Universidad Rey Juan Carlos

Escuela Superior de Ciencias Experimentales y Tecnología

Departamento de Tecnología Electrónica

Diseño y desarrollo de sistema distribuido para sensores medioambientales de bajo consumo

Proyecto de Fin de Carrera

— Autor —

Juan Antonio Fabián Simón

— Tutor —

Joaquín Vaquero López

Agradecimientos

Agradezco a mi familia su incondicional apoyo durante toda mi vida, y en particular durante el curso de mis estudios. Sin su cercanía, tolerancia y cariño difícilmente habría llegado a crecer y mirar siempre al frente con optimismo: Isabel, Antonio, Clara y Diego, sois el motor de mi pasión y el infinito combustible de mi alma. Quiero resaltar mi gratitud especial hacia mi hermano Diego, por haberme ayudado a poner en práctica nuestro último proyecto musical, gozando de su paciencia cuando mi trabajo ha sido prioridad ante nuestra música.

Agradezco enormemente a Roberto de la Prieta su compañía, apoyo y motivación desde el primer día que nos conocimos. Mi interés por la electrónica se debe, en gran medida, a las animadas charlas que tuvimos durante nuestros primeros encuentros, cuando yo era su alumno. Como resume la letra de una canción: "faltan sobre mi paleta colores intensos, que reflejen tu rara belleza". Sigo siendo tu alumno.

También debo agradecer la dedicación y el compañerismo que me han brindado muchas personas en el curso de mi carrera profesional. Sería muy extenso nombrarlas a todas, y muy injusto dejarme alguna en el tintero, pero recuerdo con emoción todo lo que he recibido durante mi periplo laboral, en empresas como Compaq Computer España, Necso Entrecanales y Cubiertas, SEPSA y SESA, y en último término, mi propia empresa Monster Electronics.

Muchas son también las personas que, desde la amistad, me han apoyado en los momentos más difíciles, y que han compartido con alegría los momentos más felices. De todas ellas he bebido ingentes tragos de superación, esperanza y calor. A todos vosotros, que sois muchos para nombrar: muchas gracias.

Aunque lejano en mi memoria, guardo un especial recuerdo por dos profesores durante mis estudios de bachillerato en el I.E.S José de Churriguera en Leganés: D. José Manuel Donoso (Física) y Rafael Cerrato (Historia). Sin darse cuenta me ayudaron a fraguar una

mente abierta y crítica frente al destino. Un día, tras un examen, Donoso me espetó una frase que ha mantenido mi ánimo vívido: "No se preocupe por ser un mal estudiante, anímese por ser un gran curioso: sólo quien cuestiona evoluciona". De Cerrato siempre recordaré sus fervorosos comentarios sobre anarquismo, educación libertaria y su interés por desarrollar nuestra curiosidad hacia las diversas culturas del mundo. Quedo agradecido a todas aquellas personas que pudiendo ser gurús, decidieron dar el paso ejemplar de convertirse en verdaderos maestros.

Sería injusto no agradecer también a toda la comunidad relacionada con la cultura y el software libre. Gracias al empeño de estas personas he adquirido, y continúo adquiriendo, muchas habilidades útiles para mi profesión y mi vida. Espero que la ilusión por un futuro libre y equitativo siga brotando de los corazones y las mentes venideras.

Por último, siento inmensa gratitud hacia todas las personas que, de alguna manera u otra, han mostrado su bondad y sonrisa al cruzarse nuestros caminos.

Resumen

Este proyecto explora las capacidades de las redes de sensores inalámbricas, realizando un prototipo funcional sobre el que experimentar. Siguiendo la línea de proyectos del **Departamento de Tecnología Electrónica**[1] de la Universidad Rey Juan Carlos, se plantea el diseño y desarrollo de un sistema integral para la adquisición de datos recogidos por una red inalámbrica de nodos sensores. Interviene también el **Centro Andaluz de Medioambiente** (C.E.A.M.A[2]) en colaboración con el laboratorio **iEcolab**[3], adscrito a la Universidad de Granada, que requiere de una solución de toma y procesado de datos medioambientales. El proyecto se orienta así hacia sistemas distribuidos y autónomos de bajo consumo energético, y en la medida de lo posible, que minimicen las tareas de mantenimiento de los nodos.

Estos nodos están equipados con sensores medioambientales, que permiten realizar medidas localizadas para un entorno ambiental cualquiera. También disponen de un módulo de comunicación inalámbrica, basado en el estándar IEEE 802.15.4 - ZigBee[4], funcionando con topología de malla o *mesh*.

Tras recoger los datos de campo, los nodos sensores envían los datos a un nodo coordinador. Éste se comunica con un sistema informático que los recibe, procesa y almacena en un servidor de bases de datos. La base de datos es accesible desde Internet mediante, por ejemplo, una página web. El nodo coordinador también permite la configuración de parámetros de la red o de los nodos sensores, permitiendo el desarrollo de interfaces gráficas para la configuración de estos parámetros.

El sistema consta de una plataforma *hardware* heterogénea, compuesta por nodos inalámbricos, sistemas empotrados, servidores, dispositivos móviles, etc. También dispone de diferentes piezas *software*, como por ejemplo: versiones de *firmware* específicas para

[1] http://www.gtebim.es
[2] http://www.ceama.es
[3] http://www.iecolab.es
[4] http://standards.ieee.org/getieee802/download/802.15.4-2011.pdf

los distintos nodos inalámbricos, *software* para el equipo que actualiza la base de datos, *software* para el servidor web que sirve la base de datos por Internet, y *software* para la inspección y configuración de la red de sensores.

La Figura 1 muestra una perspectiva global del sistema, en la que unos nodos sensores recogen datos y los envían a un servidor, que hace accesibles estos datos a los usuarios.

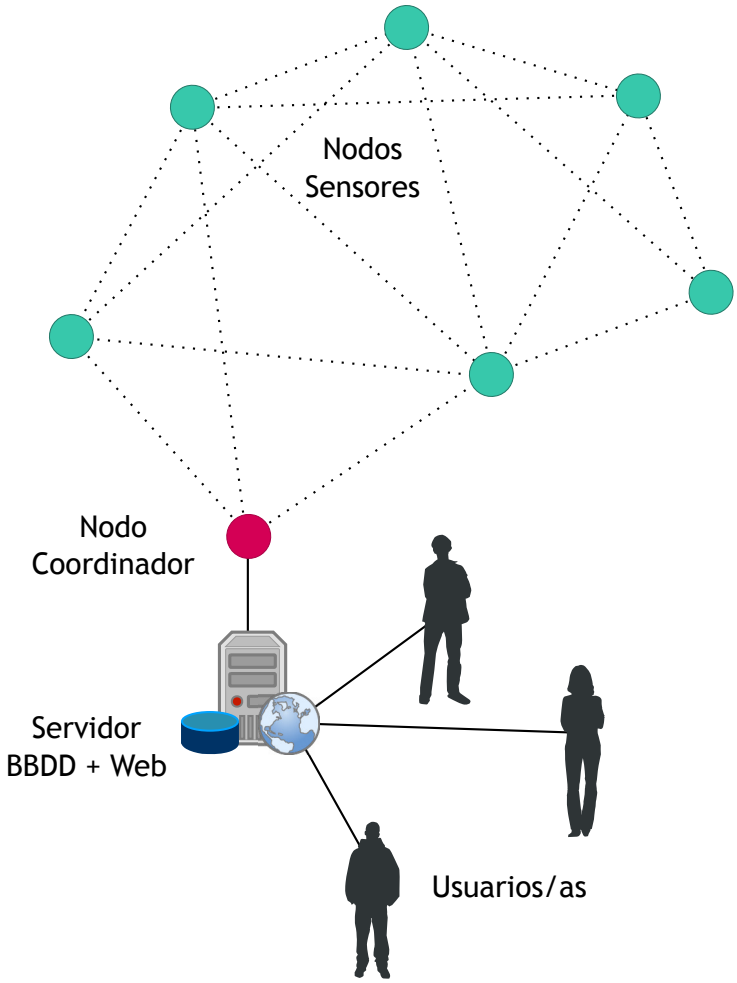

Figura 1: Esquema general del sistema, con sus principales actores: nodos sensores, nodo coordinador, servidor y usuarios.

El proyecto puede emplearse en diversos campos de aplicación, si bien se enmarca dentro de los sectores industriales agrícolas y de supervisión medioambiental. Teniendo en cuenta la alta competitividad existente en el mercado global, y el uso indiscriminado que se hace de los recursos naturales, se plantea el uso de nuevas tecnologías para ayudar a economizar los procesos, requisitos y recursos en estos sectores. Cuando la economía no se entiende estrictamente en el sentido monetario, sino en el de usar los mínimos recursos

para desarrollar una idea, el uso de nuevas tecnologías en diferentes fases del desarrollo pueden ser muy útiles para conseguir estos objetivos.

Para el desarrollo del proyecto se han utilizado herramientas y estándares libres, en la medida de lo posible. Esta decisión permite hacer un sistema fácil de replicar y de mantener, pero también accesible y con posibilidad de mejora. Todo el contenido generado (documentación, código fuente, etc.) se publica bajo licencias libres, por lo que puede ser usado, adaptado y distribuido según las necesidades de cada persona, comunidad o sector.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Motivación

La disponibilidad y el alcance de las nuevas tecnologías de comunicación inalámbrica, hacen posible el despliegue de sistemas distribuidos en casi cualquier entorno, permitiendo optimizar diferentes procesos de manera local, en cada una de sus fases.

Estas nuevas tecnologías, orientadas principalmente a gestionar y optimizar procesos, son bien aceptadas por la sociedad en general, y también por los sectores industrial y medioambiental. Si a esta aceptación se añade el uso de tecnologías y estándares libres, se consigue una buena base para crear y experimentar con tecnologías y procesos propios, a un coste monetario razonable, y favoreciendo la inclusión, adaptación y evolución de estas tecnologías por parte de los sectores mencionados. Conviene destacar que el desarrollo de tecnologías propias y de tejido industrial sólido son propicias para una buena salud económico-monetaria, prácticamente en cualquier sector.

Por otra parte, la aceptación de las comunicaciones inalámbricas por el conjunto de la sociedad, junto con la disponibilidad de bandas de frecuencia libres para uso civil, así como la facilidad para adquirir electrónica certificada, hacen asequible el disponer de tecnologías propias que puedan satisfacer las demandas de los entornos, de los procesos, o de los usuarios potenciales.

Por último, este proyecto aglutina varias disciplinas, e integra diferentes herramientas de *hardware* y *software* para construir un sistema concreto. De esta manera responde a la motivación y objetivos de un plan de estudios para Ingeniero Técnico en Informática de Sistemas, en el que se plantea una visión general de los sistemas presentes en la industria y su aplicación para resolver problemas específicos.

1.2. Descripción general y requisitos del problema

El problema básico planteado es el de construir un sistema de adquisición y supervisión de variables ambientales, adaptable a diversos entornos físicos.

El sistema se compone de elementos sensores inalámbricos conectados en red, que toman medidas para recabar información medioambiental. Esta información debe ser almacenada para su acceso histórico, y debe ser accesible desde diferentes medios (haciendo dicha información disponible a redes de ordenadores como Internet).

Cada uno de estos nodos sensores es gobernado por un microcontrolador, que puede gestionar en tiempo real tanto los sensores como las comunicaciones que le permiten formar parte de la red de nodos.

Los nodos sensores deben minimizar el consumo energético, con la finalidad de hacerlos lo más autónomos posible, reduciendo el mantenimiento, tanto para ampliar la red y el área que abarca, como el mantenimiento por parte de personal técnico.

Tal y como se mencionaba en el apartado anterior, otro requisito de la solución implica el uso de arquitecturas y herramientas libres, que permitan el reemplazo de cualquier parte, la mejora continua y la adaptación a diferentes necesidades y escenarios.

1.3. Estado del arte

En esta sección se analiza el estado del arte para las diferentes piezas requeridas por este proyecto. No se describen exhaustivamente las tecnologías existentes, y en el caso de los diferentes productos considerados, se analizan con más detalle en la sección 3.1.

1.3.1. Microcontroladores

Desde hace muchos años las arquitecturas basadas en microcontrolador se utilizan de forma masiva en casi todos los dispositivos electrónicos. Con la consolidación de estas arquitecturas, y dada su popularización y disponibilidad, han emergido comunidades de usuarios y desarrolladores en torno a ellas durante los últimos años. La labor de estas comunidades ha sido la de facilitar el acceso a la información, y han creado productos que comparten los mismos niveles de calidad y diseño presentes en los productos propietarios no libres. Los usuarios de estas arquitecturas libres son capaces de adaptar las soluciones a

sus necesidades para resolver problemas concretos, permitiendo que los microcontroladores se conviertan en herramientas de desarrollo comunes para programadores y aprendices.

Para la arquitectura de microcontrolador, se pueden encontrar diversos productos que ofrece el mercado a precios similares. Hay empresas como Atmel, Texas Instruments, Freescale o Microchip, que desarrollan productos sobre los que han emergido plataformas libres específicas.

1.3.2. Tecnologías inalámbricas

En referencia al uso de tecnologías inalámbricas, existen numerosas soluciones ya implantadas en el mercado, que ofrecen conectividad a distintos niveles mediante diferentes medios. Existen tecnologías inalámbricas que hacen uso, por ejemplo, de ondas de radio, de microondas, de infrarrojos, de ultrasonidos o de inducción electromagnética. En cuanto al radio de alcance de dichas redes, aplicables a este proyecto, existen soluciones para campo cercano, como pueden ser Bluetooth, ZigBee, Wireless-USB, CyFi, por mencionar algunas.

Estas tecnologías están siendo desarrolladas por multitud de fabricantes, que cada vez están más sensibilizados con el requisito de bajo consumo energético. Para la tecnología ZigBee, en particular, existen fabricantes como Digi International (Maxstream), Microchip Technology., Freescale o Motorola, entre otras. Algunos de estos fabricantes proveen además productos basados en microcontroladores que admiten la integración de tecnologías inalámbricas, propias o de terceros, orientadas al bajo consumo energético.

1.3.3. Dispositivos sensores

Los dispositivos sensores, capaces de medir las propiedades del entorno, han evolucionado mucho durante los últimos años, y han aparecido numerosos productos y empresas que los desarrollan. A menudo existen multitud de tecnologías y soluciones diferentes que sirven a un mismo campo de aplicación. Este proyecto requiere la medición de temperatura, humedad relativa, luz ambiental y presencia de gases contaminantes.

Temperatura

En cuanto a la medición de temperatura, existen diferentes tecnologías basadas en termopares, termorresistencias, termistores o sistemas de dilatación, entre otros. Existen soluciones de fácil adaptación a un proyecto basado en microcontroladores, con un consumo energético moderado y que no requieran de mucha electrónica adicional. Hay fa-

bricantes como Sensirion, Honeywell, Microchip, Philips o Texas Instruments, que ofrecen productos adaptables a este proyecto.

Humedad relativa

Para medir la humedad relativa del ambiente se pueden encontrar tecnologías capacitivas, resistivas y de infrarrojos, entre otras. Hay productos de fabricantes que las implementan, como Honeywell, Sensirion, Parallax o TDK, entre otras. Dentro de la amplia oferta, resultan de interés las soluciones más integradas, que requieran de la mínima electrónica adicional, y con el menor consumo energético posible.

Luz ambiente

Los sensores fotoeléctricos son aquellos sensibles a los cambios de iluminación ambiente. Existen múltiples tipos, que hacen uso de fotodiodos, fototransistores e infrarrojos, entre otros. Entre los fabricantes que ofrecen productos adecuados para el proyecto se encuentran, por ejemplo, Sharp, TAOS o Intersil.

Gases contaminantes

También existen múltiples mecanismos para medir la presencia de gases en el aire, algunos aplicables a este proyecto, como sensores de semiconductores, que suelen ser fáciles de integrar, no suelen tener un elevado consumo eléctrico y están disponibles a precios asequibles. Hay fabricantes como Honeywell, Figaro o Parallax que ofrecen productos de interés a este proyecto.

1.3.4. Soluciones integrales

Existen también fabricantes e integradores que desarrollan soluciones programables más completas, como es el caso de la empresa española Libelium Comunicaciones Distribuidas S.L., que oferta productos como Waspmote o Meshlium. Estas plataformas basadas en microcontrolador y que emplean diferentes tecnologías inalámbricas, permiten el uso de diferentes módulos sensores que ofrece la misma compañía.

Hay diversos autores que han investigado sobre soluciones similares [Bha10] [Moh10] [Ana09], si bien para este proyecto se ha tomado como referencia el Proyecto de Fin de Máster realizado en el Departamento de Tecnología Electrónica de la URJC por Cristina Avelli [Ave10]. En él se introducen con más detalles algunas de las tecnologías inalámbricas y de sensores referenciados en este proyecto.

1.3.5. Sistemas informáticos

En cuanto al sistema informático, que recoge los datos de sensores y permite el acceso a sus registros, existen soluciones que aprovechan la popularidad del *Cloud Computing*. Un ejemplo es la compañía Xively[1], que recoge datos de sensores y los pone a disposición pública, ambas acciones a través de Internet. También existen alternativas a este servicio, como es el que ofrece Sen.se desde su plataforma Open.Sen.se[2]. Ambas plataformas están o han estado relacionadas con el uso de la plataforma de *hardware* libre Arduino.

Para la implementación del sistema informático requerido para el desarrollo de este proyecto, pueden emplearse diferentes tecnologías de sistemas operativos y sistemas gestores de bases de datos.

En cuanto a los sistemas operativos, y sin considerar la arquitectura *hardware* sobre la que se instalan, se puede elegir soluciones libres como por ejemplo GNU/Linux, Haiku, Plan 9 o FreeBSD. Si se desea seguir el ejemplo de iniciativas como Xively o Open.Sen.se, se pueden utilizar máquinas virtuales para crear una nube local, o bien se pueden contratar servidores y/o servicios a terceras partes.

Por último, existen diversos sistemas gestores de bases de datos libres como pueden ser MySQL, PostgreSQL, SQLite o HSQLDB. También se pueden hacer desarrollos con tecnología NoSQL, cada vez más extendida y orientada a grandes volúmenes de datos, por ejemplo con sistemas como MongoDB.

[1] https://xively.com
[2] http://open.sen.se

Capítulo 2

Objetivos

2.1. Objetivo general

El objetivo general es el de diseñar y desarrollar toda la infraestructura necesaria para implantar una red de sensores medioambientales, y un sistema de información que permita acceder a todos los datos recogidos por los mismos. Con este desarrollo se pretende crear una base funcional de *software* y *hardware* sobre la que construir soluciones futuras.

2.2. Objetivos operativos

2.2.1. Desarrollar nodos inalámbricos

Cada nodo de la red debe recabar información acerca de la temperatura, la humedad relativa, el nivel de luz ambiente, y la presencia de gases contaminantes en el aire, como hidrógeno, monóxido de carbono, metano u otros. Para llevar a cabo esta tarea se cuenta con sensores de diferentes fabricantes.

2.2.2. Crear una red de sensores

Cada nodo sensor, junto a otros nodos sensores, conforman una red de sensores con topología de malla. Usando esta topología se pretende obtener una red que sea fácilmente ampliable, que pueda ser desplegada en entornos heterogéneos. Dicha capacidad de ampliación se refiere tanto a la facilidad de añadir nuevos nodos a la red, como a la facilidad para ampliar el área física en la que se despliega la solución, permitiendo que el área abarcada crezca en el futuro de la forma más transparente posible, y con el mínimo mantenimiento por parte de personal técnico.

2.2.3. Optimizar el consumo energético

Es también preciso obtener un bajo consumo energético de los nodos sensores, de tal forma que puedan estar alimentados mediante baterías en el futuro. Para ello debe establecerse un protocolo de actividad que regule y adecue tanto la frecuencia de adquisiciones sensoriales como la frecuencia en las comunicaciones.

2.2.4. Centralizar la información

La red de sensores envía a un servidor toda la información que genera. Este servidor debe realizar varias tareas complementarias. Por una parte aloja e implementa una base de datos con la información recogida por cada nodo sensor. Asimismo debe ejecutar en segundo plano un *software* que procese y almacene convenientemente dicha información en la base de datos.

2.2.5. Desarrollar *software* de acceso a la información

Paralelamente, este servidor debe ofrecer un mecanismo de acceso a la base de datos a través de tecnologías web. A través de estas herramientas web se permite la visualización de los nodos sensores desde diversos dispositivos, bien sean terminales informáticas de escritorio, o dispositivos móviles como teléfonos o *Tablet PC's*.

2.3. Objetivos personales

Es una buena oportunidad para evaluar las diferentes soluciones que ofrece el mercado, y de integrarlas para resolver un problema en concreto. Permite también la profundización en múltiples tecnologías y lenguajes de programación, así como el uso de diversas herramientas de *software* y *hardware* libre para llevarlas a cabo.

En último término, resulta muy satisfactorio terminar los estudios de ingeniería que, por motivos laborales, han sido intermitentemente interrumpidos y reanudados a lo largo de los años.

Capítulo 3

Materiales y métodos utilizados

Este capítulo describe los materiales y herramientas utilizados para la realización del proyecto. También se especifican las metodologías de desarrollo empleadas y se aborda la planificación seguida. Debido a la multitud de tecnologías empleadas, no se describen ni comparan exhaustivamente las alternativas existentes para las herramientas y las metodologías elegidas, sino que se enuncian los motivos relevantes de cada elección.

3.1. Materiales y herramientas

El conjunto de herramientas de *software* y *hardware* elegidas responden principalmente a tres criterios básicos: que sean productos con licencias libres, que sean gratis o asequibles, y que dispongan de una comunidad activa que garantice su continuidad.

3.1.1. Productos *hardware*

Placa Arduino

La arquitectura *hardware* basada en microcontrolador permite manejar los sensores y las comunicaciones de cada nodo sensor. Hay multitud de arquitecturas libres, disponibles actualmente, que pueden servir al proyecto. Se comercializan como PCB[1] listos para usar y entre ellas se pueden destacar:

- **Arduino[2]:** es una plataforma de *hardware* y *software* libre que permite programar, de una forma sencilla y estandarizada, algunos microcontroladores ATmega 8bit de Atmel[3]. Además proporciona un lenguaje de programación con el que se generan los programas que ejecuta la mencionada plataforma *hardware* (a estos programas para microcontroladores se les denomina *firmware*).

[1] PCB o placa de circuito impreso, del inglés *Printed Circuit Board*
[2] http://www.arduino.cc
[3] http://www.atmel.com

- **Pingüino**[4]**:** utiliza los microcontroladores PIC de Microchip de 8bit y 32Bit, más potentes[5] que Arduino. También provee un entorno de desarrollo y un lenguaje bastante compatible con Arduino. El precio de esta plataforma es ligeramente inferior al de Arduino.

- **Teensy**[6]**:** al igual que Arduino se basa en microcontroladores de Atmel. Ofrece herramientas de desarrollo similares a Arduino, y el precio es similar a Arduino.

- **mbed**[7]**:** es una plataforma que utiliza microcontroladores con arquitectura ARM de 32bit, más potente que las anteriores, y capaz de ejecutar sistemas operativos. Ofrece un amplio conjunto de herramientas de desarrollo, incluso permite editar y compilar código desde Internet y descargarlo directamente al *hardware*. Su precio es superior al de Arduino, y también su consumo eléctrico, si bien se trata de una plataforma con *hardware* más potente.

- **BeagleBone**[8]**:** esta plataforma también se basa en ARM, y tiene una creciente aceptación entre la comunidad, por lo cual se desarrollan periféricos y se publican bastantes proyectos. Es la más potente de las alternativas, y su precio es similar al de *mbed*.

Se ha elegido la plataforma Arduino porque consta de tres pilares básicos que siguen los planteamientos del proyecto: es una placa de *hardware* libre, ofrece herramientas libres multi-plataforma y consta de una excelente comunidad de usuarios y desarrolladores. También es una plataforma muy extendida, que lleva varios años disponible en el mercado a un precio asequible. Es por ello que esta plataforma está extensamente probada y depurada, y ofrece un buen diseño de referencia, a partir del cual se pueden realizar prototipos.

Para los nodos sensores (llamados *endpoints* o *end-devices*) se ha seleccionado el modelo **Arduino UNO R3**[9], que incorpora un microcontrolador ATmega 328, con 32kbytes de memoria *flash* de programa, 2kbytes de memoria SRAM, 1kbytes de memoria EEPROM, una frecuencia de reloj de 16MHz y un puerto para comunicación serie UART TTL 5Vdc/USB. A través de este puerto UART se comunican con los módulos de comunicación

[4]http://www.pinguino.cc
[5]Potencia entendida en el sentido de capacidad de proceso, frecuencia de trabajo del microprocesador, cantidad de memoria, cantidad y utilidad de los puertos de entrada/salida.
[6]http://www.pjrc.com/teensy
[7]http://mbed.org
[8]http://beagleboard.org/bone
[9]http://arduino.cc/en/Main/ArduinoBoardUno

inalámbrica.

Para el nodo coordinador se ha preferido el modelo **Arduino Leonardo**[10], que cuenta con el microcontrolador ATmega 32u4, con 32kbytes de memoria *flash* de programa, 2.5kbytes de memoria SRAM, 1kbyte de memoria EEPROM, una frecuencia de reloj de 16MHz y dos puertos serie: un puerto para comunicación serie UART TTL 5Vdc y otro para la comunicación serie por USB (CDC)[11]. Este segundo puerto serie CDC se utiliza para comunicarse con el ordenador/servidor.

Módulo de comunicaciones ZigBee

Para crear la red de nodos sensores planteada, el microcontrolador debe ser capaz de poder enviar y recibir información de forma inalámbrica, y para ello necesita estar conectado a un módulo de comunicaciones inalámbricas.

La tecnología inalámbrica elegida para el proyecto ha sido ZigBee, principalmente por su bajo consumo y por su idoneidad para la aplicación requerida. Existen diversas soluciones para dotar a un sistema de conectividad inalámbrica ZigBee, basada en el estándar IEEE 802.15.4. Para el proyecto se necesita un módulo de comunicaciones inalámbrico con certificación europea y compatible con ZigBee. Este módulo debe integrarse fácilmente con Arduino y sus herramientas, o al menos, ser de fácil adaptación. Entre las opciones que ofrece el mercado se pueden encontrar:

- Digi International: dispone de una variada oferta módulos certificados, y ofrece protocolos propietarios de comunicación y control que permiten configurar las características de la red inalámbrica. También cuenta con *software* de configuración para los módulos. De las diferentes familias y series que ofrece el fabricante, para el proyecto resulta de interés la familia XBee XB24-BxIT.

- Microchip: ofrece módulos certificados y, al igual que los módulos XBee, este fabricante posee un protocolo propietario de comunicación y control llamado MiWi, para el que ofrece herramientas y documentación. Se ha contemplado el módulo MRF24J40MA, cuya potencia de emisión es ligeramente inferior a la de los módulos XBee, aunque para grandes volúmenes resulta bastante más barato.

- Murata: también ofrece módulos certificados, si bien no ofrece herramientas para el desarrollo. Por ejemplo el módulo SN3020 ofrece más potencia de señal que los

[10] http://www.arduino.cc/en/Main/ArduinoBoardLeonardo
[11] http://en.wikipedia.org/wiki/USB_communications_device_class

anteriores, a un precio similar al de los XBee.

Los módulos de comunicaciones elegidos son los XBee XB24-BPIT-004[12] de la serie 2 de Digi International.

Al igual que el resto de módulos comparados, operan en la banda de frecuencia de 2.4GHz, con una frecuencia de envío y recepción de datos de hasta 250kbps, un alcance (en línea visual) de hasta 100m y una potencia de salida de hasta 63mW. Se ha elegido la serie 2 pues mejora a los de la serie 1 tanto en potencia de emisión como en capacidades de optimización del consumo eléctrico. Poseen un puerto serie UART TTL 3Vdc a través del cual se comunican con Arduino. Otro de los motivos para elegir estos módulos, es que son compatibles con algunas placas de expansión para Arduino. A estas placas de expansión, se las conoce con el nombre de escudos o *shields*.

Wireless SD Shield para Arduino

Como se ha comentado, el módulo de comunicaciones inalámbrica debe conectarse al microcontrolador. Para utilizar el módulo de comunicaciones XBee junto a Arduino, se ha elegido el escudo o *shield* **Wireless SD**[13] para Arduino, ya que facilita el desarrollo con tecnologías inalámbricas. Incorpora además un zócalo para tarjetas de memoria µSD, y dispone de un espacio en la PCB[14] para poder realizar prototipos, soldando en ella los componentes deseados.

Sensor de humedad y temperatura

Para la medición de humedad y temperatura se emplean sensores que deben conectarse con Arduino y el sensor elegido es el **SHT11**[15] de Sensirion. Es capaz de medir la humedad relativa en el rango de 0 % al 100 %, con un consumo de 80µW. También mide la temperatura para el rango -40°C a +125°C.

A pesar de ser más caro que otras alternativas, se trata de un sensor con unas características de consumo y resolución muy apropiadas para el proyecto. Viene muy bien calibrado de fábrica, característica que no se encuentra entre todas sus alternativas, es fácil de manejar y existen multitud de ejemplos de código fuente con implementaciones de referencia.

[12] http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/zigbee-mesh-module/xbee-digimesh-2-4

[13] http://arduino.cc/en/Main/ArduinoWirelessShield

[14] PCB, Placa de Circuito Impreso, del inglés *Printed Circuit Board*

[15] http://www.sensirion.com/en/products/humidity-temperature/humidity-sensor-sht11

Sensor de gases

Para medir el nivel de gases contaminantes en el aire, también debe conectarse un sensor con Arduino. Se ha utilizado el sensor **TGS-2600**[16] de Figaro, que presenta una alta sensibilidad para bajas concentraciones de gases contaminantes, como pueden ser el hidrógeno o el monóxido de carbono. Es capaz de detectar hidrógeno en el rango de 0ppm[17] a 30ppm, y debido a su reducido tamaño, integra un calentador para adaptar la señal, que requiere de tan sólo 42mA a 5Vdc.

Sensor fotoeléctrico (luz ambiental)

Para medir la cantidad de luz ambiente se ha escogido el sensor **TSL-2561**[18] de TAOS (compañía perteneciente a AMS AG). Este sensor se aproxima bastante a la sensibilidad del ojo humano, operando en un rango dinámico de entre 0,1Lux y 40.000Lux. A través de su salida digital por bus I^2C, ofrece dos canales de medida, uno para luz infrarroja y otro para luz en el espectro visible, con una resolución de 16bits en ambos. Se trata de un sensor de bajo consumo: unos 0.5mA en funcionamiento y menos de 15µA en reposo.

Otros componentes

Generalmente un sensor, un módulo de comunicaciones, u otros periféricos, no pueden conectarse directamente a un microcontrolador, sino que necesitan de ciertos componentes electrónicos para su correcto funcionamiento.

Para adecuar las señales de los sensores se han utilizado diversos componentes como resistencias, condensadores y transistores, que se verán más en detalle cuando se hable del esquema electrónico diseñado en la sección 4.1.1.

Sistemas informáticos

Los datos que generan los nodos sensores deberán almacenarse en un servidor central, que permita a los usuarios del sistema el acceso a estos datos. El servidor no es más que un ordenador con una tarea dedicada, y en este proyecto, se trata de un servidor virtual contratado a una empresa de alojamiento o *hosting*. Este servidor consta de un procesador AMD64 de 2 núcleos a 2.20GHz, con una memoria RAM de 1GB y un disco duro de 2TB.

[16] http://www.figarosensor.com/products/2600pdf.pdf
[17] Partes por millón
[18] http://www.ams.com/eng/Products/Light-Sensors/Light-to-Digital/TSL2561

Por otra parte, la mayoría del trabajo para este proyecto ha sido desarrollado con un ordenador Apple MacBook Pro 13", con sistema operativo OSX Lion. También se ha utilizado un ordenador de sobremesa (de arquitectura *amd64*) con sistema operativo Ubuntu 12.04 para algunas tareas puntuales.

3.1.2. Entornos de desarrollo, lenguajes de programación y productos *software*

Lenguaje Arduino

El *firmware* de los nodos sensores se ha desarrollado con la plataforma libre **Arduino**. Para programar en este lenguaje se usa el IDE[19] que esta plataforma ofrece, y que a bajo nivel emplea el compilador libre **avr-gcc**, mantenido por Atmel, junto al resto de herramientas típicas, como enlazador, ensamblador, etc. Para la elección de esta plataforma se han tenido en cuenta los siguientes aspectos:

- Ofrece un lenguaje propio fácil de aprender, ya que es un subconjunto de C++.
- Ofrece una amplio número de bibliotecas ya listas para su uso, que abstraen el *hardware* subyacente, y que son fácilmente modificables.
- Posee una comunidad de desarrolladores y usuarios muy activa, por lo que existen muchos recursos reutilizables o adaptables.
- Como diseño de referencia, y plataforma para hacer prototipos, tiene un precio asequible y su disponibilidad en el mercado es alta.

Conviene destacar que el IDE de Arduino es multi-plataforma, y solventa los inconvenientes de portabilidad inherentes al lenguaje C/C++ y a la arquitectura sobre la que se compilan los programas en este lenguaje.

X-CTU

Teniendo en cuenta las peculiaridades de una red de nodos sensores inalámbricos, se identifican dos perfiles diferentes para los nodos sensores: uno para un nodo sensor propiamente dicho (o *endpoint*) y otro para un nodo coordinador. Este nodo coordinador será el que reciba los datos del resto de nodos sensores y los transfiera a un sistema informático o servidor central.

[19]IDE, Entorno Integrado de Desarrollo, del inglés *Integrated Development Environment*

En el caso de los módulos de XBee elegidos, el fabricante ofrece una herramienta de configuración para programarlos. Es *software* propietario, aunque gratuito, de Digi International Inc., y sólo ejecuta en sistemas Microsoft Windows.

Como los módulos XBee son de la serie 2, cada perfil (nodo sensor y nodo coordinador) necesita que el módulo XBee en cuestión sea programado con un *firmware* diferente, que es proporcionado por el fabricante. Estos módulos requieren, por tanto, de un firmware específico para los módulos que funcionen en nodos sensores y otro específico para el módulo XBee del nodo coordinador.

KiCAD

El diseño del circuito electrónico necesario para adecuar las salidas de los sensores a Arduino, se ha realizado con el *software* de CAD[20] electrónico **KiCAD**[21]. Se trata de un programa libre desarrollado desde hace varios años en torno a una comunidad activa y creciente. Es fácil de usar y permite exportar los diseños electrónicos a formatos aptos para su fabricación industrial, lo cual cumple con los requisitos del proyecto.

Processing y Eclipse

Una vez los datos de los nodos sensores llegan al nodo coordinador, éste debe procesarlos y almacenarlos en la base de datos.

El *software* encargado de recibir los datos de los sensores, procesarlos y almacenarlos en la base de datos, está escrito en lenguaje **Java**. Para el desarrollo del mismo se ha utilizado el IDE **Eclipse Juno**[22], y también se han aprovechado las capacidades que ofrece el lenguaje **Processing**[23]. Hay varios motivos que justifican esta elección:

- Java es un lenguaje muy maduro con la capacidad de ejecutar en cualquier plataforma gracias a una versátil máquina virtual (JVM).
- Eclipse es un IDE muy extendido y fácil de usar, con gran cantidad de recursos disponibles en Internet.
- Processing es un conjunto de bibliotecas escritas en Java que facilitan y simplifican el manejo del puerto serie y la visualización gráfica de datos.

[20] CAD, Diseño Asistido por Ordenador, del inglés *Computer Assisted Design*
[21] http://www.kicad-pcb.org
[22] http://www.eclipse.org
[23] http://www.processing.org

Aunque Processing ofrece su propio IDE, las limitaciones del mismo han sido determinantes para elegir otro entorno como Eclipse.

Debian GNU/Linux 6.0

El servidor necesita un sistema operativo sobre el que ejecutar el resto de programas y servicios necesarios, como puede ser una base de datos donde almacenar las lecturas de los sensores.

De los sistemas operativos mencionados en la revisión del estado del arte, en la sección 1.3, se ha seleccionado Debian ya que es un sistema conocido y utilizado por el autor del proyecto.

El servidor ejecuta el sistema operativo **Debian 6.0**[24], y ha sido elegido por ser un proyecto longevo muy estable y con una amplia comunidad de desarrolladores y usuarios. Su sistema de paquetes simplifica y agiliza la instalación y actualización de *software* de terceros, y permite una cómoda administración del sistema.

MySQL Server, MySQL Client y MySQL Workbench

Cuando el nodo coordinador envía los datos al servidor, este debe almacenarlos de forma organizada y estructurada, de tal forma que se garantice la facilidad de acceso a los datos en el futuro.

De todas las alternativas enunciadas en la sección 1.3 se ha elegido MySQL, de tal manera que el sistema operativo se ejecuta un sistema gestor de bases de datos **MySQL 5.1**[25]. El *software* escrito en Java mencionado anteriormente hace uso de bibliotecas clientes para este servidor (MySQL Client). Los motivos principales para la elección de este sistema son:

- MySQL es un proyecto robusto, con una buena capacidad de personalización y configuración, así como con capacidad de escalar para grandes volúmenes de datos y usuarios (mediante el uso de *clusters*).
- Es un proyecto que goza además de gran popularidad y aceptación empresarial.
- Permite el uso de diferentes motores que respondan a los diferentes requisitos del modelo de datos y su almacenamiento[26].

[24] http://www.debian.org
[25] http://www.mysql.com
[26] http://dev.mysql.com/doc/refman/5.0/es/storage-engines.html

Para la creación, edición y supervisión de la base de datos, se ha hecho uso de la herramienta **MySQL Workbench 5.2**[27] proporcionada por MySQL.

Debido a la adquisición de la empresa desarrolladora de MySQL por parte de la compañía **Oracle**, cabe mencionar que, si bien se especula últimamente sobre la continuidad del proyecto como *software* libre, existen alternativas como **MariaDB**[28] compatibles[29] con MySQL y mantenidas por la comunidad. Estas alternativas garantizan la validez futura de la elección.

Nginx HTTP Server

Para poder atender las peticiones de usuarios desde Internet, el servidor debe disponer de un servicio de acceso HTTP, que atienda a dichas peticiones.

El acceso a la información, almacenada en la base de datos, por parte de los usuarios del sistema se realiza a través de tecnologías web. Para ello el sistema operativo ejecuta el servidor web **Nginx**[30] que aloja la web de consultas del sistema. La versión del servidor empleada es la 1.2.7.

HMTL5, CSS3, JavaScript, jQuery, PHP

La página web desde la que se consultan los datos de los sensores ha sido desarrollada integrando diferentes lenguajes, como son **HTML5**, **CSS3**, **JavaScript** y **PHP**. Para la parte de JavaScript se ha hecho uso de las bibliotecas jQuery, jQuery-UI y Datatables, que ofrecen facilidades para la gestión dinámica de datos.

Otros

Para la generación y manipulación de imágenes y gráficos se han usado dos programas: el *software* de manipulación de imágenes **Gimp**[31] y el *software* de dibujo vectorial **Inkscape**[32]. Algunos gráficos de dispositivos electrónicos han sido elaborados con **Fritzing**[33].

Para editar el código de la parte web (y algunos scripts) se ha utilizado el editor de textos **jEdit**[34].

[27] http://www.mysql.com/products/workbench
[28] http://www.mariadb.org
[29] https://kb.askmonty.org/en/mariadb-versus-mysql-compatibility
[30] http://nginx.org
[31] http://www.gimp.org
[32] http://inkscape.org
[33] http://fritzing.org
[34] http://www.jedit.org

Para la escritura de la memoria se ha utilizado LaTeX[35] , concretamente el entorno **MacTeX**[36].

Para ejecutar *software* que sólo funciona en sistemas Windows, se ha utilizado el sistema de máquina virtual de **VirtualBox**[37].

La mayoría de accesos al servidor se han realizado mediante SSH y sFTP, empleando los programas de emulación de terminal y acceso SSH para OSX **iTerm 2**[38] y el cliente de FTP y sFTP **Filezilla**[39].

Para generar y gestionar la planificación temporal y de recursos necesarios para el proyecto, se ha utilizado la herramienta **GanttProject**[40].

3.2. Metodología y planificación

Una vez se han acotado tanto el problema como sus objetivos, y tras seleccionar y evaluar los componentes que lo conforman, se pueden establecer los requisitos fundamentales que debe satisfacer la solución, y se puede definir la planificación a seguir para la resolución del problema.

Los requisitos que debe cumplir el proyecto son:

- **Solución integral.** La solución debe cubrir todas las áreas necesarias para el despliegue de un sistema funcional, que pueda ser implantado de forma inmediata por los agentes que lo requieren (URJC y iEcolab).
- **Bajo consumo energético**. La parte *hardware* debe hacer lo posible por minimizar el consumo energético, empleando la lógica electrónica necesaria.
- **Capacidad de ampliación y mejora**. Debe proveer de una base de conocimiento sólida y bien documentada, con el objetivo de su posterior mejora y adaptación a diferentes campos.

[35] http://www.latex-project.org
[36] http://tug.org/mactex
[37] https://www.virtualbox.org
[38] http://www.iterm2.com
[39] https://filezilla-project.org
[40] http://www.ganttproject.biz

- **Uso de estándares libres**. Para garantizar la accesibilidad futura al sistema, se hará un uso premeditado de soluciones libres en tantos componentes como sea posible.

- **Modularidad**. Las diversas piezas que conformen el sistema deben poder ser reemplazadas, o readaptadas, ante diferentes escenarios o circunstancias cambiantes. Para ello el diseño debe ser modular en su conjunto, y en el caso del código fuente, cada pieza debe estar lo más desacoplada posible.

3.2.1. Metodología

En cuanto a la planificación, para la implementación de este proyecto, tanto de la parte *hardware* como *software*, se ha tomado como base un modelo de desarrollo en espiral. Junto a este modelo se emplean también otras metodologías, que según la experiencia del autor, responden mejor al desarrollo de *software* y *hardware* en el ámbito industrial. Es por ello, que también se han aplicado algunos preceptos de las metodologías ágiles de desarrollo, con la intención de paliar los inconvenientes inherentes al desarrollo en espiral.

Según los modelos de desarrollo en espiral y las metodologías ágiles, se divide el problema en tareas más sencillas que aborden la solución dependiendo de las tareas ya completadas. Al tener en cuenta el trabajo realizado en la tarea anterior, este modelo permite una adaptación flexible ante posibles cambios, como pueden ser mejoras o situaciones imprevistas. Para cada tarea, una iteración en la espiral debe contemplar:

- Análisis de requisitos u objetivos.

- Diseño e implementación, teniendo en cuenta las alternativas y los riesgos asociados a cada una.

- Verificación y pruebas de los resultados de la tarea.

- Planificación de la siguiente etapa.

La inclusión de metodologías ágiles aplicadas a este proyecto, impone que en cada iteración debe existir un producto o prototipo funcional y que pueda ser entregado, de tal manera que cada iteración pueda ser evaluada desde diferentes puntos de vista complementarios. Estos puntos de vista hacen converger el enfoque del desarrollador junto al de usuario, o al posible personal técnico a cargo del mantenimiento de la red de sensores. Estas metodologías favorecen una verificación exhaustiva de cada etapa o iteración en la espiral, con la finalidad de desarrollar *software* robusto y accesible desde el principio.

3.2.2. Planificación del desarrollo

La planificación conlleva la distinción de las siguientes fases en el desarrollo:

- **Fase 0:** Diseño de alto nivel y planificación del desarrollo.
  - Estudio de los diferentes productos de microcontrolador bajo licencias libres.
  - Evaluación de los diferentes sensores aplicables al proyecto.
  - Documentación sobre productos finales, libres o propietarios, que resuelven el problema, total o parcialmente.
  - Lecturas sobre electrónica de bajo consumo.
  - Identificación de lenguajes y herramientas *software* y *hardware* utilizables para el desarrollo.
  - Esbozo esquemático de la arquitectura de todo el sistema.
  - División de tareas y planificación temporal.
- **Fase 1:** Estudio de la plataforma *hardware*.
  - Documentación y aprendizaje de la plataforma Arduino (*hardware* y *software*) y sobre los microcontroladores de Atmel.
  - Documentación de la tecnología Zigbee, y de los productos XBee de la serie 2.
  - Estudio y documentación de los sensores, y recopilación del arte previo en torno a ellos.
  - Adquisición de los materiales necesarios.
- **Fase 2:** Prototipo *hardware* de los nodos de la red inalámbrica.
  - Estudio de los diferentes interfaces de comunicación de los sensores.
  - Definición del esquema electrónico.
  - Realización de prototipos con Arduino, módulos Xbee y sensores.
- **Fase 3:** Estudio de la plataforma *software*.
  - Profundización en el lenguaje Java y en el entorno Eclipse.
  - Estudio y aprendizaje de Processing: capacidades de visualización de datos y manejo del puerto serie en un ordenador.
  - Diseño y desarrollo de prototipos *software*.

  - Definición de las clases y los tipos de datos del *software*.

- **Fase 4:** Preparación del servidor.

  - Contrato de un proveedor de alojamiento para el servidor.
  - Estudio del sistema Debian 6.0 e instalación del mismo en servidor virtual.
  - Configuración general del sistema operativo y sistemas de acceso de usuarios.
  - Configuración de Nginx, bibliotecas PHP y MySQL.
  - Creación de *scripts* necesarios.

- **Fase 5:** Especificación de los protocolos de comunicaciones.

  - Diseño del protocolo para la comunicación entre los nodos inalámbricos.
  - Diseño del protocolo para la comunicación entre el coordinador y el servidor.
  - Desarrollo de prototipos *software* para el protocolo.

- **Fase 6:** Diseño de la base de datos.

  - Profundización en el lenguaje SQL específico de MySQL.
  - Diseño e implementación del modelo de la base de datos.
  - Estudio del conector de MySQL para Java.
  - Prototipo del sistema de conexión a MySQL desde Java.

- **Fase 7:** Integración de las diferentes tecnologías.

  - Desarrollo del *software* necesario para comunicar y almacenar los datos de los nodos sensores con el servidor.
  - Diseño y desarrollo del sistema de visualización de datos de los nodos sensores *in-situ* (con un ordenador de campo, con un *software* que pueda ser manejado por un técnico del sistema).

- **Fase 8:** Creación de la web de consultas.

  - Estudio y documentación de las tecnologías web aplicables al proyecto: HTML5, CSS3, JavaScript y PHP.
  - Diseño y desarrollo del *software* web.
  - Pruebas de manejo desde diferentes dispositivos por diferentes personas.

- **Fase 9:** Revisión y actualización del sistema.

- Verificación de los objetivos satisfechos.
- Reestructuración de las partes más sensibles (del inglés *code refactoring*).
- Elaboración de documentación final a partir de la documentación intermedia generada en cada fase.

En cuanto a la duración temporal de cada una de las fases, se ha establecido un límite de entre 2 y 3 semanas para cada fase. Si bien la temporización no es estricta, se debe intentar cumplir con los objetivos en dicho tiempo, pudiendo tomar más o menos en función de la consecución de objetivos de la fase anterior. A este respecto hay que destacar que las primeras fases de documentación, estudio y diseño, se prevén más duraderas, pudiendo superar este límite, mientras que las fases de desarrollo y verificación, tienden a resolverse con más celeridad, probablemente debido a la inercia adquirida durante el transcurso de los desarrollos.

La Figura 3.1 muestra un diagrama Gantt con la planificación temporal seguida.

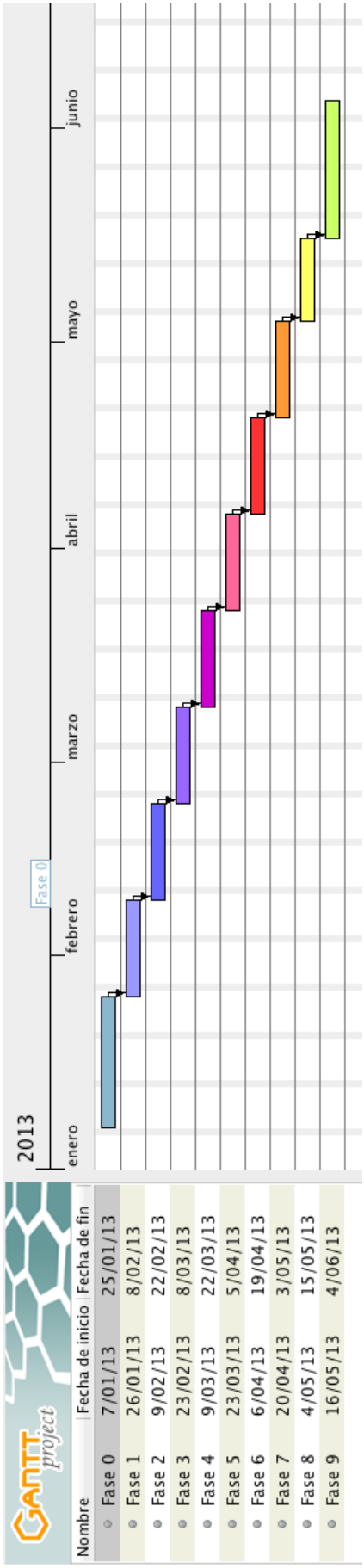

Figura 3.1: Diagrama de Gantt con la planificación temporal del proyecto. Comienza el 7 de Enero y finaliza el 4 de Junio, del año 2013.

Capítulo 4

Diseño e implementación

Este capítulo describe los detalles de diseño e implementación más relevantes para el sistema realizado. Se explican las arquitecturas del *hardware* y del *software*, se detallan las peculiaridades de los sensores, y se profundiza en las implementaciones de *software*, *firmware* y servicios utilizados.

4.1. Nodo sensor

4.1.1. Arquitectura *hardware* del nodo sensor

La Figura 4.1 muestra los componentes del sistema, entre los que se pueden ver:

- Arduino UNO.
- Wireless SD *shield*.
- Módulo XBee XB24-BWIT-004.
- Sensor de temperatura y humedad SHT11.
- Sensor de luz ambiente TSL2561.
- Sensor de gases contaminantes TGS2600.
- 2 transistores NPN (como 2N2222A, BC547C o 2N3904).
- Varias resistencias.

También se muestran esquemáticamente las conexiones de comunicación empleadas para manejar cada sensor, que serán expuestas más adelante.

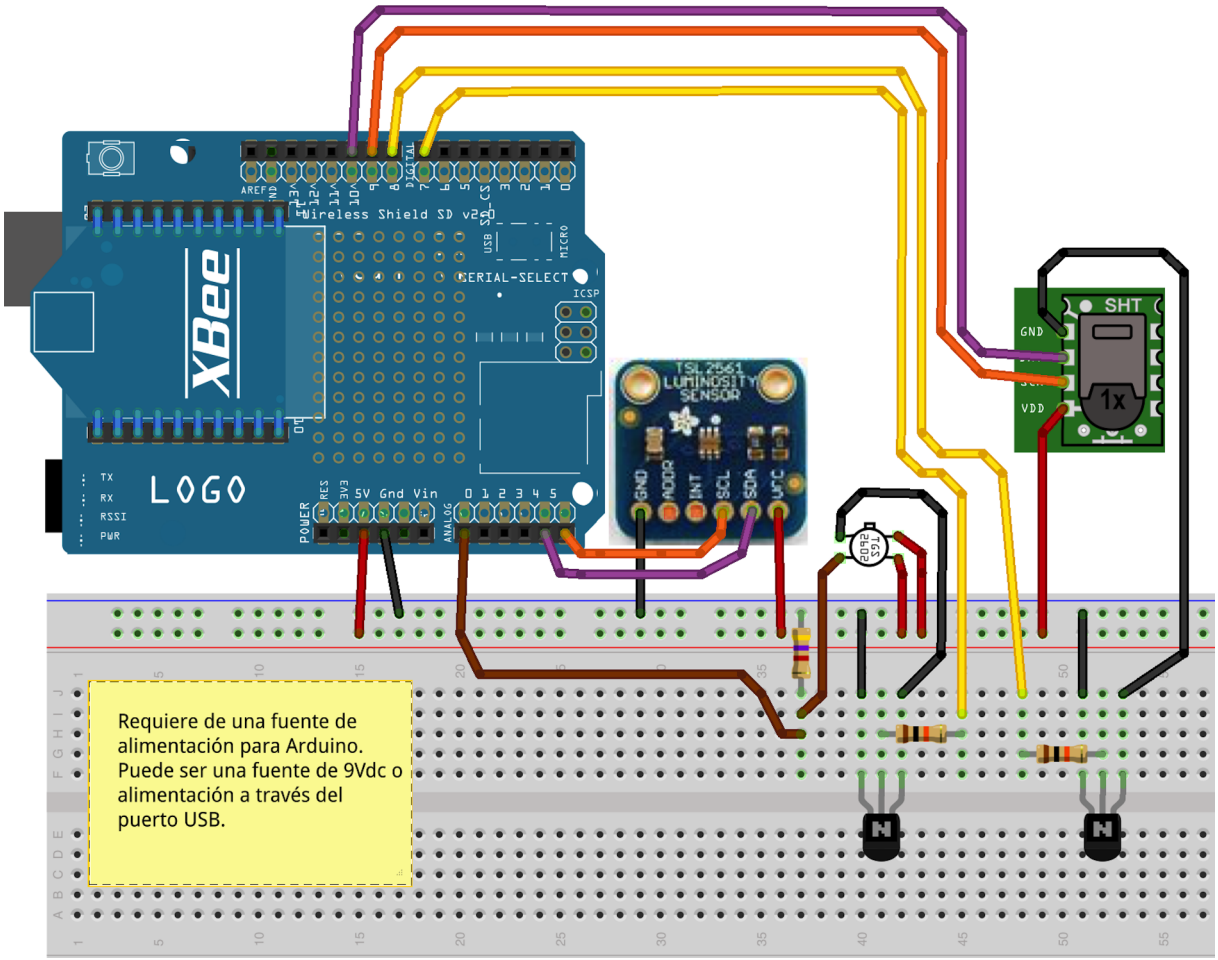

Figura 4.1: Nodo sensor compuesto por: Arduino UNO, Wireless SD shield, módulo Xbee, sensores y electrónica adicional.

Sensor de humedad y temperatura

Para medir la humedad relativa, el sensor SHT11 utiliza un sensor capacitivo, que gracias a su *ADC*[1] de 12bit o 14bit, ofrece una resolución de 0.05 % de humedad relativa, y tiene una precisión en la medida del 3 %.

Para obtener la temperatura, utiliza un sensor de banda prohibida (en inglés *bandgap temperature sensor*[2]) y ofrece medidas con una resolución de 0.01ºC y una precisión de 0.4ºC. Tanto el sensor de temperatura como el de humedad vienen calibrados de fábrica, lo cual simplifica el acondicionamiento del sensor, y facilita su uso directo.

Según se detalla en su hoja de datos [Sen08] puede efectuar una medida cada segundo, con una resolución de 12bit, estando así en funcionamiento un 10 % del tiempo de opera-

[1] Un *ADC* es un conversor analógico-digital, del inglés *Analog to digital converter*
[2] http://en.wikipedia.org/wiki/Silicon_bandgap_temperature_sensor

ción. Como el proyecto requiere de pocas mediciones diarias (un máximo de 4 por hora), puede apagarse completamente el sensor cuando no se esté usando. Para ello se puede utilizar una salida digital de Arduino, que controle un transistor BJT NPN común. Más adelante se detalla el esquema electrónico de la solución.

Para la lectura de datos desde Arduino, SHT11 dispone de una interfaz de comunicación serie digital de dos hilos, similar al bus I^2C[3] (y compatible a nivel físico con éste). Para conocer en detalle el protocolo de este sensor, se recomienda leer la hoja de datos junto a alguna implementación ya hecha del protocolo (en Internet existen varias referencias para microcontroladores de Atmel, Microchip, Freescale o Cypress, entre otros).

Sensor de luz ambiente

Como puede verse en la hoja de datos del sensor TSL2561 [AMS05], este cuenta con dos fotodiodos conectados a dos ADC de 16bit y tiene un consumo de 0.6mA en activo. Se comporta como un dispositivo esclavo en un bus I^2C, y puede obtenerse de él una medida de la cantidad de luz ambiente.

Aunque el sensor viene calibrado de fábrica, puede configurarse y ajustarse para una aplicación específica, por ejemplo modificando la ganancia de sus conversores AD desde el *software*. Si bien estos detalles se precisan en la hoja de datos, no se ha tenido en cuenta esta capacidad de personalización, pues el objetivo del proyecto es el de crear una plataforma sobre la que hacer experimentos y mejoras.

Teniendo en cuenta que los nodos sensores realizan medidas medioambientales, resulta apropiado calibrar el sensor mínimamente para que opere correctamente en entornos soleados. Por este motivo la ganancia seleccionada ha sido la mínima (ganancia cero). Hay otro parámetro que permite configurar el tipo de medida y es el tiempo de integración para la medida (llamado *integration time* en la hoja de datos). Cuando se comanda al sensor para realizar una medida, sus ADC integran las señales de los fotodiodos de luz visible e infrarroja, y esta integración tiene una duración ajustable. Se recomienda utilizar ganancias altas junto a tiempos de integración altos para condiciones de luz tenue. En el caso de este proyecto, el tiempo de integración elegido ha sido el mínimo posible, debido a que los nodos estarán, a priori, a la intemperie. Durante este tiempo el sensor permanece ocupado y no podrá realizar ninguna otra tarea, así que elegir un tiempo de integración lo más breve posible también ha sido un factor a considerar.

[3] http://www.i2c-bus.org

Sensor de gases

El elemento sensor TGS2600 utiliza una capa de óxido de metal semiconductor sobre una lámina de aluminio, permitiendo alterar su conductividad en función de la concentración de un gas en el aire. Según su hoja de datos [Fig05], es sensible a la presencia de pequeñas concentraciones de hidrógeno, monóxido de carbono o metano, que son gases contaminantes presentes, por ejemplo, en los humos. Puede utilizarse como un detector de incendios, humos y gases contaminantes.

Para su funcionamiento, el sensor requiere de un elemento calefactor. Este elemento precisa de una alimentación de 5Vdc y unos 42mA, y según la especificación, el proceso de calibración requiere un entorno controlado. La alimentación puede obtenerse de la placa Arduino, pero no puede alimentarse desde Arduino directamente, ya que según la especificación de los microcontroladores de ATmega 328 [Atm05], cada salida digital es capaz de manejar una corriente máxima de 40mA (que por prudencia podría establecerse en un máximo de 20mA prácticos).

Como se requiere un sistema que minimize el consumo, se ha empleado un transistor común BJT NPN, que hace de interruptor, permitiendo encender o apagar este sensor completamente. Desde Arduino puede emplearse una salida digital hacia la base del transistor, habilitando o inhabilitando el flujo de corriente entre el colector y el emisor.

Conviene indicar que, al requerir de un elemento calefactor, tras enceder el sensor se necesitarán unos minutos hasta que la medida se estabilice.

Para la calibración del sensor se han intentado seguir las indicaciones del fabricante:

- Entorno controlado: las condiciones ideales son de 20ºC y un 65 % de humedad relativa. El entorno de acondicionamiento ha sido un pueblo cerca de Cáceres, con una media de temperatura de 19ºC y 58 % de humedad relativa durante los días del acondicionamiento. El aire, salvo por las carreteras que circunvalan las poblaciones, está a priori bastante libre de contaminantes.
- Periodo de acondicionamiento de 7 días (en funcionamiento ininterrumpido).
- Tras el periodo de acondicionamiento comienzan las primeras medidas.

El objetivo del proyecto no es dejar perfectamente calibrado ni ajustado eléctricamente este sensor, por lo que estos procesos se han seguido lo mejor posible para tener

funcionando el sensor y poder seguir el desarrollo.

El dispositivo sensor tiene una resistencia intrínseca (llamada en la documentación RS o *sensor resistor*). Como la salida que ofrece es analógica, se necesita utilizar una resistencia externa para tener un divisor de voltaje, cuya salida se envía a una entrada analógica de Arduino, que en el prototipo se conecta a la entrada *A0* de Arduino.

Esta resistencia externa se llama RL (en la documentación se referencia a este valor como *load resistor*). Siguiendo las recomendaciones del fabricante, debe elegirse un valor para RL de entre 0,5kOhm y 10kOhm. Este valor es variable en función de la resistencia intrínseca del sensor, que debe ser medida. Como el proceso de fabricación del sensor no permite fijar el valor de RS en fábrica, el fabricante recomienda medir la resistencia intrínseca del sensor una vez se ha realizado el proceso de calibración y acondicionamiento. En el caso de este proyecto, tras 7 días con el sensor encendido se midió una resistencia intrínseca de 56kOhm, por lo que se ha usado una resistencia RL de 10kOhm.

Módulo de comunicaciones Zigbee

Para el desarrollo del proyecto se han utilizado los módulos XBee de la serie 2, concretamente la familia XB24-BxWIT-004[4]. Estos módulos disponen de una interfaz serie RS232 TTL a través de la cual se comunican con un microcontrolador, u otro dispositivo compatible. Esta interfaz serie permite tanto enviar mensajes inalámbricos a otros módulos, como mensajes para configurar parámetros del propio módulo en tiempo de ejecución.

Para crear una red se necesita que todos los nodos que la forman se comuniquen por el mismo canal de emisión, y que compartan un mismo identificador llamado *PAN*[5]. Cada nodo viene de fábrica con su propia (y única) dirección de acceso al medio[6], y permite que se pueda destinar un paquete de información a un nodo o grupo de nodos concreto [Fal10]. Esta dirección única es de 64bit, y en el argot de XBee se llama *Source address*. Cuando se envía un paquete a un nodo en concreto, se debe modificar el contenido del registro que almacena el destinatario del mensaje, y que en los módulos XBee se llama *Destination address*. También pueden configurarse los módulos para enviar mensajes de difusión *broadcast* o *multicast*.

[4]http://www.digi.com/pdf/ds_xbeedigimesh24.pdf
[5]PAN, del inglés Personal Area Network
[6]Dirección de acceso al medio o *Media Access Control* en inglés

Existen otras formas de enviar mensajes a módulos XBee, como utilizar una dirección de red de 16bit, o usar un identificador alfanumérico personalizado, pero no se han utilizado pues se usa el direccionamiento a partir partir de la dirección única de 64bit. Además hay muchas otras características interesantes para configurar los módulos, como puede ser la gestión de los estados de reposo (*sleep-modes*) o el cifrado de las comunicaciones, pero quedan fuera del alcance de este proyecto.

Cada módulo de XBee integra en su interior unos registros de memoria no volátil que almacenan la configuración del módulo, permitiendo que por ejemplo el canal, la dirección *PAN* o la dirección *MAC* puedan ser recuperadas tras un reinicio o una interrupción de suministro eléctrico del módulo.

Configurando adecuadamente los módulos XBee, se pueden obtener diversas topologías de red. En el caso de este proyecto se crea una red de malla o *mesh*, y con la herramienta XCTU mencionada en la sección 3.1.2 se establece la siguiente configuración para el nodo coordinador:

- **SH / SL**: 0013A200 / 4092D6AE
- **PAN**: 777
- **DH / DL**: 0x0000 / 0xFFFF

Y para un nodo sensor se tiene, por ejemplo, la siguiente configuración:

- **SH / SL**: 0013A200 / 4089EC22
- **PAN**: 777
- **DH / DL**: 0x0000 / 0x0000
- **JV**: 1

Todos los nodos de la red deben tener el mismo *PAN* y cada uno tendrá su propia dirección de 64bit, almacenada en los registros (*SH* y *SL*. Los registros *DH* y *DL* sirven para configurar el destinatario de un paquete de información. En el caso del nodo coordinador, se selecciona el valor 0x0000FFFF para que envíe los datos por difusión o *broadcast* a todos los nodos sensores. En el caso de los nodos sensores, se configuran los registros con valor cero. Este valor implica que sus paquetes de información serán enviados al nodo

coordinador.

El parámetro del nodo sensor *JV* puesto a 1 implica que el nodo sensor verificará si hay un coordinador en la red antes de enviar paquetes.

Por otra parte, en la hoja de datos [Dig13] de los módulos XBee se detallan dos modos de operación para los módulos de la serie 2, y que se introducen a continuación.

En el modo *Transparente* (también llamado *modo AT*), el módulo XBee es transparente a la comunicación. Esto quiere decir que lo que reciba por la entrada serie RX lo enviará inalámbricamente. Análogamente, toda información que reciba inalámbricamente la enviará a través de su salida serie TX. Dentro de este modo, existe una funcionalidad para configurar el módulo, mediante el empleo de una secuencia de inicio específica y un conjunto de comandos AT. Existen comandos AT para recuperar la dirección de red propia del módulo, o para fijar el destinatario del siguiente paquete, por ejemplo.

Existe otro modo de operación llamado *API*, en el cual se tiene más control sobre la comunicación. En este modo se puede cambiar rápidamente el destinatario de un mensaje, y se puede conocer la dirección de un paquete entrante. Requiere más control que el modo Transparente, y exige la gestión de la comunicación por parte del microcontrolador, empleando un formato de trama específico para este modo.

En los XBee serie 2, cada modo de operación requiere su propio *firmware*, por lo que se hace uso de la herramienta XCTU para programar en cada módulo el *firmware* apropiado.

Para el objetivo de este proyecto, es suficiente con emplear la serie de *firmware* para el *modo AT*. No obstante, durante el desarrollo del proyecto se intentó utilizar el *modo API* con la biblioteca **XBee-Arduino**[7] para Arduino, y **XBee-Api**[8] para Java. Por motivos de plazos se tuvo que dejar este desarrollo fuera de la planificación.

Los principales motivos para preferir el *firmware* en *modo API* son las capacidades de gestión de los estados de ahorro de energía del módulo, la capacidad de gestión y configuración de módulos remotos o la capacidad de compartir información punto a punto en tiempo de ejecución, entre otras.

[7] http://code.google.com/p/xbee-arduino
[8] http://code.google.com/p/xbee-api

Esquema electrónico

Con el *software* KiCAD se ha diseñado el esquema electrónico que recoge todos los componentes necesarios para el manejo de los sensores. La Figura 4.2 muestra este esquema:

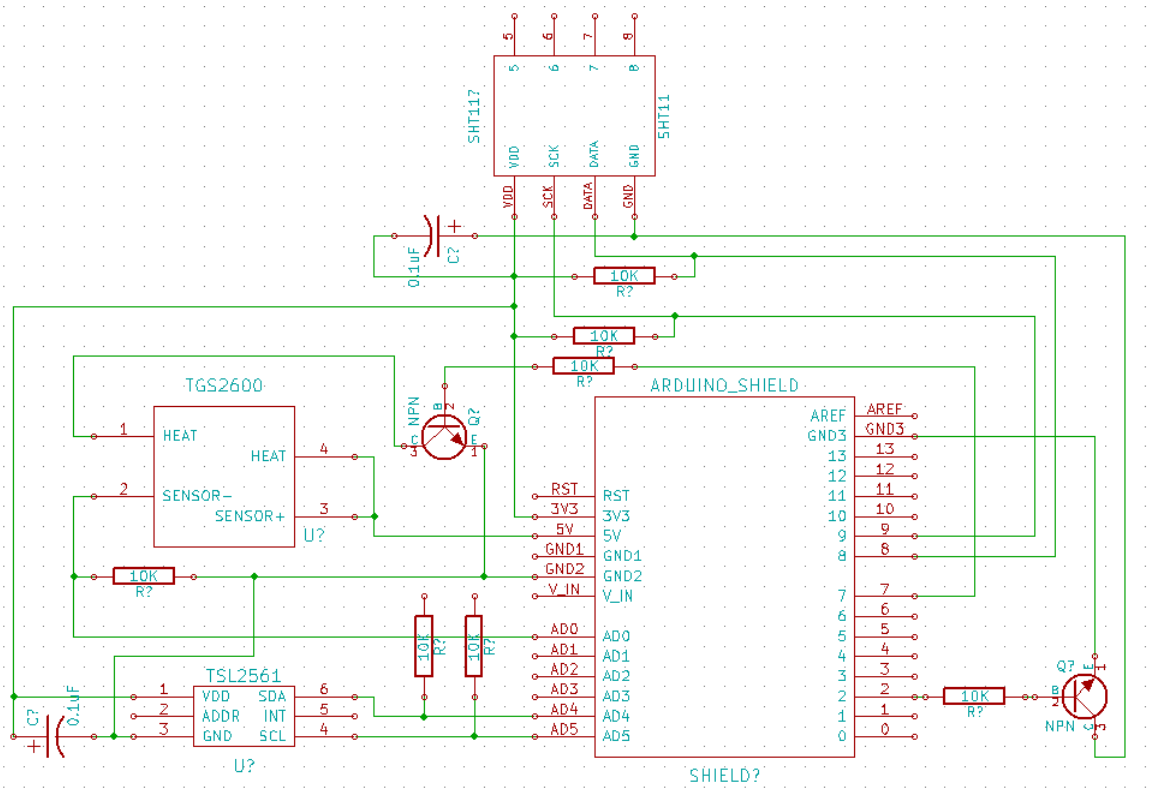

Figura 4.2: Esquema electrónico que muestra los componentes y las conexiones para un nodo sensor.

Prototipo funcional

Una vez se ha realizado el esquema electrónico, se ensamblan todos los componentes en una placa para prototipos (*protoboard* en inglés). El resultado del ensamblaje se puede ver en la Figura 4.3:

4.1.2. Arquitectura *software* del nodo sensor

Perspectiva global del *firmware* para Arduino

Desde un punto de vista general, el *firmware* debe obtener las medidas o lecturas de los sensores y enviarlas hacia el nodo coordinador. La Figura 4.4 muestra un diagrama de flujo representativo de esta perspectiva global (o función principal).

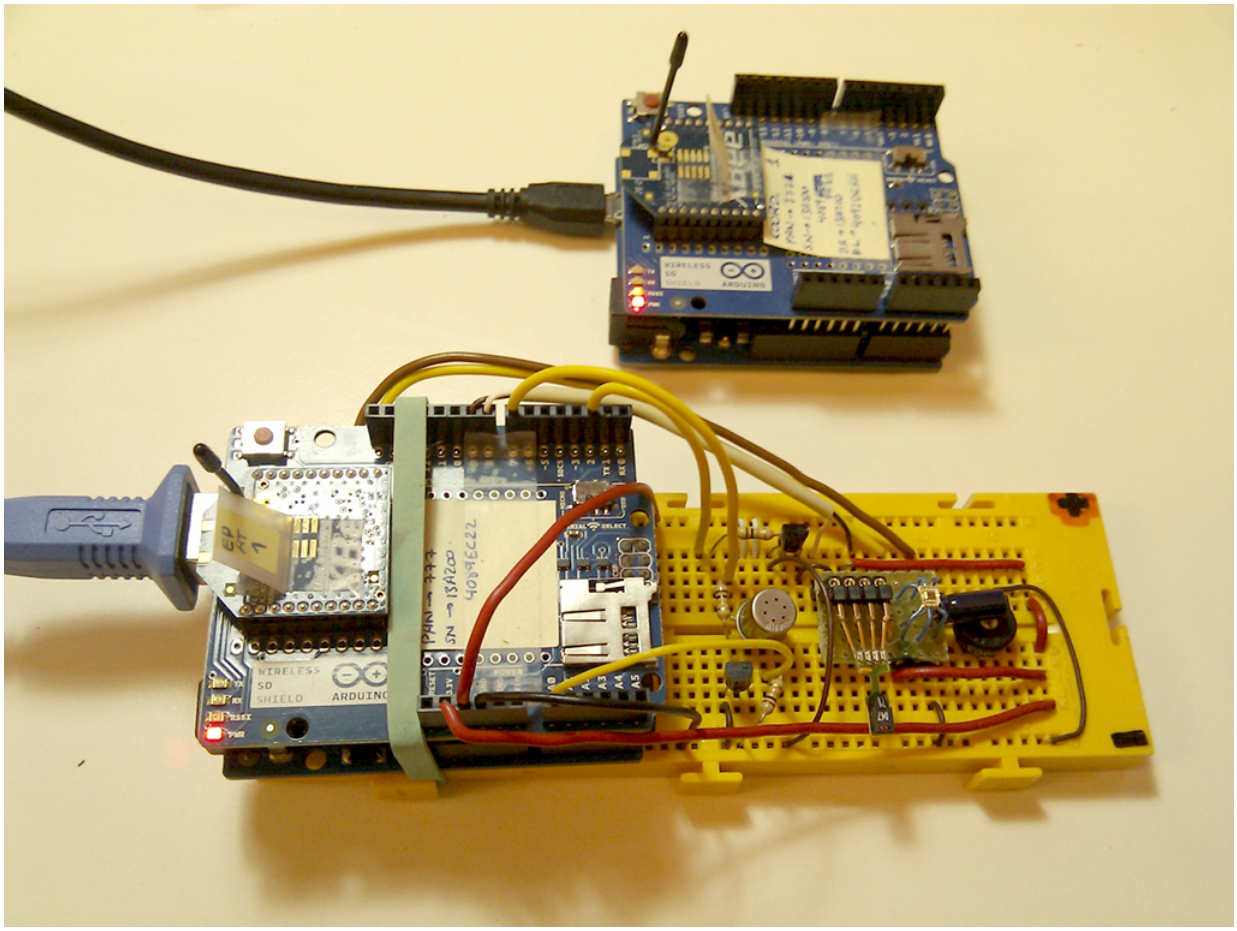

Figura 4.3: Prototipo electrónico para un nodo sensor.

Siguiendo las recomendaciones de [Igo07] se distribuye el código en varios ficheros, cada uno dedicado a un proceso. Así existe un fichero principal, que implementa la idea del diagrama de flujo de la Figura 4.4. El nombre de este fichero es *XBee_Enpoint_UNO_v005.ino*. Existen asimismo ficheros que recogen los procesos de los diferentes sensores, y un fichero adicional con estructuras de datos y funciones utilitarias globales.

Todos los procesos han sido implementados considerando las temporizaciones de adquisición requeridas por cada sensor (que necesitan periodos de actividad e inactividad), de tal manera que todos los procesos intentan realizar una multitarea colaborativa, liberando recursos del microcontrolador durante su periodo de inactividad. Es por esto que todas las funciones de los procesos reciben por parámetro el número de milisegundos desde que se inició el programa. Cada proceso tiene en cuenta estos milisegundos para llevar a cabo su tarea o dejar libre el procesador, y por ello estas funciones permiten ser invocadas reiteradamente desde el bucle del programa principal, y sincronizan su ejecución al reloj del sistema, permitiendo que sólo actúen cuando su planificación de funcionamiento lo exija.

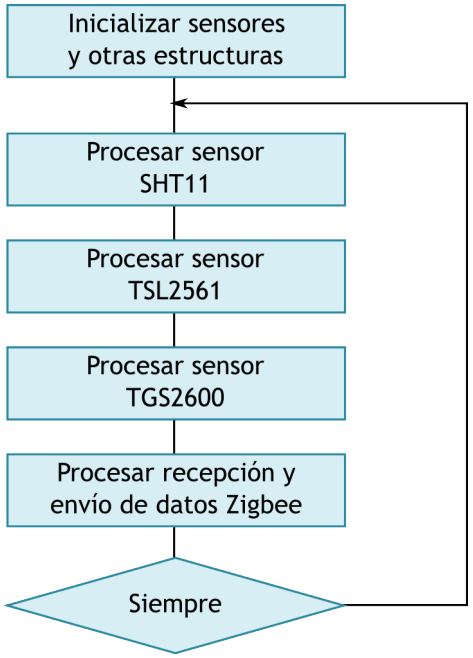

Figura 4.4: Diagrama de flujo para el programa principal.

Se podrían haber utilizado recursos de C++ más avanzados, pero se ha preferido dejar una implementación simple, lo más cercana a C posible, con la idea de que pueda ser portada a arquitecturas de microcontrolador que no dispongan de compilador C++. También se encontró una dificultad durante el desarrollo de las primeras ideas con C++ y Arduino, y es que desde una clase creada por el usuario no se puede declarar ningún miembro que sea objeto de una clase de las bibliotecas de Arduino. La forma más elemental de solventar esto es utilizando, en las clases propias, miembros que apunten a objetos externos, creados en el programa principal, y pasados por método o por constructor. No obstante, se estuvo indagando para tratar de hacerlo "lo más orientado a objetos posible", y al no poder resolverlo en tiempo aceptable, se optó por continuar el desarrollo en un estilo más cercano a C, y tratando de desacoplar, lo más posible, las diferentes funcionalidades.

Lectura y manejo del sensor de humedad y temperatura

Existen multitud de ejemplos de código para utilizar este sensor con Arduino, por lo que se ha decidido no realizar una implementación a medida, y el proyecto utiliza la biblioteca *Sensirion* para Arduino[9].

Esta biblioteca ofrece la posibilidad de realizar medidas de temperatura y humedad de forma no bloqueante, de tal forma que se comanda al sensor para realizar una medida,

[9]http://playground.arduino.cc/code/Sensirion

y se le va preguntando hasta que la tiene lista, momento en el que se obtiene su valor. En el diagrama de flujo de la Figura 4.5 se aprecia este detalle en las instrucciones que hacen referencia a si la medida está o no lista.

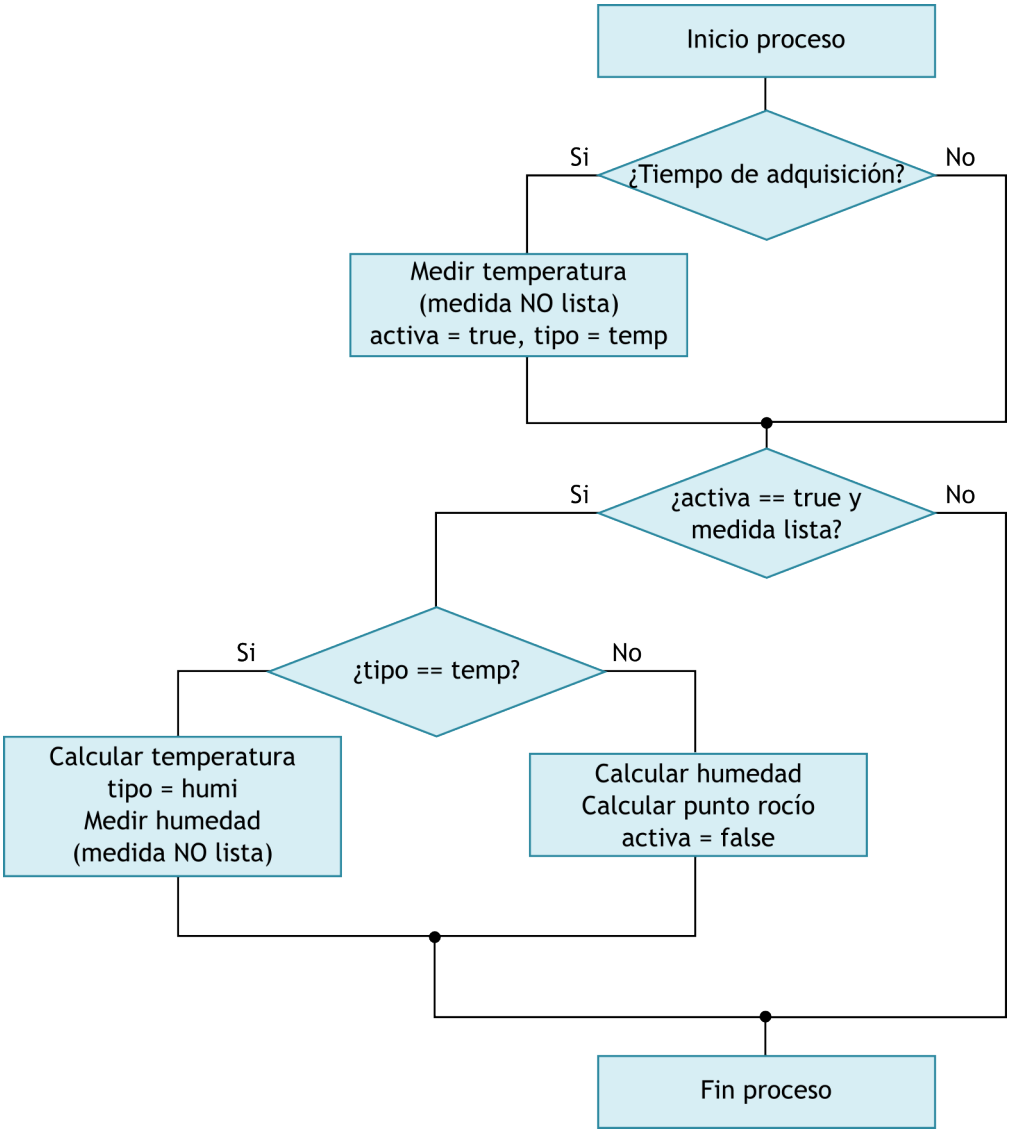

Figura 4.5: Diagrama de flujo para la adquisición de temperatura y humedad relativa, de una forma colaborativa. Se corresponde con el proceso del sensor SHT11 de la Figura 4.4.

Los valores de temperatura y humedad son almacenados en dos variables de tipo *float* (tipo que en Arduino ocupa 4 *bytes*). Estas variables serán accesibles al programa principal, o a cualquiera de los otros ficheros fuente, a través de variables *extern*.

Lectura y manejo del sensor de luz ambiente

Como ocurría en el caso del sensor de temperatura y humedad, existe también código fuente en Internet con implementaciones que simplifican el manejo de este sensor. Para

el proyecto, se ha hecho uso de la biblioteca *TSL2561* desarrollada por Adafruit[10]. Esta biblioteca utiliza internamente la biblioteca *Wire*, que facilita el uso de dispositivos I^2C en Arduino.

Como se ha comentado en la sección 4.1.1, resulta apropiado calibrar el sensor mínimamente para que opere correctamente en entornos soleados. La ganancia seleccionada por *software* ha sido la mínima de *0x* y el tiempo de integración para la medida también es el mínimo, de 13ms.

En este caso, y debido a cómo está implementada la biblioteca utilizada, no se dispone de un diseño de ejecución colaborativa, debiendo esperar el proceso el tiempo necesario para que el sensor finalice la medida. Debido a que este tiempo es bastante corto (13ms) no ha sido necesario ajustar mejor el comportamiento bloqueante para manejar este sensor, quedando un código fuente muy sencillo. La Figura 4.6 muestra el sencillo diagrama de flujo que recoge la implementación para la tarea de lectura y manejo de la información lumínica.

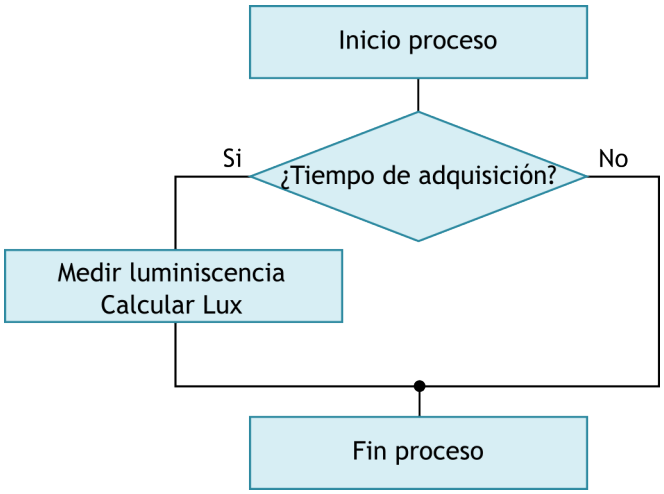

Figura 4.6: Diagrama de flujo para la adquisición y procesado del nivel de luz ambiente. Se corresponde con el proceso del sensor TSL2561 de la Figura 4.4.

Lectura y manejo del sensor de gases contaminantes

El sensor de gases proporciona los datos a través de una señal analógica, conectada a la entrada analógica A0 de Arduino. Como no se ha hecho una calibración precisa del sensor, se ha relacionado el intervalo de muestras por millón capaz de detectar, con el

[10]http://learn.adafruit.com/tsl2561

intervalo ofrecido por el conversor AD de 10bit de Arduino (intervalo [0, 1023]). El aumento o disminución de las medidas al emplear humos durante las pruebas, provocaba un notable cambio en las medidas, lo cual podría ser útil en escenarios donde sea posible encontrar una presencia muy alta de humos. En cualquier caso, para aplicaciones críticas se necesita filtrar y adaptar la conversión, para conseguir la máxima resolución posible, probablemente bajo la asesoría del fabricante.

Para la lectura de este sensor, también se ha implementado una solución colaborativa, de tal forma que el proceso ocupe pocos recursos mientras se está calentando o mientras está en reposo. Una vez el elemento sensor está acondicionado, se procede a la lectura y procesado de datos obtenidos. La Figura 4.7 muestra un diagrama de flujo que recoge el funcionamiento de este proceso.

Gestión del módulo XBee

Dado que no se usa el *modo API*, existe una funcionalidad que debe ser implementada en el *modo AT*, y es la de poder obtener el origen de la transferencia para un paquete de datos. Esto es, que el nodo coordinador sea capaz de distinguir de qué nodo sensor es cada paquete que llega. Esto implica que cada nodo sensor debe conocer su dirección de red y debe transmitirla en cada paquete de datos que envíe.

El nodo sensor, para obtener su propia dirección de 64bit, debe enviar a su módulo XBee una secuencia especial de *bytes* (representados por la cadena "+++") y esperar durante un segundo la respuesta del módulo XBee. Esta secuencia provoca que el módulo XBee espere algún comando AT. En ese momento se le pide la dirección propia de 64bit mediante comandos AT. Concretamente se le pide tanto la parte alta como la parte baja de la dirección, con los comandos **ATSH** y **ATSL**. Conviene mencionar que al solicitar la parte alta de la dirección (*ATSH*) el módulo devuelve 7 *bytes*, mientras que al solicitar la parte baja (*ATSL*) el módulo devuelve 9 *bytes*. Los dos *bytes* de diferencia que no se envían en la parte alta, se presuponen con valor cero (0x00).

La Figura 4.8 ilustra el diagrama para este mecanismo de obtención de la dirección.

Protocolo de comunicaciones sobre Zigbee

Al utilizar el modo *AT* de los módulos XBee, el envío de datos por vía inalámbrica se implementa enviando y recibiendo datos por el puerto serie de Arduino.

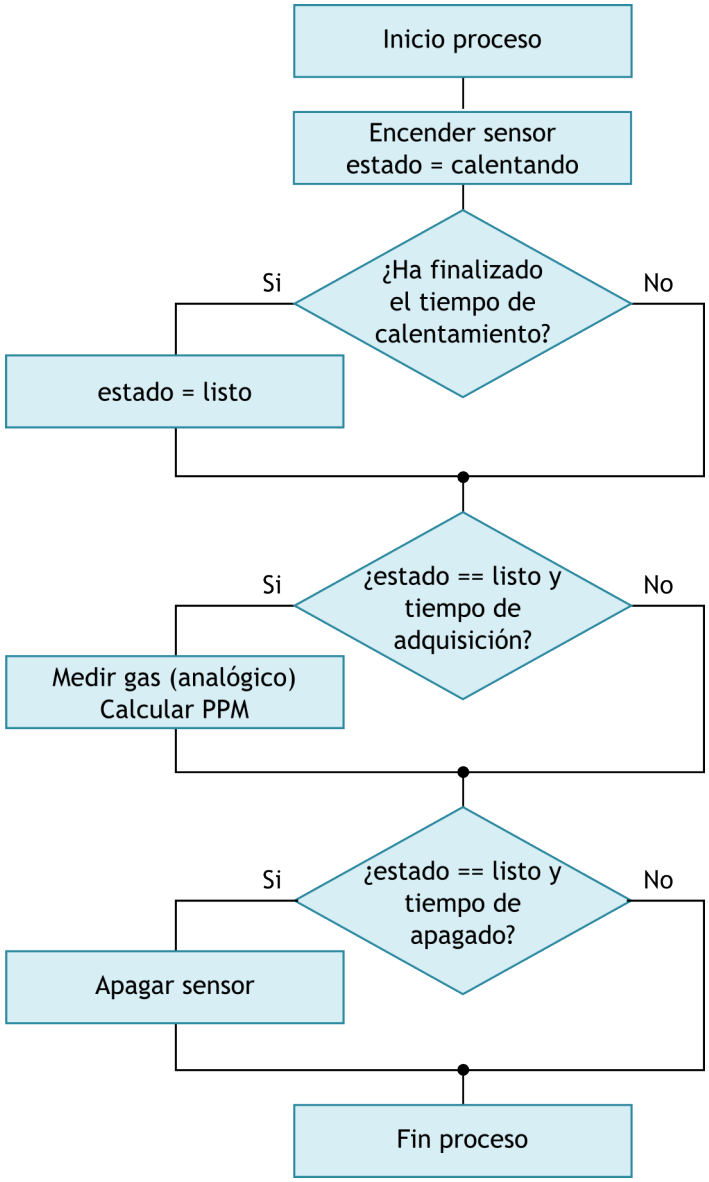

Figura 4.7: Diagrama de flujo para la adquisición y procesado de la cantidad de gases contaminantes presentes en el aire. Se corresponde con el proceso del sensor TGS2600 de la Figura 4.4.

Para la comunicación entre un nodo sensor y el nodo coordinador, se ha definido un protocolo de intercambio de paquetes de *bytes*. Cada uno de los *bytes* de un paquete conforman los siguientes campos:

- Marca de inicio de paquete: ocupa un *byte* con el valor fijo 0xFF.
- Dirección de origen: ocupa 16 *bytes* que contienen la dirección del nodo originario del paquete.
- Tipo de paquete: ocupa un *byte* que permite la ampliación del protocolo. Para el proyecto se ha implementado un único paquete consistente en el envío de los datos

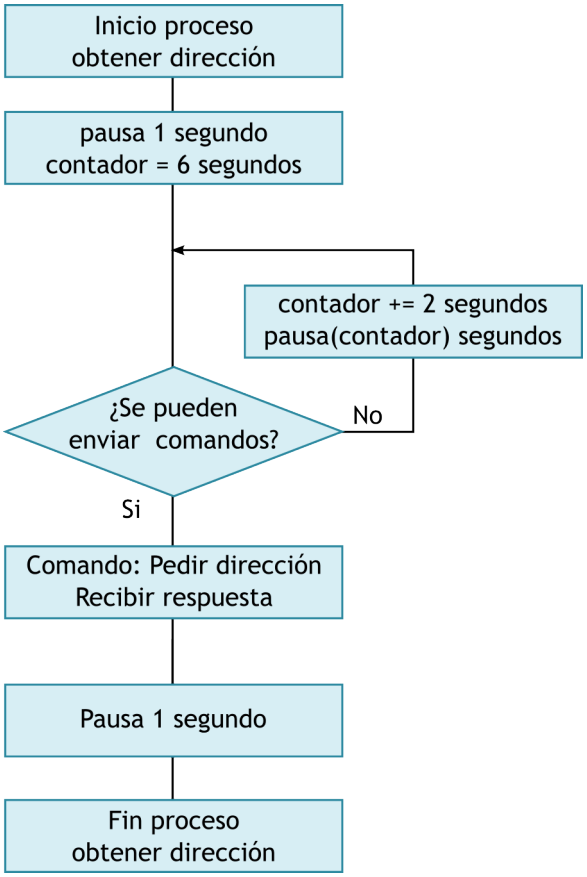

Figura 4.8: Diagrama de flujo para la lectura de la dirección de 64bit del módulo XBee.

recabados por los sensores.

- Tipo de nodo: ocupa un *byte*, cuyo valor representa a un coordinador, a un enrutador (o *router*) o a un sensor (dispositivo final o *end-device*).
- Tamaño total del paquete: ocupa un *byte*, y su valor incluye marca de inicio y CRC, además de todos los campos intermedios.
- Temperatura: ocupa 4 *bytes* que representan un número en coma flotante (el tipo *float* se representa 4 *bytes* en los micros de Atmel).
- Humedad relativa: ocupa 4 *bytes* que representan un número en coma flotante.
- Nivel de luz ambiente: ocupa 4 *bytes* que representan un número en coma flotante.
- Nivel de gases contaminantes: ocupa 4 *bytes* que representan un número en coma flotante.
- Comprobación de redundancia cíclica (*Cyclic redundancy check* en inglés). Los dos *bytes* del CRC sirven para verificar si el contenido del paquete ha sido alterado o es

Índice	Descripción	Contenido
0	Marca inicio paquete	0xFF
1	Dirección origen	16 *bytes*
17	Tipo de paquete	1 *byte*
18	Tipo de nodo	1 *byte*
19	Tamaño total del paquete	1 *byte*
20	Temperatura (*float*)	4 *bytes*
24	Humedad relativa (*float*)	4 *bytes*
28	Nivel de luz (*float*)	4 *bytes*
32	Nivel de gases (*float*)	4 *bytes*
36	CRC (MSB)	1 *byte*
37	CRC (LSB)	1 *byte*

Tabla 4.1: Estructura del paquete de datos que envía el nodo sensor al nodo coordinador.

incorrecto. El algoritmo CRC empleado es el CRC-CCITT[11] de 16bit.

La Tabla 4.1 muestra gráficamente el paquete descrito.

Proyecto *software*

En el proyecto del nodo sensor para Arduino, se encuentran los ficheros:

- **XBee_Endpoint_UNO_v005.ino**: posee el código del programa principal para el nodo sensor.
- **Comm_XBee.ino**: incluye todo el código que gestiona el módulo de XBee y la comunicación (envío) de datos hacia el nodo coordinador.
- **Sensor_SHT1x.ino**: encapsula la funcionalidad de adquisición de temperatura y humedad relativa.
- **Sensor_TSL2561.ino**: encapsula el método de adquisición y gestión del sensor de luz ambiental.
- **Sensor_TGS2600.ino**: encapsula el mecanismo de adquisición del nivel de gases contaminantes.
- **Util.h, Util.ino**: incluye herramientas y utilidades, como definiciones de tipos de datos, y funciones genéricas.

[11] http://en.wikipedia.org/wiki/Crc16

4.2. Nodo coordinador

4.2.1. Arquitectura *hardware* del nodo coordinador

En este proyecto se ha mantenido al nodo coordinador lo más desocupado posible, y no se le ha dotado de funcionalidad sensorial. Su misión es recibir los datos que envían los nodos sensores y transferirlos al servidor. No obstante, la arquitectura *hardware* del nodo coordinador puede ser la misma que la de un nodo sensor. De esta manera sería un nodo sensor especial, ya que además de tomar medidas y enviarlas al servidor, estaría encargado de atender los envíos de datos del resto de nodos.

El único cambio significativo con respecto a los nodos sensores, es que el nodo coordinador utiliza una placa Arduino Leonardo, si bien compatible a nivel de código con otras versiones de Arduino, como la versión UNO utilizada en los nodos sensores.

Como se mencionaba en el apartado 3.1.1, dispone de un microcontrolador con más capacidades, siendo la más relevante que dispone de dos puertos serie. Se ha aprovechado esta característica para construir un puente (o *bridge*) de ZigBee a USB, con un microcontrolador capaz de gestionar la comunicación mediante un protocolo específico.

Así pues, el *hardware* se reduce a una placa Arduino Leonardo, una placa *Wireless SD Shield* para Arduino, y un módulo XBee, modelo XB24-BWIT-004.

4.2.2. Arquitectura *software* del nodo coordinador

El *firmware* del nodo coordinador debe realizar dos tareas:

- **Recibir los datos de los nodos sensores**, asegurando que cumplen el protocolo y el formato de paquete. Esta tarea implica descartar los paquetes que lleguen corruptos (con CRC incorrecto).
- **Reenviar los paquetes recibidos al servidor**.

El programa principal es muy sencillo, y tras realizar la configuración inicial para ambos puertos serie (llamados *Serial* y *Serial1* en Arduino), el bucle infinito simplemente ejecuta el proceso que gestiona las comunicaciones (*xbee_Process()* en el código fuente).

Protocolo de comunicaciones con nodos sensores

La Figura 4.9 muestra el diagrama de estados que ilustra la máquina de estados que sigue la implementación del protocolo.

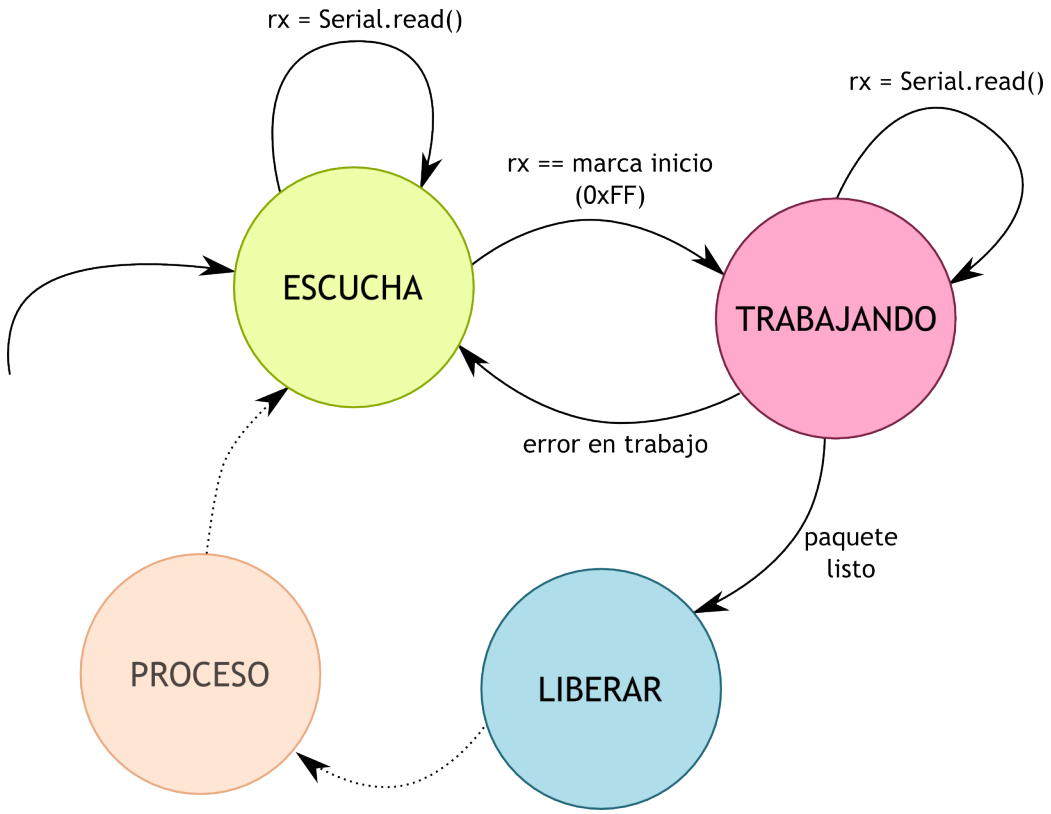

Figura 4.9: Diagrama de estados para la recepción de *bytes* según el protocolo definido.

En el diagrama aparecen dos detalles que conviene resaltar:

- el mensaje de **error en trabajo**. La recepción de un paquete requiere la lectura de un número finito de *bytes*, que van llenando un *buffer*. En el caso de superarse la longitud esperada o la máxima de un paquete, se produce una situación de error, que provoca la apocatástasis de la máquina de estados (se retorna al estado inicial de escucha y se reestablecen las variables apropiadas).

- el estado **PROCESO**. Una vez el paquete está listo para ser liberado, el proceso puede liberarlo. Este estado de proceso se encargará de mirar detalles del protocolo, como el tipo de nodo, o el tipo de paquete, para ejecutar el código conveniente. En la implementación actual, se mira el tipo de paquete para ver si corresponde a un paquete de envío de datos de sensores, en cuyo caso se acepta el paquete, que es reenviado tal cual al servidor, a través del puerto USB.

Comunicaciones con el servidor

Aunque podría haberse desarrollado un nuevo protocolo para estas comunicaciones, el nodo coordinador tan sólo reenvía al servidor los paquetes de información que recibe desde los nodos sensores. Se ha preferido realizar la misma implementación del protocolo

en el *software* del servidor y disponer de una referencia de código homogénea y funcional en nodos y servidor.

Proyecto *software*

El código fuente corresponde a un proyecto Arduino. En el proyecto, se pueden encontrar los siguientes ficheros:

- **XBee_Coordinator_Leonardo_v005.ino**: posee el código del sencillo programa principal descrito anteriormente.
- **Comm_USB.ino**: tan sólo tiene el código de inicialización del puerto serie USB, aunque podría añadirse aquí más funcionalidades para la comunicación con el servidor.
- **Comm_XBee.ino**: incluye todo el código que gestiona el módulo de XBee, la recepción de datos desde los nodos sensores y la comunicación con el servidor.
- **Util.h, Util.ino**: incluye herramientas y utilidades, como definiciones de tipos de datos y funciones genéricas.

4.3. Servidor

Como se mencionaba en la sección *3.1.1 Sistemas Informáticos*, el servidor es en realidad un servidor virtual alojado en una empresa de *hosting*, y el *hardware* es equivalente a un ordenador de escritorio de prestaciones medias. La instalación y administración del sistema operativo y los servicios se ha realizado de forma remota a través de SSH.

4.3.1. Servicios y herramientas *software*

Instalación de servicios y herramientas

Tras instalar el servidor, se instalan los servicios y programas necesarios, siendo los más relevantes: *openssh-server*, *mysql-client*, *mysql-server*, *nginx-full*, *php5*, *php5-fpm*, *php-pear*, *php5-common*, *php5-mcrypt*, *php5-mysqli*, *php5-cli*, *php5-gd*, *build-essential* y *sun-java6-jre*.

Configuración del servidor

De las tareas que debe realizar el servidor, la más crítica es el almacén de los datos recogidos por los sensores, y por este motivo la base de datos. Se configura el servidor MySQL editando su fichero de configuración en */etc/mysql/my.cnf*. Se ajusta la opción

bind-address para que MySQL se ate a la dirección IP del servidor.

También se configura el entorno de ejecución de PHP, y para ello se edita el fichero */etc/php5/fpm/php.ini*, cambiando la zona horaria a una adecuada y ampliando el límite de memoria que un *script* puede consumir a 256Mbytes(se configura con el parámetro *memory_limit*). Dado que PHP permite la creación de páginas con contenido dinámico, se cambian además los valores de los parámetros *sclient_max_body_size* a 20Mbytes y *client_body_buffer_size* a 128k, para permitir que PHP muestre páginas con gran cantidad de contenido. En el futuro, durante la fase de despliegue e implantación del sistema, estos y otros parámetros deben ser revisados y configurados apropiadamente.

Teniendo en cuenta que la web se ejecuta en un servidor virtual, se realizan algunos cambios en la configuración de Nginx, para limitar su consumo de recursos. Esta limitación de recursos se debe a que la mayoría de los accesos se presuponen para los sensores, es decir, accesos a insertar datos en la base de datos. En este proyecto no se contempla un uso masivo de la web de consulta, sino más bien un uso moderado por parte del personal a cargo de las instalaciones o de la interpretación de los datos. Se modifican los parámetros del fichero */etc/nginx/nginx.conf* como se muestra a continuación:

```
pm.max_children = 15
pm.start_servers = 4
pm.min_spare_servers = 2
pm.max_spare_servers = 5
pm.max_requests = 300
request_terminate_timeout = 30s
```

Por último se configura el servidor web Nginx, creando un *virtual host* a través del cual servir las páginas necesarias. Para dar de alta el *virtual host* se crea un nuevo fichero dentro de */etc/nginx/sites-available*, y se crea un enlace débil hacia él desde */etc/nginx/sites-enabled* para que Nginx lo active.

4.3.2. Base de datos

La base de datos que recoge la información de los nodos sensores sigue el esquema mostrado en la Figura 4.10.

Como puede observarse, entre las tablas **NODE** y **READING** existe una interrelación de orden 1:N, permitiendo que existan múltiples lecturas de sensores para un nodo.

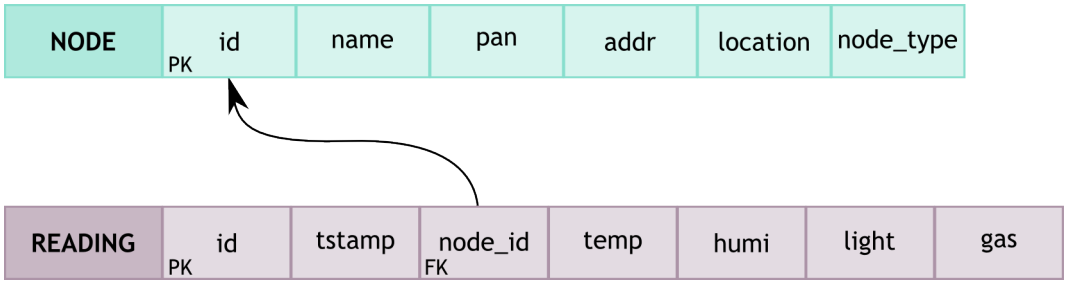

Figura 4.10: Esquema de la base de datos que almacena las lecturas de los nodos sensores.

A continuación se comentan los campos que conforman cada tabla, su finalidad y el tipo de datos elegido para su almacenamiento. Para la tabla **NODE** se tienen los siguientes campos:

- **id**: Es un identificador único para cada registro de esta tabla. Es la clave primaria de la tabla (*primary-key*) y es de tipo *INT*, con la opción de *AUTO-INCREMENT*.
- **name**: Es una cadena de caracteres con el nombre de un nodo, y sirve para simplificar la identificación de los nodos. Su tipo de datos es *VARCHAR(45)*.
- **pan**: Es el identificador de la red a la que pertenece el nodo (del inglés *Personal Area Network*). Su tipo de datos es *INT*.
- **addr**: Es una cadena de caracteres que almacena la dirección de red del nodo (que es guardada como una cadena de caracteres), con tipo de datos *VARCHAR(20)*.
- **location**: Es una cadena de caracteres que guarda la localización de un nodo. Se ha incluido este campo junto con el de *name* para facilitar la identificación y localización de un nodo a través de nombres comunes. El tipo de datos es *VARCHAR(45)*.
- **node_type**: Es un número entero que guarda el tipo de nodo. Típicamente este campo será de tipo "sensor", ya que son los nodos sensores los que realizan mediciones. Pero se ha incluido por si en el futuro deciden utilizarse nodos *routers* o coordinadores que hagan uso de sensores. El tipo de datos elegido es *TINYINT*.

La tabla **READING** contiene los campos siguientes:

- **id**: Es un identificador único para cada registro de esta tabla. Es la clave primaria de la tabla (*primary-key*) y es de tipo *INT*, con la opción de *AUTO-INCREMENT*.
- **tstamt**: Es una marca de tiempo (o *time-stamp* en inglés) que sirve para saber cuándo fue tomada/almacenada una lectura. Su tipo de datos es *DATETIME*.

- **node_id**: Es el identificador del nodo que toma la lectura de sensores. Es una clave ajena (*foreign-key*) que referencia a la tabla *NODE*, en concreto a su clave primaria *id*, por lo que su tipo de datos es *INT*.
- **temp**: Es un número decimal que almacena la lectura del sensor de temperatura. Su tipo de datos es *DECIMAL(8,4)*.
- **humi**: Es un número decimal que almacena la lectura del sensor de humedad. Su tipo de datos es *DECIMAL(8,4)*.
- **light**: Es un número decimal que almacena la lectura del sensor de luz ambiente. Su tipo de datos es *DECIMAL(8,4)*.
- **gas**: Es un número decimal que almacena la lectura del sensor de gases contaminantes. Su tipo de datos es *DECIMAL(8,4)*.

El código SQL de generación de las tablas en MySQL es el siguiente:

```
CREATE TABLE `NODE` (
        `id` int(11) NOT NULL AUTO_INCREMENT,
        `name` varchar(45) DEFAULT NULL,
        `pan` int(11) DEFAULT NULL,
        `addr` varchar(20) DEFAULT NULL,
        `location` varchar(45) DEFAULT NULL,
        `node_type` tinyint(4) DEFAULT NULL,
        PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8


CREATE TABLE `READING` (
        `id` int(11) NOT NULL AUTO_INCREMENT,
        `tstamp` datetime DEFAULT NULL,
        `node_id` int(11) DEFAULT NULL,
        `temp` decimal(8,4) DEFAULT NULL,
        `humi` decimal(8,4) DEFAULT NULL,
        `light` decimal(8,4) DEFAULT NULL,
        `gas` decimal(8,4) DEFAULT NULL,
        PRIMARY KEY (`id`),
        FOREIGN KEY (`node_id`) REFERENCES NODE(`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8
```

Como puede apreciarse, el motor de almacenamiento[12] elegido para la base de datos es **InnoDB**. Se ha elegido este frente a MyISAM (que es le motor por defecto) porque InnoDB permite tener un diseño relacional, con características *ACID*[13] y soporte

[12] http://dev.mysql.com/doc/refman/5.5/en/storage-engines.html
[13] http://en.wikipedia.org/wiki/ACID

de transacciones. Además, el motor InnoDB es más eficiente al realizar operaciones de inserción y actualización de los registros en las tablas[14].

4.3.3. *software* de comunicación y almacenamiento de datos de nodos sensores

Se describen a continuación los detalles de diseño e implementación del *software* encargado de recibir, del nodo coordinador, todos los datos generados por los nodos sensores.

Bibliotecas necesarias

El *software* hace uso de las siguientes bibliotecas:

- Processing: por los motivos ya mencionados (facilita la representación visual de datos y el manejo del puerto serie).
- MySQL JDBC: es el conector de bases de datos MySQL para Java. Se usa para hacer las consultas a la base de datos.
- ControlP5: es una biblioteca para Processing que ofrece componentes y *widgets* para añadir interfaces gráficas de usuario a las aplicaciones.
- RXTXcomm: es la biblioteca de acceso al puerto serie para Java, necesaria para poder usar las funciones de manejo del puerto serie con Processing.

Estructura del código

El código está estructurado y dividido en varias clases que encapsulan las diferentes funcionalidades. Las tres funcionalidades básicas son: recepción de datos desde el nodo coordinador, gestión de la base de datos y visualización gráfica de los datos. Al tratarse de un programa Java, se han creado varios paquetes para organizar el código, agrupando las diferentes clases que encapsulan las funcionalidades.

Existe una clase con el programa principal, que ejecuta los dos métodos requeridos por cualquier programa que use Processing: **setup()** y **draw()**. El método *setup()* se ejecuta una vez al inicio del programa, y contiene las sentencias de inicialización de las estructuras de datos y los dispositivos. En este método, básicamente, se configuran el puerto serie y la conexión con la base de datos. El método *draw()* es un bucle infinito que ejecuta las tres funcionalidades descritas: recibe datos del puerto serie, en caso de ser un paquete válido inserta su contenido en la base de datos, y por último llama a los métodos de visualización

[14] http://dba.stackexchange.com/questions/17431/which-is-faster-innodb-or-myisam

de datos. La Figura 4.11 muestra un diagrama de flujo con el sencillo funcionamiento del programa principal.

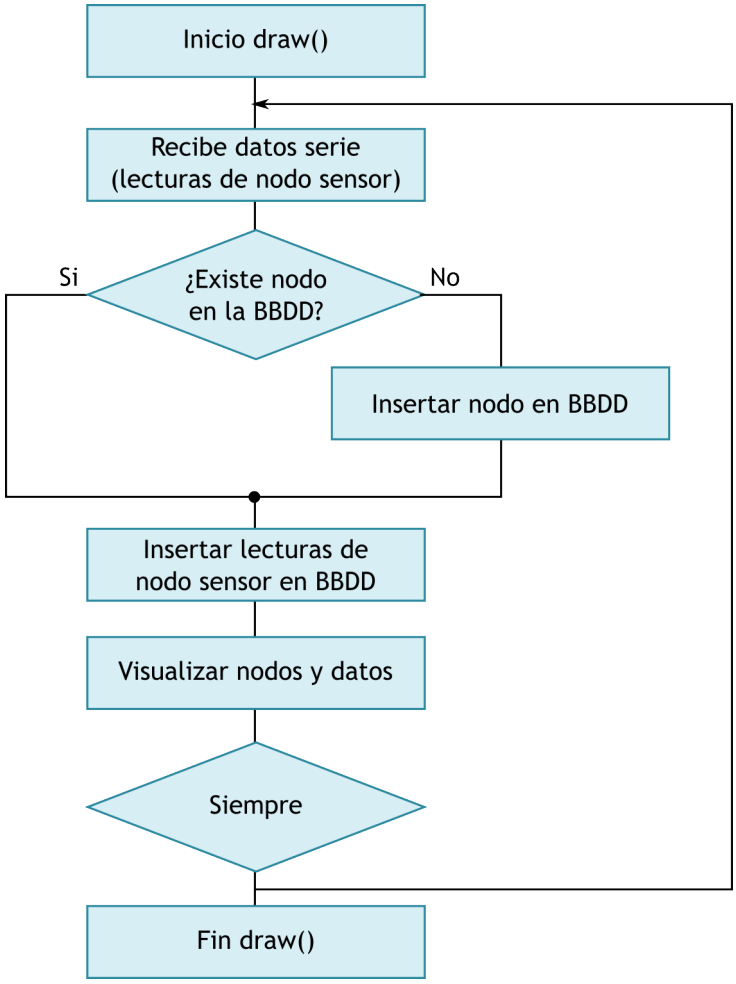

Figura 4.11: Diagrama de flujo para el *software* Java encargado de la recepción de datos de los nodos sensores y de su posterior inserción en la base de datos.

La funcionalidad que recoge el manejo del puerto serie se encapsula en la clase **SerialManager**. Esta clase, además de la gestión del puerto serie desde Java, implementa el mismo protocolo de transferencia de datos serie que ya se había implementado en el nodo coordinador, que puede verse en la sección 4.2.2. Así se dispone de dos implementaciones del protocolo, que sirven de referencia para trabajos futuros: una realizada en C/C++ con Arduino, y otra realizada en Java.

Para el manejo de la base de datos se han creado tres clases: dos de ellas representan una fila de las tablas descritas en el esquema de la base de datos, y la otra recoge la funcionalidad para el manejo de la base de datos. Así se tiene una clase llamada **DBNode** que representa un nodo inalámbrico, y cuyas propiedades son equivalentes a los

campos de la tabla *NODE*. Por otra parte, la clase **DBReading** representa una lectura de datos por parte de un nodo sensor, y cuyas propiedades reflejan igualmente los campos de la tabla *READING*. Por último, la clase **DBManager** hace uso de las clases *DBNode* y *DBReading* y contiene propiedades y métodos que hacen posible insertar, modificar y consultar los registros de la base de datos. Otro objetivo de la clase *DBManager* es el de garantizar, en la medida de lo posible, la integridad de la base de datos, y por ello todas las operaciones SQL de inserción se realizan dentro de bloques *try/catch* que permiten dar marcha atrás en la inserción (hacer *rollback*) ante cualquier error en tiempo de ejecución.

Para evaluar correctamente el envío de los datos desde los nodos sensores, se ha preparado la visualización que muestra la Figura 4.12.

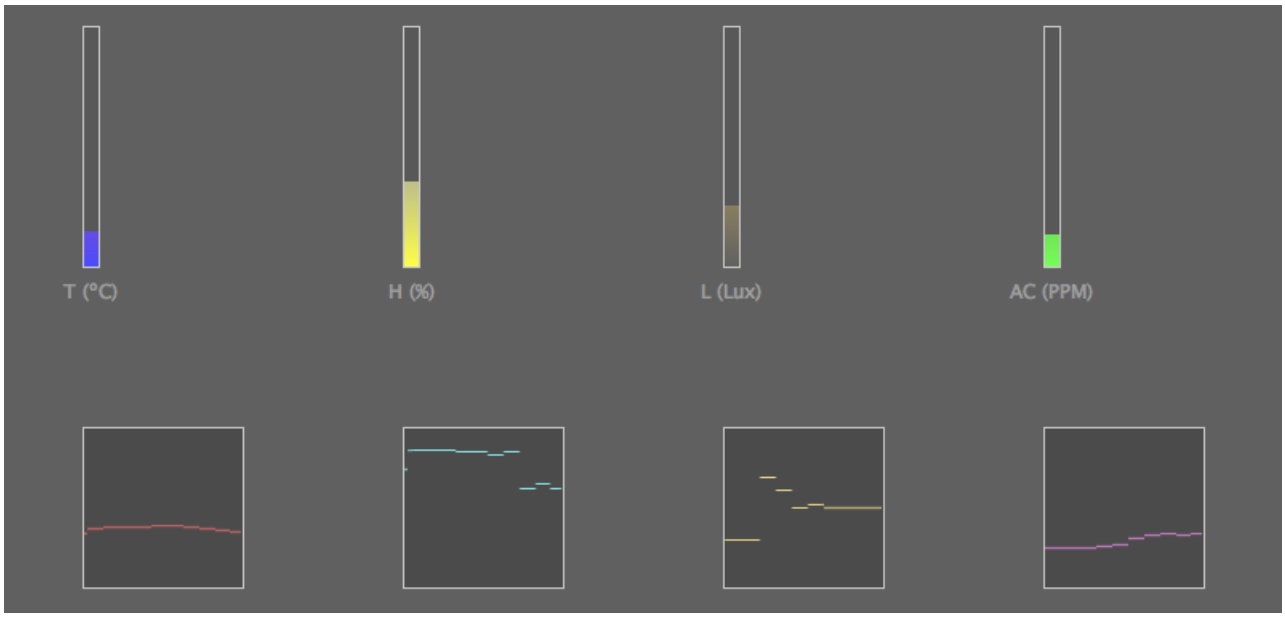

Figura 4.12: *software* de visualización de los datos recibidos desde un nodo sensor.

La Figura 4.13 muestra el diagrama de clases para el programa Java realizado.

4.3.4. Interfaz de acceso web

Para elaborar la web de consultas de la red de nodos sensores, se ha utilizado una sencilla estructura HTML5, cuyo diseño ha sido retocado a través de reglas CSS. Para dotar a la web de cierto contenido dinámico se ha usado la biblioteca **jQuery**[15] junto con

[15]http://jquery.com

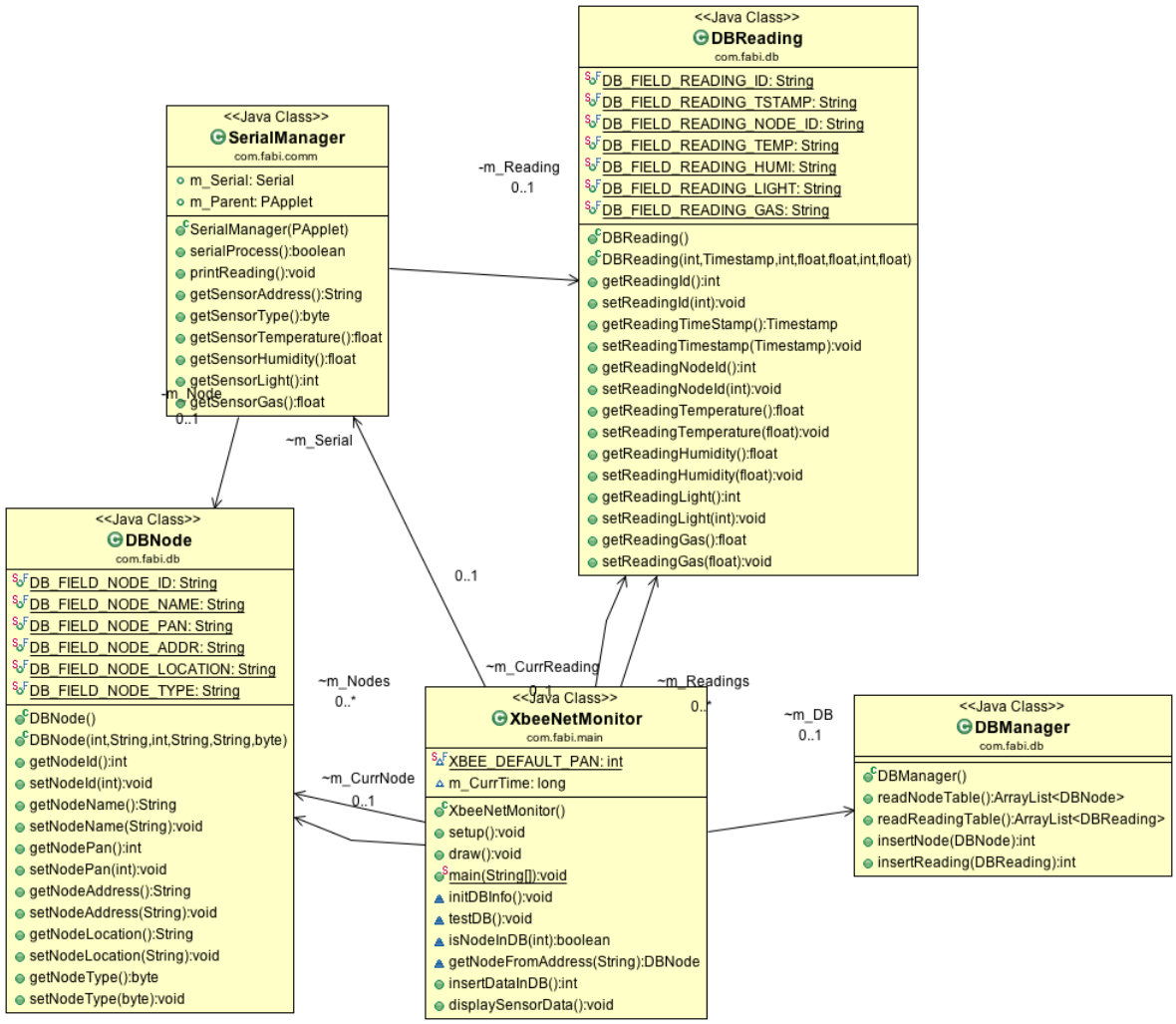

Figura 4.13: Diagrama de clases para el *software* Java encargado de recibir los datos desde el nodo coordinador.

jQuery-UI[16], y **DataTables**[17] para el manejo de tablas con JavaScript.

Se han creado tres páginas:

- **index.html**: es la página de inicio y contiene la tabla con los nodos sensores, permitiendo mostrar información relativa a ellos.
- **readings.html**: esta página contiene la tabla con todas las lecturas realizadas por los nodos.
- **about.html**: esta página contiene un texto básico con el objetivo de la web.

Tanto la página *index.html* como *readings.html* lo primero que hacen al ser cargadas es ejecutar un código JavaScript para realizar una petición al servidor web, concretamente

[16]http://jqueryui.com
[17]http://www.datatables.net

a *scripts* PHP. Estos *scripts* PHP recuperan de la base de datos el contenido de las tablas apropiadas, codifican este contenido en formato *JSON* y lo envían de vuelta a la página. Cuando las páginas reciben el contenido, el código en JavaScript se encarga de visualizar los datos en la página. Para esta visualización, se hace uso de jQuery y DataTables.

La Figura 4.14 muestra la vista de la página de inicio, con la tabla de nodos sensores. En la Figura 4.15 se muestra la vista para la página de lecturas realizadas por los nodos sensores.

Figura 4.14: Página web con el listado de los nodos sensores.

Figura 4.15: Página web con el listado de las lecturas realizadas por los nodos sensores.

Capítulo 5

Conclusiones y trabajos futuros

5.1. Implementación

Se han diseñado e implementado las diferentes piezas conforme a los objetivos y requisitos propuestos. No obstante se trata, en su conjunto, de un sistema muy básico que sin duda necesitará mucho trabajo posterior para adecuarlo mejor a la utilidad final deseada, como podría ser ofrecer servicios o productos en torno al sistema.

No obstante, y desde el punto de vista de ingeniería, el desarrollo de las diferentes piezas, tanto *hardware* como *software*, se ha hecho intentando que sean modulares y con poco acoplamiento. En la gran mayoría de los casos se ha hecho uso de herramientas libres, muy asequibles y con gran cantidad de información disponible, de tal manera que se puede profundizar en cada una de las piezas.

5.1.1. Estimación de costes

Se detalla a continuación una estimación de los costes monetarios asociados al desarrollo de la solución. No se tienen en cuenta los costes ni de ingeniería, ni de tiempo de desarrollo *software*, sino que se estima exclusivamente el precio del *hardware* utilizado para los nodos de la red inalámbrica (tanto sensores como coordinador). Con esta estimación se pretende ofrecer una orientación acerca de la posible viabilidad de un producto basado en los componentes elegidos.

La Tabla 5.1 muestra esta estimación de costes para el *hardware* de un nodo sensor. Aunque se han comparado los precios de varias tiendas de componentes electrónicos, en Internet, para conseguir el menor precio posible, hay que tener en cuenta que los gastos de envío pueden ser relevantes.

Componente	Precio Unidad	Precio 100 unidades
Arduino UNO	20,63€	20,63€
Wireless SD Shield	23,06€	22,12€
Módulo XBee	17,33€	17,33€
Sensirion SHT11	23,79€	15,10€
TAOS TSL2561	2,31€	1,98€
Figaro TGS2600	14,60€	12,85€
2 Transistores	2 x 0,12€	2 x 0,06€
7 Resistencias	7 x 0,01€	7 x 0,0065€
2 Condensadores	2 x 0,02€	2 x 0,014€
TOTAL	102,07€	90,20€

Tabla 5.1: Estimación de costes para el nodo sensor.

Componente	Precio Unidad	Precio 100 unidades
Arduino Leonardo	18,11€	17,56€
Wireless SD Shield	23,06€	22,12€
Módulo XBee	17,33€	17,33€
TOTAL	58,05€	57,01€

Tabla 5.2: Estimación de costes para el nodo coordinador.

Para el caso del nodo coordinador, los costes del *hardware* se muestran en la Tabla 5.2.

Es preciso señalar que estos costes pueden reducirse considerablemente integrando más el diseño electrónico, y fabricando las piezas desde cero. Por poner un ejemplo, y sin tener en cuenta las horas de ingeniería necesarias, se podría preparar una *PCB* que integrase Arduino con el escudo Wireless SD, reduciendo tanto el tamaño de la *PCB* como el número de componentes empleados. Arduino también consta de elementos superfluos de los que se podrían prescindir, como son por ejemplo los LED o el microcontrolador encargado de la comunicación por USB (dado que los nodos sensores sólo se comunican de forma inalámbrica).

5.2. Resultados

Se ha verificado el correcto funcionamiento para cada una de las piezas *hardware* y *software* mencionadas, y también se ha verificado su correcto funcionamiento en conjunto, por lo que los resultados respaldan la solución desarrollada.

En este sentido, es preciso señalar que las pruebas han sido realizadas con material escaso, ya que sólo se ha dispuesto de *hardware* para dos nodos sensores y un nodo coordinador. Además las pruebas no han sido exhaustivas: ha tenido una duración total

de 5 días en una habitación, cuando lo idóneo hubiese sido desplegar un sistema con más nodos en un escenario real (como puede ser un cultivo agrícola) durante más tiempo. Es prácticamente seguro que pruebas más reales y estructuradas ofrecerán resultados más precisos en cuanto a la funcionalidad desarrollada.

5.3. Trabajos futuros

Debido al amplio número de tecnologías utilizadas para el desarrollo del proyecto, hay muchas líneas de mejora, en cada una de las partes. Se enuncian brevemente algunas de las mejoras más significativas para el proyecto.

5.3.1. Nodo sensor

Entre las múltiples mejoras que permite el nodo sensor se destacan las siguientes:

- Mejorar el consumo eléctrico usando las funcionalidades que ofrece Atmel, como los *sleep-modes* o el *watchdog timer*.
- Mejorar las caracterización o modelado del sensor de gases, que en este proyecto se ha realizado de forma liviana.
- Incrementar la tasa de transferencia para la comunicación entre los nodos, que actualmente está configurada en 9600bps.
- Integrar más la electrónica, no usando las partes de Arduino que no se necesiten.

5.3.2. Nodo coordinador

Entre las múltiples mejoras que permite el nodo sensor se destacan las siguientes:

- Mejorar el consumo eléctrico e incrementar la tasa de transferencia de comunicaciones, de forma análoga a lo que se ha señalado para el nodo sensor.
- Incorporar sensores al coordinador, y tener de esta forma un nodo sensor más.
- Se podría suplir tanto este nodo coordinador como el servidor mediante el uso de sistemas electrónicos más potentes, como por ejemplo BeagleBone o RaspberryPi.

5.3.3. Servidor

De forma similar, el servidor y sus servicios podrían mejorarse en los siguientes aspectos:

- Optimizar la base de datos, quizá montando un *cluster MySQL* en el caso de tener muchos sensores.
- Optimizar Nginx para que haga un uso más adecuado de la máquina.
- Mejorar el *software* de visualización y configuración de la red, permitiendo la gestión de muchos nodos de forma más organizada.
- En el *software* de recepción de datos de los sensores, también podrían establecerse algunos parámetros configurables, como son los rangos máximos y umbrales para cada medida.

Bibliografía

[AMS05] AMS: *Datasheet TSL2561-TSL26560, Light to digital converter.* `http://www.ams.com/eng/content/download/250096/975518/143687`, Diciembre 2005. Visitada en Marzo de 2013.

[Ana09] Anastasi, Giuseppe: *Energy Conservation in Wireless Sensor Networks: a Survey.* Ad Hoc Networks, 07:537–568, 2009.

[Atm05] Atmel: *Datasheet ATmega328P, 8Bit microcontroller.* `http://www.atmel.com/Images/doc8161.pdf`, Enero 2005. Visitada en Febrero de 2013.

[Ave10] Avelli, Cristina: *Diseño e implementación de un nodo sensor autónomo e inalámbrico y del software de monitorización.* URJC, 2010.

[Bha10] Bhattacharjee, Dipanjan: *Design and Development of Wireless Sensor Node.* IJCSE, 02:2431–2438, 2010.

[Cas10] Castañeda, Natalia Prieto: *Implementación de un sistema de evaluación sensorial electrónico para el control de calidad de vinos.* Tesis de Doctorado, Universidad de Valladolid. Facultad de Ciencias, Valladolid, España, 2010.

[Com09] Community, Nginx: *Nginx documentation.* `http://nginx.org/en/docs`, Febrero 2009. Visitada en Marzo de 2013.

[Dig13] Digi: *Datasheet XBee-XbeePro ZigBee RF modules.* `http://ftp1.digi.com/support/documentation/90000976_P.pdf`, Febrero 2013. Visitada en Marzo de 2013.

[Eck06] Eckel, Bruce: *Thinking in Java, 4th Ed.* Prentice Hall, Febrero 2006, ISBN 9780131872486.

[Fal10] Faludi, Robert: *Building Wireless Sensor Networks: with ZigBee, XBee, Arduino, and Processing.* O'Reilly Media, California USA, 1ª edición, Diciembre 2010, ISBN 9780596807733.

[Fig05] Figaro: *Datasheet TGS2600, Air contaminant gases sensor.* `http://www.figarosensor.com/products/2600pdf.pdf`, Enero 2005. Visitada en Marzo de 2013.

[Fla11] Flanagan, David: *JavaScript, the definitive guide, 6th Ed.* O'Reilly Media, Marzo 2011, ISBN 9780596805524.

[Igo07] Igoe, Tom: *Making Things Talk: Practical Methods for Connecting Physical Objects.* Maker Media, Octubre 2007, ISBN 9780596510510.

[Kop03] Kopka, Helmut: *Guide to LaTeX, 4th Ed.* Addison-Wesley Professional, Diciembre 2003, ISBN 9780321173850.

[Moh10] Mohaiyedin, Idris: *Development of wireless sensor network for monitoring indoor air quality.* `http://koleksi.uitm.edu.my/digital_public/THESIS24/MOHAIYEDIN%20IDRIS%2010_24.pdf`, 2010. Visitada en Abril de 2013.

[RF07] Reas, Casey y Ben Fry: *Processing: A Programming Handbook for Visual Designers and Artists.* The MIT Press, Agosto 2007, ISBN 9780262182621.

[Sen08] Sensirion: *Datasheet SHT1x, Humidity and temperature sensor.* `http://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/Humidity/Sensirion_Humidity_SHT1x_Datasheet_V5.pdf`, Julio 2008. Visitada en Abril de 2013.