ACCELBIKE

RODRIGO CLAUDIO MIGUEZ REIN
DAVID MUÑOZ LORENZO
ALEXIS VIZCAYA HERVELLA

GRADO EN INGENIERÍA DEL SOFTWARE. FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin Grado en Ingeniería del Software

17 de Junio de 2016

Directores:

José Ignacio Gómez Pérez
Luis Piñuel Moreno

Autorización de difusión

17 de Junio de 2016

Los abajo firmantes, matriculados en el Grado en Ingeniería del Software de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores el presente Trabajo Fin de Grado: "ACCELBIKE", realizado durante el curso académico 2015-2016 bajo la dirección de José Ignacio Gómez Pérez y Luis Piñuel Moreno en el Departamento de Arquitectura de Computadores y Automática, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Rodrigo C. Míguez Rein David Muñoz Lorenzo Alexis Vizcaya Hervella

Resumen en castellano

Este proyecto consiste en el diseño e implementación de un dispositivo que permite el registro de los datos de acelerometría de un trayecto realizado por un ciclista para medir el esfuerzo experimentado por éste. Este dispositivo se fijará al manillar o la tija de una bicicleta de montaña y se complementará con la información proporcionada por el GPS de un móvil.

Los datos de acelerometría registrados por el dispositivo se enviarán a dicho dispositivo móvil mediante Bluetooth Low Energy, cuyo funcionamiento se explicará durante el transcurso de esta memoria.

Palabras clave

Bluetooth Low Energy, Bluetooth Smart, I2C, Android, App, ARM mbed, Ciclismo, Acelerómetro

Abstract

This project consists in presenting the design and implementation of a specific device which allows for registering accelerometry data obtained from a route made by a cyclist, for measuring the effort that he/she experiences. This device will be fixed to the handlebar or the seatpost of a mountain bicycle and it will be complemented by information that is provided by the mobile phone's GPS.

The data registered by the sensor will be sent to the mobile phone using Bluetooth Low Energy, whose functioning is also explained in this memory project.

Keywords

Bluetooth Low Energy, Bluetooth Smart, I2C, Android, App, ARM mbed, Cycling, Accelerometer

Índice general

Índice de figuras

Agradecimientos

A nuestras familias y amigos, que nos han estado apoyando todo este tiempo, y con los que no habría sido posible llegar hasta aquí.

A nuestros tutores, que nos han motivado a seguir adelante durante este año.

Capítulo 1

Introducción

Entre los años 2008 y 2009 se comenzó a usar el concepto Internet de las cosas (abreviado IoT) como una apuesta hacia el futuro, cuyo objetivo era interconectar dispositivos digitales a internet.

Esto ya es una realidad, con mayor frecuencia encontramos nuevos dispositivos capaces de conectarse a internet, permitiendo al usuario su manejo desde cualquier parte del mundo. Con ello se abre camino a nuevas oportunidades haciendo la vida más cómoda al usuario y proporcionándole mayor seguridad y control.

Según empresas del sector tecnológico, en 2016 se espera que haya más de 6.000 millones de dispositivos basados en este concepto.

El concepto del internet de las cosas agrupa múltiples capacidades, algunas de ellas han sido utilizadas en el proyecto, estas se nombran a continuación:

Comunicación y cooperación: Dispositivos capaces de conectarse a la red de internet y entre ellos, intercambiando datos y comunicándose con servidores.

Direccionamiento: Los objetos serán localizables y configurables desde cualquier lugar de la red de internet.

Identificación: Los objetos se identificarán en la red por medio de tecnologías como RFID (Radio Frecuency Identification), códigos de barras ópticos, NFC (Near Field Communication) y otras muchas formas.

Localización: Se podrá saber la ubicación física del dispositivo en cualquier momento.

El proyecto nace con el objetivo de emprender en esta tecnología, para ello se quiere conectar entre sí un sensor de acelerometría y un microprocesador para recibir y procesar sus datos. Dicho microprocesador debe estar provisto de Bluetooth de Bajo Consumo (en inglés, Bluetooth Low Energy o BLE) para poder enviar los datos vía Bluetooth a una aplicación móvil desarrollada en Android creada por los alumnos.

Existe una amplia variedad de dispositivos de bajo coste que podrían considerarse, ya que el mercado de los microprocesadores de bajo consumo está ahora en expansión y hay gran oferta de estos dispositivos, con entornos de desarrollo que ofrecen distintas alternativas y que previsiblemente irán reduciéndose en los próximos años cuando predomine una tecnología que se imponga a las demás.

Unas de las empresas más especializadas en este tipo de tecnologías y que resaltan en este mercado son **Nordic Semiconductor, Cypress Semiconductor, Texas Instruments (TI), Cambridge Silicon Radio (CSR)** y **Dialog Semiconductor**.

Destacamos 2 empresas que nos han ofrecido mejores soluciones de microprocesadores de bajo consumo según las necesidades del proyecto:

Nordic está a la vanguardia de la revolución inalámbrica y son especialistas en Radiofrecuencia de consumo ultra bajo (ULP) redefiniendo nuevos procesadores por su bajo consumo, precios competitivos y aumentando la potencia de estos. Esta empresa de semiconductores integra procesadores basados en la arquitectura ARM de 32 y 64 bits que predominan en el mercado de la electrónica. Son procesadores relativamente simples siendo ideales para aplicaciones de bajo consumo y tiene un precio económico.

Cypress es una empresa especializada en ofrecer soluciones de alta calidad para sistemas integrados en el sector aeroespacial, automoción, industria, redes, telecomunicaciones y electrónica de consumo, entre otros campos. Cubre una amplia gama de productos entre los cuales destacan los microprocesadores, memorias, sistemas programables en chips , lo que denominan (en inglés Programable System on Chip o PSOC), controladores táctiles capacitivos y controladores inalámbricos Bluetooth de bajo consumo.

1.1. Entornos de desarrollo

Con los dispositivos de bajo consumo nacen también multitud de entornos de desarrollo específicos para desarrollar todo su potencial, lo que dificulta la exploración de hardware, pues es necesario trabajar con un nuevo entorno cada vez.

mbed, creada por la empresa ARM, es una plataforma de desarrollo *online* que intenta hacer frente a esta variedad de entornos facilitando el desarrollo de prototipos de sistemas basados en microprocesadores.

Esta plataforma se creó para promover el Internet de las Cosas (IoT) y desarrollar proyectos mediante herramientas de construcción de hardware y software que aceleren el desarrollo de dispositivos basados en ARM.

Su enfoque está basado en la *nube* y permite codificar, añadir librerías y compilar desde un navegador web. La ventaja de programar desde cualquier navegador es que permite tener accesibilidad desde cualquier punto del mundo sólo con conexión a internet. Durante el Capítulo 4 se explicará con más detalle este entorno y cómo nos ha ayudado a llevar a cabo este proyecto.

Todo lo anterior hace de la compatibilidad con mbed una característica atractiva a la hora de elegir un dispositivo para llevar a cabo nuestro proyecto, lo que nos ha ayudado a elegir la placa de desarrollo nRF51-DK de Nordic.

1.2. Bluetooth Smart: Bluetooth de bajo consumo

Uno de los rasgos fundamentales de IoT es la conectividad. Para lograrla tenemos a nuestra disposición una gran variedad de tecnologías inalámbricas, entre ellas *zygbee*, *narrow-band* o *WiFi*; pero la que sin duda alguna ofrece el menor consumo de energía posible es *Bluetooth Low Energy* (BLE, también denominado *Bluetooth Smart*). Esta tecnología nace del Bluetooth clásico, pero marca su propio camino al sacrificar ancho de banda y períodos de envío continuados para lograr la meta de ser muy eficiente energéticamente, lo que lo hace perfecto para dispositivos que necesitan funcionar con una batería pequeña durante

largos períodos. El hecho de ser parte de Bluetooth y poderse complementar con él hace que actualmente esté presente en todos los teléfonos móviles del mercado. A esto ayuda que grandes empresas del mundo de la tecnología y las telecomunicaciones respalden BLE.

En el Capítulo 2 hablaremos de cómo se estructura e implementa Bluetooth Low Energy.

1.3. Especificación del objetivo del proyecto

El objetivo de este proyecto consiste en medir el esfuerzo realizado por un ciclista durante un trayecto. Esto se consigue combinando los valores proporcionados por un acelerómetro y los datos de posicionamiento por GPS para poder elaborar un mapa del trayecto en el que se indiquen las zonas que, por pendiente, estado del terreno, etc. supusieron un mayor esfuerzo. Además se ofrecerá más información que pueda ser calculada con estos datos, como la velocidad media o la distancia total recorrida, por ejemplo.

Para la comodidad del usuario, toda la información de la ruta se recogerá en su dispositivo móvil, del que se obtendrán los datos de localización y que se conectará a través de Bluetooth Smart con el sensor de acelerometría. De este modo, con un simple gesto podrá visualizar fácilmente los datos más relevantes.

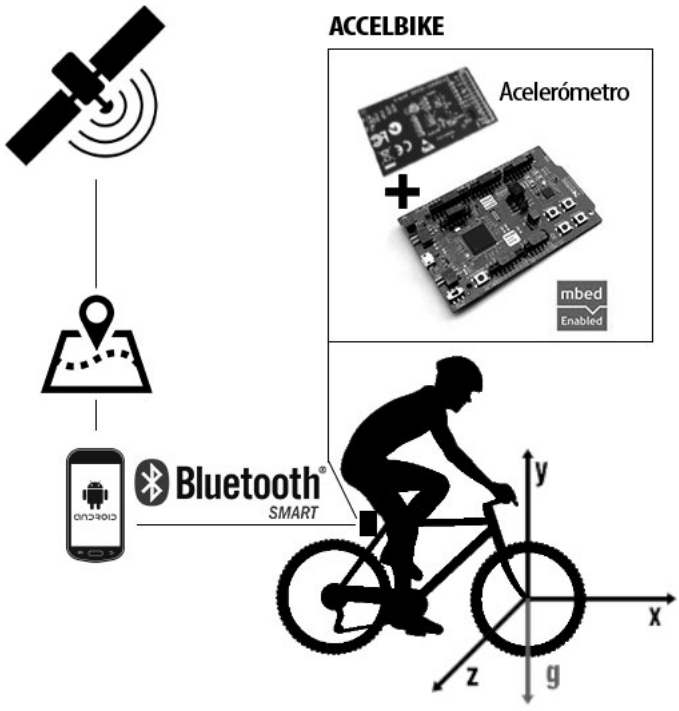

Figura 1.1: *Esquema del funcionamiento del proyecto. El dispositivo móvil recoge datos tanto de la localización como de un sensor de acelerometría.*

1.4. Contribuciones Personales

1.4.1. Rodrigo Claudio Miguez Rein

La tarea principal que he realizado durante este proyecto es la de la programación de la placa nRF51-DK de Nordic para configurar la recepción de las muestras de acelerometría utilizando el bus de datos I2C y el envío de éstas por medio de la tecnología Bluetooth Low Energy, además de diseñar y programar la parte BLE de la aplicación Android.

En cuanto elegimos este proyecto, empecé a recopilar información junto con mis compañeros sobre las distintas soluciones *System on Chip* con Bluetooth Smart integrado disponibles en el mercado. Para ello me informé sobre las tecnologías disponibles para realizar la conexión con un acelerómetro y sobre qué características necesitábamos en términos de procesamiento, memoria RAM y flash.

Durante este período también comencé a investigar sobre el funcionamiento y las carac-

terísticas de Bluetooth Low Energy, buscando material de referencia para aprender todo lo posible sobre ésta.

Una vez escogimos dos placas de desarrollo, el kit CY8CKIT-042-BLE[8] de la empresa Cypress Semiconductor y el modelo nRF51-DK de Nordic Semiconductor, comenzamos la fase de pruebas para familiarizarnos con las tecnologías que íbamos a usar. En mi caso, realicé las pruebas con la placa de Cypress. Para ello busqué tutoriales en la página web oficial de la empresa[10] y encontré un repositorio en GitHub con 100 proyectos de ejemplo[5] que me ayudaron en gran medida para entender la programación del kit y los métodos para utilizar Bluetooth Low Energy y para realizar la comunicación con otros dispositivos. Descargué la aplicación móvil *CySmart*, que permite probar los códigos de ejemplo con una interfaz limpia y clara, que además ofrece su código fuente, lo que me sirvió para comprobar cómo se realiza el enlace por Bluetooth Low Energy en el lado de Android.

Entre los ejemplos disponibles para esta placa se encuentran: un sistema de alerta, que consiste en el envío de una señal al dispositivo para apagar o encender un LED y para hacer que parpadee, una aplicación de sensor de proximidad, que aprovecha un circuito integrado en la placa de desarrollo diseñado para este fin, enviando el valor recogido por el sensor al móvil, donde, con la aplicación CySmart mencionada anteriormente, se puede visualizar como una escala.

Durante la fase de pruebas realicé el código que permitía al CY8CKIT-042-BLE leer por SPI los datos recogidos de un circuito, consistente en un conversor analógico-digital contectado a una fotoresistencia, y enviarlos por Bluetooth Low Energy a una aplicación de prueba que diseñé en Android modificando los ejemplos disponibles en el apartado BLE de la página Android Developers. Seguidamente comencé con las pruebas para recibir por I2C los datos del acelerómetro. Estas pruebas se detallarán más adelante en esta memoria.

Cuando elegimos la placa de Nordic para realizar la fase final, escogí continuar la parte del proyecto que se centraba en la programación de ésta, ya que me había resultado muy

interesante la experiencia con el kit de Cypress. Realicé la conexión por I2C de Xtrinsic Sense-Board, una placa que contiene un chip de acelerometría. Para ello busqué su *datasheet*[6] para saber qué pines utilizaba, de qué manera recolectaba las muestras y cómo había que realizar la comunicación entre nRF51-DK y este periférico para lograr una transmisión correcta de los datos. Una vez pude comprobar que se recibía la información de acelerometría, elaboré un sistema para filtrar el ruido presente en las muestras y realicé la configuración de la placa de Nordic para que las transmitiera utilizando Bluetooth Low Energy.

En la parte de la aplicación Android, ayudé con el diseño de las clases que se iban a utilizar elaborando diagramas, y proporcioné *feedback* a mis compañeros para hacerles saber en qué formato se iba a mandar la información por Bluetooth para que pudieran configurar en consecuencia la lectura de éstos en el móvil.

1.4.2. David Muñoz Lorenzo

Mi labor en el proyecto en su fase inicial, como la de mis compañeros, fue analizar en detalle los componentes que íbamos a utilizar para la elaboración del sistema de acelerometría. En primer lugar debíamos encontrar en internet los microprocesadores Bluetooth Low Energy que dieran solución al objetivo del proyecto. Se planteó hacer la búsqueda eligiendo los componentes más económicos y con las prestaciones suficientes para realizar el proyecto, siendo elaborado como producto final. Analicé en profundidad microprocesadores de diferentes marcas y características que debían ser compatibles con las conexiones SPI e I2C.

Se decidió elegir 2 placas de desarrollo, nRF51-DK de Nordic Semiconductor y CY8CKIT-042-BLE de la empresa Cypress Semiconductor, que nos ofrecían dichas conexiones, así como Bluetooth de bajo consumo y múltiples periféricos, lo cual eran soluciones más que suficientes para empezar.

Para realizar la primera prueba con la placa de Nordic, tuvimos que crear un sencillo circuito para recoger datos desde una fotoresistencia, y convertir la señal que recibíamos en

una señal digital para poder enviársela a la placa de desarrollo.

Participé en la creación de dicho circuito así como en las pruebas de desarrollo para realizar la conexión por SPI y recibir los datos que transmitía la fotoresistencia.

Tuvimos que familiarizarnos con el entorno de desarrollo llamado mbed que utilizaba la placa de Nordic. En su repositorio[11] pudimos probar ejemplos de conexiones SPI que adaptamos a la placa utilizada, mirando en su ficha técnica[7] la correcta conexión de sus pines.

La manera de interactuar con la fotoresistencia a través de la placa fue activar los LEDs que esta contiene, alternándolos según su valor de entrada. De esta forma verificamos que el paso de datos funcionaba correctamente.

La siguiente prueba de conexión fue a través de I2C, tuvimos que investigar y documentarnos bien sobre este bus de datos. Se cambió el circuito por un acelerómetro que conectamos por I2C a la placa de desarrollo.

En este punto fue necesario investigar la forma de crear una aplicación en Android, de tal forma que esta pudiera conectarse por Bluetooth Low Energy a la placa nRF51-DK de Nordic y visualizar los datos recibidos. Para realizar la prueba de conexión por Bluetooth encontramos en el repositorio de mbed algunos ejemplos que nos ayudaron a poder implementar el módulo BLE.

Realicé una maqueta visual (*mockup*) de cada vista para poder tener un primer diseño de la aplicación en el que fijarnos. Participé en la creación de la aplicación en Android desarrollando la interfaz de usuario y su estructura. Estructuré la aplicación con un tipo de componente de Android llamado Fragment que se usa mucho en la actualidad y que sirve para separar las diferentes vistas para que puedan tener su propia lógica de manera independiente. Quise darle un aspecto y usabilidad propios de las nuevas aplicaciones de última generación utilizando componentes gráficos de Material design de Google[9]. De esta forma nuestra aplicación se beneficia de los últimos avances visuales asegurando una óptima experiencia de usuario.

Realicé pruebas obteniendo los datos del GPS del móvil y realizando la petición a través de un servicio interno. Utilicé la interfaz LocationListener que nos permite conectarnos con los servicios de ubicación del dispositivo móvil. Pude obtener la latitud y longitud de la ubicación en la que nos encontramos y a través de unos parámetros podemos cambiar la distancia (metros) que actualiza el GPS así como configurar la frecuencia de muestreo (milisegundos) de las coordenadas. Posteriormente se utilizó un hilo para poder recibir los datos del acelerómetro aunque la aplicación estuviera activa en segundo plano.

Debíamos mostrar el circuito recorrido por un ciclista desde que se pone en marcha hasta que finaliza su recorrido. Para visualizar el mapa utilizamos la API de Google Maps[1]. Para poder utilizar sus funcionalidades principales tuve que registrarme en la API de Google Maps, incorporamos al proyecto una clave única que para tener acceso a las funcionalidades de su servidor. Activé los permisos necesarios para obtener el mapa de Google y sus configuraciones necesarias para ajustarlo a nuestra aplicación. Utilizando la clase polilíneas de Google pude crear las líneas sobre el mapa, se pudo visualizar con éxito en la vista Actividades.

1.4.3. Alexis Vizcaya Hervella

Mi contribución al proyecto ha sido principalmente la de estructurar el código de la aplicación Android, centrándome en la parte del análisis de los datos y los cálculos realizados con ellos, así como de la lectura y escritura de los mismos.

En el primer cuatrimestre iniciamos en conjunto una investigación general haciendo un análisis de los dispositivos que incorporaran la tecnología Bluetooth Low Energy (BLE) disponibles en el mercado que se ajustaran a los requisitos del proyecto, comparamos precios, características, etc. y finalmente escogimos las placas de desarrollo de Cypress (PSoC 4 BLE) y Nordic (nRF51-DK). Una vez elegidas las placas, investigué los buses de datos I2C y SPI puesto que estas eran las tecnologías disponibles en ambas para la comunicación con un acelerómetro. Además inicie el proceso de familiarización con los entornos de desarrollo.

En una primera etapa realice ejemplos con la placa de Cypress en su entorno de desarrollo (PSoC Creator) ya que descubrimos la existencia de un repositorio en GitHub que contenía 100 proyectos de ejemplo, probé con varios ejemplos para comprender la forma de diseñar pequeños programas en dicha placa, además disponía de una app para el teléfono móvil para poder conectar con la placa y así poder realizar estos ejemplos.

Con la placa de Nordic ocurrió algo similar, con su entorno de desarrollo mbed, cuyo repositorio oficial contiene multitud de ejemplos, con los que empecé a investigar acerca del funcionamiento de esta. Como por ejemplo uno con el que empecé a establecer la conexión Bluetooth con el móvil, consistente en detectar la pulsación de un botón en la placa y mandar la información del estado de éste a una app.

Una vez finalizada esta etapa de aprendizaje hicimos pruebas de conexión entre la placas y un circuito MCP mediante SPI. El circuito constaba de una fotorresistencia conectada a un ADC con la que comprobamos que al cambiar la luminosidad que recibía la resistencia los datos que obteníamos en la placa variaban.

El siguiente paso fue conectar un acelerómetro mediante I2C a la placa de Nordic. Pudimos comprobar que los datos se recibían correctamente a través de los LEDs de los que disponía la placa, y una vez comprobado el correcto funcionamiento, nos dispusimos a realizar la conexión mediante BLE entre una pequeña app y la nRF51-DK.

Una vez decidida la placa con la que íbamos a afrontar el proyecto, comenzamos con el diseño de la aplicación Android. Dado que nunca había programado una app para móviles, y ya que esta iba a ser para dispositivos Android, empecé un proceso de aprendizaje sobre esta plataforma y del entorno de desarrollo *Android Studio*. Para ello encontré ejemplos en la página oficial de Google, Android Developers, y diversos tutoriales, que me ayudaron a empezar a hacer pequeñas aplicaciones para comprender su funcionamiento.

Una vez asimilados los conceptos básicos, formé parte del diseño de la estructura mediante un patrón Modelo Vista Controlador (MVC) y en el desarrollo del menú principal de la aplicación.

A partir de aquí me centré en la parte de lectura y escritura de la información recibida a través de Bluetooth Low Energy y el GPS del móvil, decidiendo cómo se iban a llevar a cabo estos dos procesos. Sobre la lectura desarrollé la funcionalidad por la cual se finaliza una sesión y se traslada toda la información a un vista con un mapa de Google Maps con la ruta dibujada en forma de polilínea y una serie de campos para mostrar la información procesada. Sobre la escritura, elegí la forma de guardar los datos en la memoria externa del teléfono móvil. El proceso de guardado se realiza periódicamente, recogiendo los últimos valores recibidos, tanto del GPS como del acelerómetro, escribiendo una línea nueva en el archivo con esta información.

Para recoger los datos de las diferentes fuentes, programé la estructura de hilos que se utiliza en la aplicación, ya que teníamos que tener un control para asignar diferentes periodos de tiempo, debido a que por ejemplo no es necesario recopilar la información del GPS tan a menudo como la del acelerómetro.

Capítulo 2

Bluetooth Low Energy

Bluetooth Low Energy (BLE, también denominado Bluetooth Smart) empezó como parte de la Bluetooth 4.0 Core Specification. Diseñado originalmente por Nokia bajo el nombre de Wibree, fue luego adoptado por el Bluetooth Special Interest Group (SIG).[13][12][4]

El objetivo de Bluetooth Low Energy es complementar al Bluetooth clásico y ser la tecnología inalámbrica con el menor consumo energético posible. También tiene como objetivo llegar a una gran cantidad de objetos que hasta ahora no disponían de ninguna conectividad inalámbrica. Para lograr abarcar tantos escenarios, se necesita tener un coste muy bajo, lo cual se consigue con tres elementos clave:

- **Banda ISM:** BLE utiliza la banda ISM (Industrial, Scientific and Medical) de 2,4GHz, que, a pesar de tener numerosos problemas como poco rango o interferencias por otras tecnologías como WiFi, etc, está disponible en todo el mundo con las mismas reglas y no tiene requisitos de licencia.

- **Licencia:** Cuando la tecnología Wibree estuvo lo suficiente madura como para unirse con un grupo de estándares inalámbricos, Nokia decidió elegir al Bluetooth Special Interest Group por su excelente reputación y su política de licencias. Esta política, comparada con la de otros grupos bajo la política Fair, Reasonable and Non-discriminatory, significa que los costes de licencia de un dispositivo Bluetooth se reducen significativamente. Con un menor coste de licencia se reduce el coste por dispositivo.

- **Bajo consumo:** La mejor forma de producir un dispositivo con bajo coste es reducir el coste de sus materiales, por ejemplo las baterías. Un dispositivo BLE puede funcionar perfectamente alimentado por una simple pila de botón CR2032.

2.1. Comparación con Bluetooth Clásico

Aunque comparta con Bluetooth su nombre y mucha de la tecnología utilizada, se deben considerar diferentes tecnologías, ya que tienen objetivos de diseño completamente diferentes.

El Bluetooth clásico (también conocido como *BR/EDR*[4]) se diseñó para conectar diferentes dispositivos, empezando por teléfonos móviles y ordenadores, y con el tiempo añadiendo más casos de uso como auriculares inalámbricos, streaming de música, impresión inalámbrica, etc. Cada uno de estos casos requería cada vez más ancho de banda, por lo que se fueron añadiendo radios cada vez más rápidas (la primera versión transmitía a 1 Mbps, aumentando a 3 Mbps en la versión 2.0, hasta los cientos de megabits por segundo en su versión 3.0)

Bluetooth Low Energy está optimizado para tener un consumo de energía muy bajo, por lo que se reduce el ancho de banda, y está pensado para aplicarse en dispositivos con un coste muy bajo, con una complejidad a su vez baja. Cada capa de su arquitectura se ha optimizado para reducir el consumo de energía al realizar una determinada tarea. Por ejemplo, relajando los parámetros de la radio usada por la capa física, comparándola con la radio del Bluetooth clásico, se puede usar menos energía cuando se está transmitiendo o recibiendo datos.

Precisamente por tener estos objetivos tan diferentes, Bluetooth clásico y BLE pueden coexistir en la misma aplicación en lo que se denomina *Modo Dual* (en inglés *Dual-Mode*), de modo que se puede conectar por ejemplo a unos auriculares inalámbricos por medio de BR/EDR mientras se accede a un dispositivo de bajo consumo, como un *wearable*, por BLE.

2.2. Tecnología

Un dispositivo BLE se divide en tres partes: controlador, anfitrión y aplicación. Cada una de estas partes se divide a su vez en distintas capas que proveen la funcionalidad necesaria para operar. En la Figura 2.1 se muestra un esquema de esta estructura.

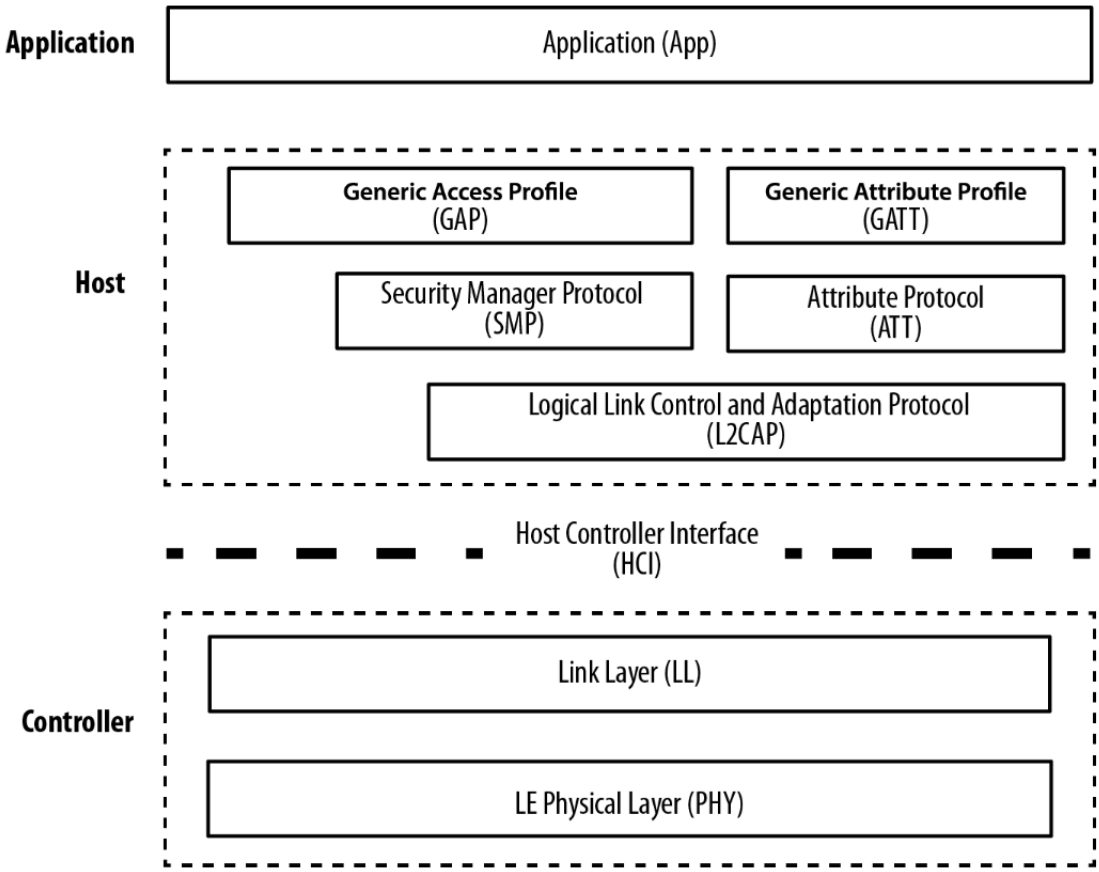

Figura 2.1: *Estructura hardware de un dispositivo BLE*

2.2.1. Controlador

Capa Física (Physical Layer, PHY). La capa física es la que contiene los circuitos capaces de comunicarse de forma analógica, modulando y demodulando las señales analógicas y transformándolas en digitales. Como ya se ha mencionado, la radio usada para BLE utiliza la banda ISM de 2,4 GHz. Esta banda se divide en 40 canales, desde los 2,4000 GHz

hasta los 2,4835 GHz. De estos canales, 37 se utilizan para el envío de datos y los 3 restantes como canales de anuncio, para iniciar conexiones y mandar datos de broadcast.

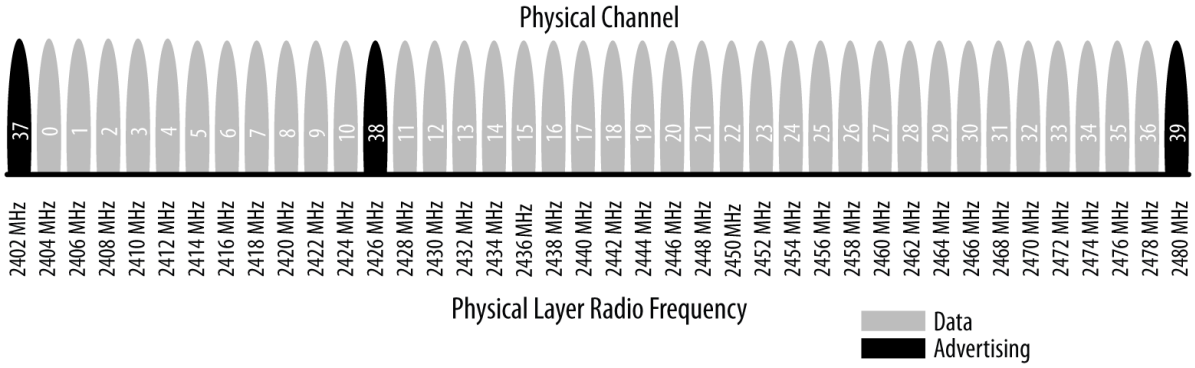

Figura 2.2: *División de la banda de frecuencia ISM en 40 canales*

Para evitar interferencias con otros dispositivos transmitiendo en la misma banda, mayormente WiFi y Bluetooth clásico, el estándar BLE utiliza una técnica llamada *frequency hopping spread spectrum*, que hace que la radio "salte" entre los distintos canales en cada conexión.

Capa de Enlace (Link Layer). Es la parte que actúa de interfaz de la capa física. Normalmente se implementa como una mezcla de software y hardware personalizado. Es responsable de anunciar, escanear, crear y mantener las conexiones. También se encarga de asegurar que los paquetes están correctamente estructurados. Por ello es probablemente la capa más compleja de la arquitectura BLE.

Interfaz Anfitrión/Controlador (Host/Controller Interface, HCI). Muchos dispositivos separan el Anfitrión del Controlador, y para estos casos, el HCI provee una interfaz estandarizada para comunicar los dos dependiendo del método de transporte físico de los datos. Entre estas interfaces se incluyen USB, SDIO o variantes de UART.

2.2.2. Anfitrión

Control Lógico de Enlace y Protocolo de Adaptación (Logical Link Control and Adaptation Protocol, L2CAP). Esta capa provee dos funcionalidades: actúa como

capa multiplexora de los protocolos, es decir, recibe los protocolos de las capas superiores y los encapsula en el formato de paquete estándar BLE (y viceversa). Asimismo, fragmenta los paquetes que se le envían desde las capas superiores para que quepan en los 27 bytes que tienen como máximo los paquetes de transmisión y vuelve a combinar los paquetes fragmentados que recibe.

Protocolo de Atributos (Attribute Protocol, ATT). Define una serie de reglas para acceder a los datos de un dispositivo. Propone una manera de encapsular estos datos en forma de *atributos*, que permiten identificarlos asignando a cada uno un handle de 16 bits y un Identificador Único Universal (Universal Unique Identifier, UUID) y restringir su acceso mediante una serie de permisos. Los campos que forman un atributo se explicarán más adelante en la Sección 2.4.2. En resumen, ATT un protocolo muy simple, en el que un cliente puede acceder a los atributos de un servidor.
Para definir la comunicación eficaz de estos atributos entre el cliente y el servidor, ATT define seis tipos de mensajes:

- Solicitudes enviadas desde el cliente al servidor.
- Respuestas del servidor a una solicitud del cliente.
- Comandos enviados del cliente al servidor que no necesitan respuesta.
- Notificaciones enviadas desde el servidor al cliente que no necesitan confirmación.
- Indicaciones enviadas desde el servidor al cliente.
- Confirmaciones enviadas por el cliente en respuesta a una indicación.
- De este modo ambos pueden establecer una comunicación con mensajes que requieran o no respuesta.

Perfil de Atributo Genérico (Generic Attribute Profile, GATT). Este perfil se asienta en el Protocolo de Atributos, y define los tipos de atributos y cómo se utilizan. Para

ello utiliza una jerarquía de capas y un modelo de abstracción de datos. Los datos ahora se encapsulan en servicios, que a su vez consisten en una o más características que pueden estar definidas por un descriptor. Estas capas se sirven de metadatos para dar más información sobre los datos que se están manejando, como nombres, unidades, etc.
Este perfil es importante a la hora de comunicar dispositivos por BLE, y se usa como base para generar perfiles específicos para determinados casos de uso, como la medición del ritmo cardíaco de una persona o el acceso al nivel de batería disponible de un dispositivo. En la Sección 2.4 se detallarán los componentes de un perfil GATT.

Perfil de Acceso Genérico (Generic Access Profile, GAP). Define cómo los dispositivos realizan procedimientos de control como el descubrimiento de otros dispositivos, conexiones entre éstos o la seguridad, es decir, dicta cómo interactúan dos dispositivos en un nivel más bajo. GAP establece diferentes normas y conceptos para regular y estandarizar las operaciones a bajo nivel de los dispositivos:

- Roles en los que los dispositivos pueden operar, estableciendo restricciones e imponiendo determinados comportamientos. Se explican con más detalle en las secciones 2.3.1 y 2.3.2.
- Modos operacionales, que establecen estados a los que un dispositivo puede cambiar durante un determinado periodo de tiempo para lograr un objetivo, como por ejemplo permitir su descubrimiento o de qué modo permite realizar la conexión.
- Procedimientos operacionales para asegurar una comunicación constante mediante una secuencia de acciones.
- Aspectos de la seguridad, incluyendo modos y procedimientos.
- Formatos de datos adicionales para datos no protocolarios.

Gestor de Seguridad (Security Manager, SM). Define los protocolos y sus modos de operar para gestionar la integridad de los enlaces, la autenticación y el cifrado entre

dispositivos Bluetooth, y provee las funciones de seguridad que se pueden utilizar para asegurar casi cualquier nivel de seguridad que sea necesario para las diversas aplicaciones.

2.2.3. Aplicación

La aplicación es la capa más alta y la responsable de contener la lógica, la interfaz de usuario y el manejo de los datos de todo lo relacionado con el caso de uso particular que se esté implementando. Es por ello que su arquitectura dependerá en gran medida de la implementación particular.

2.3. Descubrimiento de dispositivos y enlazado

BLE tiene un único formato de paquete que soporta dos tipos: de datos y de anuncio; esto reduce la complejidad de la implementación de la pila de protocolos.

Los paquetes de anuncio tienen como propósito transmitir datos sin necesidad de una conexión o descubrir esclavos para iniciar una conexión. Tienen una capacidad útil de 31 bytes, que se utilizan para incluir información sobre el dispositivo y sus capacidades, pero también se pueden incluir los datos que se quieran transmitir. Estos paquetes se envían a intervalos definidos previamente, que pueden ir desde los 20 ms hasta los 10.24 s. Con un intervalo más corto se aumenta la frecuencia de envío de información, pero también significa que tiene un mayor consumo. En nuestro caso definimos un intervalo de 1 s.

Los paquetes de datos se utilizan una vez establecida una conexión para enviar información entre dos dispositivos. Tienen una capacidad de 27 bytes, menor que los de anuncio, debido a que los paquetes de datos ofrecen la posibilidad de encriptar el mensaje, para lo que se necesitan 4 bytes para la comprobación de la integridad del mismo. Para simplificar el diseño de la Capa de Enlace, los paquetes de datos que no utilicen encriptación también tienen ese *overhead* de 4 bytes en la construcción del paquete.

Como explicamos anteriormente, BLE utiliza como máximo tres canales de frecuencia para mandar los paquetes de anuncio, y debido a que el dispositivo que anuncia y el que

escanea no están sincronizados de ningún modo, los paquetes se recibirán cuando ambos se solapen aleatoriamente, como se ve en el ejemplo presentado en la Figura 2.3.

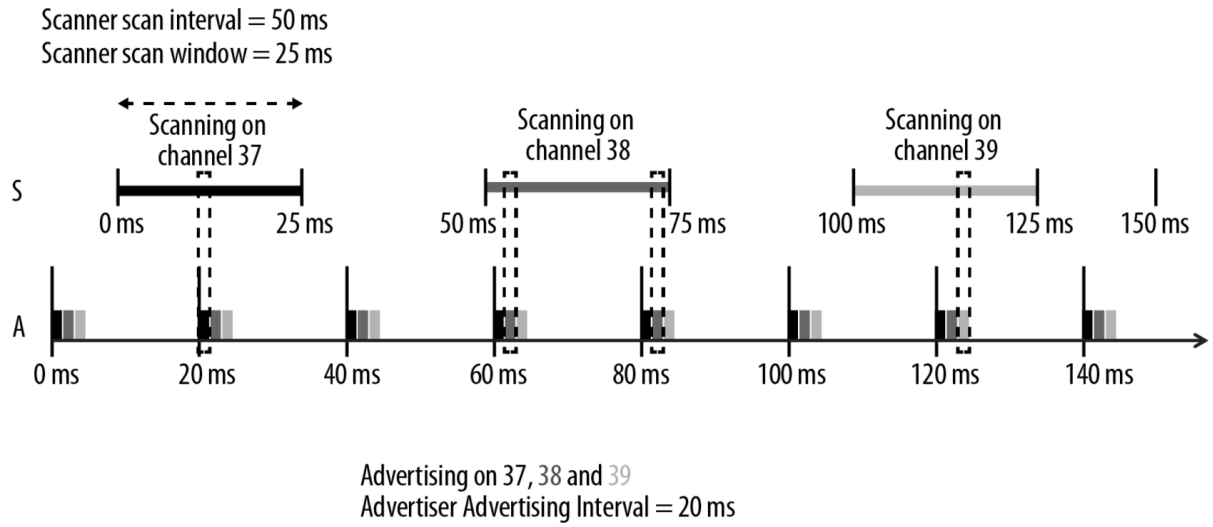

Figura 2.3: *Ejemplo de escaneo en BLE. El Scanner escanea cada 50 ms durante 25 ms en un canal determinado cada vez y el Advertiser anuncia cada 20 ms en los 3 canales. Cada vez que solapan se muestra con una línea de puntos.*

Un dispositivo BLE puede comunicarse con el mundo exterior mediante dos vías: **broadcasting** y **conexiones**.

2.3.1. Modo Broadcast

Mediante broadcasting es posible enviar datos en un único sentido a cualquier dispositivo que esté escaneando en el radio de escucha. Es la única forma de mandar datos a más de un dispositivo a la vez, aunque no permite transmitir una gran cantidad de información.

Los dispositivos pueden tomar los siguientes roles GAP:

- Broadcaster: envía periódicamente paquetes, sin importar si hay otros dispositivos escuchando.
- Observador: escanea las frecuencias establecidas para recibir estos paquetes.

Como se ha explicado antes, el método utilizado para mandar información mediante broadcasting es el de los paquetes de anuncio. BLE permite enviar un segundo paquete por

si se necesita mandar más información. Este método, llamado Scan Response, consiste en que el observador que encuentra la señal del broadcaster manda una petición para recibir el segundo paquete. Este segundo paquete de anuncio dispone de otros 31 bytes, haciendo un total de 62 bytes.

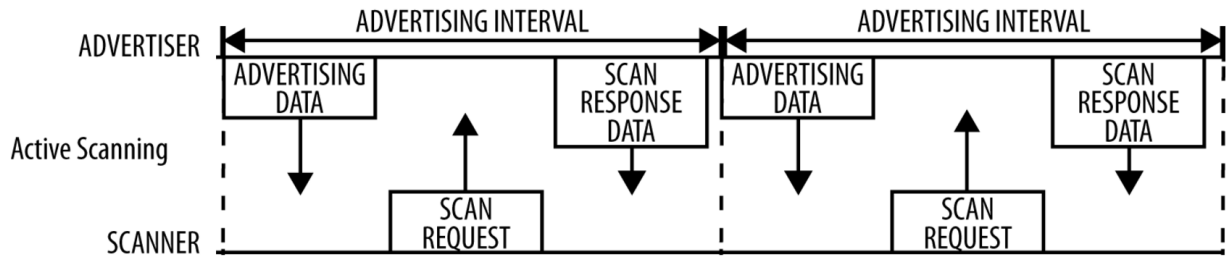

Figura 2.4: *Esquema de Scan Response. El dispositivo que escanea recibe un paquete de advertising y envía esta respuesta para requerir un segundo paquete con más información.*

Si se desea que la comunicación se realice en los dos sentidos, o una mayor privacidad (dado que, en modo broadcast, todos los dispositivos que escuchan reciben la misma información), o bien se necesita enviar más datos de los que es posible con el modo broadcast, tendremos que usar el modo conexión.

2.3.2. Modo Conexión

Una conexión permite un intercambio de paquetes de forma periódica y permanente entre dos dispositivos. En este modo tenemos dos roles GAP:

- Central (maestro): escanea las frecuencias predefinidas en busca paquetes de advertising y es el encargado de realizar las conexiones. Una vez conectado, es el encargado de gestionar la sincronización y de iniciar el envío periódico de datos. Dado que los requisitos computacionales del rol de maestro son mayores que los de los esclavos, este rol suele recaer en los dispositivos con una CPU y memoria más avanzados, como un smartphone o una tablet.

- Periférico (esclavo): se encarga de enviar los paquetes de advertising para que los centrales puedan encontrarlos y de aceptar las conexiones entrantes. El protocolo BLE

está optimizado para requerir pocos recursos, en términos de energía y memoria, para la implementación de este rol.

La principal ventaja de las conexiones es la posibilidad de organizar los datos que se envían mediante el uso de protocolos para cada campo, más específicamente el Generic Attibute Profile (GATT), que organiza los datos dentro de servicios y características.

2.4. Generic Attribute Profile (GATT)

El Perfil de Atributo Genérico expande el Protocolo de Atributo (ATT) estableciendo en detalle cómo intercambiar todos los datos del usuario a través de una conexión BLE. También provee un *framework* para todos los perfiles basados en GATT, que cubren casos de uso específicos y asegura el intercambio entre dispositivos de diferentes marcas. Todos los perfiles BLE estándar están basados en GATT y deben cumplir sus especificaciones para operar correctamente.

2.4.1. Roles

GATT define dos roles que pueden adoptar los dispositivos interconectados, que hereda directamente del Protocolo de Atributos (ATT):

- **Cliente.** Envía peticiones al servidor y recibe respuestas. El cliente GATT no tiene conocimiento sobre los atributos del servidor por adelantado, por lo que primero tiene que recopilar información sobre éstos mediante un descubrimiento de servicios.

- **Servidor.** Recibe las peticiones del cliente y responde en consecuencia. Es el rol responsable de guardar los datos y de hacer que estén disponibles para el cliente, organizándolos en atributos. Todos los dispositivos BLE que se vendan deben incluir al menos un servidor GATT básico, aunque únicamente envíe un error como respuesta.

2.4.2. Jerarquía de datos

ATT define el protocolo de atributos, cómo se estructuran y se pueden usar para intercambiar información sobre dispositivos, pero no ofrece más estructura. GATT va más allá y establece una jerarquía de datos estricta para organizar los atributos y permitir su reutilización, de modo que el acceso de información entre cliente y servidor siga una serie de reglas que se puedan usar en cualquier perfil basado en GATT.

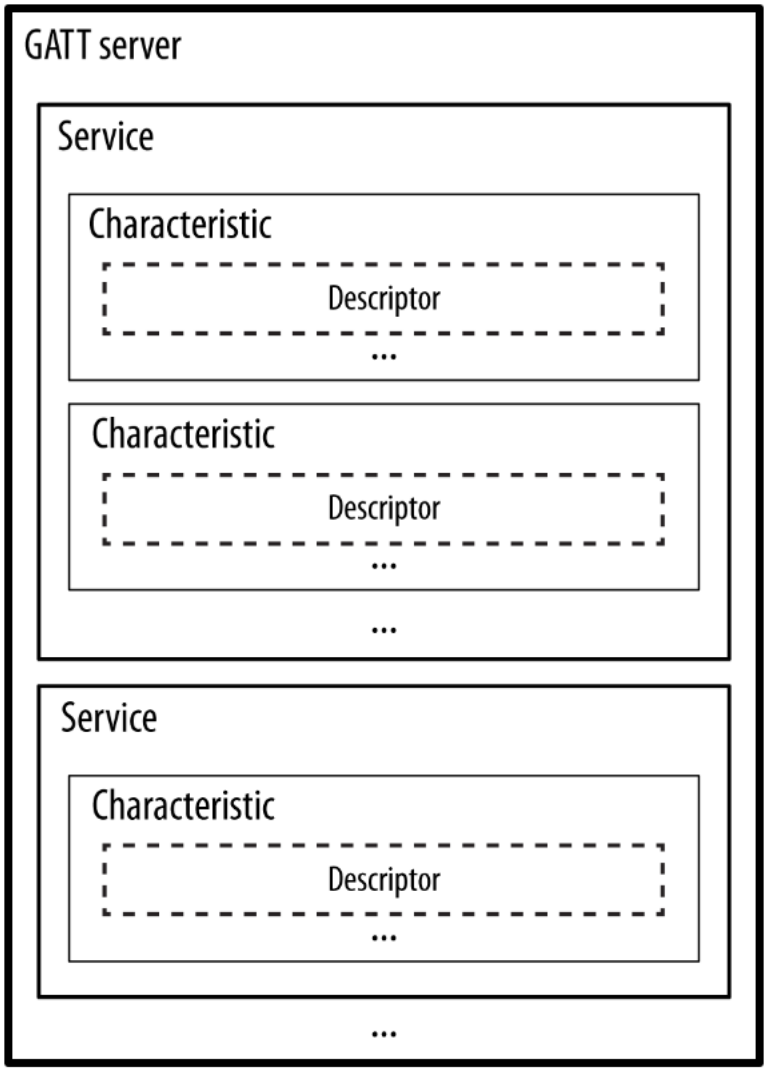

Figura 2.5: *Esquema de la estructura de datos de GATT*

Un perfil basado en GATT tendrá entonces, como se muestra en la Figura 2.5, como

mínimo un servicio, con una o más características que encapsulan a los atributos, y que pueden añadir más información a través de un descriptor.
De esta forma, si tenemos por ejemplo un dispositivo del que nos interesa sacar información sobre un sensor de temperatura y obtener el estado de la batería, tendríamos la siguiente estructura:

- Un **servicio** para la temperatura y otro para el nivel de batería.
- Dentro de cada servicio, una **característica** que encapsule ambos valores.
- De manera opcional, se puede agregar un **descriptor** dentro de alguna de las características.

Cada uno de estos niveles ofrece un modo de identificarlos y más información sobre ellos, por lo que un cliente que no conozca de qué servicios dispone un servidor, puede realizar lo que se denomina una *exploración de servicios* y así conocer qué información puede obtener. Todos los niveles se especifícan mediante atributos, ya que es la base sobre la que se asienta GATT, por ello es necesario definir primero en qué consiste un atributo.

Atributo. Los atributos son las entidades de datos más pequeñas definidas en GATT y ATT. Son piezas de información direccionables que pueden contener los datos del usuario o metadatos sobre la estructura y el agrupamiento de cada una de las capas.

Dado que GATT y ATT sólo pueden operar con atributos, los clientes y servidores que deseen realizar una interacción deberán organizarse de esta forma. Cada atributo contiene los datos del usuario e información sobre el propio atributo, contenidos en los siguientes campos:

- *Handle.* Es la parte que hace a un atributo direccionable. Consiste en un identificador de 16 bits para cada atributo en un servicio GATT particular, y se garantiza que no se modificará entre transacciones o conexiones. El valor 0x0000 denota un handle

inválido, por lo que el rango de handles disponible es de 0xFFFF (65535), aunque en la práctica el número de atributos en un servidor es más cercano a la docena.

- *Tipo.* Esta parte consiste en un UUID de 16, 32 o 128 bits que determina qué tipo de datos están presentes en el campo de valor del atributo, permitiendo por ejemplo mecanismos para encontrar un atributo directamente por su tipo. En la Sección 2.4.3 se muestra cómo se forman los UUIDs para su uso en BLE.

- *Permisos.* Los permisos son metadatos que especifican qué operaciones se pueden ejecutar en cada atributo y con qué requisitos de seguridad. ATT y GATT definen los siguientes permisos:

  - Permisos de acceso: similares a los permisos presentes en un archivo. Define si se puede acceder a un valor como sólo de lectura, de escritura, de ambos o de ninguno.
  - Encriptación: determina si es necesario algún nivel de encriptación para acceder al valor. Puede no necesitarla, o exigir una encriptación con o sin autenticación.
  - Autorización: determina si se necesita o no permiso del usuario para acceder al atributo.

- *Valor.* Contiene los propios datos sin restricción de qué tipo sean, aunque con un máximo de 512 bytes según la especificación. Dependiendo del tipo de atributo, puede contener información adicional sobre el atributo o un valor útil.

Servicio. Los servicios agrupan atributos conceptualmente relacionados. La forma de definir un servicio es a través de uno o más atributos denominados *definición de servicio*, de los cuales el primero es el que actúa de declaración de servicio y el resto pueden definir las siguientes capas, como características y descriptores.

Se puede comparar un servicio con una clase en cualquier lenguaje de programación orientado a objetos.

Característica. Las características se pueden entender como contenedores para los datos del usuario. Siempre incluyen dos atributos: la *declaración de la característica*, que ofrece metadatos sobre los datos en sí, y el *valor de la característica*, que es un atributo completo con los datos del usuario. Adicionalmente, una característica puede contener un descriptor, que es un atributo que provee de más metadatos.

Descriptor. Se utilizan principalmente para proveer al cliente con información adicional sobre las características y su valor. Se ubican siempre entre la definición de la característica y el valor del atributo de la característica y están compuestos por un único atributo cuyo UUID define el tipo de descriptor y el campo de valor contiene lo que esté definido para ese tipo de descriptor.

A continuación se detallan algunos de los descriptores más utilizados:

- *Descriptor de usuario de una característica:* contiene una descripción que el usuario puede leer consistente en una cadena de caracteres en UTF-8. Por ejemplo "Temperatura en el salón".

- *Descriptor de configuración de la característica por el cliente:* actúa como un "interruptor", deshabilitando las actualizaciones que envía el servidor.

- *Descriptor de formato de presentación de una característica:* contiene el tipo de variable de la característica, por ejemplo booleano, cadena de caracteres, enteros, etc.

2.4.3. UUIDs

Un *Identificador Único Universal* (UUID) es un número de 128 bits (16 bytes) que garantiza (al menos con una gran probabilidad) que es único globalmente. UUID es usado en otros protocolos aparte de Bluetooth, y su formato, uso y generación se especifican en el ISO/IEC 9834-8:2005.

Dado que 16 bytes ocuparía gran parte del tamaño disponible en los 27 bytes de los paquetes de datos, la especificación BLE añade dos formatos adicionales: UUIDs de 16 o 32

bits, y se forman definiendo un “final” del UUID que es constante en todos los dispositivos BLE, en los que se sustituyen los 32 primeros bits:

xxxxxxxx-0000-1000-8000-00805F9B34FB

donde xxxxxxxx son los bits a sustituir.

Capítulo 3

Exploración Hardware

El primer hito a nivel técnico era encontrar una placa de desarrollo que incluyera Bluetooth de bajo consumo y permitiera conectar un sensor de acelerometría, ya que eran indispensables para la transmisión de datos entre la placa y el dispositivo móvil.

Desde el punto de vista comercial, la intención ha sido buscar el componente con mejores prestaciones en calidad y precio con el objetivo de preparar un producto final que pudiera competir con otras opciones del mercado.

Destacamos las placas con Bluetooth incorporado y un microprocesador. En la tabla 3.1 observamos el análisis de cada elemento que se ha considerado.

Dada la naturaleza de nuestro proyecto no necesitábamos un gran poder de procesamiento ni una gran capacidad en cuanto a memoria RAM y flash, ya que el código iba a ser muy sencillo. Los dos factores principales a tener en cuenta eran que incorporase la tecnología Bluetooth Low Energy y que permitiese la conexión y el envío de datos con el acelerómetro, ya fuera mediante I2C, SPI, UART...

CHIPS CON BLUETOOTH					
EMPRESA	MODELO	PROCESADOR	FLASH	RAM	I/O
Nordic	PTR9022	ARM Cortex-M0	256 KB	16 KB	SPI, 2-WIRE, UART
Nordic	nRF51-DK	ARM Cortex-M0	256/128KB	32KB/16KB	SPI Master/Slave, 2-wire, UART, 31 GPIO
Nordic	nRF51422	ARM Cortex-M0	256/128KB	16KB	SPI Master/Slave, 2-wire, UART, 31 GPIO
TI	CC2540 / CC2541	8051	128/256KB	8KB	2 USART, ADC, 21 GPIO, SPI
TI	CC2640F128RGZT	ARM Cortex-M3	128KB	20KB	I2C, I2S, SPI, UART
Cypress	4 BLE/PRoC BLE	ARM Cortex-M0	128/256KB	16/32KB	2 SCBs, configurable como I2C, SPI o UART
Cypress	PSoC 4XX7-BLE	ARM Cortex-M0	128KB	16KB	I2C, SPI, UART, 36 GPIO

Cuadro 3.1: *Placas que integran radio Bluetooth Low Energy. Se destacan las alternativas seleccionadas para la siguiente fase de exploración*

Realizando el primer filtro pudimos observar que algunas características eran comunes entre los SoC's elegidos:

En materia de procesadores encontramos dos opciones comunes: **Cortex-M0** de ARM (Advanced RISC Machines) y **8051** de Intel. Al ver que la mayoría de los chip elegidos montaban el procesador de ARM, claramente observamos que domina el mercado de los sistemas empotrados y las empresas han optado por utilizarlo. Esto se debe a que el modelo de negocio utilizado por ARM consiste en la venta de licencias de sus núcleos[3], lo que permite a otros fabricantes diseñar su propio System On Chip (SoC) integrando tecnología ARM.

ARM Cortex-M0 nos parecía una opción más interesante, ya que este procesador tipo RISC, cuenta tanto con una arquitectura de 32 bits (frente a los 16 de Intel 8051) como con la arquitectura *Thumb*, que mejora la densidad del código para ocupar menos espacio tanto en memoria RAM como en flash.

La cantidad de memoria RAM disponible más común para este tipo de dispositivos es de 8, 16 o 32 kilobytes. La segunda opción nos pareció más que suficiente para albergar un programa sencillo como el nuestro y las variables necesarias.

En cuanto a capacidad de almacenamiento, todas las placas cuentan con memoria flash, que pueden variar entre 128 y 256 Kilobytes. De nuevo, la complejidad de nuestro programa no iba a generar un fichero compilado de gran tamaño, y no necesitábamos guardar nada más, por lo que 128 KB nos parecieron adecuados.

En cuanto a los periféricos de entrada/salida, la mayoría de las placas de desarrollo incluyen los protocolos SPI e I2C.

El protocolo SPI consiste en el envío de la señal de reloj del maestro y en cada impulso de reloj se envía un bit al esclavo y recibe un bit de éste. Los nombres de las señales son SCK para el reloj, MOSI para el Maestro Out Esclavo In, y MISO para Maestro In Esclavo Out.

El protocolo I2C, usa dos cables, uno para el reloj (SCL) y otro para el dato (SDA). El

maestro y esclavo envían datos por el mismo cable, el cual es controlado por el maestro, que crea la señal de reloj. Este protocolo utiliza direccionamiento, es decir, el primer byte enviado por el maestro se forma de 7 bits para la dirección (así que permite comunicarse con hasta 127 dispositivos) y un bit de lectura/escritura, indicando si el próximo byte vendrá desde el maestro o el esclavo. Esta tecnología se ampliará en el Capítulo 5.1.1.

3.1. Selección de plataformas de desarrollo

Una vez recopilados los modelos de que observamos en la Tabla 3.1, hemos destacado 2 placas de prototipado que cumplen con los requisitos del proyecto. Este tipo de placas ofrecen más características de las que necesitamos para el proyecto, pero suponen un primer paso para poder programar y realizar pruebas antes de pasar a chips más simples y con un menor coste.

Por un lado escogimos la placa de desarrollo de Cypress con el kit PSoC BLE y modelo de la placa con Bluetooth **CY8CKIT-042-BLE** certificado para sistemas de bajo consumo. Es un kit provisto de un chip que ofrece un procesador ARM Cortex-M0 y capacidad y conectividad suficiente como para utilizarlo de base para el proyecto.

El entorno de desarrollo para las plataformas de Cypress es un software de escritorio llamado *PSoC Creator*, en el cual podemos diseñar sistemas a través de un panel gráfico. Nos ofrece multitud de librerías disponibles para la placa utilizada y es posible codificar, compilar, y depurar código.

Consideramos también la placa de Nordic modelo **nRF51-DK** por ser un kit de desarrollo que ofrece el mismo procesador que Cypress, conectividad tanto I2C como SPI para realizar las pruebas con el sensor. Este modelo ofrece compatibilidad con la plataforma de desarrollo *mbed* de ARM, es una opción que nos resultó interesante a la hora escogerla. Dispone de una gran comunidad y soporte lo cual es de agradecer.

En la Sección 3.1.1 hablaremos más sobre los aspectos específicos de ambos entornos de desarrollo.

Al considerar que era más interesante tener una placa con procesador, sensores y periféricos descartamos los chips individuales y optamos por una opción más completa.

3.1.1. Análisis de las plataformas escogidas

Rápidamente observamos dos grandes empresas especializadas en el sector como son Nordic y Cypress. Tienen gran variedad de microprocesadores y placas de desarrollo que cumplen con nuestras expectativas.

Elegimos el kit de desarrollo de Nordic (**nRF51-DK**), que incluye Bluetooth Smart e incorpora un núcleo ARM Cortex-M0 32-bit como la mayoría de los chips que encontramos. Una memoria flash a 256/128KB con RAM de 32KB/16KB para mejorar el rendimiento de las aplicaciones.

El kit permite el acceso a todas las interfaces de entrada y salida como SPI Master/Slave, 2-wire, UART y 31 GPIOs a través de conectores. Tiene 4 LED's y ofrece también 4 botones que son programables por el usuario.

Utiliza un cable micro USB 2.0 para conectarse a uno de los puertos USB de el PC. Esto proporciona alimentación a la placa, y es compatible con la programación de destino.

La carga de programas resulta sencilla, ya que una vez conectado al ordenador, el sistema de archivos se monta como una unidad extraíble, por lo que solo debemos abrir un explorador de archivos y podremos cargar los programas a la placa arrastrando y soltando hacia la unidad.

La placa de desarrollo nRF51-DK es compatible con el entorno **ARM mbed**, que es una plataforma gratuita de prototipado rápido y experimentación con microcontroladores ARM. Provee a los desarrolladores una plataforma para realizar pruebas y prototipos en el lenguaje de programación C++. Incluye una amplia variedad de librerías, tutoriales y ejemplos, además de contar de una gran comunidad online de desarrolladores de software,

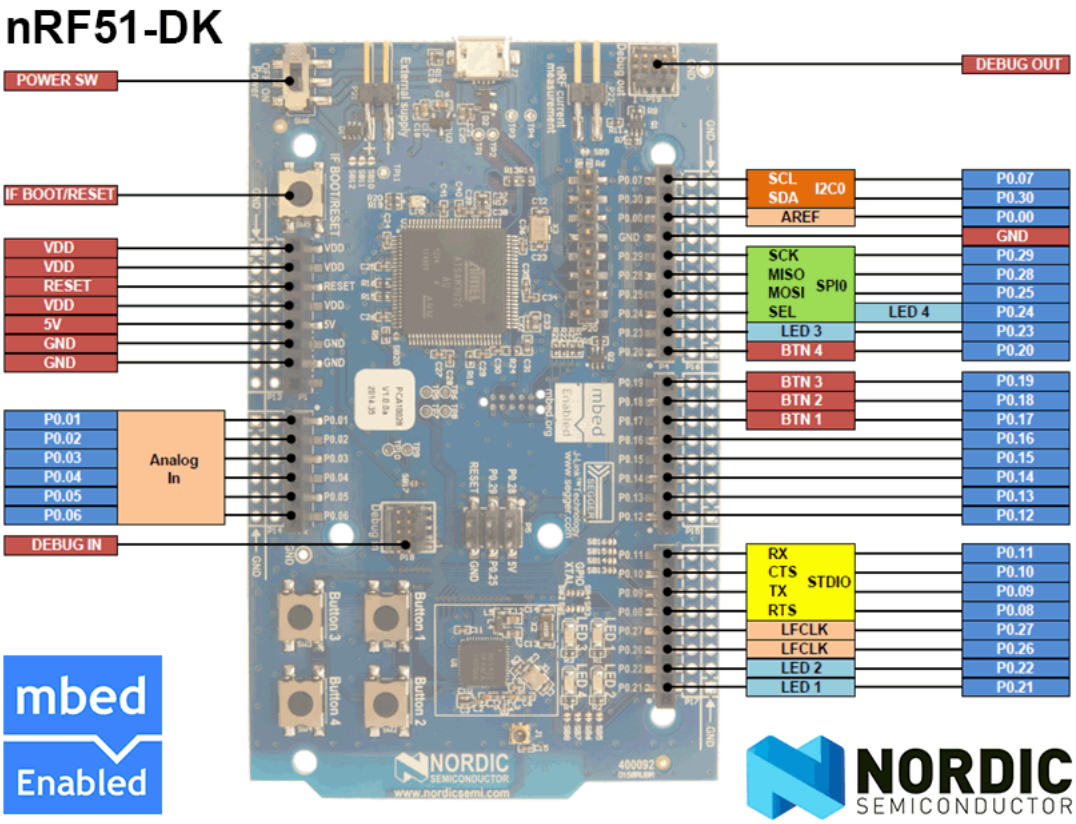

Figura 3.1: *Esquema de la disposición de pines de la placa nRF51-DK de Nordic*

cuyos códigos son habitualmente accesibles a toda la comunidad.

Otro chip con procesador que nos pareció interesante fue el modelo **CY8CKIT-042-BLE** de la empresa Cypress Semiconductor que soporta 2 dispositivos: PSoC 4 BLE y PRoC BLE.

El modelo escogido es el PSoC 4 BLE que provee de una completa solución para conectividad Bluetooth Low Energy. También monta una procesador ARM Cortex-M0 con una memoria flash de 128kB/ 256kB y RAM de 16kB / 32kB. Dispone de 4 TCPWM1, 2 SCBs2, LCD4, I2S5, y 36 GPIOs.

El kit incluye una memoria extraible que conecta la placa por Bluetooth llamado Dongle USB CySmart (BLE Dongle) que se empareja con la herramienta de emulación principal CySmart. El emparejamiento con un entorno Windows hace que sea un potente entorno de

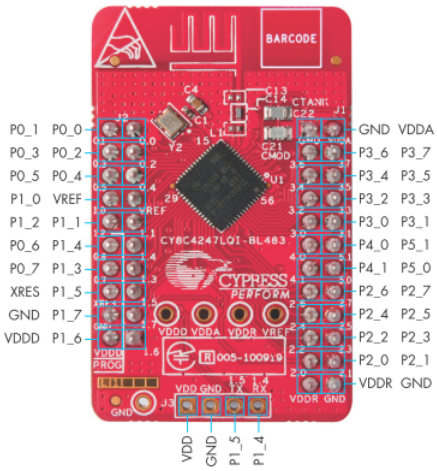

Figura 3.2: *Cypress PSoC. Esta placa contiene el microchip y todos los conectores necesarios como para usarla directamente como producto final*

depuración de Bluetooth LE.

Este kit de desarrollo de Cypress es compatible con diseños a nivel de sistema mediante PSoC Creator, un software de desarrollo que contiene numerosos proyectos de ejemplo para proporcionar diseños integrados de señal mixta Bluetooth de baja energía, el lenguaje utilizado es C.

Es sencilla diseñar pues con arrastrar y soltar los componentes se añaden al panel principal obteniendo máxima flexibilidad de diseño.

3.2. Conclusión

Tanto la placa nRF51-DK de Nordic como CY8CKIT-042-BLE de Cypress son dos placas de desarrollo perfectas para iniciar un proyecto con conectividad Bluetooth, las dos tienen idénticas características, conectores y periféricos, se les puede incluir una pila de botón CR2032 para darles autonomía y les respalda un software para desarrollo de las mismas.

En este punto tuvimos cierta controversia, ya que el software de Mbed de Nordic nos da la posibilidad de compilar código C++ en cualquier PC con internet gracias a su versión web. Nos ofrece una gran comunidad que es de agradecer y mucho contenido de ejemplos y

Figura 3.3: *PSoC 4 BLE, placa de desarrollo, en esta placa se conecta el PSoC para poder programarlo, y añade más funcionalidad incorporando elementos como botones programables, un LED RGB, un panel deslizante, etc.*

tutoriales en la web oficial. Por contra no tiene modo depuración por pasos y eso dificulta su seguimiento y depuración.

Por otro lado, PSoC Creator de Cypress nos ofrece un completo programa de escritorio muy visual al diseñar circuitos y un sistema "drag and drop" muy intuitivo. Permite modo depuración facilitando su programación. Por contra nos hemos encontrado con ciertas dudas que han sido difíciles de resolver en foros debido al poco contenido sobre el tema.

Debido al gran parecido entre las características de una placa y otra, y a que los entornos de desarrollo tienen cada uno sus pros y sus contras, nos resultó difícil elegir una de las dos para finalizar el proyecto. En el Capítulo 4 se hablará de las pruebas que realizamos con ambas y qué nos motivó a escoger la placa de Nordic frente a la de Cypress.

Capítulo 4

Instalación de entornos de desarrollo y primeros desarrollos HW

Antes de comenzar programando la funcionalidad principal de este proyecto, tuvimos que aprender a utilizar herramientas y conceptos como los entornos de desarrollo de las placas de Nordic y de Cypress o los protocolos SPI e I2C y el propio BLE, con los que no habíamos trabajado nunca, por lo que primero realizamos una serie de pruebas para familiarizarnos y así decidir cuál de ellas elegiríamos finalmente para el proyecto.

4.1. Pruebas iniciales con Cypress y Nordic

Una vez tuvimos las placas, el primer paso fue buscar códigos de ejemplo con funcionalidades parecidas a lo que íbamos a tratar en el proyecto.

Cypress pone a disposición de cualquier desarrollador que desee realizar pruebas un repositorio en GitHub con 100 proyectos que sirven de ejemplo para utilizar la funcionalidad BLE de sus dispositivos PSoC. Aparte han desarrollado una aplicación Android llamada *CySmart* para comprobar el funcionamiento de algunos de estos ejemplos.

mbed dispone de un repositorio propio donde cualquier usuario puede subir proyectos para cualquier dispositivo compatible con mbed. Estos proyectos son muy sencillos de buscar e importar desde el propio compilador. ARM mbed también dispone de una app Android, ésta llamada *nRF Master Control Panel*, que permite hacer algunas pruebas.

Como primera toma de contacto con las posibilidades de las placas para comunicarse físicamente con otros dispositivos realizamos un pequeño circuito consistente en una foto-resistencia conectada a un conversor analógico-digital MCP3008, capaz de convertir una entrada analógica de voltaje en un valor binario, el cual se transmitiría por el bus SPI.

Una vez creado el circuito comprobamos con un voltímetro que dejar pasar menos luz sobre la fotoresistencia disminuía la cantidad de voltaje. El rango de voltaje que pudimos comprobar fue de 0,82 V a 1,82 V con luz ambiente.

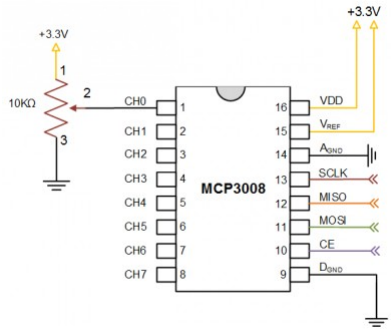

Figura 4.1: *Diagrama de pines para el conversor A/D MCP3008*

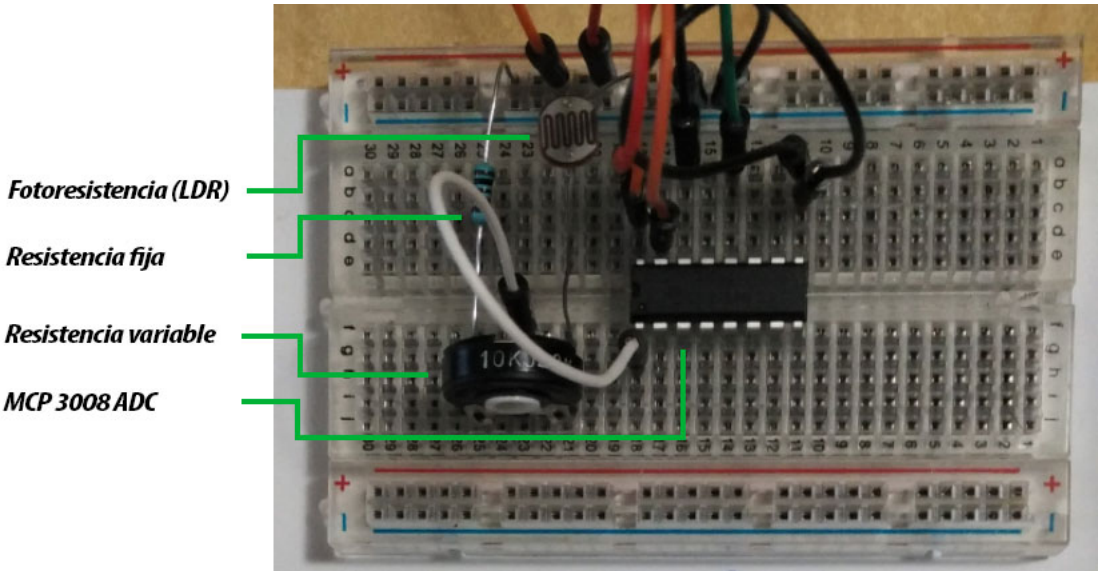

Figura 4.2: *Circuito realizado para la prueba MCP3008, con una fotoresistencia.*

Para realizar la transferencia de datos por SPI colocamos como master la placa de desarrollo y como esclavo el circuito de la fotoresistencia. Conectamos los 4 pines correspondientes en cada placa: para la trasmisión de datos (MOSI, MISO), frecuencia de reloj (SCLK) y selección del esclavo (nSS). Ambos entornos de desarrollo ofrecen ejemplos de módulo SPI con lo que fue fácil hacer funcionar el sistema una vez conectados los pines correctos.

Para comprobar en un principio si se recibían correctamente los datos, en la placa nRF51-DK de Nordic utilizamos sus 4 LED'S verdes, apagándolos o encenciéndolos según el valor ya digitalizado que recibe. De igual forma al probarlo con PSoC BLE de Cypress interactuamos con su LED RGB, utilizando los colores verde, azul y rojo para representar los diferentes datos que recibía del circuito.

Estas primeras pruebas nos sirvieron para comprobar la correcta recepción de datos, pero el objetivo era llevarlos a una aplicación desarrollada en Android. Por tanto nuestra primera toma de contacto con el desarrollo de la aplicación Android (Figura 4.3) que haría uso del módulo BLE fue en este momento, centrándonos más en la funcionalidad que en el diseño. Esta aplicación funcional nos permitió probar conceptos como el de activar desde dentro de una app el Bluetooth del móvil, realizar un escaneo para detectar los dispositivos con Bluetooth cercanos, conectarse a éstos y enviar datos. Para verificar que funcionaba correctamente, escribimos mensajes de log que se pueden visualizar en el entorno de desarrollo Android Studio.

El siguiente paso hacia nuestro aprendizaje fue conectar por I2C el Acelerómetro XTRINSIC-SENSE-BOARD Element14. Documentándonos desde su datasheet[6], conectamos los pines correctos con la placa, e iniciamos las pruebas de recepción de datos. Nuevamente, la galería de proyectos que oferece el entorno mbed nos permitió desarrollar el código de comunicación I2C sin mucha dificultad.

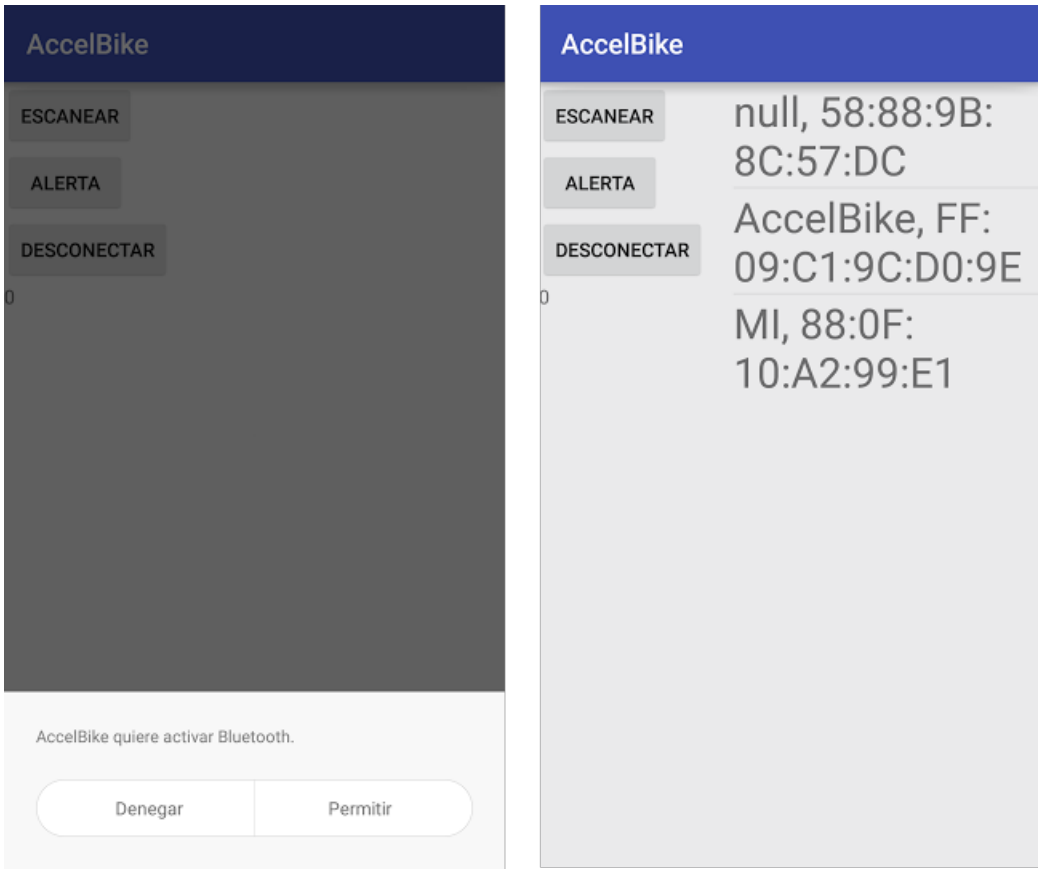

Figura 4.3: *Aplicación piloto para comprobar la conexión Bluetooth*

4.2. Acerca de los entornos de desarrollo

4.2.1. ARM mbed

La interfaz del compilador mbed es *online*, permitiendo, tras registrarse, acceder al entorno desde cualquier navegador en cualquier dispositivo con una conexión a Internet sin importar el sistema operativo del ordenador en el que se programe. El lenguaje de programación utilizado es C++, y una vez elegido a qué dispositivo va orientado el código, mbed se encarga de toda la configuración a bajo nivel, por lo que, por ejemplo, sacar una señal a través de un pin se puede realizar tan fácimente como declarar una variable *DigitalOut* con su correspondiente número de pin.

Una vez tenemos un código listo, mbed permite compilarlo y descargar el archivo hexa-

decimal a nuestro sistema si no ha habido errores de compilación. Para cargarlo en la placa de Nordic, basta con conectar ésta al ordenador por USB, lo que la abrirá en nuestro sistema como una *unidad flash* en la que podremos copiar el archivo compilado. Si ha ocurrido algún problema en la carga, o el programa cargado ha fallado por un error de ejecución, se creará en esa unidad extraíble un archivo de *log* con información sobre el error.

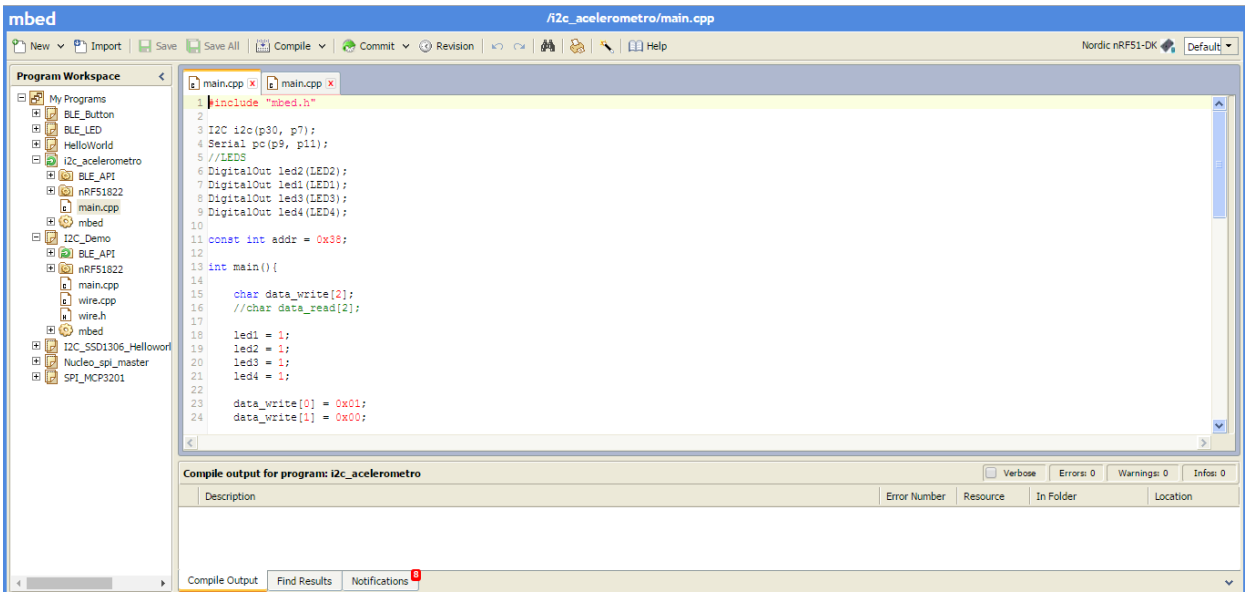

Figura 4.4: *Entorno de desarrollo mbed*

La plataforma mbed está muy orientada a la colaboración entre desarrolladores, e implementa un repositorio propio donde cualquier usuario puede subir su código y otros usuarios pueden importarlos de forma sencilla a sus proyectos. Esto se hace evidente teniendo en cuenta que en la propia interfaz se encuentra un botón *Import* que abre un buscador para encontrar códigos o librerías a través de palabras clave, lo cual resultó de gran ayuda en el período de pruebas y en la fase final, como se comentaba anteriormente.

Como puntos negativos se puede resaltar que mbed no cuenta con ninguna herramienta de *debug*, por lo que nos vimos obligados a comprobar el correcto funcionamiento de nuestros programas a través de LEDs o de *printf*s, que nos permitían enviar cadenas de texto a través del puerto USB del dispositivo. También cabe destacar que, aunque el hecho de ser online

supone una ventaja a la hora de trabajar desde varios sistemas, resulta un problema si no se dispone de conexión a Internet en un determinado momento.

4.2.2. Cypress PSoC Creator

En el caso del kit PSoC 4 BLE de Cypress, la plataforma de desarrollo utilizada es propia de la marca Cypress. Esto quiere decir que está orientada exclusivamente para la programación de las placas de esta empresa.

Este entorno de desarrollo consiste en una aplicación de escritorio específica para el sistema operativo Windows, la cual ofrece además una serie de programas de configuración para actualizar el *firmware* de la placa de desarrollo y comprobar el estado del enlace con el ordenador. El lenguaje de programación que se utiliza es C, y utiliza un sistema de *drag and drop* para especificar las funcionalidades que se van a utilizar y definir su interconexión y su configuración.

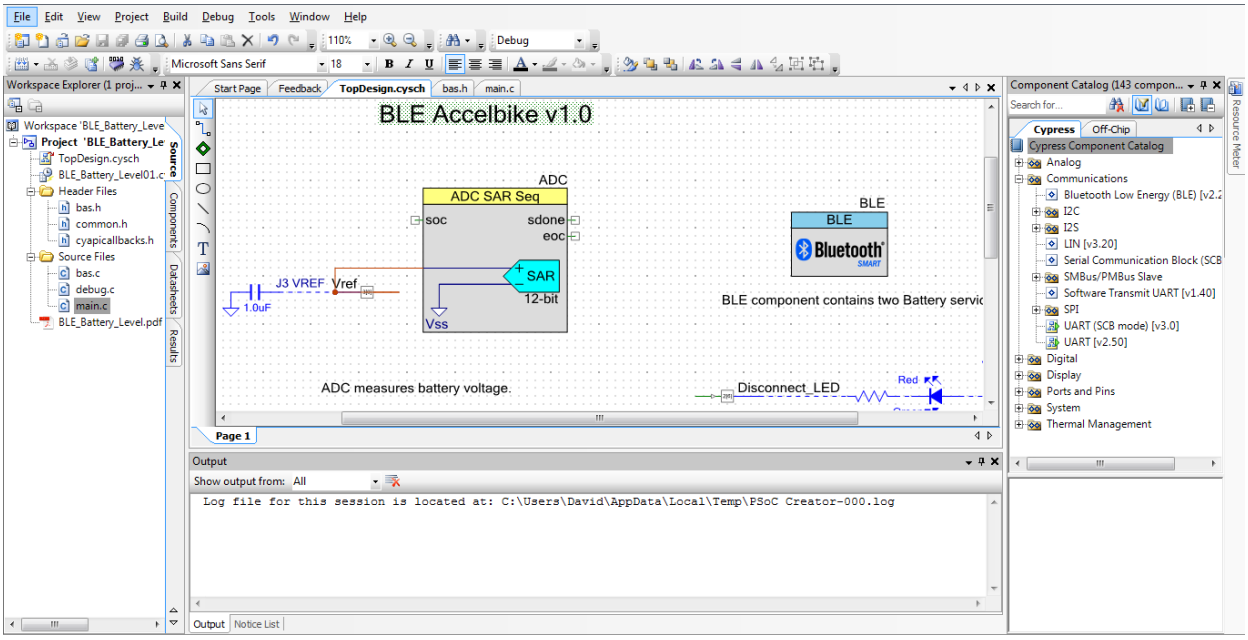

Figura 4.5: *Entorno de desarrollo PSoC Creator*

Este sistema permite, por ejemplo, buscar la componente de Bluetooth Low Energy,

arrastrarla hasta la vista de diseño, y elegir la opción *Generar Aplicación* para generar el código necesario para interactuar con BLE, creando un archivo de encabezado *BLE.h* y uno de código *BLE.c*.

Cypress también pone a disposición del desarrollador tutoriales en vídeo y una serie de ejemplos de código denominada *100 projects in 100 days*[5], subida a un repositorio en GitHub, de las que pudimos tomar ejemplo para avanzar con las pruebas. Sin embargo, la comunidad de desarrolladores era más pequeña que la de *mbed*, probablemente por ser esta plataforma específica para productos de Cypress, mientras que en el ecosistema de *mbed* se encuentran multitud de fabricantes.

A diferencia de *mbed*, PSoC Creator sí ofrece herramientas de *debug*, con puntos de ruptura y visualización del estado de las variables.

4.3. Conclusión tras las pruebas

Tras realizar las pruebas definidas en la Sección 4.1, aprendimos a desarrollar rápidamente códigos para lograr la funcionalidad deseada.

Los ejemplos de uso de Bluetooth Low Energy nos ayudaron a comprender los pasos que se debían dar para establecer correctamente una conexión, aunque tuvimos que informarnos más para comprender qué era exactamente lo que sucedía en cada paso.

Una vez entendimos el procedimiento para conectar por SPI el circuito mencionado anteriormente y la lógica que había que seguir para programar la interacción tanto con la placa de Cypress como la de Nordic, realizar la transición a I2C para conectar el acelerómetro fue un paso sencillo, pues las API de ambas placas gestiona de manera similar los dos protocolos.

Como ya mencionamos en el Capítulo 3, la decisión de continuar la fase final del proyecto con una placa u otra fue difícil, pues ambas tienen características similares y sus entornos de desarrollo tienen sus pros y sus contras, pero finalmente escogimos la placa de desarrollo

nRF51-DK de Nordic por la facilidad de encontrar información y ayuda en los foros de la comunidad. Los ejemplos ofrecidos en el repositorio propio de *mbed* fueron de gran ayuda para consolidar el aprendizaje con dicha placa y la propia herramienta nos pareció una gran apuesta para dispositivos IoT. Cabe destacar que la opción de Cypress podría haber sido viable perfectamente.

Capítulo 5

Desarrollo Hardware

Una vez terminado el período de pruebas, ya con el kit de desarrollo nRF51-DK de Nordic como opción principal y con la placa XTRINSIC-SENSE-BOARD de Element 14, que incorpora el acelerómetro MMA84910 de 3 ejes, pudimos comenzar a desarrollar la parte hardware de este proyecto.

El desarrollo se centró en dos elementos: la **comunicación por I2C** entre la placa de desarrollo y el acelerómetro y el **envío de datos mediante Bluetooth Low Energy** desde la placa hacia un dispositivo móvil. A continuación se describirán los pasos realizados.

5.1. Comunicación con el acelerómetro por I2C

El modelo nRF51-DK utiliza GPIO (General Purpose Input/Output), que son unos pines genéricos que pueden programarse como entrada o salida. Dos de estos pines se pueden configurar como un bus I2C, que se usa para comunicarse con una gran variedad de dispositivos externos, en este caso la placa XTRINSIC-SENSE-BOARD.

I2C es un bus de datos con conexión serie síncrona unidireccional (half-duplex) que se puede dar de 2 tipos:

- Maestro/Esclavo
- Esclavo/Maestro

Maestro. Inicia la transferencia generando las condiciones de inicio y parada de la señal de reloj. Transmite la dirección del esclavo y determina el sentido de la transferencia (lectura-escritura). La comunicación siempre la inicia el Master y el esclavo espera órdenes.

Esclavo. Este responde sólo cuando se dirigen a él. La temporización se controla mediante la línea de reloj del bus.

La transmisión de información se hace a través de 2 cables, uno para transmitir los datos (SDA) y otro para transmitir la señad del reloj (SCL). Otro cable para activar el acelerómetro (ENABLE) y otros 2 cables de toma de tierra (GND) y corriente (VDD) con este cablado tenemos conectado el sensor con la placa de desarrollo.

5.1.1. Lectura de datos

El acelerómetro MMA8491Q tiene 7 registros donde se guardan los datos de acelerometría que han sido recogidos en la lectura. El primer registro es un registro de estado en el que se indica si alguno de los ejes ha cambiado desde la última muestra, los 6 registros siguientes dividen cada eje en los 8 bits más significativos y los 6 menos significativos, haciendo un total de 14 bits por eje. En la tabla que se muestra en la Figura 5.2 se puede observar la información de los 7 registros.

Estos 14 bits se utilizan para especificar los valores de cada eje en un rango de +8000 mg a -8000 mg. Para ello utilizan el formato de la tabla que se puede ver en la Figura 5.3 sacada del datasheet[6]. Se puede observar que se utiliza un Complemento a 2 para definir los números negativos. La forma de parsearlos se explicará más adelante.

El acelerómetro establece una serie de pasos para poder establecer la comunicación por I2C en el modo *Multiple Byte Read*, que apunta al siguiente registro una vez concluida la lectura del anterior:

- Se manda el comando Start condition con la dirección 0x55 y 1 bit de lectura-escritura

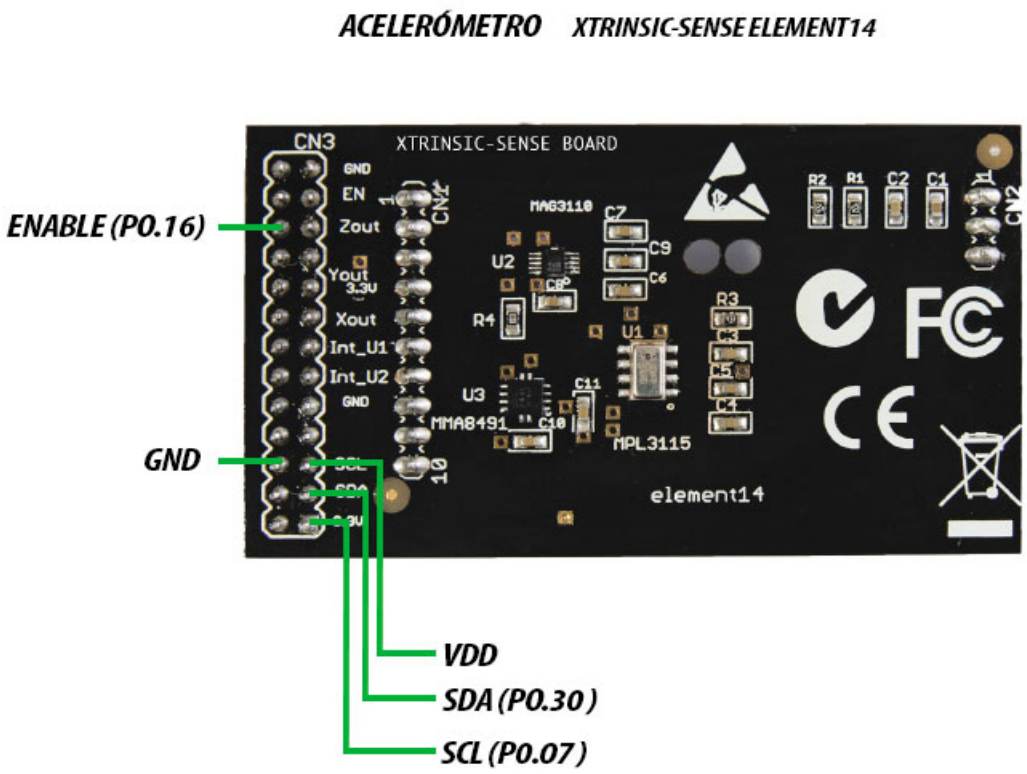

Figura 5.1: *Cableado del acelerómetro por I2C*

a 0 para indicar escritura, el esclavo responde con una trama ACK.

- Se transmite la dirección del registro que se quiere leer y el esclavo manda una trama ACK.
- Se transmite el comando Repeated Start Condition con la dirección del acelerómetro, y esta vez el bit de lectura-escritura a 1 indicando lectura.
- El esclavo manda una trama ACK y transmite los datos del registro indicado.
- Por último, se transmite una trama ACK y una señal de stop para finalizar.

Una vez recibidos estos datos en la placa de Nordic, se juntan y se parsean siguiendo la siguiente fórmula:

Name	Type	Register Address	Auto-Increment Address[3]	Default	Comment
STATUS	R	0x00	0x01	0x00	Read time status
OUT_X_MSB	R	0x01	0x02	Output	[7:0] are 8 MSBs of the 14-bit sample
OUT_X_LSB	R	0x02	0x03	Output	[7:2] are the 6 LSB of 14-bit sample
OUT_Y_MSB	R	0x03	0x04	Output	[7:0] are 8 MSBs of the 14-bit sample
OUT_Y_LSB	R	0x04	0x05	Output	[7:2] are the 6 LSB of 14-bit sample
OUT_Z_MSB	R	0x05	0x06	Output	[7:0] are 8 MSBs of the 14-bit sample
OUT_Z_LSB	R	0x06	**0x00**	Output	[7:2] are the 6 LSB of 14-bit sample

Figura 5.2: *Tabla con los registros del acelerómetro*[6]

14-bit Data	Range ±8g (1 mg/count)
01 1111 1111 1111	+8.000g
01 1111 1111 1110	+7.998g
...	...
00 0000 0000 0000	0.000g
11 1111 1111 1111	-0.001g
...	...
10 0000 0000 0001	-7.998g
10 0000 0000 0000	-8.000g

Figura 5.3: *Rango de datos del acelerómetro*[6]

```
WORD = (MSB << 6) | (LSB >> 2);

if (WORD >= 0x2000)
        WORD -= 0x4000;

NATURAL = (1000 * WORD + 512) >> 10;
```

Donde MSB es el byte con los bits más signigicativos, LSB es el byte con los bits menos significativos, y WORD y NATURAL son variables enteras. El resultado, de 14 bits, se guarda en una variable de 16, que luego se dividirá en 2 bytes separados para poder enviarlos

mediante Bluetooth.

Paso por paso, lo que hacemos es:

- Concatenamos los dos grupos de bits leidos por separado del bus.
- Cambiamos el signo si es negativo, almacenandolo como un numero negativo de 16bits.
- Cambiamos la escala de -8192 / 8191 a -8000 / 8000

5.1.2. Filtrado

En un entorno ideal, los datos que recoja el acelerómetro en un período en el que se encuentre estático deberían ser únicamente el valor de la gravedad (1g) repartido entre los 3 ejes, ya que es una aceleración constante que experimenta cualquier objeto en este planeta.

Lejos de ese caso, la información que recibimos varía ligeramente, normalmente en una escala de unos pocos mg. Para eliminar este ruido realizamos un filtrado consistente en recoger una muestra de 10 valores e insertarlos en un Búfer circular para poder filtrarlos a través de una media ponderada. El dar pesos a los datos permite que los últimos valores no se pierdan y que la media se actualice lo suficientemente rápido en caso de cambios bruscos.

5.2. Comunicación Bluetooth

Como mencionábamos en el Capítulo 2, hemos elegido el Modo de Conexión para realizar el enlace y el intercambio de datos entre la placa de Nordic y el dispositivo móvil. En este caso tendremos los roles GAP y GATT (Explicados en las Secciones 2.3.2 y 2.4.1, respectivamente) que se muestran en la Tabla 5.1.

El dispositivo móvil actúa como Central, ya que realiza el escaneo y la conexión, y como Cliente porque es el que envía las peticiones para recibir la información.

La placa nRF51-DK actúa como Periférico, pues manda paquetes de anuncio y acepta las conexiones entrantes, y como Servidor, al ser el que guarda la información y responde a las peticiones.

	GAP	GATT
Dispositivo Móvil	Central	Cliente
Placa nRF51-DK	Periférico	Servidor

Cuadro 5.1: *Roles GAP y GATT del dispositivo móvil y de la placa nRF51-DK*

Empezamos configurando el Perfil de Acceso Genérico (GAP) para establecer la información que incluyen los paquetes de anuncio:

- El dispositivo es sólo BLE y no utiliza Bluetooth clásico.
- Se le permite a otros dispositivos su descubrimiento.
- La lista de servicios que contiene, en la que aparecerá el servicio mencionado anteriormente.
- El nombre del dispositivo, dado por una cadena de caracteres.
- El método de conexión.

Toda esta información estará disponible para cualquier dispositivo que se encuentre escaneando.

Seguidamente configuramos el Perfil de Atributo Genérico (GATT). Definimos el dispositivo como servidor GATT y, para encapsular los datos del acelerómetro, creamos un servicio con UUID 0xA000, al que le asignaremos una característica con UUID 0xA001. Esto se hace mediante las clases *GattService* y *GattCharacteristic* que nos ofrece la API de mbed. Estas clases permiten establecer los siguientes parámetros:

Característica. Le podemos definir un tamaño máximo y mínimo de los datos que van a contener, en nuestro caso siempre va a tener 6 bytes (3 ejes divididos en 2 bytes cada uno). También especificamos aquí que este valor es de sólo lectura.

Servicio. Le asignamos una lista en la que únicamente aparecerá nuestra característica.

Como se ha explicado en la Sección 5.1.1, los datos extraídos del acelerómetro se guardan en la nRF51-DK en tres variables de 16 bits, aunque la API BLE de mbed obliga a actualizar el valor de una característica utilizando un array de variables de 8 bits, por lo que necesitamos dividir los 16 bits de cada eje en dos bytes separados. Esto hace que el dispositivo móvil reciba la información también de este modo, por lo que se tienen que juntar de nuevo al recibirlos.

Capítulo 6

Aplicación Android

La parte del servidor Bluetooth se ha implementado sobre la plataforma Android. La principal razón por la que optamos por esta plataforma es la de que para programar una aplicación en iOS es necesario disponer de un ordenador Macintosh, condición que no cumplíamos, y debido a que de los tres componentes del grupo, dos teníamos disponible un dispositivo Android, nos pareció un paso lógico.

Hay numerosos frameworks disponibles para programar aplicaciones Android, pero nos decantamos por la herramienta que ofrece Google, Android Studio, que al estar basado en la plataforma Eclipse nos resultó familiar, a la vez que proveía muchas funcionalidades específicas para Android.

Una aplicación Android se implementa en Java para especificar la funcionalidad y en XML para estructurar la parte gráfica. Aunque tuvimos que familiarizarnos con el modo de gestionar las vistas en Android, el hecho de basar la funcionalidad en Java nos ha facilitado desarrollar rápidamente aplicaciones sencillas para probar funcionalidades y, eventualmente, la aplicación final.

Hemos basado el diseño en el patrón Modelo-Vista-Controlador, comunicando la vista y la funcionalidad mediante comandos. En la siguiente imagen se muestra un diagrama de clases para ver cómo hemos estructurado el código de la app.

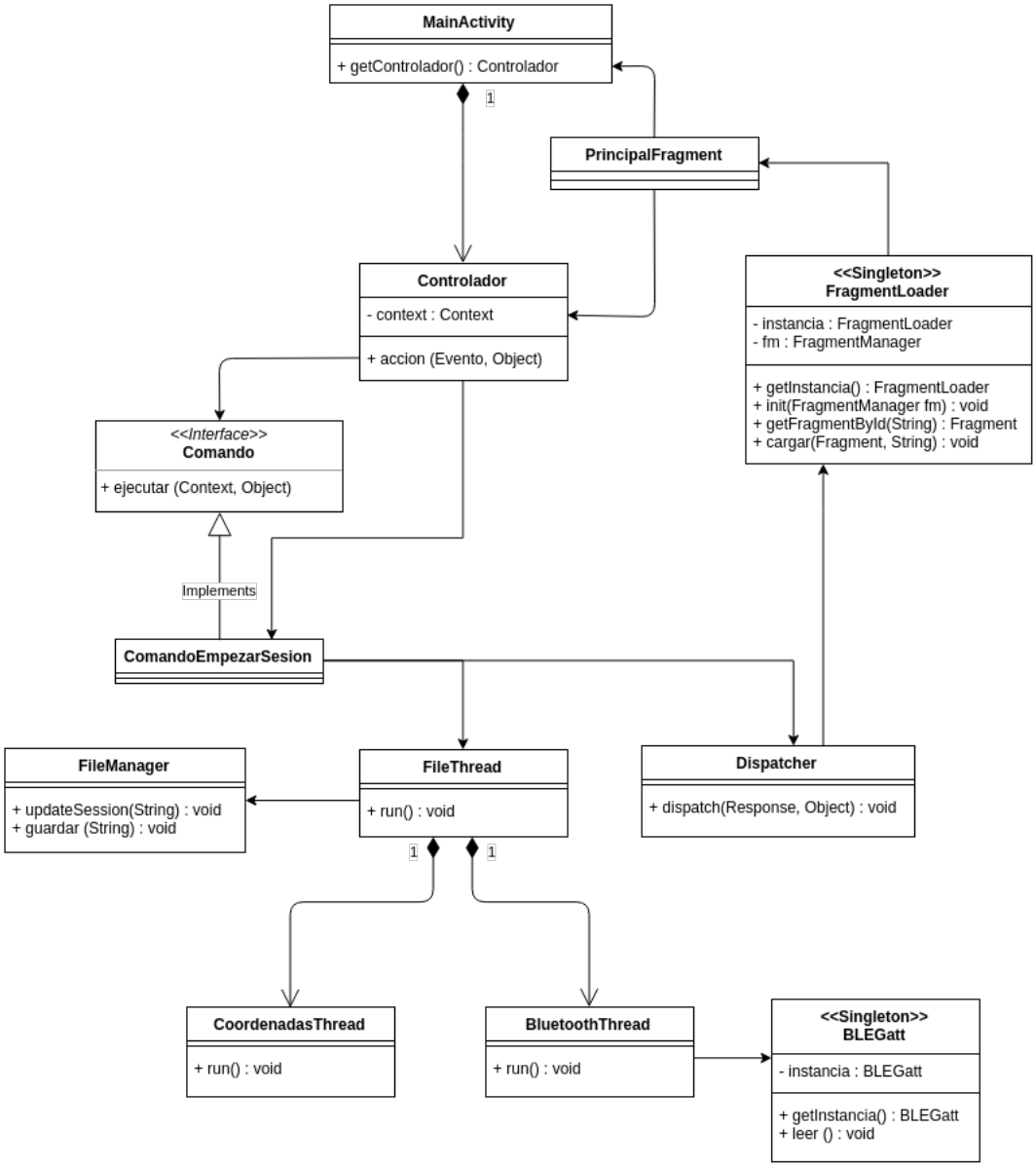

Figura 6.1: *Diagrama de clases general de la aplicación Android*

6.1. Front-End

Para la interfaz de usuario, hemos implementado una actividad principal, que se mantiene estática durante toda la vida de la aplicación, con fragmentos que se van cargando dentro de ésta. La actividad principal mantiene una barra de título en la parte superior que se encuentra siempre presente en la que hemos añadido un menú Hamburger, que permite

desplegar una lista con accesos directos a las opciones de la app, bien pulsando sobre el icono o deslizando el dedo de izquierda a derecha. Este diseño es muy utilizado en apps Android.

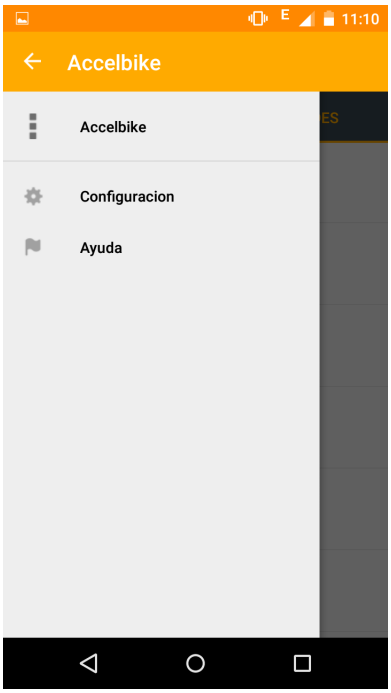

Figura 6.2: *Menú deslizable que aparece al pulsar en el botón de menú o al deslizar de izquierda a derecha*

A continuación explicaremos en detalle los fragmentos que se cargan en la app.

6.1.1. Vista principal

La primera pantalla que aparece al iniciar la aplicación. Está dividida en dos pestañas, sesión y actividades, lo que facilita acceder a las funcionalidades principales de la app. La pestaña que aparece por defecto es la de sesión, que es la que lleva a cabo el objetivo principal de este proyecto, ya que permite iniciar la recogida de datos del acelerómetro y del GPS. La forma de llevar a cabo esta funcionalidad se explicará más adelante.

En esta pestaña de sesión tenemos dos botones, inicio y parar, un cronómetro y un espacio en el que se nos va a mostrar la información de acelerometría recibida.

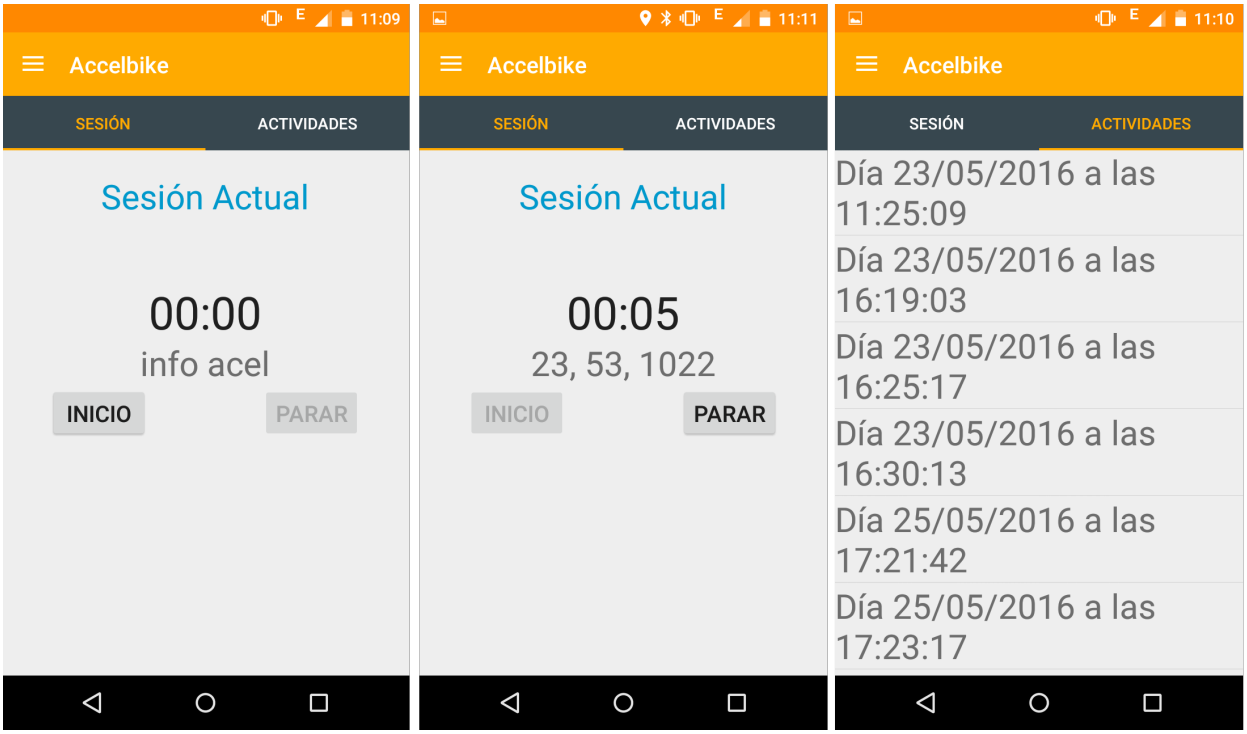

Figura 6.3: *Vista principal de la app. De izquierda a derecha: Sesión sin iniciar, Sesión iniciada y Vista de sesiones*

La segunda pestaña a la que se puede acceder es la de actividades, que muestra una lista de todas las sesiones guardadas. El nombre nos muestra el día y hora a la que se realizó. Pulsando en cada una de estas actividades lanza un fragmento con los datos de la ruta.

6.1.2. Configuración y Ayuda

En la pantalla de configuración (Figura 6.4) se permite al usuario realizar la conexión con la placa. Para ello contiene un botón para escanear los dispositivos BLE cercanos, que aparecerán en una lista en la zona inferior, y al pulsar en uno de los elementos de esta lista nos conectaremos y así podremos comenzar la sesión.

6.1.3. Vista resumen de sesión

Esta vista se lanza cuando terminamos una sesión (botón parar), o cargamos una actividad desde la pestaña de actividades. En ella se muestra un mapa con la ruta realizada, la

Figura 6.4: *Vistas de Configuración y Ayuda*

cual se colorea según la aceleración producida en cada momento. También nos muestra el tiempo, la distancia recorrida y la velocidad media del trayecto. En la Figura 6.5 se muestra un ejemplo de la vista con una ruta realizada alrededor de la Facultad de Informática.

6.2. Funcionamiento

A continuación describiremos la funcionalidad de los casos de uso más importantes de la aplicación, detallando la estructura de clases utilizada.

6.2.1. Conexión al dispositivo BLE

Para realizar la comunicación entre la placa y el móvil, primero debemos escanear los dispositivos BLE cercanos. Para ello utilizamos una clase de la API de Android llamada *BluetoothLeScanner*, que busca exclusivamente dispositivos Bluetooth Low Energy y nos devuelve la información recogida en los paquetes de anuncio sobre los dispositivos en forma de *BluetoothDevice*.

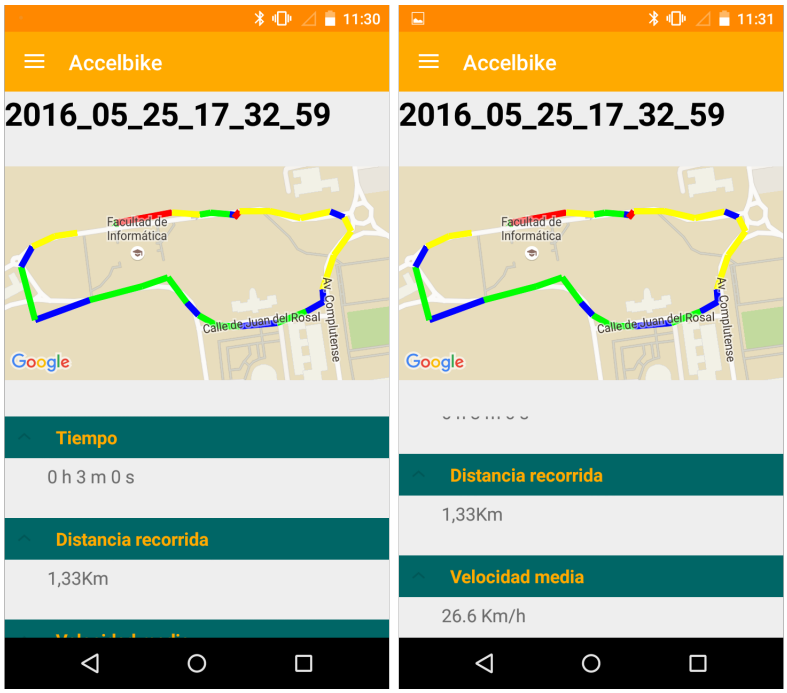

Figura 6.5: *Vista de Resúmen de Sesión. Se muestra la ruta y la información recogida.*

Una vez obtenidos, los mostramos en forma de lista en la que aparece el nombre de cada dispositivo, como se muestra en la Figura 6.4, para que el usuario elija a cuál se quiere conectar.

Hemos creado una clase *BLEGatt*, que es la que se encarga de mantener la información del enlace. Para establecer la conexión, se hace uso de esta clase, que configura el GATT para empezar a explorar los servicios, como se explicará más adelante.

6.2.2. GPS

Android por motivos de seguridad no permite activar el GPS del móvil desde una app, por lo que lo único que hacemos es avisar al sistema operativo para que abra la ventana de configuración de la ubicación. Una vez activado, la API de Android nos ofrece una clase llamada *LocationManager*, que nos permite establecer un período de actualización, de modo que guarda de manera regular la última localización conocida, a la que podemos acceder.

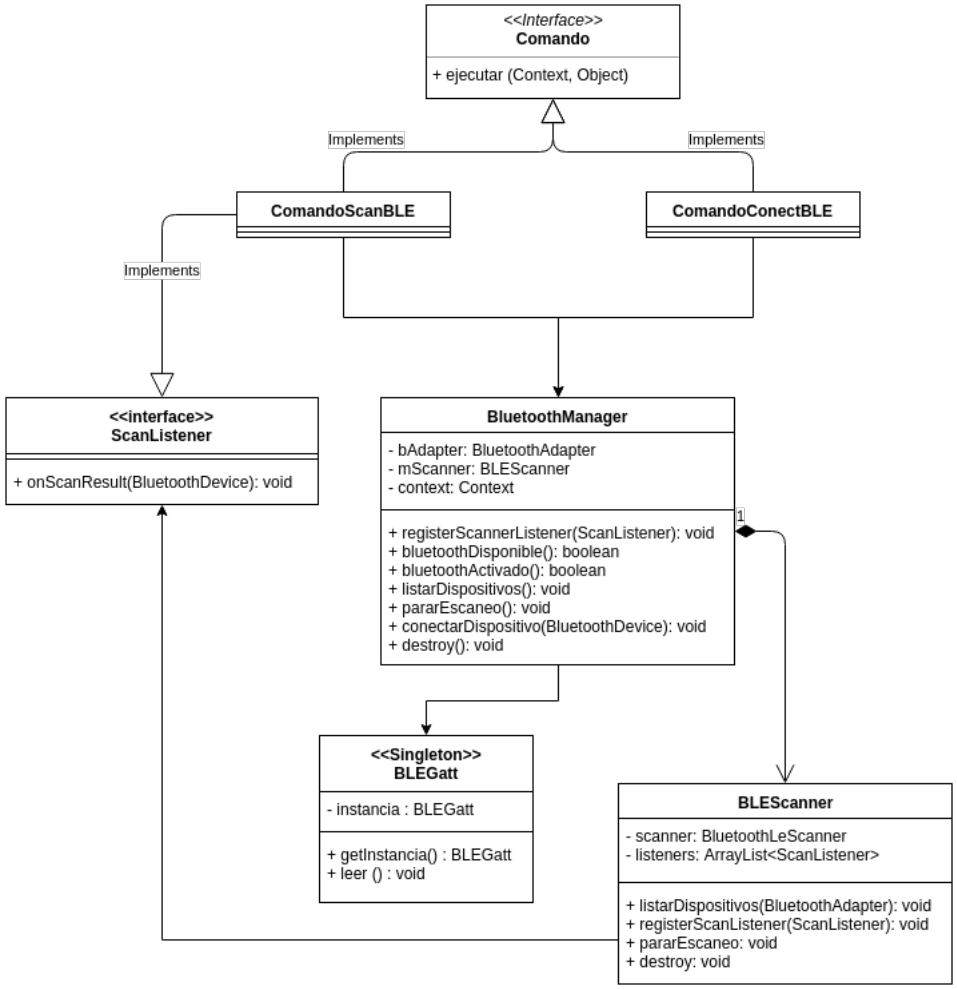

Figura 6.6: *Diagrama de las clases que intervienen en una conexión BLE*

6.2.3. Funcionamiento durante la marcha

Para llevar a cabo una sesión tenemos tres clases, como se puede ver en la Figura 6.7, que heredan de Thread: BluetoothThread, CoordenadasThread y FileThread. De ese modo, la lectura de los datos del acelerómetro, los datos de la ubicación y la escritura en el archivo se hacen de manera independiente.

BluetoothThread utiliza la clase BLEGatt explicada anteriormente para leer las características recibidas. Como se menciona en la Sección 5.2, hemos configurado la placa nRF51-DK para que tenga un servicio con UUID 0xA000, que contiene la característica con

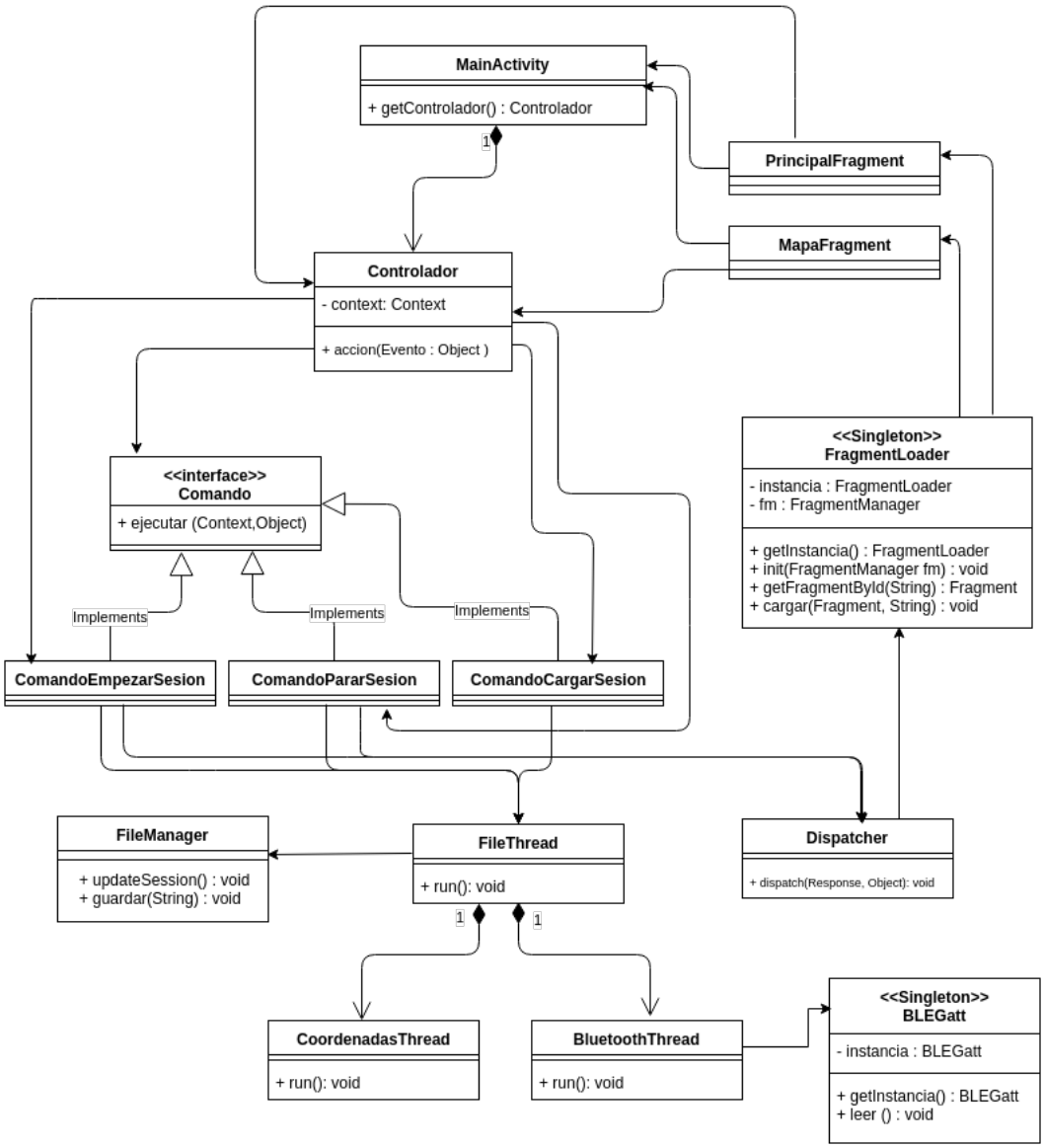

Figura 6.7: *Diagrama de las clases que intervienen en una sesión*

UUID 0xA001, que es la que nos interesa para obtener la información del acelerómetro. Para descubrir este servicio, se puede realizar lo que se llama *escaneo de servicios*, que devuelve una lista de los servicios del dispositivo, pero como ya conocemos de antemano el UUID, podemos acceder a él directamente.

Para recoger la información del acelerómetro, se accede la característica mencionada a través de su identificador, lo que devuelve una clase de Android llamada *BluetoothGattCha-*

racteristic, que podemos usar para leer el valor del GATT.

CoordenadasThread accede periódicamente a la última localización guardada en *LocationManager*, como se explicaba en la sección anterior.

FileThread se encarga de lanzar los dos hilos de BluetoothThread y CoordenadasThread y, periódicamente, recoge los datos obtenidos por ambos. Estos son guardados mediante una clase llamada FileManager que es la encargada del manejo de archivos. El formato del archivo que se crea consiste en una línea por cada muestreo dividida de la siguiente manera:

(latitud);(longitud);(coordenadaX);(coordenadaY);(coordenadaZ)

con una última línea indicando el tiempo total de la sesión de la siguiente manera:

T(tiempo en milisegundos)

Este archivo se guarda en una carpeta en el almacenamiento externo del móvil llamada accelbike; si no existe dicha carpeta, la clase se encarga de crearla.

Para detener la sesión aprovechamos una característica de los hilos en Java que nos permite diferenciarlos de manera única a través de un nombre dado por una cadena de caracteres. De este modo podemos detener un determinado hilo a través de este nombre, por lo que basta con finalizarlos para que se termine la sesión.

6.2.4. Generación de mapas tras marcha

Una vez hemos terminado una sesión, o bien hemos cargado una sesión anterior desde la vista de actividades, volvemos a llamar a la clase FileManager, esta vez para cargar el archivo. Se parsea el archivo y se agrupan las localizaciones de dos en dos calculando la distancia entre ellas y creando una polilínea a la que se le da color con los datos del acelerómetro. Cada polilínea se guarda en un array, que junto el tiempo y la distancia se inserta en una clase que actúa como Transfer. Este transfer se devuelve a la vista MapFragment, que se

encarga de mostrar los datos de velocidad media, distancia y tiempo y utilizar el array de polilíneas para pintar la ruta en un mapa de Google Maps.

Capítulo 7

Mediciones y calibrado

7.1. Mediciones de consumo

El objetivo principal de Bluetooth Low Energy es incorporarse a dispositivos de bajo consumo, y ser lo más eficiente posible energéticamente. Por ello decidimos comprobar la eficiencia de consumo de energía de nuestra placa, conectándola a un osciloscopio y recogiendo los datos del amperaje durante la ejecución del programa, en el que se recogen los datos del acelerómetro, se envían por BLE y se entra en un estado de *espera*. La gráfica resultante se muestra en la Figura 7.1.

Se puede comprobar que no tiene un consumo muy elevado, con un máximo algo menor que 9.5 mA, que es cuando realiza la comunicación por BLE y un mínimo de cerca de 5 mA en los períodos inactivos. En los períodos de inactividad observamos que consume más de lo cabría esperar. Esto es debido a que pensábamos erróneamente que la función que utilizábamos para realizar la espera pondría al procesador en modo *deep sleep*, que permitiría disminuir todo lo posible el uso de energía.

Buscando en la API de mbed, encontramos la forma de entrar en este modo, que se realiza con la función *sleep(ms)*[2]. Esta función ejecuta lo que se denomina un *Wait for Interrupt*, que es una instrucción del repertorio de instrucciones de ARM que deja al procesador en modo de bajo consumo, parando todas las funcionalidades para ahorrar energía hasta que llegue una interrupción. La función *sleep* que implementa la API de mbed determina

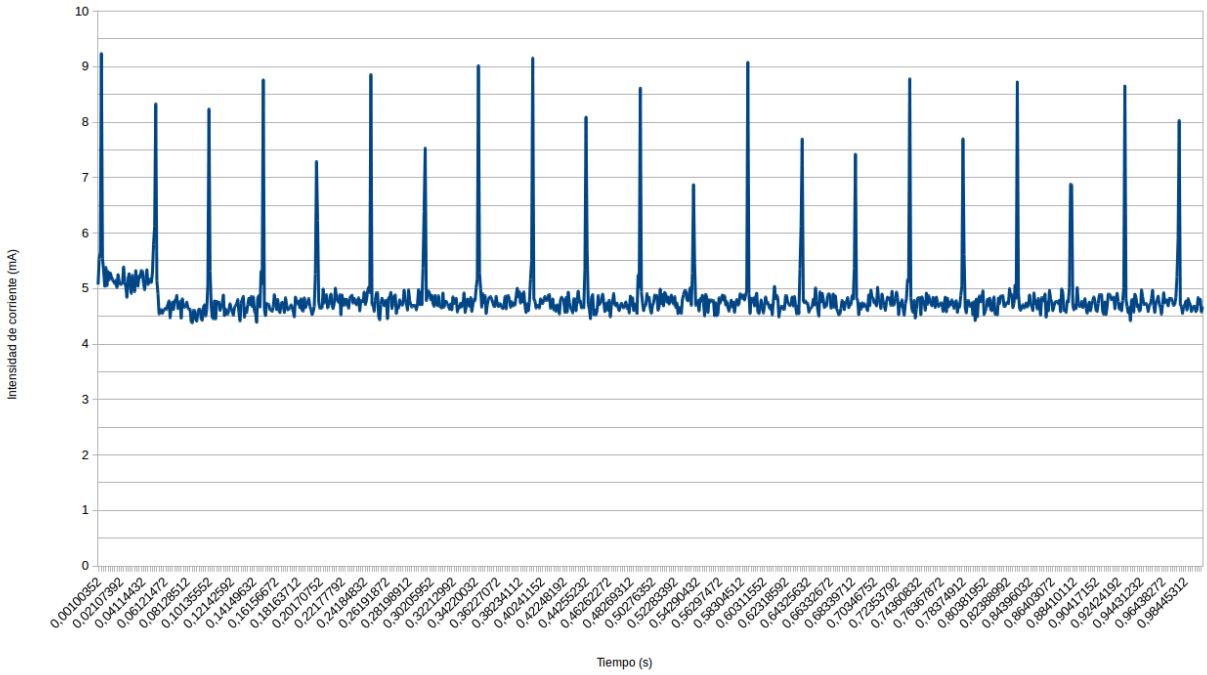

Figura 7.1: *Primera gráfica de corriente. Durante esta prueba el tiempo de espera estaba establecido en 0.1 s*

dinámicamente en qué modo de bajo consumo se entrará.

El punto negativo de esto es que no hace discriminación entre los tipos de interrupción, por lo que puede salir del modo *sleep* aunque no se lo hayamos indicado. Para evitar esto utilizamos un *callback* que se ejecutará cada segundo, activando una variable booleana. Esta variable controla un bucle en el que, si sale del modo *sleep* y no es por nuestra interrupción, vuelve a ejecutar un *sleep*.

Una vez implementada esta funcionalidad, volvimos a realizar la prueba de consumo y esta vez obtuvimos resultados más satisfactorios, como se puede apreciar en la Figura 7.2.

Como se puede ver, el consumo ha disminuido notablemente después de utilizar la función comentada, con mínimos en torno a 1.25 mA en los períodos de *sleep*.

En estas pruebas observamos la existencia de unos picos de consumo cada 50 ms, que no se aprecian en la imagen ya que la recogida de datos se daba cada segundo. Estos picos se dan durante el período de bajo consumo, y sabemos que son debidos a la componente BLE

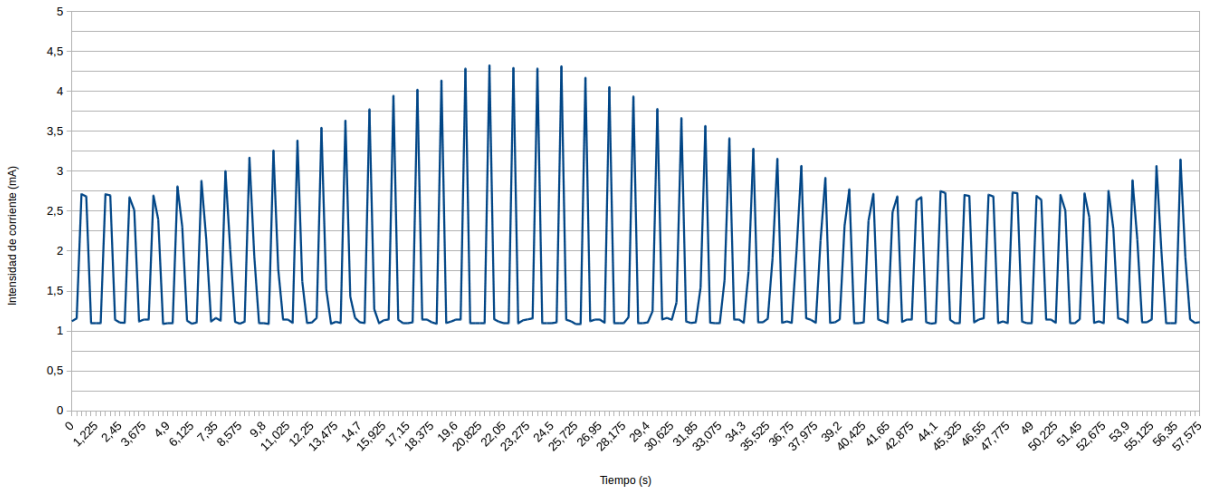

Figura 7.2: *Segunda gráfica de corriente. Cada segundo vemos un pico, que es cuando se despierta del sleep*

puesto que realizamos pruebas con este módulo desactivado y no se presentaban, aunque no sabemos exactamente por qué se producen.

7.2. Calibrado

Para conocer qué rangos de los datos de acelerometría podían ser interesantes para representarlos en la ruta, realizamos una serie de pruebas para obtener valores reales y luego los analizamos para sacar conclusiones. Pronto nos dimos cuenta de que dependiendo de la posición en la que se encuentre nuestro dispositivo, los resultados pueden variar en gran medida. Por lo que sería necesario un proceso de calibración antes de empezar la marcha.

Para nuestro proyecto, nos hemos centrado en el eje x para determinar la aceleración en cada momento. Decidimos establecer una serie de umbrales para colorear la ruta en el mapa generado, de modo que si detectamos un valor negativo, es decir, una deceleración, usamos el color azul. Si estamos en un valor cercano a 0, usamos el verde. En cuanto aceleramos ligeramente (hasta los 100 mg), usamos el amarillo, y a partir de ahí cuando se acelera aún mas usamos el rojo.

Debido a la presencia constante de la aceleración de la gravedad en cualquier objeto,

el acelerómetro recoge siempre este valor, repartido entre los ejes (x, y, z). En llano, la aceleración o deceleración se produce sólo en el eje x, pero cuando nos hallamos en una pendiente este valor se reparte entre los ejes (x, z), por lo que parte de la aceleración se juntará con el valor de la gravedad.

Capítulo 8

Conclusiones y trabajo futuro

Este proyecto nos ha permitido explorar tecnologías hardware con las que no habíamos trabajado nunca, como Bluetooth Low Energy y protocolos como SPI o I2C, lo que nos ha dado unas competencias en otros campos que no son específicamente los nuestros. Hemos aprendido a programar una placa desde cero, investigando la documentación que pone a nuestra disposición el fabricante y encontrando ejemplos para lograr los objetivos propuestos.

En el campo del desarrollo software, hemos trabajado por primera vez en el desarrollo de una aplicación Android, familiarizándonos con el entorno de programación *Android Studio*.

La línea de aprendizaje que hemos recorrido (en la que hemos empezado haciendo numerosas pruebas pequeñas antes de continuar con el proyecto final) nos ha permitido empezar por conceptos sencillos y fáciles de entender que nos permitían avanzar rápidamente a casos mas complejos.

Finalmente, hemos logrado producir una aplicación que se puede utilizar en entornos reales, con la que, fijando la placa BLE en una bicicleta, podemos realizar una ruta y visualizar correctamente los datos generados durante la misma.

Bluetooth Low Energy es un concepto muy interesante, y abre las puertas para comunicar gran cantidad de dispositivos. Con esta tecnología podríamos realizar más proyectos útiles que hasta ahora no eran posibles.

8.1. Trabajo futuro

Aunque la aplicación es funcional y cumple con los requisitos del proyecto, siempre es interesante la idea de ampliar la utilidad de nuestra aplicación.

Un ejemplo claro es el de poder aprovechar aún más el hecho de tener la aplicación ejecutándose en un dispositivo móvil, ofreciendo la posibilidad de compartir las sesiones realizadas en las redes sociales (Twitter, Facebook, ...).

Otra posibilidad que sería de utilidad es la de fijar una luz de freno a la placa, que al detectar una deceleración se encienda para indicar a conductores, peatones, etc. que se está reduciendo la velocidad para evitar accidentes.

En relación con esto, otro punto importante de cara al futuro es el de utilizar los datos que se recogen de acelerometría para comprobar movimientos muy bruscos (como los que se darían en caso de accidente) para activar un sistema que avise a los servicios de emergencia en caso de que el usuario no responda en un breve tiempo.

Se podrían realizar más cálculos con los datos obtenidos para poder sacar más información de la ruta. Además, aprovechando que la placa XTRINSIC-SENSE-BOARD contiene, aparte del acelerómetro que usamos, un magnetómetro y un sensor de presión, se podría recoger mucha más información sobre el esfuerzo realizado durante el trayecto.

Bibliografía

[1] Api de google maps. https://developers.google.com/maps/documentation/android-api/?hl=es.

[2] Api para deep-sleep de mbed. https://developer.mbed.org/teams/SiliconLabs/wiki/Using-the-improved-mbed-sleep-API.

[3] Arm system ip. http://www.arm.com/products/system-ip.

[4] Bluetooth core specification. https://www.bluetooth.com/specifications/bluetooth-core-specification.

[5] Cypress 100 projects in 100 days. https://github.com/cypresssemiconductorco/PSoC-4-BLE/tree/master/100_Projects_in_100_Days.

[6] Datasheet para el acelerómetro mma8491q. http://www.element14.com/community/servlet/JiveServlet/previewBody/54565-102-1-273580/Datasheet_MMA8491Q.pdf.

[7] Datasheet para el kit nrf51-dk de nordic. http://www.mouser.com/ds/2/297/nRF51-DK%20-%20User%20Guide%20v1_0-706637.pdf.

[8] Datasheet para el kit psoc 4 ble de cypress. http://www.cypress.com/file/137466/download.

[9] Material design de google. https://material.google.com/.

[10] Página de tutoriales de cypress. http://www.cypress.com/training/psoc-4-ble-101-video-tutorial-series-how-create-products-using-psoc-4-ble.

[11] Repositorio mbed. https://developer.mbed.org/activity/code/.

[12] Robin Heydon. *Bluetooth Low Energy: The Developer's Handbook.* Prentice Hall, 2012.

[13] Kevin Townsend, Carles Cufí, Akiba, and Robert Davidson. *Getting Started with Bluetooth Low Energy.* O'Reilly Media, 2014.

Apéndice A

Repositorios de código

A.1. Código nRF51-DK

https://developer.mbed.org/users/agufal/code/i2c_acelerometro/

A.2. Aplicación Android

https://github.com/AlexisVi/AccelbikeApp