la ocupación o no de una cama, un sensor de presencia comercial y un controlador central conformado por una Raspberry Pi y OpenHAB como software de control. Todo ello, forma un sistema inalámbrico utilizando un dispositivo RFXCOM que aporta una comunicación efectiva entre los elementos que conforman el sistema y un robot antropomorfo externo.

Abstract

The present paper aims to provide a viable technological solution to care an increasingly large elderly population , integrating a sensor for it which it has been specifically designed to be able to detect if a bed is occupied or not, a commercial presence sensor and a central controller formed by a Raspberry Pi and OpenHAB as control software. All this system provides a wireless system using a RFXCOM device that grants an effective communication between the elements of the system and an external anthropomorphic robot.

Agradecimientos

Me gustaría comenzar este trabajo mostrando mi agradecimiento a Eduardo Zalama, mi tutor de este Trabajo de Fin de Máster, por la ayuda y compresión demostrada a lo largo de la realización del mismo, aportando reflexiones e indicaciones que sin duda han sido esenciales para su finalización.

También me gustaría agradecer a la Universidad de Valladolid y al Departamento de Ingeniería de Sistemas y Automática de la Escuela de Ingenierías Industriales por haber aportado los recursos e instalaciones necesarias para la ejecución de este trabajo.

Quiero hacer una mención especial a mi familia que siempre estuvo apoyándome y creyeron en mí a pesar de las adversidades. Gracias a ellos he aprendido que el esfuerzo siempre merece la pena.

No deseo terminar sin agradecer a Ángela su paciencia y ánimo constante así como el apoyo que he sentido de mis amigos durante todo este tiempo.

A todos ellos, mi familia, mi novia y mis amigos, agradeceros la confianza depositada en mí y el haberme enseñado a crecer como persona. Sin vosotros, esto no hubiera sido posible.

Índice general

Índice de figuras

Índice de cuadros

Glosario

API

Application Programming Interface - Interfaz de programación de aplicaciones. 21

ASK

Amplitude-shift keying - Modulación por desplazamiento de amplitud. 17

BAC

Building Automation and Controls - Equipos y sistemas de automatización y control de edificios. 8

BM

Bussiness Management - Gestión de Edificios. 8

CEDOM

Asociación Española de Domótica e Inmótica. 7

CSMA-CA

Carrier Sense Multiple Access with Collision Avoidance - Protocolo de control de acceso a redes de bajo nivel que permite que múltiples estaciones utilicen un mismo medio de transmisión. 25

CVS

Control Version Software - Software de Control de Versiones. 65

HMI

Human Machine Interface - Interfaz hombre máquina. 15

ISO

International Organization for Standardization - Organización Internacional de Normalización. 15

OSI

Modelo de red descriptivo, que fue creado por la ISO en el año 1980. Es un marco de referencia para la definición de arquitecturas en la interconexión de los sistemas de comunicaciones. 14

SBC

Single-board Computer - Placa computadora. 19

TBM

Technical Bussiness Management - Gestión Técnica de Edificios. 8

Prefacio

El control y automatización de plantas industriales es a día de hoy una disciplina totalmente asentada y demandada en el mercado debido a los grandes beneficios que aporta a la explotación económica de la actividad industrial. La domótica y la inmótica pretenden acercar los beneficios del control automatizado al ámbito doméstico y el sector servicios. Tradicionalmente, estas tecnologías han estado lastradas por los altos costes en relación con los beneficios obtenidos, sin embargo, un interés cada vez más evidente de la sociedad en ellas, así como el desarrollo del software y hardware libre han democratizado el acceso a las mismas.

Capítulo 1

Introducción

Desde la Revolución Industrial, la implantación de soluciones capaces de automatizar diferentes procesos para así reducir costes y aumentar la seguridad del trabajador han sido la tónica general.

A día de hoy, cien o doscientos años después de que comenzara ese proceso, el desarrollo de la automatización industrial es evidente y este tipo de soluciones están totalmente normalizadas.

Sin embargo, no fue hasta finales de los años 70 cuando se empezó a mirar a la vivienda como posible espacio de aplicación de las técnicas desarrolladas en la industria, naciendo así la **domótica**.

Su evolución, aunque lenta e irregular, no se ha detenido, creando un espacio de trabajo cuyos objetivos no son exactamente los mismos que los existentes en la industria por lo que las soluciones dadas para ciertos problemas divergen.

Por otro lado, en estos últimos dos siglos, la evolución socio-económica que ha experimentado el mundo es realmente espectacular comparada con cualquier otro periodo de la historia.

Esta situación ha llevado a nuevas necesidades a cubrir por el ser humano a las cuales la tecnología se enfrenta cada día, para intentar brindar al mundo un futuro mejor.

Una de estos preocupantes desafíos es el **envejecimiento de la población** y es que, fijándonos en los últimos datos aportados por la ONU en su revisión del año 2015 sobre "Perspectivas de la Población Mundial"[22], actualmente en Europa el 24 % de la población está por encima de los 60 años, estimándose que para 2050 esta cifra puede alcanzar un 34 %, lo que supone un extraordinario ascenso si lo comparamos con el año 1950 donde este grupo de población representaba el 12 %.

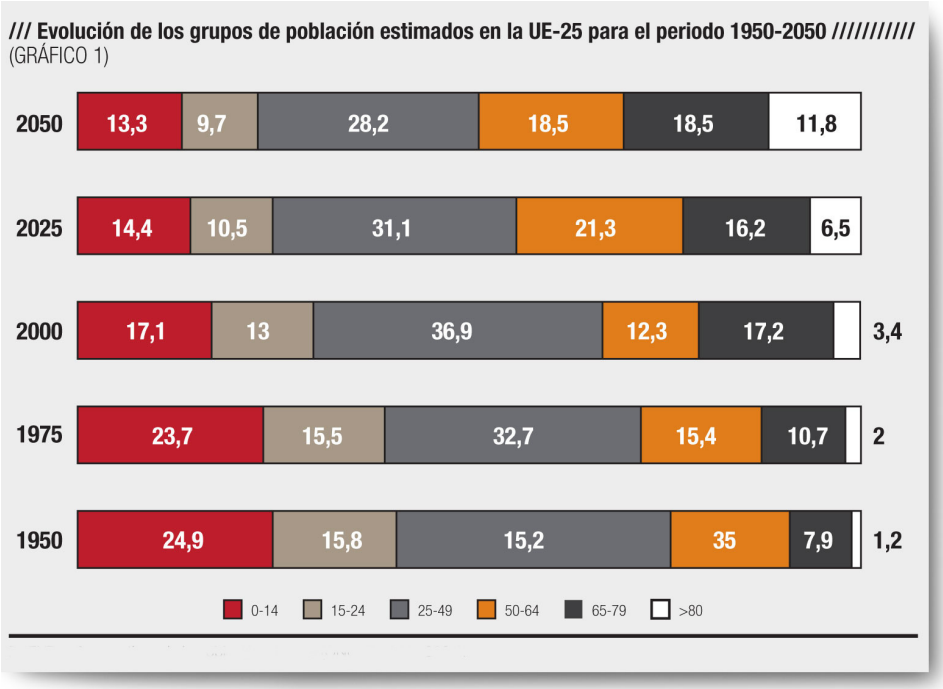

Figura 1.1: Perspectivas de la ONU sobre la distribución de la población.

Las consecuencias y retos que esta perspectiva representa son varios[17], pero desde el punto de vista que nos ocupa, son especialmente interesantes las ventajas que la automatización del hogar puede aportar al **cuidado, confort y seguridad de una población cada vez más envejecida**.

Por otra parte, gracias a la evolución que día a día experimenta el mundo de la tecnología y al abaratamiento de la misma precisamente fruto de este constante desarrollo, proyectos hasta hace poco tiempo inviables debido a su elevado coste, se convierten en **opciones reales incluso para ámbitos de reducido presupuesto**.

Así, ideas que pertenecían en exclusiva al mundo de la ciencia ficción, cobran significado con el paso del tiempo, estando disponibles para su utilización como soluciones a los complejos retos de nuestro presente y futuro.

Otra interesante consecuencia del abaratamiento de la tecnología es su democratización lo que permite a nuevos actores no sólo su consumo sino también su desarrollo. De este modo, en los últimos años hemos experimentado una verdadera explosión de proyectos software y hardware con licencias libres y abiertos a la comunidad que, en la mayoría de las ocasiones, es la responsable de su evolución.

En el mundo de la electrónica, Arduino es un célebre exponente de éste fenómeno y prácticamente se ha convertido en el estándar en soluciones electrónicas de bajo coste por su reducido coste y gran facilidad de desarrollo debido a la gran cantidad de librerías y documentación disponibles de forma gratuita.

Otro célebre protagonista es Linux y las distribuciones libres (como por ejemplo Debian o sus derivados) que, aprovechando el tirón de dispositivos de bajo coste como la Raspberry Pi, cada vez son más populares tanto para el usuario técnico como para el público común.

En este tipo de proyectos, la amplitud de la comunidad y su actividad marca en gran parte su desarrollo asociándose en múltiples ocasiones estas variables a la calidad del proyecto en sí. Además, desde hace escasos años y seguramente gracias al comienzo de la crisis económica que el mundo experimenta desde el año 2008, somos testigos de la creación de grupos locales de personas interesados en diferentes tecnologías, surgiendo así el movimiento "Maker" que se ha apoyado, creado y reforzado en todas estas alternativas abiertas.

Todas estas circunstancias han desembocado en la **existencia de alternativas libres** y en muchos casos gratuitas que se presentan como opciones reales frente a proyectos comerciales en gran número de escenarios lo que, una vez más acerca a más gente a estas soluciones y realimenta esta "rueda de innovación".

Este trabajo pretende aunar todos estos aspectos para aportar una solución basada en la automatización con herramientas libres que ayude en el problema del envejecimiento de la población.

1.1. Objetivos

Como hablábamos en el punto anterior, este trabajo pretende emplear los avances en el mundo de la automatización doméstica (domótica) y mediante elementos Open Source aportar una solución a uno de los problemas concretos asociados al envejecimiento de la población.

Concretamente, gracias a la solución aportada por este proyecto, intentaremos **ayudar en la compleja tarea que puede convertirse el cuidado y la atención de pacientes ancianos enfermos de demencia que se encuentran institucionalizados**.

De este modo, el fin del último de este trabajo es ayudar a garantizar la seguridad del paciente y facilitar la labor asistencial del personal encargado de su atención.

Fijándonos más a bajo detalle sobre los diferentes objetivos que nos marcamos cumplir, nos encontramos con varios puntos:

Por un lado, se pretende ahondar en las diferentes opciones domóticas ya disponibles para llevar a cabo tareas de monitorización de personas, haciendo especial hincapié en aquellas implementaciones libres existentes debido a su reducido coste y ser susceptibles de modificación para ser adaptadas a nuestras necesidades.

La calidad de la documentación y el tamaño y actividad de la comunidad involucrada en el proyecto serán parámetros fundamentales a la hora de seleccionar una alternativa u otra debido a, como indicábamos anteriormente, suele ser una parámetro bastante certero sobre la calidad de dicha opción y, por otra parte, creemos que facilitará cualquier tarea de mantenimiento y/o ampliación del sistema.

Por otro lado, se investigarán diferentes opciones hardware de cara a la elección de los elementos de sensorización necesarios para llevar a cabo nuestro cometido y en el caso de no existir éstas, se diseñará e implementará una **solución propia y específica**. En este último caso, y siguiendo la filosofía de buscar alternativas "Open Source", basaríamos nuestra propuesta en el **ecosistema Arduino**.

Con todo ello, plantearemos un sistema de control capaz de **detectar la presencia o no de una persona sobre una cama**, enviando el estado en todo momento al sistema para que puede ser monitorizado. Como se ha dicho en el párrafo anterior, este objetivo requerirá del diseño de toda la electrónica necesaria para llevar a cabo este cometido debido a la inexistencia de soluciones ya hechas con estas características.

También monitorizaremos otro "input" diferenciado como es la **existencia o no de una persona en una estancia** para lo cual emplearemos un sensor de presencia compatible con el sistema proyectado.

Ambos sensores proveerán de información en tiempo real a un panel de control accesible desde ordenadores y dispositivos móviles tales como smartphones o tablets autorizados. Dicho panel, y con el objetivo de asegurar la **compatibilidad multidispositivo**, seguirá un diseño *responsive*.

La información de estos sensores será empleada también para activar diferentes **alertas** que avisen al personal del Centro sobre situaciones concretas que pueden ser consideradas de riesgo para los pacientes como por ejemplo cuando un anciano desorientado salga de su habitación de noche, dando el aviso al personal antes de que el residente abandone las instalaciones.

Igualmente, **la comunicación de todo el sistema se establecerá de forma inalámbrica** ya que la ausencia de cables es un requerimiento del sistema.

Por tanto, la comunicación diseñada deberá lidiar correctamente con los problemas técnicos comunes en este tipo de instalaciones inalámbricas tales como alcance o interferencias a parte de otras posibles e inesperadas circunstancias. También deberá ser capaz de cumplir satisfactoriamente otros dos requisitos marcados por el proyecto como son:

- Ser lo más transparente posible tanto para pacientes como para el personal que trabaja en el centro.

- Ser capaz de comunicarse con un **robot antropomorfo** ya desarrollado que participa activamente en tareas de asistencia y entretenimiento de los pacientes del centro.

Además, dicho sistema debe ser perfectamente compatible y estar integrado con un estándar abierto de domótica/inmótica, para así facilitar tanto el mantenimiento como posibles ampliaciones futuras.

Por último, se pretende que el sistema resultante sea **energéticamente eficiente** y con un presupuesto ajustado sin por ello penalizar su robustez. Para ello, y como ya hemos comentado varias veces, usaremos herramientas libres y gratuitas siempre que sea posible para reducir costes tanto de licencias como de la solución en sí.

Por otra parte, a través de diversas iteraciones, buscaremos optimizar el sistema lo máximo posible pensando no sólo en su tiempo de ejecución sino también en su mantenimiento ya que en numerosas ocasiones, suele ser el apartado más caro de este tipo de soluciones debido a la inexperiencia del responsable de llevar esta tarea a cabo en este tipo de sistemas.

Por tanto, otro objetivo que se marca este proyecto es no limitarse simplemente a su ejecución y, puesto que en un futuro representará una instalación real en el Centro Asistencial San Luis situado en la localidad de Palencia, se tendrá especialmente en cuenta también las posteriores tareas de mantenimiento que se requieran en el sistema, anticipándose a ellas y facilitándolas siempre que sea posible. En este sentido, creemos que una cuidada documentación del sistema en sí junto con una serie de documentos especialmente pensados para la rápida actuación, reparación, recambio o calibración de la instalación pueden ser de gran ayuda para un hipotético futuro operario. Dicha información se verá también reflejada en este documento, ya sea dentro de la memoria en sí o como anexos finales.

Con todo, **la solución final se integrará en el Centro Asistencial antes citado mejorando las condiciones de seguridad de los pacientes y ayudando a los trabajadores en sus tareas de cuidado y de este modo aportando un valor añadido a dicho centro**.

1.2. Descripción de la memoria

La presente memoria pretende ser un reflejo de las tareas asociadas con este Trabajo Fin de Máster. De esta forma, consta de varias partes articuladas alrededor de las diferentes tareas de investigación realizadas así como de la implementación real del sistema como tal, aportando, a mayores, detalles sobre presupuesto, guías de calibración e instalación

donde se consideren necesarias o cualquier otro detalle que creamos conveniente para, a posteriori, este documento pueda servir de base para otra clase de proyectos relacionados.

Procedemos a continuación a detallar los diferentes contenidos presentes en esta memoria.

Así, en primer lugar en el capítulo *"Fundamentos Teóricos"* trataremos de forma descriptiva temas como:

- ¿Qué entendemos por domótica?
- Protocolos más comúnmente empleados (haciendo un mayor hincapié en los inalámbricos).
- ¿Qué es la Raspberry Pi?
- ¿Qué protocolos libres existen y están disponibles para su uso en la Raspberry Pi?
- Descripción de las principales características teóricas de la solución seleccionada.

A continuación, nos centraremos en el desarrollo como tal del proyecto, entrando más en profundidad en los aspectos que se han tenido en cuenta a lo largo del mismo desde un punto de vista mucho más ingenieril y dando detalles sobre la implementación en sí.

Esta información la separaremos en dos capítulos. Así, en *"Diseño y selección de los sensores empleados"* trataremos aspectos relacionados con los estudios y diseños realizados de cara a abordar los retos técnicos relativos a los sensores que comoponen la instanción mientras que en *"Configuración del nodo central y otras consideraciones técnicas"* nos centraremos en el nodo central y las diferentes configuraciones que debos llevar a cabo para su correcto funcionamiento.

Continuaremos con un capítulo *"Resultados"* donde presentaremos diferentes métricas y resultados evaluados para la solución dada.

Finalmente, y para acabar, se aportarán diferentes detalles adicionales tales como una valoración económica (capítulo *"Estudio Económico"*) que nos ayudarán a terminar dando una serie de recomendaciones a futuro así como unas conclusiones generales en un último capítulo llamado *"Conclusiones"*.

Como indicamos anteriormente, reservaremos un espacio final ya en los anexos para adjuntar aquellas guías de configuración, instalación y/o mantenimiento que consideremos oportunas para la posterior actuación en la instalación.

Capítulo 2

Herramientas de desarrollo disponibles

Dedicaremos este primer capítulo a dar una visión sobre el estado del arte que sirva, en futuros capítulos, como base de conocimiento para el desarrollo del proyecto.

De este modo, y atendiendo a los objetivos a cumplir, necesitaremos conocer las tecnologías aplicacadas en la **automatización del hogar y el sector servicios**. Creemos que sobre esta base, es realmente importante conocer los diferentes protocolos que se están empleando en la actualidad para tener una base teórica aliniada con el objetivo de estandarización que nos proponemos.

Por otro lado, nuestro objetivo de conseguir un sistema de bajo coste con elementos de software y hardware libre hace imprescindible, en nuestra opinión, el estudio de posibles soluciones para el nodo central que posibiliten la interconexión de diferentes dispositivos y sean suficientemente flexibles como para permitir añadir o eliminar elementos en un futuro. En este sentido, también exploraremos las opciones disponible relativas a software libre domótico como base sobre la que trabajar.

Por último, la necesidad marcada en los objetivos de conseguir una comunicación entre el sistema desarrollado y un robot antropomorfo hace imprescindible estudiar qué tecnologías existen al respecto y su compatibilidad con el resto del sistema.

2.1. Qué es domótica

Parece adecuado dedicar estas primeras líneas a definir que entendemos por **domótica**. Para ello nos apoyamos en la definición que nos brinda CEDOM (Asociación Española de Domótica e Inmótica):

> "La domótica es el conjunto de tecnologías aplicadas al control y la automatización inteligente de la vivienda, que permite una gestión eficiente del uso de la energía, que aporta seguridad y confort, además de comunicación entre el usuario y el sistema."

Los sistemas domóticos son capaces de obtener la información proveniente de sensores (entradas) y, tras procesarla, emitir órdenes concretas a los actuadores (salidas). Además, esta interconexión de elementos puede ampliarse para conseguir un control de la instalación desde el exterior de la vivienda. Transformando casas y hogares en lugares más humanos, más personales, polifuncionales y flexibles.

La red de control del sistema domótico se integra con la red de energía eléctrica y se coordina con el resto de redes con las que tenga relación: telefonía, televisión, y tecnologías de la información, cumpliendo con las reglas de instalación aplicables a cada una de ellas. Las distintas redes coexisten en la instalación de una vivienda o edificio. La creciente popularidad de los sistemas domóticos hace que incluso exista un instrucción técnica dedicada a ellos, en particular, la red de control del sistema domótico está regulada por la instrucción ITC-BT-51 *Instalaciones de sistemas de automatización, gestión técnica de la energía y seguridad para viviendas y edificios.*

Por otra parte, existe el término inmótica, que aunque semejantes no debemos confundirlos ya que éste se aplica a edificios del sector terciario en vez de a viviendas. Una vez más, vamos a fijarnos en la definición de CEDOM.

> "La inmótica es el conjunto de tecnologías aplicadas al control y la automatización inteligente de edificios no destinados a vivienda, como hoteles, centros comerciales, escuelas, universidades, hospitales y todos los edificios terciarios, permitiendo una gestión eficiente del uso de la energía, además de aportar seguridad, confort, y comunicación entre el usuario y el sistema."

Los equipos y sistemas de automatización y control de edificios (BAC) proporcionan funciones de control efectivas para aplicaciones como calefacción, ventilación, refrigeración, agua caliente, iluminación, etc., lo que conduce a una mayor eficiencia, tanto energética como operacional. Se pueden configurar funciones y rutinas de ahorro de energía complejas e integradas, basadas en el uso diario del edificio, dependiendo de las necesidades reales del usuario, con el fin de evitar un consumo de energía y unas emisiones de CO2 innecesarios.

Las funciones de la gestión técnica de edificios (TBM), como parte de la gestión de edificios (BM), proporcionan información sobre el funcionamiento, el mantenimiento, los servicios y la gestión de edificios; especialmente para la gestión de la energía, capacidad de

medición de registro de tendencias, y de generación de alarmas y diagnóstico del consumo de energía innecesario. La gestión de la energía es una condición para la documentación, regulación, supervisión, optimización, determinación y para soportar las acciones correctivas y preventivas que mejoren la eficiencia energética de los edificios.

Cabe destacar la estrecha relación existente entre medición, control y optimización.

> "La medición es la primera etapa que conduce al control. Si no puedes medir algo, no lo puedes comprender. Si no lo puedes comprender, no lo puedes controlar. Si no lo puedes controlar, no lo puedes mejorar." - H. J. Harrington.

2.1.1. Evolución Histórica

Podemos situar el comienzo de la domótica en los años setenta, cuando un grupo de investigadores británicos configuran el primer protocolo pensado para comunicar dispositivos entre sí al cual llamaron X-10.

En su momento se pensó que podríamos estar ante otro gran nicho de mercado como estaba siendo el sector de la electrónica generalista y las telecomunicaciones, sin embargo, desde los años noventa, su evolución se ha visto frenada debida a diferentes circunstancias:

- **El elevado desconocimiento existente entre los potenciales usuarios de esta tecnología**. Este desconocimiento lleva irremediablemente a una escasa demanda y es que, en muchos casos, se asocia esta tecnología con sistemas caros, elitistas, con una compleja puesta a punto sin que necesariamente reporten grandes beneficios que justifiquen esta inversión.

  Impartir y fomentar el uso de la domótica es un factor clave para la progresión de la misma tanto en el aspecto técnico evitando empleos erróneos de la palabra domótica, como de cara al usuario, elevando el interés social.

- Hasta hace bien poco, la domótica era una **apuesta exclusiva de pequeñas empresas y startups**[1]. Sin embargo, la escasez de recursos tanto de personal como económicos hacen que compañías con este perfil no puedan considerarse como motores de una industria como ésta.

- También podríamos citar la **absoluta indiferencia que esta tecnología suscita en el mercado de la construcción**, incapaz de ver e incluir en su oferta viviendas que ofrezcan las numerosas ventajas de la domótica como recurso de valor añadido.

[1] organización humana con gran capacidad de cambio, que desarrolla productos o servicios, de gran innovación, altamente deseados o requeridos por el mercado, donde su diseño y comercialización están orientados completamente al cliente.

Sin embargo existen razones para considerar la domótica como un mercado de especial interés y con un enorme potencial.

- La **rehabilitación energética de edificios** convierte en potenciales clientes a todo el parque de viviendas español, mientras que los requisitos de eficiencia impuestos a las viviendas en alquiler o venta, extiende tanto a nuevas como antiguas viviendas, así como a toda clase de negocios, su potencial aplicación.

- El **cuidado del medio ambiente** es una necesidad y un reto para la sociedad del siglo XXI. La domótica tiene que ser considerada como elemento imprescindible en este apartado debido a la gestión que implementa en el uso de la energía existente y aguas.

- La **evolución demográfica** que esta experimentando la sociedad, con un incremento importante de la tercera edad convierte a la domótica en el factor esencial para dotar a las viviendas de los mecanismos adaptados a las nuevas necesidades existentes desarrollando servicios para usuarios (sistemas inteligentes de climatización, teleasistencia doméstica, etc.). Por otra parte, existirá un colectivo de jóvenes cuyas necesidades de comunicación aumentarán con el tiempo. Control a distancia de la calefacción, servicios telemáticos, etcétera serán demandas necesarias de este colectivo. En definitiva, la domótica será la tecnología que asegure en cierto modo las necesidades de bienestar futuras de los usuarios en el ámbito de su vivienda.

- El incipiente **interés** que está despertando esta tecnología en empresas de mayor calado, incluso grandes empresas tecnológicas americanas. Esta tendencia se ve reflejada en muchos en lo que se ha llamado **"El Internet de las Cosas"**.

2.1.2. Arquitecturas disponibles

Típicamente los sistemas de control se pueden agrupar en dos categorías: sistemas centralizados y sistemas distribuidos. La domótica e inmótica, al ser una rama de la automatización y control, pueden seguir la misma clasificación.

Sistemas Centralizados

En esta clase de arquitectura, contamos con un controlador central, normalmente un autómata o micro-autómata, desde donde cableamos, tanto las entradas como las salidas. Estamos, por tanto ante conexiones punto a punto (P2P) entre los elementos de campo (sensores y actuadores) y el controlador.

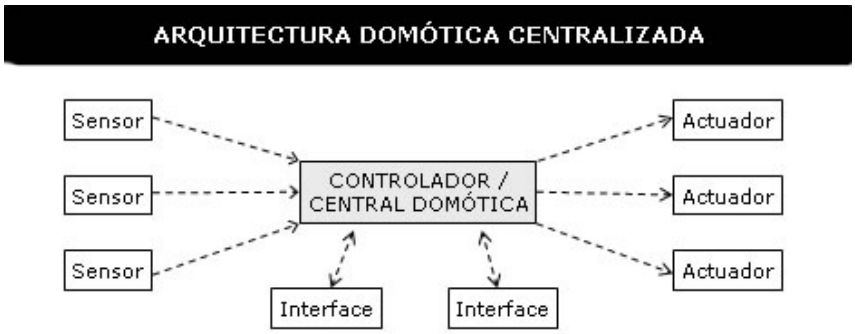

Figura 2.1: Ejemplo de arquitectura centralizada.

El funcionamiento es bastante simple y se ajusta al clásico funcionamiento de un autómata, es decir, el programa de control que se encuentra en su memoria se ejecuta cíclicamente describiendo el llamado **Ciclo de Scan**.

1. **Lectura de entradas**, creando una imagen en memoria de ellas con la cual trabajará en los siguientes pasos del ciclo.
2. **Ejecución del programa** de control almacenado en la memoria del autómata.
3. **Escritura de salidas**, actualizando todas aquellas que difieren con la información de memoria almacenada para su posición.
4. **Tareas internas del PLC**. Tareas como la comprobación de errores se ejecutan en este momento, antes de comenzar un nuevo ciclo.

En la Figura 2.1 podemos apreciar el esquema de lo que sería una arquitectura centralizada.

Para la programación de autómatas tenemos disponibles diferentes lenguajes de programación como pueden ser:

- **KOP o LD**, basado en los diagramas de contactos.
- **AWL o IL**, basado en listas de instrucciones, muy semejante al lenguaje ensamblador.
- **FUP o FBD**, basado en los diagramas de bloques.
- **ST**, lenguaje estructurado de alto nivel.
- **SFC o GRAFCET**, basado en los diagramas de contactos.

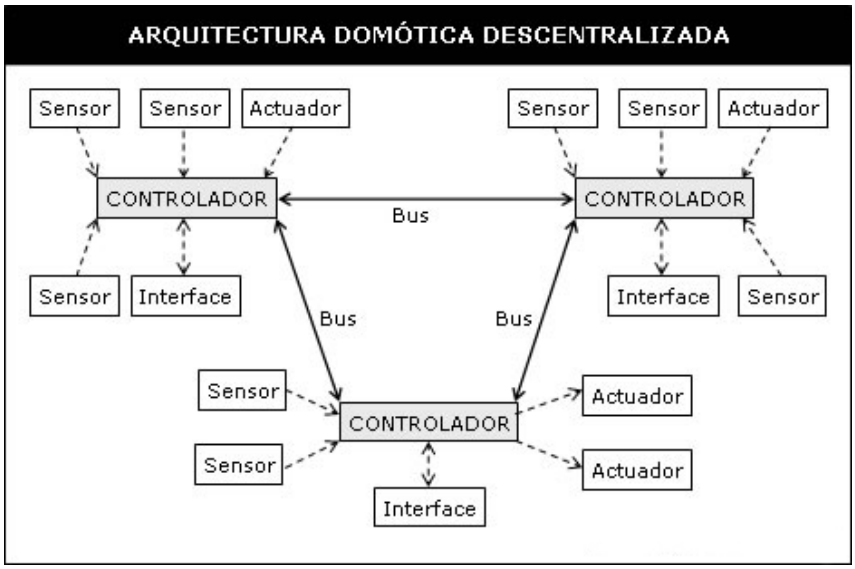

Figura 2.2: Ejemplo de arquitectura descentralizada.

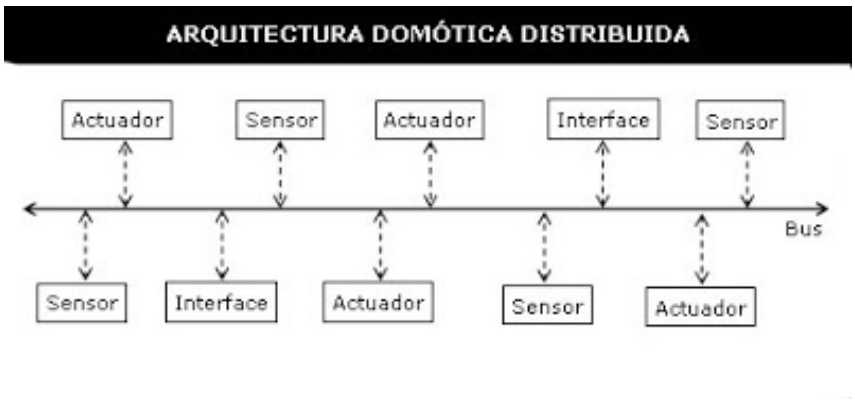

Figura 2.3: Ejemplo de arquitectura distribuida.

Sistemas Distribuidos

El caso opuesto sería emplear un arquitectura descentralizada, donde en vez de existir un único controlador, existen varios unidos por un bus de comunicaciones. (Ver Figura 2.2).

Sin embargo, dentro del mundo de la domótica y la inmótica, sin ninguna duda, prevalecen las arquitecturas distribuidas como las de la Figura 2.3.

En estas últimas arquitecturas eliminamos la necesidad de contar con uno (o varios) controladores, ya que cada elemento tiene implementado acceso al medio (al bus) y la suficiente inteligencia como para ser autónomo sin la necesidad de un "cerebro" central que lo comande.

Ejemplos de este tipo de arquitectura empezaremos a estudiar en sucesivos capítulos.

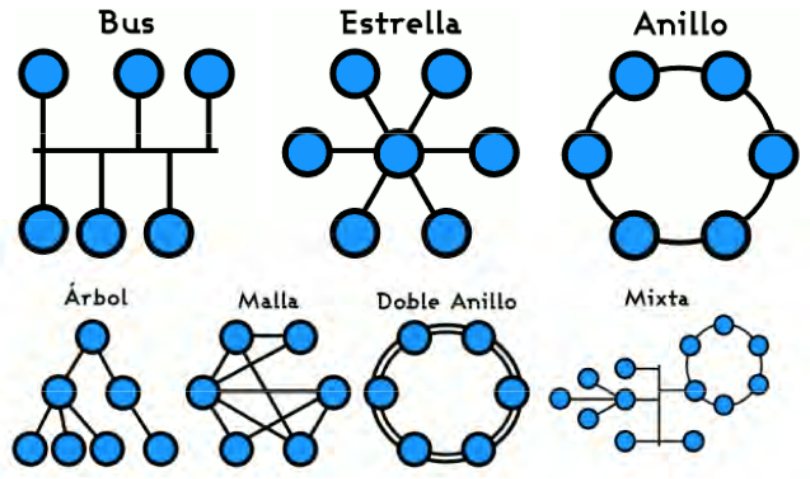

Figura 2.4: Diferentes topologías de una arquitectura distribuida.

2.1.3. Topologías

Dentro de los sistemas distribuidos debemos diferenciar también las diferentes topologías disponibles:

- **Bus**. Todos los nodos se conectan a un único canal de comunicación llamado bus.
- **Estrella**. Todos los elementos se conectan a un nodo central. Es la topología de un sistema centralizado.
- **Anillo**. Cada nodo tiene una conexión de entrada y otra de salida que lo comunica con el anterior y el siguiente.
- **Árbol**. Los nodos están conectados entre sí en forma jerárquica, formando un árbol.
- **Malla**. Cada nodo está conectado a todos los demas nodos.
- **Doble Anillo**. Igual que el anillo, pero con redundancia.
- **Mixta**. Combinación de algunas de las anteriores.

La figura 2.4 muestra de una forma gráfica muy descriptiva las diferentes topologías.

2.1.4. El papel de la domótica en nuestro proyecto

Como hemos visto, la domótica es la transposición de la automatización industrial al mundo doméstico, utilizándose el término inmótica cuando ésta transposición se lleva al sector terciario.

Parece por tanto más apropiado hablar de inmótica en nuestro uso de esta tecnología en un centro asistencial sanitario.

Concretamente, empleando inmótica pretendemos recoger la información de sensores situados en las camas y habitaciones para conocer el estado actual de los residentes y, en vez de actuar directamente sobre unos actuadores, nos marcamos el objetivo de mostrar esta información del estado del sistema a través de un panel de visualización para ayudar en las tareas asistenciales del personal del centro, así como la comunicación con un robot antropomorfo que ayuda en dichas tareas.

En cuanto a la arquitectura, y sabiendo por los requisitos del sistema que éste va a ser inalámbrico, nos vamos a centrar en un sistema centralizado donde todos los sensores se comuniquen con este nodo central, es decir, tendremos una topología en estrella.

2.2. Protocolos domóticos

Existen numerosos protocolos domóticos disponibles para tareas como la que nos proponemos para este proyecto, por lo que parece interesante mencionar en este punto dichas opciones y de éste modo conocer el estado del arte actual de la domótica / inmótica.

Para empezar, existen una serie de protocolos inicialmente concebidos para la comunicación alámbrica en los cuales no nos vamos a detener en exceso precisamente debido a esta circunstancia aunque a posteriori prácticamente la totalidad de estos protocolos se han adaptado para permitir una comunicación sin cables.

Dentro de estos protocolos, los más importantes y conocidos son:

- **KNX**: sistema descentralizado, es decir, no requiere de un controlador central de la instalación. En él, todos los dispositivos que se conectan al bus de comunicación de datos tienen su propio microprocesador y electrónica de acceso al medio.

  KNX está basado en el **modelo OSI**[2] y es el sucesor de tres tecnologías: el European Home Systems Protocol (EHS), BatiBUS, y el European Installation Bus (EIB or Instabus) y a día de hoy presume de ser "el único ESTÁNDAR abierto para todas las aplicaciones de control de la vivienda y el edificio, como por ejemplo el control de la iluminación y las persianas, así como variados sistemas de seguridad, calefacción, ventilación, aire acondicionado, monitorización, alarma, control de agua, gestión de energía, contador, así como electrodomésticos del hogar, audio/vídeo y mucho más" según la organización que lo gestiona, la **KNX Association**.

[2]Modelo de red descriptivo que fue creado por la ISO en el año 1980. Es un marco de referencia para la definición de arquitecturas en la interconexión de los sistemas de comunicaciones.

- **LonWorks**: estándar domótico que también se puede utilizar en la industria y en inmótica. La arquitectura del modelo de Lonworks fue definida en 1990 por **Echelon**, una compañía multinacional que se dedica al mercado de las redes de control, como es en este caso una red domótica.

  La filosofía que sigue esta compañía consiste en desarrollar el hardware (módulos OEM) que necesitan los fabricantes para integrar cualquier aparato dentro de una red Lonworks. En ocasiones también fabrica los chips y transceptores, pero otras compañías (Toshiba y Cypress) también los producen bajo licencia. Echelon también comercializa routers y kits de desarrollo.

  Cualquier dispositivo hardware de Lonworks se basa en un microcontrolador especial llamado **Neuron chip**. Tanto el modelo funcional del Neuron chip, como el protocolo LonTalk fueron definidos por Echelon en 1990. Actualmente, la norma ANSI/EIA 709.1-A-1999 recoge el protocolo LonTalk e IEEE-1473-1999 acepta Lonworks como estándar para la automatización del ferrocarril.

- **BACnet**: (Building Automation and Control Networks) es un protocolo norteamericano originado en 1987 creado para la automatización de las viviendas y redes de control de una forma centralizada. Fue patrocinado por la **asociación norteamericana de fabricantes e instaladores de equipos de calefacción y aire acondicionado** (ASHRAE) y en 1995 se convierte en un estándar ANSI e ISO (ISO 16484-5:2003).

  Su objetivo era realizar una gestión energética inteligente, con la finalidad de crear un protocolo abierto que permitiera interconectar los sistemas complejos como los de aire acondicionado y calefacción de las viviendas.

  Principalmente está pensado para el nivel de **supervisión**, aunque puede ser empleado también en el nivel de control. Representa la información en término de **objetos** a los cuáles se puede acceder a través de peticiones e iteraciones denominadas **servicios**. El protocolo incluye los servicios Who-Is, I-am, Who-Has y I-Have, utilizados para la detección de Objetos y Dispositivos. Otros servicios como Read-Property y Write-Property son usados para la lectura o escritura de datos.

- **Modbus**: protocolo de la capa de aplicación que proporciona comunicaciones maestro-esclavo entre recursos inteligentes. Fue desarrollado por Modicon (actualmente Schneider Automation) en 1979. Es una especificación abierta muy extendida en el mundo industrial debido a su simplicidad. Usado en dispositivos como PLC, HMI, drivers, sensores o actuadores remotos.

  Define una estructura de mensajes que puede ser reconocida por los diferentes dispositivos independientemente del tipo de red de comunicaciones utilizada. El protocolo describe el proceso para acceder a la información de un dispositivo, cómo debe responder éste y cómo se notifican las situaciones de error.

  Es soportado por redes industriales Modbus y por redes estándar.

- **X10** La tecnología X-10 de corrientes portadoras[3] fue desarrollada entre 1976 y 1978 por ingenieros en Pico Electronics Ltd, en Glenrothes, Escocia. Proviene de una familia de chips, que son los resultados de los proyectos X (la serie X). Esta empresa comenzó a desarrollar el proyecto con la idea de obtener un circuito que se pudiera implementar en un dispositivo para ser controlado remotamente.

  Este fue el primer módulo que podía controlar cualquier dispositivo a través de la línea de corriente doméstica (120 ó 220 V y 50 ó 60 Hz), modulando impulsos de **120 kHz**. (Ausencia de este impulso=0, presencia de este impulso=1). Con un protocolo sencillo de direccionamiento se podía identificar cualquier elemento de la red, en total 256 direcciones.

  El protocolo contemplaba 16 grupos de direcciones llamados **"House-codes"** y 16 direcciones individuales llamadas **"Unit-codes"**. A este protocolo se le añadieron "tiras" de comandos llamados "control strings" que no son mas que ceros y unos agrupados formando comandos preestablecidos por el propio protocolo.

  Estas señales las podían recibir todos los módulos, pero sólo actuaba sobre aquel al que iba dirigida (los primeros bits de la señal eran el identificador del módulo).

  La frecuencia de transmisión era la de la corriente eléctrica (50 ó 60 Hz.), y la señal completa incluyendo dirección y función ocupaba 48 bits, o sea que para mandar una señal a un dispositivo a una frecuencia de 50 Hz. (hablaríamos de un ancho de banda de 50 bits por segundo) se tardaría casi un segundo.

  Hoy en día X10 es un estándar y a la vez un fabricante de estos mismos productos y productos compatibles con X10 (alarmas, televisiones, contestadores, interfaces de ordenador, etc.). A pesar de que sólo tiene seis funciones, ha cubierto un hueco muy importante en el mercado y se ha consolidado como una buena y barata línea de productos, lo que ha acercado la domótica a presupuestos menores.

Por otra parte, existen protocolos que originariamente se establecen pensando en el aire como medio de transmisión. En este apartado hablaremos de algunas de las diferentes posibilidades existentes para implementar soluciones inalámbricas, siendo sus arquitecturas mucho más interesantes dentro del ámbito de nuestro proyecto.

Es importante señalar como cualquiera de las opciones inalámbricas van a tener que lidiar con el problema de la autonomía, por lo que su consumo es una aspecto crítico en el diseño. A cambio, este tipo de soluciones son mucho más flexibles precisamente debido a la ausencia de cables (y por tanto una instalación "de obra").

- **EnOcean** es una tecnología de comunicación inalámbrica de bajo consumo sin estándar internacional. Cuenta con la alianza EnOcean que mantiene el protocolo y

[3]Conocida en inglés como PLC - Power Line Carrier; caracterizada por emplear la red eléctrica como medio de transmisión.

garantiza la interoperabilidad.

Trabajo bajo una frecuencia de **868MHz** por lo que no existen problemas con la saturada banda de los 2.4GHz. La velocidad de transmisión es de 25 kbit/s, con modulación ASK[4] y su rango de cobertura está especificado en **300 metros** en condiciones ideales.

Una ventaja muy importante de esta tecnología es que permite que los sensores puedan trabajar sin activar la recepción radio constantemente, por lo que pueden **trabajar sin batería**, obteniendo la energía "del ambiente" o de la propia interacción con el usuario. Esto aporta dos cosas:

- Puede haber sensores con muy poco mantenimiento.
- El consumo que introducen en el sistema es cero.

- **Z-Wave**: tecnología de comunicación inalámbrica sin estandarizar. Sin embargo, existe la Alianza Z-Wave para garantizar la interoperabilidad de los dispositivos empleados. El estándar es **cerrado** y por tanto es necesario ser miembro para acceder a él. Aún así es fácil encontrar documentación de este protocolo.

  Trabaja en la banda de los **868MHz** evitando la gran cantidad de emisoras en la banda de los 2,4GHZ y puede llegar a trabajar a 40 kbit/s pudiendo operar en rangos de hasta **30 metros** en condiciones ideales.

  La topología de red es tipo **malla** y cada elemento se comporta como un nodo que puede ser receptor o emisor reenviando el mensaje. Permite realizar agrupaciones en grupos para asociar la misma funcionalidad a todos los elementos del grupo. Igualmente permite el **empleo de escenas**.

  Recientemente se ha evolucionado al conocido como **Z-Wave+** que mejora sus especificaciones (principalmente en consumo).

  Posiblemente sea el protocolo inalámbrico más conocido y empleado en la actualidad para dar cobertura en aplicaciones domóticas en general al tener una buena relación entre características / precio.

- **ZigBee**: es una tecnología de comunicación que basa sus niveles de enlace y radio en el **estándar IEEE 802.15.4** Para el resto de niveles no hay un estándar internacional, pero, de igual manera que EnOcean y Z-Wave, existe la ZigBee Alliance que se encarga de mantener la interoperabilidad en los niveles superiores al enlace.

  ZigBee es una tecnología ampliamente usada, tanto en entornos profesionales como de aficionados. La Alianza ZigBee (mayor que las de Z-Wave y EnOcean) cuenta con más de 200 miembros. Dado el volumen de ventas de dispositivos ZigBee, sus módulos pueden ser más baratos.

[4] Amplitude-shift keying - Modulación por desplazamiento de amplitud.

ZigBee es un conjunto de protocolos mucho más sofisticado que Z-Wave. Las especificaciones, mucho más largas, son públicas. Sin embargo, para obtenerlas hay que aceptar un compromiso de confidencialidad, pues no está permitido usar la información para fines comerciales (no se pueden vender productos si no estás en la Alianza). Sin embargo, sí que permiten el estudio de ZigBee para ámbitos universitarios o para realizar pruebas técnicas.

Creemos muy interesante conocer el estado del arte actual para, independientemente de si empleamos uno de los listados o no, tener una visión global sobre las diferentes aproximaciones tecnológicas que se están dando a problemas como los que pretendemos afrontar en este proyecto.

2.3. Arduino

Arduino es una **plataforma de hardware libre** que proporciona una placa de circuito impreso con un microcontrolador, usualmente Atmel AVR, puertos digitales y analógicos de entrada/salida y toda la electrónica necesaria para el funcionamiento de dicha placa (reguladores, limitadores de corriente, etc).

Arduino además es el nombre de la empresa encargada del desarrollo del producto. Es importante saber que bajo la nomenclatura de Arduino se esconden no una sino varias placas puesto que, como dijimos antes, Arduino no es la placa en sí, sino la plataforma que nos provee de diferentes modelos según las características que deseemos y que se traducen en diferentes voltajes de operación, más o menos entradas analógicas y/o digitales u otras características técnicas.

Posiblemente sea el proyecto de hardware libre más conocido mundialmente. Su historia comienza en 2006 y desde entonces la comunidad alrededor del proyecto ha crecido de forma exponencial convirtiéndose en una referencia siempre en mente a la hora de desarrollar electrónica en proyectos de prototipado rápido.

Se estima que existan **más de un millón de placas en manos de usuarios de todo el mundo** contando los diferentes modelos sacados al mercado. Esta situación hace que exista una comunidad alrededor del proyecto gigante que mejora el producto, asesora en cuanto a cuestiones técnicas (a través de los foros oficiales) y, sobre todo, ha desarrollado en estos últimos diez años una gran cantidad de librerías compatibles que hacen relativamente sencillo encontrar una que se asemeje a tus requerimientos.

Para nosotros, la elección de Arduino si tenemos que realizar algún diseño electrónico ha estado clara desde un primer momento debido a estas ventajas que aporta frente a otros proyectos semejantes. Especialmente relevante ha sido el hecho de que estemos trabajando

con protocolos de comunicación. La dificultad intrínseca al protocolo y las necesidades de crear un cliente capaz de operar con él se ven claramente minimizadas si se usan librerías de terceros o adaptaciones de las mismas que simplifican gran parte de esta complejidad por lo que el uso de Arduino nos dotará de la **agilidad y precisión** que buscamos en el proyecto, cumpliendo además los objetivos relativos al uso de hardware libre y de bajo coste.

2.4. Raspberry Pi

Raspberrry Pi es una **placa computadora (SBC) de bajo coste** desarrollada en Reino Unido por la Fundación Raspberry Pi con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas.

Presenta una gran aceptación mundial y se ha convertido en una de las herramientas favoritas dentro de la cultura Maker o DIY (Do It Youself – Háztelo tu mismo). Este éxito han llevado a la fundación a ir desarrollando modelos nuevos y evoluciones que mejoran las características iniciales de éste dispositivo presentado a principios de 2012.

Concretamente, para este proyecto, se pretende emplear el modelo Raspberry Pi 2 que cuenta con las siguientes características:

- Procesador Broadcom BCM2836 quad-core ARM Cortex-A7 a 900MHz.
- 1GB LPDDR2 SDRAM de memoria RAM.
- 4 puertos USB.
- 40 pines GPIO.
- Puerto HDMI.
- Salida de auido analógica y digital.
- Interfaz de cámara (CSI) y display (DSI) integrada.
- MicroSD.
- Puerto Ethernet.
- Módulo gráfico VideoCore IV 3D.
- Reducido Coste. Entre 35 y 40 euros.
- Capacidad para funcionar con distintas distribuciones ARM GNU / Linux y Windows 10.

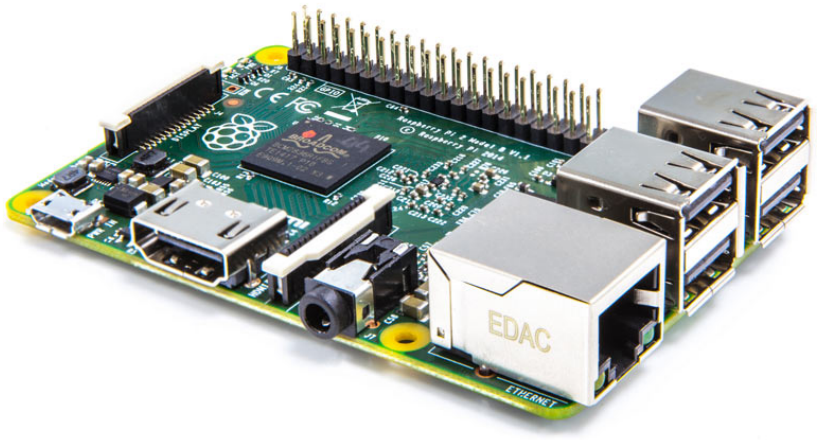

Figura 2.5: Raspberry Pi.

El diseño no incluye un disco duro o una unidad de estado sólido, ya que usa una tarjeta MicroSD para el almacenamiento permanente; tampoco incluye fuente de alimentación o carcasa. En la Figura 2.5 podemos ver el aspecto que presenta una placa Raspberry Pi 2.

Como hemos visto en sus características, la Raspberry Pi 2 posee una **conexión Ethernet 10/100** y, si bien es cierto que podría echarse en falta una conexión Wi-Fi, gracias a los cuatro puertos USB incluidos podríamos suplir dicha carencia con un adaptador Wi-Fi USB.

Es importante señalar que los puertos tienen una limitación de corriente, por lo que si queremos conectar discos duros u otro dispositivos tendremos que hacerlo a través de un hub USB con alimentación.

La Raspberry Pi se entrega sin sistema operativo; éste deberemos instalarlo sobre una tarjeta MicroSD que introduciremos en la ranura de la Raspberry Pi. Existen diferentes sistemas como Raspbian, Arch Linux, RaspBMC, Pidora, OpenELEC, Snappy Ubuntu Core o Windows 10 según nuestras necesidades.

El tamaño y su potencia, suficiente para las actividades que deseamos llevar a cabo, unido a su escaso **consumo de 3W**, así como la posibilidad de añadir interfaces externas que nos permitan comunicarnos con distintos protocolos inalámbricos a través de algunos de los puertos USB que cuenta la convierten en una excelente opción como nodo central de nuestro sistema domótico.

Si en el capítulo anterior hablábamos de Arduino como la referencia en el hardware libre para microcontroladores, Raspberry Pi es su análogo dentro del mundo de los microprocesadores, cumpliendo también el objetivo de bajo coste propuesto para este proyecto.

2.5. Domótica Libre

Desde hace unos años, e influenciados por el movimiento Maker, han surgido una serie de opciones libres, basadas en Linux, que buscan hacer de centro domótico de nuestra vida.

Estas opciones, intentan integrar bajo una misma interfaz diversos elementos que tiene cualquier instalación domótica con APIsAPI de Internet y gran parte de los dispositivos inteligentes y que no podríamos etiquetar como domótico sino entran mucho más de lleno en el conocido como **Internet de las Cosas** (IoT).

Para la realización de este proyecto, se pretende usar alguna de estas soluciones como eje del sistema, equipando al controlador central de un sistema Linux capaz de hacer funcionar la implementación elegida.

Para ello, parece interesante antes detenerse a evaluar las distintas opciones disponibles en el mercado para poder llevar a cabo una elección basándonos en criterios objetivos y es precisamente en este apartado donde vamos a hablar de ello.

2.5.1. Domoticz

Domoticz es un software libre de control domótico disponible para las plataformas Windows y Linux y caracterizado por **consumir muy pocos recursos del sistema** lo que lo convierten en una solución muy interesante si queremos combinarlo con un algún controlador Low Cost como puede ser una Raspberry Pi.

Domoticz ofrece soporte a un gran número de protocolos domóticos como pueden ser EnOcean, X10 o Z-Wave así como una buena integración con diferentes dispositivos inalámbricos como mini estaciones meteorológicas o cámaras IP.

Cuenta además con una interfaz web y aplicaciones móviles que convierten su consumo multiplataforma en una tarea sencilla.

Es uno de los primeros softwares de este tipo, aunque desgraciadamente y según nuestra opinión, la interfaz está algo desactualizada respecto las tendencias actuales y, lo que es más importante, los diferentes módulos que lo componen están poco desacoplados del "core" del sistema, lo que tiene la principal repercusión de un complicado desarrollo de nuevos módulos y funcionalidades.

Esta situación ha llevado a que el número de módulos y tecnologías disponibles, aunque amplio, sea algo menor que otras opciones.

El software está escrito en C++ y la documentación no resulta demasiado completa ni

atractiva lo que lleva a que gran parte de las dudas relacionadas con el software se acaben debatiendo e intentando solucionar en el foro de este proyecto libre (lo que siempre es una forma algo caótica de recolectar información).

2.5.2. Jeedom

Jeedom es un software francés mucho más moderno en cuanto a sus planteamientos y que se ha convertido en una solución relativamente conocida dentro del mercado doméstico europeo.

Este software es parte de una solución completa e integrada de la compañía que lo desarrollo y gracias a la cual podríamos comprar un dispositivo con el software instalado y listo para funcionar. Sin embargo, el software es libre y podemos crear nuestro propio sistema basándonos en él.

Aunque de una apariencia más moderna y con una arquitectura interna mucho más modular, su juventud hace que, según nuestra opinión, no estemos ante una solución suficientemente robusta como para implementarla en un sistema real.

Cuenta con el menor número de módulos entre las opciones que vamos a analizar y su documentación, que se encuentra en gran parte en francés (la versión traducida es bastante escasa), es bastante incompleta.

De nuevo es fácil tener que recurrir al foro oficial para consultar toda clase de dudas relacionas con su implementación, donde de nuevo nos volvemos a encontrar con la barrera del idioma.

Para finalizar, un elemento que me ha parecido diferente e interesante es el listado de los elementos disponibles dentro de una especie de "tienda" o "market" que convierten su instalación en una tarea mucho más visual y sencilla.

2.5.3. OpenHAB

OpenHAB presume de ser un **sistema domótico Open Source totalmente agnóstico en cuanto a tecnología se refiere**. Esto quiere decir que cubre una gran cantidad de sistemas: Windows, OSX, Linux, sistemas embebidos como NAS, etc.

Su diseño se basa en una **arquitectura totalmente modular** donde tenemos el core del sistema y sobre el que se construyen alrededor los diferentes módulos que le añaden funcionalidad.

Está escrito en Java y su modularidad ha permitido que cuente con muchos módulos

diferentes que cubren gran cantidad de elementos hardware y protocolos.

Por poner un par de ejemplos sencillos, con OpenHAB podríamos trabajar con relativa sencillez con KNX, ZWave, X10, el termostato Nest, bombillas HUE, bases de datos, servicios Web como IFTTT o una API que nos sirva el tiempo y la previsión.

Realmente en cuanto a funcionalidad creemos que es el sistema de los estudiados que más posibilidades nos ofrece. Además, modificar algún módulo para adaptarlo a nuestras necesidades es una tarea, que aunque compleja, es abordable lo que siempre le aporta un plus de configurabilidad.

El diseño visual, sin ser espectacular, es relativamente moderno y la documentación es la más completa de las consultadas, tanto del Core como de los módulos accesorios.

Cuenta con un diseño responsive que adapta la disposición de los elementos de los paneles de control al tamaño de la pantalla lo que facilitan su consumo y visualización multiplataforma. Además cuenta con aplicaciones oficiales para iOS y Android.

De cualquier modo, cuenta con una amplia comunidad que, una vez más se reúnen alrededor del foro oficial y que en muchos casos es la referencia oficial para solucionar los errores con los que nos encontremos y que resaltan las carencias de la documentación en algunos apartados.

Como puntos negativos, destacar que el proyecto es alemán y parte de la comunidad utiliza esta lengua para la resolución de dudas (lo que resulta excluyente para aquellos no-hablantes) y que, sin ninguna duda, estamos ante el proyecto más grande de los consultados, lo que si bien es cierto que aporta muchas ventajas, también tiene las desventajas de ser la opción más pesada en cuanto a consumo de recursos y espacio en disco así como resultar ciertamente compleja en algunos aspectos de su configuración.

Precisamente para su configuración cuenta con un entorno preparado que deriva de Eclipse (totalmente multiplataforma) y que busca facilitar la vida al instalador / configurador del sistema (aunque según mi opinión, no es una herramienta que esté demasiado depurada y presenta bastantes errores que pueden llegar a confundir más que a ayudar).

Por último, reseñar a modo informativo que actualmente la versión estable es la 1.X pero que se encuentra en beta una versión 2.X que, aunque aún en desarrollo, promete ser realmente una opción de futuro muy sólida ya que tanto a nivel de arquitectura interna como de diseño, está totalmente alineada con proyectos tan importantes dentro del panorama del "Internet de las Cosas" como la IoT Eclipse Fundation, siguiendo una serie de estándares que garantizan su interoperabilidad.

Con esta información, creemos que **la opción más estable y que más opciones y flexibilidad nos aporta es OpenHAB** por lo que éste será el software base sobre el

que se articulará el sistema desarrollado en este proyecto.

2.6. RFXCOM

RFXCOM es un controlador domótico que, conectado por USB a un ordenador y mediante un software compatible, **nos permite controlar de forma inalámbrica módulos domóticos RF 433 Mhz** basados en una gran variedad de protocolos, como por ejemplo X10, Chacon/D-IO, Lighwave, Lexibook, sensores Oregon y muchos más.

Su flexibilidad, ha facilitado que se convierta en poco tiempo en un elemento típico, casi estándar, en la comunicación inalámbrica con protocolos domóticos o elementos que reciben/envían información de forma pública en esta radiofrecuencia como pueden ser centrales meteorológicas.

Es muy común, encontrarnos referencias a él en diferentes sistemas libres que pretenden comunicarse con elementos propios mezclados con otros elementos propietarios que usan alguno de los múltiples protocolos con los que es compatible.

De hecho, es muy típico su empleo con algunos de los sistemas de domótica Open Source de los que hablamos en la sección 2.5. Dichos sistemas proveen de librerías y utilidades que facilitan la comunicación entre este elemento hardware y el sistema, lo que facilita enormemente el trabajo con un dispositivo RFXCOM.

Además, de la comunicación en sí, este dispositivo tiene otras interesantes funciones[12] que facilitan tanto la instalación en nuevos entornos como su posterior mantenimiento. Por ejemplo, el firmware es actualizable por lo que con tan sólo actualizarnos a nuevas versiones, obtendrías las ventajas que estas nuevas versiones aportes, mejorando también la estabilidad del sistema.

En la Figura 2.6 podemos ver la apariencia física externa del dispositivo RFXCOM.

El hecho de que se comunique mediante USB con el dispositivo host también le aporta una **gran flexibilidad** ya que permite su utilización en gran número de dispositivos independientemente del dispositivo y su arquitectura. Precisamente, esta ha sido una de las principales características que han llevado a su elección como elemento hardware de comunicación ya que podemos fácilmente conectarlo a una Raspberry Pi (recordamos las características especiales de la misma al poseer una arquitectura ARM).

Este dispositivo además aporta un **extra de seguridad y robustez** al ser fabricado por personal profesional que lleva desarrollando soluciones de este tipo desde el año 2000. Todas sus características han sido sometidos a intensas pruebas y posee el certificado CE.

Figura 2.6: Vista de RFXCOM.

Otro punto muy a tener en cuenta es la posibilidad de utilizar este dispositivo no sólo como receptor sino también como **transmisor**, ampliando así sus posibilidades de forma exponencial aunque debemos tener en cuenta que para nuestro cometido en este proyecto, no vamos a emplear estas características de transmisión.

También implementa CSMA-CA[5] para **evitar colisiones de paquetes de datos** lo que aporta a nuestras comunicaciones un extra de robustez y fiabilidad. Así, este multiplexado, permite al dispositivo "escuchar" diferentes dispositivos incluso si estos están emitiendo a la vez.

Por último, y también importante, RFXCOM es un dispositivo con un **consumo de energía realmente comedido** (0.14W) lo que ahonda en nuestro objetivo de eficiencia.

Todas estas interesantes características, son compatibles con un precio muy competitivo que se sitúa alrededor de los 100€ según proveedor por lo que su elección, en combinación con la Raspberry Pi, convierten al nodo central en una **solución barata y muy flexible**.

[5]CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) es un protocolo de control de acceso a redes de bajo nivel que permite que múltiples estaciones utilicen un mismo medio de transmisión.

Figura 2.7: Logotipo de ROS.

2.7. ROS

ROS (Robot operating System)[14] es un **framework flexible que persigue facilitar el desarrollo de software específico para robots**. A través de una colección de herramientas librarías y convenciones, simplifica notablemente la tarea de crear comportamientos robóticos complejos y robustos, soportando una amplia variedad de plataformas robóticas.

El proyecto, licenciado bajo una licencia libre **Creative Commons** intenta simplificar la complicada tarea de crear software específico para robots. Este es el motivo de su creación y desarrollo.

Además, aporta un grado de estandarización de forma que tareas cuya operativa variaba considerablemente entre diferentes plataformas tienen ahora un marco común y una interfaz única. De esta forma, el proyecto ha conseguido **democratizar el acceso a la robótica** ya que gracias a las abstracciones que provee, dota al programador de un entorno más sencillo para programar la lógica deseada.

El proyecto no oculta su intención de servir como "background" del desarrollo de software robótico colaborativo. De esta forma, animan a realizar cualquier aportación al mismo de forma que expertos de diferentes ámbitos pueden dotar de pequeñas utilidades que en su conjunto cubran un rango ámbito de aplicación. De hecho, este modelo colaborativo ha dejado su huella en la comunidad y no es raro encontrarnos con el logotipo de ROS (Ver figura 2.7) en numerosos proyectos de universidades y empresas privadas que bien emplean ROS y/o aportan nuevos módulos a la comunidad a través de sus investigaciones.

Nuestro caso concreto no aporta ningún añadido sobre ROS sino que hemos creído interesante dejar configurado nuestro sistema domótico para que pueda comunicarse con ROS. Concretamente, ROS será el **nexo en común entre nuestro sistema domótico y el robot antropomorfo** que se encontrará situado en el Centro Asistencial aportando diferentes servicios tanto a los residentes como a los empleados y la elección de este framework viene impuesta por los propios objetivos del proyecto al estar éste ya implementado en el robot antropomorfo con el que queremos interactuar.

Por tanto, nuestro desempeño en este trabajo respecto a ROS se limita a la interconexión de OpenHAB a ROS, pudiendo configurar diferentes reglas lógicas en OpenHAB que disparen acciones en uno u otro elemento de la comunicación. Por poner algunos ejemplos, esta interconexión nos permite relacionar alarmas con actuaciones concretas de un robot. Así, si detectáramos a través del sistema domótico que un residente desorientado sale de su habitación y se dirige hacia la puerta de salida del Centro, podemos, aparte de activar la correspondiente alarma en el centro de control, enviar al robot para acompañar al residente de nuevo a su habitación, o al menos bloquear la salida hasta que el personal humano llegue.

Capítulo 3

Diseño y selección de los sensores empleados

A lo largo de este capítulo, nos centraremos en los diferentes aspectos relativos al desarrollo electrónico del proyecto en sí. De este modo, expondremos los **elementos hardware seleccionados para conformar nuestro sistema**, entrando en diferentes aspectos técnicos como el diseño de la electrónica necesaria, selección y estudio del sistema de alimentación o las diferentes optimizaciones realizadas para mejorar el comportamiento del sistema.

3.1. Sensor de Presencia

Existe una amplia variedad de sensores de presencia en el mercado que nos permiten detectar si una estancia está o no ocupada. Como explicamos con anterioridad, éste es un dato que nos interesa conocer ya que lo consideramos importante si queremos mejorar la seguridad de los residentes.

El objetivo por tanto no es monitorizar en todo momento dónde se encuentran los residentes, sino más bien saber si una habitación está ocupada y, en especial, conocer cuando queda vacía para poder, si es necesario, alertar al personal del Centro Asistencial.

Como el lector puede observar, los objetivos de este tipo de control son relativamente modestos, pudiendo prescindir de sistemas complejos que aúnan cámaras de videovigilancia y visión artificial. Con sensores PIR (infrarrojo pasivo), mucho más simples y económicos, podemos cumplir nuestros objetivos.

Este tipo de sensor miden la luz infrarroja que los objetos de su campo de visión irradian por lo que no mide exclusivamente presencia humana. Sin embargo, en principio,

el centro no tendrá elementos o animales que puedan alterar el comportamiento de este tipo de sensores.

Como indicábamos en la introducción de esta memoria, queremos que el sistema sea lo menos invasivo posible, utilizando elementos inalámbricos siempre que sea posible. Además, contamos con un receptor RFXCOM en el dispositivo central que es capaz de recibir y decodificar comunicaciones en varios protocolos inalámbricos que trabajan en la banda de los 433MHz.

Es por tanto nuestro objetivo, encontrar algún dispositivo comercial en alguno de los protocolos soportados. Ya empezábamos esta sección comentando la amplia oferta de productos que cubren todas las imposiciones que hasta ahora hemos puesto, sin embargo, el abanico de posibilidades se reduce considerablemente cuando introducimos nuestras dos últimas limitaciones en cuanto a este sensor:

- Ángulo de Visión aceptable (mayor de 100º)
- Alcance moderado (mínimo 10 metros)

Aunque el alcance no es un parámetro demasiado complejo de cumplir (es muy fácil encontrar sensores con alcances superiores a 15 metros), **el ángulo de visión suele ser bastante reducido**.

Finalmente hemos seleccionado tres sensores diferentes:

- Home Easy HE851.
- Chacon PIR 2000.
- ActiveHome X10 MS13pr.

Todos ellos cumplen las necesidades del proyecto pero el segundo, de la marca Chacon, fue rápidamente eliminado como una opción viable debido al **escasa disponibilidad para comprar dicho producto en España**.

Finalmente, **hemos seleccionado el modelo de Home Easy por ser técnicamente semejante al de ActiveHome pero más económico**.

A continuación, en la tabla 3.1, indicamos sus principales características:

En la Figura 3.1, se puede apreciar su aspecto físico.

En cuanto a su instalación, sólo tenemos que dar dos pequeñas indicaciones:

Cuadro 3.1: Características Sensor de Presencia HomeEasy.

Alimentación	3.3V; 2 pilas AAA
Frecuencia	433.92MHz
Protocolo	HomeEasy
Canales	1
Tiempo de activación	5s - 10 min
Ángulo de visión	110º
Longitud de Detección	6m
Alcance	Hasta 50m
Dimensiones del producto	18,5 x 11 x 4 cm
Peso	100 gramos
Montaje	En pared

Figura 3.1: Home Easy HE851.

- Su colocación se hará en pared, en una posición con un buen ángulo de visión sobre la habitación y la puerta.
- El tiempo de activación corresponde al tiempo que transcurre entre que una persona es detectada (presencia a 1) y se desactiva (presencia a 0). Lógicamente, si sigue detectando a alguien, sigue enviando un valor 1. Es por tanto más bien un tiempo de desactivación y para que sea lo más rápido posible, nos interesa que este valor sea lo más bajo posible. Este modelo tiene un selector de tres posiciones, que colocamos en el valor inferior (5s).

3.2. Sensor de la Cama

Como ya adelantábamos al comienzo de esta memoria, el sensor de la cama será el elemento encargado de conocer el estado (ocupado o vacío) de la cama en todo momento para informar al elemento central donde está alojada la lógica del sistema y que éste pueda ejecutar las diferentes rutinas acorde a su programación.

Estamos por tanto ante un sensor digital donde diferenciamos dos partes principales:

- Lectura del estado de la cama.
- Transmisión de dicho estado.

Sin embargo, tras hacer un búsqueda exhaustiva de las diferentes opciones comerciales disponibles, hemos podido constatar la poca variedad de soluciones de este tipo que existen en la actualidad.

Además, dichas soluciones son caras y no se adaptan a las características que buscamos, principalmente debido a dos factores:

Por un lado, **sólo se encargan de la lectura del estado de la cama** y, como mucho, se podría comprar un dispositivo opcional que actúa como alarma, emitiendo una señal sonora al cambiar de estado.

Por otro lado, son sensores que normalmente van conectados directamente a la red eléctrica. En los objetivos del proyecto ya comentábamos la necesidad de que estos sensores fueran inalámbricos para evitar que la existencia de cables en la cama pudiera dar lugar a accidentes o averías por parte tanto del personal del centro como de los propios residentes.

Es cierto que algunos de estos sistemas se pueden completar con la instalación de baterías que vende el propio fabricante pero, desgraciadamente, estas son muy voluminosas y realmente caras.

Figura 3.2: Sensores de cama comerciales[4].

Esta serie de desventajas, podrían intentar ser corregidas adaptando estos sensores comerciales a nuestras necesidades aplicándoles una serie de modificaciones. Sin embargo la ausencia total de documentación y el empleo de protocolos de comunicación propios y cerrados hacen de esta labor una tarea prácticamente imposible.

De cualquier forma, creemos conveniente mostrar el aspecto que dichas soluciones presentan (ver figura 3.2) así como tener como referencia su precio que oscila entre los 60 y 100 euros.

Por tanto, y visto la incapacidad para emplear material comercial que cumpla los propósitos del proyecto, **diseñaremos una solución propia y específica capaz de cubrir nuestras necesidades, con un coste comedido y, aunque inalámbrica, capaz de tener una autonomía razonable**.

Estos objetivos requieren de un microcontrolador que centralice al adquisición de datos tomados desde un sensor (activo o pasivo) con la resolución suficiente como para discernir entre un estado de la cama ocupado o vacío. Además, este elemento central será el motor lógico encargado de procesar y enviar la información a través de un elemento transmisor.

A lo largo de los siguientes apartado veremos en detalle los diferentes elementos que conforman el sensor de cama que hemos diseñado.

Cuadro 3.2: Consumos diferentes Arduinos.

Placa	Consumo (mA)
Arduino Pro Mini	18.7
Arduino Micro	21.1
Arduino Nano	32.3

3.2.1. Arduino

Creemos que el mejor punto de partida es el núcleo central del sensor; el microcontrolador elegido. Éste será el elemento **encargado de procesar entradas y salidas acorde a un programa que le hayamos cargado**.

Como ya hemos hablado con anterioridad, para este proyecto, vamos a emplear una plataforma hardware de código abierto, basada en una sencilla placa con entradas y salidas, analógicas y digitales llamada Arduino.

Sin embargo, bajo la nomenclatura "Arduino", existen numerosas placas que difieren entre ellas en tamaño, forma, número de entradas y salidas o características eléctricas.

Las pretensiones de este proyecto hacen que el número de entradas y salidas digitales y analógicas no sea crítico. Nos resultan factores mucho más importantes a la hora de decantarnos por uno u otro **la eficiencia que consigue la placa al completo y que tenga un reducido tamaño**.

Empezando por este segundo criterio, entre todas las opciones disponibles hemos seleccionado las siguientes placas:

- Arduino Pro Mini
- Arduino Micro
- Arduino Nano

Una vez seleccionadas estas tres placas como referencia acorde a su tamaño, hemos pasado a comprobar su consumo energético. Seleccionaremos la que sea capaz de operar con el consumo más bajo.

En la tabla 3.2, podemos apreciar la corriente consumida por cada Arduino.

Como podemos ver, aunque no existe una gran diferencia entre el Arduino Pro Mini y el Arduino Micro, el primero consume 1.5 mA menos. Por tanto, una vez vistos estos valores, elegimos como modelo de Arduino a emplear al **Arduino Pro Mini**.

Esta placa, tiene las características técnicas[1] señaladas en la tabla 3.3.

Cuadro 3.3: Características Arduino Pro Mini.

Microcontrolador	ATmega328
Alimentación placa	3.35 - 12 V (modelo 3.3V) o 5 - 12 V (modelo 5V)
Voltaje de operación	3.3V or 5V (según modelo)
Pins digitales E/S	14
Pins PWM	6
UART	1
SPI	1
I2C	1
Entradas Analógicas	6
Interrupciones externas	2
Corriente DC por pin	40 mA
Memoria Flash	32KB de los cuales 2 KB son usados por el bootloader
SRAM	2 KB
EEPROM	1 KB
Velocidad de reloj	8 MHz (versión 3.3V) o 16 MHz (versión 5V)

Como podemos ver en la tabla, en numerosas ocasiones se hace referencia a la existencia de dos modelos diferenciados de 3.3V y 5V. Excepto las diferencias mostradas en la tabla, todas las características de ambas versiones son las mismas.

Debemos elegir entre una u otra versión. Teóricamente, el modelo de 3.3V, al estar alimentado a un voltaje inferior y operar internamente a una frecuencia y voltaje menor, debería tener un consumo más reducido. Sin embargo, como veremos en el siguiente apartado, existen elementos en la circuitería que conforma el resto del sensor que requieren tensiones de 5V.

Esto hace que, si nos decidiéramos por la versión de 3.3V, deberíamos diseñar un circuito que opere con dos niveles de tensión para poder alimentar los diferentes elementos del sensor.

Aunque en el siguiente apartado vamos a profundizar más en las razones, podemos adelantar ya que **hemos seleccionado la versión de 5V**, por lo que se puede concluir que el corazón del sensor lo conforma un Arduino Pro Mini[1] versión 5V.

3.2.2. Circuito diseñado

Una vez hemos elegido el microcontrolador, y sabiendo que cuenta con suficientes entradas y salidas (analógicas y digitales) para cumplir con los objetivos propuestos, se puede comenzar con el diseño el circuito en sí.

[1]Según la fuente consultada (incluida documentación oficial) podemos ver referenciada esta placa como Arduino Mini Pro o Arduino Pro Mini. Por consistencia a lo largo de esta memoria, nosotros emplearemos esta última.

Como ya comentábamos en el anterior apartado, el sensor (y por tanto su diseño) puede fácilmente dividirse en la lectura del estado de la cama por un lado y la transmisión hacia el elemento central por otro. El modo de tomar la lectura de la cama viene claramente determinado por el sensor seleccionado pero esta elección es independiente de la transmisión al realizarse ésta a través del transmisor, por lo que, inicialmente, empezaremos por esta última característica, hablando a continuación del sensor en sí.

Transmisión

La única comunicación de la que vamos a dotar al sensor es de transmisión. Por tanto, no será objeto de diseño ninguna característica de recepción de señal, tan sólo **emisión**. Una vez hecha esta aclaración, conviene también recordar que el nodo central, que será el receptor de la señal que enviemos desde el sensor que estamos diseñando, estará equipado con un dispositivo RFXCOM que es capaz de trabajar con varios protocolos de comunicación en la banda de los 433MHz.

Por tanto, nuestro sensor deberá emitir una trama con la información correspondiente a través de un protocolo que opere en dicha banda y sea compatible con RFXCOM. Por suerte, la cantidad de protocolos disponibles en esta frecuencia es amplio y tenemos opciones tan reconocidas como X10 o HomeEasy.

Sin embargo, no debemos perder de vista tampoco que estamos trabajando con un Arduino y la elección tanto del protocolo como del módulo emisor que seleccionemos estará también muy influenciado por esta circunstancia.

Con estos dos datos en mente y teniendo en cuenta que estamos buscando mantener unos costes bajos, rápidamente nos encontramos con un módulo de radiofrecuencia que trabaja en la banda de los 433 MHz, es barato y está especialmente diseñado (y probado) para ser montado junto con un Arduino. Estamos hablando del emisor diseñado por Seed Studio y fabricado por SparkFun cuya referencia es SEE-RF433.

Este módulo, con un precio inferior a los 3 euros, es muy empleado dentro de la comunidad Arduino lo que hace que existan **varias librerías disponibles para trabajar con él**. Su aspecto físico puede apreciarse en la Figura 3.3.

Su conexión física es muy sencilla ya que cuenta de 3 pines:

- **VCC**: de 5 a 12 V según potencia de transmisión deseada.
- **DATA**: conectado a un puerto digital de nuestro Arduino para generar la trama de datos.
- **GND**: conectado a la referencia de masa de nuestro circuito.

Figura 3.3: Receptor-transmisor de radiofrecuencia, banda 433MHz.

Para saber el voltaje de transmisión que precisábamos, realizamos varias pruebas y llegamos a la conclusión de que **el mínimo de 5V nos aportaba suficiente alcance**, cubriendo nuestras necesidades. Esta circunstancia nos permite ahorrarnos un elevador de tensión así como, lógicamente, permite al circuito tener un menor consumo.

Una vez seleccionado el módulo, nuestra primera aproximación fue emplear el protocolo X10 por ser este un protocolo relativamente común dentro de la domótica (y como se vio en los fundamentos teóricos, con una larga historia).

Para trabajar en conjunto con este módulo y protocolo, empleamos una librería Open Source llamada x10 y mantenida por Tom Igoe[15] que cuenta con cierto reconocimiento dentro de la comunidad Arduino.

Sin embargo, aunque su modo de empleo es bastante sencillo, su **potencia es bastante limitada** ya que la API es muy estricta y es fácil caer en puntos no cubiertos por la librería.

Precisamente, esto fue lo que nos ocurrió a nosotros ya que al intentar enviar datos adicionales (y perfectamente soportados por el protocolo) la API no nos ofrecía modo alguno de realizar dicha operación.

Además, y más preocupante, **el comportamiento era errático en cuanto activábamos modos de bajo consumo en nuestro Arduino**, recibiendo con mucha frecuencia datos corruptos en nuestro RFXCOM.

Estas dos circunstancias nos llevaron a probar varias librerías e incluso plantearnos sustituir el módulo de comunicación por uno que ofreciera más potencia de cómputo. Así hicimos tests con librerías como:

- homeeasyhacking[2]

[2]homeeasyhacking - https://github.com/bruce33/homeeasyhacking

- 433Utils[3]
- rc-switch[4]
- RemoteSensor[5]
- 433MHzForArduino[6]

Aunque ninguna de ellas cubría nuestras necesidades, las dos últimas (muy relacionadas entre sí) presentaban gran parte de las características que buscábamos y se presentaban como opciones robustas en nuestras pruebas.

Además, su documentación era no sólo de calidad en cuanto a la API disponible, sino que presentaban el trabajo de ingeniería inversa realizado por sus creadores así como información detallada sobre el protocolo bajo el cual operaban y que era 100 % compatible con nuestro RFXCOM.

Así, esta vía nos permitió **estudiar el protocolo en cuestión (CRESTA)**[5]. Dicho protocolo es realmente muy sencillo y está pensado para **enviar datos de temperatura, humedad y nivel de batería en una única trama**. Puede que al lector le llame la atención la poca flexibilidad en cuanto al tipo de envío (temperatura - humedad); esto se debe a que el protocolo CRESTA esta específicamente pensado y diseñado para enviar los datos de estaciones meteorológicas de bajo coste.

Sin embargo, ambos datos no dejan de ser números en formato "float" (coma flotante). Esta circunstancia ha sido determinante a la hora de tomar la decisión de, basándonos en estas dos librerías y aprovechando el protocolo CRESTA y la documentación que tenemos sobre él, crear **nuestra propia librería capaz de transmitir dos señales diferentes junto con el nivel de la batería**.

Así es como surge la siguiente librería:

```
139 void SensorTransmitter::sendTempHumi(int analogValue, bool digitalValue, bool batteryLow) {
140         byte buffer[10];
141 
142         // Note: analogValue is 10x the actual analogValue! So, 23.5 degrees is passed as 235.
143 
144         buffer[0] = 0x75;                   /* Header byte */
145         buffer[1] = (_channel << 5) | _randomId ;  /* Thermo-hygro at channel 1 (see table1)*/
146 
147         if (batteryLow == true) {
```

[3] 433Utils - https://github.com/ninjablocks/433Utils
[4] rc-switch - https://github.com/sui77/rc-switch
[5] RemoteSensor - https://github.com/hjgode/homewatch/tree/master/arduino/libraries/RemoteSensor
[6] 433MHzForArduino - https://bitbucket.org/fuzzillogic/433mhzforarduino/wiki/Home

```
148                 buffer[2] = 0x0e;                /* Package size byte for sensor */
149         }
150         else {
151                 buffer[2] = 0xce;                /* Package size byte for sensor */
152         }
153
154         if ( analogValue < 0 ) {
155                 buffer[5] = 0x4 << 4;         // High nibble is 0x4 for sub zero values...
156                 analogValue = -analogValue; // Make analogValue positive
157         } else {
158                 buffer[5] = 0xc << 4;                 // ...0xc for positive
159         }
160
161         // Note: analogValue is now always positive!
162         buffer[4] = (((analogValue % 100) / 10 ) << 4) |          // the "3" from 23.5
163                                 (analogValue % 10);         // the "5" from 23.5
164         buffer[5] |= (analogValue / 100); // the "2" from 23.5
165
166         buffer[6] = ((digitalValue / 10) << 4) | (digitalValue % 10); // BCD encoded
167
168         buffer[7]=0xff; /* Comfort flag */
169
170         sendPackage(_transmitterPin, buffer);
171 }
```

No vamos a entrar en los detalles de la implementación puesto que básicamente son adaptaciones que se han realizado a la última de las librerías mencionadas teniendo en cuenta cómo funciona el protocolo CRESTA.

Las razones de muchas de estas lineas se articulan en base al conocimiento por ingeniería inversa que se ha obtenido de cómo funciona el protocolo gracias al documento antes mencionado y disponible, tanto en los anexos (B) como en el CD que acompaña a esta memoria.

Si me gustaría detenerme brevemente en cómo se instancia el objeto "Sensor" (no olvidemos que las librerías en Arduino se programan en C++, un lenguaje de programación con orientación a objetos donde tenemos instancias del objeto y métodos disponibles para ejecutar rutinas de objetos instanciados) y cómo enviar información.

```
96                 SensorTransmitter(byte transmitterPin, byte randomId, byte channel);
```

Instanciamos el sensor pasándole los siguientes parámetros:

- **transmitterPin**: pin de Arduino que envía la información.

- **randomId**: un identificador único de cada sensor (nos permite un correcto direccionamiento).
- **channel**: canal de comunicación. En nuestro caso siempre será 1.

Este proceso, en nuestro código, se ejecuta antes del setup:

```
31 SensorTransmitter transmitter(tx, id, channel);
```

donde tx, id y channel con constantes fijadas anteriormente:

```
20 #define id 6 // Address of device (must be unique)
```

```
22 // Global settings
23 #define tx 7              // Pin number for the 433mhz transmitter
```

```
29 #define channel 1         // Transmitter channel
```

De este modo, tenemos disponible el método "sendTempHumi" cuya interfaz es como sigue:

```
104                 void sendTempHumi(int analogValue, bool digitalValue, bool batteryLow);
```

Como puede verse, hemos reservado un valor analógico entero y dos digitales (uno de ellos específico para enviar una alerta de batería baja) como información de la trama. Empleamos dicho método cada vez que queremos enviar información a nuestro RFXCOM.

A continuación, tenemos un ejemplo de uso real en el programa desarrollado:

```
75   transmitter.sendTempHumi(value * 10, false, batteryStatus < batteryLimit);
```

El único detalle que nos queda por nombrar es la necesidad de enviar el "analogValue" multiplicado por 10 ya que, internamente en el procesamiento de dicho valor y conversión a array de bytes, se efectúa una división entre 10 dentro del algoritmo de conversión que evita se introduzcan fallos en los **truncamientos internos de decimales**.

Estado de la Cama

La otra parte importante de la que consta nuestro sensor es aquella encargada precisamente de leer el estado de la cama. Esta tarea puede llevarse a cabo de varias formas aunque consideramos las más importantes:

- Reconocimiento óptico / visual.
- Reconocimiento mecánico.

La primera de las opciones, rápidamente la hemos descartado debido a los requisitos de potencia y consumo energético que requieren todas las operaciones de reconocimiento visual.

Por tanto nos centramos en **métodos mecánicos de detección**. Todos los estudiados se basan en resistencias variables cuyo valor cambia según alguna propiedad física relacionada directa o indirectamente con el peso que se pone encima.

De este modo, hemos valorado las siguientes opciones:

- Células de carga.
- Resistencias Variables por Flexión.

Las primeras, han sido empleadas en algunos estudios anteriores[16] obteniendo buenos resultados. Su funcionamiento es sencillo. Las células son alimentadas a 5V entre su cable VCC y su homólogo GND, obteniendo por un tercer cable DATA una **tensión proporcional al peso o carga que estén recibiendo**.

Estos sensores son los empleados en las básculas domésticas y es un sistema relativamente barato. En la Figura 3.4, podemos apreciar que forma física presentan dichos sensores.

Sin embargo, no se adapta especialmente bien a nuestras necesidades debido, principalmente a dos motivos:

Como podemos ver en diferentes estudios ya publicados[16], cuando estamos empleando sensores de este tipo en camas, es recomendable situarlos en las patas de la cama.

Figura 3.4: Células de carga.

Dicho estudio recomienda colocar una célula en cada pata para **equilibrar las cargas entre las patas y ganar precisión**.

Podríamos intentar colocar únicamente dos células, una en cada pata de forma que estén opuestas y en diagonal, colocando algún tipo de calzado en el resto de las patas para evitar perder estabilidad. Sin embargo, incluso aunque dicha opción fuese viable, por cada cama que queremos sensorizar, deberíamos contar con dos células, colocadas de forma separada y que de algún modo deben ser cableadas a un elemento común que evalúe el valor de ambas (en nuestro caso, nuestro Arduino).

Esta situación, indudablemente complica la instalación, obligando a la existencia de un cableado en la cama que no podemos despreciar.

Por otro lado, no son especialmente eficientes si nuestra intención es que estén activas siempre como en principio queremos que ocurra. Esto es debido a que la señal de salida es fruto de la caída de tensión producida dentro de la célula como respuesta a la variación interna de la resistencia que experimenta de forma proporcional al peso soportado. Al siempre estar soportando un peso debido a su localización debajo de las patas de la cama, constantemente está experimentando un consumo que si bien podría intentar optimizarse minimizando estas cargas constantes mediante muelles o algún otro elemento de amortiguación, no deja de ser una circunstancia que complica el diseño y construcción del sensor sin por ello obtener un beneficio claro.

Por tanto, ambas desventajas nos llevan a complicar de manera innecesaria la tarea de medición por lo que, al menos por el momento, son descartadas.

Si acudimos a estudios y trabajos anteriormente realizados por investigadores de todo el mundo, rápidamente nos damos cuenta que el otro método empleado para detectar el estado de una cama son los **sensores de flexión**.

Existen diferentes sensores acorde con la geometría a la que deben adaptarse tal y

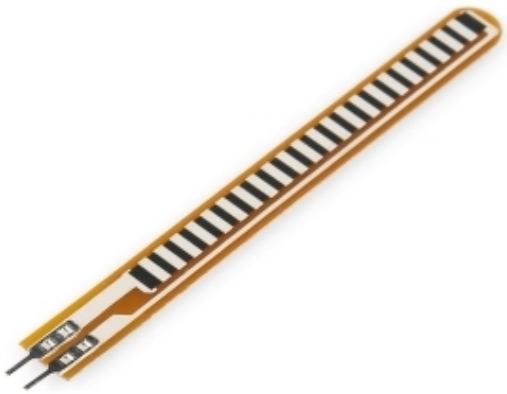

Figura 3.5: Sensor de Flexión alargado.

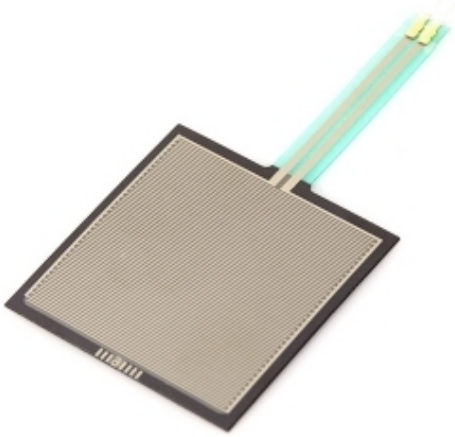

Figura 3.6: Sensor de Flexión cuadrado.

como se puede apreciar en las Figuras 3.5 y 3.6.

En nuestro caso, hemos seleccionado los sensores cuadrados ya que nosotros **buscamos cubrir una superficie** (dos dimensiones) y estos son más apropiados que los alargados, mejores opciones en el caso de querer medir una única dimensión.

Otros estudios[18] crean mallas con esta clase de sensores que después colocan encima de la cama como si de una manta se tratara. Los resultado de estos estudios constatan una gran precisión que lleva incluso a pensar en ellos para llevar a cabo tareas de **análisis de calidad del sueño**[20] pudiendo incluso servir de ayuda en la detección con éxito de patologías como la Apnea del sueño[21].

De hecho, creemos que esta clase de mallas son el fundamento bajo el cual se fabrican las soluciones comerciales que vimos al inicio de este apartado.

Sin embargo, nuestras necesidades no requieren de niveles de precisión tan elevados, por lo que, al menos inicialmente, vamos a comprobar si con un único sensor de este tipo, colocado de forma adecuada, podemos detectar **con precisión** si la cama está ocupada

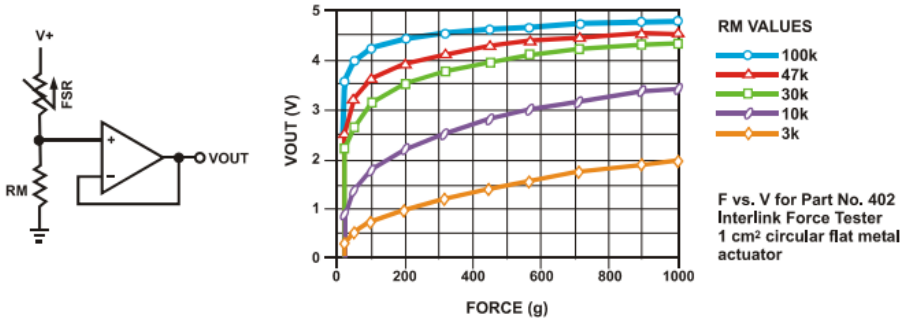

Figura 3.7: Interfaz recomendada.

o no por una persona.

El sensor que hemos elegido corresponde a un modelo fabricado y vendido por Sparkfun cuya hoja de características está disponible en los anexos de esta memoria. En dicho datasheet, a parte de dimensiones físicas, podemos apreciar el circuito recomendado de la Figura 3.7.

En dicha figura, vemos como se hace uso de un amplificador operacional. También podemos ver como el voltaje de salida aumenta a medida que aumenta la carga (medida en g). Este comportamiento indica que el valor de la resistencia es máximo cuando no existe carga y va decreciendo a medida que está más cargada.

Ya con el sensor en nuestras manos, podemos comprobar como el valor de la resistencia cuando no está cargada es muy elevado (mayor a los $20M\Omega$). Con este valor, **podemos considerar el circuito como abierto**. Esta situación nos viene muy bien puesto que esta será la situación en la que esté gran parte del día y en ella no existirá ningún tipo de consumo.

Por otro lado, situándola ya en la cama, encima de una de las lamas del somier, el valor de dicha resistencia varía entre los $125k\Omega$ y los $50k\Omega$ dependiendo de factores propios de la propia cama como puede ser el peso propio del colchón.

Otra característica que hay que tener en cuenta es que si no queremos usar un amplificador operacional u otro tipo de elemento auxiliar, las fluctuaciones en el valor de la resistencia respondiendo a las variaciones del peso experimentado por la cama también serán registradas puesto que nuestro estamos realizando mediciones analógicas de forma indirecta: al hacer pasar por ella una intensidad, nos provoca una caída de tensión (que medimos).

Este valor de tensión es analógico, pero en este caso **se desea que sea digital para poder trabajar con el como interrupción** (hablaremos de ello más adelante). Además, interesa conocer en exclusiva si hay o no una persona encima de la cama, lo cual claramente es una variable digital.

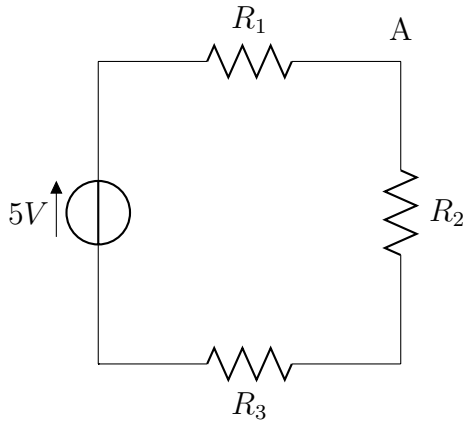

Figura 3.8: Representación sencilla del circuito de medición.

Dicho esto, debemos saber que si en un pin configurado como entrada digital en un Arduino aplicamos una tensión inferior a los 2V, éste tomará dicho valor como un 0 lógico. Por contra, si aplicamos un valor superior a los 2.7V, Arduino lo interpretará como un 1 lógico. El espacio entre ambos es una zona inestable de la cual que debemos huir ya que fácilmente puede inducir al error.

Estas características las emplearemos en el diseño realizado para, sin necesidad de convertir propiamente la señal de analógica a digital, poder usar dicha entrada como digital.

Con toda esta información, procedemos a diseñar el circuito en sí. Realmente, una vez explicadas todas las consideraciones anteriores, el proceso de diseño es sencillo y nos basaremos en **un simple divisor de tensión**.

La resistencia variable (sensor) estará conectada a los 5V y en serie con dos resistencias. Una de ellas, la primera, será un potenciómetro que nos permita ajustar o calibrar la caída de tensión producida en la resistencia variable, mientras que la segunda, será una simple resistencia de $1k\Omega$ que servirá de protección contra cortocircuitos en caso de que el valor del potenciómetro esté fijado a 0 y el sensor sufra un by-pass (o su resistencia sea 0). La resistencia de $1k\Omega$ estará conectada a GND, cerrando el circuito.

La medición la haremos entre la resistencia variable (sensor) y el potenciómetro e irá conectada a una entrada digital de nuestro Arduino (D2 en nuestro caso). Visto todo esto y sabiendo los valores con los que debemos trabajar para que Arduino funcione correctamente, tan sólo nos queda hacer los cálculos necesarios para obtener el valor del potenciómetro y que el sistema funcione en el punto que nosotros deseamos teniendo un **margen de calibración bueno**. Podemos rápidamente representar el circuito en la Figura 3.8.

En dicha figura, R1 representa nuestro resistencia variable (sensor), R2 el valor máximo

del potenciómetro y R3 la resistencia de protección de $1k\Omega$. El nodo marcado como "A" representa el punto de conexión eléctrica al Arduino y es el lugar donde debemos estar por debajo de 2V cuando la cama esta vacía y por encima de 3V si está llena.

El circuito es muy sencillo y los cálculos se llevan a cabo rápidamente aplicando la Ley de Ohm.

$$V = R \cdot I \rightarrow V = (R1 + R2 + R3) \cdot I \rightarrow I = \frac{V}{R1 + R2 + R3} \tag{3.1}$$

$$5 - A = R1 \cdot \frac{5}{R1 + R2 + R3} \tag{3.2}$$

En la ecuación 3.2, sabiendo que R1 toma valores entre $125k\Omega$ y $50k\Omega$, R3 = $1k\Omega$ y A cuando hay una persona debe ser al menos 3V:

Para $125k\Omega$ (caso más desfavorable):

$$R2 = 124k\Omega \tag{3.3}$$

Sin embargo, tras la experimentación práctica, hemos podido constatar como no es necesario emplear un potenciómetro de un valor tan elevado ya que aunque entre mayor sea su valor mejor trabaja bajo peso, también peor lo hace sin carga por lo que es necesario ajustar este valor experimentalmente hasta obtener un valor que cumpla un buen equilibrio entre ambas.

Hemos detectado que este valor son $75k\Omega$, estando un 90 % de las ocasiones configurado con un valor de $65 - 75k\Omega$. Por tanto, y también como compromiso con los resultados teóricos, creemos que logramos un buen equilibrio y flexibilidad con un **potenciómetro de** $100k\Omega$ **que inicialmente configuraremos a la mitad de su valor**.

Sólo nos queda por añadir que, la célula de medición como tal (R1) irá situada físicamente en un lugar diferente al resto del circuito, por lo que en éste, situaremos un **conector de dos pines** para conectarla sencillamente.

En cuanto al código necesario para hacer la lectura, se trata de una **simple lectura digital**:

```
24 #define sensor 2          // Bed sensor pin
```

```
73    int value = digitalRead(sensor);            // Bed State
```

Control de la Batería

Aunque más adelante hablaremos con más detenimiento sobre la batería encargada de alimentar el sistema, en este punto de diseño del circuito debemos indicar cómo ésta estará conectada al circuito a través de dos pines de conexión.

Una característica que hemos implementado en el circuito es la **capacidad de medir la carga de la batería**. Para ello, medimos el voltaje que nos entrega, sin embargo, como veremos más adelante, dichos voltajes pueden ser superiores a los 5V máximos que podemos leer con nuestro Arduino sin dañarlo. Por tanto, hemos recurrido a un divisor de tensión, con ambas resistencias iguales de tal forma que en su punto medio, la tensión medida será la mitad de la existente en bornes de la batería, estando ahora este valor acotado en todo momento por debajo de los 5V.

Dicha medición la llevaremos a una entrada analógica (A3) donde efectuaremos la lectura. Como veíamos en la subsección relativa a la transmisión, el parámetro que enviaremos no será la carga en sí de la batería sino un booleano indicando si la carga es baja o no. Este valor límite lo hemos obtenido mediante experimentación, y en el apartado específico sobre la batería veremos que valor toma y el porqué.

Optimizaciones Hardware

Nuestro sistema ya tiene todos los elementos necesarios para funcionar correctamente, leyendo y enviando la información que requerimos para cumplir los objetivos marcado por el proyecto. Sin embargo, existen varios puntos en el diseño electrónico que pueden ser mejorados.

En primer lugar, el circuito montado para llevar a cabo un control de la batería es muy ineficiente. Inicialmente comentábamos que ambas resistencias debían ser iguales, pero si atendemos de nuevo a la Ley de Ohmn podemos comprobar que valores de resistencia iguales pero pequeños hacen que circulen intensidades no despreciables que al final suponen consumos a tener en cuenta.

$$V = 2 \cdot R \cdot I \rightarrow I = \frac{V}{2 \cdot R} \tag{3.4}$$

Simplemente sustituyendo en la ecuación 3.4 $R = 1k\Omega$ (valor inicialmente seleccionado) y $R = 100k\Omega$ (valor actual) vemos la gran mejoría en cuanto a consumo.

Sin embargo, incluso con un valor de resistencia elevado, estamos introduciendo un consumo que aunque pequeño, al ser constante, repercute negativamente en la vida de la batería. Para evitar esta situación, hemos recurrido a añadir al circuito un **transistor NPN** (concretamente un BC338) que actúa como interruptor digital.

Así, hemos conectado el colector a 5V, la base a un pin digital del Arduino (D3) que activa o desactiva el interruptor, con una resistencia limitadora de $1k\Omega$, y el emisor al divisor de tensión. Así cuando queremos leer el valor de la batería (y sólo en ese momento) activamos el interruptor, poniendo el pin digital del Arduino en alto, y la corriente pasa a través del interruptor.

Esta operación, unida al aumento del valor de las resistencias hacen que tanto por tiempo como por valor, el consumo del circuito encargado de la medición de la carga de la batería sea despreciable.

Sin embargo, esta operación, lleva asociadas dos complicaciones que debemos tener en cuenta:

Por un lado, debemos ser conscientes que existe un **periodo de tiempo muy pequeño de estabilización del transistor** cuando lo activamos hasta que en el emisor tenemos de forma estable la misma tensión que en el colector. Esto nos obliga a introducir un delay mínimo (experimentalmente hemos comprobado que 100ms es un buen valor) hasta realizar la medición para dar tiempo suficiente a que esta estabilización se haya producido.

Por otro lado, estamos midiendo un valor analógico que nuestro Arduino convierte a un valor digital a través de un **conversor analógico-digital**. Estos elementos tampoco son instantáneos, teniendo un pequeño tiempo de estabilización que también hemos comprobado es inferior a 100ms. Deberemos introducir en nuestro código también este delay para dar tiempo a Arduino a procesar correctamente el valor analógico.

Transcurrido este tiempo, podemos desactivar de nuevo el interruptor, poniendo el pin D3 a 0, con lo que se corta el paso de corriente a través del transistor y este circuito deja de consumir.

Por último, aprovechando que hemos introducido un interruptor que puede entregar a su salida 5 o 0 Voltios, sabiendo que las mediciones de la batería las realizaremos justo antes de enviar la transmisión hacia el nodo central y aprovechando que la alimentación del módulo de transmisión es precisamente de 5V, podemos conectar dicho módulo a la salida del transistor, evitando así que este conectado y consumiendo permanentemente cuando esta operación no es necesaria.

Con esta operación nos ahorramos 0.3 mA extra.

Con todo lo comentado en este apartado, podemos mostrar el código que efectúa las diferentes tareas de las que hemos hablado.

En primer lugar, la configuración de los pines, tal y como hemos ido diciendo:

```
25 #define switchBattery 3   // Switch battery pin
26 #define analogBattery 3   // Read battery pin
27 #define batteryLimit 440  // Lower battery limit (check documentation)
```

A continuación podemos ver como se realiza la medición del estado de la batería:

```
64 int readBattery() {
65   digitalWrite(switchBattery, HIGH);          // ON transistor
66   delay(100);                    // Waiting until reading is stable
67   int battery = analogRead(analogBattery) ; // reading and mapping
68   delay(100);                    // Waiting for ADC
69   return battery;
70 }
```

Y por último, como se envía la señal a través del módulo transmisor:

```
72 void sendSignal() {
73   int value = digitalRead(sensor);            // Bed State
74   batteryStatus = readBattery();              // Update Battery Status
75   transmitter.sendTempHumi(value * 10, false, batteryStatus < batteryLimit);
76   digitalWrite(switchBattery, LOW);           // OFF transistor
77 }
```

3.2.3. Esquemático y Layout

Una vez realizadas todas estas consideraciones estamos en posición de poder plantear el esquemático del circuito electrónico que formará el sensor para la cama.

Para esta tarea, se ha empleado el software **Eagle** de la compañía CadSoft. Su elección se ha visto condicionado por dos aspectos fundamentalmente:

- Producto con licencia Freeware.

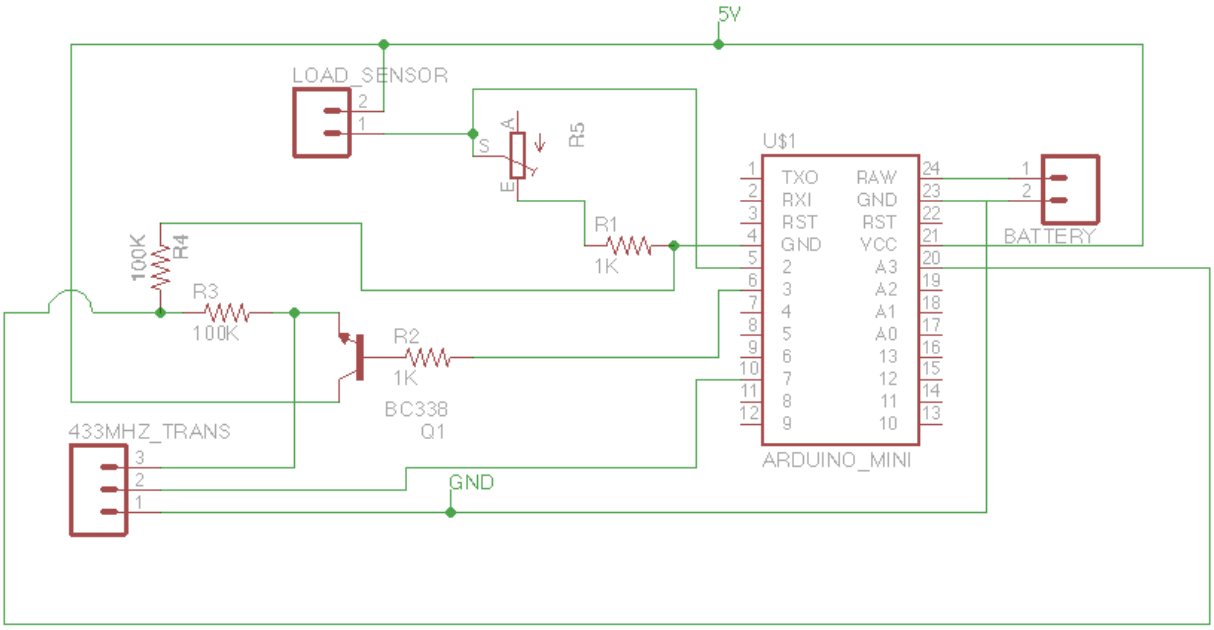

Figura 3.9: Esquemático del Sensor de la Cama.

- Gran adopción dentro la comunidad DIY lo que con el tiempo ha dado lugar a que existan gran número de librerías disponibles con las que agilizar el diseño tanto del esquemático como del Layout.

Además, con este mismo software podemos generar el esquemático y a partir de él crear el Layout, ayudándonos de algunas herramientas como el autoenrutado de las pistas.

En definitiva, no es una solución que pueda estar a la altura de soluciones más complejas como Orcad o Altium, pero para los requerimientos de esta placa, las ayudas que nos ofrece esta solución gratuita son suficientes para acabar de diseñar nuestra placa.

Por tanto, de este software se han obtenido dos ficheros con el esquemático (Figura 3.9) y el Layout final de la placa (Figura 3.10).

3.2.4. Optimizando el consumo: modo sleep

Aunque el sistema, con los elementos que hemos visto hasta ahora funciona correctamente, si medimos la intensidad consumida vemos como ésta alcanza los 19.9mA. Dicho valor es totalmente incompatible con nuestro objetivo de conseguir una autonomía superior a los seis meses, por lo que llegados a este punto es necesario optimizar todo el sistema para llegar a consumos mucho más reducidos.

El circuito en sí ya lo optimizamos en el anterior apartado, buscando emplear los elementos de los que disponíamos de la forma más eficiente posible. Así, por ejemplo, ya aprovechamos el transistor para controlar la alimentación del transmisor.

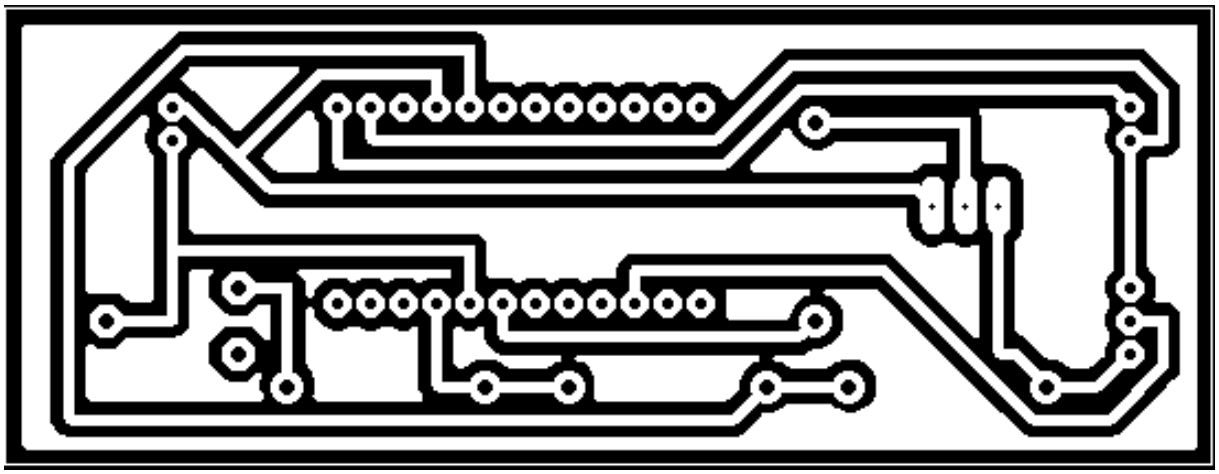

Figura 3.10: Layout del Sensor de la Cama.

Por tanto, vamos a centrarnos en este apartado en la optimización vía software del Arduino ya que tenemos margen de maniobra y es el elemento que mayor consumo introduce en el sistema.

La forma más habitual para reducir el consumo de un microcontrolador es trabajar con el modo “sleep"de éste. Dicho modo, reduce el consumo a costa de perder gran número de prestaciones.

En nuestro caso, estamos empleando Arduino, cuya API no consta de órdenes específicas para trabajar con el modo “sleep". Sin embargo, debemos siempre tener presente que debajo de Arduino tenemos disponible toda la API de Atmel AVR ya que estos son los microcontroladores que emplea.

En este sentido debemos conocer perfectamente la existencia de 6 modos de energía diferentes que desactivan más o menos características para conseguir consumos más reducidos[2]. De este modo, y de mayor a menor consumo, tenemos:

- idle
- adcNoiseReduction
- powerDown

Table 9-1. Active Clock Domains and Wake-up Sources in the Different Sleep Modes.

	Active Clock Domains					Oscillators		Wake-up Sources							
Sleep Mode	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	clk_{ASY}	Main Clock Source Enabled	Timer Oscillator Enabled	INT1, INT0 and Pin Change	TWI Address Match	Timer2	SPM/EEPROM Ready	ADC	WDT	Other I/O	Software BOD Disable
Idle			X	X	X	X	X(2)	X	X	X	X	X	X	X	
ADC Noise Reduction				X	X	X	X(2)	X(3)	X	X(2)	X	X	X		
Power-down								X(3)	X				X		X
Power-save					X		X(2)	X(3)	X	X			X		X
Standby(1)						X		X(3)	X				X		X
Extended Standby					X(2)	X	X(2)	X(3)	X	X			X		X

Notes: 1. Only recommended with external crystal or resonator selected as clock source.
2. If Timer/Counter2 is running in asynchronous mode.
3. For INT1 and INT0, only level interrupt.

Figura 3.11: Diferentes modos de bajo consumo de AVR.

- powerSave
- powerStandby
- standby
- powerExtStandby

En la Figura 3.11 podemos ver las características de cada uno de los modos.

Podemos trabajar directamente con estos modos, manejando de forma manual todas las características del sistema. Sin embargo, disponemos de una fantástica librería desarrollada por RocketStream[13] que realmente facilita mucho esta tarea, ofreciéndonos una sencilla API desde la que manejar el modo sleep de nuestro Arduino.

Concretamente, debemos especificar los siguientes valores:

- **SLEEP_xS**: Fija el tiempo que estará "dormido" usando para ello el Watchdog.
- **ADC_OFF**: Apaga los convertidores Analógico a Digital.
- **BOD_OFF**: Apaga el circuito de Brown Out Detection, que es un circuito que sirve como detector de niveles de tensión peligrosamente bajos que podrían incluso llegar dañar los circuitos.

Para nuestro caso de uso vamos a **apagar los convertidores Analógico-Digital y el circuito Brown Out Detection**. Este último, aunque introducimos un riesgo en

el circuito, realmente creemos que compensa la reducción de consumo obtenida con su desactivación.

Debemos tener en cuenta que estos elementos que apagamos en el modo sleep, están perfectamente disponibles al "despertar" nuestro Arduino.

En cuanto al parámetro SLEEP_xS hemos encontrado una limitación en la librería para su uso ya que esta función hace uso del Watchdog interno del microcontrolador lo cual **establece un tiempo máximo de 8 segundos**, correspondiente a SLEEP_8S.

Debido a nuestra necesidad de aumentar estos tiempos, hemos desarrollado una pequeña función vía código que recibe cualquier valor en segundos y pone el Arduino en modo sleep con los convertidores Analógico-Digital y el circuito BOD apagados durante dicho tiempo.

A continuación, podemos ver dicho fragmento de código:

```
44 void sleep(int sec) {
45   while (sec >= 8) {
46     LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF);
47     sec -= 8;
48   }
49   if (sec >= 4) {
50     LowPower.powerDown(SLEEP_4S, ADC_OFF, BOD_OFF);
51     sec -= 4;
52   }
53   if (sec >= 2) {
54     LowPower.powerDown(SLEEP_2S, ADC_OFF, BOD_OFF);
55     sec -= 2;
56   }
57   if (sec >= 1) {
58     LowPower.powerDown(SLEEP_1S, ADC_OFF, BOD_OFF);
59     sec -= 1;
60   }
61   sendSignal();
62 }
```

Realmente, dicho código se encarga de controlar el tiempo haciendo **breves comprobaciones sobre el tiempo transcurrido desde la última orden de entrada en modo sleep**, haciendo uso de intervalos de 8 segundos para tiempos superiores a éste valor, e intervalos más pequeños a medida que nos acercamos a cero.

El consumo no se ve comprometido por esta práctica, ya que transcurren milésimas de segundo entre que despierta, comprueba el tiempo transcurrido y se vuelve a dormir. Es más, el polímetro, ni siquiera aprecia un cambio en el consumo.

Además de este "despertar" programado, hacemos uso de las **interrupciones**[3] para, en el caso de producirse un cambio en el valor del estado de la cama, el Arduino se despierte y envíe la información antes de volverse a dormir.

Esta interrupción la fijamos en el setup inicial mediante la siguiente instrucción:

```
attachInterrupt(digitalPinToInterrupt(sensor), handler, CHANGE);
```

En dicha sentencia, podemos apreciar:

- digitalPinToInterrupt(sensor): función de Arduino que devuelve a que interrupción corresponde un determinado pin ya que no todos los pines de Arduino pueden ser empleados como interrupciones y no existe relación directa entre número de pin y número de interrupción.
- handler: función que se ejecutara cuando dicha interrupción sea detectada. Hablaremos de ella a continuación.
- CHANGE. Existen diferentes tipos de interrupción a saber:
  - LOW: la interrupción se dispara cuando el pin está en estado LOW.
  - CHANGE: cuando pase de HIGH a LOW o viceversa. (usada en nuestro caso al querer detectar cualquier cambio).
  - RISING: dispara en el flanco de subida (al pasar de LOW a HIGH).
  - FALLING: Dispara en el flanco de bajada (al pasar de HIGH a LOW).

Por último, cabe destacar como al dispararse una rutina de interrupción, ésta tiene **prioridad sobre cualquier otra tarea que esté ejecutando el microcontrolador**. La prioridad se manifiesta en lelvar a cabo una acción y hacerlo lo más rápido posible por lo que también se ignoran los sleep o los sleepMillis de la API de Arduino.

Esta situación nos lleva a una **dificultad a la hora de poder medir la carga actual de la batería** ya que, como veíamos en el apartado anterior, precisábamos de 0.1 segundos entre que activamos el transistor y éste se estabilizaba para poder medir la tensión existente y otros 0.1 segundos para dar tiempo al conversor Analógico-Digital a efectuar la operación de conversión.

Esta casuística no puede ser realizada durante la interrupción al no poder ejecutar ningún tiempo de espera. Sin embargo, visto que la carga de la batería la estamos enviando cada minuto, no es necesario enviarla también con cada interrupción.

De esta forma, la función handler, es exactamente igual que sendSignal pero esta vez **sin las operaciones asociadas a la lectura del estado de la batería**, tal y como vemos a continuación:

```
79 void handler() {
80   int value = digitalRead(sensor);            // Bed State
81   // Battery Status is not refreshed because we are inside an interruption
82   // and the delays that ADC needs does not work.
83   digitalWrite(switchBattery, HIGH);
84   transmitter.sendTempHumi(value * 10, false, batteryStatus < batteryLimit);
85   digitalWrite(switchBattery, LOW);           // OFF transistor
86 }
```

3.2.5. Código Arduino

Con todo ello, el programa que tiene cargado el Arduino, de forma completa, sería tal y como sigue a continuación:

```
1  /*****************************************************************************
2   * TITLE: BedSensor.                                                         *
3   *                                                                           *
4   * DESCRIPTION: This Arduino sketch uses Low Power Mode and Interruptions to *
5   * check and send the state of the bed thanks to a force resistive sensor.   *
6   * This sketch uses LowPower library.                                        *
7   * Their use is parametizable                                                *
8   *                                                                           *
9   * LICENSE: GNU GPL v3.                                                      *
10  *                                                                           *
11  * VERSION: 1.0.                                                             *
12  *                                                                           *
13  * AUTHOR: Ricardo Vega Alonso<ricardo.vega@ricveal.com>                     *
14  *****************************************************************************/
15
16 #include <LowPower.h>
17 #include <SensorTransmitter.h>
18
19 // Specific settings
20 #define id 6 // Address of device (must be unique)
21
22 // Global settings
23 #define tx 7              // Pin number for the 433mhz transmitter
24 #define sensor 2          // Bed sensor pin
25 #define switchBattery 3   // Switch battery pin
```

```
26 #define analogBattery 3   // Read battery pin
27 #define batteryLimit 440  // Lower battery limit (check documentation)
28 #define maxTime 60        // Max time in seconds between interruptions
29 #define channel 1         // Transmitter channel
30
31 SensorTransmitter transmitter(tx, id, channel);
32 int batteryStatus = batteryLimit;  // Store the battery status
33
34 void setup() {
35   pinMode(sensor, INPUT);
36   pinMode(switchBattery, OUTPUT);
37   attachInterrupt(digitalPinToInterrupt(sensor), handler, CHANGE);
38 }
39
40 void loop() {
41   sleep(maxTime);                    // Sleep time
42 }
43
44 void sleep(int sec) {
45   while (sec >= 8) {
46     LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF);
47     sec -= 8;
48   }
49   if (sec >= 4) {
50     LowPower.powerDown(SLEEP_4S, ADC_OFF, BOD_OFF);
51     sec -= 4;
52   }
53   if (sec >= 2) {
54     LowPower.powerDown(SLEEP_2S, ADC_OFF, BOD_OFF);
55     sec -= 2;
56   }
57   if (sec >= 1) {
58     LowPower.powerDown(SLEEP_1S, ADC_OFF, BOD_OFF);
59     sec -= 1;
60   }
61   sendSignal();
62 }
63
64 int readBattery() {
65   digitalWrite(switchBattery, HIGH);          // ON transistor
66   delay(100);                    // Waiting until reading is stable
67   int battery = analogRead(analogBattery) ; // reading and mapping
68   delay(100);                    // Waiting for ADC
69   return battery;
70 }
71
72 void sendSignal() {
73   int value = digitalRead(sensor);            // Bed State
74   batteryStatus = readBattery();              // Update Battery Status
```

```
75    transmitter.sendTempHumi(value * 10, false, batteryStatus < batteryLimit);
76    digitalWrite(switchBattery, LOW);           // OFF transistor
77  }
78
79  void handler() {
80    int value = digitalRead(sensor);            // Bed State
81    // Battery Status is not refreshed because we are inside an interruption
82    // and the delays that ADC needs does not work.
83    digitalWrite(switchBattery, HIGH);
84    transmitter.sendTempHumi(value * 10, false, batteryStatus < batteryLimit);
85    digitalWrite(switchBattery, LOW);           // OFF transistor
86  }
```

3.2.6. Tamaño y duración de la batería

Como varias veces se ha indicado a lo largo de este proyecto, la autonomía del sistema es un **requerimiento crítico** puesto que incide de forma notable en la usabilidad y mantenimiento del mismo.

En este sentido, nos hemos marcado como objetivo una autonomía **superior a seis meses** que permita al sistema trabajar al sistema de forma prácticamente automática en este periodo de tiempo.

La consecución de este objetivo se ha realizado de forma conjunta actuando en el tamaño de la batería así como en la capacidad del sistema de baterías seleccionado.

En cuanto al primer punto, y como vimos en el apartado anterior, hemos logrado reducir el consumo energético a 0.45 mA mientras el sensor está en modo sleep y asciende a 19,3 mA cuando está activo.

Sin embargo, los tiempos en uno y otro modo hacen que los tiempos en los que el sensor está activo sean despreciables. Esto es debido a que el sensor está permanentemente en modo sleep excepto en dos momentos concretos:

- Cuando detecta un cambio en el estado de la cama (estos cambios deberían producirse menos de diez veces por día).
- Cada X tiempo establecido en la configuración del sensor que hace una comprobación programada del estado del sensor.

Cualquiera de estas circunstancias hace que el sensor salga del modo sleep y entre en una rutina de recogida y envío de datos que dura aproximadamente 0.25 segundos.

Esto hace que por cada 60 segundos "dormido" y consumiendo 0.45 mA pasa 0.25 segundos consumiendo 19.3 mA lo que el modo "activo" represente menos del 15 % de la energía consumida por el sensor.

De cualquier modo vamos a ser completamente rigurosos en nuestros cálculos y no vamos a estimar este valor. De este modo, para calcular el consumo del sistema:

$$\frac{(19{,}3\frac{mA}{s} \cdot 0{,}25s + 0{,}45\frac{mA}{s} \cdot 60s)}{60{,}25s} = 0{,}528mA \tag{3.5}$$

$$6meses \cdot 30\frac{días}{mes} \cdot 24\frac{horas}{día} \cdot 0{,}528mA = 2280mAh \tag{3.6}$$

Por tanto, la capacidad del sistema de baterías que seleccionemos debe ser superior a estos 2280mAh que hemos obtenido. Por otra parte, debemos obtener un voltaje superior a 5V pero no nos conviene que se supere este valor por mucho ya que el regulador lo convertirá en pérdidas innecesarias.

El voltaje de la batería está relacionado con la reacción electroquímica de la batería y por tanto es característica de la composición química de la misma. Además, poniendo en serie varias baterías sumamos sus voltajes.

Sabiendo estas dos características, creemos que las pilas AA o AAA son las mejores opciones que tenemos disponibles para nuestro sistema. Son baratas y devuelven 1.5V, por lo que cuatro en serie nos entregan un voltaje de 6V.

Descartamos las pilas AAA puesto que su capacidad típica (900-1155 mAh.) es inferior a nuestros requerimientos. Nos quedamos por tanto con cuatro pilas AA.

Ahora vamos a testear un modelo concreto que destaca por su disponibilidad y muy reducido precio. Estamos hablando de las **pilas AA ALKALISK de Cadmio-Mercurio** que comercializa IKEA. Sin embargo, tenemos dudas sobre la capacidad real de dichas pilas por lo que las hemos ensayado.

Para ello, con un equipo de testeo adecuado que nos asegura una corriente de descarga constante, que en nuestro caso hemos fijado en 160mA, hemos ido midiendo cada 10 minutos el voltaje disponible en bornes. El test lo hemos realizado con un pack de 4 pilas en serie tal y como pretendemos emplearlo como fuente de alimentación para nuestro sensor.

El resultado de este test puede apreciarse en la Figura 3.12.

Como puede observarse, hemos detenido la descargar tras 900 horas de funcionamiento. Hemos tabulado en la tabla 3.4 los datos que consideramos más interesantes.

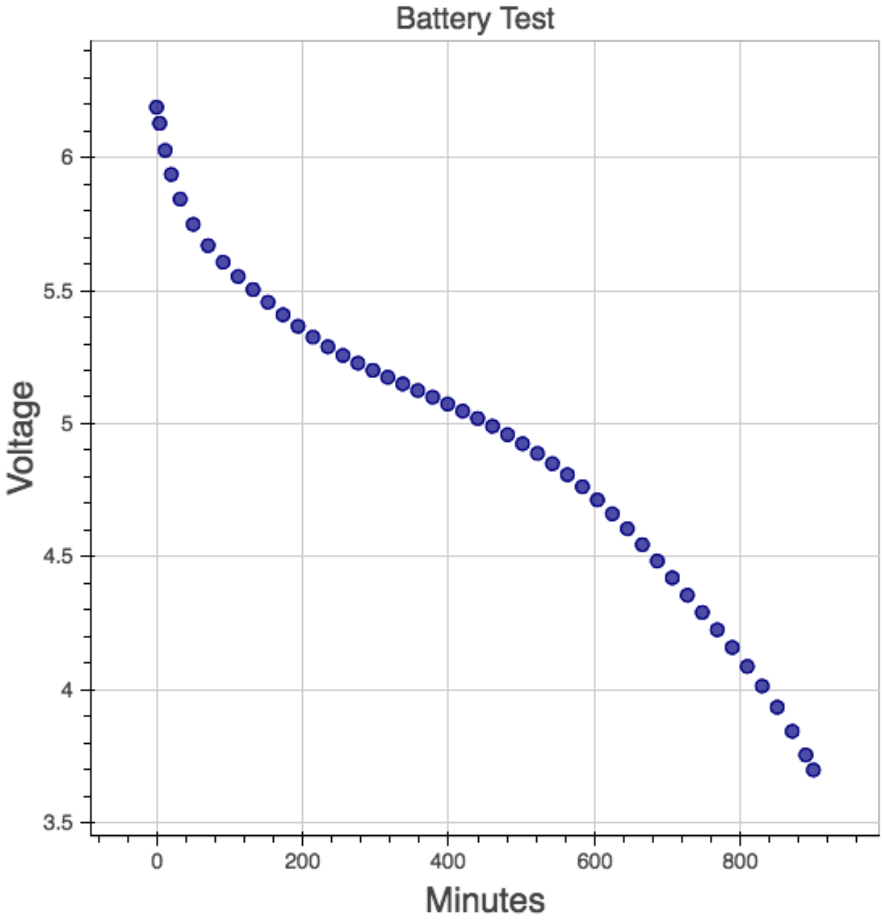

Figura 3.12: Descarga de baterías usadas.

Cuadro 3.4: Resultados Test Batería

Voltaje inicial	6.35 V
Voltaje medio durante la descarga	4.93 V
Capacidad medida	2540 mAh

Cuadro 3.5: Listado de Componentes (con precio)

Componente	Precio Unitario (€)	Unidades	Importe (€)
Emisor RF 433MHz	2.75	1	2.75
Tira pines macho paso 2,54mm	0.26	1	0.26
Arduino Pro Mini 5V 16MHz	16.46	1	16.46
Sensor Fuerza Cuadrado	9.73	1	9.73
Placa fotosensenible positiva	1.08	1	1.08
Conector Hembra	0.11	1	0.11
Porta pilas 4 AA	0.90	1	0.90
Res. Ajustable horizontal 100k ohmnios	0.45	1	0.45
Transistor BC338	0.13	1	0.13
Resistencia 1k Ohmnio	0.04	2	0.08
Resistencia 100k Omhnio	0.18	2	0.36
Pila ALKALISK AA	0.20	4	0.80
		TOTAL	33.12

Como vemos, obtenemos una **capacidad superior a la requerida**. Concretamente, con el empleo de estas pilas, conseguimos una autonomía de:

$$\frac{2540mAh}{0{,}528mA} \cdot \frac{1día}{24horas} \cdot \frac{1mes}{30días} = 6{,}68meses \tag{3.7}$$

Queremos también aprovechar el estudio que hemos hecho de la descarga del conjunto de pilas para establecer una **alerta de "Batería Baja"** que avise al operario sobre la necesidad de cambiar las pilas. Creemos que un punto interesante podría ser aproximadamente cuando se haya consumido un 85 % de la capacidad total disponible.

Fijándonos en la figura 3.12 vemos que este valor corresponde aproximadamente con un **voltaje de 4.4V**. Hemos comprobado que a este voltaje e incluso inferiores, el sistema sigue funcionando de forma estable. Por lo que apuntamos este valor como el parámetro "batteryLimit" que habíamos visto en el código de Arduino.

3.2.7. Componentes

En este punto final, podemos ya recopilar todo lo visto en esta sección y ofrecer en la tabla 3.5 un listado de los componentes que conforman el sensor diseñado.

Aunque faltan por añadir algunos costes indirectos, ya podemos apreciar como el precio es muy inferior al de los productos comerciales que habíamos visto. Además, cabe recordar que este diseño es específico para este proyecto, cumpliendo con todos los objetivos que nos marcábamos, mientras que por contra, los sensores comerciales habíamos visto que presentaban una serie de problemas a la hora de emplearlos en nuestro sistema.

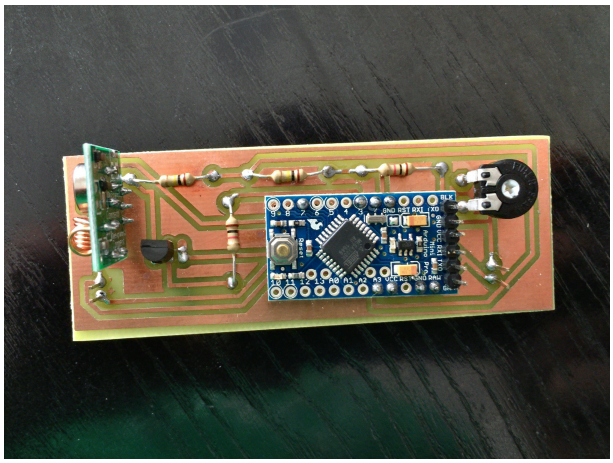

Figura 3.13: Aspecto final del sensor desde una perspectiva aérea.

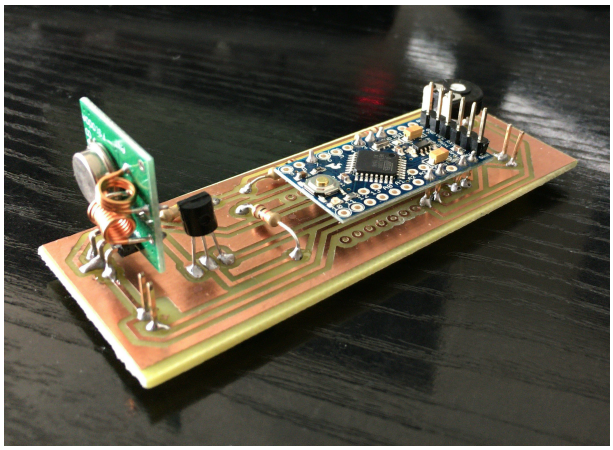

Figura 3.14: Aspecto final del sensor en perspectiva.

La PCB la hemos generado mediante revelado litográfico y los componentes se han soldado en ella de manera manual.

En las Figuras 3.13 y 3.14 podemos apreciar el aspecto final que presenta la placa:

Capítulo 4

Configuración del nodo central y otras consideraciones técnicas

En este capítulo, también relativo al desarrollo técnico de la memoria, nos centraremos más en el elemento central de la instalación (Raspberry Pi 2) y todas las configuraciones que hemos tenido que realizar al respecto para asegurar un **funcionamiento estable y eficiente de la instalación**.

También dedicaremos una sección a hablar del entorno de desarrollo que hemos empleado para el desarrollo de este proyecto y unas pequeñas consideraciones técnicas finales de carácter general.

4.1. Raspberry Pi

Como elemento central del sistema hemos seleccionado una **Raspberry Pi 2**. Tal y como ya explicamos en los primeros apartados teóricos de esta memoria, sus características técnicas, aunque modestas, son más que suficientes para la mayoría de aplicaciones y nuestro caso no es una excepción.

Será la encargada de ejecutar OpenHAB y tendrá conectada por USB un RFXCOM.

Creemos que no merece la pena volver a recordar aquí sus principales características ya que están expuestas en el apartado correspondiente de los fundamentos teóricos. Por contra, entedemos que en esta sección debemos centrarnos en la configuración de la misma.

Existen diferentes modos de llevar a cabo la preparación-instalación de una Raspberry Pi por lo que la opción seleccionada no es única.

En primer lugar, tenemos que **cargar un sistema operativo**. Debido a nuestra

familiaridad con sistemas operativos basados en Debian, hemos elegido la distribución Raspbian (que cuenta con el añadido de ser la opción que recomienda la Raspberry Foundation).

El proceso de instalación es bastante sencillo. Tenemos que descargar la imagen (archivo IMG) de la página oficial. Una vez descargada, deberemos transferir la imagen a una tarjeta MicroSD que deberemos conectar a nuestra Raspberry.

Para ello, existen diferentes opciones según el Sistema Operativo que se esté, empleando. Para Windows recomendamos emplear Win32DiskImager mientras que para Linux/Unix (nuestro caso) la opción que recomendamos es emplear la herramienta por linea de comando "flash" que aunque no es oficial, está desarrollada por una importante comunidad dentro del desarrollo de Raspberry como es "hypriot".

Una vez creada la SD, la conectamos a la Raspberry Pi y arrancamos el dispositivo para llevar a cabo una sencilla configuración totalmente guiada que no requiere ningún tipo de mención especial por nuestra parte excepto que, por motivos de eficiencia y visto que vamos a usar la Raspberry exclusivamente en **modo no gráfico**, hemos bajado la memoria GPU reservada a 16Mb de modo que tenemos más memoria disponible para los procesos del sistema.

4.1.1. Actualizando el Sistema

Es considerado una buena práctica actualizar el sistema operativo a la última versión disponible tan pronto como sea posible. Para ello, cuando tengamos conexión a Interent, ya sea vía WiFi o mediante cable, ejecutaremos los siguientes comandos:

```
1 sudo apt-get update
2 sudo apt-get upgrade
```

4.1.2. Instalando Openhab

A continuación procedemos a instalar OpenHAB.

El proceso entero se basa en descargar el entorno, descomprimirlo, cargar la configuración inicial (de la que hablaremos en el siguiente punto) y dar los permisos adecuados.

Por sencillez, **hemos creado un script que realiza todas estas tareas de forma automática** para, como hemos expresado en numerosas ocasiones, ganar en agilidad y rapidez.

Dicho script se encuentra disponible en el CD que acompaña a esta memoria de forma aislada, pero nuestra idea ha sido aprovechar la potencia del control de versiones para agilizar el proceso de instalación y configuración.

De cualquier modo, reproducimos a continuación dicho script:

```
sudo wget https://bintray.com/artifact/download/openhab/bin/distribution-1.8.1-runtime.zip
sudo unzip distribution-1.8.1-runtime.zip
sudo rm distribution-1.8.1-runtime.zip
sudo rm -r LICENSE.TXT README.TXT addons configurations webapps
sudo git checkout linux
sudo chown -hR pi:pi /opt/openhab
chmod +x start.sh
chmod +x start_debug.sh
sudo mv autostart /etc/init.d/openhab
sudo chmod a+x /etc/init.d/openhab
sudo update-rc.d openhab defaults
```

Si está acostumbrado a trabajar con entornos Unix, seguramente todos los comandos ejecutado le serán familiares. Sin embargo, puede llamarle la atención la sentencia "sudo git checkout linux", donde estamos haciendo un cambio de rama en un repositorio git.

4.1.3. Usando GIT como CVS

CVS son las abreviaturas en inglés de **Software para el Control de Versiones**. Aunque existen distintos CVS disponibles en el mercado, creemos que el más avanzado y el que mejor se adapta a nuestros requerimientos es Git.

Por qué usar un CVS

Puede que el lector se haga esta pregunta. Al usar software de estas características, estamos automáticamente habilitando la posibilidad de alamacenar todas las diferentes versiones por las que pasa un fichero a lo largo de su vida.

Esta característica es muy práctica durante el desarrollo del proyecto ya que nos permite recuperar de una forma muy sencilla el contenido de un cierto fichero en versiones anteriores a la actual, ya sea de forma parcial o total.

Además, aunque en el caso de GIT no sea estrictamente necesario, es muy habitual que exista un servidor que almacene toda la información para funcionar con el CVS.

Este servidor remoto (llamado repositorio) actua también como "backup" y nos permite **recuperar nuestra información ante eventuales problemas que puedan surgir** (pérdida de datos del sistema, robos, borrados accidentales, etc).

Estos servidores pueden ser propios y alojados en nuestra propia infraestructura o podemos usar soluciones de terceros, existiendo soluciones gratuitas disponibles. En nuestro caso, hemos empleado precisamente una de estas soluciones de terceros gratuitas.

Concretamente hemos hecho uso de **Gitlab**.

Por último, hemos querido exprimir al máximo las posibilidades que nos ofrece GIT para basarnos en él para servirnos como motor de configuración que nos haga mucho más sencillo éste proceso.

El lector podrá ver en el CD la existencia de una carpeta llamada "openhab-1.8.1" que cuenta con una carpeta ".git". Aunque no queremos entrar en profundidad en los entresijos de GIT ya que creemos que escapa del alcance del proyecto, muy brevemente podemos comentar que es un CVS distribuído de forma que cada equipo que cuente con el proyecto puede replicar por completo el árbol de versiones sin necesidad de un nodo central. Esta carpeta ".git" posee precisamente esta información; es donde git guarda toda la información que necesita para funcionar correctamente. Por favor, en ningún momento modifique o elimine esta carpeta o el sistema automatizado podría dejar de funcionar.

Además necesitará tener instalado GIT en el sistema pero dicho software viene ya por defecto en el sistema operativo que hemos instalado en la Raspberry.

Volviendo a la carpeta "openhab-1.8.1" del CD, ésta deberá ser copiada a alguna carpeta del sistema (proponemos "/opt/openhab", eliminando así el código de la versión). En dicha carpeta tendremos pues el script de instalación del cual hablabamos junto con la carpeta ".git".

Ejecutamos el script mediante:

```
1 sudo sh custom_install.sh
```

Hasta "sudo git checkout linux", estamos descargando OpenHAB y a partir de ahí, con esa sentencia nos movemos a una rama Linux que contiene todo lo necesario para trabajar en este entorno. Al hacerse el cambio de rama, automáticamente los ficheros de la carpeta cambian, recuperándose éstos desde la carpeta ".git". Dichos ficheros ya están configurados para trabajar con un OpenHAB configurado, al menos la mayor parte. Posibles ajustes los veremos detenidamente en la sección específica sobre OpenHAB.

El resto del script simplemente otorga los **permisos necesarios a la carpeta de OpenHAB** y sus respectivos ejecutables. Por último, a partir de la línea 8, **se configura OpenHAB para que arranque automáticamente al iniciarse la Raspberry**, evitándonos tener que hacer esta tarea de forma manual cada vez que reiniciamos la Raspberry.

4.1.4. Mejoras de rendimiento

A mayores, hemos realizado unas pequeñas configuraciones adicionales con las que buscamos mejorar el rendimiento de nuestra Raspberry para las necesidades que nos ocupan.

Es importante entender que estas optimizaciones son completamente opcionales y por tanto **no se ejecutan de forma automatizada junto con el resto de la instalación**, por lo que si el lector quiere contar con ella en sus desarrollos, deberá ejecutarlas de forma manual.

La mayoría de estas recomendaciones han sido extraídas de la guía oficial de instalación para esta plataforma[8].

En primer lugar, vamos a **minimizar la carga de escritura/lectura efectuada sobre la tarjeta de memoria** y que afecta negativamente al rendimiento del sistema y a la vida útil de dicha tarjeta. Es mucho más preferible usar la memoria RAM para este tipo de rápidas escrituras y lecturas con información temporal.

Ejecutamos el siguiente comando para hacer efectivo el cambio:

```
1 sudo systemctl enable tmp.mount
```

Por otra parte, nuestro sistema no consta de un entorno visual de escritorio, sino que todo es accesible vía comandos o vía web. Esta situación hace que prácticamente no usemos la GPU, por lo que **podemos reducir memoria dedicada a esta para potenciar la empleada por la RAM**.

Para ello, empleamos la herramienta raspi-config y en configuración avanzada aplicamos este cambio (recomendamos reservar sólo 16MB para la GPU).

Por último, **vamos a deshabilitar la configuración de la TV** (activa por defecto). Para ello, abrimos "/etc/rc.local" y añadimos:

```
# Limit GPU IRQs
fbset -xres 16 -yres 16 -vyres 16 -depth 8
/opt/vc/bin/tvservice -o
```

Con estas tareas deberíamos tener nuestra Raspberry lista y configurada para trabajar con OpenHAB. Si está realizando una configuración de cero, le recomendamos que llegados a este punto, reinicie el dispositivo para comprobar que todo funciona correctamente.

4.2. OpenHAB

Una vez tenemos instalado el sistema en la Raspberry Pi y ésta está configurada y preparada para arrancar OpenHAB, procedemos a configurar éste.

Es importante destacar que existe un proceso de configuración relativamente extenso en el cual se monta la estructura de la aplicación en sí. Sin embargo, el proceso está claramente facilitado por el proyecto "demo" de OpenHAB, configurado inicialmente para funcionar.

En este proyecto, hemos modificado y/o eliminado las configuraciones no necesarias para nuestros propósitos de la "demo" oficial, quedándonos con un **proyecto mucho más ligero que nos permite ahorrarnos configuraciones superfluas que no vamos a emplear**.

Dicha operación de creación de una base que funcione sobre la que apoyarnos para crear la aplicación tal y cómo queremos que sea, ya viene dada en el CD que acompaña a la memoria y ha sido desplegada a través del script que veíamos en la sección anterior por lo que podemos comenzar con el proceso de configuración como tal.

Dicho proceso de configuración puede dividirse fácilmente en varias etapas y creemos que hablar de ellas de una en una será la opción más clara.

También, a nivel general queremos explicar que para arrancar OpenHAB tenemos disponibles dos scripts:

- start.sh
- start_debug.sh

Ambos inician OpenHAB pero la segunda opción lo hace en modo DEBUG lo que implica que devuelve mucha información por consola en el proceso y está destinado a ser un inicio para desarrolladores.

4.2.1. RFXCOM

Es el primer elemento que debemos configurar ya que todas las comunicaciones inalámbricas pasan a través de él. Todos los elementos externos y gran parte de las funcionalidades que nos provee OpenHAB son dadas a través de "bindings".

Estos "bindings", no dejan de ser pequeñas librerías con toda la lógica necesaria para poder añadir un dispositivo o funcionalidad nueva al sistema general que forma OpenHAB.

El caso de RFXCOM es precisamente este último por lo que antes de poder emplear el dispositivo en el sistema debemos instalar el correspondiente "bindings". Sin embargo, de nuevo esta tarea viene ya dada por el script ejecutado en la sección anterior, pudiendo comprobar que en "addons" se encuentra el fichero:

- "org.openhab.binding.rfxcom_1.9.0.201601240042.jar".

A partir de este momento debemos configurarlo. Acudimos al fichero "openhab.cfg" localizado en la carpeta "configurations".

En ella, deberemos fijar el puerto serie que ocupa nuestro dispositivo en el sistema. El valor que tome será diferente según el equipo, variando el formato entre sistemas operativos. De este modo, en nuestra Raspberry Pi la configuración quedaría así:

```
965 ############## RFXCOM Binding ###################
966
967 # Serial port of RFXCOM interface
968 # Valid values are e.g. COM1 for Windows and /dev/ttyS0 or /dev/ttyUSB0 for Linux
969 rfxcom:serialPort=/dev/tty.usbserial-A1Z5E3FS
970
971 # Set mode command for controller (optional)
972 # E.g. rfxcom:setMode=0D00000003530000C2F00000000
973 # rfxcom:setMode=
974
```

Otro proceso que debemos llevar a cabo es configurar el RFXCOM en sí para que trabaje con los protocolos que deseamos. Para ello, deberemos descargar de la página oficial de RFXCOM, bajo la sección "Downloads", el software para Windows **RFXmngr**, abrirlo y seleccionar HomeEasy y Hideki/UPM como protocolos soportados.

Llegados a este punto, deberíamos arrancamos OpenHAB como indicábamos anteriormente y comprobamos que se detecta correctamente nuestro RFXCOM tal y como podemos apreciar en la Figura 4.1.

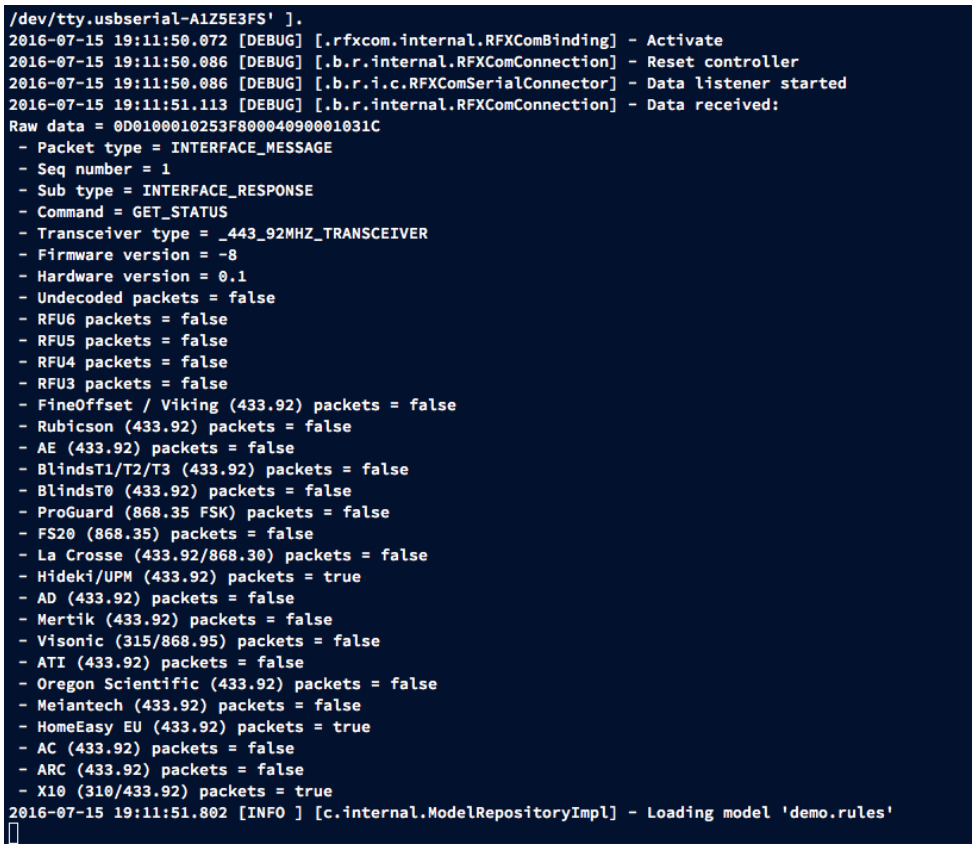

Figura 4.1: RFXCOM funcionando.

4.2.2. Items

Llegados a este punto, tenemos configuradas todas las interfaces de entrada que usaremos en OpenHAB por lo que podemos empezar a configurar el sistema en sí y las relaciones que existen entre ellos.

Configuraremos los elementos virtuales (entidades) con los que trabaja OpenHAB y que sirven de abstracción lógica sobre los elementos físicos que hemos tratado en secciones anteriores.

Dicha configuración se establece en "configurations/items/demo.items" y su contenido es mostrado a continuación:

```
1 Switch Presencia1                    { rfxcom="<9376.2:Command" }
2 Number CamaSensorState1          { rfxcom="<8462:Temperature" }
3 Number CamaBatteryLevel1          { rfxcom="<8462:BatteryLevel" }
```

Existen diferentes tipos de variable, pero en nuestro caso, empleados Switch (semejante a un booleano que puedo tomar dos valores [ON|OFF]) y NUMBER (auto-descriptivo).

A continuación se establece el **mapeo entre el dispositivo en sí, el nombre que queremos que tome en nuestro sistema y el dispositivo físico**. En nuestro caso, estos dispositivos físicos corresponden a RFXCOM[11].

Es importante destacar que cada interfaz tiene su propia metodología para llevar a cabo el mapeo y habrá que seguir la documentación correspondiente para ella, en nuestro caso:

- https://github.com/openhab/openhab/wiki/RFXCOM-Binding

No queremos centrarnos en volver a explicar lo que la documentación oficial hace ya con ejemplos, pero básicamente, el mapeo se realiza a través del **ID del dispositivo** (que deberemos ver a través de la consola) y la propiedad que queremos mapear, que dependerá del protocolo empleado pero en nuestro caso es Command para HomeEasy y Temperature y BatteryLevel para el sensor de cama que usa CRESTA.

Esta configuración se repite ocho veces, una por cada habitación, con los identificadores concretos para cada caso. Se puede comprobar en el CD que acompaña a la memoria el contenido completo de este fichero.

4.2.3. Rules

Por otra parte, **la lógica de la aplicación se realiza a través de reglas** en el fichero "configurations/rules/demo.rules". En él, a través de un pseudo-lenguaje de programación, se definen qué acciones se deberán llevar a cabo según la situación del sistema.

Esta configuración, se podría considerar como la más importante de todo el proyecto ya que **define cómo se comportará el sistema en su conjunto y las características que hemos implementado**.

En primer lugar, vamos a ver el aspecto que tiene el fichero:

```
1 import org.joda.time.*
2 import org.openhab.core.library.types.*
3
4 var DateTime camaLastUpdate1
5 var DateTime camaLastUpdate2
6 var DateTime camaLastUpdate3
7 var DateTime camaLastUpdate4
8 var DateTime camaLastUpdate5
9 var DateTime camaLastUpdate6
```

```
10 var DateTime camaLastUpdate7
11 var DateTime camaLastUpdate8
12 var Number maxTime = 5
13
14 rule "save1"
15         when
16                 Item CamaBatteryLevel1 received update
17         then
18                 camaLastUpdate1 = now
19
20 rule "save2"
21         when
22                 Item CamaBatteryLevel2 received update
23         then
24                 camaLastUpdate2 = now
25
26 rule "save3"
27         when
28                 Item CamaBatteryLevel3 received update
29         then
30                 camaLastUpdate3 = now
31
32 rule "save4"
33         when
34                 Item CamaBatteryLevel4 received update
35         then
36                 camaLastUpdate4 = n
37
38 rule "save5"
39         when
40                 Item CamaBatteryLevel5 received update
41         then
42                 camaLastUpdate5 = now
43
44 rule "save6"
45         when
46                 Item CamaBatteryLevel6 received update
47         then
48                 camaLastUpdate6 = now
49
50 rule "save7"
51         when
52                 Item CamaBatteryLevel7 received update
53         then
54                 camaLastUpdate7 = now
55
56 rule "save8"
57         when
58                 Item CamaBatteryLevel8 received update
```

```
59          then
60                  camaLastUpdate8 = now
61
62  rule "check"
63          when
64                  Time cron "0 0/1 * 1/1 * ? *"
65          then
66                  var DateTime checkDate = now.minusMinutes(maxTime)
67              if (camaLastUpdate1 != null && camaLastUpdate1.isBefore(checkDate)) {
68                      logInfo("Cama 1", "No Signal - Battery Status ??" )
69                      CamaBatteryLevel1.postUpdate(0)
70              }
71              if (camaLastUpdate2 != null && camaLastUpdate2.isBefore(checkDate)) {
72                      logInfo("Cama 2", "No Signal - Battery Status ??" )
73                      CamaBatteryLevel2.postUpdate(0)
74              }
75              if (camaLastUpdate3 != null && camaLastUpdate3.isBefore(checkDate)) {
76                      logInfo("Cama 3", "No Signal - Battery Status ??" )
77                      CamaBatteryLevel3.postUpdate(0)
78              }
79              if (camaLastUpdate4 != null && camaLastUpdate4.isBefore(checkDate)) {
80                      logInfo("Cama 4", "No Signal - Battery Status ??" )
81                      CamaBatteryLevel4.postUpdate(0)
82              }
83              if (camaLastUpdate5 != null && camaLastUpdate5.isBefore(checkDate)) {
84                      logInfo("Cama 5", "No Signal - Battery Status ??" )
85                      CamaBatteryLevel5.postUpdate(0)
86              }
87              if (camaLastUpdate6 != null && camaLastUpdate6.isBefore(checkDate)) {
88                      logInfo("Cama 6", "No Signal - Battery Status ??" )
89                      CamaBatteryLevel6.postUpdate(0)
90              }
91              if (camaLastUpdate7 != null && camaLastUpdate7.isBefore(checkDate)) {
92                      logInfo("Cama 7", "No Signal - Battery Status ??" )
93                      CamaBatteryLevel7.postUpdate(0)
94              }
95              if (camaLastUpdate8 != null && camaLastUpdate8.isBefore(checkDate)) {
96                      logInfo("Cama 8", "No Signal - Battery Status ??" )
97                      CamaBatteryLevel8.postUpdate(0)
98              }
99          end
```

Si el lector está familiarizado con el lenguaje de programación Java, seguramente entienda el código sin mayores explicaciones. Vemos como realmente no es Java, sino "algo" parecido, un pseudo-lenguaje inspirado en Java.

En primer lugar, importamos dos librerías, la primera, Jodatime, nos ayudará en el manejo de variables temporales, mientras que la segunda pertenece al propio framework

de OpenHAB y sirve para emplear más tipos de datos a parte de los dados por defecto.

A continuación, definimos una variable DateTime por cada cama, donde guardaremos la última vez que se actualizó, dicha variable.

También hemos definido una constante llamada "maxTime" donde guardamos, en minutos el tiempo máximo (ahora veremos para qué).

A partir de aquí, comenzamos con la definición de las reglas. Vemos que las ocho primeras reglas son iguales y corresponden cada una a uno de los sensores.

En general, la estructura de las reglas sigue un:

"Cuando pase A, ejecuta B".

En el primer juego de reglas, le estamos diciendo que cada vez que recibamos una actualización sobre el estado de la batería, almacenamos en la variable que definíamos antes el momento actual, de modo que en todo instante tengamos en esas variables cuándo se produzco la última comunicación.

Este valor lo usamos en la siguiente regla "check". En este caso, el evento que lanza la regla no es un cambio o recepción de datos, sino que estamos ante un tarea periódica que se ejecuta según una **expresión Cron**[6].

En nuestro caso, la expresión cron es "0 0/1 * 1/1 * ? *" y corresponde con "cada minuto". Por tanto, cada minuto ejecutamos la regla.

En ella, a alto nivel, **comprobamos si han pasado más del número de minutos definidos por "maxTime" desde la última actualización de cada sensor** y, en caso afirmativo, suponemos que la batería se ha agotado por lo que actualizamos el sistema activando la alerta de batería baja para dicho sensor.

Si hacemos el análisis más a bajo nivel, vemos como en "checkDate" almacenamos la fecha mínima de actualización que deberán tener todos los sensores para considerar que su batería no se ha acabado.

A continuación, uno por uno en cada sensor, comprobamos que el valor de la última actualización no es nulo (si lo fuera, nunca habría sido activado en el sistema y causaría un fallo en el resto de comprobaciones) y que su fecha última de actualización, la cual vimos hace unos instantes como almacenábamos, es anterior a la fecha mínima calculada al comienzo de la regla.

En caso de que esta validación sea positiva significa que hace más tiempo que el definido por "maxTime" que no se actualiza y por tanto, mediando "logInfo" mandamos

un mensaje explicando la situación y actualizamos el valor del estado de la batería del sensor de la cama a 0, lo cual significa batería baja y es **mostrado en el panel de información como una alerta**.

4.2.4. Interfaz Gráfica

Por último, en esta sección vamos a explicar cómo hemos configurado el panel principal gracias al cual los trabajadores de la instalación puedan conocer el estado del sistema en todo momento.

En primer lugar es importante destacar como el propio framework incluye un modo de construir interfaces que por defecto son responsive.

Como hemos indicado en otros puntos de esta memoria, la cualidad de una página web de ser responsive significa que automáticamente se adapta a la pantalla según la resolución de esta por lo que, independientemente del dispositivo que empleemos para su visualización (PC, tablet, smartphone, etc), el contenido se mostrará perfectamente adaptado a su pantalla.

A la hora de crear un panel gráfico en OpenHAB, principalmente tenemos dos opciones. La primera sería usar estas características embebidas mientras que la segunda es más compleja pero también más flexible y requeriría crear una API REST con los datos de nuestro sistema y consumirla desde una aplicación móvil y/o web que tendríamos que construir nosotros desde cero.

Nuestras necesidades en cuanto al panel de control son bastante simples y no precisaremos más características de las ya dadas por la interfaz por defecto, por lo que hemos decidido emplear está implementación[7].

A continuación, podemos ver una sección (la más representativa) del fichero "configurations/sitemaps/demo.sitemap" donde podemos apreciar como se lleva a cabo la configuración del panel.

```
1  sitemap demo label="Centro Asistencial"
2  {
3    Frame label="Habitación 1" {
4          Text item=Presencia1 label="Presencia" visibility=[Presencia1==OFF] icon="selfSlider-0"
5          Text item=Presencia1 label="Presencia" visibility=[Presencia1==ON] icon="selfSlider"
6          Text item=CamaSensorState1 label="Estado Cama: Vacía" visibility=[CamaSensorState1==0] icon="sun"
7          Text item=CamaSensorState1 label="Estado Cama: Ocupada" visibility=[CamaSensorState1==1] icon="moon"
8          Text item=CamaBatteryLevel1 label="Batería Baja" visibility=[CamaBatteryLevel1==0] icon="energy"
9          }
```

Como vemos, la configuración se lleva a cabo mediante la definición de Sitemaps con

Frames (marcos) dentro de ellos. En primer lugar, definimos el título del panel, en nuestro caso “Centro Asistencial”.

A continuación, definimos un Frame correspondiente a una habitación. En él, establecemos de nuevo con la etiqueta “label” el nombre del marco (en nuestro caso, Habitación 1).

A partir de este momento, en el interior del Frame, podemos incluir elementos. El lector podrá apreciar que el nombre de dichos elementos corresponde con los “items” que definíamos en puntos anteriores de esta sección.

También se puede apreciar como se establece el tipo de dato y que este no tiene porqué ser igual al del ítem, pudiendo realizarse casteos (cambios de tipo) automáticos en algunos casos, como por ejemplo de “Number” a “Text”.

Por último, se establecen una serie de propiedades extra:

- **“label”**: como anteriormente sucedía, el nombre que aparecerá en pantalla.
- **“visibility”**: visibilidad del elemento. Sólo se verá si la expresión indicada al evaluarse es verdadera.
- **“icon”**: icono que acompaña al texto, situándose justo delante de él.

Se puede apreciar como existe una **cierta duplicidad en las variables mostradas**. Esta situación se da ya que los iconos mostrados y el texto, varían según el valor que tome la variable. Podemos fijarnos que **las propiedades de visibilidad son antagónicas** para aquellos objetos visuales que comparten variable de referencia (item).

En cuanto al indicador de la batería, vemos que sólo se muestra en caso de que el estado de la misma sea baja, mostrando entonces una alerta de batería baja.

Dicha configuración del Frame para una habitación se repite siete veces más, con las variables propias de cada habitación. Si el lector lo desea, puede comprobar el contenido integro de este fichero en el CD que acompaña a esta memoria.

Con esta sencilla configuración, tendríamos disponible el panel de visualización. En la Figura 4.2 podemos apreciar el aspecto que presenta visto desde un navegador Web (en este caso Chrome) en un PC mientras que en la Figura 4.3 podemos apreciar su aspecto desde un dispositivo móvil (iOS).

Por otra parte, hemos querido ilustrar en esta memoria algunas de las diferentes casuísticas que podemos apreciar en el sistema en las Figuras 4.4, 4.5, 4.6, 4.7:

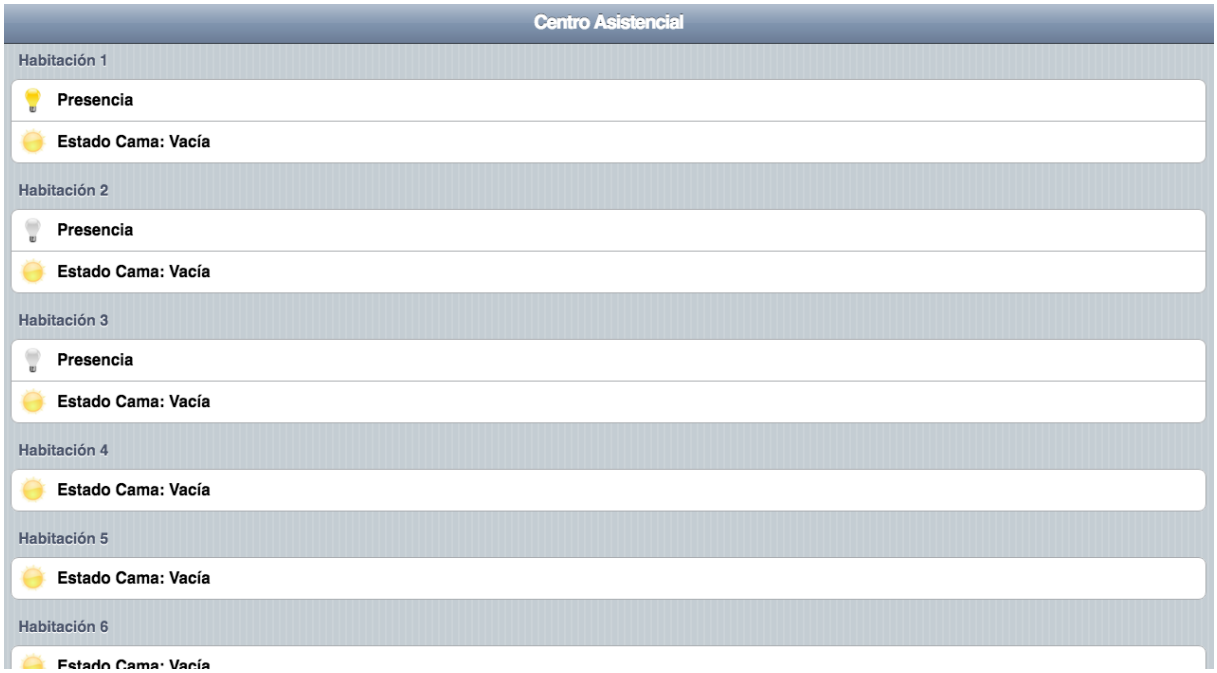

Figura 4.2: Apariencia del panel de control desde un navegador Web.

Figura 4.3: Apariencia del panel de control desde iOS.

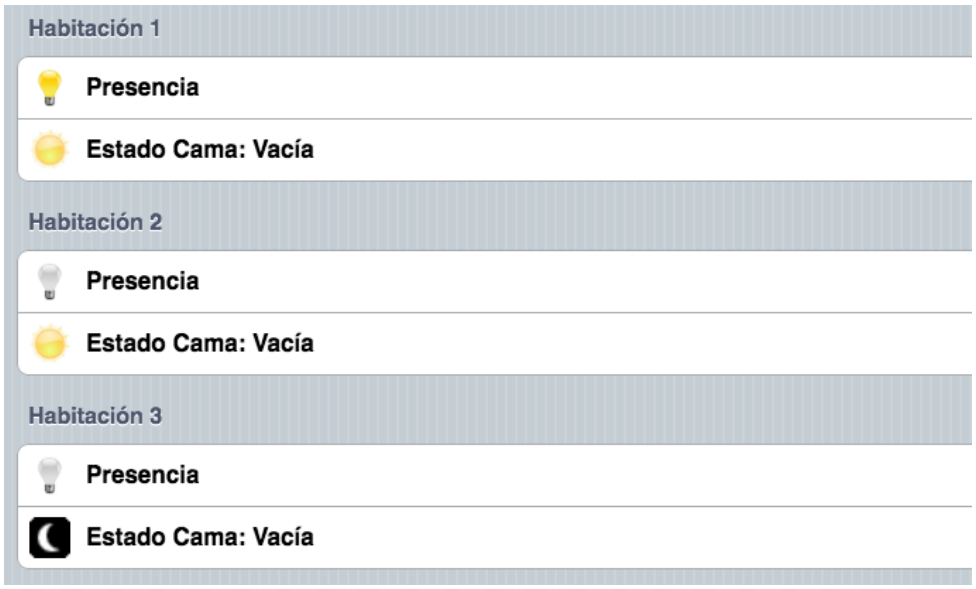

Figura 4.4: Diferentes opciones en el funcionamiento multi-sensor.

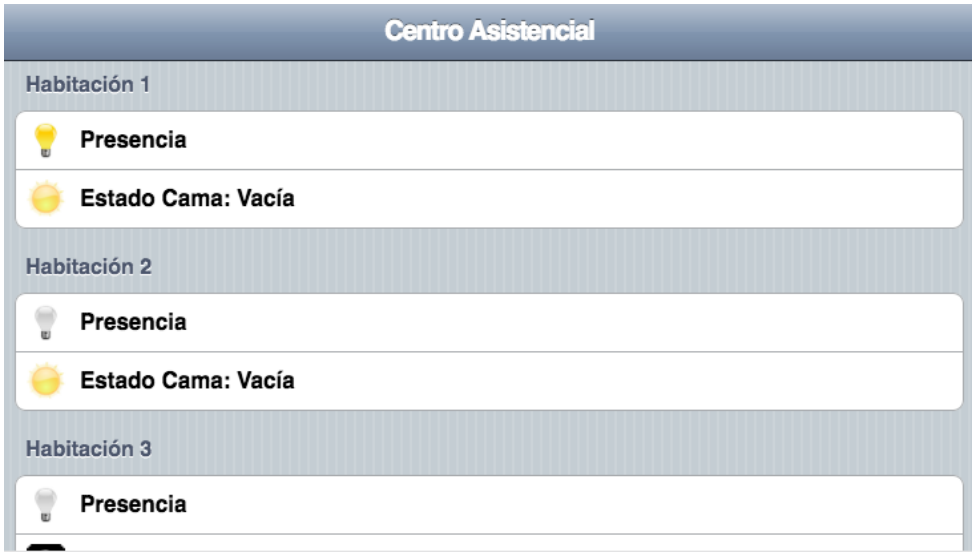

Figura 4.5: Apariencia del panel desde un dispositivo Android apaisado.

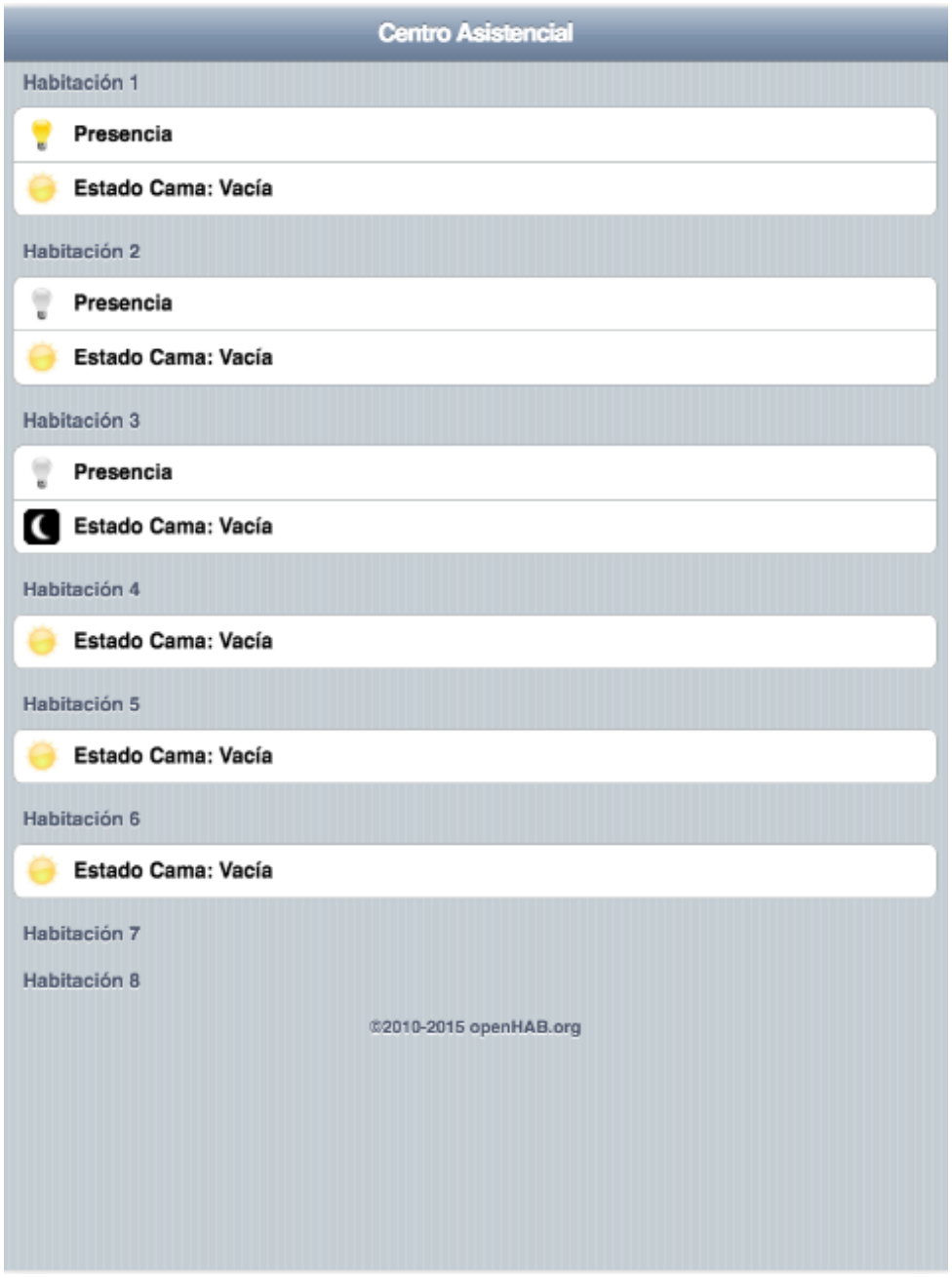

Figura 4.6: Apariencia del panel de control desde una tablet.

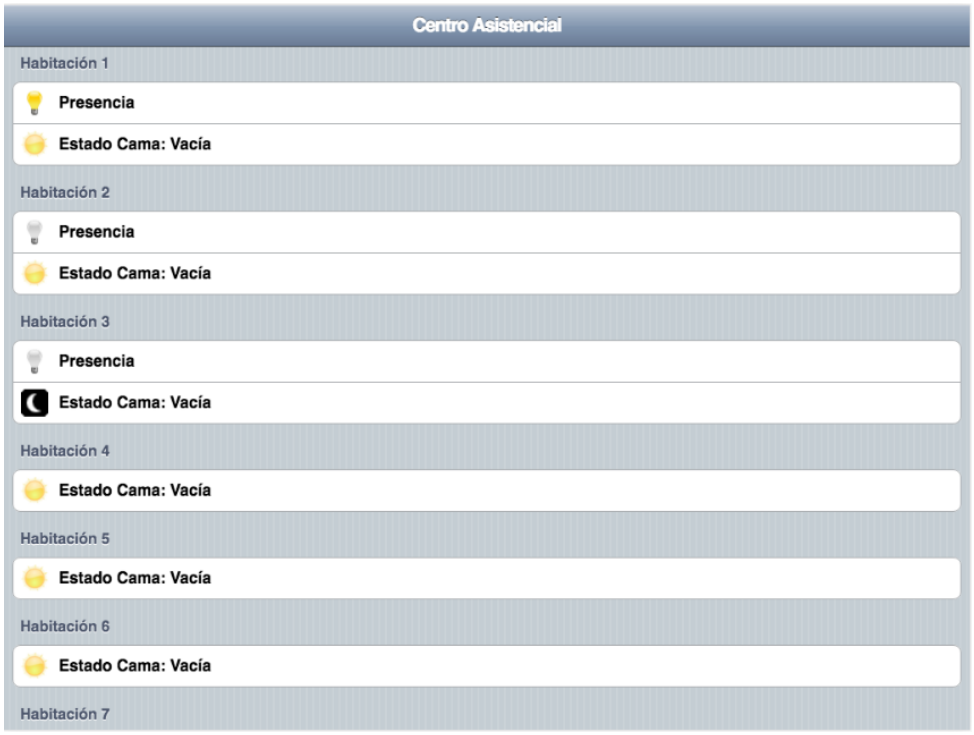

Figura 4.7: Apariencia del panel de control desde una tablet en formato apaisado.

4.2.5. ROS

A continuación, llevamos a cabo un proceso similar al realizado con RFXCOM pero, en esta ocasión para **configurar ROS**. Sin embargo, esta vez, el proceso de configuración no es en OpenHAB.

La comunicación entre ROS y OpenHAB es dado por "iot_bridge", hasta hace poco llamado "openhab_bridge". Sin embargo, esta librería funciona en ROS (recordamos que ROS es un framework en si mismo) y se comunica con OpenHAB a través de la API REST que este último despliega.

Hemos decidido incluir esta configuración dentro de la sección de OpenHAB ya que en el alcance de este proyecto no se encuentra la configuración del robot antropomorfo (que es quien emplea ROS para comunicarse).

Concretamente, en esta subsección, vamos a documentar las configuraciones y comprobaciones que hemos tenido que llevar a cabo para que el sistema quede preparado para enlazarse con ROS de forma rápida y sencilla.

ROS está especialmente pensado para ejecutarse en entornos LINUX (concretamente Ubuntu y Debian) por lo que hemos virtualizado uno y configurado tal y como se indica en la documentación oficial del Framework[10].

A continuación, hemos instalado "iot_bridge" en el sistema que acabamos de configu-

rar siguiendo los siguientes pasos:

```
cd catkin_ws/src
git clone https://github.com/corb555/iot_bridge.git
cd ..
catkin_make
```

Posteriormente, debemos editar el fichero “iot_bridge/config/items.yaml” con la dirección IP de nuestra Raspberry Pi donde tenemos ejecutando el servidor OpenHAB.

Una vez llegados a este punto, tenemos una sistema ROS básico donde poder comprobar el enlace con OpenHAB. Pasamos ahora a configurar OpenHAB.

Anteriormente hablábamos de los “Items” pero no comentábamos una funcionalidad que estos tienen asociada puesto que no la empleábamos. Esta funcionalidad que ahora sí cobra especial relevancia es la **capacidad de definir grupos como conjuntos de elementos**. Vamos a crear un grupo ROS y todos los elementos que pertenezcan a este grupo ejecutaran las acciones de entrada/salida que configuremos.

Es importante destacar que los elementos deben pertenecer directamente al grupo, puesto que el “iot_bridge” no es capaz de alcanzar más de un nivel de búsqueda por lo que subgrupos no serán detectados.

Realmente, lo que hace este puente entre los dos sistemas es **duplicar y sincronizar ciertas variables en ambos entornos para que podamos operar con ellas desde ambos frameworks**. Vamos a realizar una pequeña prueba de concepto.

En nuestro fichero de items, definimos un nuevo grupo llamado ROS. Para ello, al comienzo del fichero añadimos:

```
Group ROS (All)
```

Además, visto que queremos enlazar (por poner un ejemplo) los sensores de presencia, vamos a añadir estos sensores al grupo que acabamos de crear:

```
Switch Presencia1                    { rfxcom=''<9376.2:Command'' } (ROS)
```

Como vemos, la sintaxis para añadir un elemento a un grupo es muy sencilla y tan sólo requiere poner entre paréntesis el grupo al que queremos que pertenezca el elemento en cuestión. Añadimos todos los sensores de presencia a este grupo.

Ahora ya tenemos ambos sistemas correctamente configurados y podemos proceder a probarlos.

En primer lugar, arrancamos ROS:

```
roslaunch iot_bridge iot.launch
```

El framework debería arrancar sin problemas.

A continuación, vamos a comprobar que recibimos información desde OpenHAB. Como indicábamos al comienzo, este puente que interconecta ambos sistemas se basa en la API REST que levanta OpenHAB.

Dicha API funciona por defecto (aunque podemos desactivarla si lo deseáramos) y nuestro primer paso va a ser probarla directamente desde el navegador. Para ello, abrimos una ventana nueva e introducimos la siguiente URL:

- http://IP-RPI:8080/rest/items/ROS

donde "IP-RPI" corresponde con la IP local de nuestra Raspberry Pi.

Deberíamos obtener una **respuesta XML con los estados de todos los sensores de presencia del sistema**. A continuación, nos dirigimos al sistema donde tenemos arrancado ROS y ejecutamos:

```
rostopic echo /iot_updates
```

A medida que cambia el estado de los sensores de presencia en OpenHAB, vemos el nuevo estado en ROS, certificando que la comunicación OpenHAB ⇒ ROS es correcta.

Por último, vamos a comprobar la comunicación inversa (ROS ⇒ OpenHAB). Para ello, "iot_bridge" ya incorpora una pequeña funcionalidad que nos facilita el proceso de testing. Ejecutamos los siguientes comandos en nuestro sistema ROS:

```
cd catkin_ws/src/iot_bridge/scripts
./iot_test  item_name item_value
```

donde item_name y item_value deben ser sustituidos por el valor de uno de los items de nuestro sistema OpenHAB y un valor válido para el tipo de elemento del que se trata respectivamente. Por ejemplo:

```
cd catkin_ws/src/iot_bridge/scripts
./iot_test  Presencia1 ON
```

En el panel de control que hemos configurado, debemos ver como se actualiza el valor aunque el sensor no haya mandado ninguna señal.

Lógicamente, el ejemplo planteado no tiene sentido en el sistema real ya que no deberíamos modificar el valor que obtenemos de un sensor de forma artificial pero si podría servirnos para por ejemplo activar un indicador en el panel de control que alerte que el robot (equipado con ROS) está acudiendo a una zona de trabajo.

Estas configuraciones no están incluidas en los entregables del CD ni en el diseño del sistema final puesto que son absolutamente dependientes del robot equipado con ROS. Es más, como hemos visto, si quisiéramos que el robot actuase ante un evento concreto de nuestro sistema ROS, deberíamos configurar la lógica de respuesta en el propio ROS.

Sin embargo, y por último, las tareas que este trabajo a realizado al respecto dejan patente, por un lado, l**a sencillez y rapidez de configuración** y, por otro, comprueban que **el sistema queda preconfigurado para este tipo de acciones puesto que estas eran las tareas que cubren el alcance propuesto del proyecto**.

4.3. Entorno de Desarrollo

Queríamos acabar este capítulo hablando brevemente de las herramientas de desarrollo que hemos empleado a lo largo de este proyecto. Ya comentamos con anterioridad que en todo momento hemos empleado GIT como control de versiones. Por tanto, éste se puede considerar el centro de nuestra estrategia de desarrollo. Sin embargo, creemos que hablar de metodologías específicas poco puede aportar al lector, por lo que nos centraremos directamente en el software empleado.

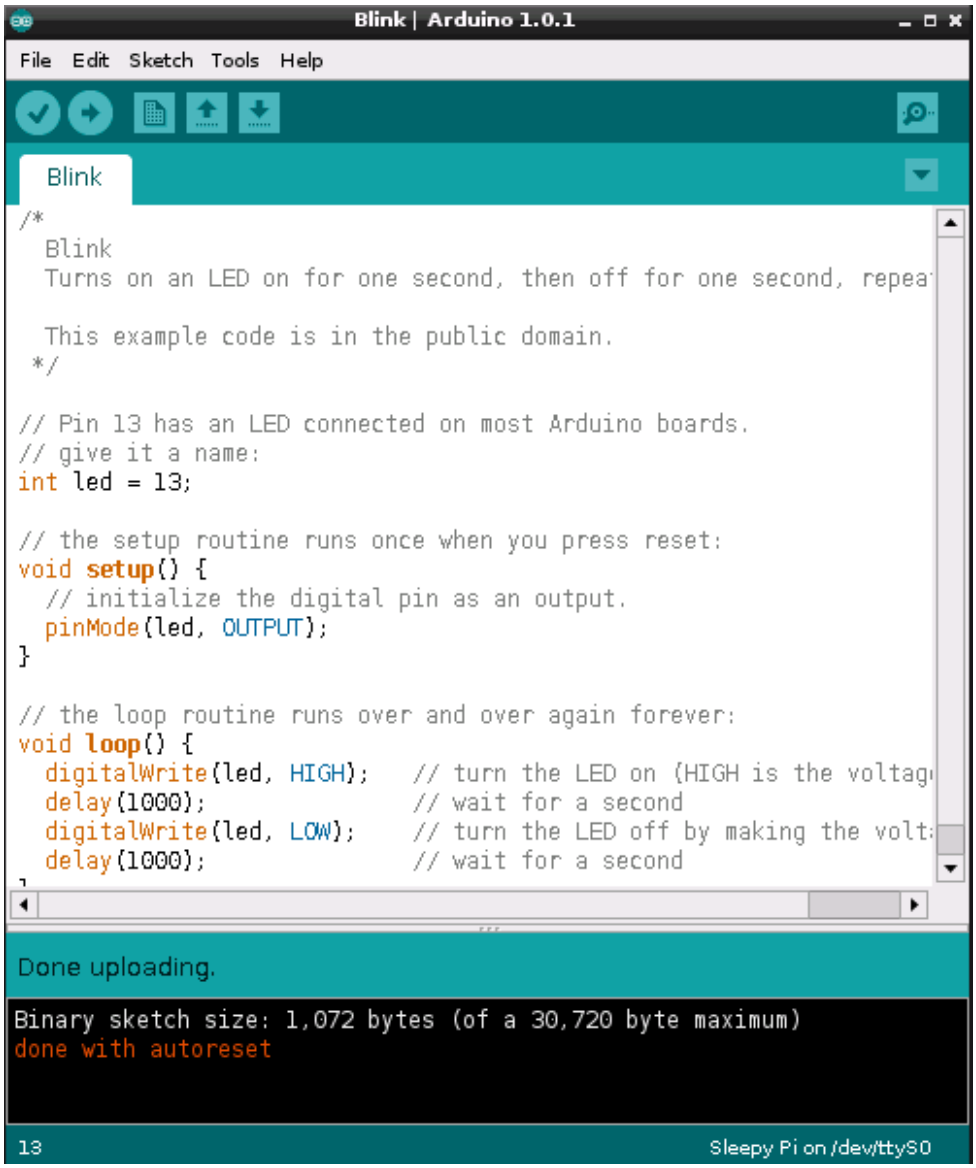

Figura 4.8: IDE oficial de desarrollo con Arduino.

En primer lugar vamos a hablar del **IDE oficial de Arduino** (ver figura 4.8), que aunque simple, ha evolucionado en los últimos años y creemos que es la herramienta más rápida a la hora de trabajar con Arduino.

Sin embargo, una complicación que nos ha surgido a la hora de trabajar con el IDE ha sido al seleccionar Arduino Pro Mini como placa de desarrollo. Dicha placa no consta de un puerto USB para conectarlo al PC y la programación debe llevarse a cabo mediante un cable conversor USB-Serie. En nuestro caso, hemos seleccionado una versión que realiza Sparkfun y, en el propio cable, lleva integrada toda la electrónica. En la figura 4.9 podemos apreciar su aspecto físico.

Hacer funcionar el IDE con este cable requiere instalar los drivers y modificar el programador, pero es una tarea sencilla. Sin embargo, a medida que el proyecto ha ido avanzando en sus fases, cada vez ha sido más complejo trabajar con el IDE. **La razón**

Figura 4.9: Cable de programación. Conversor USB-Serial

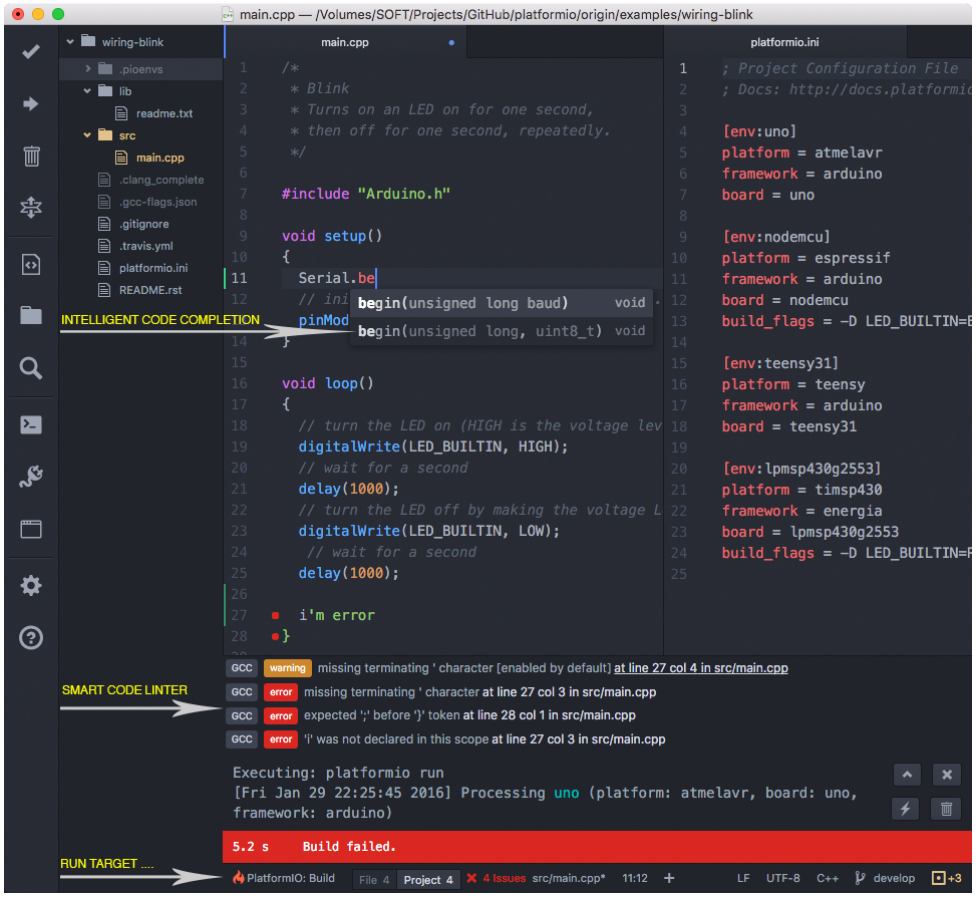

Figura 4.10: IDE formado por Atom y Platformio.

principal es el manejo que éste hace de las dependencias. Tenemos que añadir cada librería que queremos probar al directorio general de la instalación de nuestro Arduino.

Esta situación lleva a que según probamos varias librerías, perdamos de vista cuales realmente necesitamos y cuales no para nuestro proyecto. Además, este IDE no tiene ningún tipo de soporte a GIT, por lo que el control de versiones del código se hace más complicado.

Dichos problemas nos han llevado a trabajar con otro IDE y el resultado ha sido fantástico. El IDE empleado se llama **Atom** y, al igual que el de Arduino, es software libre. Sin embargo, éste no es específico para programar microcontroladores sino que es de propósito general y puede ser empleado para prácticamente cualquier tarea de programación y/o configuración.

La mecánica de este editor se basa en su facilidad para añadir **nuevas funcionalidades a través de plugins**. En nuestro caso, hemos usado uno que convierte Atom en un potente IDE para Arduino y otras plataformas y microcontroladores. Estamos hablando de **Platformio**. En la Figura 4.10 puede apreciarse su apariencia y algunas de sus características.

Este proyecto, también libre, ha desarrollado una serie de plugins para Atom que

emplean su herramienta (escrita en Python) y desde la cual y a través de la línea de comandos, somos capaces de cargar nuevos programas al dispositivo, pudiendo, el mismo programa, cargarlo en diferentes entornos, plataformas y usando diferentes programadores de una forma realmente muy sencilla, ya que tan sólo requiere tener definidos tantos perfiles (con sus respectivas configuraciones) como opciones queramos tener disponibles. El plugin, se encarga de aprovechar Atom para dotar de una sencilla interfaz gráfica a esta herramienta.

Los proyectos construidos con Platformio tienen una **estructura mucho más portable y replicable en otros entornos** ya que por una lado tiene las librerías (específicas del proyecto con el que estamos trabajando), por otro el código fuente y por otro los perfiles de configuración.

A continuación, reproducimos el perfil de configuración principal del proyecto, mediante el cual, empleando el cable conversor del que hablábamos, transferimos los programas a un Arduino Pro Mini.

```
1  [env:production]
2  platform = atmelavr
3  framework = arduino
4  board = pro16MHzatmega328
5  upload_port = /dev/cu.usbserial-AJ03JHRH
```

El único parámetro que se necesita ajustar es el upload_port, según el puerto serie que ocupe el conversor USB-serie y su sistema operativo.

Gracias a estas herramientas, que además soportan control de versiones con GIT, ha sido **mucho más sencillo trabajar con los constantes cambios que hemos tenido que realizar en este proyecto**.

Por otro lado, ya hablamos con anterioridad del uso de EAGLE tanto para dibujar el circuito como para posteriormente generar el Layout de la placa que hemos diseñado.

Por último, tan sólo nos queda hablar de **OpenHAB Designer**. Esta herramienta, desarrollada bajo el paraguas de OpenHAB está pensada para facilitar la creación de configuraciones, items y reglas, de una forma gráfica. Para ello, usa otro muy conocido IDE libre llamado Eclipse, disponible para varios entornos de desarrollo pero seguramente más conocido por Java.

Debemos recordar que OpenHAB está escrito en Java y cuenta con algunas de las características de este lenguaje de programación como el tipado (característica de aquel lenguaje de programación que especifica el tipo de objeto o variable que está empleando).

En estos entornos, es especialmente útil un IDE de desarrollo capaz de alertarte de posibles problemas en el código de forma dinámica (sin necesidad de compilar y ejecutar el programa).

Sin embargo, debemos decir que la sensación ha sido ciertamente agridulce al detectar como errores gran parte de la sintaxis correctamente escrita (y que de hecho, funciona correctamente). Esto induce a error en muchas circunstancias y retrasa el desarrollo al confundirte con los errores que lanza.

Por otra parte, no cuenta con plantillas que faciliten la creación de rutinas típicas. Donde si que aporta un mayor confort es a la hora de **localizar los ficheros de documentación** ya que en vez de tener que estar navegando entre carpetas, tenemos iconos bastante auto-explicativos que nos guían a la hora de encontrar el fichero de configuración que buscamos.

En general, el uso de este software para configurar OpenHAB nos ha parecido bastante prescindible y hemos acabado abriendo los ficheros de configuración uno a uno con Atom y editándolos desde aquí también.

4.4. Consideraciones generales finales

Queremos cerrar este capítulo con una serie de consideraciones de carácter general acerca del sistema diseñado y su uso.

En primer lugar, recordamos que el nodo central es la Raspberry Pi 2 y es ella quien hace de servidor y donde se está ejecutando OpenHAB. Las comunicaciones de los sensores se realizan en la banda de los 433 MHz y son recibidas a través del RFXCOM, pero existen otra serie de posibles conexiones como todas aquellas llevadas a cabo para ver el panel de control, que se ejecutan bajo el **protocolo TCP/IP**.

Por tanto, deberemos apuntar tanto navegadores como aplicaciones a la IP local que tiene este dispositivo. Precisamente por este motivo, es más que recomendable configurar la Raspberry Pi con una **IP estática**, con el objeto de no tener que cambiar la configuración de los clientes de visualización si el Router cambiara la IP dada a la Raspberry mediante DHCP.

También, de forma intrínseca, estos aspectos requieren de una red local configurada donde existan algunos elementos de red como un router y posibles protecciones a través de hardware o software que limiten el acceso a la red a personal no autorizado.

Estos aspectos los consideramos puramente relativos a la instalación y su configuración de datos, por lo que **no han sido abordados por este trabajo** pero ello no implica

que no sean necesarios.

En este sentido, se da libertad para que la comunicación sea por cable de red o inalámbrica por WiFi.

Por otra parte, haciendo una redirección de puertos en el router hacia la Raspberry Pi, podríamos hacer dicho **sistema accesible desde el exterior** pero de nuevo es una cuestión exclusiva de red por lo que no hemos entrado en ella.

También queremos constatar que **el elemento central deberá estar lo más equidistante posible de todos los sensores**, minimizando en la medida de lo posible la máxima distancia entre nodo central y sensores.

Por diseño, sabemos que dicha distancia máxima son 8 metros por lo que deberemos superar dicha distancia en las pruebas que realizaremos y que detallaremos en la capítulo de resultados.

En dicho capítulo también haremos las pruebas correspondientes para asegurar que dicho sistema es compatible con el mayor número de dispositivos posibles.

Por último, indicar la conveniencia de, si dicha red va a ser pública o compartida con una pública, **asegurar a nivel de red una correcta configuración que evite el acceso indeseado de personas no autorizadas al panel de control**.

Capítulo 5

Resultados

Con todo el sistema ya configurado, es el momento de realizar las **pruebas que acrediten el cumplimiento o no de los objetivos marcados al comienzo de esta memoria**.

Gran parte de estas pruebas se han realizado directamente durante el tiempo de desarrollo pero hemos querido reservar un apartado donde hacer un pequeño resumen de ellas.

En primer lugar, hemos comprobado como el núcleo del sistema funciona correctamente al ejecutarse como una instalación desde cero con los pasos que hemos indicado en la memoria.

La Raspberry Pi 2 cumple sus objetivos como **nodo central barato y eficiente**. Su consumo no supera en ningún momento los 3,5W y tanto la carga del procesador como el uso de la memoria se mantienen en todo momento por debajo del 70 %.

Por otra parte, todo el software empleado en ella no ha presentado ninguna clase de error o fallo, tanto en el desarrollo como en el posterior proceso de prueba general.

En cuanto a los sensores empleados, **se ha comprobado su alcance, fiabilidad y consumo**.

Empezando por el dispositivo de HomeEasy hemos detectado un rendimiento claramente inferior al esperado. Su alcance real se sitúa cercano a los **12-15 metros en entornos con paredes y otros obstáculos**, muy lejos de los valores teóricos de 50 metros que indicaba el fabricante.

Además, hemos tenido que sustituir la pila que alimenta el sistema a los 7 meses, valor también por debajo del año indicado por el fabricante.

En cuanto a su fiabilidad, arroja resultados medios. La integridad de la señal es co-

rrecta y **en ningún momento hemos detectado tramas defectuosas**. Sin embargo, el proceso de detección en sí del sensor no se comporta todo lo bien que creemos debería puesto que aún con la sensibilidad al máximo, hay ocasiones en las que tarda hasta 5 segundos en detectar la presencia humana incluso cuando esta se está moviendo justo delante del sensor.

Si hablamos de estas mismas variables en el sensor de cama obtenemos:

- **Alcance**: 15m.
- **Autonomía**: 6,5 meses. El proceso de testeo se llevo acabo en el desarrollo de esta memoria.
- **Fiabilidad**: Alta, tanto en la transmisión como en la detección.

Sus mayores problemas al respecto tienen que ver con **falsos negativos** arrojados debido al movimiento que una persona tiene al dormir. Sin embargo, dichos errores en muchos casos son imperceptibles ya que rápidamente se vuelve a enviar el valor correcto y, en el caso de que esto no sea así, el envío de seguridad que se hace cada minuto lleva a un **alto margen de confiabilidad**.

En conjunto ambos sensores forman la detección del estado de la habitación y, a la vista de los resultados que acabamos de obtener, podemos decir que en su conjunto presentan una **autonomía levemente superior al medio año y un alcance de la señal inalámbrica algo superior a los 12 metros**.

Ambas variables están por encima de los objetivos de 6 meses y 10 metros que nos fijábamos en los objetivos. Otro resultado destacable es la **buena respuesta que ha dado RFXCOM en todo momento**, incluso ante el envío de señales simultáneas, la tecnología de multiplexado de la que dispone ha asegurado en todo momento la recepción adecuada de las tramas.

En cuanto a las interferencias, no hemos sido capaces de detectar ningún caso donde todos los elementos del sistema conectados a la vez causen interferencias o dañen la integridad de los mensajes inalámbricos enviados.

Creemos que es un apartado complejo y aún con los resultados que hemos obtenido, necesita de varios meses de uso en el Centro para comprobar “in situ” si allí tampoco se producen errores de esta naturaleza, por lo que queremos poner en interrogante los resultados obtenidos en nuestras pruebas al respecto.

Por último, el panel de control muestra en todo momento el estado recibido por los sensores. OpenHAB, a parte de cumplir el objetivo de este trabajo de ser un framework libre, se ha comportado como una **opción excelente y muy fiable y flexible**.

Las pruebas con ROS pueden verse en la sección correspondiente del capítulo de Desarrollo pero sus conclusiones son positivas en cuanto a integración con el resto del sistema a través de OpenHAB.

Hemos realizado varias pruebas comprobando la compatibilidad del sistema (concretamente del panel de control y visualización) en los siguientes sistemas operativos:

- Windows 7, Windows 10
- OSX El Capitan
- Linux (Ubuntu y Debian)

Además hemos probado diferentes configuraciones alternando los sistemas operativos arriba mencionados y los siguientes clientes:

- Chrome
- Firefox
- Opera
- Internet Explorer
- Microsoft Edge
- App nativa Android
- App nativa iPhone
- App nativa iPad
- Navegador por defecto Android
- Navegador por defecto iOS

En todos ellos el resultado ha sido excelente, **cumpliéndose la característica "responsive" que buscábamos**.

Capítulo 6

Estudio Económico

A lo largo de este capítulo, haremos un desglose de los costes asociados a este proyecto para poder estimar los mismos a lo largo de su vida útil. De igual modo, calcularemos una serie de indicadores económicos para validar la **rentabilidad de la instalación**.

Por tanto, en este apartado comprobaremos la **viabilidad económica** del proyecto en su ciclo de vida estimado.

6.1. Costes directos

En esta sección evaluaremos aquellos costes directamente imputables al proyecto.

6.1.1. Recursos empleados

Como se ha indicado ya en repetidas ocasiones a lo largo de la presente memoria, gran parte del proyecto se ha realizado empleando software libre lo que, entre otras ventajas, representa un muy considerable ahorro en cuanto a coste de licencias.

De este modo, hemos empleado las siguientes herramientas sin coste:

- Atom IDE.
- Arduino IDE.
- EagleCAD (versión Freeware).
- Scipy, Simpy, Numpy y otras herramientas matemáticas y de análisis de datos basadas en Python.

Cuadro 6.1: Coste anual del personal.

Concepto		Cantidad (€)
Sueldo Bruto		24000
Seguridad Social	0.3	7200
Coste Total		31200

Cuadro 6.2: Días efectivos anuales.

Año medio	Fines de Semana	Festivos	Vacaciones	Total
365	104	23	25	213

- OpenHAB.
- Framework ROS.
- Gitlab (CVS).

Todo ello se ha empleado usando un Macbook Pro 13´´ cuyo coste sí que debemos considerar.

6.1.2. Costes de Personal

El presente proyecto ha sido realizado íntegramente por **una única persona**. El perfil de dicha persona responde ante la de ingeniero junior con un año de experiencia laboral. Con estos datos, hemos considerado el coste anual de dicha persona mostrado en la tabla 6.1.

A continuación vamos a estimar los días laborables que posee cada año. Dichos cálculos aparecen reflejados en la tabla 6.2.

Con esta información y dividiendo el coste anual del personal entre el número de días laborables por año y entre ocho (suponiendo ocho horas de trabajo al día) obtenemos:

$$\frac{31200€}{213\ días} \cdot \frac{1\ día}{8\ horas} = 18{,}31 \frac{€}{hora} \tag{6.1}$$

A continuación, en la tabla 6.3 la distribución temporal de las diferentes tareas que han compuesto este proyecto para así poder tener una referencia del coste temporal del proyecto:

Con todos estos datos ya tenemos toda la información necesaria para calcular el coste directo asociado al personal involucrado en la realización del proyecto:

Cuadro 6.3: Distribución temporal de las tareas ejecutadas.

Tareas	Duración (horas)
Investigación general	2
Investigación Openhab	3
Investigación Otras Alternativas	4
Investigación RFXCOM	8
Investigar Sensores Cama	8
Investigar PIR	8
Investigar sensores piezoeléctricos	4
Investigación Arduino	20
Investigación Interrupciones	25
Investigación Modo Sleep	25
Investigación ROS	4
Crear plantilla Presentación	2
Crear plantilla Memoria	3
Selección componentes	16
Diseño Circuito	12
Circuito a Eagle	12
Control Batería	8
Programación Arduino	16
Comunicaciones	28
Librería comunicaciones	18
Librería bajo consumo	16
Modificaciones circuito	32
Prototipo 1	8
Calibración	8
Instalación Raspberry Pi	12
Instalación RFXCOM	8
Instalación OpenHAB	14
Configuración OpenHAB	25
Prototipo 2	5
Testing Batería	2
Selección batería	4
Prototipo 3	2
Configuración ROS	6
Fabricación	8
Testing	12
Documentación	175
Gestión	15
Presentación Resultados	30
TOTAL	608

Cuadro 6.4: Amortización equipo informático.

MacBook Pro 13"(€)	Vida (años)	Amort. (€/año)	Corrector uso imputable	Amort. real (€)
1499	4	374.75	0.36	133.71

Cuadro 6.5: Costes de I+D

Componente	Precio Unitario (€)	Unidades	Importe (€)
Arduino Micro	19.9	1.0	19.90
Sensor cuadrado	9.73	1.0	9.73
Resistencias	0.05	6.0	0.30
Emisor-Receptor RF 433	2.75	1.0	2.75
	TOTAL		32.68

$$608\ horas \cdot 18,31 \frac{€}{hora} = 11132{,}39\ € \tag{6.2}$$

6.1.3. Amortización equipamiento informático

Como indicamos anteriormente, tan sólo deberemos considerar el coste de amortización del ordenador empleado para el desarrollo de dicho al proyecto al no tener costes asociados a software o sistema operativo.

Al tratarse de material informático, vamos a aplicar una **amortización lineal simple** al ser ésta la forma más habitual de realizar ésta tarea.

En la tabla 6.4 podemos ver los cálculos empleados para llevar a cabo este proceso.

6.1.4. Costes material empleado

Los costes recogidos a continuación representan el coste que ha supuesto para el proyecto los diferentes elementos empleados durante la elaboración.

Debemos tener en cuenta cómo este proyecto cuenta con dos etapas diferencias:

Por un lado, ha existido una amplia tarea de investigación donde se ha trabajado con pruebas de concepto con elementos reales que, lógicamente, han representado un coste al proyecto aunque gran parte de estos elementos hayan sido descartados en favor de otros. En la tabla 6.5 se muestran estos elementos con sus respectivos costes. Utilizaremos el término Costes de I+D para referirnos a estos costes.

Por otro lado, tenemos un coste completamente asociado al producto/instalación desa-

Cuadro 6.6: Costes de la Instalación.

Componente	Precio Unitario (€)	Unidades	Importe (€)
Cable Programador	15.71	1.0	15.71
RFXCOM	112.95	1.0	112.95
Rasoberry Pi	41.95	1.0	41.95
	TOTAL		170.61

Cuadro 6.7: Costes por habitación domotizada.

Componente	Precio Unitario (€)	Unidades	Importe (€)
Sensor Cama	33.1225	1.0	33.12
Sensor Presencia	24.95	1.0	24.95
	TOTAL		58.07

rrollada. A su vez, estos costes, podemos diferenciarlos entre costes de la instalación (que podemos considerar únicos independientemente del número de elementos conectados a ella) y unos costes por habitación domotizada. Vamos a hacer un desglose de los costes en las tablas 6.6 y 6.7 respectivamente, diferenciando estos valores.

6.1.5. Costes directos totales

Por último, sabemos que el coste directo total del proyecto será la suma de los diferentes costes planteados y que el alcance original del proyecto cubría ocho estancias. Ver tabla 6.8.

6.2. Costes indirectos

Los costes indirectos son aquellos asociados a servicios administrativos, telecomunicaciones, electricidad, calefacción y otros servicios de carácter general.

Cuadro 6.8: Costes Directos Totales.

Costes	Precio Unidad (€)	Unidades	Precio (€)
Coste Investigacion	32.68	1	32.68
Costes Habitaciones	58.07	8	464.58
Costes Instalación	170.61	1	170.61
Coste Salarial	11132.39	1	11132.39
Amortizaciones	133.71	1	133.71
		TOTAL	11933.98

Cuadro 6.9: Costes Totales.

Costes	Precio (€)
Coste Directos	11933.98
Costes Indirectos	1790.10
TOTAL	13724.07

Para su cálculo, aplicaremos un porcentaje del 15 % de los costes directos como estimación de los mismos al tratarse éste valor de un registro habitual en proyectos de Investigación y Desarrollo en proyectos públicos tal y como recoge el Plan Estatal de Investigación Científica y Técnica y de Innovación 2013-2016[19].

Aplicando dicho porcentaje tenemos unos costes indirectos de:

$$11132{,}39€ \cdot 0{,}15 = 1794{,}96\ € \tag{6.3}$$

6.3. Costes totales

Por último, podemos calcular los costes totales de la instalación como la suma de costes directos e indirectos. Podemos ver resumida esta operación en la tabla 6.9.

En la Figura 6.1 podemos apreciar la distribución y peso de cada apartado estudiado en los costes totales.

Como vemos, **el verdadero coste de este proyecto corresponde con el esfuerzo humano realizado** puesto que dicho coste, alcanza aproximadamanete el 80 % del coste total del proyecto.

Por contra, los materiales empleados, realmente cumplen el objetivo planteado al principio que buscaba conseguir una instalación barata.

Además, sabemos que la instalación diseñada tiene capacidad para añadir más elementos. Añadir elementos apenas aumenta el coste de la instalación puesto que las ocho habitaciones suponen un 4 % por lo que cada una contribuye con un 0.5 %.

Por tanto, ya sabíamos que nuestra instalación, técnicamente escalaba bien. Ahora también conocemos que **económicamente compensa escalarla al máximo**.

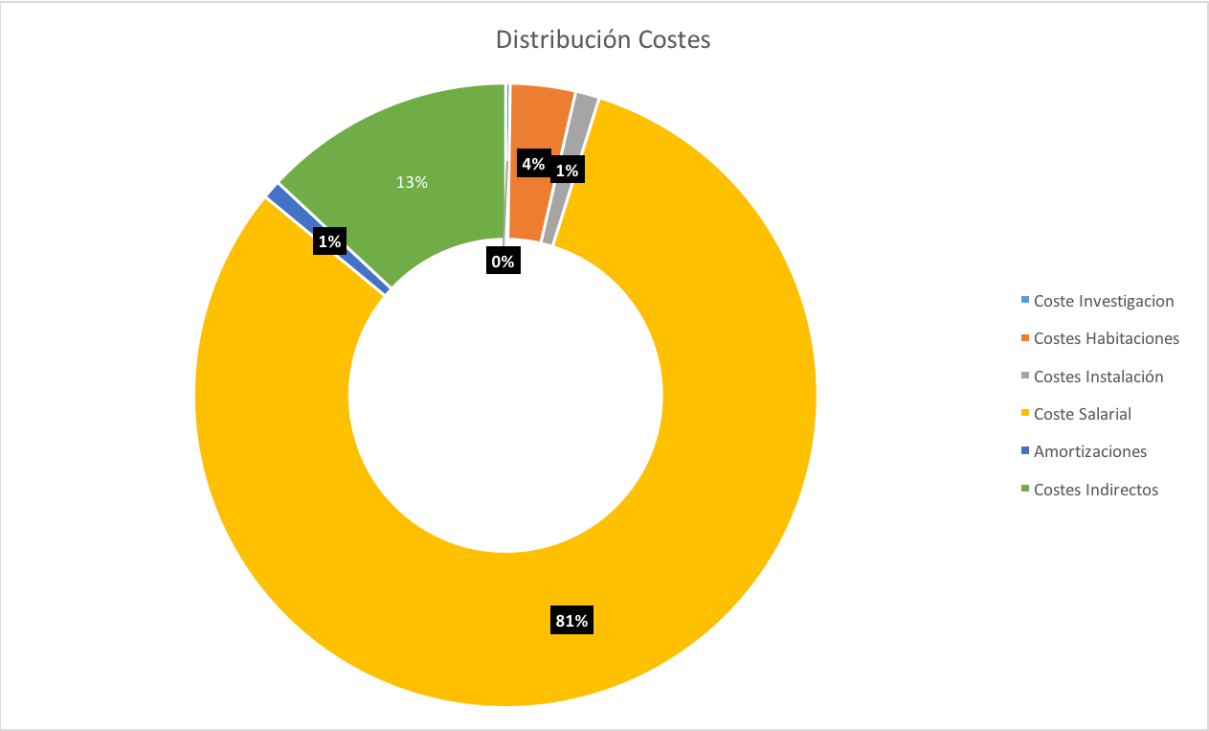

Figura 6.1: Distribución de los diferentes costes del proyecto.

Cuadro 6.10: Presupuesto Ejecución Material.

Concepto		Coste (€)
Total Ejecución Material		13724.07
Beneficio Industrial	0.08	1097.93
Redacción de Proyecto	0.08	1097.93
Dirección de Obra	0.06	823.44
IVA	0.21	3516.11
TOTAL Obra		20259.48

6.4. Presupuesto

Sin embargo, los costes totales indicados no incluyen IVA ni beneficio industrial. En la tabla 6.10, podemos apreciar el Presupuesto de Ejecución Material.

6.5. Financiación

Aunque como vemos, no es una cantidad excesivamente elevada, hemos decidido financiarla para segmentar la inversión a lo largo de 3 años. En este sentido, vamos a pedir un **préstamo por valor de 10.000 €** que nos conceden a un 8 %. Los intereses se deducen de forma lineal en 3 años.

En la tabla 6.11 podemos ver los costes que supone la financiación año a año.

Cuadro 6.11: Financiación.

AÑO	ANUALIDAD (€)	INTERÉS (€)	AMORTIZACIÓN (€)	CAP. VIVO (€)	CAP. AMORTIZADO (€)
0				10000.00	0.00
1	-3880.34	800.00	3080.34	6919.66	3080.34
2	-3880.34	553.57	3326.76	3592.90	6407.10
3	-3880.34	287.43	3592.90	0.00	10000.00

Cuadro 6.12: Costes explotación.

Concepto	Coste (€)
Recambio Pilas	16.00
Consumo Raspberry Pi	2.86
Mantenimiento	34.12
Total	52.98

6.6. Costes de explotación

En este apartado, consideraremos el coste anual que tiene el funcionamiento de la instalación. Realmente, los costes de explotación son realmente mínimos ya que se limitan prácticamente al coste energético de la instalación.

Concretamente, tenemos un consumo relacionado con el gasto en pilas y otro relacionado al consumo energético de la Raspberry Pi.

Como vimos en apartados anteriores, la duración de las pilas es aproximadamente de seis meses, por lo que deberemos cambiar todas las pilas 2 veces al año. Además, tenemos que cambiar la pila del sensor de PIR aproximadamente en el mismo periodo de tiempo. Esto es:

$$2 \cdot 0{,}2€ \cdot 5\frac{pilas}{habitación} \cdot 8habitaciones = 16€ \tag{6.4}$$

En cuanto al consumo eléctrico de la Raspberry Pi, sabiendo que el precio del kWh es de 0.102 €/kWh y el consumo de las Raspberry es de 3.2W.

$$3{,}2W \cdot 0{,}102€/\text{kWh}/1000\frac{W}{kW} \cdot 24\frac{h}{día} \cdot 365\frac{días}{año} = 2{,}86€ \tag{6.5}$$

En la tabla 6.12 podemos apreciar dichos costes junto con los costes de mantenimiento, estimados en un 20 % el coste directo de los elementos de la instalación.

Es destacable reseñar como **los costes de mantenimiento son claramente el mayor coste que deberemos enfrentar a lo largo de la vida útil del producto**. En estos costes de mantenimiento hemos incluido diferentes acciones como posibles recalibra-

ciones, el cambio físico de las pilas o reinicios y recalibraciones del sistema.

6.7. Beneficio Obtenido

Una característica típica de las instalaciones domóticas es la **dificultad para estimar el retorno de la inversión realizada en ellas** al tener un gran peso el aumento del confort de los usuarios de la instalación entre las ventajas más destacables, métrica ésta difícilmente cuantificable económicamente.

El aumento de la seguridad, aunque evidente, también es complicado de estimar mediante un valor monetario.

El resto de características asociadas a instalaciones domóticas como puede ser aumento de la eficiencia energética realmente no es aplicable a nuestro proyecto.

Nos encontramos por tanto con una dificultad evidente a la hora de dar un valor económico al beneficio que obtenemos de la instalación año tras año.

Para solventar dicha dificultad se propone el siguiente método: la instalación proyectada permite utilizar la sensorización y la automatización para detectar situaciones peligrosas para los residentes de la instalación. Esta tarea, es necesaria y actualmente es llevada a cabo por personal del Centro. Las características inteligentes de la instalación no permiten reemplazar la actividad humana ya que el foco del sistema está precisamente en servir de alerta temprana para ayudar a dicho personal. Sin embargo, creemos que puede **minimizar la cantidad de personal por turno**, estimando esta reducción de personal en un TCAE (técnico en cuidados auxiliares de enfermería).

Según datos del INE [9], podemos estimar el sueldo anual de un profesional con estas características como 15.000 €anuales. Entenderemos dicho valor como el ahorro interanual conseguido con el sistema planteado, o como el beneficio de la instalación.

6.8. Flujo de Caja

La instalación tiene estimado un ciclo de vida de 5 años, con un reemplazo de la Raspberry todos los componentes que conforman la instalación (lo que supone el reemplazo de los componentes a su valor futuro y 250€de mano de obra).

Hemos tomado una actitud conservadora, manteniendo los ingresos a lo largo del tiempo pero aumentando todos los costes un 3 %, lo que correspondería a un aumento del IPC de este valor y que a su vez significaría un crecimiento de la economía mucho mayor

Cuadro 6.13: Flujo de Caja.

AÑO	Ingresos (€)	Gastos (€)	Flujo de Caja (€)	Flujo Acumulado (€)
0		10259.48	-10259.48	-10259.48
1	15000.0	3933.32	11066.68	807.20
2	15000.0	3934.91	11065.09	11872.30
3	15000.0	3936.54	11063.46	22935.76
4	15000.0	244.32	14755.68	37691.43
5	15000.0	59.63	14940.37	52631.80

Cuadro 6.14: Indicadores Económicos.

Indicador	Valor
TIR	1.08
VAN (€)	49000.40
Q (€)	2.42
PAYBACK (años)	1.00

del previsto.

Con todos los datos que hemos visto en este estudio económico, estamos en posición de mostrar el flujo de caja previsto para nuestra instalación (ver Tabla 6.13).

Como vemos el flujo de caja es muy positivo pero vamos a ayudaros de una serie de indicadores económico-financieros antes de dar una valoración definitiva de estos resultados.

6.9. Indicadores Económico-Financieros

En la tabla 6.14, podemos ver reflejados los diferentes indicadores calculados.

Los resultados obtenidos ponen de manifiesto la **excelente rentabilidad** que supone el proyecto. Al ahorro de costes obtenido, podríamos añadirle el aumento de residentes que podría llevar asociado al verse atraídos por las características del sistema diseñado con lo que el beneficio y rentabilidad seguiría aumentando.

Absolutamente todos los indicadores obtenidos son excelentes. Podríamos llegar a la siguiente conclusión, por cada euro que invertimos, a lo largo de cinco años conseguimos 3,44€. El retorno de la inversión se produce ya en el primer año, consiguiendo más de 50.000€más por dicha inversión de lo que actualmente cualquier banco nos daría en un depósito a plazo fijo.

Conclusiones

El progresivo envejecimiento que durante los últimos años está experimentando la población en los países desarrollados y las perspectivas que ahondan en esta tendencia conforman un escenario con **importantes retos que superar**. En este sentido, la tecnología puede aportar soluciones que faciliten la transición hacia este presente y futuro.

Es precisamente en este sentido donde el uso de la domótica y la inmótica pueden tener más relevancia, sin embargo, su lenta evolución en comparación con otras tecnologías han convertido a estas en tecnologías caras y por tanto reservadas a grandes presupuestos; aunque esta situación parece estar revirtiéndose en los últimos años gracias a la eclosión de **proyectos libres que ponen de manifiesto el interés generalizado en democratizar el acceso al que se conoce como "mundo inteligente"**.

Nuestro proyecto se ha desarrollado en este marco de trabajo, intentando aportar un sistema al Centro Asistencial San Luis de Palencia capaz de **aumentar la seguridad de los pacientes institucionalizados y facilitar las tareas de cuidado a la plantilla del centro**, todo ello con un presupuesto comedido que conviertan a la solución dada en una opción viable para su instalación real.

Para ello hemos tenido que integrar elementos de terceros y un sensor desarrollado "ex profeso". Las herramientas empleadas a lo largo del diseño, así como las tecnologías empleadas en el mismo siguen, casi en su totalidad, licencias "copyleft" y tienen un carácter gratuito y estandarizado que hacen que su **expansión futura no sea problemática a la vez que hemos mantenido los costes reducidos**.

A nivel técnico, creemos haber conseguido buenos resultados en cuanto a fiabilidad y eficiencia en el sistema empleado, exprimiendo sus características y empleando **optimizaciones tanto hardware como software**. Creemos que el caso del sensor construido, encargado de medir el estado de una cama, es especialmente relevante al conseguir unas prestaciones prácticamente idénticas a las obtenidas con el sensor de presencia, que en este caso era comercial.

Un sistema inalámbrico es indudablemente más complejo que su homólogo cableado pero creemos que los resultados obtenidos ponen de manifiesto la **superación de los**

retos intrínsecos a la comunicación sin cables.

Otro punto que nos gustaría destacar es la **flexibilidad y escalabilidad del sistema**, aceptando el mismo sin ninguna modificación adicional más elementos incluso si estos son diferentes a los actuales o usan un protocolo de comunicación diferente siempre que este pertenezca a la **banda de los 433MHz**. Esta característica es debida al empleo de un receptor-transmisor RFXCOM y creemos que su elección ha sido uno de los grandes aciertos del proyecto.

Además, el sistema diseñado ha mostrado cómo enlazarlo con un robot que emplee ROS no es sólo sencillo sino también muy rápido, lo que permite que el sistema diseñado un el robot antropomorfo que use este framework **puedan operar de forma conjunta aumentando sustancialmente las prestaciones de esta solución tecnológica**.

Creemos que el análisis económico ha dejado patente que **no sólo existe una viabilidad técnica sino también económica**. Los costes del proyecto han estado claramente influenciados por el coste de investigación y desarrollo humano, pero **los costes físicos son realmente mínimos**. Además, hemos visto como los costes de mantenimiento a lo largo de su vida útil son también muy reducidos, con lo que los resultados acerca de la viabilidad y conveniencia de llevar a cabo su instalación desde el punto de vista económico son realmente **claros y muy favorables**.

Aunque se ha pretendido dar una visión completa entrando suficientemente en profundidad en cada tema, no cabe duda que han quedado aspectos pendientes para futuras revisiones.

En este sentido, creemos que todos los aspectos relacionados con la **integración del sistema dentro de la instalación** pueden ser tratados con una mayor precisión en futuros estudios, concretamente todo aquello relacionado con la instalación de red actual del Centro Asistencial, sus posibles modificaciones y configuración para adaptarse a las necesidades generales del sistema diseñado y, particularmente a todas aquellas cuestiones donde la **seguridad de la información es la protagonista**, aislando los datos sensibles de accesos no autorizados, ya sea por parte de usuarios de la misma red sin los permisos suficientes o ante ataques externos (cibercriminales).

Por otro lado, creemos que el enfoque que aporta la inclusión de la Raspberry podría ser diferente, empleando la misma exclusivamente como elemento central de comunicación pero no alojando en ella OpenHAB como tal. Creemos que la lógica de **la aplicación podría externalizarse hacia una gestión *"On Cloud"*** donde un proveedor de servicios dote a sus clientes de comunicación con sus sistema OpenHAB (domóticos en general), realizándose el acceso y la configuración del sistema desde su plataforma. Esta idea, puede ser explorada en futuros trabajos y nos llevaría a un entorno donde los elementos físicos en sí, se acercan mucho más al **paradigma del "Internet de las Cosas"**, pudiendo

incluso acabar desarrollando una idea de negocio viable.

Por último, creemos que el éxito o fracaso de la instalación va a estar determinado en gran parte por el mantenimiento que se de a la misma. En este sentido, hemos aportado en los anexos documentación complementaria para conocer el sistema y los elementos que lo componen en profundidad, así como los protocolos a seguir en actuaciones de instalación y mantenimiento. Creemos que su cumplimiento es crítico para asegurar el correcto funcionamiento del sistema a lo largo de su vida útil.

Con todo ello, creemos que **el sistema desarrollado ha sido capaz de cumplir todos los objetivos impuestos, consiguiendo diseñar una solución sencilla en su uso, mantenible a lo largo de su vida útil y que aporta un valor añadido real a la instalación**.

Bibliografía

[1] Arduino - arduinoboardpromini. https://www.arduino.cc/en/Main/ArduinoBoardProMini. (Accessed on 07/10/2016).

[2] Arduino y el modo sleep | tutoriales arduino. http://www.prometec.net/el-modo-sleep-en-arduino/. (Accessed on 07/09/2016).

[3] Arduino y las interrupciones | tutoriales arduino. http://www.prometec.net/interrupciones/. (Accessed on 07/09/2016).

[4] Bed alarms for seniors | bed alarm pads | smart caregiver. http://smartcaregiver.com/fall-prevention/sensor-pads/bed-alarms/. (Accessed on 07/10/2016).

[5] Cresta weather sensor protocol. https://github.com/hjgode/homewatch/blob/master/arduino/libraries/RemoteSensor/docs/CrestaProtocol.pdf. (Accessed on 07/10/2016).

[6] Cron expressions. https://docs.oracle.com/cd/E12058_01/doc/doc.1014/e12030/cron_expressions.htm. (Accessed on 07/15/2016).

[7] Explanation of sitemaps · openhab/openhab wiki. https://github.com/openhab/openhab/wiki/Explanation-of-Sitemaps. (Accessed on 07/16/2016).

[8] Hardware faq · openhab/openhab wiki. https://github.com/openhab/openhab/wiki/Hardware-FAQ. (Accessed on 07/12/2016).

[9] INE componentes del coste laboral - trimestre 1/2016. http://www.ine.es/dyngs/INEbase/es/operacion.htm?c=Estadistica_C&cid=1254736045053&menu=ultiDatos&idp=1254735976596. Accessed: 2016-07-02.

[10] kinetic/installation/ubuntu - ros wiki. http://wiki.ros.org/kinetic/Installation/Ubuntu. (Accessed on 07/16/2016).

[11] Rfxcom binding · openhab/openhab wiki. https://github.com/openhab/openhab/wiki/RFXCOM-Binding. (Accessed on 07/15/2016).

[12] RFXCOM rfxtrx433e usb ha controller. http://www.rfxcom.com/. Accessed: 2016-06-22.

[13] rocketscream/low-power: Low power library for arduino. https://github.com/rocketscream/Low-Power. (Accessed on 07/09/2016).

[14] ROS about. http://www.ros.org/about-ros/. Accessed: 2016-06-22.

[15] tigoe/x10: x10 library for arduino. https://github.com/tigoe/x10. (Accessed on 07/10/2016).

[16] A. M. Adami, M. Pavel, T. L. Hayes, and C. M. Singer. Detection of movement in bed using unobtrusive load cell sensors. *IEEE Transactions on Information Technology in Biomedicine*, 14(2):481–490, 2010. ID: 1.

[17] JOSÉ LUIS FERNÁNDEZ, CLARA PARAPAR, and MIRIAM RUÍZ. El envejecimiento de la población, 2015.

[18] A. Gaddam, S. C. Mukhopadhyay, and G. S. Gupta. Intelligent bed sensor system: Design, expermentation and results. In *Sensors Applications Symposium (SAS), 2010 IEEE*, pages 220–225, 2010. ID: 1.

[19] SECRETARÍA DE ESTADO DE INVESTIGACIÓN DESARROLLO E iNNOVACIÓN. Plan estatal de investigación científica y técnica y de innovación 2013-2016. PDF, 2015.

[20] J. M. Kortelainen, M. van Gils, and J. Pärkkä. Multichannel bed pressure sensor for sleep monitoring. In *2012 Computing in Cardiology*, pages 313–316, 2012. ID: 1.

[21] G. G. Mora, J. M. Kortelainen, E. R. P. Hernández, M. Tenhunen, A. M. Bianchi, and M. O. Méndez. Evaluation of pressure bed sensor for automatic sahs screening. *IEEE Transactions on Instrumentation and Measurement*, 64(7):1935–1943, 2015. ID: 1.

[22] Department of Economic United Nations and Population Division. Social Affairs. World population prospects: The 2015 revision, key findings and advance tables. PDF, 2015.